

6-5-2015

UAVino

Matthew Belesiu
Santa Clara University

Aaron Chung
Santa Clara University

Kirby Linvill
Santa Clara University

Nathan Carlson
Santa Clara University

Phillip Coyle
Santa Clara University

See next page for additional authors

Follow this and additional works at: https://scholarcommons.scu.edu/idp_senior

 Part of the [Computer Engineering Commons](#), [Electrical and Computer Engineering Commons](#), and the [Mechanical Engineering Commons](#)

Recommended Citation

Belesiu, Matthew; Chung, Aaron; Linvill, Kirby; Carlson, Nathan; Coyle, Phillip; and Peekema, Megan, "UAVino" (2015). *Interdisciplinary Design Senior Theses*. 9.
https://scholarcommons.scu.edu/idp_senior/9

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Interdisciplinary Design Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

Author

Matthew Belesiu, Aaron Chung, Kirby Linvill, Nathan Carlson, Phillip Coyle, and Megan Peekema

Santa Clara University

Department of Electrical Engineering

Department of Mechanical Engineering

Department of Computer Science and Engineering

5 June 2015

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Matthew Belesiu

Aaron Chung

Kirby Linvill

Nathan Carlson

Phillip Coyle

Megan Peekema

ENTITLED

UAVino

BE ACCEPTED IN PARTIAL FULFILLMET OF THE REQUIRMENTS FOR THE DEGREES OF


Bachelor of Science in Electrical Engineering

Bachelor of Science in Mechanical Engineering

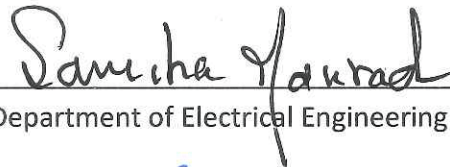
Bachelor of Science in Computer Science and Engineering



Thesis Advisor



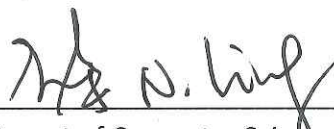
Thesis Advisor



Chair, Department of Electrical Engineering



Chair, Department of Mechanical Engineering



Chair, Department of Computer Science and Engineering

UAVino

by

Matthew Belesiu

Aaron Chung

Kirby Linvill

Nathan Carlson

Phillip Coyle

Megan Peekema

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering

Bachelor of Science in Mechanical Engineering

Bachelor of Science in Computer Science and Engineering

School of Engineering

Santa Clara University

Santa Clara, California

5 June 2015

UAVino

Matthew Belesiu

Aaron Chung

Kirby Linvill

Nathan Carlson

Phillip Coyle

Megan Peekema

Department of Electrical Engineering

Department of Mechanical Engineering

Department of Computer Science and Engineering

Santa Clara University

2015

Abstract

UAVino is a drone solution that uses aerial imagery to determine the overall plant health and water content of vineyards. In general, the system focuses on automating crop inspection by taking aerial imagery of a vineyard, conducting post-processing, and outputting an easily interpreted map of the vineyard's overall health. The project's key innovation is an auto-docking system that allows the drone to automatically return to its launch point and recharge in order to extend mission duration. Long term, UAVino is envisioned as a multi-year, interdisciplinary project involving both the Santa Clara University Robotics Systems Laboratory and local wineries in order to develop a fully functional drone agricultural inspection service.

Acknowledgements

The team would like to thank the following people for their help in making UAVino a reality:

- Professors Christopher Kitts and Shoba Krishnan for their constant guidance throughout the year as project advisors.
- The many members of the Santa Clara University Robotics Systems laboratory for providing the team with generous grant money, lab space to work in, and constant mentorship in a variety of fields. In particular, Thomas Adamek was instrumental in helping the team with hardware and software needs as well as serving as a general mentor over the course of the year.
- Santa Clara University's Undergraduate Engineering Programs department for their generous grant money. Specifically, Shane Wibeto for coordinating team reimbursements and logistical needs for a variety of presentations throughout the year.
- American Society of Mechanical Engineers Silicon Valley Section for their generous grant money.
- Jeff Ota and the Intel Corporation for sponsoring the team with Edison microcontroller boards, the multispectral imaging camera, and other hardware components that were critical to the system's success.
- John Aver of Aver Family Vineyards for providing the team with the opportunity to conduct real-world field deployments throughout the year, and for lending insight into current vineyard inspection techniques.
- Don MacCubbin in Santa Clara University's machine shop for his assistance and guidance in creating and manufacturing a variety of parts.
- The many friends, family, and faculty members who supported the team in a variety of capacities throughout the year.

Table of Contents

1. Introduction	1
1.1 Drone Background.....	1
1.2 Agricultural Inspection Techniques.....	2
1.3 Agricultural Inspection Systems.....	4
1.4 Agricultural Inspection Needs.....	5
1.5 Problem Statement	7
2. Systems-Level Design.....	8
2.1 Design Overview.....	8
2.2 Customer Needs.....	9
2.3 System Requirements	11
2.4 System Sketch	12
2.5 Functional Analysis.....	13
2.5.1 Functional Decomposition	13
2.5.2 Summary of System Inputs, Outputs, and Constraints.....	15
2.6 Product Competition.....	15
2.7 Key Systems Issues	17
2.7.1 Docking Station Charging Method	17
2.7.2 Docking Mechanism.....	18
2.8 System-Level Design Layout.....	19
2.9 Team and Project Management.....	21
2.9.1 Project Challenges and Constraints	21
2.9.2 Budget.....	22
2.9.3 Timeline.....	22
2.9.4 Design Process	23
2.9.5 Risks and Mitigations	24
2.9.6 Team Management.....	25
3. Docking Station	27
3.1 Overview	27
3.2 Requirements.....	27
3.3 Options and Tradeoffs.....	28
3.4 Design.....	29
3.4.1 Docking Station Frame.....	29
3.4.2 Docking Station Platform	32
3.4.3 Guidance Cones	33
3.4.4 Contact Plates	35

3.5	Recharging Circuitry	36
3.5.1	Overview	36
3.5.2	Wireless Network.....	39
3.5.3	Charging Routine.....	40
3.5.4	Power Delivery.....	41
3.5.5	Safety	43
3.6	Testing and Verification	44
3.7	Charging Future Work	45
3.8	Summary	47
4.	Octocopter Additions.....	48
4.1	Overview	48
4.2	Requirements.....	49
4.3	Ring Mounting System	50
4.3.1	Finite Element Analysis Overview.....	51
4.3.2	Finite Element Analysis Expectations	52
4.3.3	Finite Element Analysis Results	52
4.4	Vibration Isolation Camera Mount	53
4.5	Charging Feet	55
4.6	Landing Algorithm Hardware	58
4.7	Summary	59
5.	Landing Algorithm	61
5.1	Overview	61
5.2	Requirements.....	62
5.3	Hardware.....	63
5.3.1	Mobius Vision Camera	63
5.3.2	Intel Edison.....	63
5.3.3	Pixhawk Autopilot Board	65
5.4	Landing Algorithm Stages.....	66
5.4.1	Overview	66
5.4.2	Locate Docking Station	67
5.4.3	Movement Calculation.....	69
5.4.4	Flight Command Execution	73
5.5	Summary	74
6.	Data Processing	76
6.1	Overview	76
6.2	Requirements.....	77

6.2.1	Customer Requirements	77
6.2.2	Image Resolution Requirements.....	77
6.2.3	Image Overlap Requirements	77
6.3	Key Components	78
6.3.1	Multispectral Camera	78
6.3.2	Image Mosaicking Software.....	78
6.3.3	Image Processing Software.....	79
6.4	Post Processing Approach	79
6.4.1	Flight Parameters and Camera Settings.....	79
6.4.2	Image Mosaicking	81
6.4.3	Vegetation Index.....	84
6.5	Post Processing Review	85
6.6	Verification	87
6.6.1	Proof of Concept	87
6.6.2	Verification of Crop Health Data.....	87
6.7	Long Term Monitoring	89
6.8	Summary	89
7.	System Testing	91
7.1	Overview	91
7.2	Vehicle Weight Tests	91
7.3	Coverage Capability Tests	93
7.4	Recharging Capability Tests	94
7.5	Vision Recognition Tests	95
7.6	Summary	96
8.	Costing Analysis.....	97
8.1	Budget and Costs.....	97
9.	Commercialization Plan	98
9.1	Executive Summary	98
9.2	Background.....	98
9.3	Goals and Objectives.....	99
9.4	Key Technologies.....	99
9.5	Customers and Marketing.....	100
9.6	Manufacturing.....	100
9.7	Service Cost and Price	101

10. Engineering Standards and Realistic Constraints.....	102
10.1 Ethics Analysis	102
10.2 Legal Analysis	103
10.3 Health and Safety Analysis	104
10.4 Manufacturability Analysis.....	104
10.5 Social Impact Analysis	105
10.6 Arts	107
11. Summary and Conclusions	108
11.1 Project Summary	108
11.2 Future Work	109
References	111
A. Product Design Specification	114
B. Customer Interview Questions	115
C. Decision Matrices	117
D. Product Sketches.....	121
E. Project Timeline	126
F. Project Budget	128
G. System Inputs, Outputs, and Constraints	129
H. Detailed Assembly and Parts Drawings	130
I. Determining Ground Sample Distance Constant.....	166
J. Product Datasheets	168
K. UAVino Charge Control Source Code	179
L. UAVino Landing Control Source Code	184
M. Senior Design Conference Slides	219

List of Figures

1.1	Modern Drone Application Examples	2
1.2	Multispectral Image and NDVI Example.....	3
1.3	Tree Grading Analysis Example	4
1.4	California Drought	6
2.1	System Requirements Flowchart.....	11
2.2	System Sketch.....	12
2.3	Functional Decomposition.....	14
2.4	Competing Platforms.....	16
2.5	Docking Mechanism	19
2.6	System Block Diagram	20
2.7	Project Timeline.....	23
2.8	Team Structure	25
3.1	Completed Docking Station	30
3.2	Docking Station Frame	30
3.3	Adjustable Foot Detail	31
3.4	Docking Station Drawer	32
3.5	Docking Station Platform.....	32
3.6	Docking Station Platform Alignment Bars	33
3.7	Initial Guidance Cone Design.....	34
3.8	Final Guidance Cone Design	34
3.9	Docking Station Contact Plates	35
3.10	Docking Station Contact Plates with Octocopter	36
3.11	Octocopter Battery.....	36
3.12	Revolectrix CellPro Multi-4 Charger	37
3.13	Raspberry Pi Microcontroller	38
3.14	Netis AV1200 Wireless USB Adaptor.....	39
3.15	Charging Routine Software Flowchart.....	40
3.16	Docking Station Circuit Diagram.....	42
3.17	Docking Station Components	43
3.18	Docking Station Charging Performance	45
4.1	3D Robotics X8 Ococopter	48
4.2	Ring Mounting Bracket Detail	50
4.3	Ring Mounting Bracket Loading Conditions	51

4.4	Bracket Horizontal Load Stress Distribution.....	52
4.5	Ring Horizontal Load Stress Distribution.....	53
4.6	Vibration Isolation Camera Mount CAD Rendering	54
4.7	Vibration Isolation Camera Mount.....	54
4.8	Charging Foot CAD Rendering	55
4.9	Charging Foot Bottom Detail.....	56
4.10	Charging Foot Top Detail.....	56
4.11	Octocopter Circuit Diagram.....	57
4.12	Mobius Camera Detail	58
4.13	Intel Edison Detail.....	59
5.1	Vision Guided Landing System Components.....	61
5.2	Docking Station Alignment Pattern	62
5.3	Mobius Action Cam	63
5.4	Intel Edison	64
5.5	Pixhawk Autopilot.....	65
5.6	Landing Algorithm Logic Flow.....	66
5.7	Separation of Onboard Processing and Flight Control.....	66
5.8	Landing Algorithm Error	67
5.9	Docking Station Location Example	68
5.10	Landing Algorithm Movement Types	69
5.11	Movement Type Selection Process	70
5.12	Ground Distance Calculation Example	71
6.1	Tetracam ADC Micro	78
6.2	Blurry Multispectral Image.....	80
6.3	Overexposed Multispectral Image	81
6.4	Agisoft Orthomosaic Example	82
6.5	Photoshop Orthomosaic Example.....	83
6.6	Reflectance vs. Wavelength	85
6.7	Data Processing Method Overview	86
6.8	Diseased Plant NDVI Example	87
6.9	NDVI Week-to-Week Comparison.....	88
7.1	Mission Planning Software Flight Path.....	93
7.2	Docking Station Charge Curves	94
7.3	Landing Algorithm Error	95

C.1	System Requirements Prioritizing Matrix.....	117
C.2	Enclosure Concepts Decision Matrix	118
C.3	Position Method Decision Matrix.....	119
C.4	Charging Method Decision Matrix.....	120
D.1	Contact Plate Charging Concept	121
D.2	Induction Charging Concept	122
D.3	Guidance Cone Docking Concept	123
D.4	Magnet Guidance Docking Concept	124
D.5	Gravity Guidance Docking Concept	125
J.1	Intel Edison Datasheet (Page 1)	168
J.2	Intel Edison Datasheet (Page 2)	169
J.3	Pixhawk Datasheet (Page 1)	170
J.4	Pixhawk Datasheet (Page 2)	171
J.5	Mobius Datasheet (Page 1)	172
J.6	Mobius Datasheet (Page 2)	173
J.7	Pixhawk Datasheet (Page 1)	170
J.8	Revolextrix CellPro Multi-4 Datasheet	174
J.9	Raspberry Pi Datasheet	175
J.10	Netis AC1200 Wireless Adaptor Datasheet (Page 1)	176
J.11	Netis AC1200 Wireless Adaptor Datasheet (Page 2)	177
J.12	Tetracam ADC Micro Datasheet.....	178

List of Tables

2.1	UAVino Competitor Specifications	16
3.1	Intel Edison and Raspberry Pi Features Comparison	38
4.1	Octocopter and Additional Component Weights.....	49
5.1	Intel Edison, Raspberry Pi, and Beaglebone Black Comparison.....	64
5.2	Movement Type Selection Process	70
5.3	Velocity Control Methods.....	74
7.1	Vehicle Weights	92
8.1	UAVino Cost Breakdown	97
9.1	UAVino Cost Breakdown	101
10.1	Arts Requirement	107
A.1	Product Design Specification.....	114
F.1	UAVino Donations	128
F.2	UAVino Expenses.....	128
G.1	Inputs.....	129
G.2	Outputs.....	129
G.3	Constraints.....	129
I.1	Average GSD Constant Values.....	167

CHAPTER 1

Introduction

1.1 Drone Background

Over the past few years, personal use drones have surged in popularity due to a dramatic increase in their capabilities. Together, the efforts of casual hobby enthusiasts and professional developers have resulted in the creation of advanced control systems and sensors that have opened the doors to this new technology; drone research and experiments that used to require funding on a professional level are now available to the everyday civilian. Although personal drones have come under criticism because of potential safety and privacy issues, many industries stand to benefit from this emerging technology, provided it is applied in a safe and sensible way.

Already, drones have been modified to meet a wide variety of needs and scenarios, as shown in Figure 1.1. One of the first notable drone applications occurred in 2013 when the Seattle Police Department considered their use for a wide variety of tasks, including crowd monitoring [1]. Ultimately, the Department was forced to abandon its plans due to public concern over privacy, but not all potential drone applications have met this same fate [2]. In September 2014, the Federal Aviation Administration granted regulatory exceptions to several video production companies, paving the way for drone use in the film industry [3]. One application that has actually garnered public excitement is Prime Air, a drone package delivery service currently being developed by Amazon.com [4]. Although somewhat futuristic, the benefits resulting from such a system with regards to reduced carbon emissions, faster delivery times, and lower overall costs are intriguing.

While drones are certainly poised to increase profits for well-established industries, they are also well-suited for social benefit applications. In late 2014, a graduate student at Delft University of Technology in the Netherlands equipped a quadcopter with a defibrillator in order to create a concept drone that could be used to enhance

emergency response services [5]. Additionally, drones have been deployed for animal tracking and poaching prevention in the developing world, including rural Africa [6, 7].



Figure 1.1: Left: Prototype drone for Amazon’s planned Prime Air delivery service. Photo courtesy of amazon.com. Right: Conceptual defibrillator-equipped drone developed by Delft University of Technology. Photo courtesy of tudelft.nl.

Provided that the Federal Aviation Administration is able to meet its 2015 deadline to integrate drones into the National Airspace System, it is estimated that the unmanned aerial vehicle industry will create 70,000 new jobs and make \$13.6 billion in economic impact by 2018. By 2025, those numbers reach over 100,000 new jobs and over \$80 billion in economic impact [8]. Given the number of applications towards which drones have already being applied, this technology has an exciting future with the potential to positively affect the world.

1.2 Agricultural Inspection Techniques

In particular, the agricultural industry stands to benefit from the use of drones due to the relative ease with which an aerial vehicle can cover the large land area occupied by crops. One potential application is crop monitoring, as drones are ideal for modern field inspection methods that involve multispectral and infrared aerial imaging. Although these advanced inspection techniques are still actively being developed, research has shown that, with the correct post processing, these image types can be used to accurately yield information about crop health.

One processing method uses multispectral images to generate a Normalized Difference Vegetation Index (NDVI), which provides information about differing chlorophyll levels

in crops in order to examine their health. The NDVI is determined with two multispectral bands: near-infrared reflectance and visible reflectance. Dividing the difference in these two quantities by their sum yields a fraction that indicates vegetation in a particular area as well as the concentration of chlorophyll in the leaves of plants. A larger NDVI indicates dense vegetation, while a smaller value suggests poor crop health [9]. An example multispectral image and its processed NDVI version are shown in Figure 1.2.



Figure 1.2: Example of a raw multispectral image and its processed counterpart. Green indicates vegetation in good health, yellow indicates moderate health, and red indicates poor health or areas of non-vegetation.

Additionally, the temperature information returned in infrared images can be used to determine the water content of soil and the water stress in crops. In general, colder temperatures of surrounding soil and plant canopies indicate higher water content and lower water stress, and vice versa [9]. In some cases, fractional vegetation coverage obtained through the NDVI can also be used to conduct more detailed soil moisture analysis using a technique called the ‘triangular method’ [10]. Thermal imaging has also been used to detect pathogens and disease within plants with methods that also utilize overall canopy and leaf temperatures [11, 12].

Research involving these imaging techniques for agricultural use includes a 2003 NASA study, in which a small drone equipped with multispectral and hyperspectral imaging cameras was used to conduct research flights at California vineyards. Ultimately, the test flights showed that these imaging techniques could provide accurate information

and high resolution data regarding percentage vegetation cover [13]. A similar 2003 study conducted by California State University Monterey Bay and NASA AMES Research Center used the NDVI to determine the leaf area of vineyards and concluded that these methods showed promise with regards to indicating vegetation cover and plant canopy health [14]. Additional multispectral imaging experiments with successful results include a 2001 study at Adelaide University that determined wine grape varieties based on chlorophyll level calculations and a 2005 study at the University of Georgia that differentiated field types based on vegetation density information [15, 16].

1.3 Agricultural Inspection Systems

In industry, determining crop health with multispectral and infrared imaging data has largely been conducted with satellites, as a variety of companies offer the required services at competitive prices. An example of such services is shown in Figure 1.3.

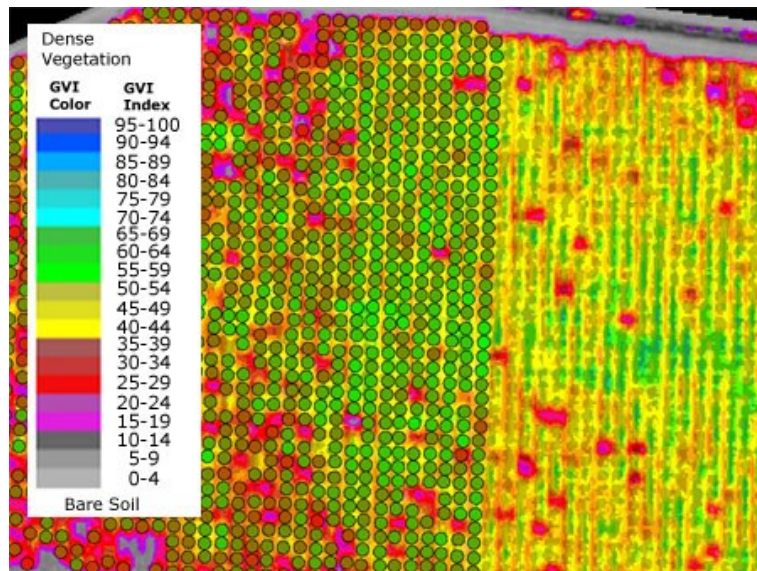


Figure 1.3: Example of Tree Grading analysis on a 0.6m resolution satellite imaging using DigitalGlobe’s AgroWatch™ custom software. Image courtesy of Satellite Imaging Corporation.

Some satellite imaging companies advertise resolutions of up to 0.5m at costs on the order of \$20 per acre per year [17]. Although satellite imagery is an attractive option, the image resolution, while impressive, may not be sufficient to yield the required

health data. Additionally, satellite images can only be taken on days with no cloud cover, which may be difficult depending upon a particular field's location. Moreover, farmers must contract with external companies in order to obtain satellite imaging data, meaning that crop health information is not always immediately accessible. Therefore, while satellites may be suitable for some crop monitoring situations, they do not provide a perfect solution.

Drone platforms are able to take image data at ground resolutions on the order of centimeters per pixel, much greater than any available satellite. Additionally, such systems are highly flexible, as they can be purchased directly by an individual farmer and deployed at will. This precise, versatile, and on-demand nature makes drones ideal for crop monitoring. This potential has already been recognized, as many small drone platforms are available for aerial imaging applications. One such product is the Precision Hawk Lancaster Platform, which is a highly customizable drone that can be tailored for a wide variety of industries, including agricultural monitoring, mining, and infrastructure surveying. While this system is extremely advanced and highly capable, its base price is \$25,000 and therefore is prohibitively expensive to many smaller farmers [18]. More recently, 3D Robotics released their Aero-M and XM-8 aerial mapping platforms, which are in the \$5,000 range. These products are clearly trying to meet the demand for a cheaper drone solution, but they require technological knowledge in order to build and operate [19].

Ultimately, multispectral and infrared imaging technology has proven to be a promising method of measuring crop health, and drones have been recognized as an ideal aerial imaging platform. However, these two technologies have yet to be mated into a complete crop monitoring system that is both affordable and straightforward to implement.

1.4 Agricultural Inspection Needs

The need for effective and efficient crop inspection methods has recently been highlighted in California, where water conservation is paramount due to severe levels of

drought. In 2013, California had the driest year in recorded state history and Governor Jerry Brown declared a drought emergency, including asking the state to voluntarily cut waste water usage by 20 percent [20]. This unprecedented drought, shown graphically in Figure 1.4, has forced farmers into a state of duress and placed added pressure on their crops.

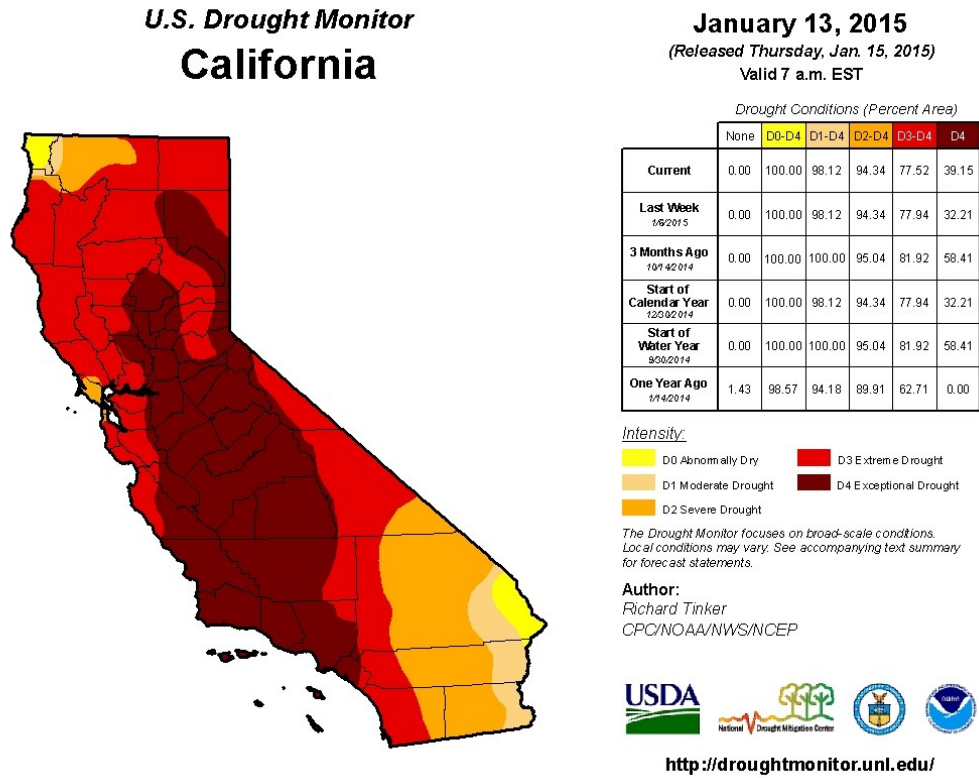


Figure 1.4: Graphical depiction of the extent of California’s drought as of January 2015. The U.S. Drought Monitor is jointly produced by the National Drought Mitigation Center at the University of Nebraska-Lincoln, the United States Department of Agriculture, and the National Oceanic and Atmospheric Administration. Map courtesy of NDMC-UNL.

Particularly under stress are California’s small vineyard owners, who operate with relatively low excess capital and are having difficulty coping with extremely high water costs. If an inexpensive and easy-to-use drone crop monitoring system was developed, it could greatly help these vintners by providing information regarding where and how to most efficiently water their fields. Winemaking is a staple of California’s economy and local, family-run vineyards are a hallmark of the wine industry. Therefore, an

opportunity exists to meet a customer need as well as make a positive social impact using drone technology.

1.5 Problem Statement

The long-term vision of UAVino is to develop a fully autonomous drone crop inspection system. The objective in this first year of the project was to demonstrate that multispectral imaging is a viable method of determining crop health using drones and to lay the foundations for autonomous docking and recharging. In order to accomplish this goal, a docking station with recharging capability was designed and constructed, computer vision algorithms were researched and implemented, and multispectral data processing techniques were tested. Additionally, a variety of octocopter modifications were made in order to allow the vehicle to take multispectral images, dock precisely and recharge with the station, and fly autonomously using a computer vision algorithm. Although the system is not yet fully autonomous, the end result of this year's work is a successful proof-of-concept with a built and tested docking and recharging station, an in-progress landing algorithm, and tested data processing techniques. Thus, future teams are well-positioned to continue developing UAVino in order to meet its ultimate goal of bringing valuable crop health information to real world vineyard customers.

CHAPTER 2

Systems-Level Design

2.1 Design Overview

UAVino is envisioned as a multi-year project that Santa Clara University students can continually build upon and develop as new drone technologies become available. The project in its current state represents one academic year of effort aimed at developing a proof-of-concept agricultural inspection system capable of photographing crops and yielding plant health data. Long term, the goal is to reach a level of autonomy where UAVino is capable of operating without any human monitoring. Such a system might include features such as multiple octocopters, collision detection and avoidance, and real-time post-processing. Thus, there are numerous educational and business benefits associated with continued project development.

The key technological innovation developed in this system proof-of-concept is a portable station that the octocopter automatically returns to, docks with, and then recharges from in order to extend mission duration. The need for this station arises from the relatively low flight time of octocopter vehicles, which is typically under 15 minutes per flight battery. Therefore, inspecting large farms requires multiple flights in order to complete. Typically, this limitation has required the vehicle operator to manually replace or recharge the flight battery multiple times in order to continue a mission. However, UAVino's docking station mitigates this need, as it greatly increases system autonomy and reduces operator workload.

Ultimately, UAVino is imagined as a crop inspection service that students in the Santa Clara University Robotics Systems Laboratory will be able to provide to local farmers on an as-needed basis. Via this service-type implementation, individual farmers will not need to invest significant capital to purchase the system outright or spend time training and familiarizing themselves with how to operate complicated technology. Instead, interested farmers will simply request the service when needed and benefit from the

crop health information produced. Additionally, this service model allows Santa Clara University students to gain an understanding of business fundamentals and experience real-world customer interaction.

2.2 Customer Needs

After developing the idea for UAVino, interviews were conducted with relevant stakeholders and consumers to gather more information about the potential uses and scope of the project and thus further refine its initial concept. Professor Kitts and Thomas Adamek, who serve in advisory roles on the project and are akin to investors and long-term operations managers, were interviewed for their interest and concerns regarding UAVino's long-term viability. Lindsay Kalkbrenner, Director for Sustainability at Santa Clara University, was interviewed in order to research what type of tasks UAVino might be applied towards besides agricultural inspection. Lastly, UAVino's real vineyard customer, John Aver of Aver Family Vineyards, was interviewed in order to understand more about how vineyard inspections are currently performed and what type of final data product would most help in determining vine health. Overall, these interviews yielded a list of priorities that aided in setting goals and deliverables throughout the year. Actual interview questions are available in Appendix C.

As a stakeholder in the project, Professor Kitts reinforced his belief that UAVino would best be implemented as a service provided by the Robotics Systems Laboratory to local customers, rather than as a product sold at market. Ultimately, his long-term goal is to make the project self-sustainable financially so that it can benefit engineering education, seed real-world research projects, and provide a real benefit to real people. Although there are several other companies currently working on developing drones for agricultural use, Professor Kitts stated that this competition serves as good motivation and that he is not concerned UAVino will fail to make an impact. Instead, his major concerns lie with developing and using the product in a safe and professional manner in order to maintain strong customer relationships.

Thomas Adamek shared similar thoughts as those of Professor Kitts, as he also believes UAVino would best be implemented as a service instead of a product. This view stems from the likelihood that teaching a farmer who is inexperienced with robotics and remote controlled aircraft would prove difficult. Therefore, it would be easier for that farmer to simply contract with a service provider rather than learn an entirely new platform. Because Thomas is one of the stakeholders who will continue working firsthand with this project in future years, he is particularly interested in ensuring that progress is well documented so that it is easy for others to transition onto the project.

Lindsay Kalkbrenner was excited to discuss the potential applications of UAVino on the Santa Clara University campus, as such deployments are planned long term expansions of the project beyond agricultural monitoring. When asked where and how the system could help, she highlighted field water content monitoring as a primary need. Currently, the University uses both recycled water and drinking water for fields and landscaping. Because water in general, particularly potable water, is expensive, it is critical to use this resource efficiently. Kalkbrenner felt that UAVino could help monitor water usage on campus, especially places that use expensive drinking water, such as the Mission Gardens and Buckshaw Stadium, to help develop more sustainable watering practices.

The insight provided by John Aver was used to identify specific needs and requirements for UAVino's final data product when monitoring agricultural field health. Because Aver Family Vineyards is a small and local business, it was not surprising to learn that the typical method of inspection for these types of vineyards is very time consuming and labor intensive, as it typically involves literally walking up and down vineyard rows and inspecting individual vines by sight. While Aver felt that this type of inspection is invaluable and would always be required on some level, he welcomed any additional data provided by UAVino that could help identify problem areas so that he could more easily focus manual inspections. In this regard, providing both vegetation indices and water content measurements would be extremely beneficial in order to help ensure the health of each grape vine.

2.3 System Requirements

Using this customer feedback in conjunction with the initial design concept, a series of requirements, shown in Figure 2.1, were created to guide future decisions.

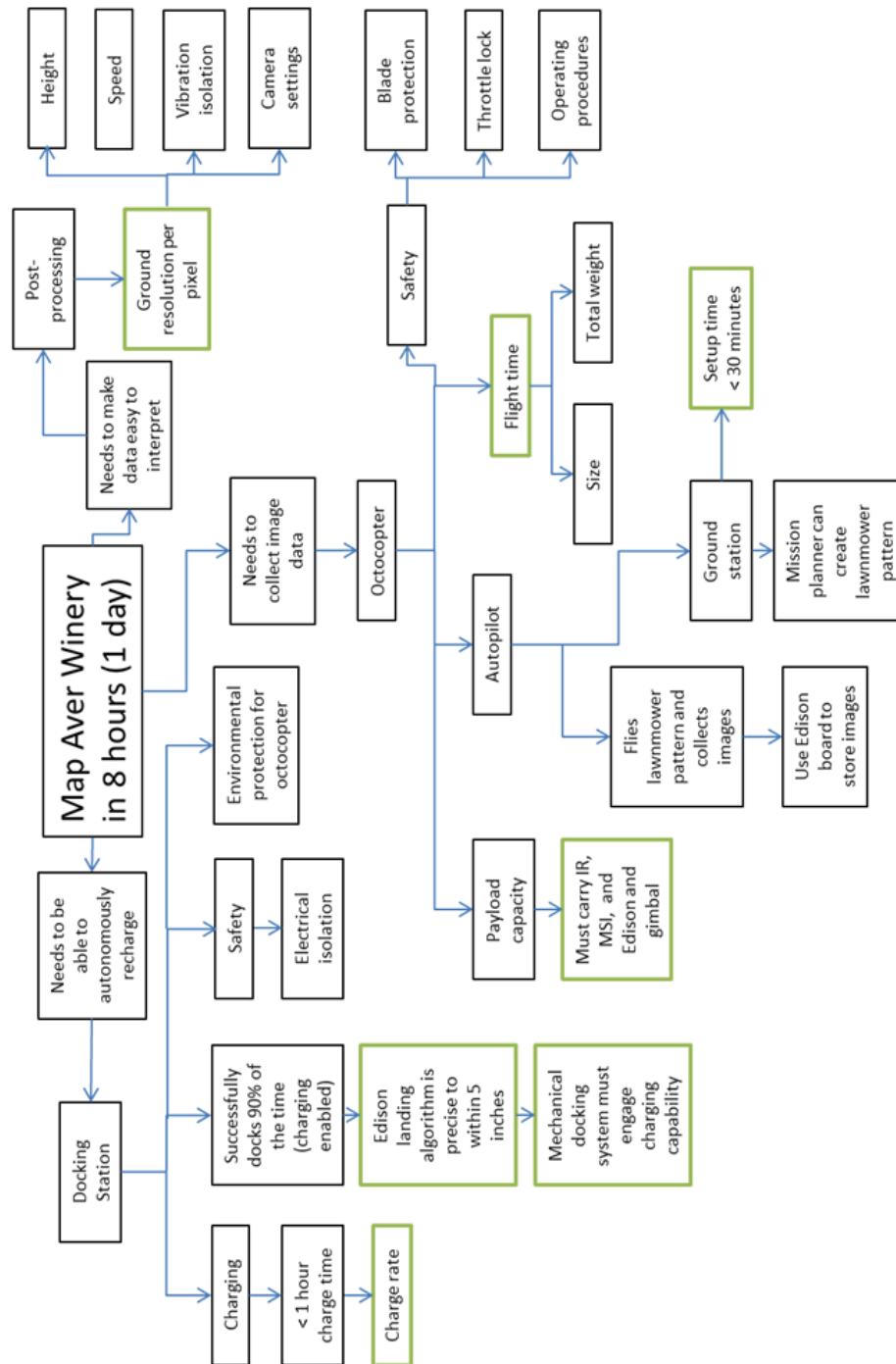


Figure 2.1: UAVino system requirements flowchart.

Putting system requirements in flowchart format allowed for easy determination of root level system requirements, which are shown in green boxes in the Figure. These items are:

- The docking station must be able to recharge the octocopter battery within 1 hour.
- The octocopter must have a flight time of at least 15 minutes.
- The overall setup time of the system must take less than 30 minutes.
- The octocopter must be able to carry the necessary camera payload in addition to an Intel Edison microcontroller and recharging electronics.
- The final data product must have a ground resolution of at least 0.5cm/pixel.

2.4 System Sketch

Figure 2.2 shows the process by which UAVino inspects an agricultural field.

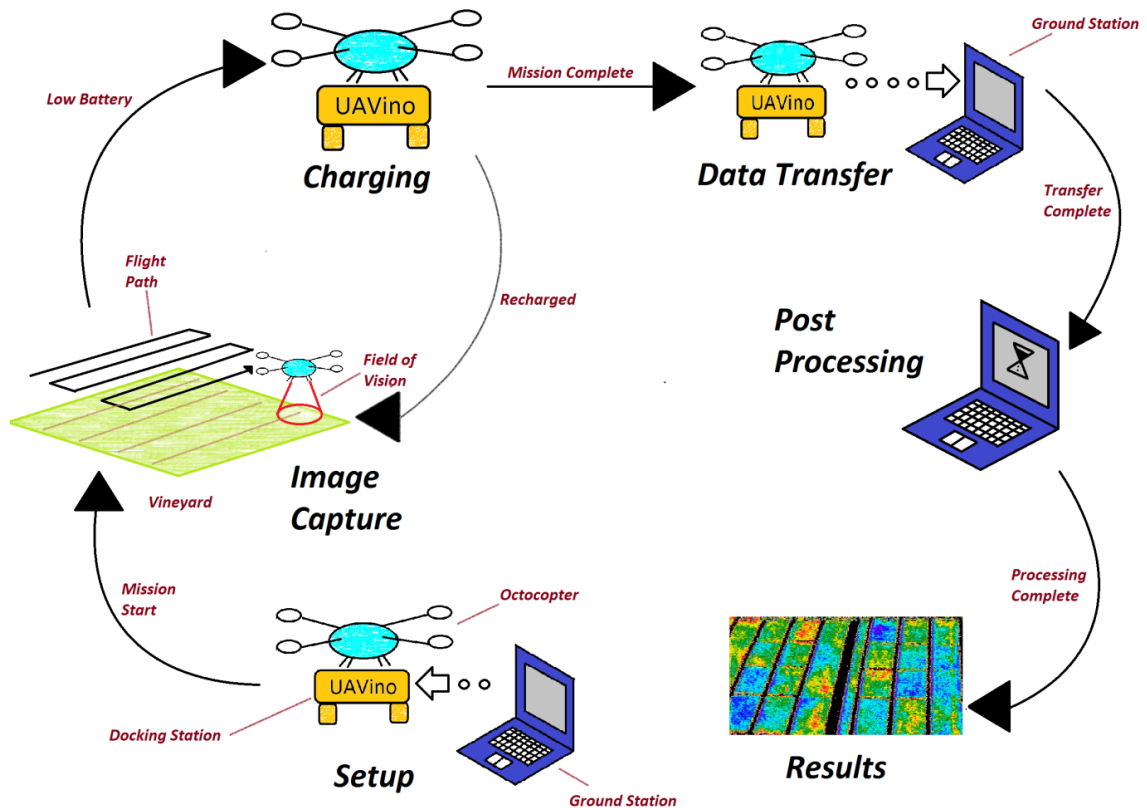


Figure 2.2: Sketch showing the process by which UAVino inspects an agricultural field

In general, a UAVino mission begins with setting up the docking station and ground station at the desired field site and then sending flight path coordinates to the octocopter via a command uplink. When ready, the octocopter departs from the docking station, flies to the desired field, and then overflies it in a grid-like pattern to collect imaging data. When the octocopter's battery reaches a specified level, the vehicle stops flying the grid pattern and returns to the docking station in order to recharge. Once the battery is full, the octocopter takes off, returns to the vineyard, and resumes mapping. After the entire mission is complete, the octocopter once again returns to the docking station to transfer image data to the ground station for analysis.

During the mission, the ground station receives flight telemetry data so that mission operators are aware of the vehicle's position, height, speed, flight mode, and other parameters. Additionally, the octocopter is linked to GPS in order to determine positional data.

After the mission is complete, the multispectral images are processed to yield a vegetation index, which determines overall crop health, and the infrared imagery is interpreted to find a field's water content. This data product is provided to the agricultural customer in the form of an easily interpreted aerial map depicting a particular field's health

2.5 Functional Analysis

2.5.1 Functional Decomposition

UAVino is divided into three main components: the octocopter docking station, the octocopter vehicle, and the ground control station. These subsystems must seamlessly work together to ensure a successful result, as the project relies on the octocopter's ability to successfully fly in a grid pattern over an agricultural field and then automatically dock with and recharge via the docking station. Figure 2.3 shows the key features of each system component.

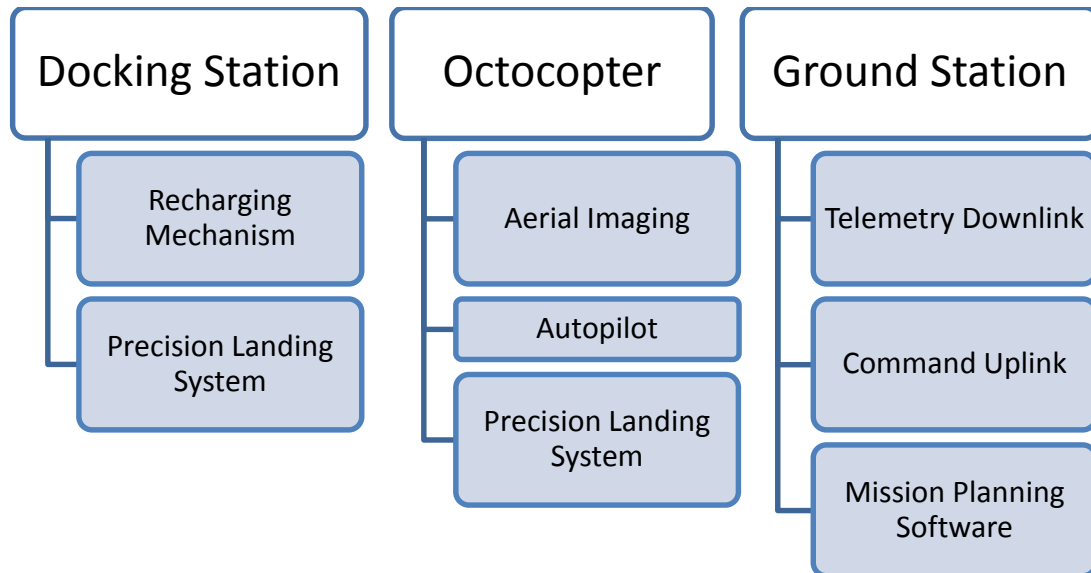


Figure 2.3: Functional decomposition of UAVino.

Because of its limited battery life, the octocopter must be able to recharge during its mission so that it can ultimately map an entire agricultural field. This need is accomplished via a docking station that the octocopter automatically returns to and lands on. The docking station contains a commercial battery charger and power source to autonomously recharge the octocopter’s flight battery after the vehicle has docked. The station also features mechanical components that aid in precisely positioning the octocopter during landing so electrical contact between the vehicle and station is established.

The octocopter is responsible for collecting imaging data via multispectral and infrared imaging cameras as well as flying in a grid-like pattern over fields via a GPS-linked autopilot. Additionally, the vehicle carries a microcontroller and vision camera coupled with the autopilot in order to precisely control the drone while landing on the docking station. This software-driven landing solution is augmented by mechanical components that ensure custom fitted electrical contacts on the vehicle’s feet properly mate with components on the docking station to enable recharging capability.

While flying, a ground control station connected to the octocopter via a telemetry link provides positional data and flight mode parameters to the operator. Additionally, this

ground station is equipped with mission planning software that allows for the development of flight path coordinates that are uploaded to the vehicle before a mission.

2.5.2 Summary of System Inputs, Outputs, and Constraints

For the docking station, octocopter, and ground station to function together and create a functioning system, a variety of data are obtained from and shared between components in the form of inputs, outputs, and constraints. In general, the octocopter uses GPS positional data to follow a predefined flight path and visual cues from a camera to precisely land on the docking station. While charging, the docking station takes power from an external source to recharge the octocopter's battery as well as monitors the individual voltage of each flight battery cell to ensure a safe, balanced charge takes place. During the mission, the ground control station monitors the status of the octocopter using telemetry. Specific inputs, outputs, and constraints for UAVino are listed in Appendix G.

2.6 Product Competition

Aerial mapping, particularly for agricultural use, is a quickly growing field with regards to small drones. Therefore, it is not surprising that numerous platforms for aerial crop monitoring already exist and several more are likely to be released in the near future. Two of the most prominent platforms and their competitive offerings are described below and photographs are available in Figure 2.4.

- 3D Robotics XM-8
 - Includes Pix4Dmapper LT 3DR Edition software
 - 14 minute flight time
 - 0.7in/pixel maximum ground resolution
- Precision Hawk Lancaster Platform
 - Highly customizable for a wide variety of applications
 - On-board diagnostics

- Fixed wing platform; not a multirotor drone
- Water landing capability
- All-in-one package
- Precision Drone PaceSetter
 - Real-time video streaming
 - Telemetry
 - Route scouting software



Figure 2.4: Top-Left: 3D Robotics X8-M multirotor platform. Photo courtesy of 3drobotics.com. Top-Right: Precision Hawk Lancaster fixed wing platform. Photo courtesy of precisionhawk.com. Bottom-Center: Precision Drone Pacesetter platform. Photo courtesy of precisionhawk.com.

Table 2.1 summarizes the key specifications of these competitors’ platforms and compares them to UAVino’s offerings.

Table 2.1: Specifications for several agricultural drone competitors.

Feature	3D Robotics	Precision Hawk	Precision Drone	UAVino
Price	\$5,400	\$25,000	\$17,500	\$6,050
Weight	5.4lbs	3lbs	4.5lbs	7.0lbs
Flight Time	14mins	5hrs	20mins	10mins
Camera	12MP	Multiple	11MP	Multispectral
Autopilot	Yes	Yes	Yes	Yes
Notable Features	Hobby-style, Entry Level System	Fixed Wing Platform	Real-Time Video Streaming	Autonomous Docking Capability

After researching competitors' offerings during the initial system design, it was determined that the main areas UAVino could capitalize on were price, simplicity, and the addition of a docking station. With the exception of 3D Robotics' octocopter, the platforms are extremely expensive—well above the upfront investment a small scale vineyard owner might be willing to pay for a new, complicated, and largely unproven technology. Thus, UAVino's service style-implementation with low upfront cost and minimal risk to the vineyard owner is a key marketing opportunity for the system.

Also, while some of the competing models are sold as all-in-one packages, they all appear fairly complicated to set up and actually use. One of the goals of UAVino was to lessen the time required to actually begin collecting data after the system is deployed on location. In order to accomplish this task, development focused on making the system straightforward and intuitive to use.

Finally, none of the competitors offer the use of a docking station or platform to recharge the system, which is critical due to the somewhat remote location of agricultural fields and the relatively low flight time of octocopters. Therefore, being able to successfully develop and implement a remote charging station was identified as a major competitive advantage for UAVino.

2.7 Key Systems Issues

2.7.1 Docking Station Charging Method

Three options were considered for the method by which the octocopter would recharge after landing on the docking station: inductive charging, charging via electrical contact plates, and charging via a battery connector. Concepts drawings for each of these charging methods and how they would integrate with the docking station and octocopter are available in Appendix E. While each of these designs had merits, charging via contact plates was ultimately selected because of its balance between ease of manufacturing and robust charging. The major concern with this design centered on the safety risk of exposed electrical plates, but it was decided that this risk could be

sufficiently mitigated by installing protective covering over the contacts and programming the charging software such that current only flows from the station when the octocopter is properly docked with it.

Inductive charging was particularly attractive because it would not require extremely precise landing capability, thus greatly simplifying the docking challenge. Additionally, inductive charging is currently being explored in a wide variety of industries and has even been demonstrated in drone recharging applications, so more resources would be available regarding how to create such a system. However, this method was not selected because although it has benefits, manufacturing the required electrical components would have proven difficult and the end results would not have been capable of transmitting enough current to recharge the octocopter battery in a reasonable amount of time.

Using a regular battery connector to recharge the octocopter would have been the safest option, as it would not include any exposed electrical connections. However, the precision of the docking algorithm required to mate the male and female ends of a battery connector prohibited it from being practical with regards to UAVino.

2.7.2 Docking Mechanism

The requirement of the docking mechanism was to ensure that the octocopter could reliably and safely land on the station platform such that the recharging mechanism engaged. Because the contact plate charging method selected required an accuracy of 2 inches, it was determined that a computer vision system alone could not deliver the precision needed. Thus, the final docking solution involves both a computer vision algorithm to bring the octocopter to the approximate position of the docking station and a mechanical mechanism to precisely position the vehicle on the recharging platform.

A wide variety of mechanical possibilities were considered that are available in Appendix E. However, because of a limited build time, the final solution needed be

straightforward. This fact helped focus design solutions towards strategies that involved simple, passive elements. The final design selected uses a set of guidance poles that extend from the base of the docking station in a tapered fashion. A pair of rings attached to the octocopter position themselves around these poles, locking the vehicle's lateral position, and the vehicle then descends onto the platform in a precise location. This design is illustrated in Figure 2.5.

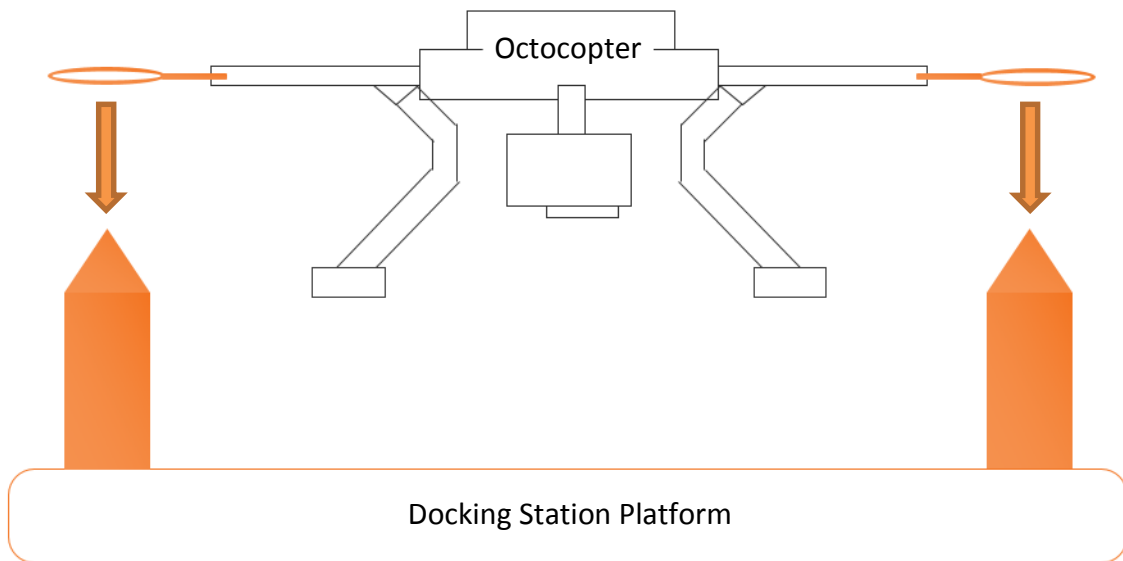


Figure 2.5: Overview of the octocopter ring and cylindrical cone docking method.

This mechanical docking system was extremely easy to manufacture and implement and proved reliable, provided that the computer algorithm positions the octocopter properly for the rings and tapered cones to mate. One initial concern with this method was the potential for vehicle propellers to accidentally strike the poles while the autopilot was working to correctly position the octocopter. However, this risk was mitigated in the landing algorithm software by ensuring that the vehicle does not descend below a height that would allow propellers to contact the poles until the correct lateral position is achieved.

2.8 System-Level Design Layout

Figure 2.6 shows a block diagram of UAVino's system components.

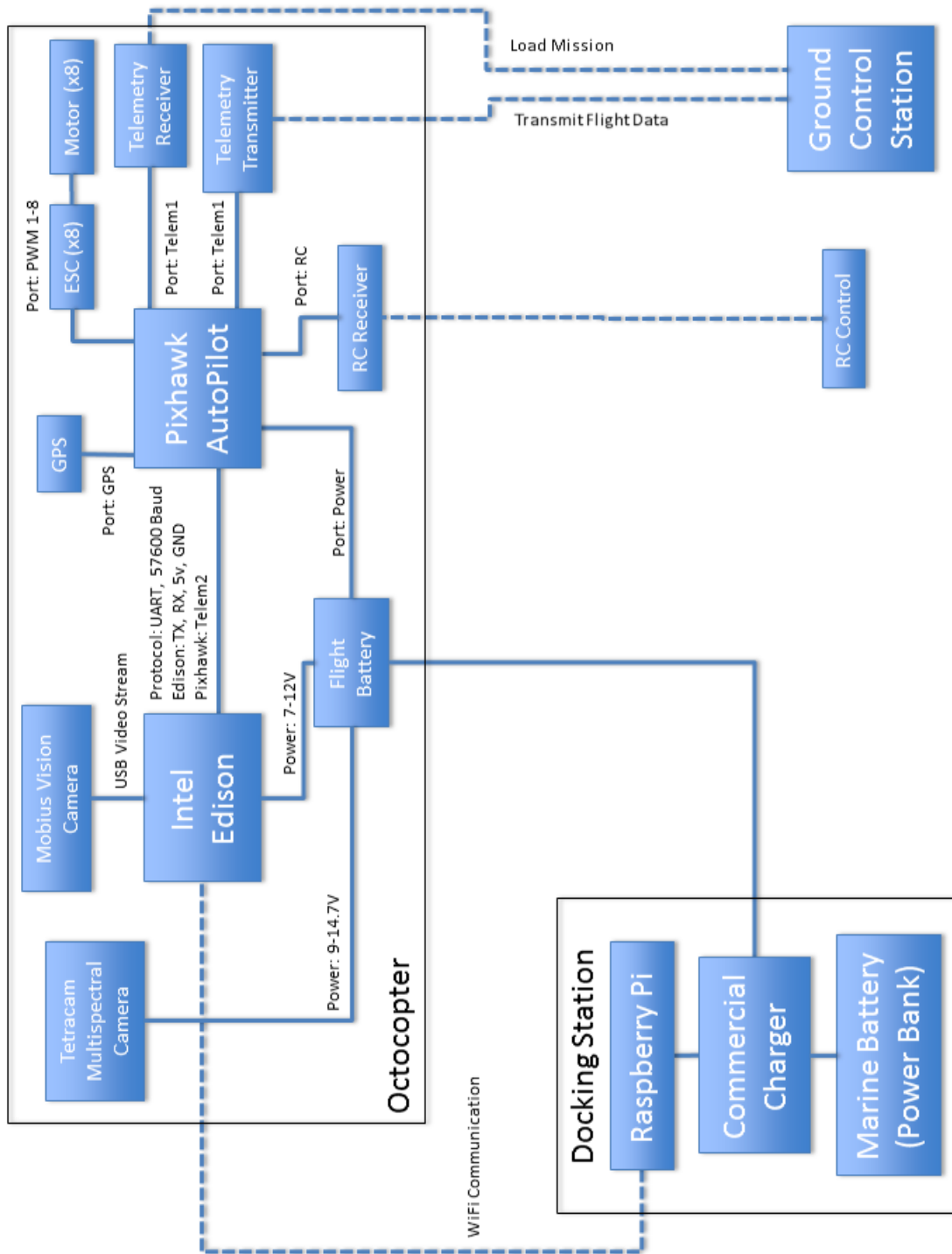


Figure 2.6: UAVino block diagram.

In general, the system is divided into the octocopter, the docking station, and the ground control station. The octocopter is treated as the hub of the system, which is powered by a lithium polymer flight battery and is centered upon an Intel Edison

microprocessor, which handles landing algorithm computation, and the Pixhawk Autopilot, which is responsible for flight control. When landed, the octocopter connects to a Raspberry Pi microcontroller on the docking station via WiFi in order to initiate charging. The station is powered independently by a 12V marine battery. During flight, the ground station sends and receives data via a telemetry radio and the operators are able to take manual control in an emergency via a 2.4Ghz radio system.

2.9 Team and Project Management

2.9.1 Project Challenges and Constraints

Of the many challenges associated with creating UAVino, one of the most difficult and crucial elements was how to automatically dock the octocopter with the docking station. While the team had several promising ideas for how to complete this mission task, there were no professional drone recharging stations that UAVino could draw from or compare against. Additionally, while the equipment and code for flying drones in predetermined paths is fairly advanced, these default sensors and algorithms were not precise enough to dock the octocopter consistently and reliably. Therefore, the degree to which the docking station and associated landing algorithms needed to be built and tested from scratch meant significant time and effort had to be budgeted for their development.

Beyond the design scope of the project, the physical distance of the vineyard customer and complicated setup required to conduct octocopter flights on Santa Clara University's campus made testing relatively difficult to conduct. In general, test flights required at least one week of preparation with and notice to the involved parties. Therefore, advanced planning and strict adherence to project deadlines was required in order to ensure that test flights were not wasted opportunities. Ultimately, the ability of all team members to pace their individual tasks and bring the required items together for test flights helped ensure success of the project.

With regards to the team in general, a major challenge was working together to manage the schedules of six individual team members taking classes for three different majors. Because of the limited times during which all members could meet and discuss team-wide issues, maintaining effective communication through email and cell phones was critical to ensure that individual items were progressing as needed. When all-team meetings were held, they focused on ensuring seamless integration between the different designs of computer, mechanical, and electrical engineers. Throughout development, it was imperative to review bottlenecks and resource contention to verify that one group's progress or lack thereof did not hinder another's.

Although team members were generally friendly and worked well together, great care was taken in electing team leaders and delegating tasks. The immense work associated with creating UAVino needed to be distributed in a way that made sense and remained fair to all members. Ultimately, the goal was to allow everyone to take ownership of specific parts of the project so that they could feel proud of the final product delivered at the end of the year.

2.9.2 Budget

UAVino's \$3,750 budget is comprised of grants from Santa Clara University's School of Engineering and the Silicon Valley Section of the American Society of Mechanical Engineers. Large expenses, such as the base 3D Robotics X8 octocopter and multispectral imaging cameras, were provided by the Santa Clara University Robotics Systems Laboratory and Intel Corporation, respectively. More budget details and specific cost breakdowns are available in Chapter 8 and Appendix F.

2.9.3 Timeline

Figure 2.7 shows an overview of UAVino's project timeline over the course of the academic year.

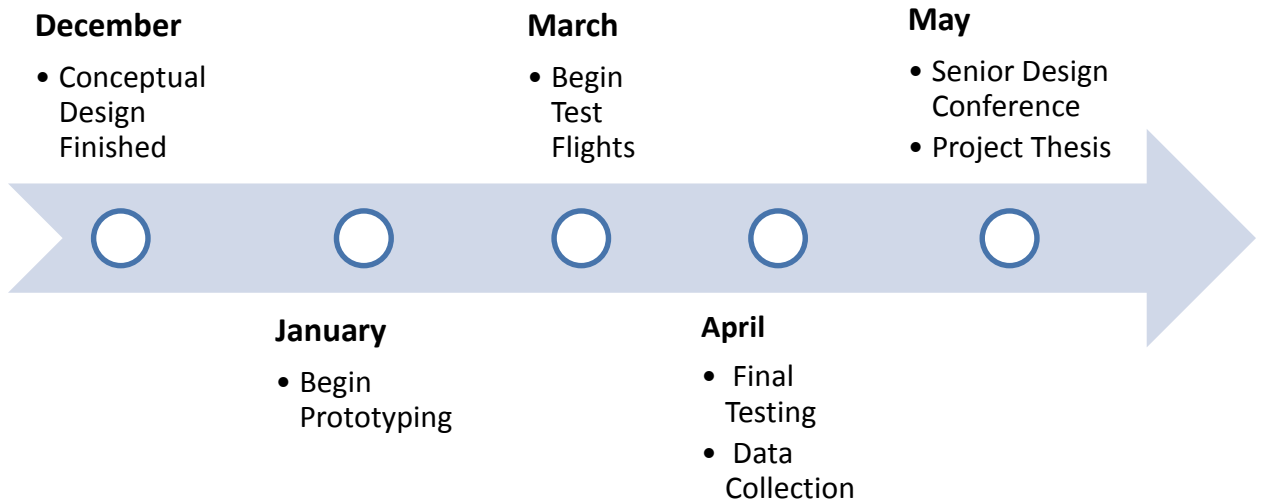


Figure 2.7: Overview of UAVino project timeline.

In general, the fall academic quarter was spent brainstorming docking solutions, researching current drone autopilot capabilities, experimenting with vision recognition software, and building a relationship with the vineyard customer. Following Thanksgiving, a major team meeting was held to finalize the conceptual design, particularly the docking station solution, so that a complete system picture was in place leading into winter break. Docking station manufacturing, landing algorithm development, and recharging circuit design and bench testing began immediately upon return in January. These build phases and core project development lasted through February, at which point system integration began. Basic test flights at local parks were conducted in early March and lasted until the end of the winter academic quarter. Actual vineyard test flights began in April, immediately at the beginning of spring quarter, and lasted through the remainder of the year as the system continued to be refined and developed. A detailed project Gantt chart is available in Appendix F.

2.9.4 Design Process

Because of UAVino’s interdisciplinary nature involving both mechanical and computer engineers, component design tasks were delegated to the group of engineers best geared towards creating a solution. At this level, members were responsible for

individually brainstorming solutions and then scheduling a meeting with impacted team members to discuss the merits of each idea. From these meetings, a final solution was created that tried to combine the positive aspects of each individual design. For solutions that required input from both computer and mechanical engineers, the entire team came together to discuss how well individual ideas would integrate. Typically, these individual and all-team brainstorming sessions were conducted weekly to streamline communication and ensure all options were considered before choosing a final design.

Some aspects of UAVino, such as the docking station and landing algorithm, were extremely complicated systems to develop, so the design process was a highly interactive one. Tweaks and adjustments occurred well into the prototyping phase, especially for the mechanical docking mechanism, in order to achieve an optimum solution.

2.9.5 Risks and Mitigations

Public safety is of the utmost importance when developing any engineering project. With regards to UAVino, safety is a particularly important consideration since drone malfunctions run the serious risk of damaging property or causing injury. To help mitigate hazards, significant effort was directed towards equipping the octocopter with both passive and active safety features. Ultimately, it is the team's responsibility to uphold safety as the project's most important concern and do what is necessary to ensure that the final product is as safe as possible.

Beyond the risks associated with system design, human error increases the possibility of malfunctions during operation. UAVino requires a series of complex steps for proper operation, and failure to follow these procedures greatly increases the chances of an accident. To reduce the potential error incurred by human operation, detailed flight procedures and checklists were developed so that a safe operation exists and can be followed during every deployment.

2.9.6 Team Management

This year's UAVino team was led by a student manager responsible for ensuring that key deadlines were met and that the overall project was progressing as needed. This manager also allowed for a single point of contact between team members, and the project's academic advisor, and the real-world customer. Below the student manager, UAVino was broken down into software and hardware teams, with one member overseeing each area. In order to complete the various tasks within the project, all team members were assigned various responsibilities and were tasked with working at an appropriate pace to complete the required work on time. Figure 2.8 shows the task breakdown of UAVino's team.

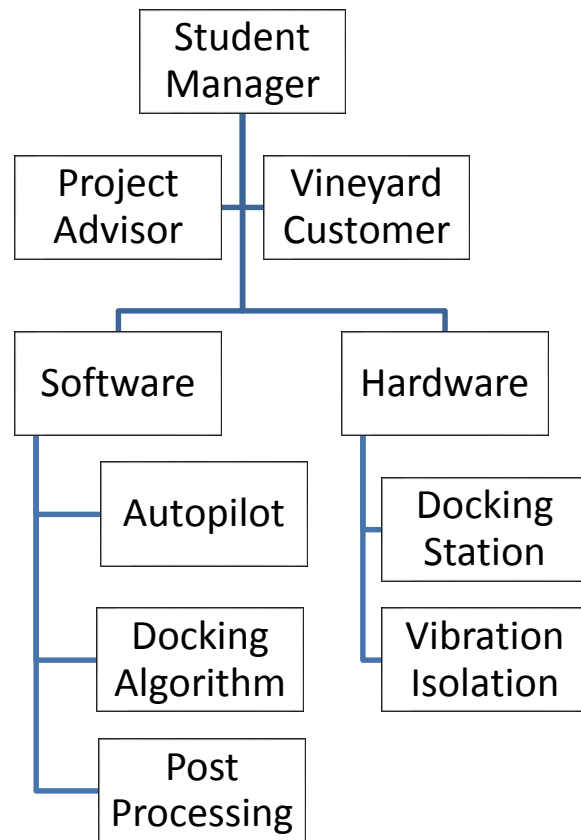


Figure 2.8: Breakdown of UAVino's team structure.

In general, meetings with the entire team occurred twice per week in order to review any matters relating to the project as a whole and resolve any concerns. One of these

weekly meetings included the project advisor to provide feedback and guidance as necessary. Meetings within the software and hardware sub-teams of the project occurred as needed and typically took place at least once per week. Conflicts were resolved by gathering those affected by the decision in question, discussing the matter in detail, and then voting on an appropriate solution.

Team dynamics played a crucial role in the way UAVino was managed. It was the responsibility of all team members to work towards creating an open and non-judgmental environment where everyone felt comfortable sharing ideas. To help meet this goal, effective communication between members was critical, meaning that all team members needed to respect and allow others to voice their opinions.

CHAPTER 3

Docking Station

3.1 Overview

The docking station is an extremely important aspect of UAVino, as it is the component that most separates the system from competitors. Unlike other drone crop inspection systems, which are limited in battery life, the docking station allows UAVino's octocopter to repeatedly recharge until the mission is complete, thus greatly expanding range and duration. Ultimately, it is the centerpiece that enables UAVino to autonomously map a vineyard and provide customers with the agricultural information necessary to make logical decisions.

The docking station serves as a major technological innovation for UAVino because no comparable product is currently available that allows both precise drone landing and recharging capability. Over the past few years, a variety of companies have made great strides towards achieving both of these goals independently, but rarely have the two technologies been paired together. By combining both landing and recharging functions, UAVino acts as a stand-alone system that offers easy and affordable access to agricultural crop monitoring.

3.2 Requirements

The key requirement of the docking station is that it allows the octocopter to land precisely such that an electrical connection between the station and octocopter is established and the flight battery is able to recharge consistently and reliably. To accomplish this goal, a computer vision based landing algorithm on the octocopter works in conjunction with mechanical devices on the docking station to properly orient and land the vehicle on the charging platform. The mechanical docking mechanism consists of tapered cones on the charging platform and a pair of rings on the octocopter. When landing, the vision guided algorithm works to align the octocopter's rings with the

tapered cones, effectively fixing the lateral position of the vehicle, so that it can then descend smoothly onto the platform in the precise location needed.

Recharging the octocopter's battery means that high levels of current must flow between the docking station and octocopter, which give rise to a variety of safety requirements. To ensure that the docking station is safe, master safety switches, warning lights, and fuses are built into the charging circuitry to help reduce the risk of electrical shock and general component damage. Also, due to the flammability risk associated with charging lithium polymer batteries, safety algorithms are included in the charging software that shut off current flow to the battery should any anomalies occur.

Finally, the docking station must be rugged enough to withstand its intended operational environment. Although the station is not designed to be permanently situated in a field, it does spend extended periods of time outside and thus must be able to withstand repeated exposure to the elements. This requirement is met largely by material choice. For example, the docking station is made almost entirely from wood so that it does not experience rust and other deteriorating effects that upset non-natural materials. Although certain elements of the station, such as the charging contacts, are more susceptible to the elements, the station as a whole is designed to endure the climate it experiences on a day to day basis.

3.3 Options and Tradeoffs

One of the most important options for the docking station is the method by which it recharges the flight battery, as the design for many subsequent components are derived from this decision. During the conceptual design phase, the two most logical options considered were charging via induction with a custom charge controller or via a series of contact plates interfaced with an off-the-shelf battery charger. While each of these methods had benefits, the contact plate option was chosen because it was the most practical to implement and sustain. Inductive charging did offer a variety of attractive options, namely a decreased reliance on landing accuracy since connecting small

electrical plates on the octocopter and docking station would no longer be required. However, inductive charging would be slower than contact charging and since a fast recharging time was a goal to maximize mapping capability, this option was bypassed.

Once the charging plate option was chosen, an additional decision arose regarding how to best facilitate battery charging. One option involved mounting a charge controller on the drone itself, while another simply used a commercial battery charger housed in the docking station. A drone-mounted charging controller seemed promising, as it would allow for fewer contacts and a larger surface area, therefore reducing the accuracy required for landing. However, this option was ultimately avoided because of electrical circuit complexity and added vehicle weight. The final solution uses a programmable commercial battery charger located inside the docking station to facilitate charging and the only additional weight to the drone are five 20-gauge wires soldered onto the contact feet that lead directly into the balancing plug of the flight battery.

Lastly, the general docking station design included a high level tradeoff regarding overall complexity. Having components such as the docking station frame professionally manufactured and assembled was an option to increase the overall appearance of the design and ensure functionality. However, contracting with professional machine shops would increase cost and take longer to construct compared to team members building on-site during spare time. Given the extremely short timeframe of the project and the necessary functionality of the docking station, it was concluded that designing and manufacturing all components in-house would be more conducive to meeting design and build deadlines. Therefore, design also focused on using easily purchased off-the-shelf parts to reduce cost and increase ease of manufacturing.

3.4 Design

3.4.1 Docking Station Frame

At its most basic level, the docking station is a rectangular table with a drawer to house charging electronics and a platform for the octocopter to land on. Although relatively

simple, the station is well-suited to deliver the functionality required. Figure 3.1 shows an overview of the completed docking station.



Figure 3.1: Completed docking station

Figure 3.2 shows a CAD model of the docking station frame, the sides and floor of which are made from 3/4" maple plywood. The frame also contains a 3/4" square alignment bar made from poplar that runs across the top and mates with a corresponding alignment bar on the platform. The entire frame is supported by four 18" high legs made from 4" square Douglas fir. Standard wood screws hold the frame together.



Figure 3.2: Docking station frame CAD rendering

A pivot foot mounted on a length of 1/4-20 all-thread is screwed into a matching threaded insert on the bottom of each station leg. The height of each individual foot can be adjusted and this feature, combined with the fact that each foot rests on a pivot swivel, helps level the station if it rests on uneven ground. A detail of the adjustable foot is shown in Figure 3.3.

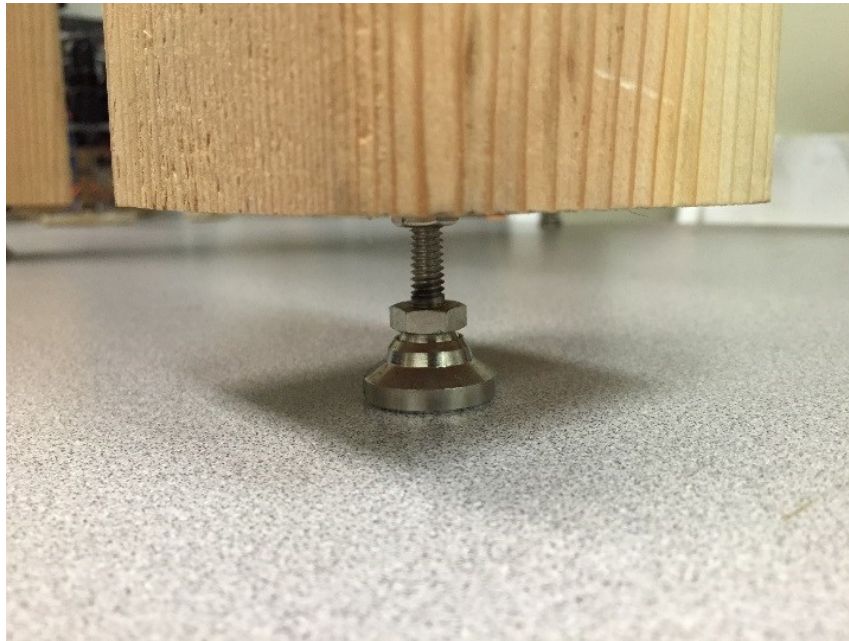


Figure 3.3: Docking station adjustable foot detail

Although the leveling feature is useful, its effect is minimized by the small size of the feet, which sink into loose soil due to the station's weight. Thus, while the general design is correct, a future improvement needed is to increase the surface area of each foot in order to mitigate this effect.

The drawer, shown in Figure 3.4, measures 22" x 22.5" x 8" and is made from the same 3/4" maple plywood as the docking station frame. It is equipped with a handle for easy operation and is held together with standard wood screws. The drawer is not mounted on drawer hinges, but does slide in and out of the frame freely so that it can be completely removed in the event that total access is required to the electronics inside. Figure 3.4 displays a CAD image of the drawer along with a picture showing the interior electronics.

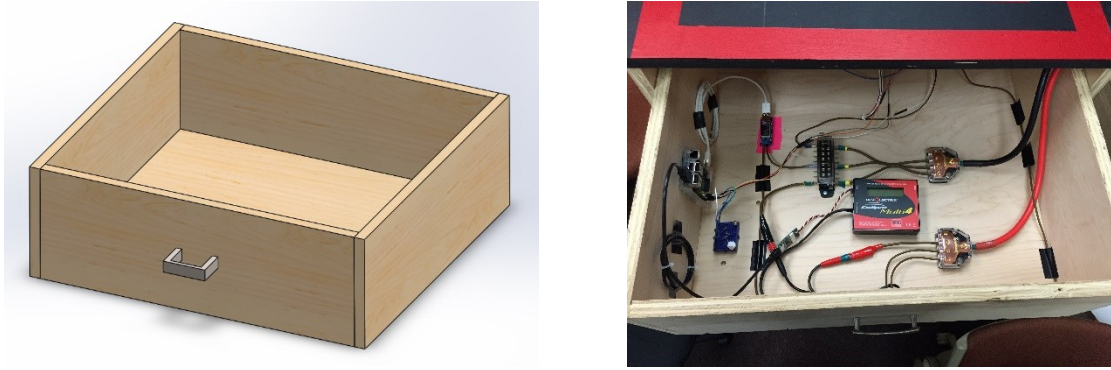


Figure 3.4: Docking station drawer and interior electronics

3.4.2 Docking Station Platform

The station platform is made from a 2' x 3' piece of 3/4" maple plywood and is painted a distinct red and black target shape so that the octocopter's vision camera is able to recognize the station from the air. The platform dimensions are derived from what space is required to comfortably accommodate the octocopter, the cone mechanism, and charging plates. Shown in Figure 3.5, the platform is fully removable and serves as a secondary means of accessing the charging and safety circuitry inside the station should the operator not wish to remove the drawer.

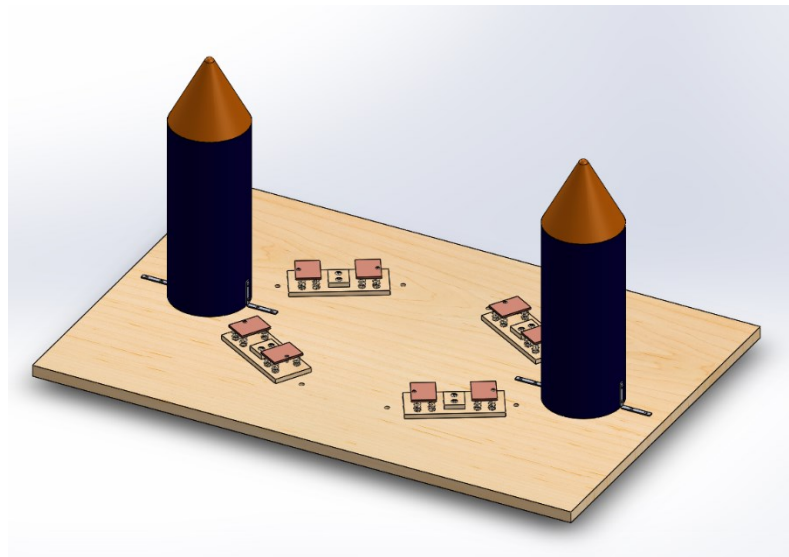


Figure 3.5: Docking station platform CAD rendering

To allow for proper alignment each time the platform is placed on the station, alignment bars are attached to the underside of the platform that mate with the inside surfaces of

the station frame. This allows the platform to be placed in the exact same location each time and not experience lateral motion during operation. A detail of this feature is shown in Figure 3.6.



Figure 3.6: Docking station platform alignment bars

3.4.3 Guidance Cones

The purpose of the guidance cones are to mate with the rings on the octocopter so that the vehicle's lateral position becomes locked and it can then land on the station platform in the proper orientation. The motivation behind locking the vehicle's position arises from ground effect, which adversely affects the flight characteristics of the vehicle as it comes close to the ground and makes it difficult to land precisely. Additionally, depending upon how hard of a landing the flight controller makes, the octocopter may land where desired but then bounce off target. By ensuring that the octocopter always lands in the same position and in the same configuration, the cone mechanism helps create a reliable system that ensures the vehicle correctly engages the charging contact plates with each landing.

The initial cone design called for hard plastic soccer cones to be mounted atop ABS pipe, as the smooth plastic finish would minimize friction between the octocopter's rings and the cones during docking. However, attaching these cones to the ABS pipe in an acceptable manner proved difficult, as there was little surface area on the cones to work

with and the glue used created a lip at the joint. During testing, this lip caused the octocopter rings to catch and therefore did not allow for a smooth landing or takeoff. This initial design is shown in Figure 3.7.



Figure 3.7: Initial docking station cone design using hard plastic soccer cones.

Ultimately, the plastic cones were abandoned in favor of a redesign. The final version, shown in Figure 3.8, uses open cell Styrofoam cones.



Figure 3.8: Docking station guidance cone

Styrofoam is ideal for this application because it is strong enough to hold shape and combat the loads experienced by the octocopter's rings, yet weak enough that should the octocopter accidentally hit the cones while docking, the propellers would slice through the cone cleanly rather than become lodged in it and likely cause a crash. Each cone is glued atop a 9" long section of 4" diameter ABS pipe, which is then connected to the platform via two right angle brackets and wood screws. The Styrofoam is coated with epoxy resin to help smooth the rough Styrofoam finish and to prevent general wear and tear over time.

3.4.4 Contact Plates

Recharging is facilitated through eight copper plates that the octocopter rests on after landing. These plates are made from copper and measure 2" x 1.5" x 0.125". Each plate is glued to four small steel compression springs to ensure that it remains in contact with the corresponding electrical connection on the octocopter. A set of two contact plates rest in a wood bracket that is fastened to the docking platform with wood screws. Figure 3.9 shows a detailed view of a contact plate pair and Figure 3.10 shows the same contact plate with the octocopter foot resting on it.

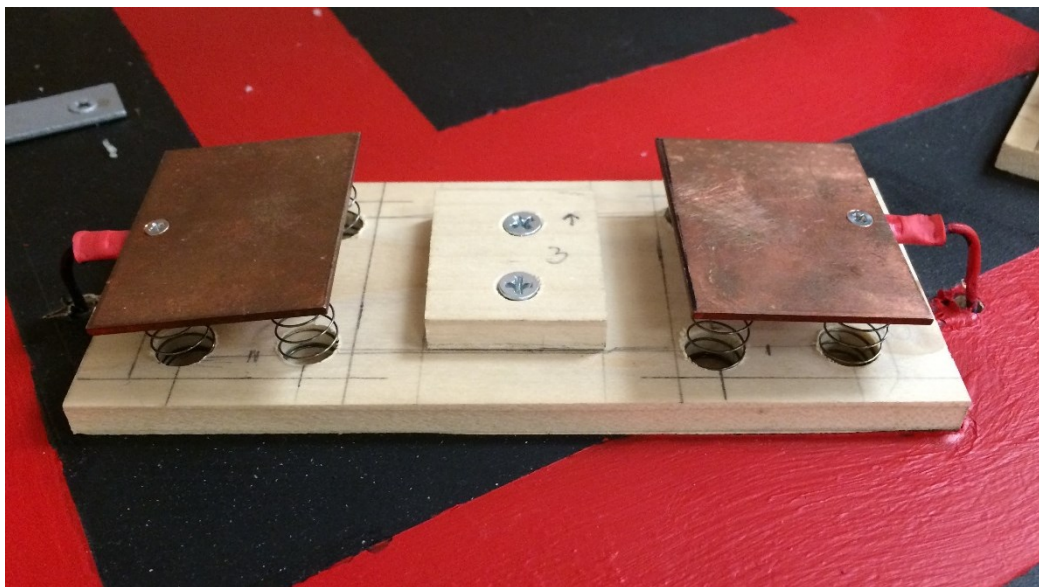


Figure 3.9: Two docking station contact plates fastened to compression springs and attached to the station via a wood bracket.

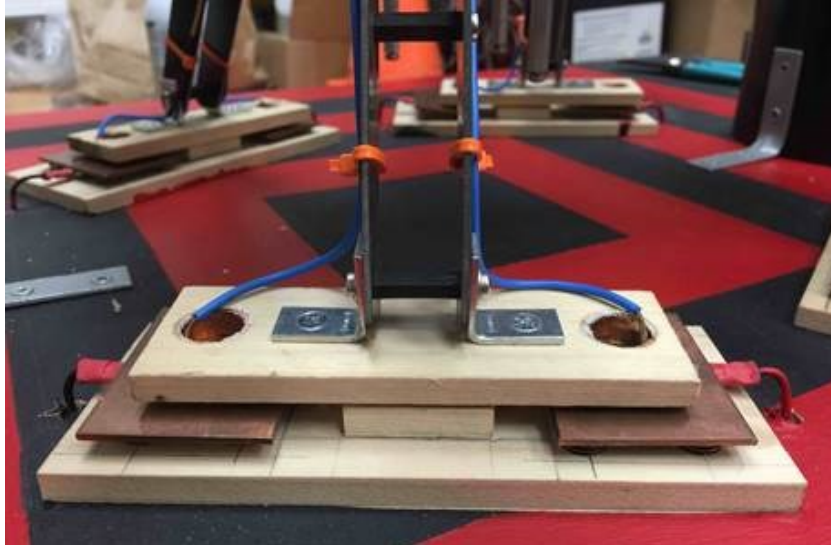


Figure 3.10: Octocopter foot resting on docking station contact plates.

Each copper plate is connected electrically to a wire via a crimp-on connector and small machine screw. These wires then run through the platform and into the charger housed within the station.

3.5 Recharging Circuitry

3.5.1 Overview

The octocopter is powered by a 4 cell 14.8V 6000mAh lithium polymer battery, as shown in Figure 3.11.



Figure 3.11: Octocopter battery

Lithium polymer batteries require balancing, meaning that each cell needs to be charged independently in order to allow for safe, even charging. This method requires a specially designed “smart” balance charger that is capable of monitoring the voltage and current of each cell. Such charging is facilitated in the docking station through the Revolextrix CellPro Multi-4 battery charger, which is shown in Figure 3.12.



Figure 3.12: Revolextrix CellPro Multi-4 battery charger

This charger was selected because it provides serial input and output using UART communication at a 19200 BAUD rate. This protocol is used to programmatically start, stop, and provide status regarding battery charging. The contact plates to each of the contact feet on the drone are connected directly to this commercial charger. This charging solution is ideal, as housing the CellPro Multi-4 in the docking station eliminates the need for any additional hardware on the drone and thus helps minimize the weight of the vehicle.

Two microprocessor options were considered to interface with the CellPro Multi-4 and control the docking station: the Intel Edison and the Raspberry Pi. Details of both processors are available in Table 3.1.

Table 3.1: Intel Edison and Raspberry Pi features comparison.

	Intel Edison	Raspberry Pi
USB Ports	1	4
GPIO Pins	20	40
Processor Architecture	Dual-Core x86 @500mhz	Single-Core ARM @ 700mhz
Random Access Memory	1GB	512MB
WiFi	Integrated	External USB (Separate)
Display Out	None	HDMI

Ultimately, the Raspberry Pi was chosen for its ease of use, as it features a well-supported package manager that allows for easy installation of required drivers and subcomponents. Additionally, the Raspberry Pi is geared more towards the operational needs of controlling the docking station, as it acts as a centralized hub. The Intel Edison, on the other hand, is more suitable as a powerhouse computational device. A Raspberry Pi is shown in Figure 3.13.

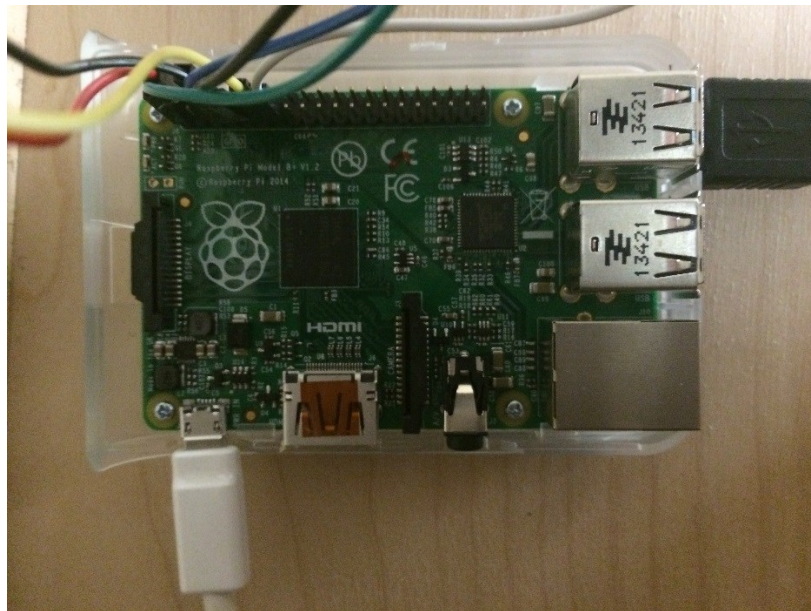


Figure 3.13: Raspberry Pi microcontroller.

The Raspberry Pi microcontroller interfaces and communicates with the CellPro Multi-4 charger via Revolectrix's FUMI-3 FTDI USB interface. This solution is straightforward because the Raspberry Pi natively supports FTDI virtual COM port drivers. The Raspberry Pi connects to the FTDI interface via USB, and the FTDI interface is connected directly to the charger. When connected, the Raspberry Pi is able to send commands to and

retrieve responses from the charger. The charger regularly sends a status string, which contains various information about the charger and the charge status so that the Raspberry Pi can check for anomalies and terminate charging if needed.

3.5.2 Wireless Network

In order to facilitate communication between the octocopter and docking station, the Raspberry Pi is configured as a wireless access point. This feature enables the Intel Edison microcontroller on the octocopter to establish communication via WiFi after landing in order to begin charging. The Raspberry Pi uses a Netis WF2190 AC1200 Wireless USB adaptor, shown in Figure 3.14, which eliminates the need for an external wireless router.



Figure 3.14: Netis WF2190 AV1200 Wireless USB Adaptor

The Raspberry Pi is configured to function as a WiFi access point by acting as a router following Dynamic Host Configuration Protocol (DHCP). The microcontroller runs a DHCP client called uDCHP and has a static IP address on which the client listens for connections. A client called hostAPD sets up a wireless access network with a secure WPA encryption. The Raspberry Pi also runs software called iptables, which enables Network Address Translation (NAT) to allow for multiple devices to connect to the

Raspberry Pi without individual devices fighting for resources that would otherwise be bottlenecked by atomic availability.

3.5.3 Charging Routine

Figure 3.15 shows the software flowchart for the recharging procedure.

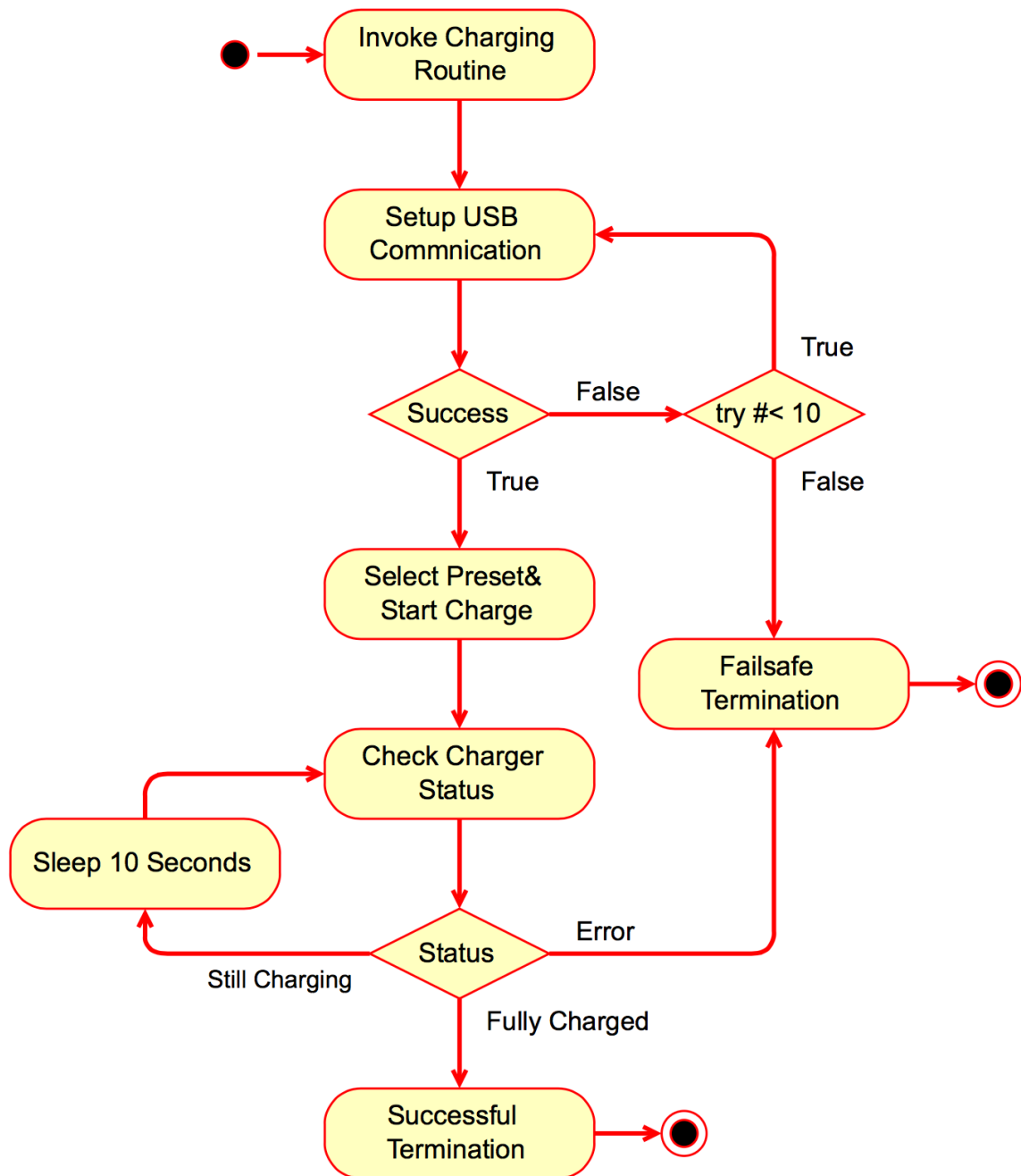


Figure 3.15: Charging routine software flowchart.

The ideal scenario is for the Intel Edison on the octocopter to automatically connect to the Raspberry Pi's wireless hotspot and then request charging to be initiated. However, this automated wireless connection has not yet been implemented. The remaining charge routine procedures as described in Figure 3.15 are implemented in a Python recharging script. Once the script is invoked, the Raspberry Pi interfaces with the CellPro Multi-4 charger through the USB FTDI serial interface to begin charging. Initially, the code attempts to establish a connection with the USB FTDI serial interface. If communication fails, it repeats up to ten attempts before entering a failsafe state.

Once communication with the charger is established, the Raspberry Pi sends a series of commands that selects the desired charge preset. The charge preset ensures that the charger operates within the required specifications, such as number of cells and overall capacity, for the octocopter's flight battery. Afterwards, the charger sends the commands that initialize charging. While the charger is not finished charging, the Raspberry Pi checks the status of the charger every ten seconds. Each time the Raspberry Pi checks the charger status, it monitors for an error code, which is signaled by the battery in the event an anomaly occurs. An error code can be generated for a number of reasons, such as if docking station contacts become shorted or disconnected, or if the charger's supply voltage or current falls outside of acceptable parameters.

When the Raspberry Pi detects that the charger is finished charging, the Python script ends in the success state. In the future, this success state will need to signal the octocopter to perform its takeoff routine. However, the Python script terminates in the failsafe state if it detects an error status from the charger's status string at any point during charging. Whenever the system enters the failsafe state, the charger becomes inactive and the octocopter remains on the docking station indefinitely. The charging source code is available in Appendix K.

3.5.4 Power Delivery

The entire docking station is designed as a standalone system: one that can operate independently of external power sources and wireless network requirements. This

capability arises from the fact that the station is meant for deployment in remote and unaccommodating locations, namely agricultural fields. The system is powered by a 12V 60AH marine battery, which is spill-proof, high capacity, and ultimately maintenance free. 2-gauge wires connect the marine battery to the docking station, where power is divided using two four-point power distribution blocks. All parallel connections to the marine battery have fuses in the event of a short circuit or other anomaly.

Figure 3.16 shows an overview of the charging station circuitry.

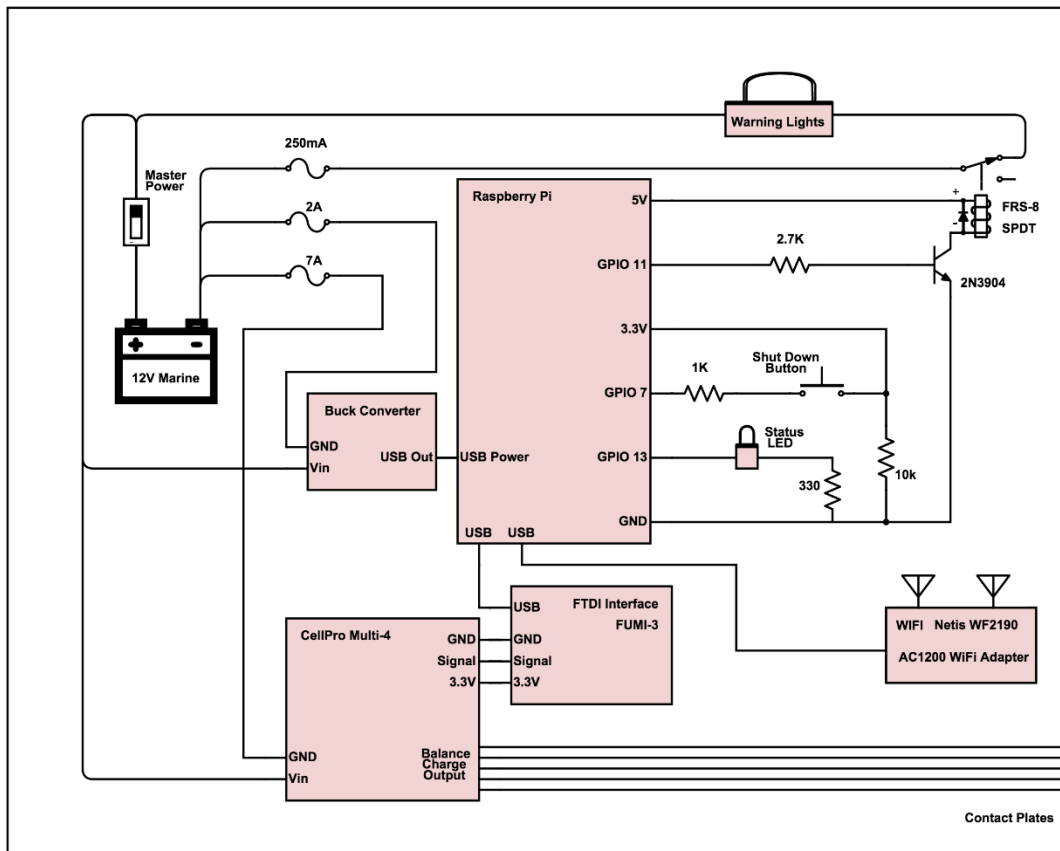


Figure 3.16: Docking station circuit diagram

Although the CellPro Multi-4 charger and LED warning lights can be powered directly from the 12V marine battery supply, the Raspberry Pi requires a steady 5V supply with a max current draw of 2A. In addition, this source must reach the Raspberry Pi via a USB cable with microUSB connector. This requirement is met using a Drok DC/DC buck converter with USB output, which is capable of providing a regulated 5V output across a

4.5V-40V input range. The particular buck converter used contains a digital display indicating the marine battery voltage and thus allows for visual voltage monitoring without any additional equipment.

Figure 3.17 shows the docking station electronics with annotations identifying each component.

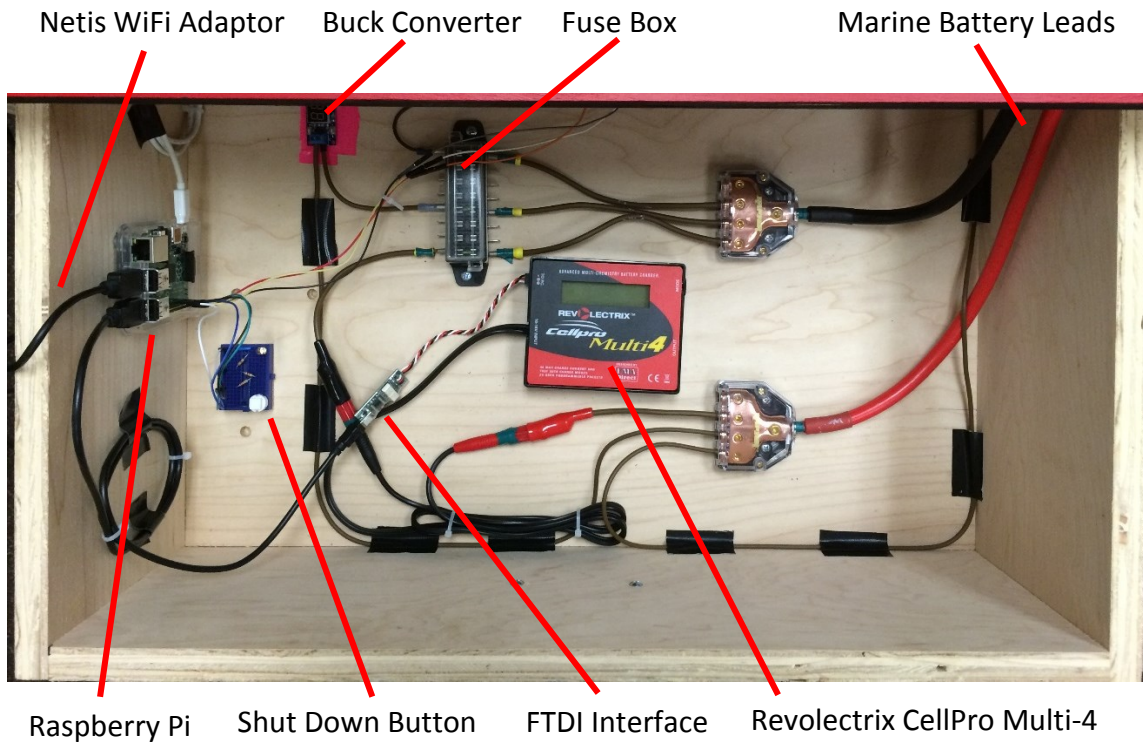


Figure 3.17: Docking station electronics with each component identified.

3.5.5 Safety

Current levels near 3A are needed to recharge the octocopter quickly and therefore electrical safety of the docking station is paramount. A wide variety of precautionary systems have been installed to allow for safe operation. One such system is a master switch that instantly shuts off power to the entire station in the event of an emergency. Before the 12V marine battery is able to supply any power to the station, this switch must be physically engaged, which helps serve as a reminder to users whether or not the system is live. Along with the safety switch, two large red LED lights are located on the docking station platform. These lights power on when the octocopter is charging so

that operators know the contact plates are not safe to touch. These lights are activated using the Raspberry Pi and a 5V SPDT relay in conjunction with a 2N3904 transistor. Lastly, each power line running in parallel between the two contacts of the marine battery is protected with a fuse.

3.6 Testing and Verification

Autonomous landing tests were not conducted due to difficulties with the vision landing algorithm, which are discussed in Chapter 5. However, manual flight tests were conducted in an attempt to gauge the effectiveness of the cone and ring system. For these tests, the octocopter was placed on the station and a series of trials were conducted in which the octocopter took off from the station platform. To gauge the success of each of these trials, team members closely observed the rings as they slid up the cones upon takeoff. If the rings did not get caught at any point during the takeoff, the trial was considered a success. Out of the five trials conducted, four were successful. There was one trial where the octocopter briefly got stuck, but was able to recover and continue its liftoff. In general, the octocopter was easily and smoothly able to ascend up the cones and lift off from the station platform. These successful tests were able to demonstrate the proper functionality of the cone and ring mechanical docking mechanism.

To determine charging performance of the docking station, a test was conducted to compare two charging scenarios. In one setup, the octocopter's battery was charged via the docking station's contact plates and the built-in CellPro Multi-4 charger, which was powered via the deep cell marine battery. In the second scenario, both the contact plates and marine battery were eliminated. Instead, the octocopter's battery was charged by plugging it directly into the CellPro Multi-4 commercial charger, which was plugged into a standard wall outlet for power. The goal of this test was to compare the charging time of the docking station to a more traditional charging method that would be employed if the system were operated manually by humans instead of

autonomously. The results of these test are shown in Figure 3.18, which depicts charge curves using each method.

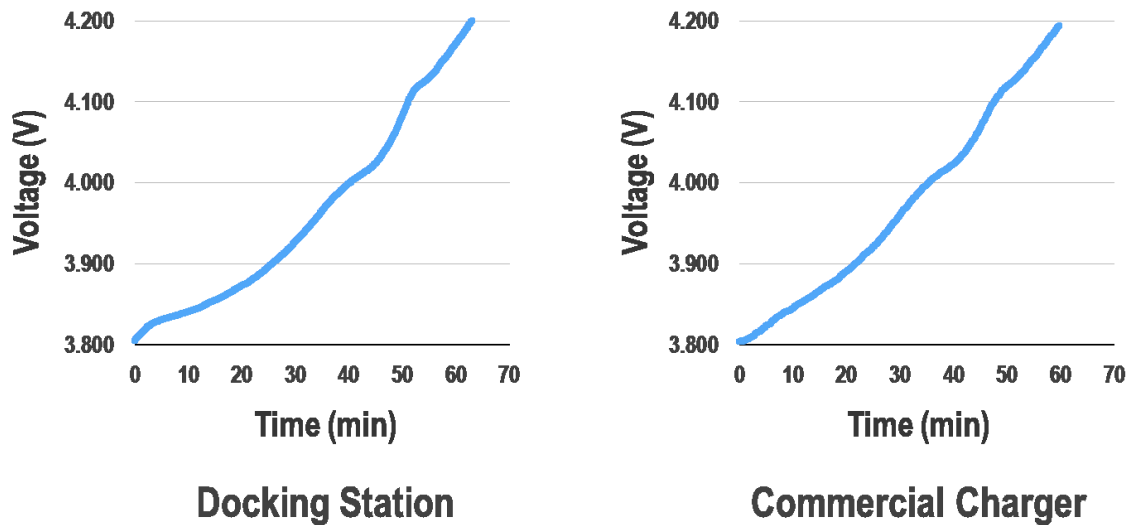


Figure 3.18: Docking station charge performance results.

The fact that the charge curves for the docking station and the control test are nearly identical verifies that the docking station can effectively act as a stand-alone system and charge the octocopter’s battery just as efficiently as if it was charged via an at-home setup using a standard wall outlet. Although charge time varies based on the initial voltage level of the battery, Figure 3.18 demonstrates that charge times using the docking station are close to 60 minutes, which is under the 90 minute goal set as success criteria. This relatively fast recharge time allows for increased coverage capability and data collection and agricultural crop monitoring. Ultimately, the insight gained in these charge tests substantiate the design choices made when creating the charging method for the docking station and prove that the docking station can act as a high performance stand-alone platform for UAVino’s autonomous operation.

3.7 Charging Future Work

Although the charging routine has been written, the Intel Edison is not yet programmed to automatically connect to the Raspberry Pi's WiFi hotspot and request charging. Future teams will need to implement a network socket server on the Raspberry Pi to

constantly listen to a network port for any incoming socket connections. Then the Intel Edison can be programmed to connect to the Raspberry Pi's listening socket to establish communication between the octocopter and docking station. The socket server on the Raspberry Pi should be integrated with the Cellpro Multi-4 serial communication and charging code. Ideally, it should also be implemented in Python so that the serial communication code can be imported as a module and the charging script can be directly embedded within the socket server code itself. The socket server should also constantly monitor the socket connection and halt the charge on detection of a lost connection. An easy option would be for the Python script to manage a Linux process control system such as Supervisor, which would launch the script when the Raspberry Pi is powered on. The Supervisor process control system also provides logging and automatic process restarting in the event that the Python socket server code crashes.

The charging system has been tested, but not with the drone powered on. When flight battery is connected and the vehicle is idle on the docking station, it still draws 500mA of current. The system may still work with the discharge leads connected, but the CellPro Multi-4 Charger will likely need a preset set to a lower charge voltage in the event that the battery voltage drops lower as a result of the power drawn. Additionally, if the leads are still connected, the battery might never hit 100% capacity as seen from the charger, unless the stopping float voltage is set lower. This problem may also be solved by having a docking station controlled relay placed in series with a discharge leads on the battery. In this situation, the docking station can activate the relay via the extra contact feet on the drone, enabling the octocopter to effectively be turned off while charging.

Furthermore, the charge time may be improved by using an alternate commercial battery charger. One such option is the Revolextrix CellPro PowerLab 8 serial charger, which has the ability to charge from the battery's discharge leads as well as from its balancing cable. The PowerLab 8 charger has a much higher charge rate, but longevity concerns and limited heat dissipation of the battery must be taken into consideration.

3.8 Summary

The docking station is a stand-alone platform that provides a unique solution for UAVino's autonomous docking and recharging. The station uses a static, mechanical cone mechanism that mates with rings on the octocopter to allow the vehicle to smoothly descend onto the station's platform. Once landed, the octocopter rests on spring loaded copper contact plates, which allow the octocopter's onboard battery to establish an electrical connection with the docking station to facilitate recharging.

In its current state, the docking station has demonstrated functional recharging capability that is comparable to recharging performance from a standard wall outlet. Additionally, WiFi communication framework between the docking station and octocopter has been laid out in order to initiate charging, although this system has yet not been fully implemented and tested. The mechanical ring and cone mechanism has successfully been manually tested, although the system is not yet capable of autonomous flight.

CHAPTER 4

Octocopter Additions

4.1 Overview

Although a variety of off-the-shelf multirotor vehicles are available that provide a wide range of capabilities, no platform was suitable for UAVino that did not require modifications and additions. In order to allow for autonomous landing and recharging capability as well as multispectral data collection, a variety of components have been designed and added to the octocopter's main frame. In general, these additions include a set of rings that mate with the cone system on the docking station, a mounting plate for the multispectral imaging camera, electrical contact feet that enable battery recharging, and a vision camera and microprocessor that handle automated landing capability. The overarching goal with these modifications and additions was to provide the required functionality while keeping added weight to a minimum. Most multirotor drones have a flight time near 15 minutes and in order to maximize UAVino's mapping capability, minimizing vehicle weight was critical.

The base flight vehicle is 3D Robotics' X8 octocopter, which is shown in Figure 4.1.



Figure 4.1: 3D Robotics X8 octocopter drone

This octocopter is widely popular as a hobby-style multirotor drone because it is highly modular and is based upon software that is open source. The drone comes equipped with a Pixhawk autopilot board, GPS and compass modules, and built-in telemetry so

that it is nearly ready-to-fly when received and very little assembly is required. These built-in functions made the X8 the ideal choice for UAVino, as it allowed development to focus on the required modifications rather than on trying to include basic radio and autopilot functionality.

Table 4.1 lists the major components added to the vehicle and their weights, as well as the total weight of the flight-ready octocopter.

Table 4.1: Octocopter and additional component weights

Item	Quantity	Item Weight (g)	Total Weight (g)
Unloaded Octocopter	1	1965	1965
Flight Battery	1	605	605
Multispectral Imaging Camera	1	90	90
Vision Camera	1	40	40
Ring Bracket	2	40	80
Vibration Isolation Camera Mount	1	195	195
Charging Foot	4	35	140
Intel Edison Microprocessor	1	75	75
Total Octocopter Weight			3190

4.2 Requirements

The modification with the most stringent requirements is the set of guidance rings, as this component is critical for automated docking. These rings mate with corresponding cones on the docking station in order to fix the lateral position of the vehicle and allow it to descend smoothly onto the platform. When the octocopter lands and takes off, asymmetric thrust or external factors such as wind cause the rings to push or pull against the cones and it is crucial that the design of this system withstand these loads, which are somewhat unpredictable. Materials with both lightweight and high strength characteristics are used in the ring mounting system, but finite element analysis was conducted in order to verify the design.

In addition to attaching the multispectral imaging camera to the octocopter, the camera mounting requirement is that it damps out vehicle vibrations so that the camera is able to take clear, high quality pictures. As they rotate, the octocopter’s motors are a source

of high frequency vibrations that travel through the vehicle's frame and inherently affect anything connected to it. In order to quell these frequencies, the camera mount includes passive vibration isolation components that act to stabilize the camera and limit the motion it experiences.

The key requirement for the octocopter's contact feet is that the component successfully establishes an electrical connection with corresponding contacts on the docking station. This connection must be established consistently and reliably with each landing. Due to the high levels of current that are needed to charge the octocopter's battery, the contact feet are made from a material with a high electrical conductivity and that will not fail due to excessive heat.

4.3 Ring Mounting System

Figure 4.2 shows an overview of the ring mounting bracket.



Figure 4.2: Ring mounting bracket detail.

The ring itself is made from a 5" diameter bamboo embroidery hoop and is connected to a 1/4" square carbon fiber rod via two aluminum angle brackets and a plated steel M4 screw. The carbon fiber rod is epoxy set in a custom machined acrylic block that is attached to the octocopter frame via two stainless steel M3 screws.

The choice of materials for the ring system was crucial in order to provide the required strength without adding unnecessary weight. Carbon fiber and bamboo were ideal for this application, as it provides a sturdy structure that weighs just 40g.

4.3.1 Finite Element Analysis Overview

Since the ring mounting system is so critical to the success of UAVino's autonomous docking capability, detailed analysis was conducted on the bracket system to determine how it would perform in operation. Loading situations for the ring assembly were selected based upon what types of forces the actual octocopter might experience when landing. In reality, these forces are hard to predict when considering how environmental factors, such as wind and ground effect, might influence the vehicle's flight. Ultimately, it was decided to test a horizontal and vertical load applied on the bamboo ring, as well as a rotational torque applied at the joint of the ring and carbon fiber rod. These situations are shown in Figure 4.3.



Figure 4.3: Loading conditions applied for ring mounting bracket finite element analysis.

During UAVino's operation, most loading phenomena on the ring bracket are a combination of these three conditions. By testing each situation individually, the intent was to determine which areas of the structure are weakest and what types of loading scenarios are cause for concern.

For each of the two directional force loading conditions, a 2 pound force was applied. This load was determined based on the worst case scenario that the ring structure would need, which is half the entire weight of the vehicle, approximately 7 pounds. Because there are two ring brackets, this load is assumed to be shared equally.

4.3.2 Finite Element Analysis Expectations

It was expected that the analysis would show the most critical joint is the connection between the bamboo hoop and carbon fiber rod, as the aluminum brackets and bolts joining these two components bear the brunt of any load applied to the bamboo hoops. In particular, the corner of the bend in the aluminum brackets is an area of concern, as stress concentrations resulting from such geometry are likely to magnify the stresses seen in this area.

The failure criteria for the ring and carbon fiber rod joint was set as any stress exceeding the yield strength of the aluminum bracket, as such a stress would permanently deform the bracket and prevent the ring from properly aligning with the rest of the docking system.

4.3.3 Finite Element Analysis Results

Figure 4.4 shows the stress distribution in the assembly when the ring is subjected to a 2lb horizontal load applied towards the acrylic mounting bracket. This loading scenario results in a maximum compressive stress of 1674psi and a maximum tensile stress of 6905psi. Figure 4.5 shows the same stress distribution, but is focused on the joint between the bamboo ring and the carbon fiber rod.

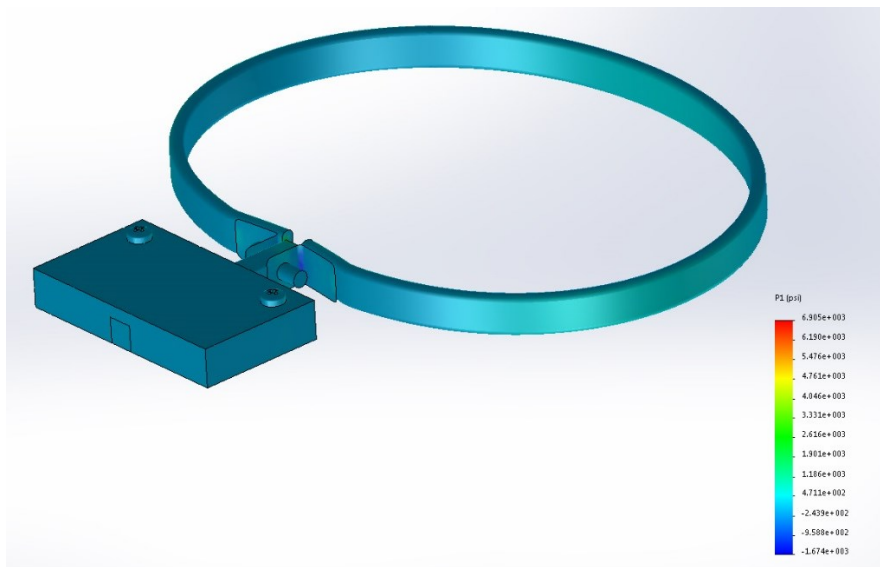


Figure 4.4: Stress distribution in the ring bracket resulting from a 2lb horizontal load.

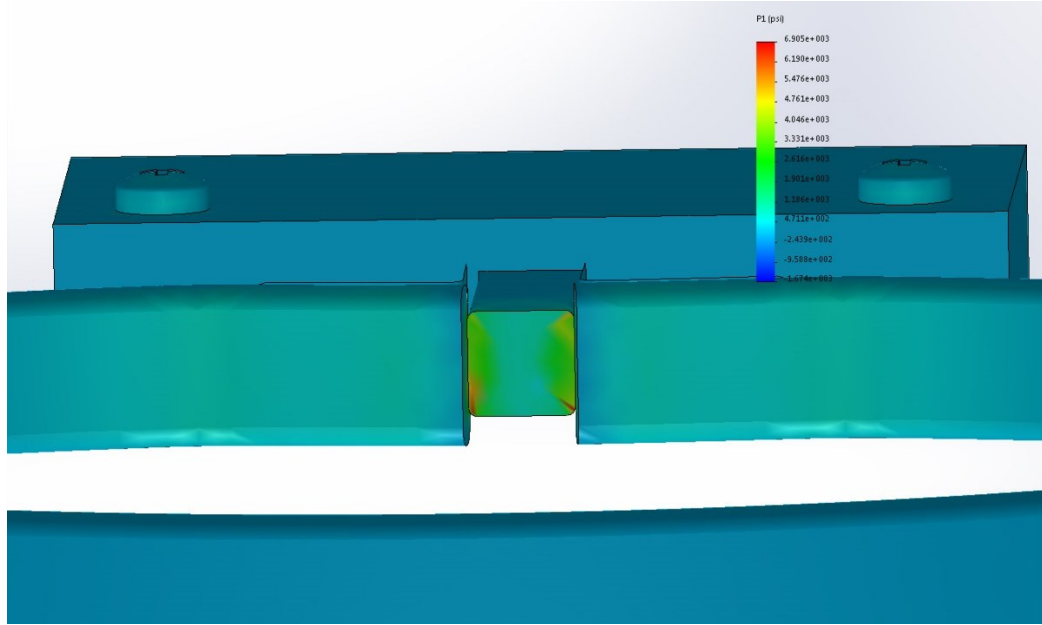


Figure 4.5: Stress distribution in the ring joint resulting from a 2lb horizontal load.

Similar finite element analysis was conducted for the other two loading conditions. When the ring is subjected to a 2lb vertical load directed upwards, it results in a maximum compressive stress of 1015psi and a maximum tensile stress of 8761psi. The areas of highest tensile stress in this scenario are located on the bottom corner fibers of the carbon fiber rod. When the ring is subjected to a 2in-lb counterclockwise torque applied about the axis parallel to the carbon fiber rod, the result is a maximum compressive stress of 106psi and a maximum tensile stress of 618psi.

After testing the three key potential loading conditions that the ring mounting system could experience in operation, it was determined that the system would be able to withstand the loads and function properly. The highest stress observed in the three tested models was a tensile stress of 8761psi, which is well below the yield strength of any material used in the ring mount. These results validate the strength and safety of the ring mounting system.

4.4 Vibration Isolation Camera Mount

Figures 4.6 and 4.7 show theoretical CAD and actual manufactured versions of the vibration isolation multispectral camera mount.

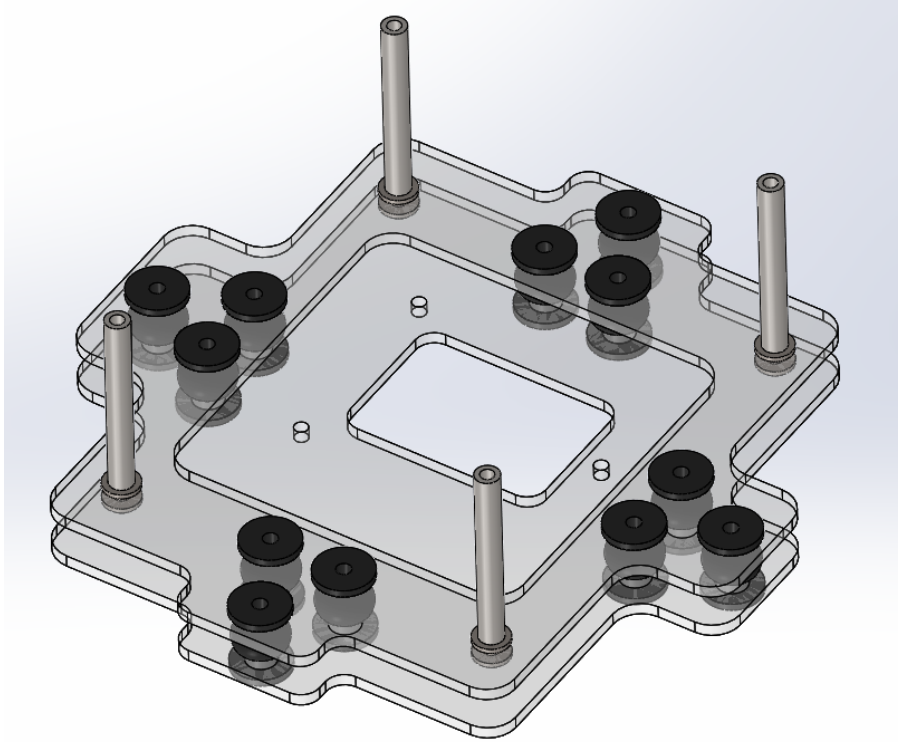


Figure 4.6: Vibration isolation camera mount CAD rendering

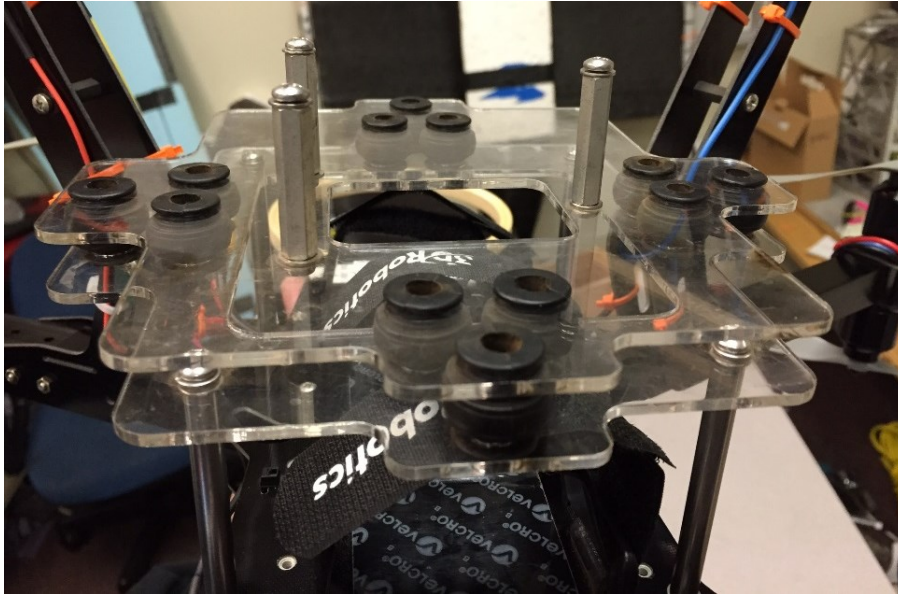


Figure 4.7: Vibration isolation camera mount.

The camera mount consists of two laser cut acrylic plates held together by 12 small vibration damping pads, which have been repurposed from an industry grade gimbal. The entire mount is attached to the octocopter frame using four 2.5" long aluminum

standoffs and 6-32 machine screws. Including the standoffs, the vibration isolation camera mount weighs 195g.

During testing, the multispectral camera attached to this vibration isolation mount was able to provide suitable photographs for post processing and therefore this component is viewed as successful. However, performance of this vibration isolation mount compared to industry counterparts or the lack of a vibration isolation device altogether has not been characterized. Future work is required in this area to provide better insight into whether or not this design can be improved and the degree to which it is actually needed.

4.5 Charging Feet

The charging foot concept is shown in Figure 4.8.

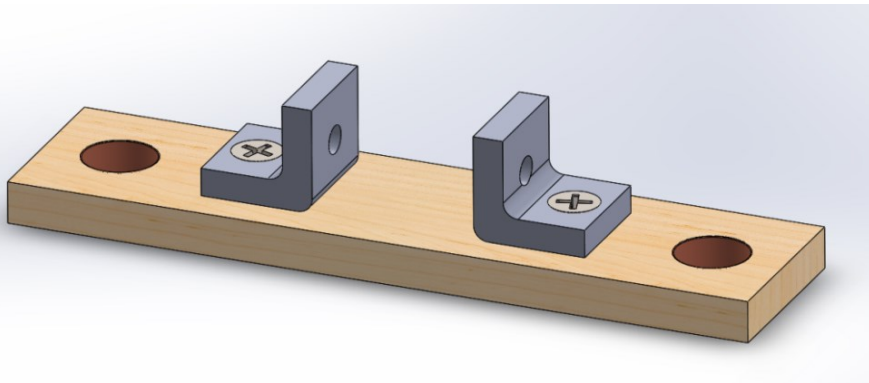


Figure 4.8: Charging foot CAD rendering.

One charging foot is connected to the bottom of each of the four legs of the octocopter. Each foot is made from a piece of 5" x 1.25" x 0.3" poplar which has two copper plugs press fit and then glued into it. These copper plugs have a machined flat bottom so that they rest flat against the contact plates on the docking station. Additionally, a small lip on each plug, shown in Figure 4.9, allows it to sit just below the bottom of the wood piece so that just the plug is in contact with the station, meaning that the full weight of the octocopter rests on these connections points to help ensure a solid electrical connection.

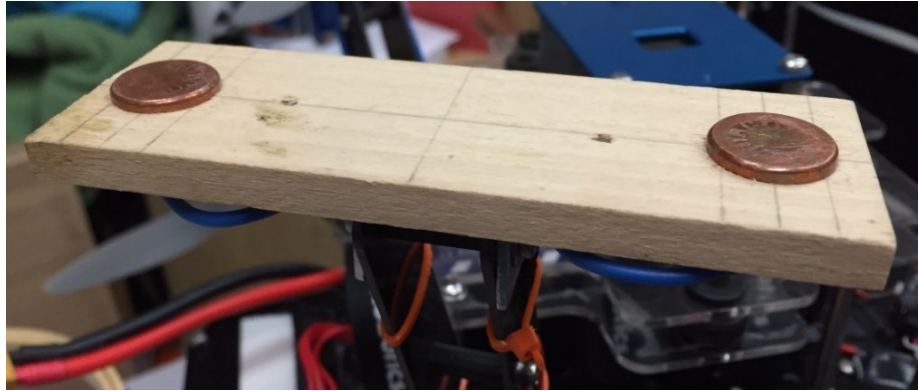


Figure 4.9: Charging foot bottom detail

While the main purpose of the feet is to enable charging, an added benefit is that they provide added stability for the octocopter as it lands. Each foot is connected to one of the octocopter's legs via right angle brackets and M3 machine screws, which is shown in Figure 4.10.

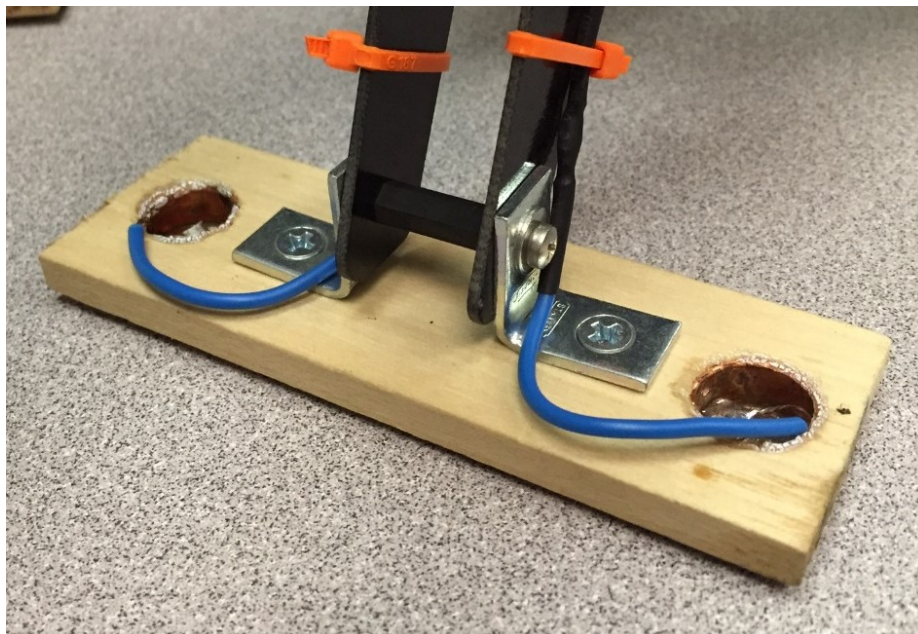


Figure 4.10: Charging foot detail

20-gauge wire is soldered into each copper plug and then routed up the octocopter legs to the flight battery's balancing plug. In this way, the feet act as an extension of the lithium polymer battery's balancing node that the off-the-shelf charger housed in the docking station is then able to charge through. Figure 4.11 shows a wiring schematic depicting how the contact feet are connected to the octocopter flight battery.

With two copper plugs per foot and four feet total, the octocopter has a total of eight connection points. Currently, only five of these points are being used. The remaining three could be used in the future to support batteries with more cells or faster charging rates.

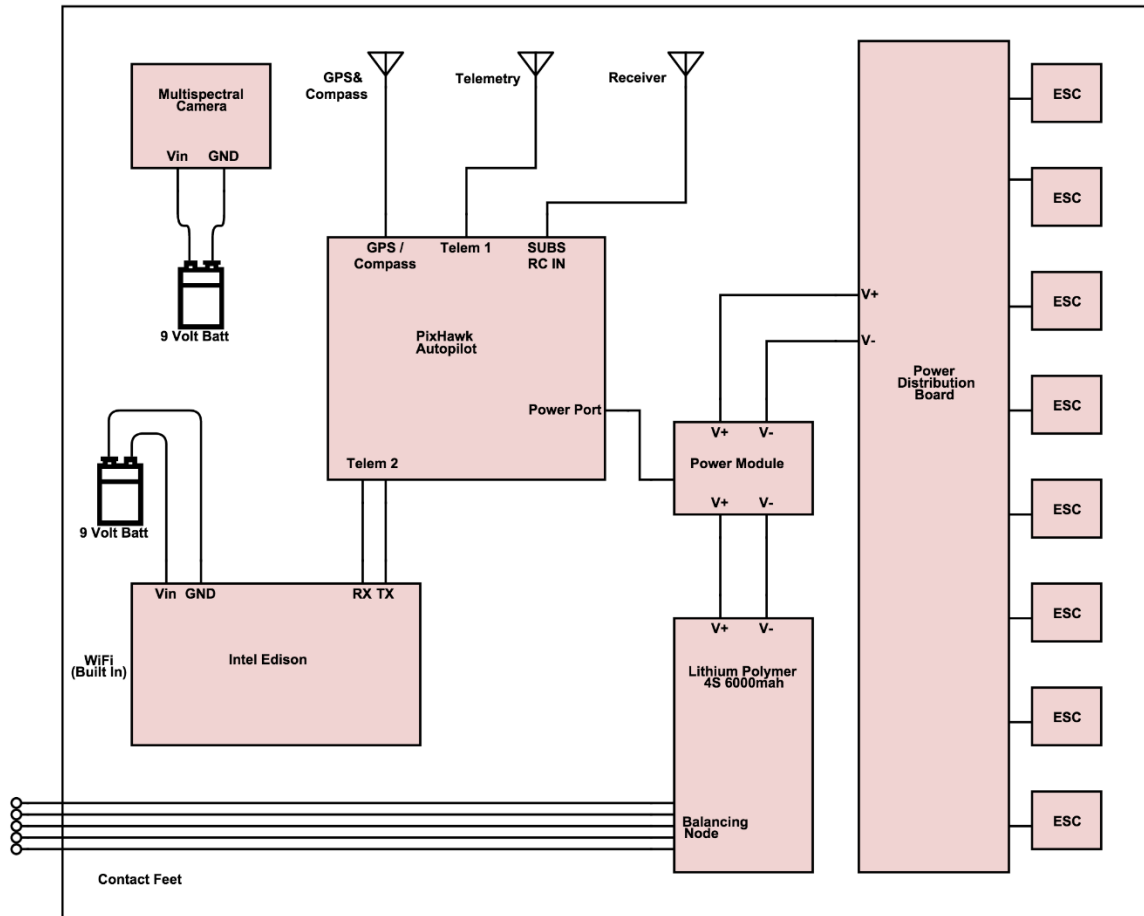


Figure 4.11: Octocopter circuit diagram

Note that Figure 4.11 shows that the multispectral camera and Intel Edison are powered independently from 9V batteries. In the future, these components should ideally be powered from the octocopter's flight battery. Such a system could be implemented through a buck converter connected to the flight battery and that is in parallel with the rest of the octocopter's electronics. In this way, the buck converter would act as a voltage regulator as well as a voltage step-down or step up. However, the buck converter would need to be inserted after the power module, as this component sends important battery voltage remaining and current draw information to the Pixhawk

autopilot. If the buck converter were connected before the power module, the Pixhawk autopilot may underestimate the battery life remaining and cause the vehicle to enter a failsafe state.

4.6 Landing Algorithm Hardware

In addition to the ring brackets required for the mechanical docking system, hardware elements are also required for the vision landing system. These components are the Mobius vision camera, shown in Figure 4.12, and an Intel Edison microprocessor, shown in Figure 4.13. The Mobius camera is a compact video camera that is mounted to the bottom of the octocopter using Velcro. This camera takes pictures at a regular interval and sends them to the Intel Edison, which then runs software to identify the docking station and maneuver the octocopter towards it. The Intel Edison rests on the top of the octocopter and is loosely held in place with zip ties. Detail on these two components and how they specifically work to control the octocopter is discussed in Chapter 5.

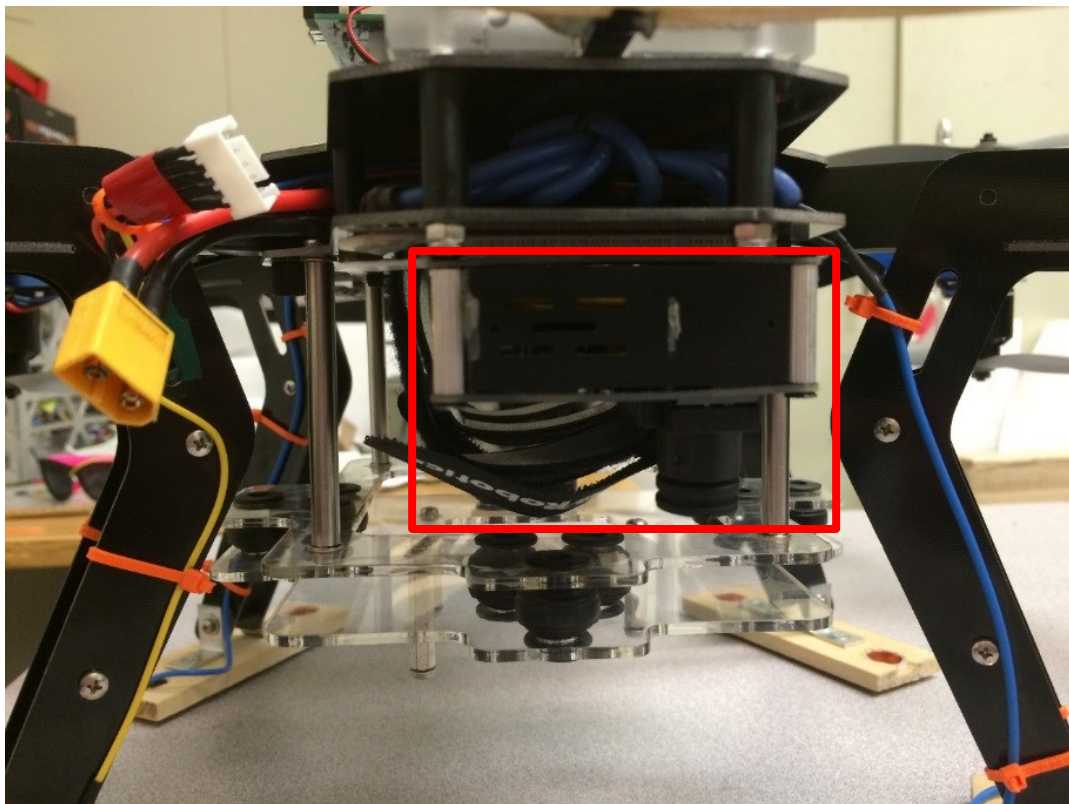


Figure 4.12: Detail of Mobius vision camera, boxed in red, mounted to the octocopter.

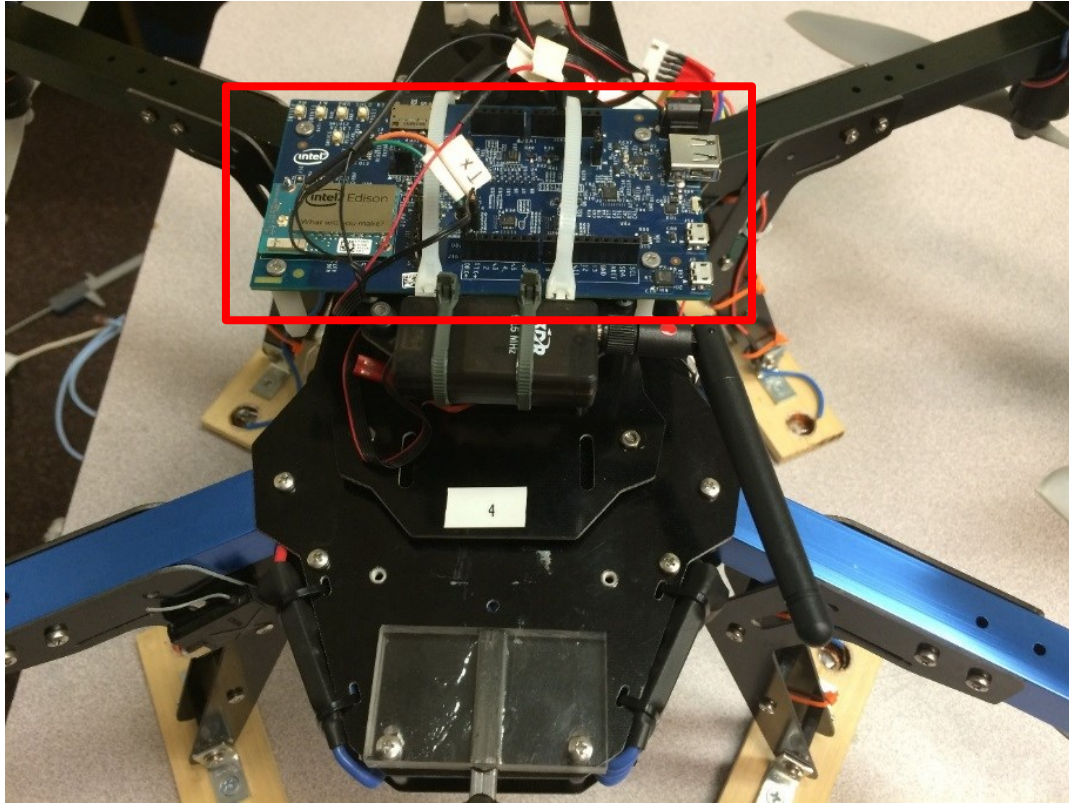


Figure 4.13: Detail of Intel Edison, boxed in red, mounted to the octocopter.

4.7 Summary

To allow for UAVino’s autonomous mapping operation, certain modifications had to be made to the octocopter vehicle. In general, these additions include a set of guidance rings that mate with the cone system on the docking station, a mounting plate with vibration isolation for the multispectral imaging camera, electrical contact feet that enable battery recharging, and a vision camera and computing platform that handle automated landing capability. Since multirotor drones have a relatively short flight time, it was crucial to consider the weight of each component during design and the impact it would have on the system’s flight time and, by extension, its coverage capability.

Testing these added components has proved that they function properly. Finite element analysis was conducted on the ring mounting system for a variety of loading conditions, and results show that the system can withstand its intended operation. Additionally, recharging testing has verified that octocopter’s charging feet can establish and

maintain an electrical connection between the octocopter's battery and the docking station. Lastly, the vibration isolation camera mount experienced field tests and was able to yield clear and useful multispectral imaging data, therefore lending confidence in its performance. Overall, these octocopter additions allow for the successful implementation of UAVino's unique method for the autonomous mapping capability.

CHAPTER 5

Landing Algorithm

5.1 Overview

A critical part of UAVino’s autonomous operation is the ability of the octocopter to land itself precisely on the docking station, as this task enables the vehicle to recharge and ultimately extend mission duration and range. Although GPS helps the octocopter return to the general vicinity of the docking station, it does not provide the accuracy needed to align the ring and cone guidance system. In order to achieve precision landing, UAVino uses a vision-guided landing system that involves a standard video camera and co-processing board, both of which are mounted to the octocopter. The interaction between these components and the octocopter’s built-in control system is shown in Figure 5.1.

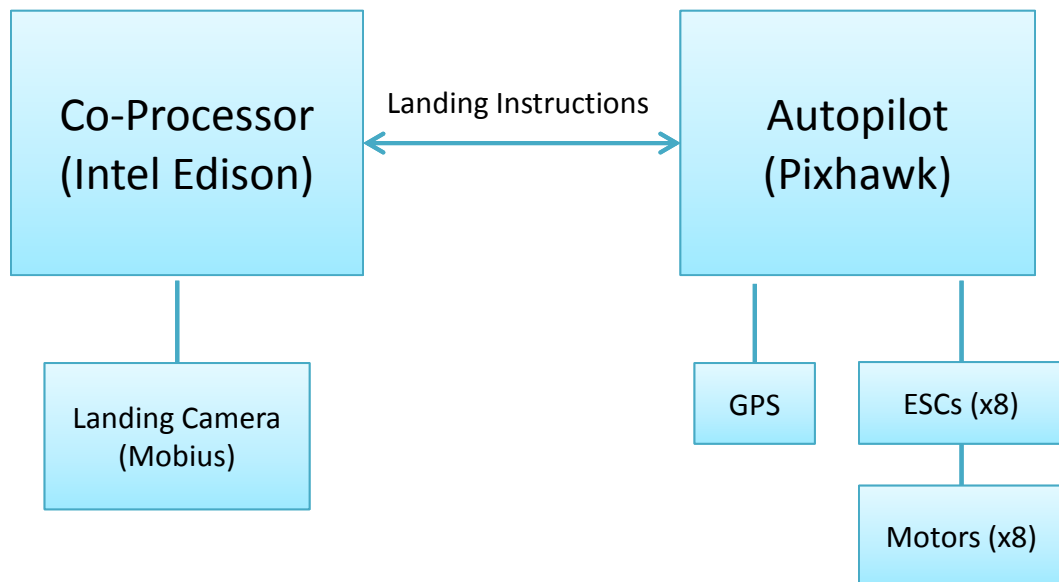


Figure 5.1: Vision guided landing system components.

When operating, the vision camera sends pictures to the processing board, which identifies the docking station via its distinct painted surface. Using the picture, an algorithm calculates the movement required to center the vehicle over the station and

then sends commands to the vehicle's autopilot board in order to make those movements. This algorithm is repeated until the vehicle is hovering directly above the docking station, at which point it begins descending. The two steps of centering and descending are periodically repeated so that the vehicle can adjust its position and ensure it remains centered. Ultimately, the vehicle engages the docking rings and cones that allow precise landing on the station's charging terminals.

5.2 Requirements

In order for the octocopter to successfully dock and recharge, the landing algorithm must be able to orient the octocopter within 2.5in of center so that the vehicle's guidance rings engage the station's cones. Achieving this level of accuracy hinges upon the landing algorithm consistently detecting the docking station in various orientations and lighting conditions so that it can continually send movement commands to correctly position the vehicle.

To facilitate consistent recognition, the docking station is painted with recognizable red and black concentric squares. This pattern is shown in Shown in Figure 5.2 and was selected because it minimizes reflectance, yet remains visually distinct.

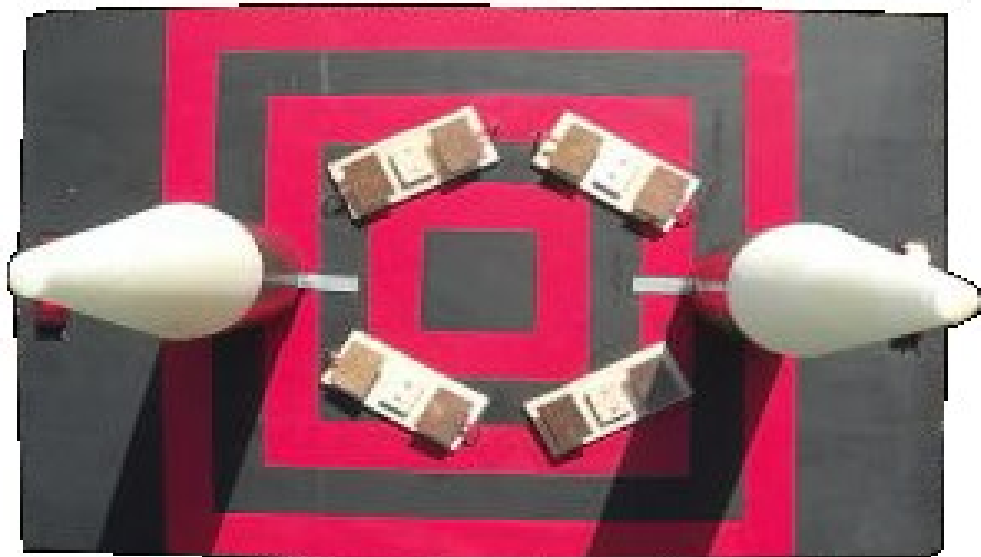


Figure 5.2: Docking station alignment pattern

Lastly, the entire vision algorithm relies on the autopilot's ability to position itself correctly and precisely given the direction commands it receives from the landing algorithm software.

5.3 Hardware

5.3.1 Mobius Vision Camera

The need for a lightweight vision camera that produces high quality images for image recognition is fulfilled by the Mobius Action Camera 1080P HD Mini Sports Cam, which is shown in Figure 5.3.



Figure 5.3: Mobius Action Cam vision camera

Figure 5.3 shows the Mobius converted via a GoPro Form Factor kit, which allows for easier mounting and also reduces the component's weight to 30g. The Mobius was selected because it offers a wide range of features, including still image capture, time-lapse image capture, video capture, and video streaming. Of these modes, the Mobius' streaming ability is particularly important to UAVino, as it allows the co-processing board to pull still frames for analysis on demand.

5.3.2 Intel Edison

The landing algorithm requires processing speed and power in order to analyze images and compute new flight commands easily. In order to deliver these requirements, the Intel Edison, shown in Figure 5.4, was selected as the co-processing board.

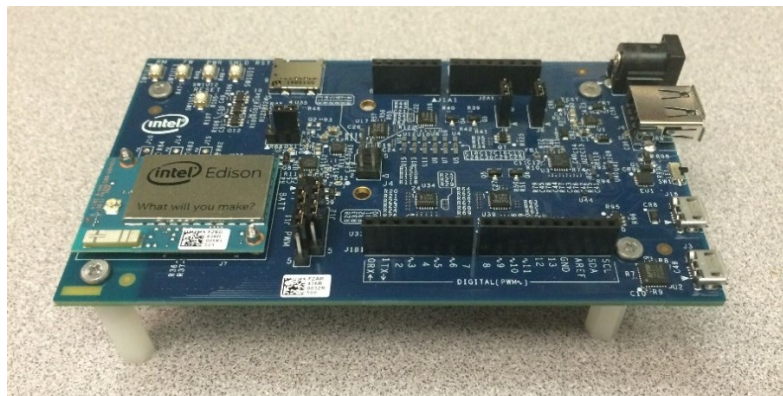


Figure 5.4: Intel Edison computing platform

The Edison was chosen over similar competitors, such as the Raspberry Pi, due to its processing power and low energy consumption. Table 5.1 compares the Intel Edison to competing Raspberry Pi B+ and Beaglebone Black microcontrollers.

Table 5.1: Comparison between the Intel Edison, Raspberry Pi B+, and Beaglebone Black

Microprocessor	Pros	Cons
Intel Edison	<ul style="list-style-type: none"> • Dual-core processor; • Integrated Wi-Fi, Bluetooth LE • Support for Yocto Linux Arduino and Python • Lower Power consumption (3.3V - 4.5V @ <1W) 	<ul style="list-style-type: none"> • 1 USB port • Expensive • No video output (HDMI, Direct LCD, Composite) • I/O connectors require extra boards
Raspberry Pi B+	<ul style="list-style-type: none"> • Broadcom VideoCore IV GPU • 4 USB ports • Video output (HDMI, Direct LCD, Composite) • Inexpensive 	<ul style="list-style-type: none"> • No Integrated Wi-Fi or Bluetooth. • Higher power consumption (5V @ 600 mA) • Less processing power
Beaglebone Black	<ul style="list-style-type: none"> • ARM® Cortex-A8 Processor • 4GB Onboard Flash • USB, Ethernet, micro HDMI ports 	<ul style="list-style-type: none"> • 1 USB port • Expensive • No Integrated Wi-Fi or Bluetooth.

The Intel Edison is offered as computing platform for internet-of-things products. It is designed to be fast, powerful, efficient, and easily connectable to other devices via WiFi

and Bluetooth. This processor provides a fast and powerful platform for processing images and translating the results into useful data that is sent to the octocopter's autopilot. The Edison also gives the ability to establish a wireless hotspot, which is useful for communicating with the docking station to initiate or stop charging as well as for operators to remotely login to run or monitor scripts.

The need for low power consumption is critical in a system where power is a limited resource and is crucial to system success. The less energy the octocopter uses powering the co-processor, the more flight time the octocopter has. In addition to its low power consumption, the Intel Edison's dual-core CPU easily handles the tasks of image recognition and flight command generation at a fast enough speed to allow for efficient octocopter flight.

5.3.3 Pixhawk Autopilot Board

UAVino uses the Pixhawk autopilot board, shown in Figure 5.5, to control the octocopter.



Figure 5.5: Pixhawk autopilot board. Photo courtesy of 3drobotics.com.

The Pixhawk is built by 3D Robotics and is included as the default autopilot for their X8 octocopter, which is the platform upon which UAVino is based. The autopilot runs the NuttX Real Time operating system over a PX4 driver layer and is built with integrated multithreading. It also provides numerous ports for serial communication and PWM control, which allows it to power the octocopter and communicate with motors, sensors and ground control stations.

5.4 Landing Algorithm Stages

5.4.1 Overview

The vision-guided algorithm that lands the octocopter is composed of three stages: locating the docking station, determining the steps needed to move to the octocopter towards the docking station, and sending commands to execute these movements to the vehicle's autopilot. The overall landing algorithm flow is shown in Figure 5.6.

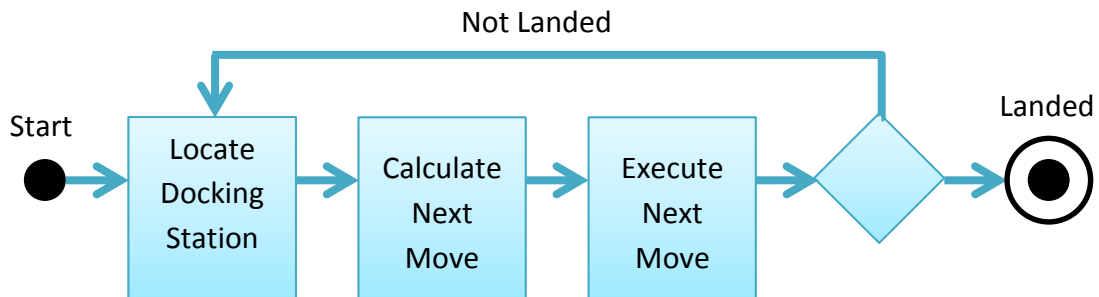


Figure 5.6: Landing algorithm logic flow

In general, the docking station is first located using images from the Mobius camera processed using a Haar Cascade classifier. Then, the distance to the docking station is calculated and commands for the octocopter are created. Finally, these commands are sent to the octocopter's autopilot and executed. As shown in Figure 5.7, the execution of these stages is split between the autopilot board, which directly controls the octocopter, and the Intel Edison co-processor, which executes the computations for target location and movement calculation.

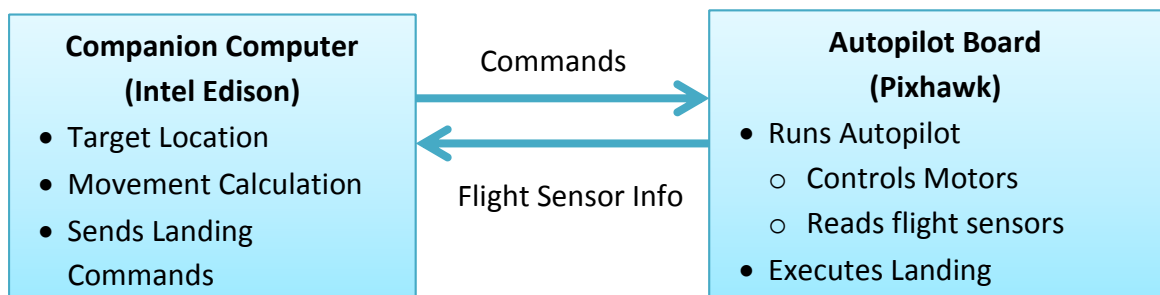


Figure 5.7: Separation of onboard processing and flight control

5.4.2 Locate Docking Station

In the first phase of the landing algorithm, the location of and distance to the docking station relative to the drone is determined using computer vision. The need for computer vision became apparent after exploring other positioning devices. GPS, which is included by default with the Pixhawk autopilot, is an excellent asset for locating the general position of the docking station, but is only accurate to within approximately 1.5m. Differential GPS (DGPS) could be used, which is accurate to about 4in, but requires the use of a reference station and differential GPS locators. Although DGPS can deliver close to UAVino’s accuracy requirement of 2.5in, it can only be used for latitude and longitude determination. Therefore, while DGPS could be used to accurately center the octocopter over the docking station, auxiliary systems would be required to orient the drone’s yaw axis and calculate its altitude, resulting in added system complexity. Figure 5.8 shows the maximum error of UAVino’s computer vision system

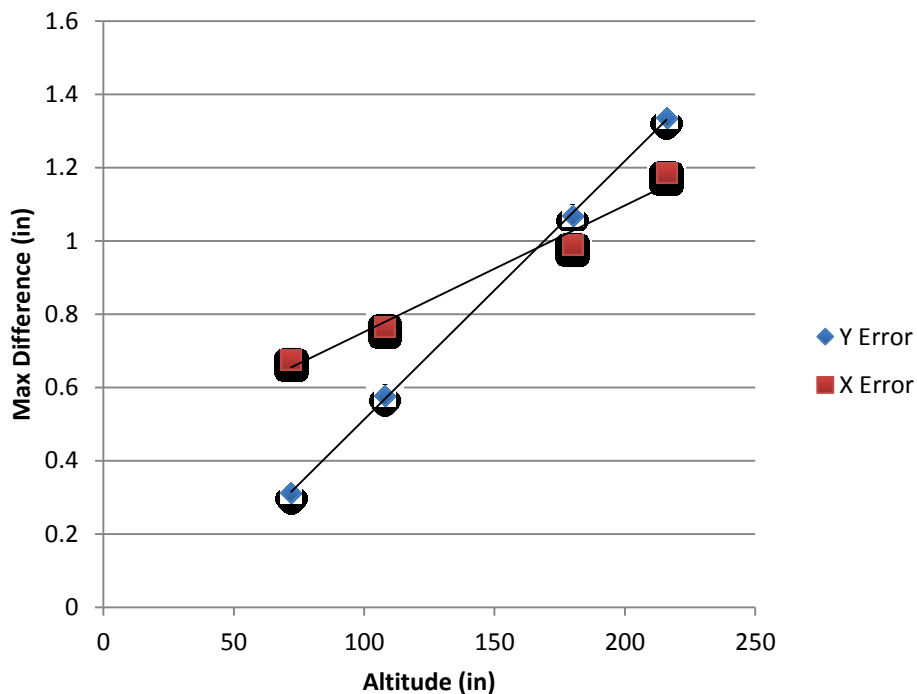


Figure 5.8: Error between estimated lateral distance using UAVino’s computer vision algorithm and the true lateral distance versus altitude. In a lab setting, the error at each altitude was calculated by taking the maximum difference between the estimated and true distance over a series of five trials at each altitude.

Computer Vision offers the best solution for vehicle orientation because it can adapt to a dynamically changing environment and still provide the accuracy required to dock. Via, Figure 5.8 it is clear that the maximum error of UAVino's computer vision system is well within the 2.5in margin of error. It also shows that as the drone descends towards the docking station, the error between measured distance and actual distance decreases, thus improving the likelihood of a successful dock.

Once the Intel Edison obtains images from the Mobius camera, the microprocessor uses a Haar Cascade classifier to locate the docking station within the image, which is shown in Figure 5.9.

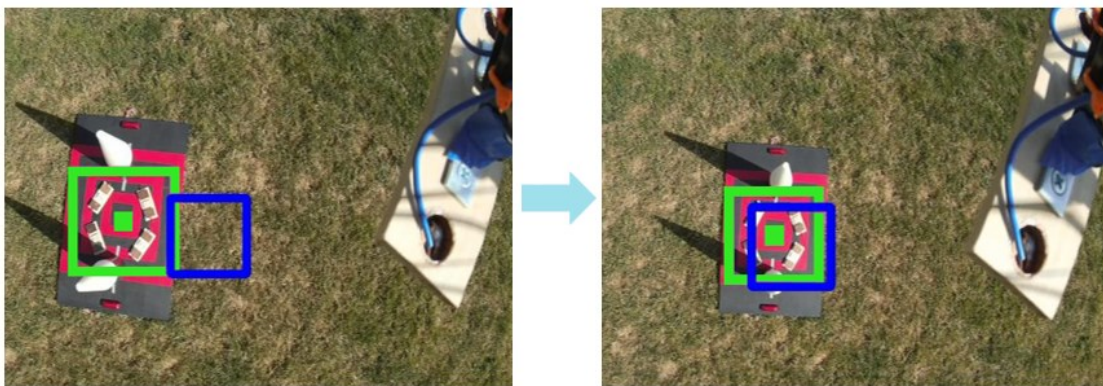


Figure 5.9: Example of docking station being located (green box) and the octocopter orientating itself over the located target.

The Haar Cascade classifier was trained and run using a computer vision library called OpenCV, which was selected due to its reliability, open source license, cross platform support, and excellent supporting documentation. Haar Cascade classifiers are created using a supervised learning approach, meaning that they are trained on a series of positive images containing the target object and negative images without the target object. A large and diverse set of positive and negative images, particularly with regards to lighting conditions and image orientation, is required to train an accurate classifier that works well in various environments.

Red and black were chosen as the colors for the docking station due to their distinctness, their lack of resemblance to nature, and their ability to flood the color

spectrum, all of which made classifier training easier. During testing, these colors yielded promising results in a lab environment. However, field testing revealed that the classifier recognized a large number of false positives. These results are likely due to the fact that black closely resembles shadows, which were present in nearly all images.

UAVino's landing algorithm was originally developed in C++ for performance reasons, but was later ported to Python since DroneKit, the API used to control the octocopter, is written for Python.

5.4.3 Movement Calculation

After locating the docking station in an image, the Intel Edison calculates the drone movement required to center the drone over the station. There are three different kinds of movement the octocopter must make in order to properly orient itself: lateral movement, yaw rotation, and vertical movement, which are illustrated in Figure 5.10.

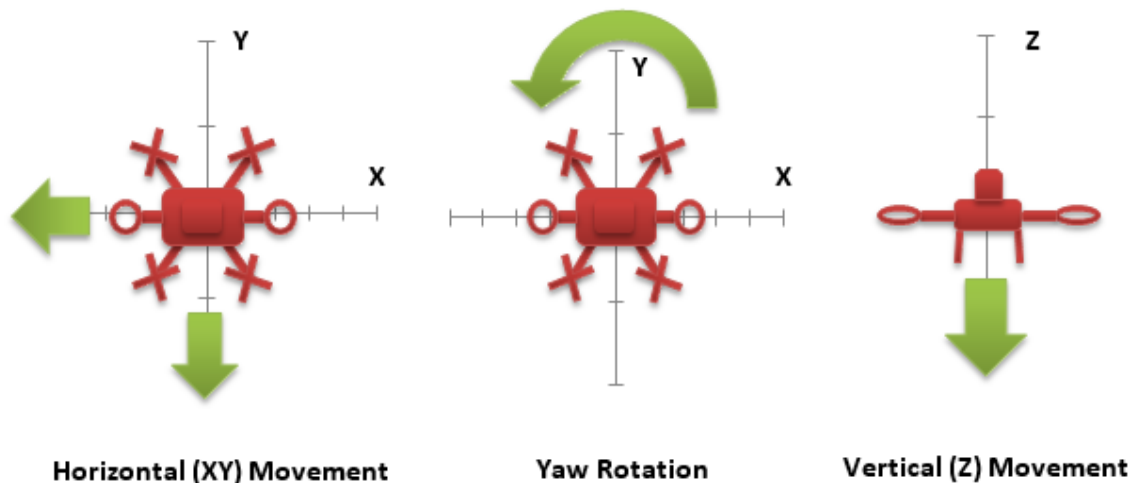


Figure 5.10: The three movement types generated by UAVino's landing algorithm.

First, horizontal movement is used to center the octocopter over the docking station. Then, yaw rotation is used to align the rings on the octocopter with the cones on the docking station. Finally, vertical movement is used to descend the octocopter and mate the guidance rings with the station's cones. UAVino is programmed to execute only one kind of movement in each iteration of the landing algorithm to make the system simple to

debug, as it is easier for the operator to visualize the thought process of the octocopter. The process by which the octocopter chooses which kind of movement to make is described in Table 5.2 and further illustrated in Figure 5.11.

Table 5.2: Movement type selection process

If the vehicle is...	Then execute...
Not centered over the docking station	Horizontal move
Centered over the docking station, the hoops are not aligned with the station's cones	Yaw rotation
Centered over the station and properly aligned with the station's cones	Vertical move

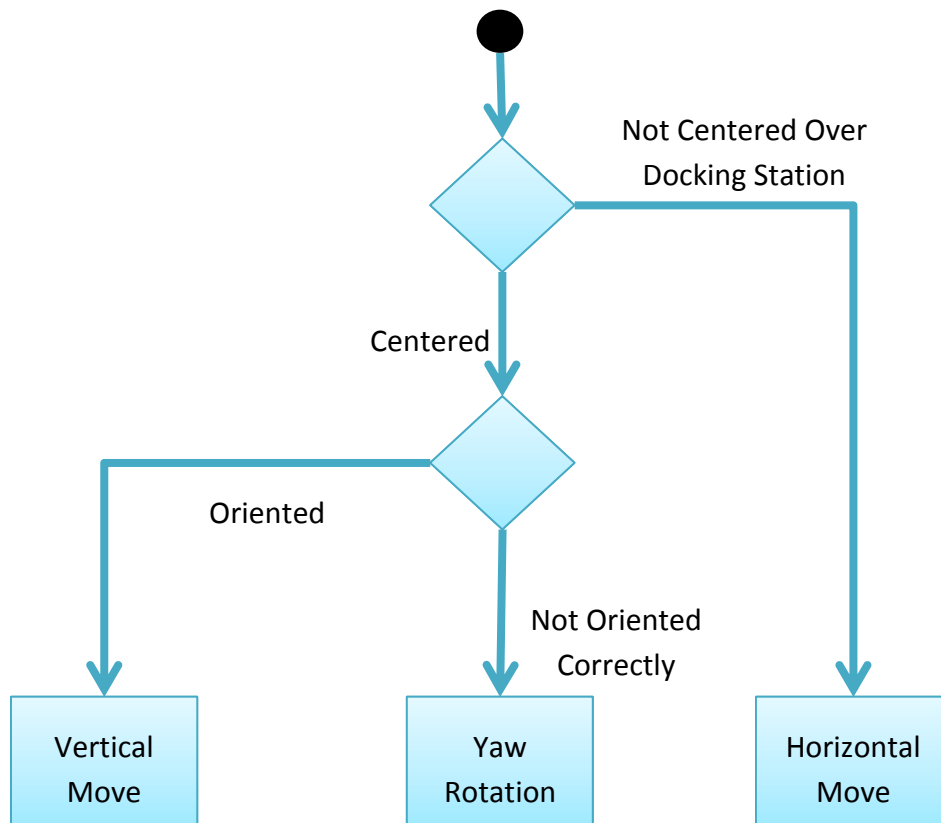


Figure 5.11: Movement type selection process

Horizontal movement is calculated by finding the distance in pixels from the center of the detected docking station to the center of the landing camera, which is shown in Figure 5.12.

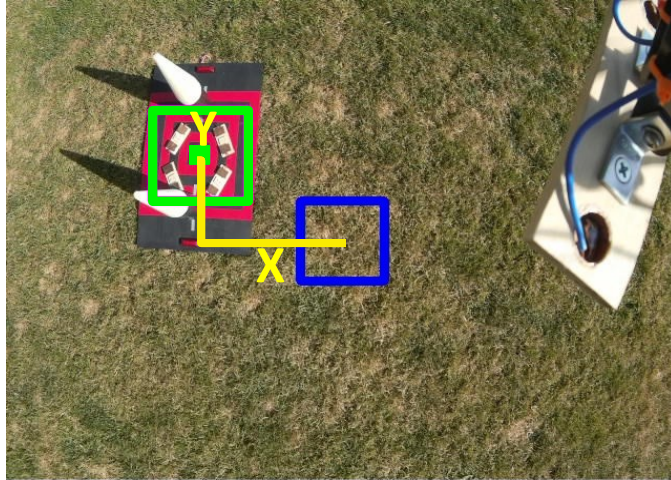


Figure 5.12: Calculating distance in pixels from Mobius camera image

The pixel distance is converted to ground distance by calculating the Ground Sample Distance (GSD), which is the ground distance represented by a pixel. The GSD is linear with respect to the camera's distance from the ground and is calculated by

$$GSD = cr$$

where GSD is the ground sample distance, c is a constant, and r is the relative altitude. The constant can theoretically be calculated from the camera's sensor size and focal length, but experimental methods were used to determine c in UAVino's landing algorithm because the sensor size and focal length of the Mobius camera are not readily available. Appendix I discusses the experimental process by which this constant was determined.

Because the GSD is calculated using altitude, any error in determining relative altitude decreases landing accuracy. However UAVino's landing algorithm constant is roughly 0.0025, so an altitude error of 30 feet is required to introduce a lateral position error on the order of 1in. Therefore, effects of altitude error are minimal. UAVino's onboard barometer is used to determine relative altitude since it is accurate enough for this application and can be used without adding complexity to the system. Should more precision be needed in the future, a radar or sonar altimeter could be used to more accurately determine altitude.

It is also possible for the landing algorithm to determine drone altitude using the GSD constant and the known distance between two detected objects in the analyzed image. The relevant equation is

$$r = \frac{\left(\frac{d}{p}\right)}{c}$$

where r is the relative altitude, d is the known distance, p is the known distance represented in pixels, and c is the constant. Determining drone altitude using this equation and the known width of the docking station proved less accurate than the onboard barometer because the detected width of the station fluctuates with factors such as camera angle or lens distortion.

The distances and directions of movement calculated from the analyzed image are represented in the camera's frame of reference. However, the drone's Pixhawk autopilot only supports navigation in the North East Down (NED) frame of reference. The movements are converted from the octocopter's body frame to the NED frame using the following transformation, where yaw is ϕ roll is ψ , and pitch is θ .

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} \cos(\phi) \cos(\theta) & \cos(\phi) \sin(\psi) \sin(\theta) - \sin(\phi) \cos(\psi) & \sin(\phi) \sin(\psi) + \cos(\phi) \cos(\psi) \sin(\theta) \\ \sin(\phi) \cos(\theta) & \cos(\phi) \cos(\psi) + \sin(\phi) \sin(\psi) \sin(\theta) & \sin(\phi) \cos(\psi) \cos(\theta) - \cos(\phi) \sin(\psi) \\ -\sin(\theta) & \sin(\psi) \cos(\theta) & \cos(\psi) \cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

The movements are passed to the flight command execution stage after they have been transformed into the NED frame. Note that this transformation requires the Z commands to be given in the camera's frame of reference. If the Z commands are given in the Down axis, the transformation instead becomes

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} \cos(\phi) \cos(\theta) & \cos(\phi) \sin(\psi) \sin(\theta) - \sin(\phi) \cos(\psi) & 0 \\ \sin(\phi) \cos(\theta) & \cos(\phi) \cos(\psi) + \sin(\phi) \sin(\psi) \sin(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Yaw rotation has not yet been implemented in UAVino, but groundwork has been laid for this feature to be added using the center of the detected docking station and another fiducial marker.

The vertical movement algorithm is designed to descend half of the drone's altitude with each iteration. In order for this method to become fully operational, it needs to be supplemented with a stage to descend a fixed distance once the octocopter is close enough to the docking station that the ring and cone mechanism is engaged.

5.4.4 Flight Command Execution

After the Intel Edison correctly calculates and translates the required movement vector of the drone, a command corresponding to this vector is sent to the Pixhawk autopilot. Ultimately, communicating between the Edison and Pixhawk requires encoding, transmitting, and decoding a message. However, this detail is abstracted away by the Pixhawk's ArduPilot software and DroneKit, an API that supports both sending commands to and receiving flight information from an autopilot board running ArduPilot software. Thus, the only task the Intel Edison must focus on is actually generating the movement command to send to the autopilot board.

UAVino relies on two main commands for vehicle movement: changing the octocopter's lateral position and changing the octocopter's yaw orientation. DroneKit and ArduPilot provide a function that instructs the vehicle to change its position; however, this function is only as precise as the autopilot's return-to-home method, which is approximately 1m. Because UAVino requires accurate movement to within 2.5in, the algorithm instead uses a function provided by DroneKit and ArduPilot to instruct the vehicle to change its velocity.

There are a two main approaches to accurately controlling a vehicle by velocity. One method is a camera-polling approach, which sets a vehicle velocity and then continuously analyzes images from a camera to determine whether the vehicle needs to continue moving, readjust course, or stop. The main disadvantage of this method is the

relatively long time it takes to obtain and process an image, as any delays in the algorithm reduce accuracy. However, one key advantage of camera-polling is that the stopping position is determined dynamically. Thus, if a gust of wind were to blow the vehicle off course, the algorithm would naturally adjust itself and continue towards the correct stopping position. Another advantage of a camera-polling approach is that only the direction of travel is required; no distance measurement is needed.

A second approach to controlling a vehicle by its velocity is a time-polling approach, which sets a specified velocity, uses time to estimate the distance travelled, and stops the vehicle once the desired distance has been reached. The main advantage of this approach is that it can be executed more quickly than a camera-polling approach, but it comes at the cost of the stopping position being determined statically at the beginning of the movement. Thus, the vehicle is not be able to effectively respond to a disturbance during movement. Another disadvantage of the time-polling method is that it requires both a velocity and accurate distance to execute. Table 5.3 shows a comparison between the camera-polling and time-polling methods.

Table 5.3: Velocity control methods

	Camera-Polling Method	Time-Polling Method
Pros	<ul style="list-style-type: none"> • Dynamically determines course • Only requires a direction 	<ul style="list-style-type: none"> • Quick response
Cons	<ul style="list-style-type: none"> • Slow response 	<ul style="list-style-type: none"> • Statically determines course • Requires both a direction and a distance

UAVino’s landing algorithm combines the time-polling method with a repeated approximation approach to provide both a quick movement calculation and mild course correction.

5.5 Summary

In order for the octocopter to recharge, it must be able to precisely land on and establish an electrical connection with the docking station. GPS cannot deliver the level

of precision required, so UAVino uses a vision landing system for guidance. This algorithm relies on visually locating the docking station using an onboard camera and a Haar Cascade classifier. Once the docking station has been located, the octocopter navigates to the docking station, centers over it, and then descends onto the landing platform. Although there has been some success with recognizing the docking station and flying towards it, the system is currently unable to achieve autonomous landing. These difficulties are linked to the Haar Cascade classifier being inadequately trained, leading to either a low detection rate or a high false positive detection rate. Future work needs to be done to either better train an accurate Haar Cascade classifier or to change the overall method of locating the docking station, such as using infrared beacons. Additionally, a proportional-integral-derivative control system needs to be implemented to more precisely execute the landing commands sent to the octocopter. Overall, the landing algorithm has a modular framework, so these future implementations can easily be integrated into the existing platform.

CHAPTER 6

Data Processing

6.1 Overview

This year, a straightforward processing method has been developed to convert multispectral images collected by the octocopter into crop health information that is useful to real-world customers. In general, this data conversion is accomplished by running the multispectral images through various software packages that return vegetation indices, which are numerical values that provide an indication of plant health. Ultimately, the goal is for this data to augment a customer's existing knowledge to help determine overall crop health, as well as locate areas of stress or concern.

The first step in post processing the multispectral data is to manually review the collected images and remove ones with defects such as blurriness or overexposure. Additionally, the general set of pictures is examined to ensure that it contains sufficient coverage of the inspected agricultural field. The images are then used to create a photo-mosaic map of the agricultural field, which is processed to depict a vegetation index. The final data product is an aerial map of the inspected fields showing color-coordinated crop health information that is accessible and easy to interpret.

This year, UAVino focused on providing the foundational methods for post processing and succeeded in creating a proof of concept. However, the most powerful results from multispectral data come from long term monitoring, which has yet to be implemented. This type of monitoring creates a well-established standard against which new data is compared, thus making it easier to determine if crop behavior is unusual or indicative of water stress or disease. It is through monitoring vineyards over a longer period of time that UAVino will be able to warn customers about changes in plant health before crops show physical signs of disease.

6.2 Requirements

6.2.1 Customer Requirements

The most important requirement for image post processing is that the final data product be presented in an easy to interpret and accurate form for the customer. This requirement includes the following elements:

- The customer should not have to perform any additional processing on the data after it is received in order to determine health information.
- The collected data needs to be presented with an easily accessible overview so that the customer is not overwhelmed by the amount of information received.
- A method of interpreting the data needs to be presented along with the data itself to ensure that the customer understands the implications of the data, particularly what it does and does not depict and to what accuracy it can be trusted.

6.2.2 Image Resolution Requirements

Although not an explicit requirement to obtain vegetation index data, obtaining aerial pictures with a 1.0cm per pixel resolution is UAVino's image requirement. Achieving this level of precision is one key reason why a multirotor flight vehicle was chosen, as such platforms can fly lower and slower than fixed wing drones to obtain better resolution data. A 1.0cm per pixel image resolution is at least double the resolution of fixed wing platforms, which is typically around 2cm per pixel, and many times greater than satellite imagery, which is typically around 0.5m per pixel.

6.2.3 Image Overlap Requirements

Agisoft, the orthographic photo-mosaicking software used, requires 80% forward overlap and 60% side overlap between multispectral images in order to successfully create an aerial map of the inspected agricultural fields. At a minimum, 50% forward overlap and 40% side overlap is required to create an aerial map using Adobe Photoshop, although this software does not provide orthographic mapping capability.

6.3 Key Components

6.3.1 Multispectral Camera

Multispectral data is gathered using a Tetracam ADC Micro multispectral camera, which is mounted to the bottom of the octocopter and takes pictures at a regular interval as the vehicle flies over agricultural fields. The camera, shown in Figure 6.1, is ideal for UAVino's application because it is both lightweight and compact.

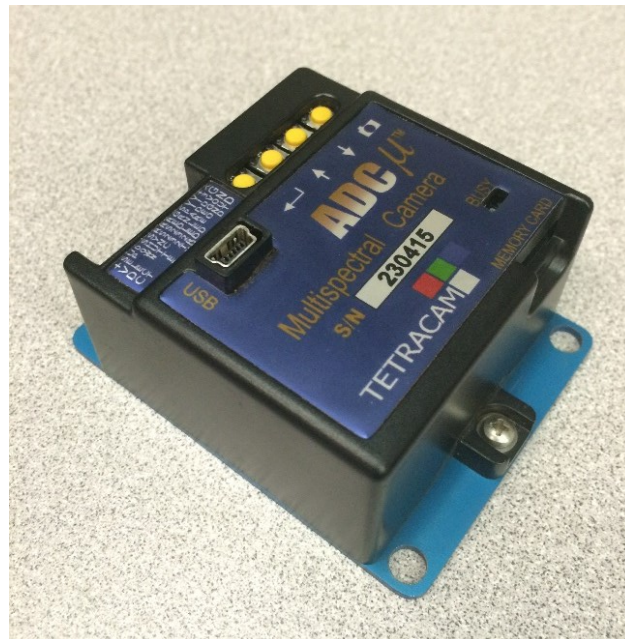


Figure 6.1: Tetracam ADC Micro multispectral imaging camera.

Measuring 2.97" x 2.33" x 1.29" and weighing only 90g, the ADC Micro takes images in the red, green, and near infrared multispectral bands and saves composite false color images in a proprietary format on a removable micro-SD card.

6.3.2 Image Mosaicking Software

In order to provide data in an easy to interpret format, image-mosaicking software is required to convert the multispectral images into an aerial map of the entire agricultural field of interest. This map provides the customer with a single image from which to interpret data, instead of hundreds, and thus allows for quick understanding of the information gathered.

Image-mosaicking is conducted using two software packages: Agisoft and Adobe Photoshop. Agisoft creates a single orthomosaic map from a set of images, meaning that individual images are shifted and scaled by the program so that each pixel in the final output represents the same amount of land area. Achieving the level of precision required to create such a map is difficult, so Photoshop is also used as a more basic image-mosaicking package that simply stitches images together without creating an even perspective. Although Photoshop results are less accurate, it is a faster and easier way of getting data and serves as a point of comparison when orthographic data is able to be obtained through Agisoft.

6.3.3 Image Processing Software

After creating a single aerial map from the gathered image data, it is processed into a vegetation index map using Pixelwrench2, a software package provide by Tetracam. This program also converts Tetracam’s proprietary image format into more cross-compatible versions and provides the ability to control multispectral camera settings through a file stored on the camera’s micro-SD card.

6.4 Post Processing Approach

6.4.1 Flight Parameters and Camera Settings

The quality of multispectral images taken during flight depends upon a number of vehicle parameters, which were optimized by conducting a series of test flights with different settings and comparing the resulting data.

Achieving the necessary image overlap requirements is primarily a function of flight speed, altitude, and image capture rate. However, due to internal data processing, the ADC Micro camera has a maximum capture rate of one photo every three seconds. Therefore, this capture rate is treated as fixed so that image overlap becomes dependent upon vehicle speed and altitude only. Flying at a higher altitude allows for faster flight speed and increased area coverage, but comes at the cost of decreased image resolution. Because one key interest with UAVino is the increased resolution

capabilities of multirotor drones compared to their fixed wing counterparts, image resolution was generally favored over vehicle flight time during testing for optimal flight parameters.

Ultimately, test flights were conducted at an altitude of 20m to provide a final image resolution of approximately 20mm per pixel. At this altitude, the flight speed required to obtain the desired forward overlap is 2m/s or less. A lateral speed of 1.75m/s was used in order to achieve the desired 80% forward coverage without sacrificing the octocopter's ability to map a reasonable amount of land area per flight battery.

One recurring problem encountered when gathering data was blurry pictures, which could still be included in the final image mosaic but caused decreased performance in the mapping software. Typically, blurry photos occurred when the octocopter experienced sudden movements due to turbulence or oversensitive corrective inputs by the flight controller. This behavior was most prevalent on windy days and short of re-tuning the control software, little could be done to prevent such motion. Flying excessively fast could also cause motion blur, but the selected 1.75m/s lateral speed was slow enough that blurriness was not a problem during normal flight on days with relatively calm air. Figure 6.2 shows an example of a blurry multispectral image.



Figure 6.2: Blurry multispectral image

A much more severe problem prevalent throughout early testing was overexposure, which rendered images unusable and created gaps in the final data product. Figure 6.3 shows an example of an overexposed image.

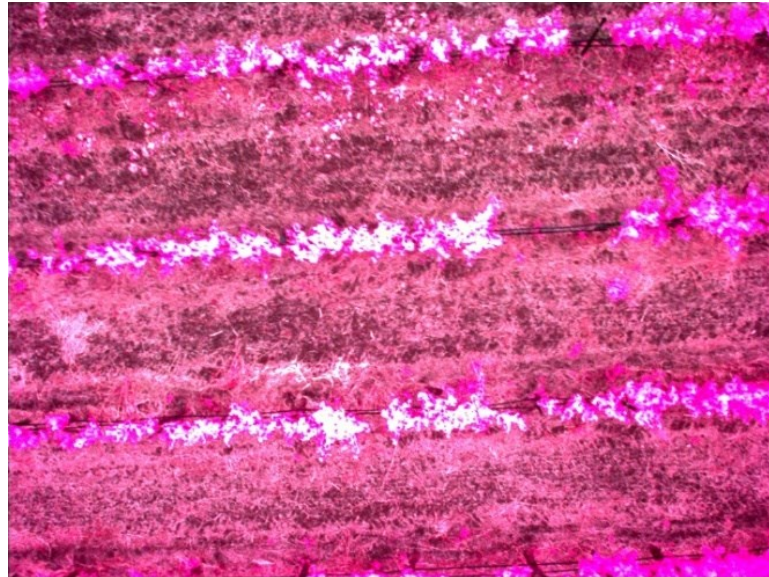


Figure 6.3: Overexposed multispectral image

Overexposure is a photography phenomenon that occurs when too much light is let into a camera's aperture while taking a photo. During initial test flights, the ADC Micro's automatic exposure setting was used, which yielded overexposure rates as high as 40% in some data sets. Initially, overexposure was thought to be caused by excessive sunlight and flight speed, but changing both of these variables had little impact on overexposure rates. Eventually, it was determined that the camera's automatic exposure setting was changing the exposure with each image and would occasionally select an incorrect setting due to lighting conditions, thus yielding overexposed images. This problem was eliminated by selecting a manual exposure setting that remained constant during flight. With some experimentation, a 5ms exposure setting was selected, which yielded less than 4% overexposed images and greatly decreased the severity of data loss.

6.4.2 Image Mosaicking

Unfortunately, successful ortho-mosaicking has not yet been implemented with UAVino due to software difficulties. However, meaningful results have been generated using

Photoshop, which presents images in a mosaicked form that can still be effectively communicated to customers.

The ortho-mosaicking software, Agisoft, has very strict overlap requirements in order to create a complete aerial map. During initial flight tests, no useful data could be extracted due to the extreme number of overexposed and blurry images reducing the amount of overlap to the point where Agisoft could not successfully complete mapping. An example of these poor initial results is shown in Figure 6.4. As the amount of overlap increased with adjusted flight speeds, the quality of the maps produced also increased, but results were still well short of a complete map of the vineyard capable of being processed with a vegetation index.

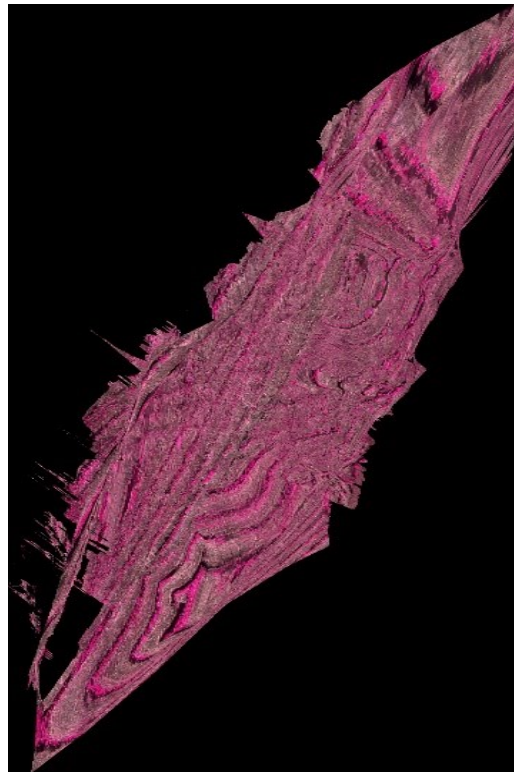


Figure 6.4: Agisoft orthomosaic example

Once the problem with overexposure in the multispectral images was addressed, there was another marked improvement in the quality of the maps being produced due to the increased data being provided to Agisoft. However, these results were still unsatisfactory for vegetation index processing.

Because of these problems, Photoshop was employed as an alternative photo-mosaicking software package and produced far better results. Even using data where a number of photos had to be removed due to overexposure, Photoshop could still produce a complete aerial map of the vineyard, an example of which can be seen in Figure 6.5. Although Photoshop does not provide orthographic capability, it is sufficient for demonstrating a proof-of-concept using vegetation index analysis.



Figure 6.5: Photoshop mosaic example

6.4.3 Vegetation Index

There are a variety of equations and vegetation indices which can be used to obtain crop health information from multispectral data, such as the Normalized Difference Vegetation Index (NDVI), Soil Adjusted Vegetation Index (SAVI), and Green NDVI. UAVino uses the Normalized Difference Vegetation Index (NDVI) because it is based on spectral bands which the Tetracam ADC Micro camera collects and is currently the most widely used vegetation index in the application of multispectral data. Additionally, UAVino began with no knowledge of agricultural monitoring or the processing of multispectral imaging, so it was prudent to experiment with the most well-documented and understood post processing method in order to develop a proof-of-concept.

The NDVI is based on the equation

$$NDVI = \frac{NIR - R}{NIR + R}$$

where *NDVI* is the Normalized Difference Vegetation Index returned on a scale between +1 and -1, NIR is the near infrared channel reflectance, and *R* is the red channel reflectance.

The near infrared channel is used in the NDVI because healthy and unhealthy vegetation have very different levels of reflectance within this band. This difference is caused by photosynthesis, which occurs much more in healthy vegetation and reflects a greater amount of the near infrared spectrum. In contrast to near infrared reflectance, the red reflectance of healthy vegetation is lower than that seen in unhealthy vegetation. This phenomenon leads to a normalized number, as the high levels of NIR reflectance seen in healthy vegetation result in an NDVI value close to +1. As the amount of NIR reflection decreases with a decrease in plant health, the NDVI correspondingly decreases towards a more neutral value of 0. If the multispectral image contains an area with no vegetation, the NDVI is close to a minimum value of -1 since $-R/R$ dominates the equation.

Figure 6.6 depicts how the reflectance of these two channels change between healthy and unhealthy vegetation.

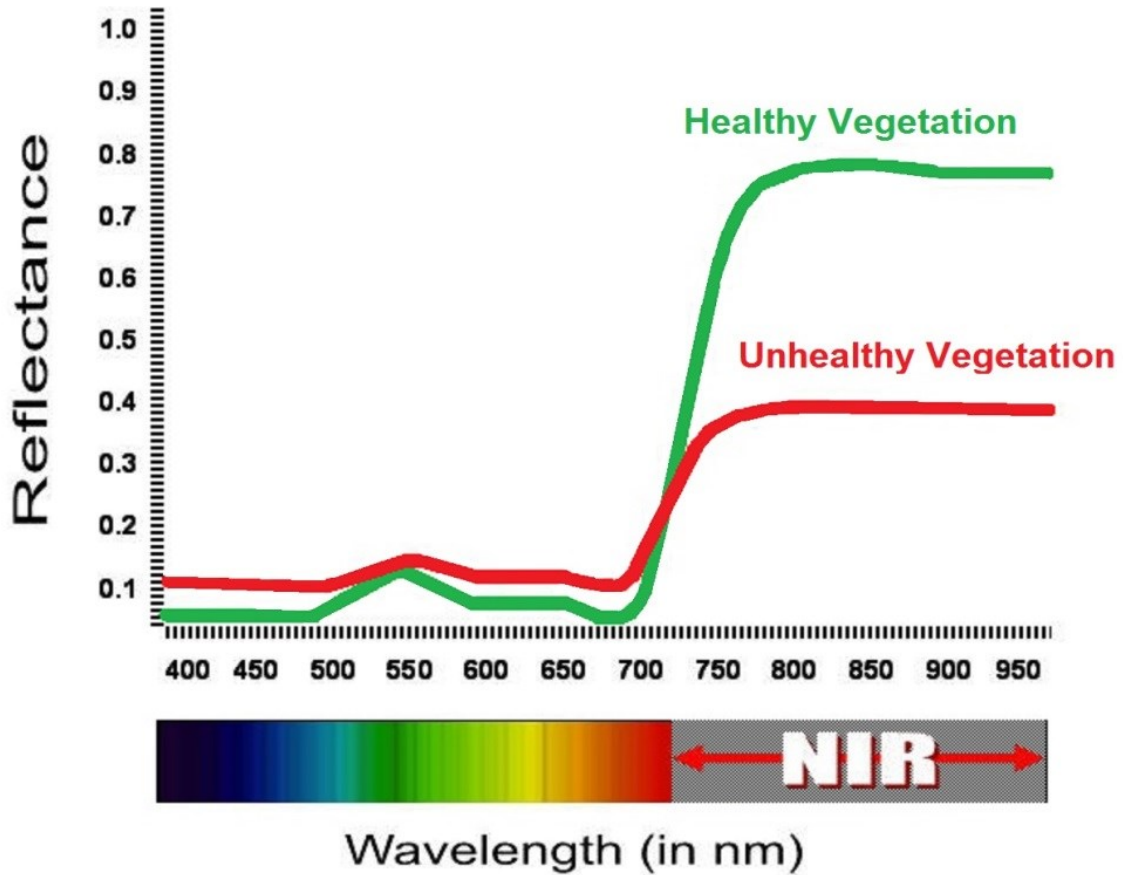


Figure 6.6: Graph of reflectance values across the visible and near infrared spectrum for healthy versus unhealthy vegetation. Photo courtesy of tetracam.com.

UAVino generates NDVI data via Tetracam’s Pixelwrench2 software, which looks at the red, green, and NIR reflectance values stored on a per-pixel basis.

6.5 Post Processing Review

Figure 6.7 provides a more general overview of UAVino’s post processing methods. These steps were discussed in detail in Section 6.4, but are shown in a more abbreviated view for simplicity.

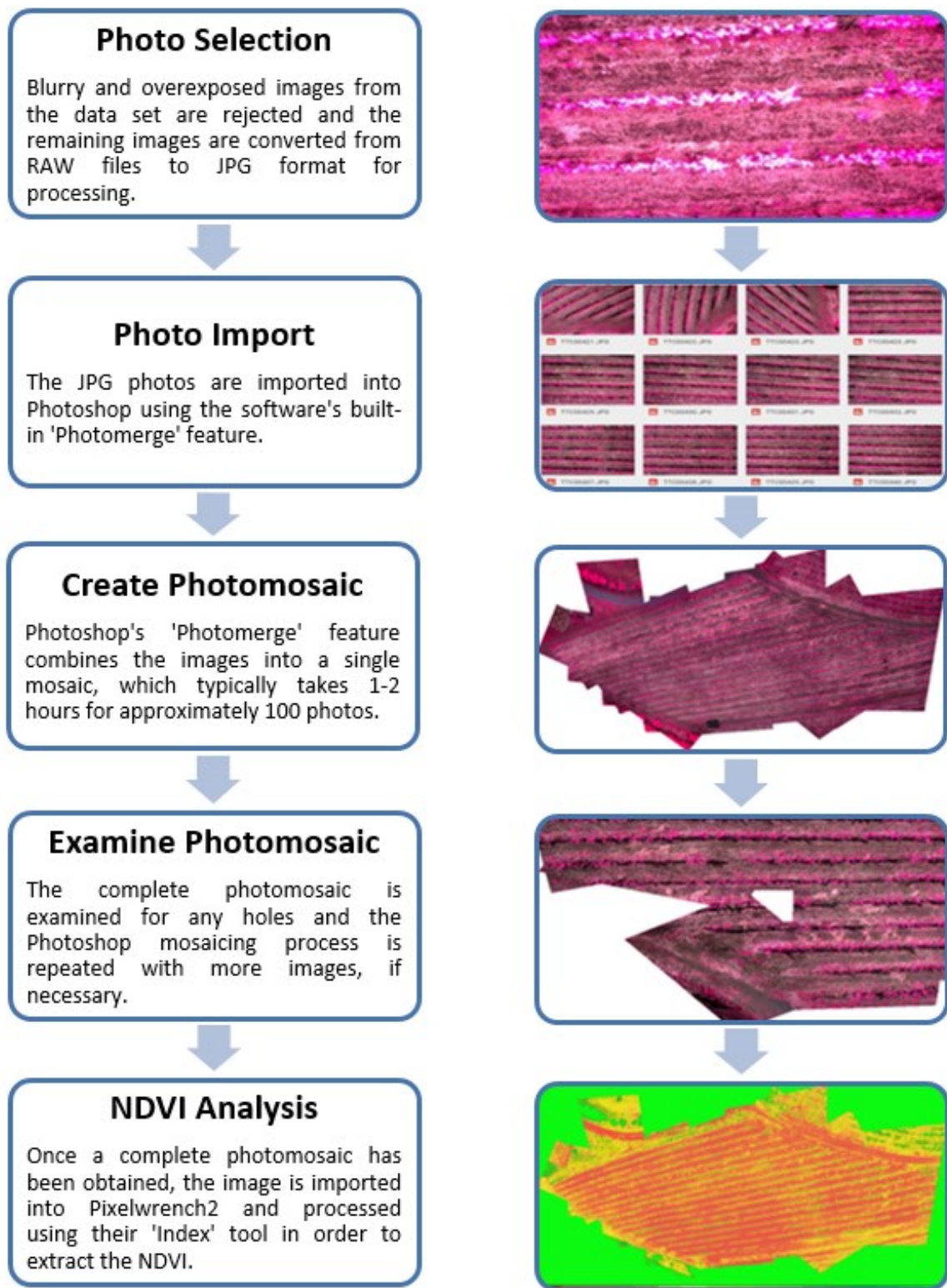


Figure 6.7: Overview of the entire data processing method.

6.6 Verification

6.6.1 Proof of Concept

In order to verify that multispectral data could show failing crop health and that the correct post-processing methods were being employed, testing was conducted to identify a known problem area within the vineyard. Ultimately, UAVino's goal is to locate problem areas before they become known to a customer, but conducting a proof of concept test was critical in proving the feasibility of this data collection process and the authenticity of the results. Figure 6.8 shows the results of this test.

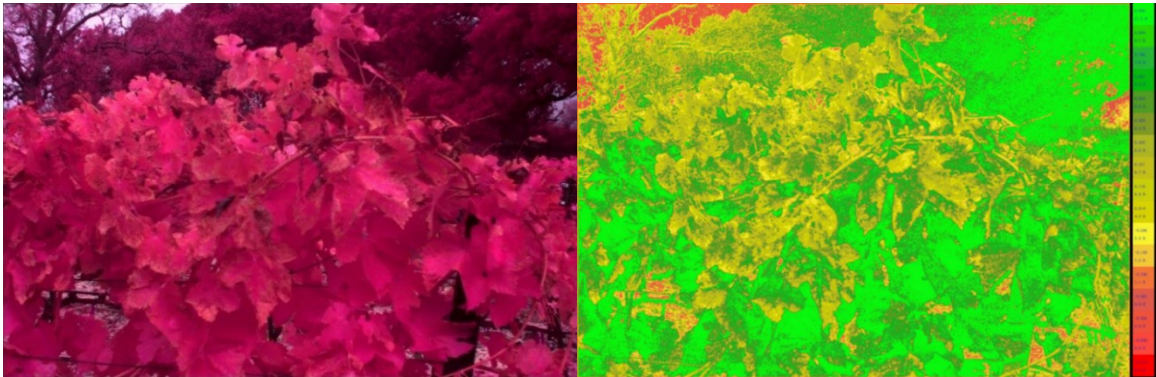


Figure 6.8: Raw multispectral image of a diseased vine and its NDVI processed counterpart. The yellow-green colors in the center of the scale at the right side of the NDVI processed image correspond to NDVI values of 0.0 to 0.4.

The vine in Figure 6.8 was identified by a vineyard owner as suffering from leaf-roll disease, which is clearly visible in the NDVI processed image. The leaves towards the top of the image exhibit a yellow-green color, which correspond to an NDVI of 0.0 to 0.4. These leaves are clearly less healthy than those towards the bottom of the processed image, which are bright green, corresponding to an NDVI near 1.0.

6.6.2 Verification of Crop Health Data

Although a limited amount of data has been obtained thus far, a week-to-week comparison of growth provides an interesting sample of how the NDVI changes with time in a particular field. Figure 6.9 shows the development of the same vineyard over a three week span, during which the vines are undergoing their spring growth spurt.

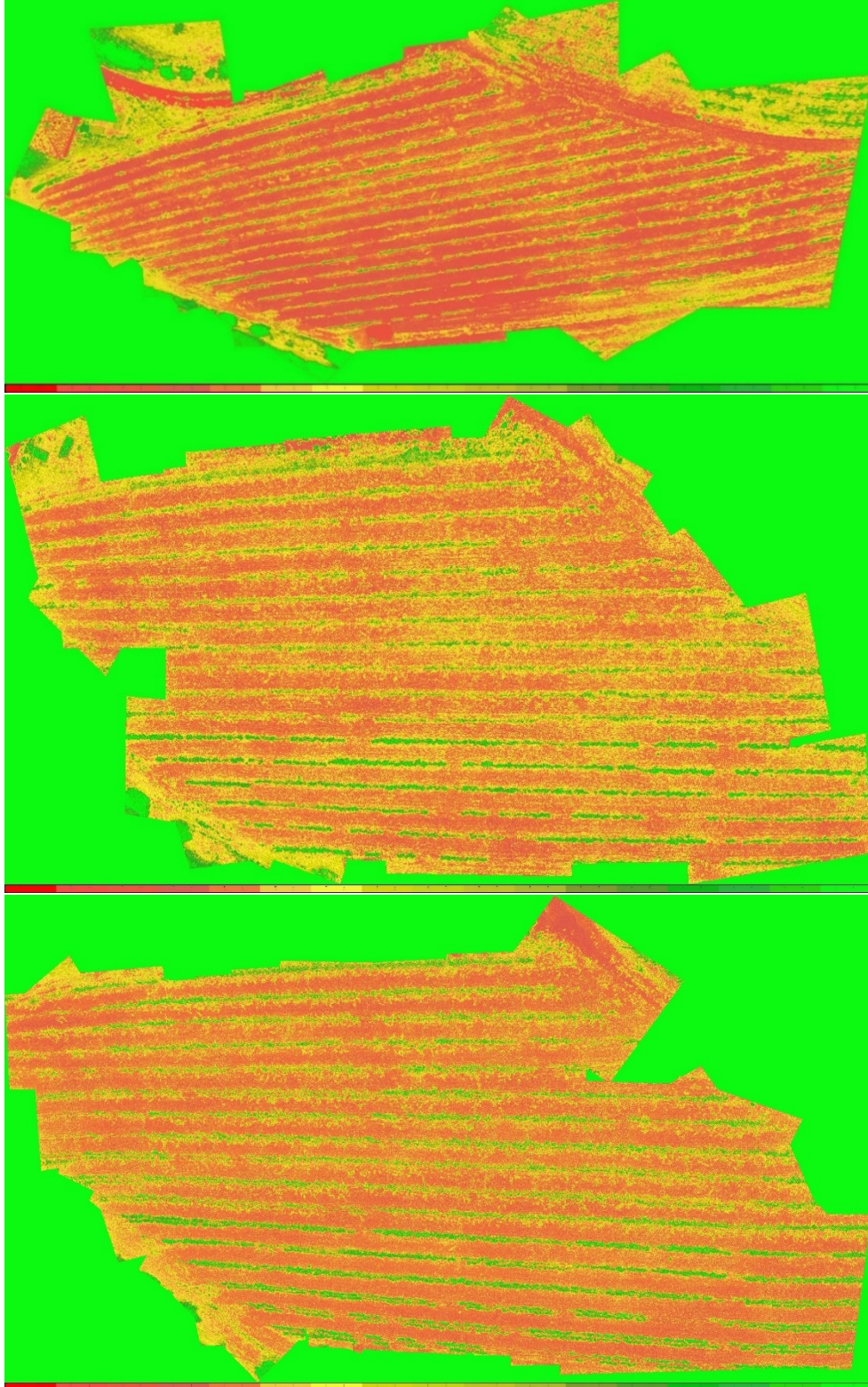


Figure 6.9: NDVI processed images showing the change in a vineyard over the course of three weeks. The uppermost image corresponds to week 1, the middle image corresponds to week 2, and the lower image corresponds to week 3.

Via Figure 6.8, it is clear that a significant amount of growth occurs from the first to the second week, particularly in the bottom six rows of vines. These six rows also show a higher NDVI value, indicating a higher level of photosynthetic activity taking place. Although the change from the second to the third week is not as marked, it is still clear that growth is happening, as the percentage of vegetation falling within the higher NDVI ranges continues to increase. Continuing data collection in this manner would help provide an understanding of how this particular field should develop over the course of the spring, and any deviation from this pattern in future years could be easily noticed.

6.7 Long Term Monitoring

While successfully identifying problems already known to a vineyard owner and showing vineyard growth over time serves as a proof of concept, the real motivation behind UAVino is to help customers identify unhealthy crops before physical evidence is found on the plants themselves. In order to achieve this goal, it is necessary to maintain long term monitoring of an agricultural field. By watching crops over time, it is possible to create a health standard that can be used as a comparison point for future tests. Ultimately, knowing what a particular field should look like at different points throughout the season leads to early detection of anomalies as new data is collected.

Unfortunately, UAVino's short development cycle did not provide enough time to generate a well-established standard upon which to base long term monitoring, as the drone vehicle was not ready to fly until April, at which point plant growth was minimal. Additionally, this early spring timeframe was dedicated towards determining how to best collect data and minimize overexposed and blurry images. However, with data collection and post processing methods now largely functional, future UAVino teams are excellently positioned to begin long term monitoring.

6.8 Summary

The post processing phase of UAVino is a series of steps that takes images collected during flight and turns them into a final data product showing crop health. First, images

are examined and blurry or overexposed photos not suitable for processing are rejected. Second, the remaining images are mosaicked using Photoshop in order to create a single image depicting the acreage mapped. Once a complete photomosaic has been obtained, the resulting image is then processed using Pixelwrench2 software in order to obtain the NDVI, which gives actual crop health data. While a foundation has been set for post processing using this technology, there is still work to be done, as future teams must continue gathering multispectral data to implement long term monitoring. Additionally, work is required to obtain orthomosaicked images using Agisoft software, which provides a more accurate mapping result than Photoshop.

CHAPTER 7

System Testing

7.1 Overview

To gauge the effectiveness of UAVino’s various subsystems, a variety of tests were conducted to characterize performance. The results of these tests were then compared to goals set during the project’s conceptual design phase, which are listed in Appendix A as the Product Design Specification. Ultimately, the build timeline did not allow for sufficient testing of each individual goal. Instead, a few high level tests were conducted in order to confirm key areas of the Product Design Specification and show that although UAVino is only one year into a multi-year development phase, the project is making progress towards becoming a fully autonomous system.

The main performance areas analyzed were octocopter weight, octocopter coverage capability, docking station recharging capability, and vision recognition system accuracy. Assessing whether or not the goals for these major components are met lends insight into the current status of the project and which areas require the most focus of future teams in order to ensure that the project ultimately becomes a viable solution for autonomous crop monitoring.

7.2 Vehicle Weight Tests

Octocopter and various component weights were measured and compared to predictions because these values heavily impact the flight time and coverage capability of UAVino. Significant design effort went towards creating components that are lightweight, yet durable, and assessing the results of these designs helps define whether or not they are successful at performing their intended functions.

Table 7.1 shows the weights of the various components of the octocopter as well as the weight of the flight-ready vehicle, all of which were measured using a traditional gram scale.

Table 7.1: Vehicle Weights

Item	Quantity	Item Weight (g)	Total Weight (g)
Unloaded Octocopter	1	1965	1965
Flight Battery	1	605	605
Multispectral Imaging Camera	1	90	90
Vision Camera	1	40	40
Ring Bracket	2	40	80
Vibration Isolation Camera Mount	1	195	195
Charging Foot	4	35	140
Intel Edison Microprocessor	1	75	75
Total Octocopter Weight			3190

In the Product Design Specification, 7 pounds was set as the predicted flying weight of the vehicle given the empty weight of the base octocopter and the planned additions. As seen in Table 7.1, the actual vehicle weight is very close to this expected value and thus this design criteria is met.

More important than the final vehicle weight value is whether or not this weight allows sufficient flight time to map an agricultural field. Therefore, tests were conducted during operation to gauge an average flight time over a series of flights and battery levels. The predicted flight time goal was 10 minutes, and the fully loaded octocopter was able to achieve this goal in several successive tests. Although this 10 minutes flight time is a decrease from the advertised 12-13 minutes of the empty octocopter, these tests confirm that the added weight of the components does not affect the flight time of the octocopter to the extent that it hinders mapping capability.

It is important to note that the 12-13 minute advertised flight time is an approximate number, as actual results depend up factors such as weather, aerodynamics, and overall piloting ability. Therefore, an additional test not conducted, but that would be appropriate to lend more insight into flight time, would be to conduct drone inspections without the precision docking components. Such a configuration would result in over 300g of weight being removed from the octocopter, leading to an increased flight time. Flight time data could then be compared between flights with and without the

automated docking components in order to more precisely determine the flight time penalty for seeking a fully autonomous system versus a system with a manual operator responsible for takeoff and landing.

7.3 Coverage Capability Tests

The octocopter weight directly affects flight time, which directly affects the coverage capability of the octocopter. In order to determine how much land area UAVino is able to map per unit time, multiple tests were conducted at Aver Family Vineyards in which the drone repeatedly flew the same pre-defined flight path comprising an area of approximately 0.5 acres. This test flight path is shown in Figure 7.1. The flight parameters, such as altitude and lateral speed, were adjusted until viable multispectral data was obtained, and then measurements were taken to see how many times the drone could fly the path on a single battery. This information was then used to extract coverage area per unit time.



Figure 7.1: Mission planning software depicting flight path.

With the octocopter flying the path shown in Figure 7.1 at a height of 20m and a lateral speed of 1.75m/s, the drone was able to cover the 0.5 acre area flight path twice with each battery. These two flights, lasting nearly 10 minutes in total, indicate that UAVino has a coverage capability of approximately 1 acre per flight battery, which equates to 6 acres per hour, assuming continuous flight. Adjusting for recharging using the docking station, which takes approximately 60 minutes to recharge a battery that has been

flown for 10 minutes, UAVino is able to cover approximately .85 acres per hour. Although this number is lower than desired, it is sufficient to handle small-scale vineyards and allow for practical and timely vineyard crop inspection.

7.4 Recharging Capability Tests

Another factor determining how much land area UAVino can cover is the time it takes to recharge the octocopter's battery. The goal is to minimize charging time while still remaining safe, as doing so increases the time that the drone is able to spend mapping. To quantify the performance of the docking station's recharging capability, tests were conducted to compare the charging rate of the station versus conventional recharging methods.

For the docking station test, recharging was facilitated via the station's housed commercial charger and power was supplied from the system's 12V marine battery. The octocopter was placed on the station's charging contacts and the charge sequence was initiated. The conventional charging method test was facilitated by the same commercial charger, but plugged into a standard wall outlet as the power source and the battery was connected directly to the charger rather than via the station's contact plates. Figure 7.2 shows charge curves depicting both of these tests.

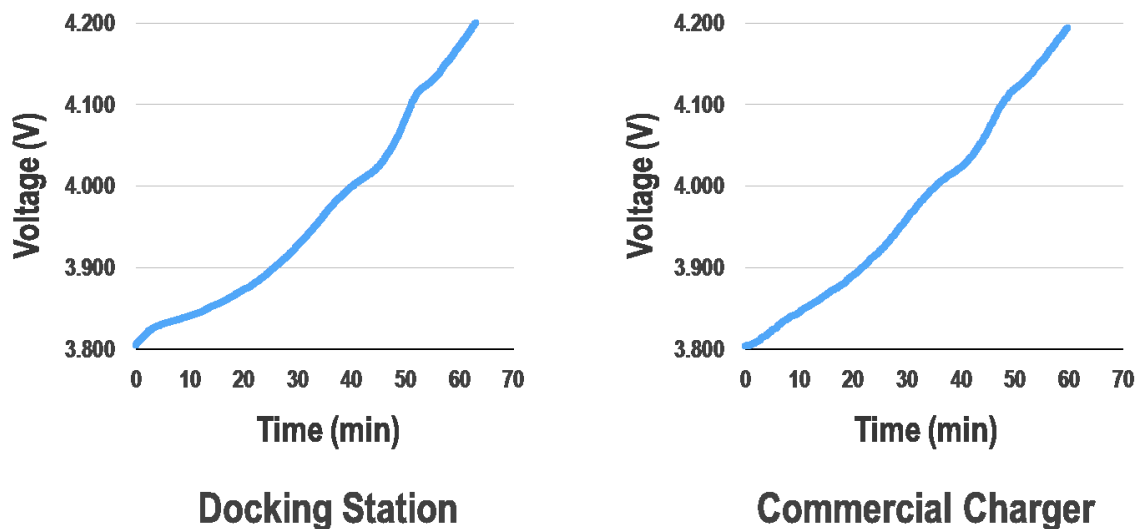


Figure 7.2: Charge curves for docking station and standard commercial charger.

As Figure 7.2 shows, it is clear that the two charge curves are nearly identical and therefore the recharging performance of the docking station is as good as conventional recharging methods. Additionally, the docking station is able to recharge a battery in under 90 minutes, which is a goal set forth in the Product Design Specification. Although battery recharge time does depend upon the level of discharge within the battery, both of these tests were conducted with batteries that had been flown for approximately 10 minutes with the octocopter, which is close to the flight time that the system will actually experience during operation.

7.5 Vision Recognition Tests

The automated landing system is not yet functional and therefore it is not possible to conduct tests regarding this subsystem as a whole. However, it is known that at least a 2.5 inch accuracy is required for the octocopter to successfully dock with the station. Therefore, lab tests were conducted to compare the landing algorithm's estimated lateral error against an actual known value at varying altitudes. Figure 7.3 shows these test results, which ultimately prove whether vision recognition is capable of meeting the 2.5 inch requirement.

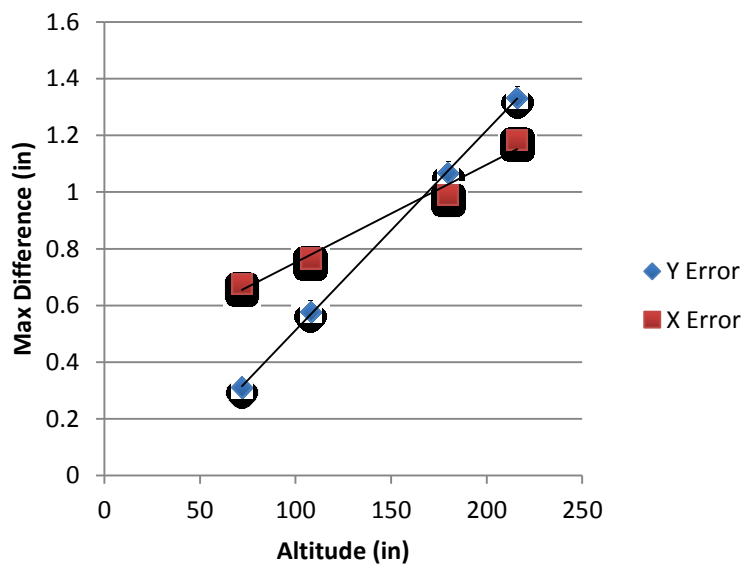


Figure 7.3: Lab tests showing the error between the landing algorithms' estimated lateral distance and the true lateral distance versus altitude.

Figure 7.3 shows that UAVino's vision recognition algorithm is able to deliver the required accuracy at all altitudes. Furthermore, the system becomes more precise as the octocopter descends towards the station, thus proving that this system is a promising method of allowing the octocopter to achieve automated landing.

7.6 Summary

To ensure that all of the subsystems of UAVino are able to work in harmony and eventually allow for fully autonomous crop monitoring, it is crucial to conduct tests to measure their performance. Testing was primarily conducted on vehicle weight, system coverage capability, docking station recharging, and vision recognition algorithms. These key subsystems have their respective challenges, but each is critical to the full integration and autonomous functionality of UAVino.

Ultimately, tests for octocopter weight, docking station charging, and vision recognition proved satisfactory. While the coverage capability of .85 acres per hour is lower than desired, it is sufficient for a proof-of-concept system geared for small-scale vineyards. The major limitation to coverage capability is the recharging time of the octocopter, as it must spend nearly six times as long recharging as it does flying. Therefore, improvements to this system are the most likely way to increase coverage capability in the future.

CHAPTER 8

Costing Analysis

8.1 Budget and Costs

UAVino’s allocated budget comprises of a \$3,000 grant from the Santa Clara University School of Engineering and a \$750 grant from the Silicon Valley Section of the American Society for Mechanical Engineers. Additional support for UAVino comes from the Santa Clara University Robotics Systems Laboratory, which has supplied the team with the base 3D Robotics X8 octocopter and a variety of tools to work with. The multispectral camera and Edison microcontroller boards have been donated by Intel Corporation. UAVino is grateful to these sponsors for their support of the project.

Table 8.1 shows the cost of UAVino’s major components. More detailed budget analysis is available in Appendix F.

Table 8.1: General UAVino cost breakdown

Component	Cost	Notes
Multispectral Imaging Camera	\$2,500	Donated by Intel
Octocopter with Telemetry	\$1,500	Provided by SCU RSL
Octocopter Hardware and Electronics	\$450	Intel Edison Donated by Intel
Docking Station Hardware	\$300	
Docking Station Electronics	\$600	
Travel	\$350	
Bench testing Electronics	\$350	
Total	\$6,050	

CHAPTER 9

Commercialization Plan

9.1 Executive Summary

Over the past few years, a dramatic increase in the capabilities of small aerial drones has created a potential for their use in a wide variety of commercial applications. In an attempt to enter this market, UAVino seeks to utilize octocopters in the field of vineyard inspection to monitor overall crop health using multispectral and infrared imaging technology. Long term, UAVino is envisioned as a multi-year, interdisciplinary project involving both the Santa Clara University Robotics Systems Laboratory and local wineries in order to develop a fully functional drone agricultural inspection service.

9.2 Background

The goal of UAVino is to develop a fully autonomous drone agricultural inspection system that is offered by the Santa Clara University Robotics System Laboratory as a service to local wineries and other agricultural operations. In general, the system focuses on automating crop inspection by taking aerial imagery of a vineyard, conducting post-processing, and outputting an easily interpreted map of the vineyard's overall health. The project's key innovation is an auto-docking system that allows the drone to automatically return to its launch point and recharge in order to extend mission duration.

Aside from drones, the primary methods for obtaining crop health include sensor networks and satellite imagery, both of which are lacking. For example, sensor networks are generally unavailable to small scale agricultural operations due to their complexity and upfront cost. Instead, owners of small wineries typically resort to manual inspection techniques, which are labor intensive and far from ideal. Similarly, satellite imagery is problematic due to the need to contract with an external imaging service as well as weather and data resolution limitations.

Although agricultural-use drones are a quickly growing field and a number of companies are looking to offer systems similar to UAVino, there is no company that currently offers a fully autonomous, easy-to-deploy contract service to determine crop health. Therefore, UAVino stands to offer a unique service to the agricultural industry.

9.3 Goals and Objectives

UAVino is imagined as a crop inspection service that students in the Santa Clara University Robotics Systems Laboratory will be able to provide to local farmers on an as-needed basis. Via this service-type implementation, individual farmers will not need to invest significant capital to purchase UAVino outright or spend time training and familiarizing themselves with how to operate complicated technology. Instead, interested farmers will simply request the service when needed and benefit from the crop health information produced.

From a monetary standpoint, the goal of UAVino is to have the system break even so that it can be self-sustaining and continue to serve as a base for future educational projects associated with the Robotics Systems Laboratory. Any profit gained from conducting agricultural inspection would likely be reinvested into the system for future improvements. Thus, the goal is not to make the system profitable to the extent that it could be implemented as a full scale business with several employees, although such an implementation might be possible. Instead, the objective is to create a solid educational platform that pays for itself and allows future Santa Clara University students to gain an understanding of business fundamentals and experience real customer interactions.

9.4 Key Technologies

The key technological innovation developed in UAVino is a portable station that the octocopter automatically returns to, docks with, and then recharges from in order to extend mission duration. The need for this station arises from the relatively low flight time of octocopter vehicles, which is typically under 15 minutes per flight battery. Therefore, inspecting large amounts of acreage requires multiple flights in order to

complete. Typically, this limitation has required the vehicle operator to manually replace or recharge the flight battery multiple times in order to continue a mission. However, UAVino's docking station mitigates this need, as it greatly increases system autonomy and reduces operator workload.

9.5 Customers and Marketing

UAVino's target customers are small California vineyard owners who operate with relatively low excess capital therefore are on a tight budget. Currently, these vineyards are experiencing added difficulty due to high water costs associated with California's drought. If an inexpensive and easy-to-use drone crop monitoring system was developed, it could greatly help these vintners by providing information regarding where and how to most efficiently water their fields.

UAVino has already established a working relationship with one vineyard near Gilroy, CA in order to conduct testing. There are over 25 local vineyards in the Santa Clara Valley alone, and some of these vineyards have already expressed interest in UAVino should the system become fully-functional. Additionally, given that Napa and Sonoma Valleys, which are the epicenter of winemaking in California, are both within reasonable driving distance from Santa Clara University, the potential market for UAVino is vast.

Because the goal of UAVino is to maintain a close and personal relationship with customers in order to provide them with meaningful data while using their vineyards as a means of testing an educational project, any project expansion would likely occur slowly and by word of mouth amongst local vineyard owners. No extensive advertising or marketing campaigns are anticipated in widening the project.

9.6 Manufacturing

Given the relatively simple service that the Robotics Systems Laboratory would offer to local vineyards and the fact that a small number of docking stations and octocopters would actually need to be built and modified, manufacturing could likely take place in-

house by students to minimize overhead. The required components to build a drone and docking station pair are listed in Table 9.1.

Table 9.1: General UAVino cost breakdown

Component	Cost
Multispectral Imaging Camera	\$2,500
Octocopter with Telemetry	\$1,500
Octocopter Hardware and Electronics	\$450
Docking Station Hardware	\$300
Docking Station Electronics	\$600
Travel	\$350
Bench testing Electronics	\$350
Total	\$6,050

9.7 Service Cost and Price

Although the upfront cost of \$6,050 per drone and docking station pair is relatively significant, very little maintenance is required long-term and therefore the system would become profitable after recouping the initial investment. Given that UAVino will remain a project that rotates on annually, collecting sufficient funds to replace the system on an annual basis would make it sustainable from an educational standpoint.

Testing revealed that UAVino can map approximately 10 acres in one day. Given that most small scale vineyards are less than 20 acres, it is reasonable to assume that UAVino could map one vineyard per week, given that testing would typically need to occur on weekends to minimize impact on student academic workloads. Due to Santa Clara University’s 10-week quarter system, assuming one vineyard mapping trip per week would equate to 30 trips per year. Therefore, garnering \$200 from each trip would yield an annual income of \$6,000, which would be sufficient to purchase a new drone and docking station combination each year.

After a sufficient clientele is built up and more data is collected on how many acres each vineyard is on average, the \$200 per visit could better be broken down into an hourly or per acre rate.

CHAPTER 10

Engineering Standards and Realistic Constraints

10.1 Ethics Analysis

As UAVino was developed, several important ethical considerations were held paramount. These same considerations must be adhered to as the project progresses in future years. First, intellectual property is of importance due to the rapidly growing field of unmanned aerial vehicles. Both entrepreneurs and well-established companies are working to develop technology for drones, and therefore it is UAVino's responsibility to thoroughly research and understand what possible ideas and solutions are claimed as intellectual property by these parties. Because so much energy is currently being directed towards drones for agricultural research, special attention must be paid to these types of projects in particular to ensure that UAVino's design solutions do not inadvertently infringe upon any approved patents. Beyond intellectual property, it is also the responsibility of UAVino to analyze the research conducted by other universities and institutions to ensure that the team does not accidentally claim any previously published findings regarding agricultural drones as unique to UAVino.

Another major ethical consideration is to ensure that the finished system delivers accurate and reliable information. The real-world customers who contract with UAVino trust that the information provided regarding crop health is accurate. Supplying these customers with false data, even if done so inadvertently, could lead to significant crop and profit loss. Therefore, the system must be subject to rigorous testing to verify that it works as intended and is able to provide the best possible data.

Thirdly, it is the responsibility of UAVino operators to keep in mind the privacy of others while the drone is flying. Although agricultural fields are somewhat remote, the camera equipment on board the octocopter could capture individuals who do not wish to be photographed. Care must be taken to ensure that the system is operated in such a manner that respects the privacy of others by censoring parts of images that

accidentally depict individuals. Invasion of privacy is currently one of the biggest criticisms of personal drones, so giving special consideration to this seemingly remote possibility is critical, as failing to do so might negatively contribute to the drone privacy argument.

Provided that these considerations are carefully monitored during all phases of the project, UAVino can be an ethically justified system given the positive impact it stands to make on the agricultural industry. California is currently in a state of severe drought and small scale farmers are hard pressed to reduce water usage. UAVino has the potential to help these individuals more accurately monitor the condition of their crops and therefore better cope with this challenge. Because of the major benefits UAVino can bring to an industry in need and the ability to mitigate the ethical risks associated with doing so, the project is justified in its cause.

10.2 Legal Analysis

The United States Federal Aviation Administration (FAA) is beginning to implement a series of drone regulations to help integrate this new technology into the National Airspace System. While the exact details of these regulations might seem vague in some areas and excessive in others, they are in place to keep the general public safe. Therefore, UAVino developers and operators must become familiar with these laws in order to prevent an inadvertent violation and ensure that all activity is conducted legally. Ensuring continued compliance with these regulations is of particular importance to future teams who wish to take UAVino's design further, as government regulations are expected to change significantly over the next few years as unmanned aerial drones continue to be safely introduced into the nation's airspace.

Aside from FAA restrictions, future UAVino applications might involve flights at Santa Clara University, meaning that University-specific guidelines must also be read and understood. Therefore, it is necessary for the team to communicate and coordinate with the University's Facilities and Health and Safety departments to ensure that operations are conducted in compliance with all relevant parties.

10.3 Health and Safety Analysis

Safety is a primary concern for UAVino, as drone malfunctions have the very real possibility of damaging property or causing injury, particularly if the vehicle is airborne when a failure occurs. To minimize the risk of such an incident, detailed flight procedures have been developed and are strictly adhered to during operation. Although it is not possible to completely eliminate the safety risks of operating an aerial drone, ensuring that the same checklist is followed in the same way during every deployment reduces the possibility of human error causing an accident.

Beyond following checklists and procedures, it is the team's responsibility to use common sense when operating and conduct thorough testing to ensure that the system is safe to the fullest extent possible. To operate UAVino knowing that a potential safety flaw exists would be a serious ethical violation, as customers must be able to trust that the system can be used without seriously risking injury. To see physical or monetary harm result from the manifestation of such a flaw would be truly awful, especially if something could have been done to prevent the issue.

10.4 Manufacturability Analysis

Given that the centerpiece of UAVino, the octocopter, is based upon a commercially available drone vehicle with fairly straightforward modifications, the system is easily manufactured should it ever need to be massed produced or should other parties be interested in replicating it for academic use. All system components, including the docking station, were manufactured using basic machine shop tools; no professional knowledge or manufacturing was required.

Although it was not an overarching goal during manufacturing, design effort was put into using off-the-shelf parts in order to reduce cost and make the system easily modifiable for future system developers. Combined, the modified octocopter, without the multispectral camera, and the docking station cost less than \$2,500. Therefore, the system is an extremely affordable base platform that can be extended for other aerial

applications. For example, UAVino could easily be applied towards heat mapping or videography applications with the implementation of the proper camera system.

Overall, UAVino is a straightforward, easy-to-manufacture system that embodies the current open source movement of personal use drones so that others can build and improve upon its design in the future.

10.5 Social Impact Analysis

With regards to social impact, one major goal of UAVino is to help vintners reduce water usage through more efficient practices. Vineyard health and crop yield are highly dependent upon the amount of water stress within the crops and a significant amount of the growing effort goes towards achieving a correct water stress balance.

Within California, water usage is a focal point due to severe levels of drought, and the agriculture industry has been targeted the state's largest water consumer. Overall, California's agricultural industry accounts for 80% of the state's water consumption, yet only 2% of its economic activity [21]. Therefore, a huge gap exists between the impact agriculture causes on the state's water budget versus the benefit it brings in terms of money. A particularly interesting fact surrounding this statistic, however, is that if California's agricultural businesses could reduce water usage by just 12.5%, it would allow for statewide residential and industrial use to increase by 50% [22]. This considerable increase in availability for water in residential and industrial applications would be able to relieve significant stress associated with California's current drought.

Given how a relatively small reduction in agricultural water usage can yield a significant amount of relief on the wider population, a method of judging UAVino's potential is whether or not the system might be able to provide a 12.5% reduction in water usage on a small, per vineyard basis. If so, this reduction could prove significant if similar agricultural drone solutions were implemented on a larger scale statewide. Although a 12.5% reduction is ambitious, analysis shows that UAVino and similar systems can make

a meaningful impact, perhaps 6%, by helping with efficient watering methods called irrigation scheduling and regulated deficit irrigation.

A recent study by the Natural Resources Defense Council looked at potential efficient watering techniques, defined as “measures that reduce water use without affecting the benefits water provides” [23]. Overall, the article discussed three methods which, when combined, could reduce water usage by 5.6-6.6 million acre-feet per year, or 17-22% if implemented statewide in California. Of the methods discussed, UAVino could greatly aid with two: irrigation scheduling and regulated deficit irrigation. Irrigation scheduling relies on careful planning involving local weather predictions, soil water content, and plant water requirements to most efficiently water crops. Regulated deficit irrigation is a technique applied towards crops which have periods in which they are drought resistant, such as almonds, pistachios, and wine grapes. The idea with regulated deficit irrigation is that during drought resistant periods, these crops can undergo a significant reduction in watering without causing detrimental effects to their health. If these two practices were adopted on a large scale, they could account for a savings of up to 4.8 million acre-feet during a dry year in California. Based on the statistics cited in the Natural Resources Defense Council study, employment of these methods could amount to water savings of roughly 15%, even more than the 12.5% required to cause a 50% relief in residential and industrial usage.

Both irrigation scheduling and regulated deficit irrigation involve closely monitoring crop growth in order to conserve water. One article on the subject stresses that “Regardless of the type of irrigation program used, there is a need to develop scientific irrigation scheduling procedures. In particular, if [deficit irrigation] is used, monitoring the soil or plant water status is even more critical for minimizing risk, given the uncertainties in determining the exact water requirements” [24]. The remote data which can be provided by drones, such as multispectral and infrared imaging, could be particularly helpful in this process and is cited by the same article as a method of monitoring that could help with the application of deficit irrigation. Ultimately, multispectral crop

monitoring systems such as UAVino can give an indication of crop vigor, which has a direct correlation to plant water stress. Providing farmers with this information could help them better schedule their irrigation and provide them with a higher level of comfort in adopting these reduced watering practices. Although the presence of remote monitoring systems such as UAVino may not lead to widespread adoption of these practices, if the increased information provided could even result in a 30% adoption of irrigation scheduling and deficit irrigation, it would result in a 5% agricultural water savings.

10.6 Arts

As part of satisfying the Santa Clara University Core Arts and Humanities requirements, members of UAVino have contributed original drawings, sketches, and CAD models to the project. Table 10.1 lists a sampling at least one such artifact, and a reference to it, for each of the team members.

Table 10.1: Arts requirement

Team Member	Description	Location
Matthew Belesiu	Landing algorithm error and movement types.	Figure 5.8 and Figure 5.10.
Nathan Carlson	Recharging sketches and docking station CAD drawings.	Figure D.4 and Appendix H.
Aaron Chung	Docking station and octocopter circuit diagrams.	Figure 3.16 and Figure 4.11.
Phillip Coyle	Recharging and docking concept sketches.	Figure D.1 and Figure D.3.
Kirby Linvill	Landing algorithm logic flow and movement selection process.	Figure 5.6 and Figure 5.11.
Megan Peekema	Recharging and docking concept sketches.	Figure D.2 and Figure D.5.

CHAPTER 11

Summary and Conclusions

11.1 Project Summary

The long term goal of UAVino is to create and offer a fully autonomous drone crop inspection system to small-scale, local vineyards. Although that goal has not yet been met, significant progress has been made during this project's first year, which was geared towards creating a proof-of-concept. Currently, the system consists of an octocopter vehicle that has been modified with guidance rings for precision docking and equipped with a multispectral imaging camera to take specialized photographs. Automated landing has not yet been implemented, but significant progress has been made, including the creation of a computer vision landing algorithm that identifies the docking station and navigates the vehicle towards it. Despite the lack of precision docking, the octocopter has collected numerous multispectral images, allowing for the creation of post processing methods that output an easily interpreted data product depicting crop health. Additionally, UAVino consists of a docking station that has demonstrated capability of recharging the octocopter's flight battery to extend mission duration and range. In the future, wireless communication between this station and the octocopter will be tested in order to allow for completely autonomous recharging. Once functional, UAVino will be a complete system that seamlessly integrates with current vineyard practices, meaning that it is a cost effective solution with minimal infrastructure required for deployment.

By working in conjunction with a real-world customer to develop the UAVino, the end result is one that is practical and has been demonstrated in actual application. Through this type of verification, the system stands to make a real-world impact on the agricultural industry. More generally, UAVino contributes to the growing field of commercial drone applications by pushing the boundaries of autonomous drone operations. Through its demonstration of precise automated landing capability, the

project will hopefully stand as an inspiration for others to develop and expand similar drone technology.

11.2 Future Work

Although the system is not yet complete, a solid foundation has been laid for future teams to continue developing UAVino and ultimately reach fully autonomous mapping capability. Undoubtedly, the system which is most difficult to develop and still requires the largest amount of work to complete is autonomous docking. This year, the team made solid progress towards using a vision camera and microprocessor to recognize and navigate the drone towards the docking station, although actual autonomous landing was not completed. Future teams will inherit the basic capability of controlling the drone via this vision recognition algorithm, but will need to further develop consistent and more accurate forms of judging the drone's location in relation to the docking station. A potential solution to this problem may include augmenting the current vision system with infrared beacons or modulated lights in order to decrease false positive identifications. Such a system may also help with yaw orientation control to align the octocopter's hoops with the docking station cones, which has yet to be implemented. Additionally, more work is required to create and then tune a proportional-integral-derivative control system to more accurately fly the octocopter with the vision guided algorithm and reduce phenomena such as overshoot when attempting to center over the station. Future teams will also need to implement the flight management algorithm that controls overarching drone decisions, such as at what point the drone must stop mapping and return to the docking station to recharge, as well as what sections of the desired fields still require mapping during a mission. Finally, although framework for wireless communication between the drone and docking station has been implemented, this system still requires testing and debugging. Ultimately, these features are the key elements that must be solved before a fully autonomous system is possible.

A great strength of UAVino is that the system is highly modular and therefore future teams can augment the initial functionality to provide better crop health data. For

example, a straightforward expansion is to collect infrared data in addition to multispectral data, which would allow for direct measurement of soil and plant water content in addition to overall health. The combination of these two monitoring techniques could give vineyard owners a much more powerful picture of their field's health and provide insight on how to more effectively use water resources. Infrared imagery would also open the doors to entirely different monitoring systems, such as inspecting buildings for sources of heat loss.

UAVino is just a small subset of a much broader and more exciting movement towards implementing drones for commercial applications. Therefore, this system is likely to be presented with new opportunities as others work to develop new capabilities and make drones more adept at completing complex tasks. In the future, UAVino could evolve into a complex and comprehensive crop monitoring system that uses multiple drones interacting on a real-time basis in order to more efficiently map larger areas of crops. Overall, the future of drones and, with it, the future of UAVino, is a thrilling and ever-expanding horizon that should be closely watched in the years to come.

References

- [1] Warwick, Graham. "Sky Patrol." *Aviation Week & Space Technology* 174.32 (2012): 55. Web. 29 Nov. 2014.
- [2] Marks, Paul. "Drone Backlash Begins." *New Scientist* 217.2906 (2013): 24. Web. 29 Nov. 2014.
- [3] Dorr, Les Jr. and Alison Duquette. "U.S. Transportation Secretary Foxx Announces FAA Exemptions for Commercial UAS Movie and TV Production." *Federal Aviation Administration*, 25 Sep. 2014. Web. 29 Nov. 2014.
- [4] "Amazon Prime Air." *Amazon.com*, n.d. Web. 8 Dec. 2014.
- [5] Hallewas, Claire. "TU Delft's ambulance drone drastically increases chances of survival of cardiac arrest patients." *Delft University of Technology*, 27 Oct. 2014. Web. 8 Dec. 2014.
- [6] Goodyer, Jason. "Drone Rangers." *Engineering & Technology* (17509637) 8.5 (2013): 60-61. Web. 7 Jan. 2015.
- [7] Marks, Paul. "Flying Conservationist." *New Scientist* 220.2938 (2013): 19. Web. 7 Jan. 2015.
- [8] Dillow, Clay. "What is the drone industry really worth?" *Fortune*, 12 Mar. 2013. Web. 8 Dec. 2014.
- [9] Carlson, Toby N., Gillies, Robert R., and Eileen M. Perry. "A Method to Make Use of Thermal Infrared Temperature and NDVI Measurements to Infer Surface Soil Water Content and Fractional Vegetation Cover." *Remote Sensing Reviews* 9.1-2 (1994): 161-73. Web 8 Dec. 2014.

- [10] Carlson, Toby. "An Overview of the "Triangle Method" for Estimating Surface Evapotranspiration and Soil Moisture from Satellite Imagery." *Sensors* 7.8 (2007): 1612-1629. Web 8 Dec. 2014.
- [11] Vadivambal, R., and Digvir S. Jayas. "Applications of Thermal Imaging in Agriculture and Food Industry—A Review." *Food and Bioprocess Technology* 4.2 (2011): 186-99. Web 8 Dec. 2014.
- [12] Stoll, M., Schultz, H.R., and Berkelmann-Loehnertz, B. "Exploring the sensitivity of thermal imaging for *Plasmopara viticola* pathogen detection in grapevines under different water status." *Functional Plant Biology* 35.4 (2008): 281–288. Web. 9 Jan. 2015.
- [13] Johnson, L., Herwitz, S., Dunagan, S., Lobitz, B., Sullivan, D., and Slye, R. "Collection of Ultra High Spatial and Spectral Resolution Image Data over California Vineyards with a Small UAV." *International Symposium on Remote Sensing of Environment*. 2003.
- [14] Johnson, L., Roczen, D., Youkhana, S., Nemani, R., and D. Bosch. "Mapping Vineyard Leaf Area with Multispectral Satellite Imagery." *Computers & Electronics in Agriculture* 38.1 (2003): 33. Web. 10 Jan. 2015.
- [15] Lacar, F., Lewis, M., and Grierson, I. "Use of Hyperspectral Imagery for Mapping Grape Varieties in the Barossa Valley, South Australia." *Institute of Electrical and Electronics Engineers*. 6 (2001). Web. 18 Nov. 2014.
- [16] Ashish, D., McClendon, R., and Hoogenboom, G. "Land-Use Classification of Multispectral Aerial Images Using Artificial Neural Networks." *International Journal of Remote Sensing*: 30.8 (2009). Web. 18 Nov. 2014.
- [17] *Cropio*. Web. 18 Nov. 2014.
- [18] *Precision Hawk*. 1 Jan. 2014. Web. 18 Nov. 2014.

- [19] *3D Robotics*. Web. 18 Nov. 2014.
- [20] Bernstein, Lenny. "California Drought Hits Farmers Hardest." *The Washington Post*. N.p., 9 Feb. 2014. Web. 17 Nov. 2014.
- [21] "The Drying of the West." *The Economist*. The Economist Newspaper, 22 Feb. 2014. Web. 19 Apr. 2015.
- [22] Tabarrok, Alex. "The Economics of the California Water Shortage." *Marginal Revolution*. 19 Mar. 2015. Web. 19 Apr. 2015.
- [23] Cooley, Heather. "Agricultural Water Conservation and Efficiency Potential in California." *Natural Resources Defense Council* (2014).
- [24] Fereres, Elias and Mari´a Auxiliadora Soriano. "Deficit Irrigation for Reducing Agricultural Water Use." *Journal of Experimental Botany* 58.2 (2007): 147-59. Web. 21 Apr. 2015.

APPENDIX A

Product Design Specification

Revision: 5

Date: 26 May 2015

Datum: 3D Robotics X8 Octocopter

Table A.1: Product Design Specification

Elements / Requirements	Units	Datum	Target / Range
System Cost	Dollars	5,400	8,000
Octocopter Weight	Pounds	5.4	7
Flight Time	Minutes	12-13	10
Recharging Time	Minutes	60	90
Setup Time	Minutes		20
Octocopter Digital Storage	Gigabytes		128
Multispectral Camera Battery Life	Minutes		120
Vision Camera Battery Life	Minutes		120
System Mapping Capability	Acres/Hour		2
Thermal Range	Degrees Fahrenheit	32-104	30-105
Vision Algorithm Accuracy	Inches		2.5
Octocopter Size	Inches	24"x24"x8"	30"x24"x12"
Docking Station Size	Inches		36"x24"x60"

APPENDIX B

Customer Interview Questions

Below are the interview questions asked to Professor Christopher Kitts.

- Because you envision UAVino as a possible multi-year project, what is the ultimate long term value or goal of it? Is the idea simply to make money by providing a service or product to vineyard owners, or is the return on investment instead in the networking opportunities that might stem from developing UAVino and gaining name recognition as a knowledgeable drone developer?
- Is the ultimate end result of UAVino a product that you would ultimately sell to vineyard owners to make a profit, or would you instead create a service using that product and then rely on vineyard owners to contract with you when they need vineyards inspected? Whether it's a product you sell, a service you provide, or a combination, what's the motivation behind your answer?
- As a stakeholder or investor, what are the primary concerns you have that might result in UAVino being unsuccessful. Among these potential concerns, is there any fear that UAVino might be 'white noise' given that the number of companies seeking to make a profit via aerial mapping drones is growing rapidly?
- Given that aerial mapping drones are aimed at a fairly niche market, how much of that market would UAVino need to capture in order to be successful? If it will ultimately be a product that you sell, how many units would need to be sold to justify development costs? If it will be a service that you provide, how many customers would you need in order to make it justifiable as a business venture?

Below are the interview questions asked to Thomas Adamek.

- Do you see this product being more viable as a service which someone can rent, or something to be bought and operated by the customer. Why?

- What do you think is the potential long term value of this project as it may be a multi-year project? What do you think future groups could expand upon?
- Does this project have any elements that stand out to you as fulfilling a unique need in the field? If yes—what? If no—do you think it has potential to develop in that direction?
- Is there anything we could do as a group to make a transition of this project to a future team easier for you?
- Do you have any concerns that may keep this project from being successful?

Below are the interview questions asked to Lindsay Kalkbrenner.

- How can our project help on campus?
- How often do you do field inspections on campus?
- What are the safety restrictions and regulations for flying a UAV on campus?
- What types of data do you want collected?
- Would you be interested in working exclusively with future senior design teams?
- If we were selling our product, would you like to purchase the system for Santa Clara University and be responsible for performing inspections, or would you rather the Robotics System Laboratory provide field inspections as a service and then contract this service as needed?

Below are the interview questions asked to John Aver of Aver Family Vineyards.

- What are the current methods you use for vineyard inspection?
- How often do you inspect the vineyard?
- Conceptually, what types of data would be useful to you in order to augment current inspection techniques and help determine vineyard health information?
- Would you be willing to buy and then train yourself on the system we develop, or would you prefer to have it available as a service provided by a company that you could then call in on a regular basis?
- In general, what is the yearly cycle of your vineyard?

APPENDIX C

Decision Matrices

Project:	UAVino													
System:	Docking Station													
Date:	11/11/2014													
	1	2	3	4	5	6	7	8	9	10	11	12	SUM	FACTOR
1 Weight	1	0.5	0	1	0	0.5	0	0	0	0	0.5		2.5	6
2 Size	0.5	1	0	1	0	0.5	0	0	0	0			2	4
3 Charging Time	1	1	1	1	0	1	0.5	0.5	0.5	1			6.5	15
4 Housing	0	0	0	0	0	0.5	0	0	0	0			0.5	1
5 Positioning Precision	1	1	1	1	1	1	1	0.5	1	1			8.5	19
6 Aesthetics	0.5	0.5	0	0.5	0	0	0	0	0	0			1.5	3
7 Reliability	1	1	0.5	1	0	1	0.5	0.5	0.5	1			6.5	14
8 Safety	1	1	0.5	1	0.5	1	0.5	0.5	0.5	1			7	16
9 Ease of Manufacturing	1	1	0.5	1	0	1	0.5	0.5	0.5	1			6.5	14
10 Setup Time	0.5	1	0	1	0	1	0	0	0	0			3.5	8
11	1	1	1	1	1	1	1	1	1	1			10	
12	1	1	1	1	1	1	1	1	1	1			11	
Fill in Purple squares above														45
Fill in upper triangle of the matrix														100
Working across each row, determine if the criterion in that row is more important (1), same importance (0.5) or less important (0) than the criterion in that column														

Figure C.1: Prioritizing matrix of system requirements.

Design Project =	UAVino				System=	Enclosure Method			
	TARGET	DESIGN IDEAS							
	or								
CRITERIA	FACTOR	1 = Baseline	Hinged Enclosure		Accordian		Overlapping Leaf		
Time - Design	15	15		5		5		30	
Time - Build	20	20		10		15		40	
Time - Test	3	3		1		2		5	
Time Score	10		10		3.89		5.83		18.89
Cost - Prototype	\$ 40.00	\$ 40.00		\$ 20.00		\$ 40.00		\$100.00	
Cost - Production	\$ 100.00	\$100.00		\$ 70.00		\$ 70.00		\$200.00	
Cost Score	10		10		6.00		8.50		22.50
Weight	6	3	18	1	6	1	6	2	12
Size	4	3	12	1	4	1	4	3	12
Charging Time	15	3	45	3	45	3	45	3	45
Housing	1	3	3	4	4	3	3	1	1
Positioning Precision	19	3	57	2	38	2	38	4	76
Aesthetics	3	3	9	2	6	2	6	5	15
Reliability	14	3	42	4	56	3	42	1	14
Safety	16	3	48	4	64	4	64	2	32
Ease of Manufacturing	14	3	42	2	28	2	28	1	14
Setup Time	8	3	24	3	24	3	24	3	24
TOTAL			300.0		285.1		265.7		223.6
RANK			1.0		2.0		3.0		4.0
% MAX			100.0%		95.0%		88.6%		74.5%
MAX			300.0						

Figure C.2: Ranking of enclosure design concepts using decision matrix.

Design Project =		UAVino		System=		Positioning Method	
	TARGET	DESIGN IDEAS					
	or						
CRITERIA	FACTOR	1 = Baseline	PE Positioning	Magnetic			
Time - Design	10	10	50	10			
Time - Build	30	30	60	15			
Time - Test	5	5	20	5			
Time Score	10	10	36.67	8.33			
Cost - Prototype	50	50	150	20			
Cost - Production	100	100	200	50			
Cost Score	10	10	25.00	4.50			
Weight	6	3	18	3	18	4	24
Size	4	3	12	4	16	4	16
Charging Time	15	3	45	3	45	3	45
Housing	1	3	3	5	5	4	4
Positioning Precision	19	3	57	2	38	1	19
Asthetics	3	3	9	4	12	4	12
Reliability	14	3	42	2	28	1	14
Safety	16	3	48	4	64	4	64
Ease of Manufacturing	14	3	42	1	14	4	56
Setup Time	8	3	24	4	32	4	32
	TOTAL		300.0		230.3		293.2
	RANK		1		3		2
	% MAX		100.0%		76.8%		97.7%
	MAX		300.0				

Figure C.3: Ranking of positioning method concepts using decision matrix.

Design Project = UAVino		System= Charging Method					
	TARGET	DESIGN IDEAS					
	or						
CRITERIA	FACTOR	1 = Baseline	Connector	Inductance			
Time - Design	20	20	20	30			
Time - Build	40	40	40	40			
Time - Test	15	15	30	30			
Time Score	10	10	13.33	15.00			
Cost - Prototype	\$ 100	\$ 100	\$ 200	\$ 150			
Cost - Production	\$ 150	\$ 150	\$ 200	\$ 150			
Cost Score	10	10	16.67	12.50			
Weight	6	3	18	2	12	4	24
Size	4	3	12	2	8	3	12
Charging Time	15	3	45	4	60	1	15
Housing	1	3	3	3	3	3	3
Positioning Precision	19	3	57	1	19	5	95
Asthetics	3	3	9	3	9	3	9
Reliability	14	3	42	4	56	3	42
Safety	16	3	48	4	64	4	64
Ease of Manufacturing	14	3	42	2	28	1	14
Setup Time	8	3	24	3	24	3	24
	TOTAL		300.0		273.0		294.5
	RANK		1.0		3.0		2.0
	% MAX		100.0%		91.0%		98.2%
	MAX	300.0					

Figure C.4: Ranking of charging method concepts using decision matrix.

APPENDIX D

Product Sketches

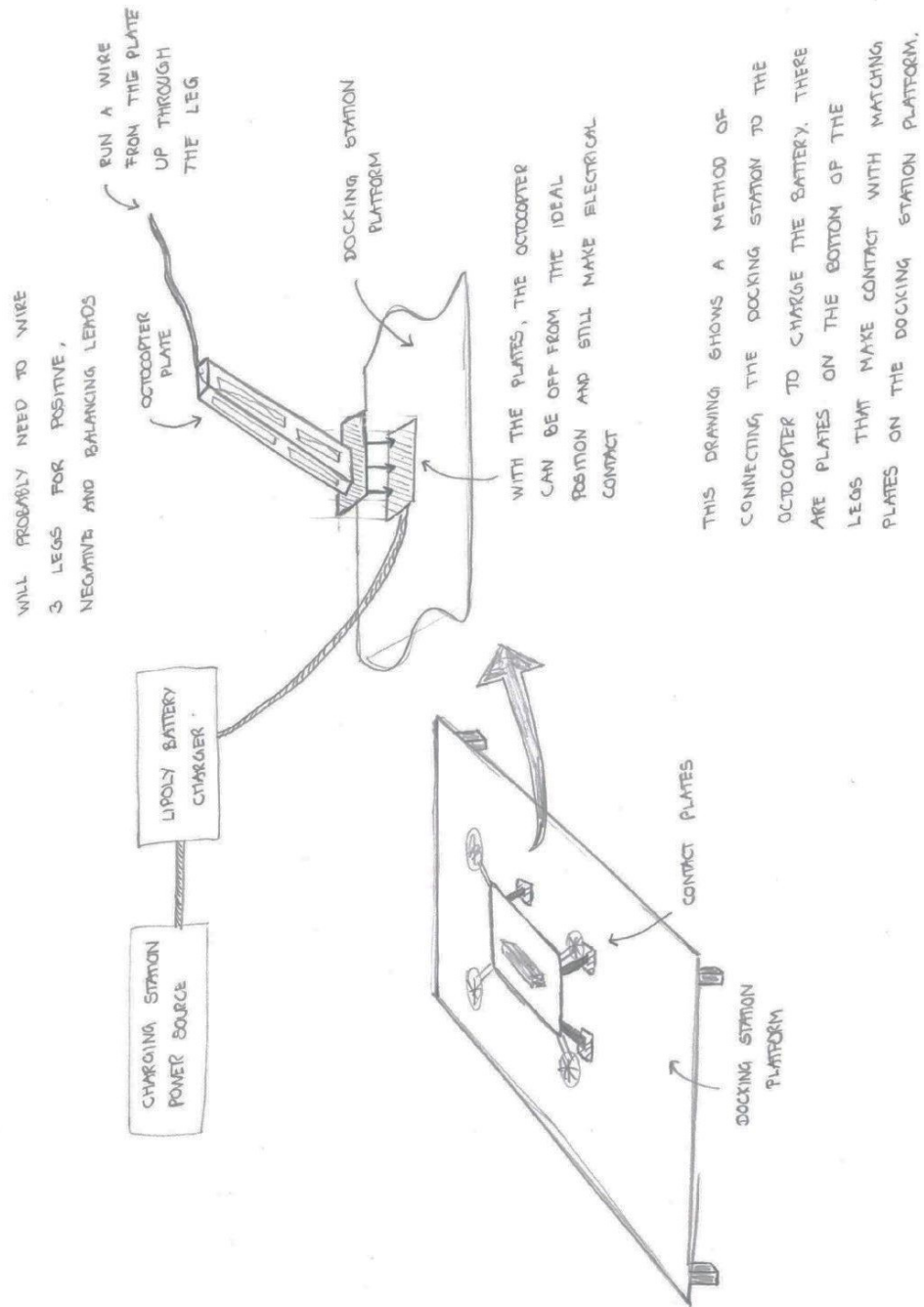


Figure D.1: Charging concept design using contact plates. Drawing by Phillip Coyle.

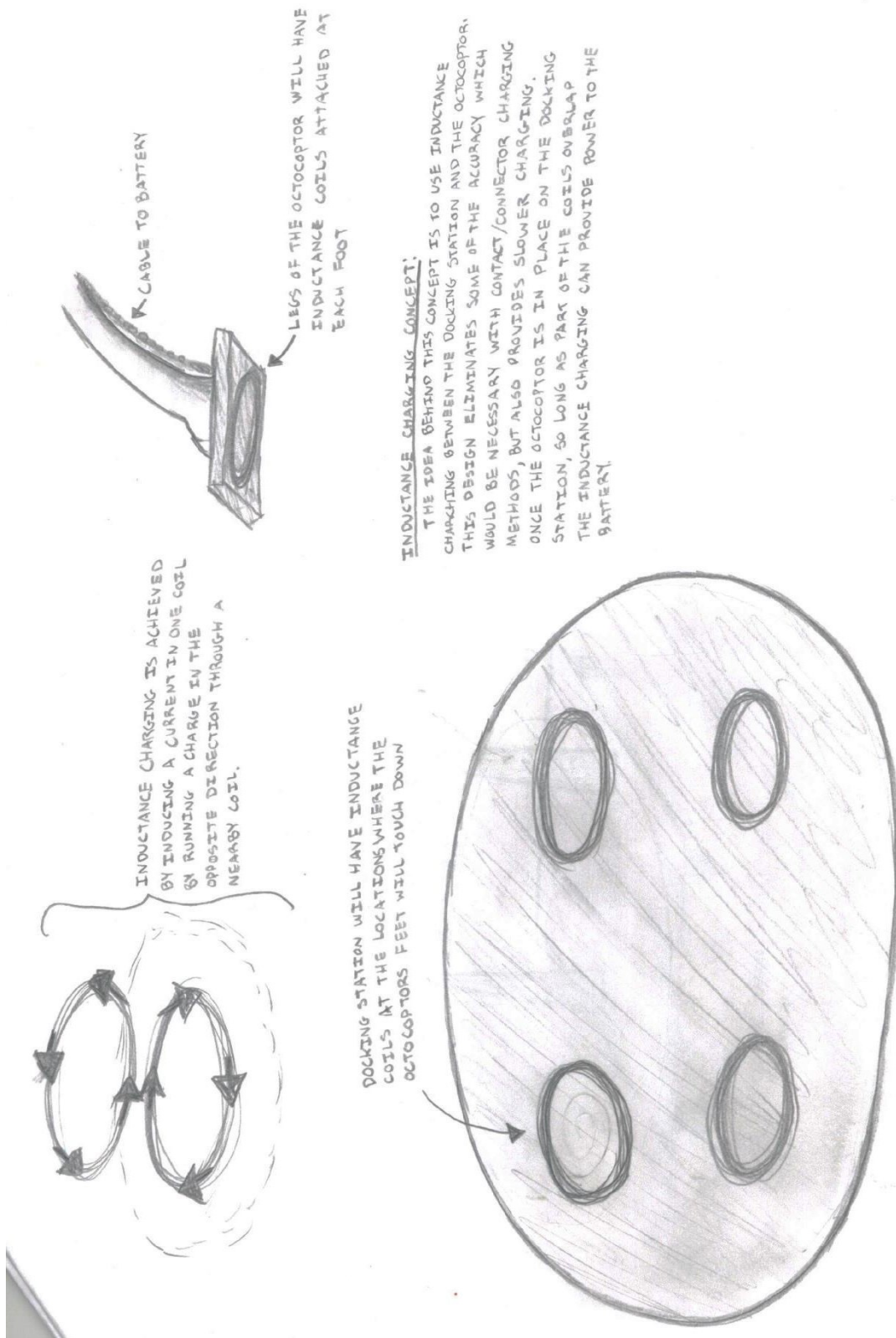
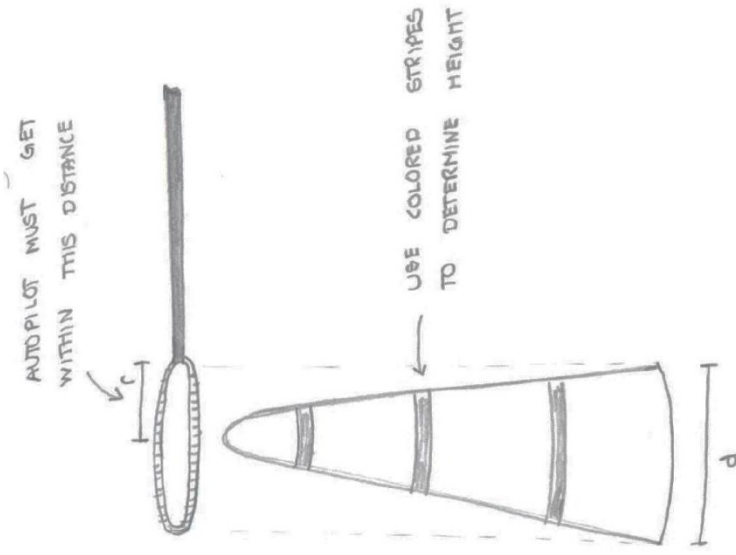
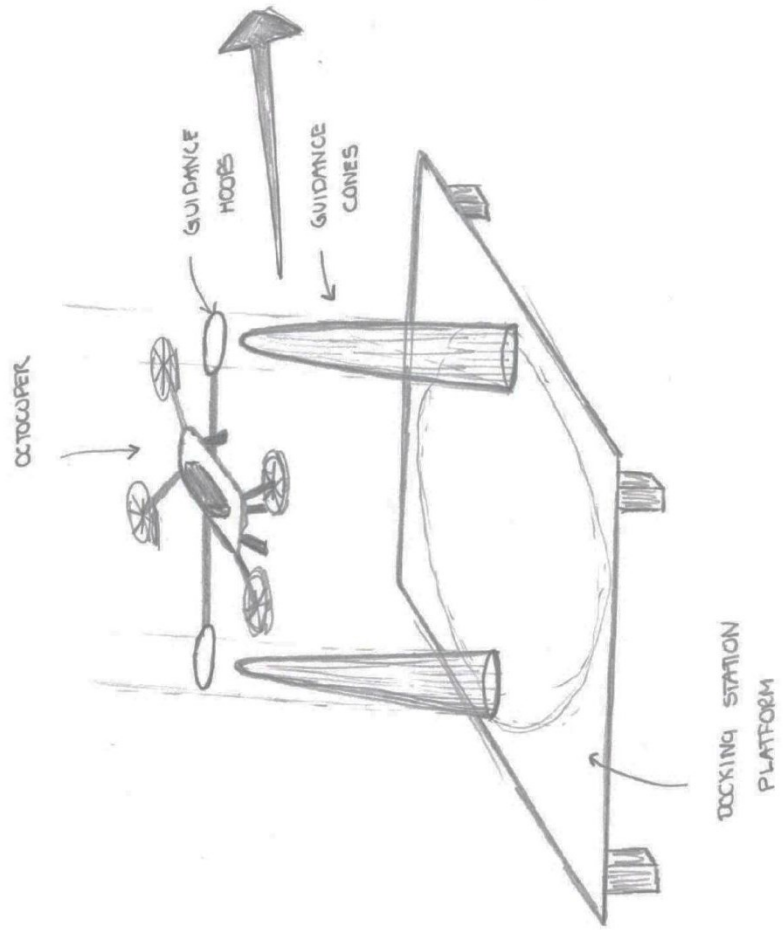


Figure D.2: Charging concept design using induction. Drawing by Megan Peekema.



THIS DRAWING SHOWS AN IDEA FOR THE ACTUAL DOCKING SYSTEM THAT IS TOTALLY PASSIVE. THE OCTOCOPTER HAS 2 HOOPS AND THE DOCKING STATION HAS 2 CONES WITH BASE DIAMETERS EQUAL TO THE HOOPS TO GUIDE THE COPTER TO THE RIGHT LOCATION.



114

Figure D.3: Docking concept using cones for guidance. Drawing by Phillip Coyle.

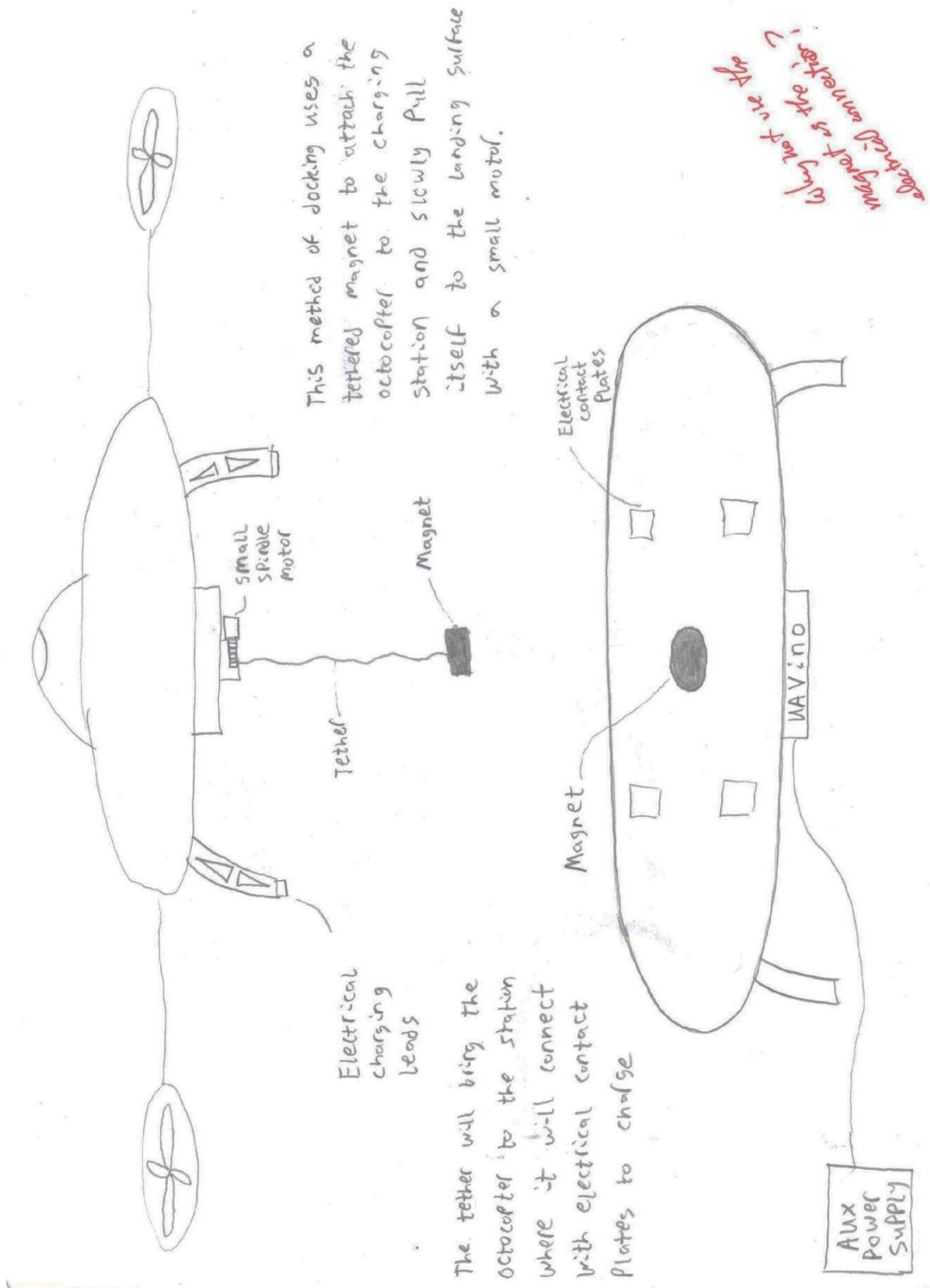
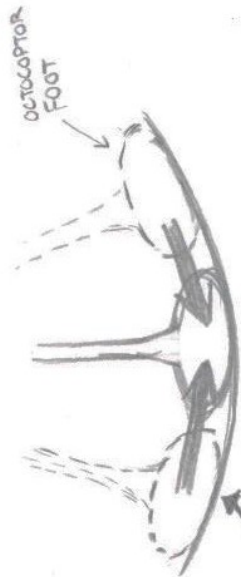


Figure D.4: Docking concept using magnet for guidance. Drawing by Nathan Carlson.



CURVED SURFACE OF THE DOCKING STATION WILL CAUSE THE FOOT OF THE OCTOCOPTOR TO ALWAYS SEEK THE POSITION OF LOWEST POTENTIAL ENERGY.

POTENTIAL ENERGY POSITIONING CONCEPT:

THE IDEA OF THIS CONCEPT IS TO USE A CURVED SURFACE TO POSITION THE OCTOCOPTOR USING A POTENTIAL ENERGY. THERE WILL BE A WELL ON THE TOP OF THE DOCKING STATION IN THE DESIRED FINAL POSITION OF THE OCTOCOPTOR. DEPENDING ON WHERE THE OCTOCOPTOR LANDS ON THE DOCKING STATION ITS FEET WILL TEND TO SLIDE DOWN TOWARDS THE DESIRED FINAL POSITION.

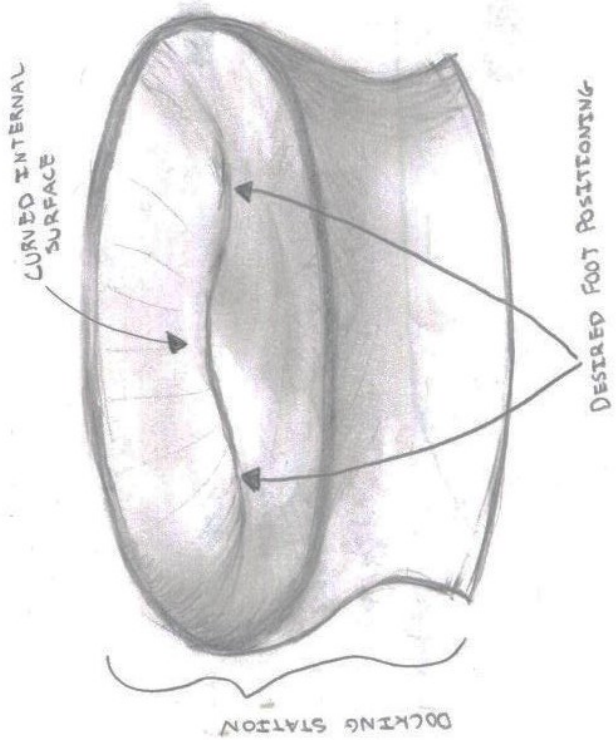
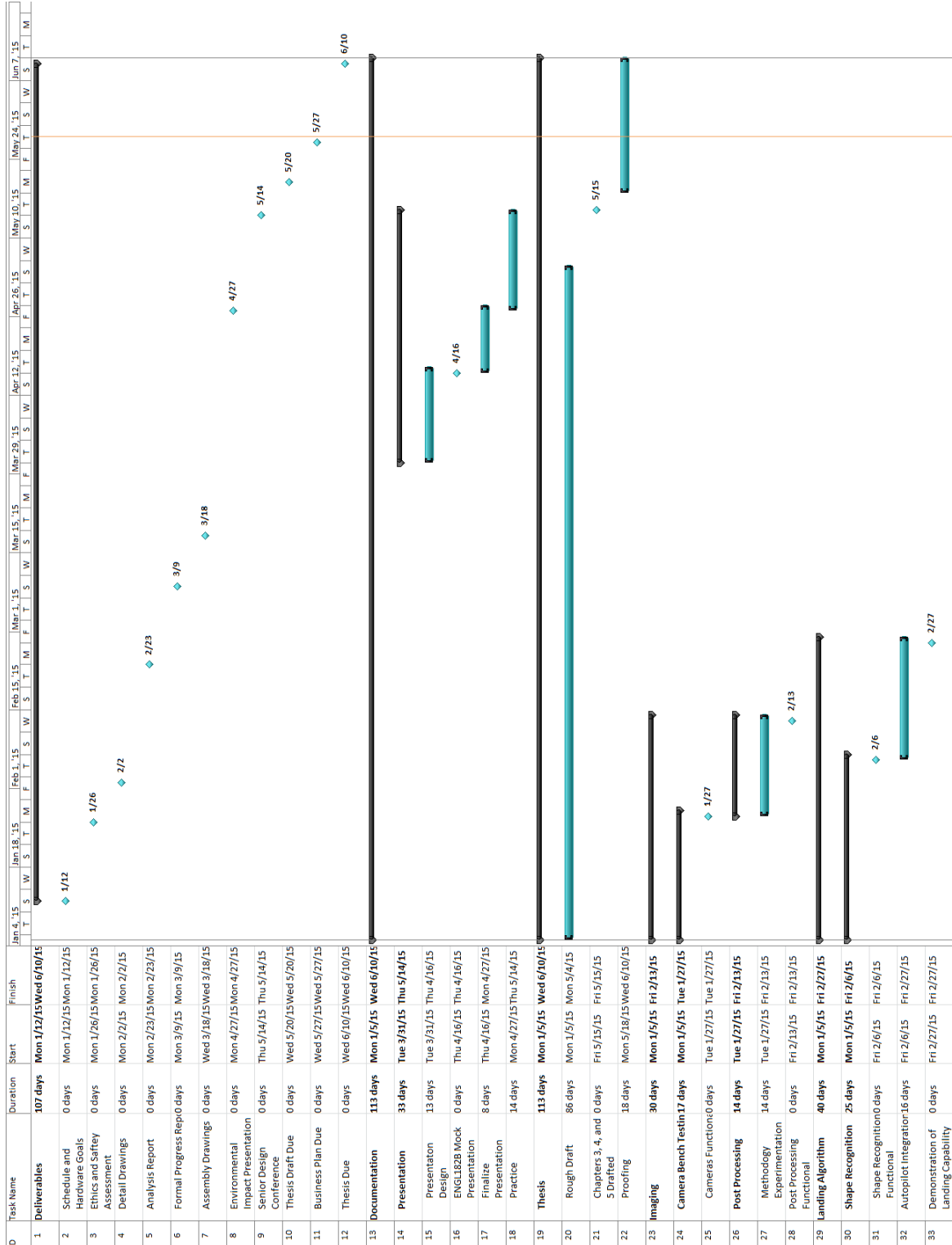
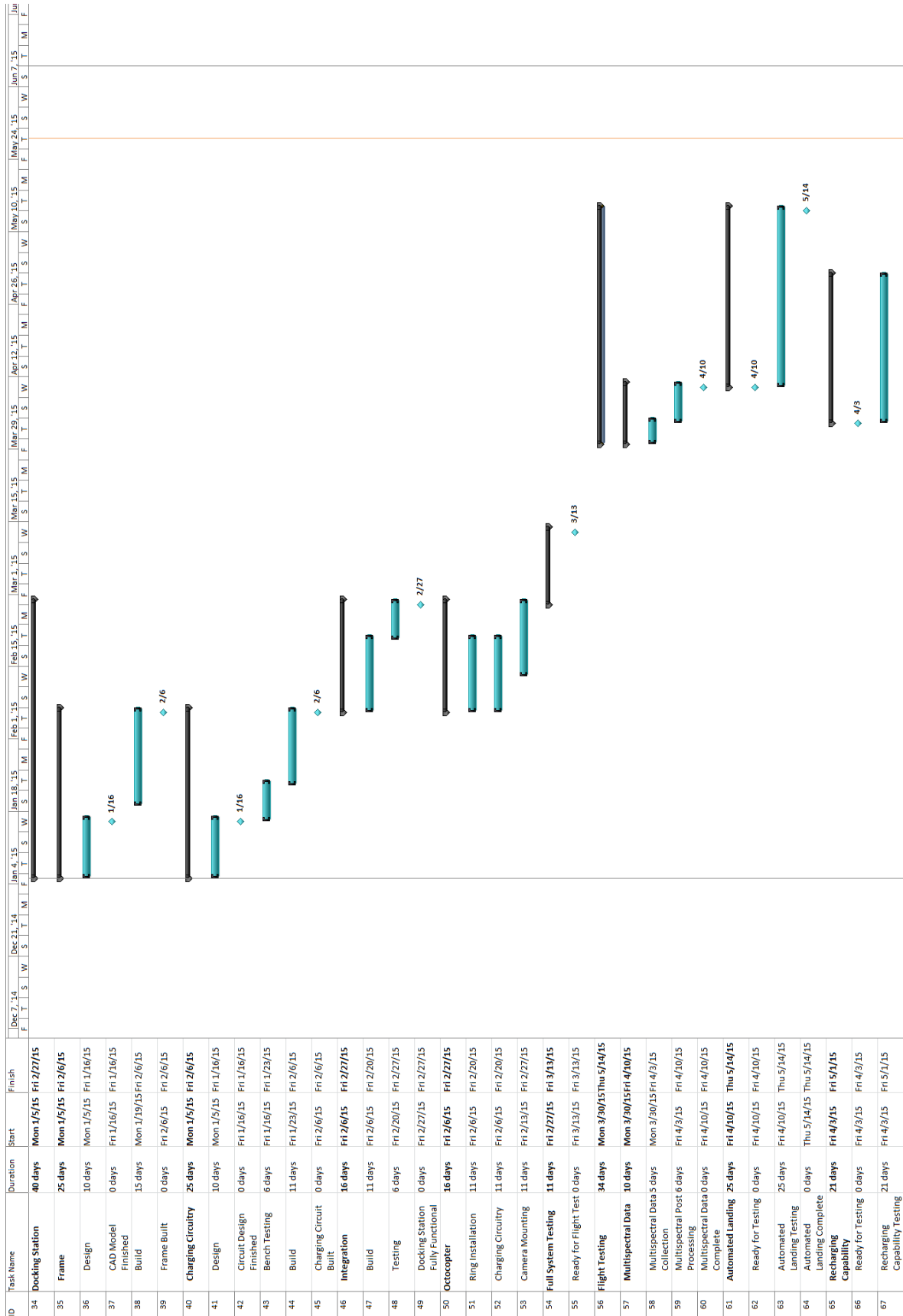


Figure D.5: Docking concept using gravity for guidance. Drawing by Megan Peekema.

APPENDIX E

Project Timeline





APPENDIX F

Project Budget

Expenses and donations for UAVino and any relevant notes regarding specific items are listed in the tables below.

Table F.1: UAVino Donations

Component	Cost	Notes
Multispectral Imaging Camera	\$2,500	Donated by Intel
Octocopter with Telemetry	\$1,500	Provided by SCU RSL
Edison Microcontroller	\$99	Intel Edison Donated by Intel
Santa Clara University School of Engineering	\$3000	
ASME Silicon Valley Section	\$750	
Total	\$7,849	

Table F.2: UAVino Expenses

Component	Cost	Notes
Multispectral Imaging Camera	\$2,500	Donated by Intel
Octocopter with Telemetry	\$1,500	Provided by SCU RSL
Octocopter Hardware and Electronics	\$450	Intel Edison Donated by Intel
Docking Station Hardware	\$300	
Docking Station Electronics	\$600	
Travel	\$350	
Bench testing Electronics	\$350	
Total	\$6,050	

APPENDIX G

System Inputs, Outputs, and Constraints

Table G.1: Inputs

Docking Station:	External power source used for charging
Octocopter:	GPS position data used by the vehicle's autopilot
	Flight plan coordinates from mission planner
	Rate of image capture
	Height from radar altimeter
	Battery Power
	Visual positioning data from onboard camera
Ground Control Station:	Coordinates for mission planner in order to control flight path of octocopter
	Image Data
	Battery Power

Table G.2: Outputs

Docking Station:	Power to lithium polymer battery on octocopter
	Passive positioning assistance for octocopter as it lands
Octocopter:	Thermal/multispectral images
Ground Control Station:	Normalized Vegetation index
	Water content data
	Flight path for octocopter generated by algorithm based on waypoint data

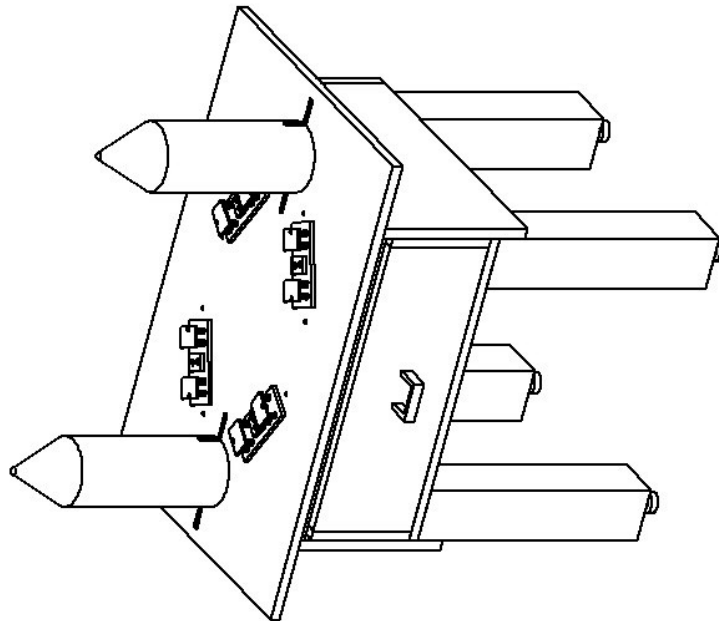
Table G.3: Constraints


Docking Station:	Charging rate of chosen charging method
	Cannot interfere with propellers of octocopter as it is landing
	Needs to be able to be carried by two people easily
	Contacts must be protected so station is electrically safe/isolated
	Must provide level surface on which the octocopter can land
Octocopter:	Weight of payload octocopter is able to carry
	Limited accuracy of positioning based on GPS coordinates
	Flight time based on battery life
	Memory for data storage of images
Ground Control Station:	Processing Power of system used

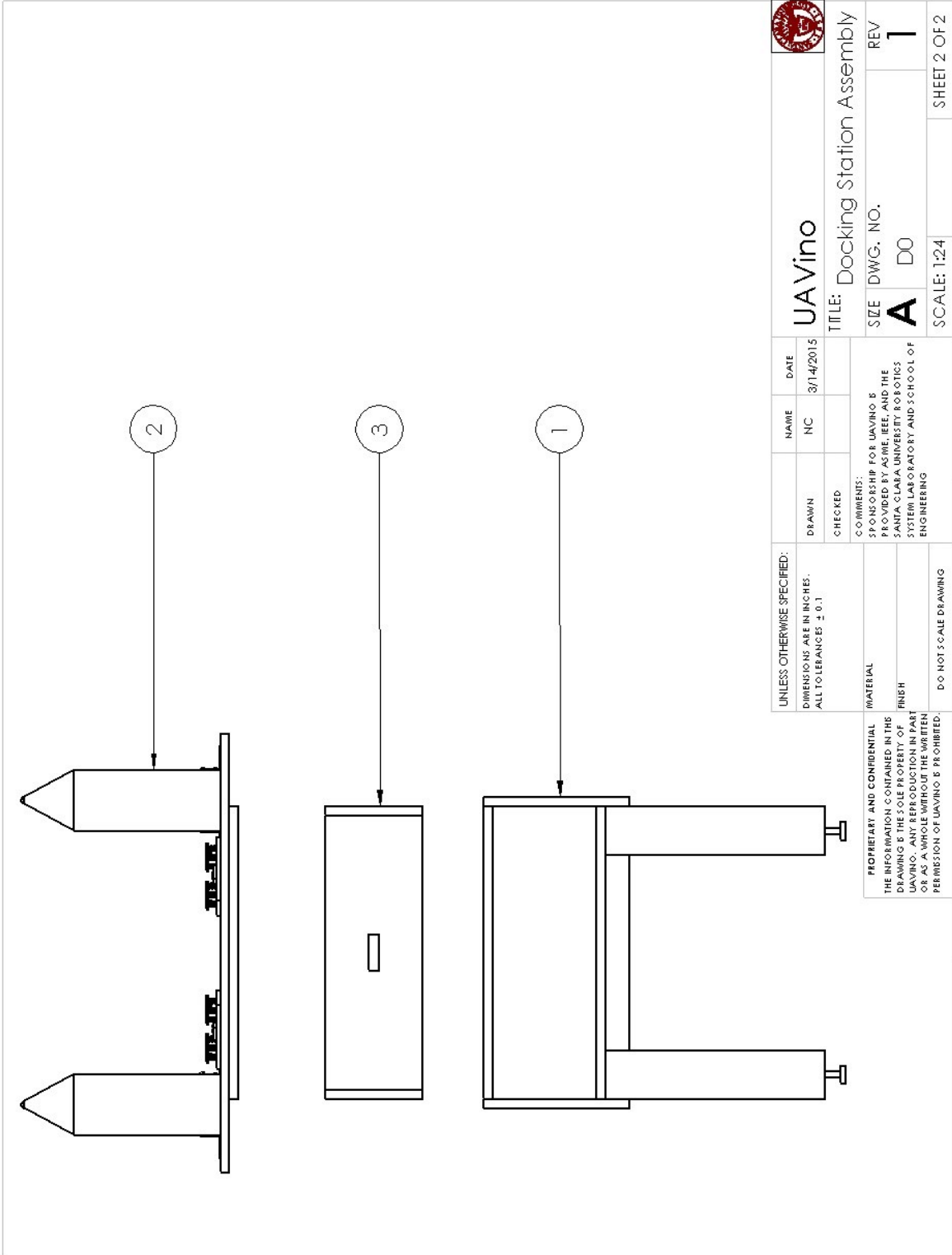
APPENDIX H

Detailed Assembly and Parts Drawings

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	DA3	Frame Assembly	1
2	DA1	Platform Assembly	1
3	DA2	Drawer Assembly	1



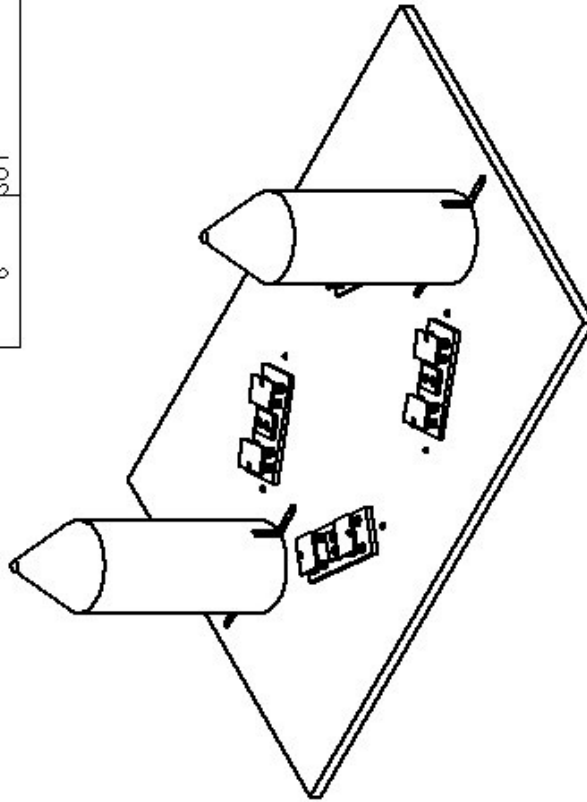
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES : 0.1		DRAWN	CHECKED	NAME NC	DATE 3/14/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF SANTA CLARA UNIVERSITY. NO PART OF THIS DRAWING IS TO BE REPRODUCED OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO. ® PROHIBITED.		COMMENTS: MEMBERSHIP FOR UAVINO. ® PROVIDED BY ASURE. BEE. AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		TITLE: UAVino Docking Station Assembly		
MATERIAL FINISH DO NOT SCALE DRAWING		SCALE: 1:10		SIZE DWG. NO. A D0		REV 1
				SHEET 1 OF 2		1




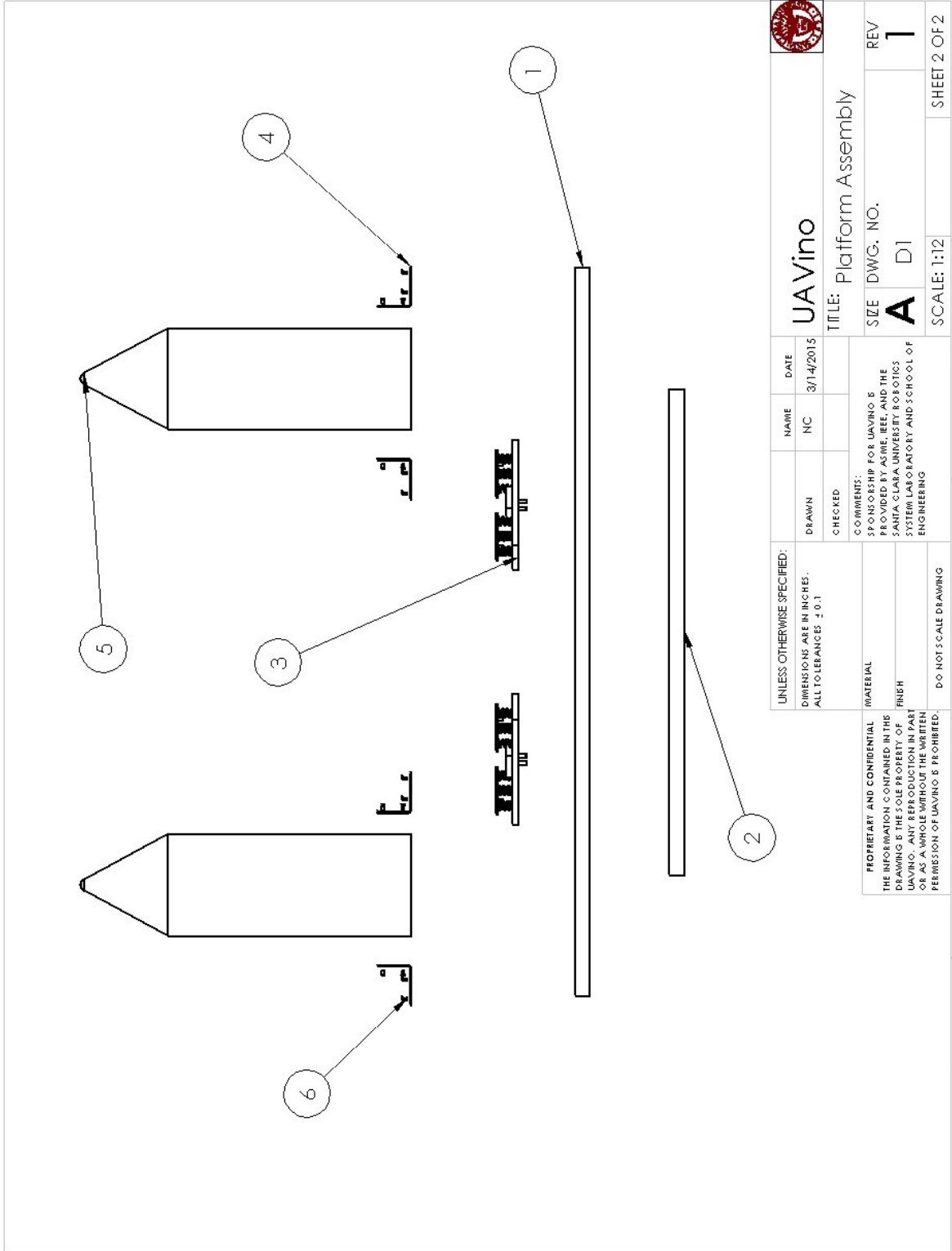
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	CHECKED	NAME NC	DATE 3/14/2015
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ACMIE, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING			
MATERIAL FINISH		TITLE: Docking Station Assembly SIZE DWG. NO. A DO SCALE: 1:24			
DO NOT SCALE DRAWING		3		2	


TITLE: Docking Station Assembly SIZE DWG. NO. A DO SCALE: 1:24		REV 1
SHEET 2 OF 2		1

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	D01	Platform	1
2	D02	Platform Alignment	2
3	DA4	Plate Stand Assembly	4
4	D17	Platform Bracket	4
5	DA5	Cone Assembly	2
6	S01	#4-48x0.25 Wood Screws	16

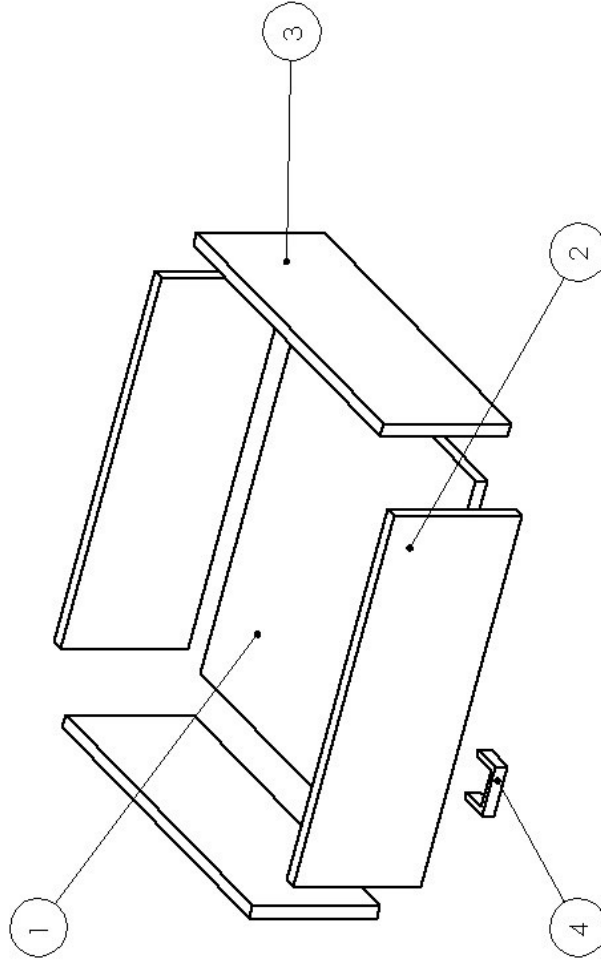


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN NC	NAME NC	DATE 8/14/2015	
PROPRIETARY AND CONFIDENTIAL MATERIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UNIV. OF NORTH CAROLINA AT CHARLOTTE OR AS A WHOLE, WITHOUT THE WRITTEN PERMISSION OF UNIV. OF NORTH CAROLINA AT CHARLOTTE.		CHECKED	U.A.V.I.N.O. TITLE: Platform Assembly		
COMMENTS: SPONSORSHIP FOR UNIV. OF NORTH CAROLINA AT CHARLOTTE PROVIDED BY ASME FEE AND THE SANJIA CHARLA UNIVERSITY POLYMER SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE DWG. NO. A D1		REV 1	SCALE: 1:12 SHEET 1 OF 2
DO NOT SCALE DRAWING		1 2 3 4 5		1	



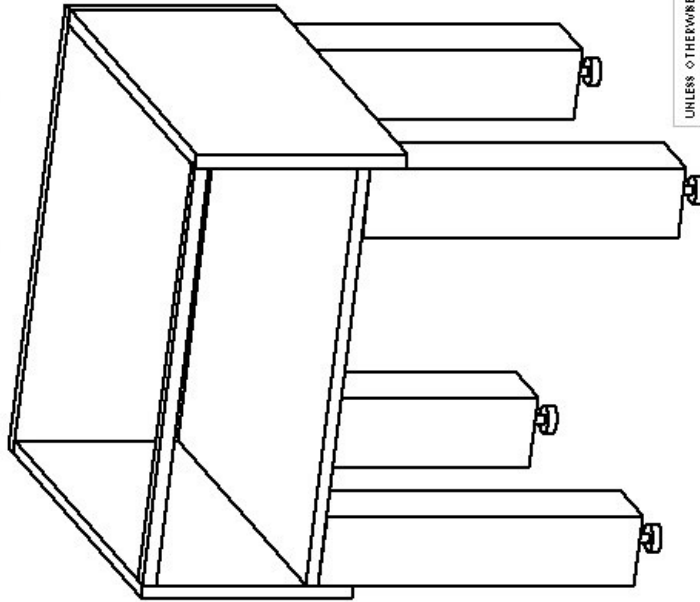
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME NC	DATE 3/14/2015		
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		CHECKED	COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Platform Assembly	
MATERIAL FINER		SCALE DWG. NO.		REV		
DO NOT SCALE DRAWING		A D1		1		
SCALE: 1:12		SCALE: 1:12		SHEET 2 OF 2		


ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	D03	Drawer Bottom	1
2	D04	Drawer Front	2
3	D05	Drawer Side	2
4	D06	Drawer Handle	1

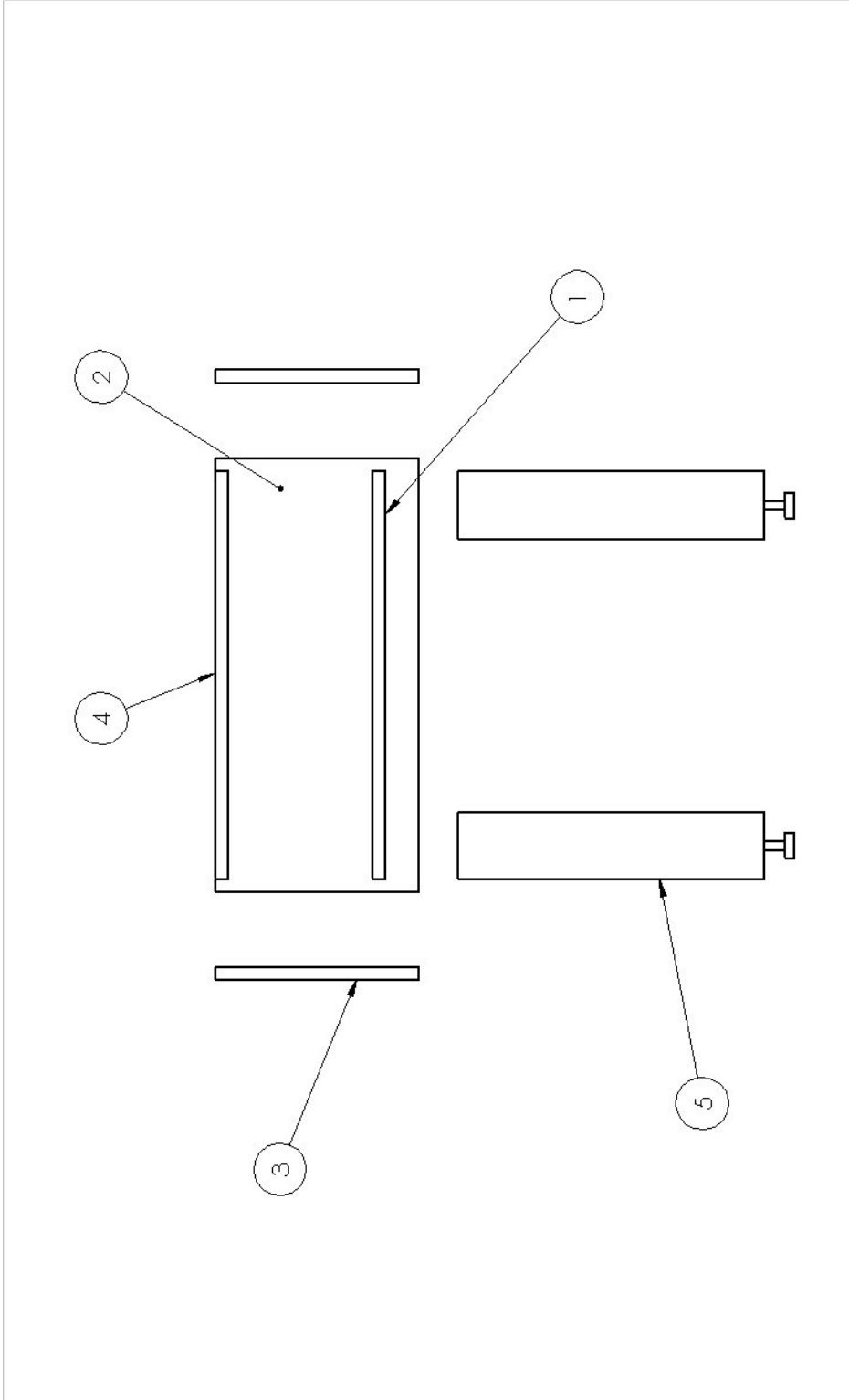



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 3/14/2015			
DRAWN		CHECKED	UAVino TITLE: Drawer Assembly			
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE A	DWG. NO. D2	REV 1
DO NOT SCALE DRAWING		SCALE: 1:8		SHEET 1 OF 1		

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	D07	Frame Bottom	1
2	D08	Frame Front	1
3	D09	Frame Side	2
4	D10	Frame Bar	1
5	D11	Leg Assembly	4

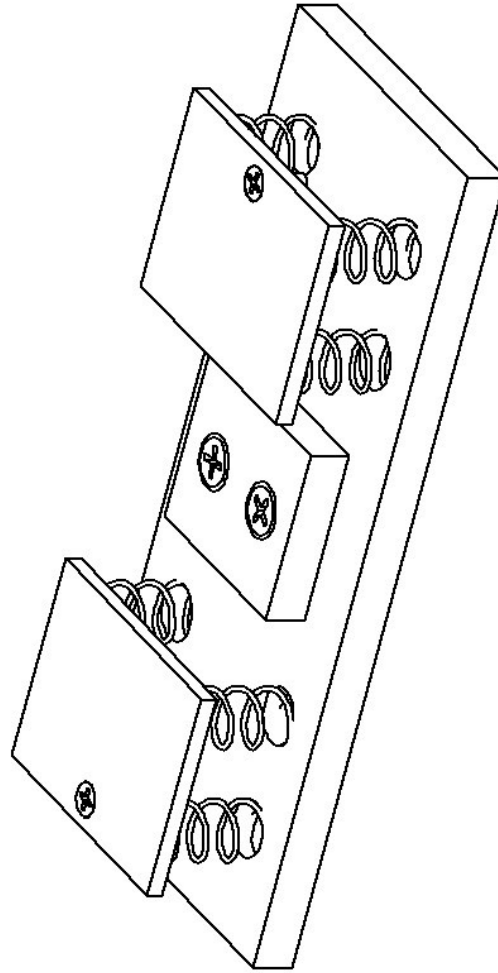



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	CHECKED	NAME NC	DATE 3/14/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Frame Assembly		
SIZE DWG. NO. A D3		SCALE: 1:12		REV 1		SHEET 1 OF 2

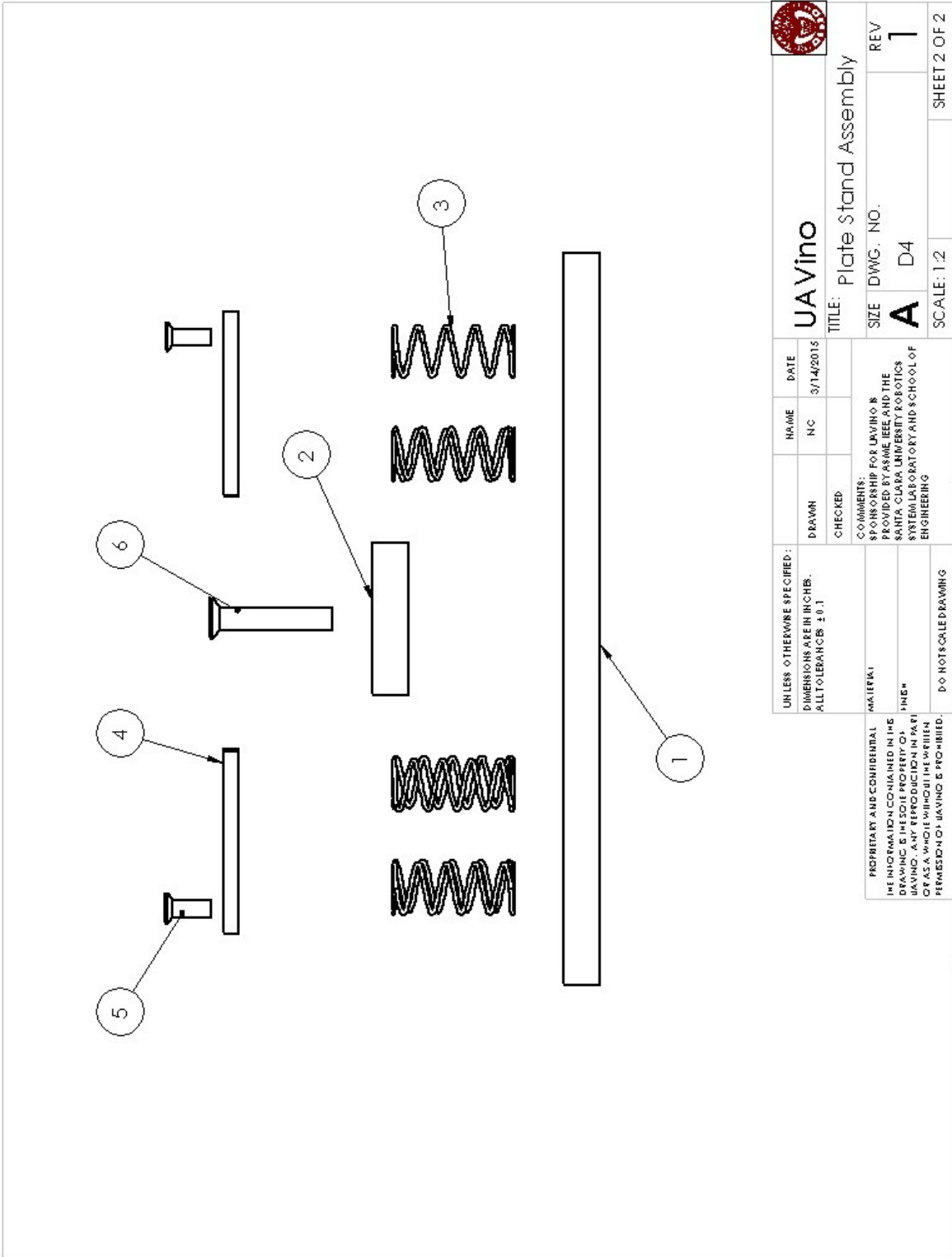


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME NC	DATE 3/14/2015		
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		CHECKED	UAVino TITLE: Frame Assembly			
MATERIAL FINISH		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING			SIZE DWG. NO. A D3	REV 1
DO NOT SCALE DRAWING		SCALE: 1:12		SHEET 2 OF 2		

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	D18	Plate Stand Base	1
2	D19	Plate Stand Top	1
3	D20	Spring	8
4	D21	Copper Plate	2
5	S02	#6-40x0.375 Machine Screws	2
6	S03	#10-24x1 Wood Screws	2



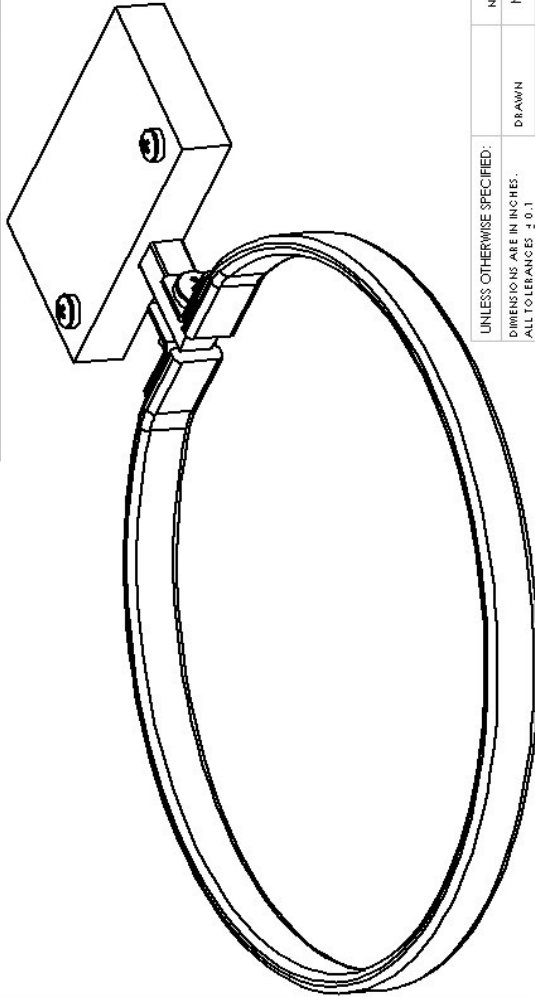
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES: ± 0.1		NAME NC	DATE 3/14/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		
MATERIAL FINISH		DRAWN CHECKED		TITLE: Plate Stand Assembly SIZE DWG. NO. A D4 SCALE: 1:2
DO NOT SCALE DRAWING		REV 1		SHEET 1 OF 2



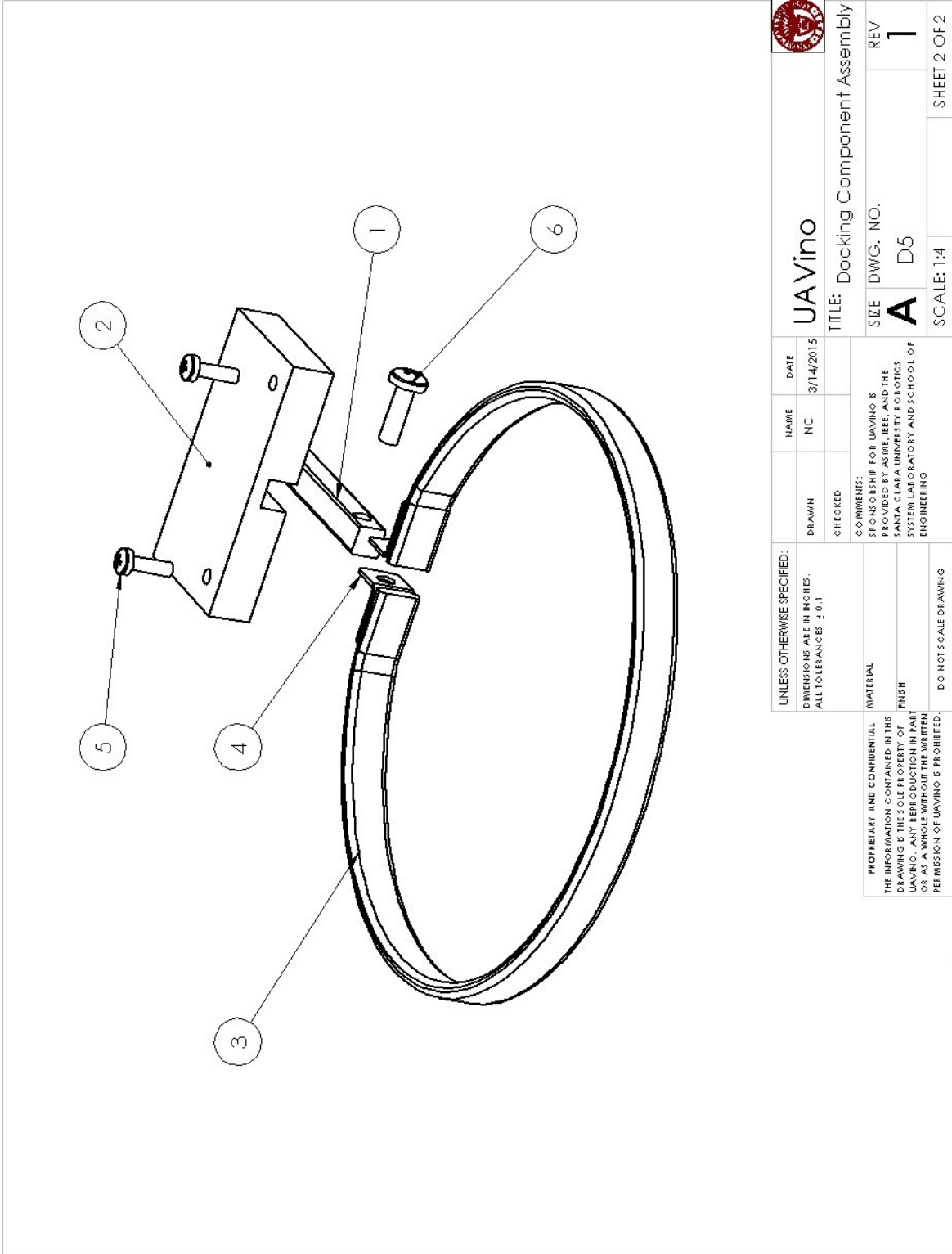
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 3/14/2015		UAVino TITLE: Plate Stand Assembly	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		DRAWN CHECKED	COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE A	DWG. NO. D4
DO NOT SCALE DRAWING		3		SCALE: 1:2		SHEET 2 OF 2

5 | 1 | 2 | 3 | 4 | 5

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	P02	Carbon Fiber Rod	1
2	P03	Ring Mounting Bracket	1
3	P01	Bamboo Hoop	1
4	P04	L-Bracket	2
5	B18.6.7M - M3 x 0.5 x 10 Type I Cross Recessed PHMS -- 10N		2
6	B18.6.7M - M4 x 0.7 x 13 Type I Cross Recessed PHMS -- 13N		1

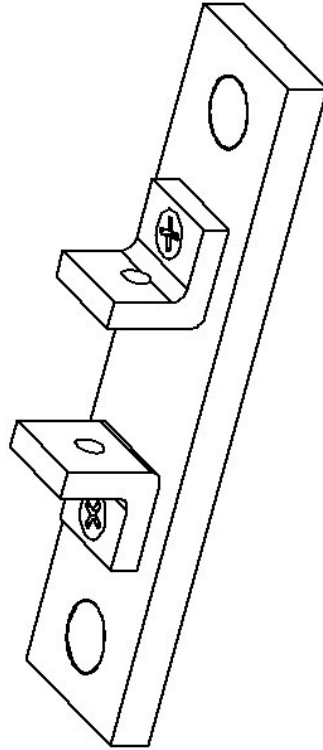



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES ALL TOLERANCES ± 0.1		DRAWN	NAME NC	DATE 3/14/2015		
MATERIAL FINISH		CHECKED	UA VINO TITLE: Docking Component Assembly			
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UA VINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UA VINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UA VINO IS PROVIDED BY ASME, IEEE AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE A	DWG. NO. D5	REV 1
DO NOT SCALE DRAWING		SCALE: 1:1		SHEET 1 OF 2		

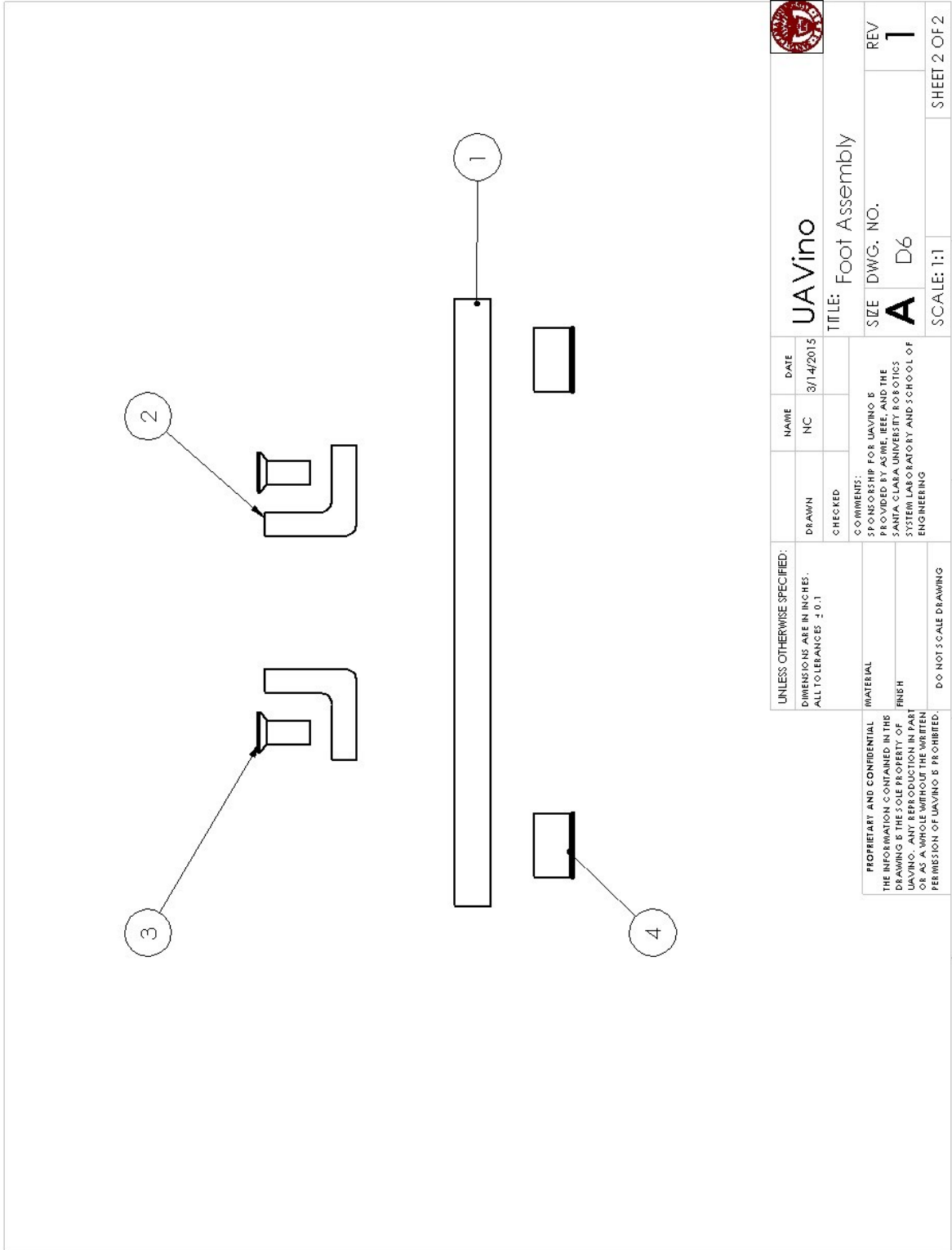


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 3/14/2015	 UA VINO	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UA VINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UA VINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UA VINO IS PROVIDED BY ASIME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Docking Component Assembly	
MATERIAL FINISH DO NOT SCALE DRAWING		SIZE DWG. NO. A D5		REV 1	
		SCALE: 1:4		SHEET 2 OF 2	

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	F01	Foot Base	1
2	F02	Foot Bracket	2
3	S06	# 10-24x0.4375 Wood Screw	2
4	F03	Copper Plug	2

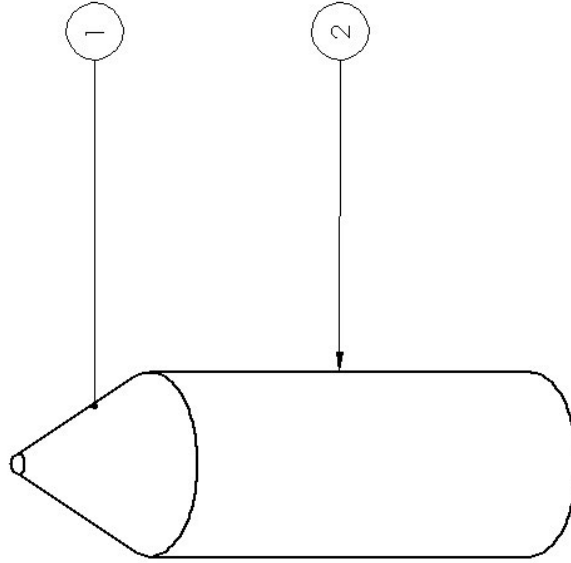



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 3/14/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		DRAWN CHECKED		
MATERIAL FINISH		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		REV 1
DO NOT SCALE DRAWING		3		SHEET 1 OF 2
5		2		1



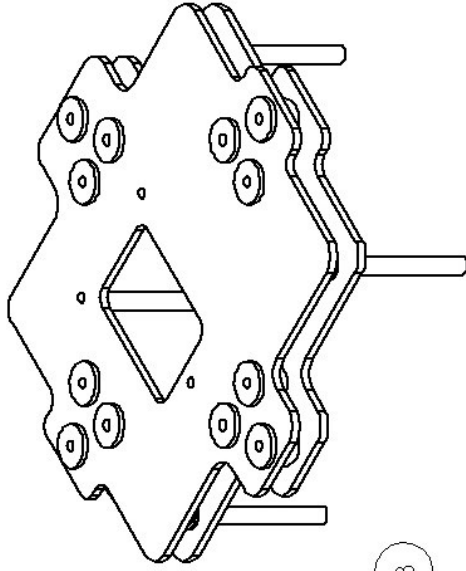
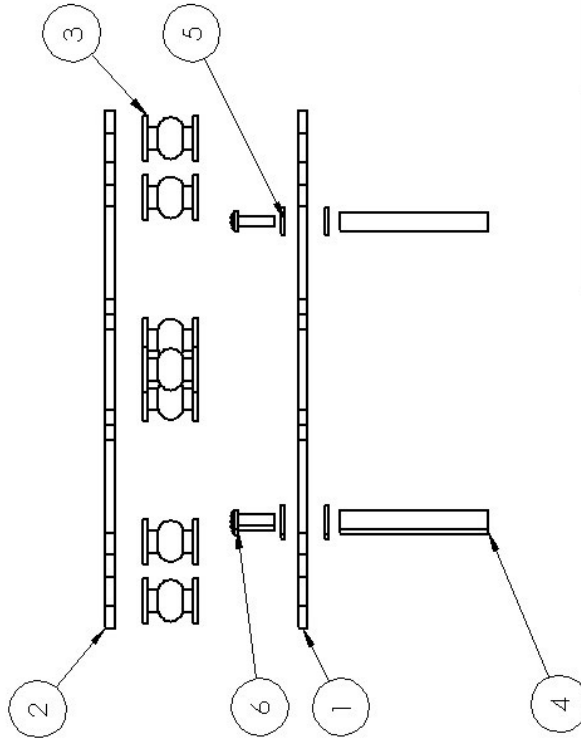
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME NC	DATE 3/14/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		CHECKED	COMMENTS: SPONSORSHIP FOR UAVINO BY PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		
MATERIAL FINER		DO NOT SCALE DRAWING		SIZE DWG. NO. A D6	REV 1
				SCALE: 1:1	SHEET 2 OF 2


ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	C02	Cone	1
2	C01	Vertical Cylinder	1

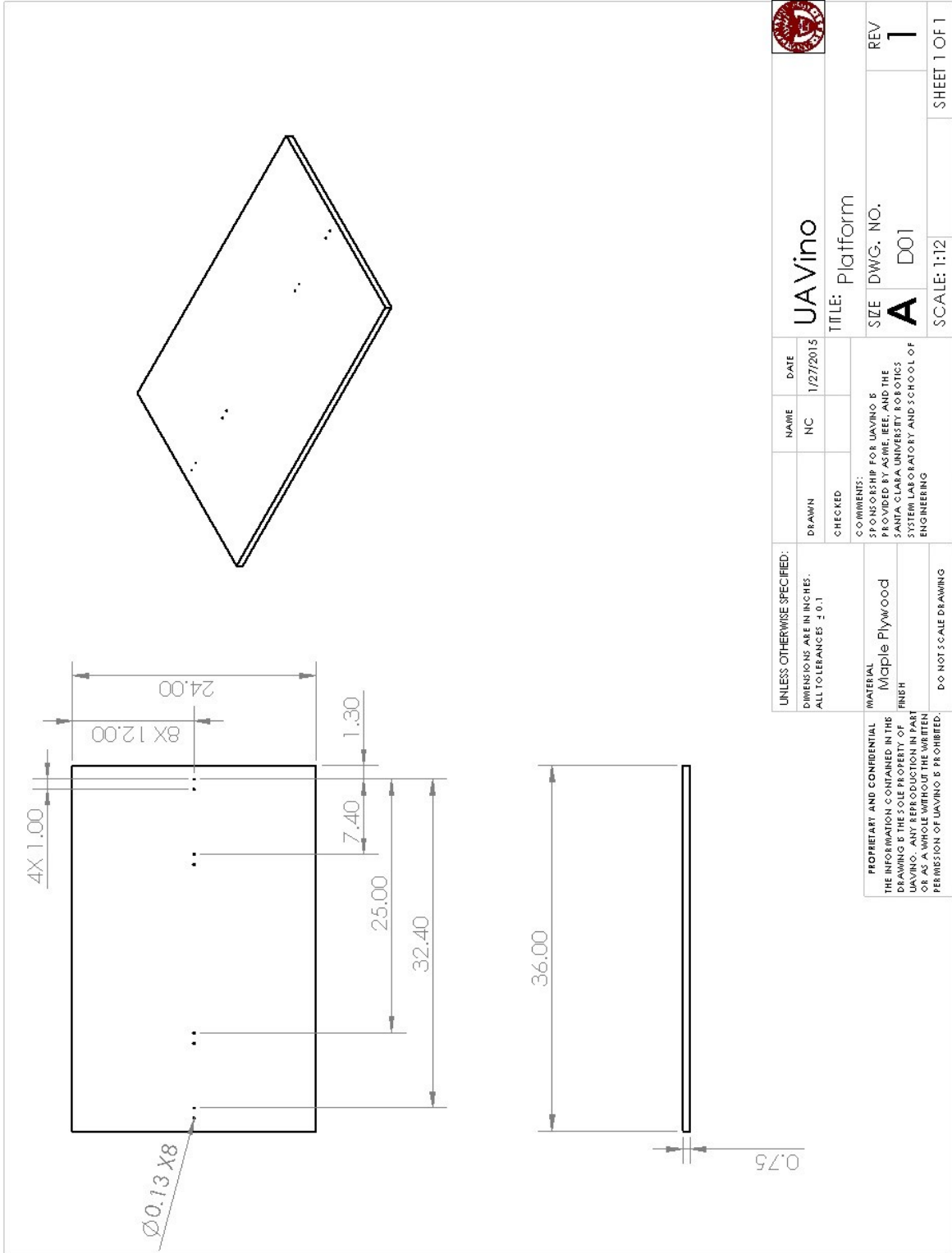


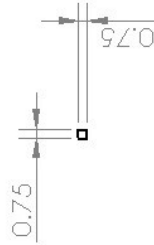
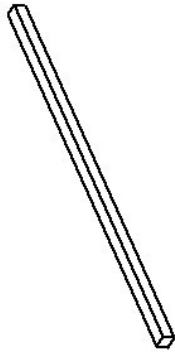
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME	DATE	
		CHECKED	NC	3/14/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ADRIE, LLC, AND THE SYSTEMS LABORATORY OF THE SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Cone Assembly	
DO NOT SCALE DRAWING		SCALE: 1:4		REV 1	
3		2		SHEET 1 OF 1	
5		1			


ITEM NO.	PART NUMBER	QTY.
1	M01	1
2	M02	1
3	Vibration Mount	12
4	Standoff	4
5	Preferred Narrow FW 0.138	8
6	CR-PHMS 0.138- 32x0.5x0.5-N	4

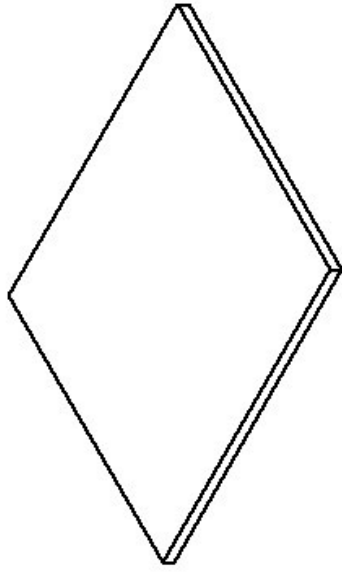
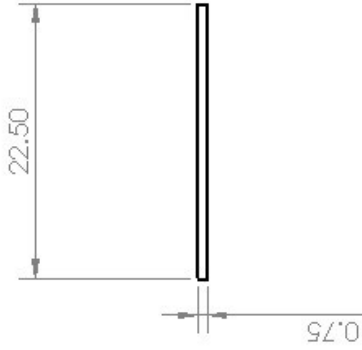
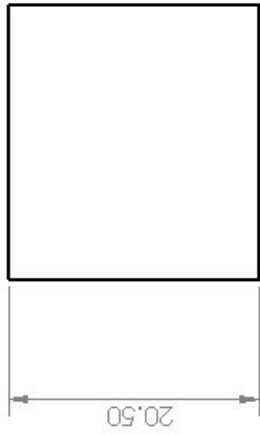


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME MP	DATE 3/11/2015	
PROPERTY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		DRAWN CHECKED		
MATERIAL FINISH		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE DWG. NO. A MAT REV
DO NOT SCALE DRAWING		SCALE: 1:2		SHEET 1 OF 1

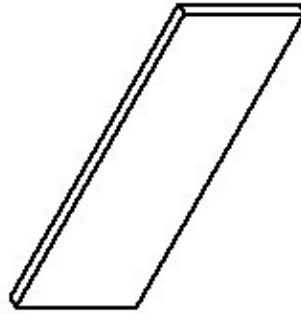
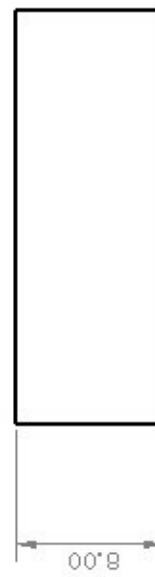
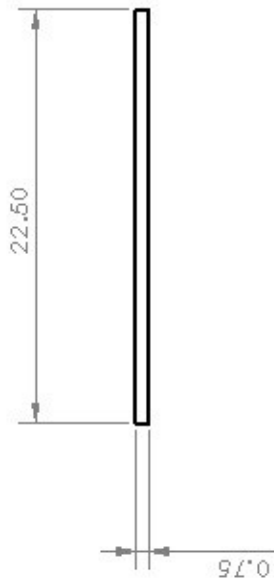




UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 1/27/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		DRAWN CHECKED		
MATERIAL Maple Plywood FINER		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		REV 1
DO NOT SCALE DRAWING		3		2
5		1		1



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN		NAME	DATE	 UAVINO
MATERIAL: Multiple Plywood 1/8"		CHECKED		HC	1/27/2018	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPECIFICATIONS FOR UAVINO IS PROVIDED BY UAVINO. SEE AND LINE DRAWING FOR DIMENSIONS. CONTACT SYSTEMS MANUFACTURING AND SCHEMATIC ENGINEERING		SIZE DWG. NO. A D03		REV
DO NOT SCALE DRAWING		SCALE: 1:8		SHEET 1 OF 1		1



UAVino

TITLE: Drawer Front

SIZE DWG. NO. REV

A D04 1

SCALE: 1:4 SHEET 1 OF 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES.
ALL TOLERANCES ± 0.1

DRAWN: _____ DATE: _____
CHECKED: _____

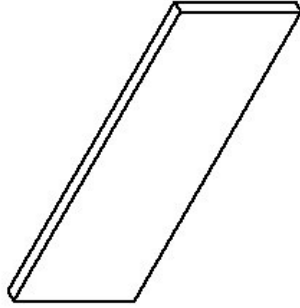
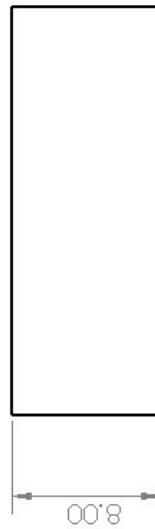
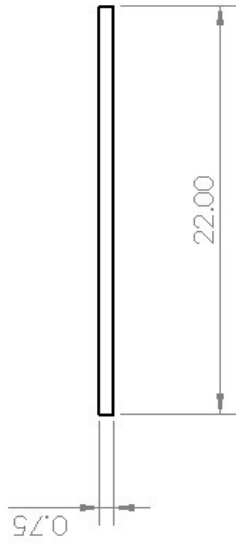
COMMENTS:
SPONSORSHIP FOR UAVINO IS
PROVIDED BY ASARE BEE AND THE
SANTA CLARA UNIVERSITY ROTARIES
SOCIETY FOR ROTARCTRY AND SCHOOL OF
ENGINEERING

MATERIAL:
Maple Plywood

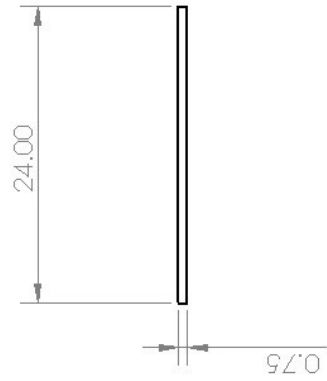
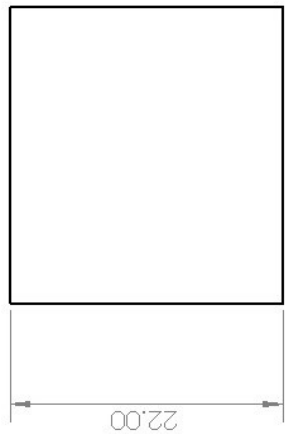
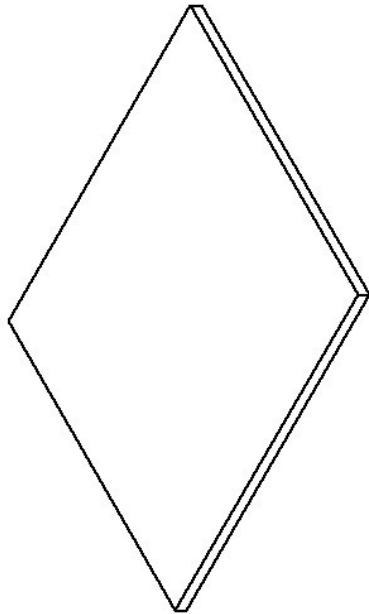
PROPERTY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS
DRAWING IS THE SOLE PROPERTY OF
UAVINO. ANY REPRODUCTION OR PART
THEREOF WITHOUT APPROVAL OF THE
PERMISSION OF UAVINO IS PROHIBITED.


DO NOT SCALE DRAWING

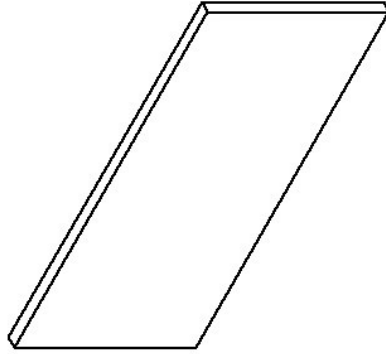
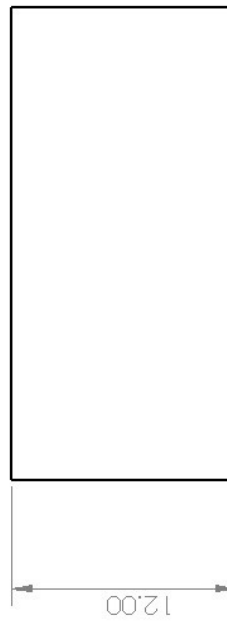
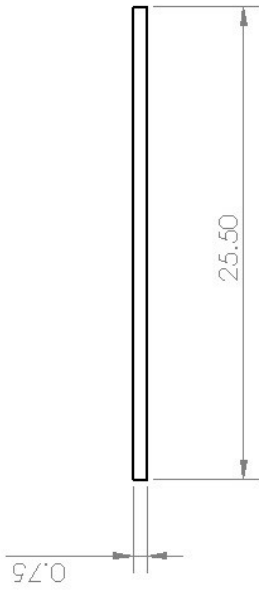
1 2 3 4 5




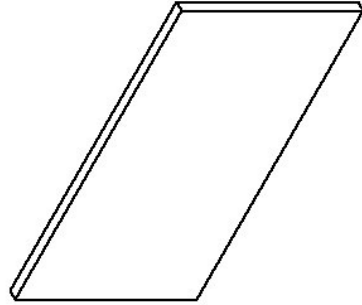
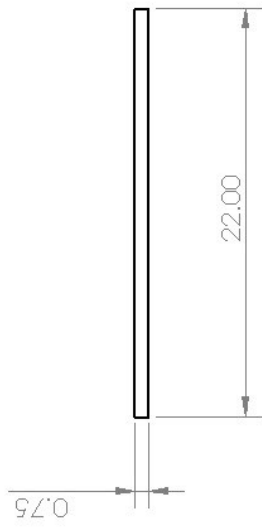
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 1/27/2018	
MATERIAL Maple Plywood FINISH		DRAWN CHECKED		
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SAHFA, CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Drawer Side SIZE DWG. No. A D05
DO NOT SCALE DRAWING		SCALE: 1:4		REV 1 SHEET 1 OF 1




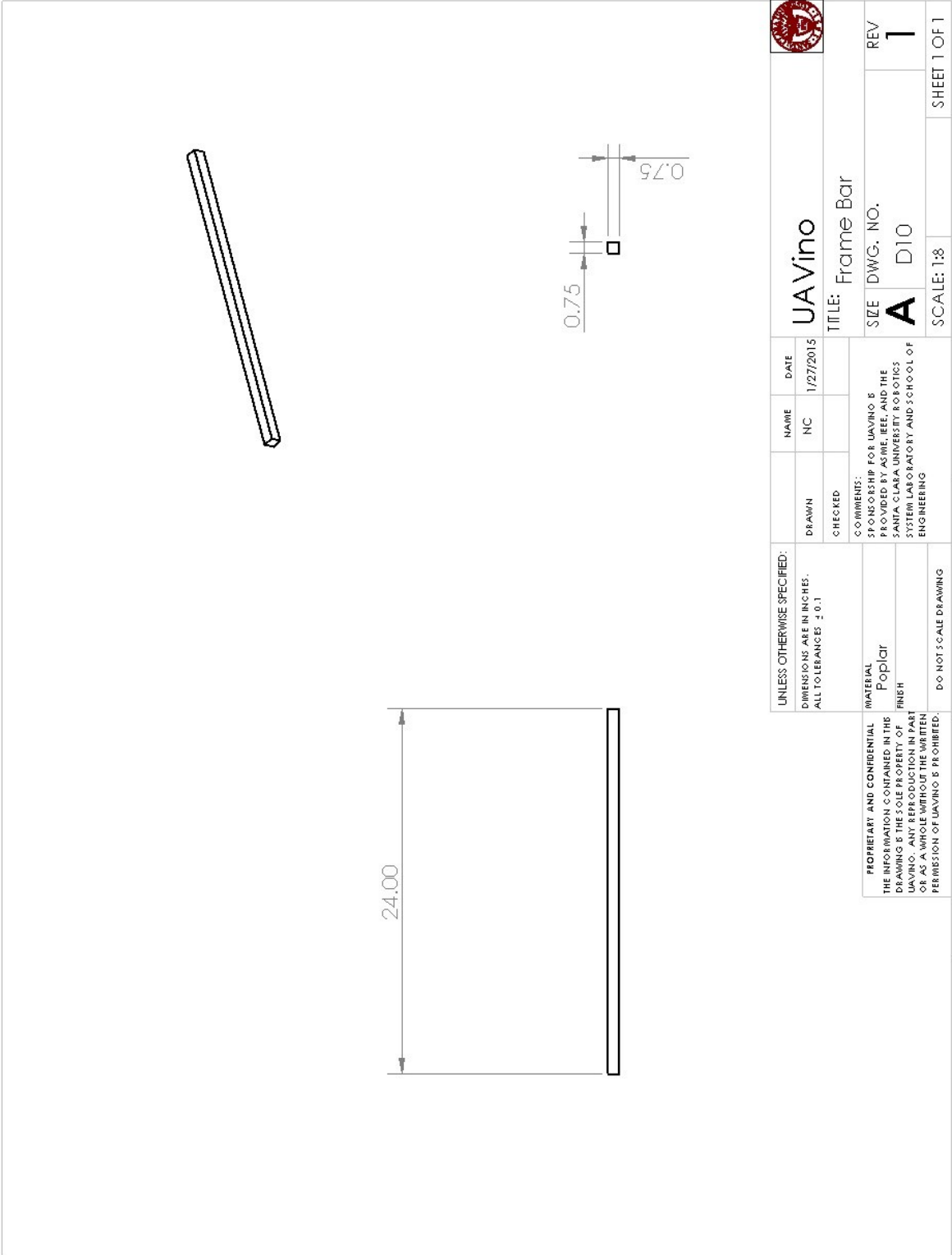
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 1/27/2015	
MATERIAL Maple Plywood		DRAWN	CHECKED	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.	COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING	SCALE: 1:8		SIZE DWG. NO. A D07 REV 1
DO NOT SCALE DRAWING			SHEET 1 OF 1	



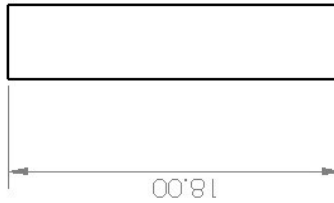
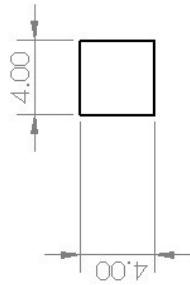
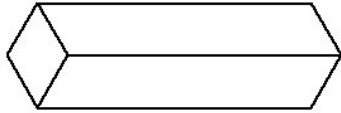
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME	DATE	
CHECKED	NC	NC	1/27/2015	UAVino	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Frame Front	
MATERIAL Maple Plywood		SIZE		DWG. NO.	REV
DO NOT SCALE DRAWING		A		D08	1
		SCALE: 1:8		SHEET 1 OF 1	




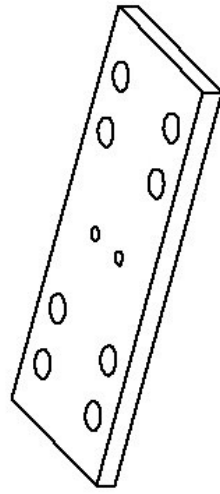
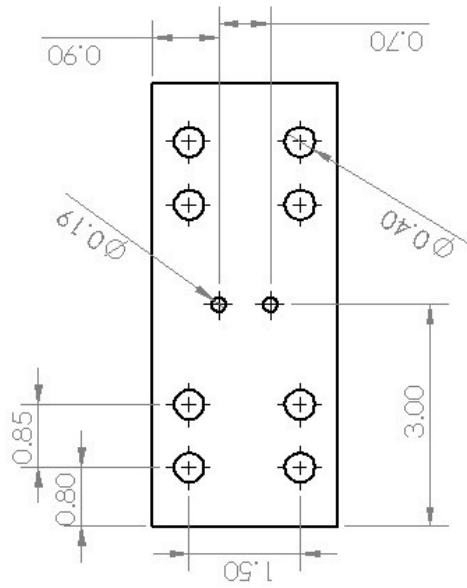
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC		DATE 1/27/2015	UAVino		
MATERIAL Maple Plywood		DRAWN		TITLE: Frame Side		SIZE DWG. NO.	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		CHECKED		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		SCALE: 1:4	REV 1
DO NOT SCALE DRAWING		SCALE: 1:4		SCALE: 1:4		SHEET 1 OF 1	




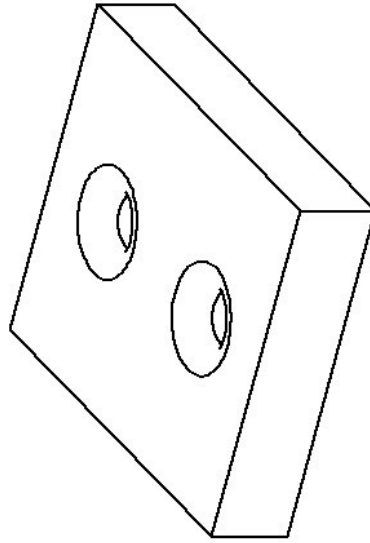
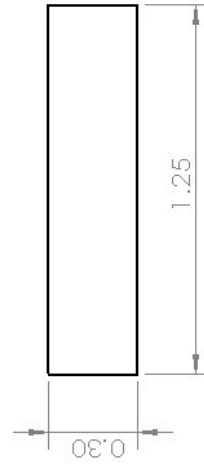
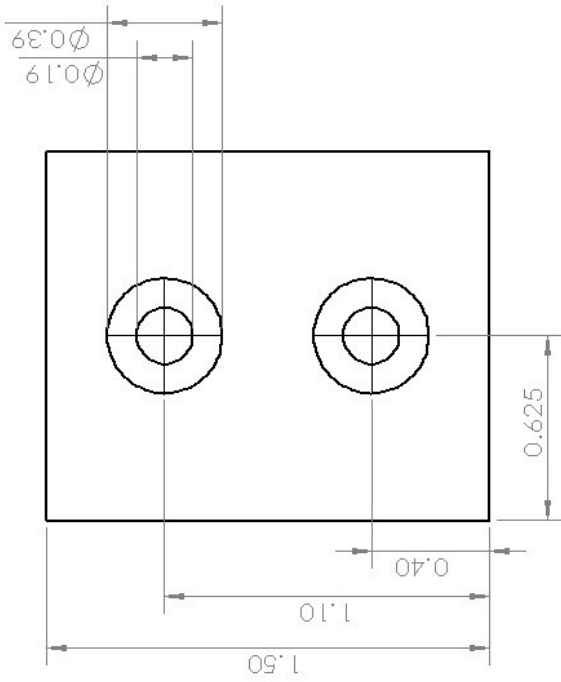
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 1/27/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UA VINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UA VINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UA VINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		
MATERIAL Poplar FINISH		TITLE: Frame Bar		REV
DO NOT SCALE DRAWING		SCALE: 1:8		1
5		3		2
1		1		SHEET 1 OF 1




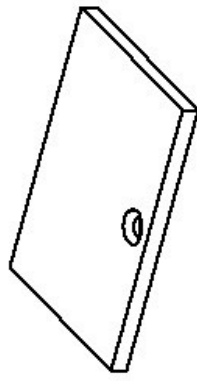
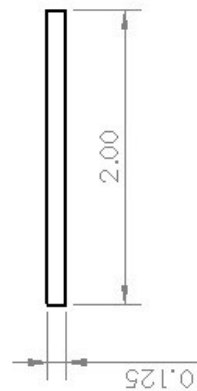
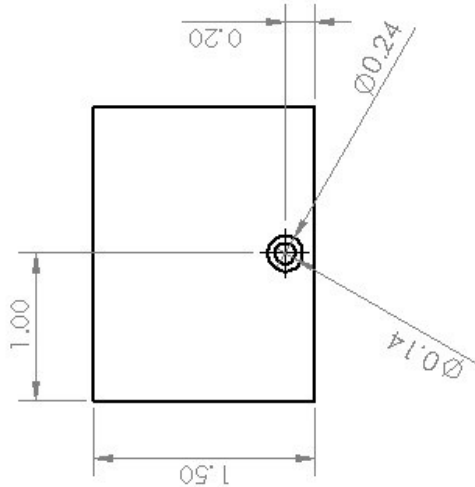
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 1/27/2015	UAVino		
DRAWN		CHECKED	TITLE: Leg		REV	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASIRE, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		SIZE A	DWG. NO. D11	1
DO NOT SCALE DRAWING		SCALE: 1:8		SHEET 1 OF 1		




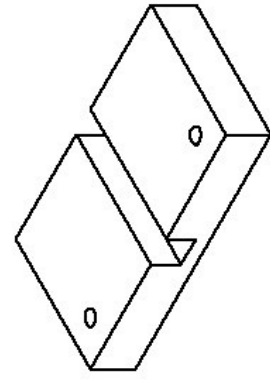
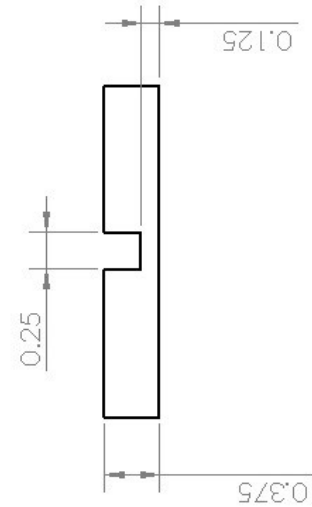
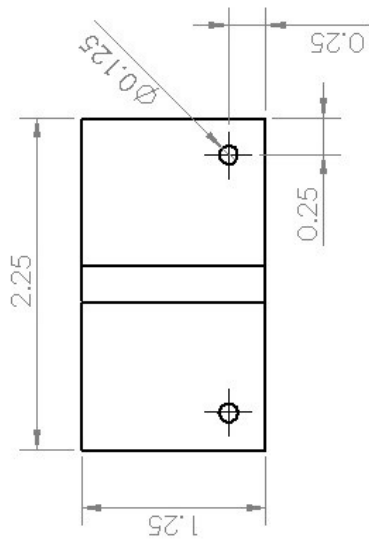
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 3/17/2018	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF LAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF LAVINO IS PROHIBITED.		DRAWN CHECKED		
MATERIAL Poplar FINISH		COMMENTS: SPONSORSHIP FOR LAVINO IS PROVIDED BY ASME, IEEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE DWG. NO. A D18
DO NOT SCALE DRAWING		SCALE: 1:2		REV 1
5	3	2	1	SHEET 1 OF 1




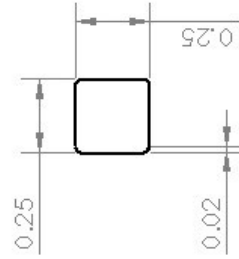
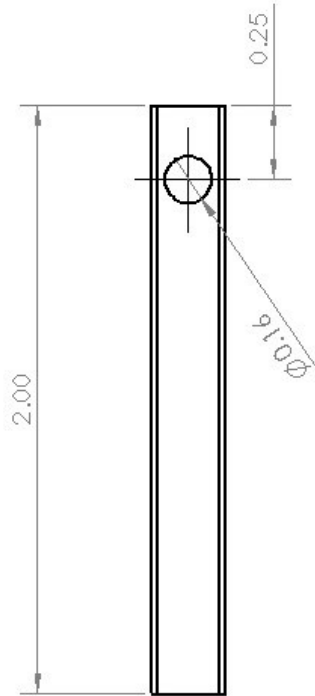
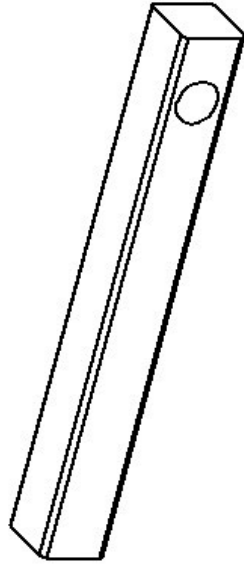
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. TOLERANCES: ± 0.1		NAME NC	DATE 3/17/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		DRAWN CHECKED		
MATERIAL POPCOT FINISH		COMMENTS: SPONSORSHIP FOR UAVINO B PROVIDED BY ASURE, IEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		REV 1
DO NOT SCALE DRAWING		3		SHEET 1 OF 1 1




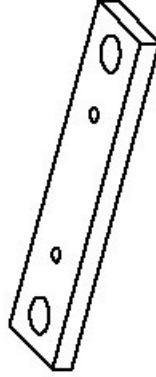
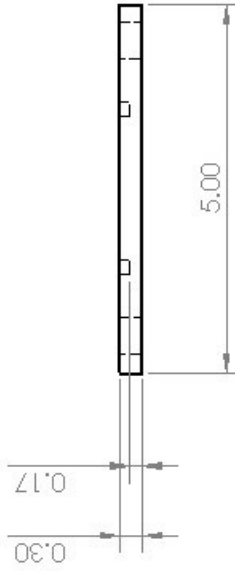
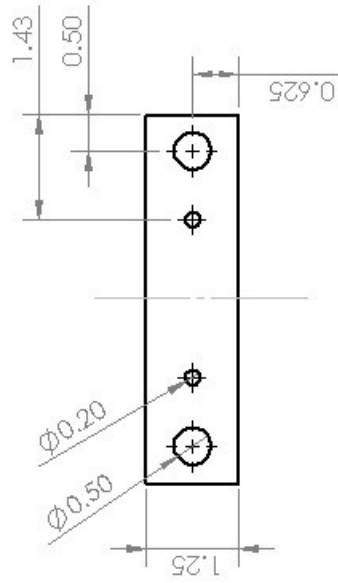
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME	DATE		
CHECKED	N/C	3/17/2015	UA VINO TITLE: Copper Plate			
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		SIZE A	DWG. NO. D21	REV 1
DO NOT SCALE DRAWING		SCALE: 1:1		SHEET 1 OF 1		




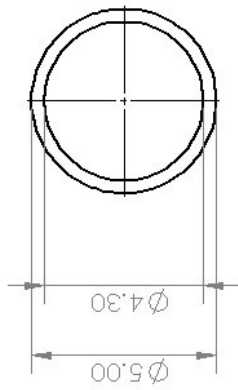
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALLOWANCES ± 0.1		DRAWN		NAME	DATE	
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALLOWANCES ± 0.1		CHECKED		NC	2/9/2015	
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO & ASSOCIATED BY AIR FORCE AND THE AIR FORCE RESEARCH AND DEVELOPMENT SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		TITLE: Ring Mounting Bracket		SIZE DWG. NO. A RO1 REV 1
DO NOT SCALE DRAWING		SCALE: 1:1		SHEET 1 OF 1		




UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	NAME	DATE	
CHECKED	NC	3/17/2015	UA Vino TITLE: Carbon Fiber Rod		
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UVA VINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UVA VINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR LAVINO 8 PROVIDED BY ASME IEE AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING		SIZE DWG. NO. A P02	REV. 1
MATERIAL: Carbon Fiber		D.O. NOT SCALE DRAWING		SCALE: 2:1	SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	CHECKED	NAME H C	DATE 3/17/2015	 UAVino	TITLE: Foot Base	REV
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTARY: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASHME, IEEE, AND THE SANDIA CORP. UNIVERSITY ROBOTICS SYSTEMS LABORATORY AT THE SCHOOL OF ENGINEERING		SIZE DWG. NO. A F01			SCALE: 1:2	SHEET 1 OF 1
MATERIAL: Poplar INCH*		DO NOT SCALE DRAWING		3		2		1



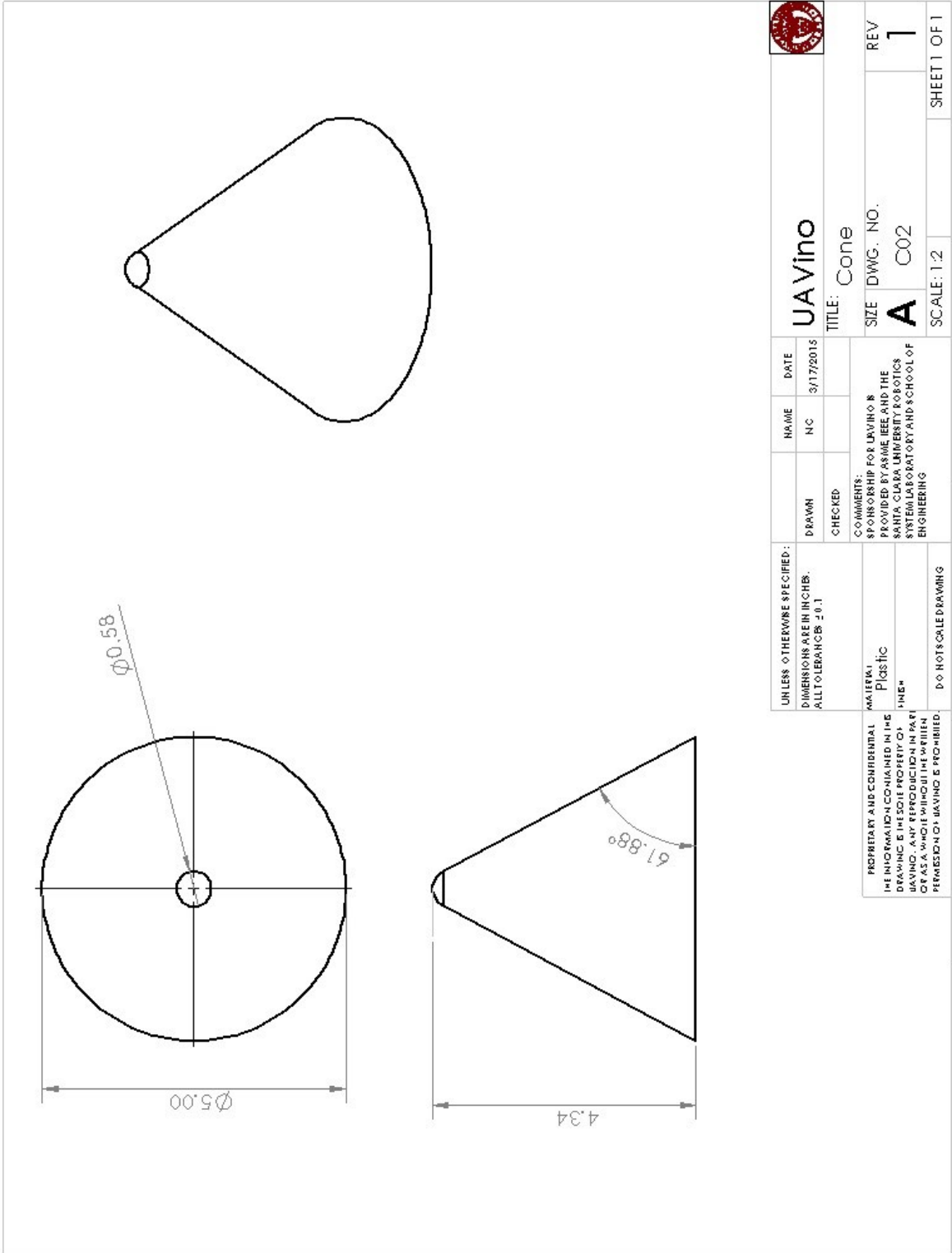
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES : 0.1		NAME NC	DATE 3/17/2015		
MATERIAL Acrylonitrile Butadiene Styrene (ABS) FINISH		CHECKED			UAVino TITLE: Vertical Cylinder
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO 8 BY STRENGTH ABS JEE, AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE DWG. NO. A C01	REV 1
DO NOT SCALE DRAWING				SCALE: 1:4	SHEET 1 OF 1

5

3

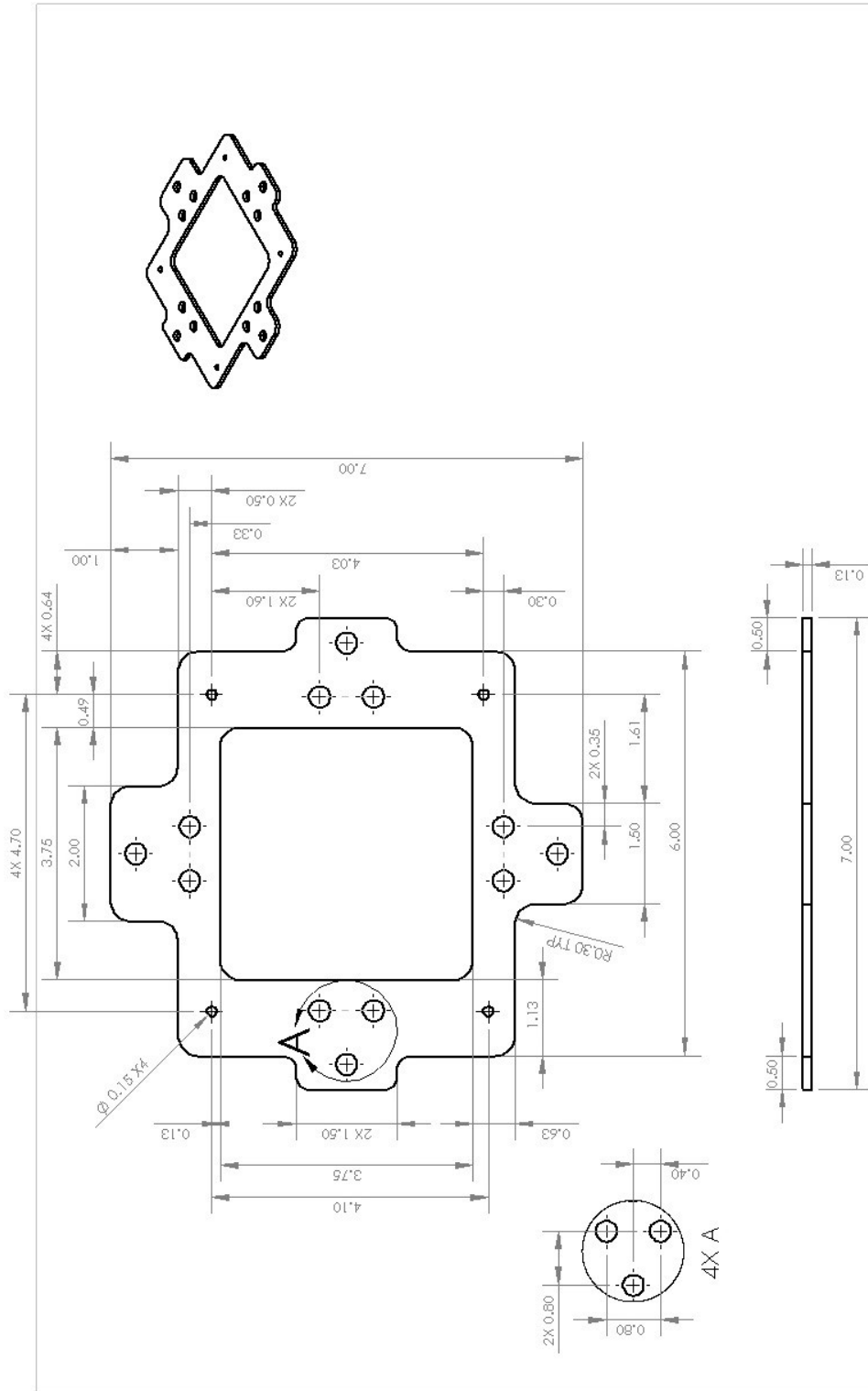
2

1



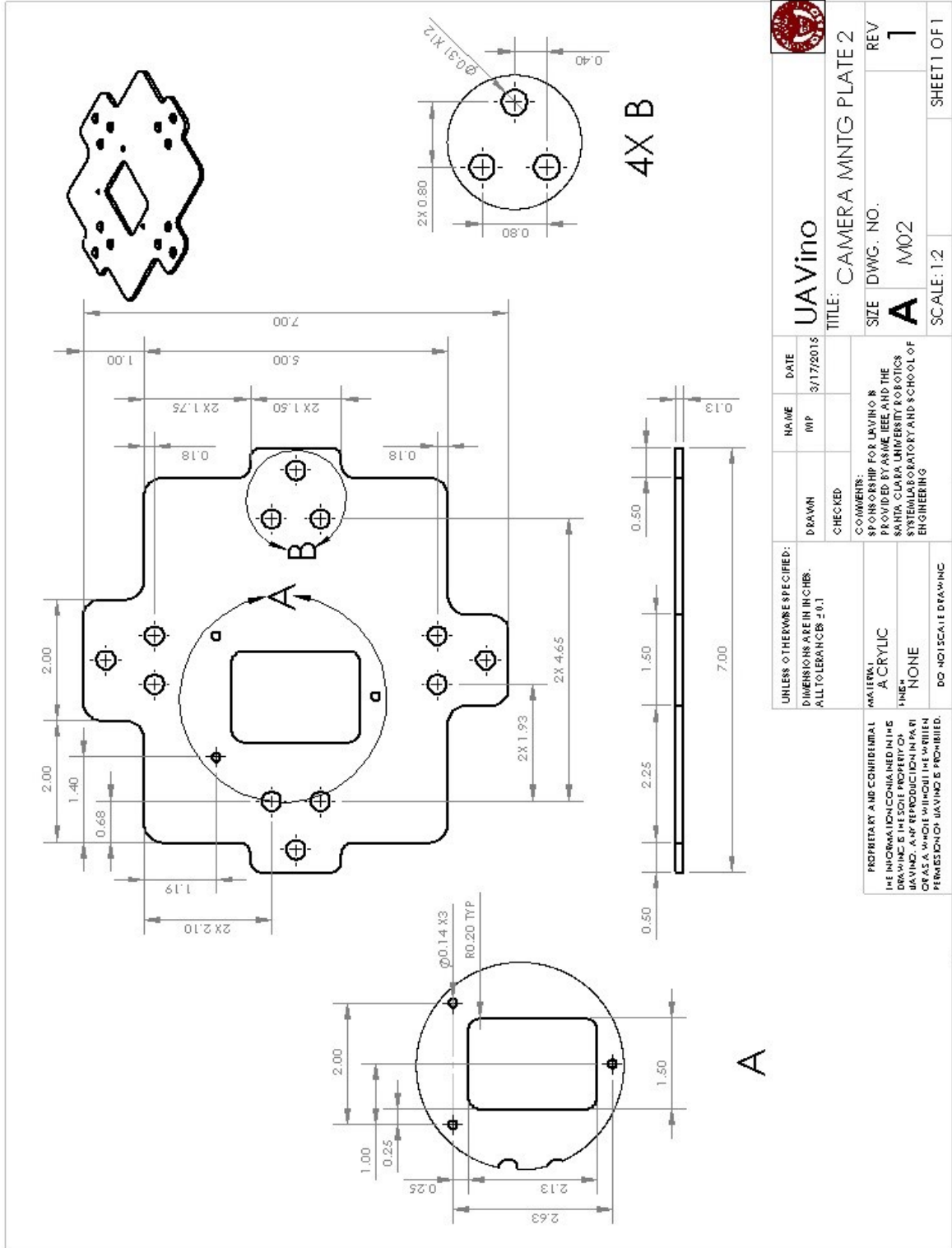
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME NC	DATE 3/17/2015		
DRAWN		CHECKED			UAVino TITLE: Cone
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.		COMMENTS: SPONSORSHIP FOR UAVINO IS PROVIDED BY ASME, IEEE AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEM LABORATORY AND SCHOOL OF ENGINEERING		SIZE DWG. NO. A C02	REV 1
DO NOT SCALE DRAWING		SCALE: 1:2		SHEET 1 OF 1	

5 3 2 1



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		DRAWN	CHECKED	NAME MP	DATE 3/14/2015
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. NO PART OF THIS DRAWING OR ANY INFORMATION HEREIN MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS WITHOUT THE WRITTEN PERMISSION OF UAVINO & PROHIBITED.	MATERIAL ACRYLIC	COMMENTS: SPONSORSHIP FOR UAVINO & PROCESSES BY ASHLEY WEEB AND THE SANTA CLARA UNIVERSITY ROBOTICS SYSTEMS LABORATORY AND SCHOOL OF ENGINEERING			
TITLE: CAMERA MNTG PLATE 1		UAVino		SCALE: 1:2	
REV		SIZE DWG. NO. A M01		SHEET 1 OF 1	

5 3 2 1



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES. ALL TOLERANCES ± 0.1		NAME MP	DATE 3/17/2015
DRAWN		CHECKED	COMMENT: SPONSORSHIP FOR UAVINO IS PROVIDED BY AMIE BEE AND THE SANTA CLARA UNIVERSITY ROBOTICS ENGINEERING DEPARTMENT AND SCHOOL OF ENGINEERING
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF UAVINO. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF UAVINO IS PROHIBITED.	MATERIAL ACRYLIC	FINISH NONE	DO NOT SCALE DRAWING
TITLE: CAMERA MNTG PLATE 2		SIZE A	DWG. NO. M02
REV		SHEET 1 OF 1	

APPENDIX I

Determining Ground Sample Distance Constant

Ground Sample Distance (GSD) is the ground distance represented by a pixel in an image. It is roughly linear with respect to the distance between the camera and the ground, which is approximated by the octocopter's altitude. The GSD is calculated by multiplying the relative altitude by a constant, c , where

$$c = \frac{\text{Pixel Size}}{\text{Focal Length}}$$

Reliable details for the pixel size and focal length of the Mobius vision camera were not readily available. Therefore, this constant was determined experimentally by setting the camera at a known fixed distance away from a target, measuring the true distance from the target to a secondary marker, and then measuring the pixel distance from the target to the marker. The constant is calculated as

$$c = \frac{\text{True Distance}}{(\text{Pixel Distance})(\text{Camera distance})}$$

The pixel distance was calculated using OpenCV to detect the target using a Haar Cascade classifier and measuring the distance between the center of the detected target and the marker. The measurement was repeated for four different positions of the target at each distance and was repeated ten times for each position. The target was positioned:

- Above the center of the camera frame
- Below the center of the camera frame
- To the left of the center of the camera frame
- To the right of the center of the camera frame

The calculated constant for each position and camera distance is shown in Table I.1

Table I.1: Average GSD Constant values for each camera distance and position

Camera Distance (inches)	Target Above Center	Target Below Center	Target Left of Center	Target Right of Center
72	0.00206214	0.002142313	0.002374061	0.002518516
108	0.002048804	0.002132202	0.002300974	0.002465543
144	0.002179538	0.000930099	0.000749191	0.00240162
180	0.002145062	0.002242788	0.002283587	0.002437097
216	0.002083795	0.002336668	0.002235786	0.002481399

The constants calculated for the Y positions (above and below) and for the X positions (left and right) resulted in different GSD constants. This is likely due to the Mobius' fisheye lens, which causes some distortion. The variance in measurements grew as camera distance increased, so the final GSD x and GSD y constant was calculated using only the measurements at distances 72 inches and 108 inches. The final constants are

$$GSD_x = 0.002414773$$

$$GSD_y = 0.002096365$$

Product Datasheets

Product Brief
Intel® Edison



Intel® Edison Development Platform

Introduction

The Intel® Edison development platform is designed to lower the barriers to entry for a range of inventors, entrepreneurs, and consumer product designers to rapidly prototype and produce "Internet of Things" (IoT) and wearable computing products.

Intel® Edison Board for Arduino*

Supports Arduino Sketch, Linux, Wi-Fi, and Bluetooth.

Board I/O: Compatible with Arduino Uno (except 4 PWM instead of 6 PWM):

- 20 digital input/output pins, including 4 pins as PWM outputs.
- 6 analog inputs.
- 1 UART (Rx/Tx).
- 1 I²C.
- 1 ICSP 6-pin header (SPI).
- Micro USB device connector OR (via mechanical switch) dedicated standard size USB host Type-A connector.
- Micro USB device (connected to UART).
- SD card connector.
- DC power jack (7 to 15 VDC input).

Intel® Edison Breakout Board

Slightly larger than the Intel® Edison module, the Intel® Edison Breakout Board has a minimal set of features:

- Exposes native 1.8 V I/O of the Edison module.
- 0.1 inch grid I/O array of through-hole solder points.
- USB OTG with USB Micro Type-AB connector.
- USB OTG power switch.
- Battery charger.
- USB to device UART bridge with USB micro Type-B connector.
- DC power supply jack (7 to 15 VDC input).

Intel® IoT Analytics Platform

- Provides seamless Device-to-Device and Device-to-Cloud communication.
- Ability to run rules on your data stream that trigger alerts based on advanced analytics.
- Foundational tools for collecting, storing, and processing data in the cloud.
- Free for limited and noncommercial use.

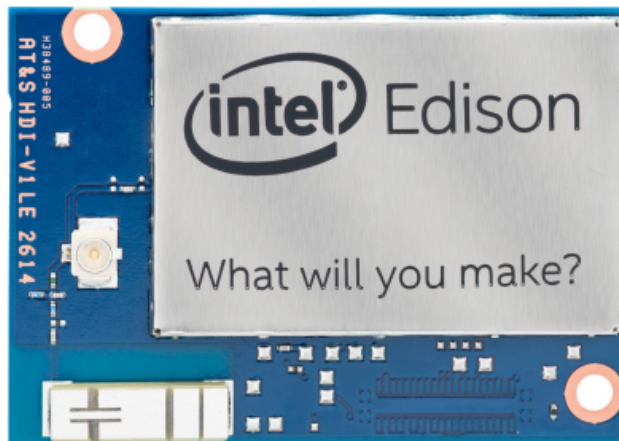


Figure J.1: Intel Edison datasheet (Page 1)

Intel® Edison Development Platform

PHYSICAL	
Form factor	Board with 70-pin connector
Dimensions	35.5 × 25.0 × 3.9 mm (1.4 × 1.0 × 0.15 inches) max
C/M/F	Blue PCB with shields / No enclosure
Connector	Hirose DF40 Series (1.5, 2.0, or 3.0 mm stack height)
Operating temperature	32 to 104°F (0 to 40°C)
EXTERNAL INTERFACES	
Total of 40 GPIOs, which can be configured as:	
SD card	1 interface
UART	2 controllers (1 full flow control, 1 Rx/Tx)
I2C	2 controllers
SPI	1 controller with 2 chip selects
I2S	1 controller
GPIO	Additional 12 (with 4 capable of PWM)
USB 2.0	1 OTG controller
Clock output	32 kHz, 19.2 MHz
MAJOR EDISON COMPONENTS	
SoC	22 nm Intel® SoC that includes a dual-core, dual-threaded Intel® Atom™ CPU at 500 MHz and a 32-bit Intel® Quark™ microcontroller at 100 MHz
RAM	1 GB LPDDR3 POP memory (2 channel 32bits @ 800MT/sec)
Flash storage	4 GB eMMC (v4.51 spec)
WiFi	Broadcom® 43340 802.11 a/b/g/n; Dual-band (2.4 and 5 GHz) Onboard antenna or external antenna (SKU configurations)
Bluetooth	Bluetooth 4.0
POWER	
Input	3.3 to 4.5 V
Output	100 ma @3.3 V and 100 ma @ 1.8 V
Power	Standby (No radios): 13 mW Standby (Bluetooth 4.0): 21.5 mW (BTLE in Q4-14) Standby (Wi-Fi): 35 mW
FIRMWARE + SOFTWARE	
CPU OS	Yocto Linux® v1.6
Development environments	Arduino® IDE Eclipse supporting: C, C++, and Python Intel XDK supporting: Node.JS and HTML5
MCU OS	RTOS
Development environments	MCU SDK and IDE



Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com/design/literature.htm>.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

Intel, the Intel logo, Atom, Pentium, Quark, and Xeon are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2014 Intel Corporation. All rights reserved.

♻️ Please Recycle

331179-001



Figure J.2: Intel Edison datasheet (Page 2)

RB-Trs-32
3DR PX4 Pixhawk Advanced Autopilot



Pixhawk is an advanced autopilot system designed by the PX4 open-hardware project and manufactured by 3D Robotics. It features advanced processor and sensor technology from ST Microelectronics® and a NuttX real-time operating system, delivering incredible performance, flexibility, and reliability for controlling any autonomous vehicle.

The benefits of the Pixhawk system include integrated multithreading, a Unix/Linux-like programming environment, completely new autopilot functions such as Lua scripting of missions and flight behavior, and a custom PX4 driver layer ensuring tight timing across all processes. These advanced capabilities ensure that there are no limitations to your autonomous vehicle. Pixhawk allows existing APM and PX4 operators to seamlessly transition to this system and lowers the barriers to entry for new users to participate in the exciting world of autonomous vehicles.

The flagship Pixhawk module will be accompanied by new peripheral options, including a digital airspeed sensor, support for an external multi-color LED indicator and an external magnetometer. All peripherals are automatically detected and configured.

Features

- Advanced 32 bit ARM Cortex® M4 Processor running NuttX RTOS
- 14 PWM/servo outputs (8 with failsafe and manual override, 6 auxiliary, high-power compatible)
- Abundant connectivity options for additional peripherals (UART, I2C, CAN)
- Integrated backup system for in-flight recovery and manual override with dedicated processor and stand-alone power supply
- Backup system integrates mixing, providing consistent autopilot and manual override mixing modes
- Redundant power supply inputs and automatic failover
- External safety button for easy motor activation

Figure J.3: Pixhawk datasheet (Page 1)

- Multicolor LED indicator
- High-power, multi-tone piezo audio indicator
- microSD card for long-time high-rate logging

Specifications

Microprocessor

- 32 bit STM32F427 Cortex M4 core with FPU
- 168 MHz/256 KB RAM/2 MB Flash
- 32 bit STM32F103 failsafe co-processor

Sensors

- ST Micro L3GD20H 16 bit gyroscope
- ST Micro LSM303D 14 bit accelerometer / magnetometer
- MEAS MS5611 barometer

Interfaces

- 5x UART (serial ports), one high-power capable, 2x with HW flow control
- 2x CAN
- Spektrum DSM / DSM2 / DSM-X® Satellite compatible input
- Futaba S.BUS® compatible input and output
- PPM sum signal
- RSSI (PWM or voltage) input
- I2C®
- SPI
- 3.3 and 6.6V ADC inputs
- External microUSB port

Power System

- Ideal diode controller with automatic failover
- Servo rail high-power (7 V) and high-current ready
- All peripheral outputs over-current protected, all inputs ESD protected

Weight and Dimensions

- Weight: 38g (1.31oz)
- Width: 50mm (1.96")
- Thickness: 15.5mm (.613")
- Length: 81.5mm (3.21")

Figure J.4: Pixhawk datasheet (Page 2)

Basic Specifications:

- Rechargeable LIPO Battery (820mah), with Battery Charger Manage IC
- Video: 1080P 30FPS FULL HD, 720P 60FPS, 720P 30FPS HD H.264/AVC1 video codec, selectable file format.
- Photo: 2304 x 1536, 1920 x 1080, 1280 x 720, supports Time Lapse Photo Shooting
- microSD card slot, supports up to 32GB (some 64 & 128GB cards), must use Class 4 or above.
- USB2.0, plug and play, easy connection with computers, no driver needed.
- Live TV video output while recording videos with the TV out cables (TV out cable sold separately)
- Super mini size, only around 61mm (L) x 35mm (W) x18mm (H)
- Super light-weighted: only approx 38g!
- Multifunctional, you can use it as an ActionCam, GunCam, BowCam, Pocket camcorder, sports camera, driving recorder, a camera, a webcam, a removable USB drive.



What's special about this Mobius ActionCam?

1. It's a camera designed and developed by users and for users. And it's on-going. We will keep improving this camera and add more features by firmware updating. If you have any feedback / comments / suggestions, please contact us!
2. This camera is so small and light, and it has wide field of view with minimum fisheye distortion. It has a quality lens and it has outstanding image quality, corner to corner sharpness and uniform brightness across entire frame. Check out the video samples here, and from other users. Also check out the great Mobius forum maintained by Tom Frank. You will get plenty of helpful information there as well.
3. This camera ships with most updated and stable firmware of the time, and it works perfectly out of box. Don't worry about the complicated settings (actually it's not complicated at all, you will find it fun and easy later).
4. Below is a list of the current Mobius settings for the current firmware version; and there will be more to follow.

Figure J.5: Mobius datasheet (Page 1)

- Set the video resolution and frame rate for Video Mode 1 & 2 separately.
 - Selectable Video Resolutions (1080p, 720p & WVGA)
 - Selectable Video Frame Rates (5 fps, 10 fps, 15 fps, 20 fps, 25 fps, 30 fps, 50 fps, 60 fps) to corresponding resolutions.
 - Field of View (FOV) Narrow & Wide selectable to corresponding resolutions.
 - Set Photo Size (2304×1536, 1920×1080, 1280×720, 848×480)
 - Set Time Lapse photo shooting (from 0.25s a photo to 30s & custom with Day, Hours & Minutes settings)
 - Video Clip length/Movie Cycle Time (3minutes, 5minutes, 10 minutes, 15 minutes, and max to 4G byte)
 - Time Lapse Video; (0.25s, 0.5s, 1s, 2s, 5s, 10s, 30s & custom with Hours & Minutes settings)
 - Video Motion Detect with Sensitivity & Timeout Settings
 - Video File Type selectable; MOV, AVI, MP4 or WAV (sound only)
 - Video Image Settings; user configurable; Sharpness, Exposure, Contrast, Saturation, Color Options, White Balance)
 - You can set loop recording On or Off
 - You can set time stamp on or off
 - You can mute the video sound if needed.
 - You can flip the video 180 degrees (when you use the camera upside down)
 - You can set the Movie quality (Low, Standard, High, default is Standard)
 - You can set WDR (Wide/High Dynamic Range) for low light filming.
 - To avoid camera sudden powering on in your pocket, you can set the power on delay.
 - You can auto power off waiting time and set it off.
 - You can set Auto record on with power (helpful when using Mobius as Driving recorder / Car Dash Camera)
 - You can set the LED indicator light off if needed (default is on).
 - If you use Android devices, you may use them to set this camera or view your videos. (App can be located and downloaded from the Google store, by Theau2000)
 - You can set TV out video mode and 4:3 or 16:9.
5. The camera can be configured manually by editing the SYSCFG.txt, but it is suggested to us Isoprop's great windows GUI ([msetup](#)). The simple and extremely user-friendly program is fully plug-and-play and includes an integrated User Manual. (a similar tool for Mac based systems also exists; [MobiusManager.App](#))



Figure J.6: Mobius datasheet (Page 2)

Charger specifications

For battery types	Lithium Polymer (1s to 4s balanced, 1s to 2s unbalanced)* Lithium Ion (1s to 4s balanced, 1s to 2s unbalanced) Lithium Manganese (1s to 4s balanced, 1s to 2s unbalanced) A123 (2s to 4s balanced, 1s to 5s unbalanced)* NiCd (1s to 12s) NiMH (1s to 12s) 6V and 12V Lead Acid batteries
Pack capacity	5mAh to 32Ah (charge time limited to maximum of 8 hours)
Input voltage	10 to 16VDC, reverse polarity protected
Input current	5A maximum
Power conversion	125kHz switcher operating at 90% efficiency
Output current	Up to 4A
Cell balancing	To within 5mV for 2s to 4s LiPo, Li-Ion, Li-Mn or A123 packs
Voltage calibration	Cell voltage measurements are factory calibrated to a standard traceable to NIST; calibration is to $\pm 6\text{mV}$
Current calibration	Charge current is factory calibrated on a 4A standard; calibration is to $\pm 1\text{mA}$
Measurement accuracy	Voltage: $\pm 6\text{mV}$ Charge current: $\pm 1\%$ Capacity added to pack: $\pm 1\%$ Percent capacity ("Fuel"): $\pm 5\%$
Serial data output	19.2kbps, 8 bits, 1 start bit, 1 stop bit, no parity

**charger can be used with balanced 2s to 4s packs having a node connector (an appropriate FMA adapter may be required)*

Figure J.7: Revoletrix CellPro Multi4 datasheet



Raspberry Pi



MODEL B+

Product Name Raspberry Pi Model B+

Product Description The Raspberry Pi Model B+ incorporates a number of enhancements and new features. Improved power consumption, increased connectivity and greater IO are among the improvements to this powerful, small and lightweight ARM based computer.

Specifications

Chip	Broadcom BCM2835 SoC
Core architecture	ARM11
CPU	700 MHz Low Power ARM1176JZFS Applications Processor
GPU	Dual Core VideoCore IV® Multimedia Co-Processor Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	512MB SDRAM
Operating System	Boots from Micro SD card, running a version of the Linux operating system
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V, 2A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Audio Output	3.5mm Jack, HDMI
USB	4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
JTAG	Not populated
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	SDIO

Figure J.8: Raspberry Pi datasheet

netis
Easy Network, Trustable

AC1200 Wireless Dual Band USB Adapter

WF2190

802.11 AC 867 Mbps	5dBI	USB 3.0	Soft AP
WPS	Win8 & 8.1	MAC OS	Linux

Features:

- >Maximum speed up to 2.4GHz 300Mbps and 5GHz 867Mbps
- >Integrate the latest 802.11 AC technology, use in 2.4 G or 5 G Networks
- >2 * 5dBI detachable antennas allowing for stronger antenna upgrades
- >Advanced security feature with the latest WPA2 encryption to protect the network
- >Easy connection to a wireless network at a push of the WPS button
- >Access Point feature to set up a Wi-Fi hotspot to share the Internet
- >Easy to install and configure with the setup CD and utility

Brief:

The netis WF2190 is designed to connect a desktop or notebook computer to a wireless network and access high-speed Internet connection. It's compatible with 802.11a/b/g/n/ac devices and provides the wireless transfer speed up to 2.4GHz 300Mbps or 5GHz 867Mbps, offering a better performance on online gaming, HD video streaming, and VoIP phone calling. With the 802.11n MIMO and 802.11ac technology, It ensures a strong and stable wireless connection and allows you to enjoy the wireless freedom around your home. Moreover, it features with a WPS button which helps you easily setup a secure wireless connection in a snap.

www.netis-systems.com

Figure J.9: Netis AC1200 Wireless Adaptor datasheet (Page 1).

Specification:

Wireless	
Standards	IEEE 802.11a/b/g/n/ac
Signal Rate	2.4GHz Up to 300Mbps; 5GHz UP to 867Mbps
Frequency Range	2.4-2.4835GHz; 5.180-5.825GHz
Wireless Mode	Station, Access Point
Wireless Security	Support 64/128-bit WEP, WPA-PSK/WPA2-PSK, 802.1 X
Hardware	
Port	USB 3.0
Button	WPS
Dimensions(L x W x H)	92.5 x 27 x 15 mm
Others	
Certification	FCC, CE, KC
System Requirements	Windows XP (32/64bits), Windows Vista (32/64bits), Windows 7 (32/64bits), Windows 8 (32/64bits), Windows 8.1 (32/64bits); Linux (as Ubuntu 12.10/13.04/13.10/14.10); Mac OS 10.4/10.5/10.6/10.7/10.8/10.9/10.10
Environment	Operating Temperature: 0°C~40°C Operating Humidity: 10%~90% non-condensing Storage Temperature: -40°C~70°C Storage Humidity: 5%~90% non-condensing
Package	1 * WF2190 1 * USB Cradle 1 * Driver CD 1 * Quick Installation Guide

Topological Graph:



Figure J.10: Netis AC1200 Wireless Adaptor datasheet (Page 2).

Specifications

3.2 megapixel CMOS sensor, 2048 x 1536
Sensor Image Array 6.59x4.9 mm
Sensor pixel pitch 3.18 μm .

Permanently mounted long pass filter in front of lens
Image storage to micro SD cards in Tetracam RAW or DCM lossless format
Miniature 8.3 mm fixed focus and aperture lens
USB Disk interface
Multi-pin I/O connector for use with Tetracam designed accessories

Image Capture (speed dependent on SD card features)

Capacity: (DCM10) Approx. 2.3MB per image
(RAW10) 6.15MB per image
(RAW8) 3.07MB per image

Rate: (DCM10) Capture to end of cycle: 7 sec.
(RAW10) Capture to ready : 2.8 sec.
(RAW 8) Capture to ready : 3 sec.

Inputs

6 to 12 VDC
RS-232 For user controls and NMEA GPS sentences
External Trigger
USB 2.0 Data Connection

Outputs

NTSC or PAL video
USB 2.0 Data Connection

ADC Micro Dimensions

4.5 x 3.0 x 0.9 in. (75 mm x 59 mm x 33 mm) lens included
3.53 oz. (100 gr.)

Figure J.11: Tetracam ADC Micro datasheet.

APPENDIX K

UAVino Charge Control Source Code

CellproMulti4.py

```
#!/usr/bin/env python

import time
import RPi.GPIO as gpio

relayPin = 11

class CellproMulti4:
    def __init__(self, communication_object, charger_number=0):
        self.communication_object = communication_object
        self.charger_number = charger_number

        gpio.setmode(gpio.BOARD)
        print 'Setting up pin %d' % self.pin
        gpio.setup(relayPin, gpio.OUT)
        gpio.output(relayPin, gpio.LOW)

    def SelP(self, preset_num):
        to_send_string = "SelP" + chr(preset_num)
        crc =
self.communication_object.sendStringAndFetchCRC(to_send_string)
        #TODO: calculate CRC
        return True

    def Sel(self, command_letter):
        if( command_letter in ['B'] ):
            to_send_string = "Sel" + command_letter
            crc =
self.communication_object.sendStringAndFetchCRC(to send string)
            return(crc == 0x05DC)
            #print self.communication_object.sendStringAndFetchBytes(
to_send_string, 149 )
        else:
            raise Exception("Invalid Command Letter %s" %
command_letter)

    def start_charge(self):
        status = self.get_status()
        if( status["mode"] == 0 ):
            self.Sel('B') # go to confirm battery phase
            time.sleep(2)
            status = self.get_status()
            time.sleep(2)
            if( status["mode"] == 0 ): # ensure no error
```

```

        self.Sel('B') # start charging battery
        gpio.output(relayPin, gpio.HIGH)
        return True

def stop(self):
    status = self.get_status()
    if( status["mode"] in [6,7,8,9,11] ):
        if( self.Sel('B') ):
            time.sleep(1)
            status = self.get_status()
            if( status["mode"] == 0 ):
                gpio.output(relayPin, gpio.HIGH)
                return True
    return False

def choose_preset( self, number ):
    if( type(number) == type(0) and number >= 0 and number <= 24 ):
        return self.SelP( number )
    else:
        raise Exception( "Invalid preset %d" % number )

def isError(self, status=None):
    if not status:
        status = self.get_status()
    return status["mode"] == 99

def clear_error(self, status=None):
    if not status:
        status = self.get_status()
    if( status["mode"] == 99 ):
        if( self.Sel('B') ):
            time.sleep(1)
            status = self.get_status()
            if( status["mode"] == 0 ):
                return True
    return False

def isCharging(self, status=None):
    if not status:
        status = self.get_status()
    return True if status["mode"] in [6, 7] else False

def get_status(self):
    ret = {}
    raw_byte_array =
self.communication_object.sendStringAndFetchBytes(
"Ram"+chr(self.charger_number), 149 )
    print raw_byte_array
    print len(raw_byte_array)
    #print raw_byte_array.index(20)
    def word( first_index ):
        return
(raw_byte_array[first_index]<<8)+raw_byte_array[first_index+1]
    def dword( first_index ):
        return (word(first_index)<<16)+word(first_index+1)

```

```

def sword( first_index ):
    ret = word(first_index)
    if( ret >= 32768 ):
        ret = ret - 65536
    return ret

ret["mode"] = raw_byte_array[-13]
ret["loaded_preset"] = raw_byte_array[84]
ret["version"] = raw_byte_array[0]/100.0

#TODO: cell voltages calculate incorrectly
#ret["cell_voltages"] = [ x * 5.12 / 65536 for x in
[word(i*2+1) for i in range(0,4)] ]

self.last_status = ret
return ret

```

CellproMulti4 Serial.py

```
#!/usr/bin/env python

#runs on
#inspired by https://github.com/coryrc/battery-cycler
import serial,time

class CellproMulti4_Single_Serial:
    def __init__( self, serialPortFilename ):
        self.filename = serialPortFilename
        self.open()

    def sendStringAndFetchCRC(self, some_string):
        result = self.sendStringAndFetchBytes(some_string,2)
        return (result[0]<<8)+result[1]

    def sendStringAndFetchBytes(self, some_string, number_bytes):
        try:
            self.usbSerialInterface.flushInput()
            self.usbSerialInterface.write(some_string)
            self.usbSerialInterface.flush()
            s_ret =
self.usbSerialInterface.read(number_bytes+len(some_string))
        except OSError, e:
            print "Serial port no longer exists"
            self.usbSerialInterface.close()
            self.open()
            return self.sendStringAndFetchBytes(some_string,
number_bytes)
        retList = [ord(x) for x in s_ret]
        for i in range(0, len(some_string)):
            del retList[0]
        return retList

    def open(self):
        attemptNum = 1
        while(attemptNum <= 10):
            try:
                self.usbSerialInterface = serial.Serial(self.filename,
19200, timeout=.2)
            return
            except serial.serialutil.SerialException, e:
                print "Open serial error: " + str(e)
                time.sleep(2)
                attemptNum += 1
        exit(1)

    def close(self):
        self.usbSerialInterface.close()
```

ChargeRoutine.py

```
import CellproMulti4
from CellproMulti4_Serial import CellproMulti4_Single_Serial
import time

if __name__ == "__main__":

    communication = CellproMulti4_Single_Serial("/dev/ttyUSB0")
    chargerController = CellproMulti4.CellproMulti4(communication)

    chargerController.clear_error()

    print chargerController.get_status()
    chargerController.start_charge()
    print chargerController.get_status()

    #get status once, no need to constantly keep getting more statuses
    status = chargerController.get_status()
    while chargerController.isCharging(status):
        if chargerController.isError(status):
            chargerController.stop()
            chargerController.clear_error()
            exit(1) #failsafe state, return with error status 1,
            overlying script will handle error
            time.sleep(10)
            status = chargerController.get_status()

    exit(0)
```

APPENDIX L

UAVino Landing Control Source Code

DetectObject.py

```
"""
Contains methods used to detect and display targets using OpenCV
"""

__author__ = 'Kirby'
import cv2
import cv

# Returns detected targets
def detect_targets(frame, cascade):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cv2.equalizeHist(gray, gray)
    targets = cascade.detectMultiScale(gray, 1.1, 2,
cv.CV_HAAR_SCALE_IMAGE, (30, 30))
    return targets

# Displays an image with boxes around the detected targets
def display_targets(frame, targets):
    for (x, y, w, h) in targets:
        # Draws a bounding rectangle around the object and a small
rectangle at the center of the object
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 8)
        cv2.rectangle(frame, (x + w/2 - 5, y + h/2 - 5), (x + w/2 + 5,
y + h/2 + 5), (0,255,0), 8)
        cv2.imshow("Detected Objects", frame)

def detect_and_display_targets(frame, cascade):
    targets = detect_targets(frame, cascade)
    display_targets(frame, targets)

# Overlays boxes around the detected targets and returns the image
def overlay_targets(frame, targets):
    for (x, y, w, h) in targets:
        # Draws a bounding rectangle around the object and a small
rectangle at the center of the object
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255,0,255), 8)
        cv2.rectangle(frame, (x + w/2 - 5, y + h/2 - 5), (x + w/2 + 5,
y + h/2 + 5), (255,0,255), 8)
    return frame

# saves an image to the given file
def save_image(frame, save_file):
    return cv2.imwrite(save_file, frame)
```

GenerateDirections.py

```
"""
Contains methods that generate the directions needed to center a camera
over a target
"""

__author__ = 'Kirby'

import Command as cmd

# Speed in cm
# TODO: figure out good default speeds (maybe tiered system, close and
not close)
default_speed = 20

# divide the altitude by this each time until close to the ground
# TODO: figure out both this factor and the altitude threshold
altitude_descent_factor = 2

# Values used to check to see if the object is "close enough" to
# the center of the target (in pixels)
# TODO: figure out the desirable center buffer
center_width = 80
center_height = 80
center_width_buffer = center_width / 2
center_height_buffer = center_height / 2

def get_next_command(obj_center_x, obj_center_y, pic_width, pic_height,
current_altitude, GSD_constant_x, GSD_constant_y):
    """
    Generates directions needed to center a camera over a target

    This method generates directions for a mobile camera
    and stationary target. A stationary target is not required but the
    camera must be mobile. If instead you want to
    center a target using a stationary camera, the horizontal (xy)
    directions generated by this function should be
    reversed.

    :param obj_center_x:
    :param obj_center_y:
    :param pic_width:
    :param pic_height:
    :param current_altitude:
    :param GSD_constant_x:
    :param GSD_constant_y:
    """
    dist_per_pixel_x = GSD_constant_x * current_altitude
    dist_per_pixel_y = GSD_constant_y * current_altitude
    pic_center_x = pic_width / 2
    pic_center_y = pic_height / 2

    x_dist = 0
    y_dist = 0
    z_dist = 0
```

```

# TODO: add in code to orient the drone by rotating if needed

# Calculate x direction movement (+x is right, -x is left)
if abs(obj_center_x - pic_center_x) > center_width_buffer:
    x_dist = (obj_center_x - pic_center_x) * dist_per_pixel_x

# Calculate y direction movement (image y direction which means 0
is at the top)
if abs(obj_center_y - pic_center_y) > center_height_buffer:
    y_dist = (obj_center_y - pic_center_y) * dist_per_pixel_y

# Move in the z direction
if x_dist == 0 and y_dist == 0:
    # TODO: add in positive z direction for the sake of completion
    z_dist = (current_altitude -
current_altitude/altitude_descent_factor)
    next_command = cmd.Command(cmd.CommandKind.Z_TRANSLATION,
frame=cmd.RefFrame.BODY,
                                vector=cmd.Vector(x=0, y=0,
z=z_dist), speed=default_speed)

# Move in the xy plane
else:
    next_command = cmd.Command(cmd.CommandKind.XY_TRANSLATION,
frame=cmd.RefFrame.BODY,
                                vector=cmd.Vector(x=x_dist,
y=y_dist, z=0), speed=default_speed)

return next_command

```


c_constructs.py

```
"""
Contains the class definitions needed to emulate some useful C
constructs

Currently only enums are implemented
"""

__author__ = 'Kirby'

class Enum(set):
    """
    Helper class to mimic C's enumerations
    """
    def __getattr__(self, name):
        if name in self:
            return name
        raise AttributeError('Invalid enumeration value')
    def __setattr__(self, name, value):
        raise AttributeError('Cannot change the value of an
enumeration')
    def __delattr__(self, item):
        raise AttributeError('Cannot delete the value of an
enumeration')
```

Command.py

```
"""
Implements the Command class which contains a command to send to a
copter
"""

__author__ = 'Kirby'

from math import radians, cos, sin, pi
from numpy import array, dot
from c_constructs import Enum

# For the body frame:
#
# +X points towards the right side of the copter
#
# +Y points towards the back side of the copter
#
# +Z points up away from the copter
#
# Rotations are yaw rotations
#
# For the NED frame:
#
# +X points North
#
# +Y points East
#
# +Z points Down towards the Earth
#
# Rotations are yaw rotations
#
# Note: at yaw = pitch = roll = 0:
#
#  $N = -Y$ 
#
#  $E = X$ 
#
#  $D = -Z$ 

#: Enumeration that indicates the type of command
CommandKind = Enum(["XY_TRANSLATION", "Z_TRANSLATION", "YAW_ROTATION"])

# RefFrame should properly be an enumeration
# RefFrame represents the frame of reference of the command
# See the comments above CommandKind for explanations of each frame of
reference
#: Enumeration that indicates the frame of reference of the command
RefFrame = Enum(["BODY", "NED"])

class Vector():
    """
    Helper class that contains x, y, and z coordinates for a 3-
    dimensional vector
    """
```

```

:param x: x component vector
:param y: y component vector
:param z: z component vector
"""
def __init__(self, x=0, y=0, z=0):
    self._x = x
    self._y = y
    self._z = z

@property
def x(self):
    """
    Get x component

    :return: x
    """
    return self._x

@property
def y(self):
    """
    Get y component

    :return: y
    """
    return self._y

@property
def z(self):
    """
    Get z component

    :return: z
    """
    return self._z

def __str__(self):
    return "X: {:.2f} \tY: {:.2f} \tZ: {:.2f}".format(self._x,
self._y, self._z)

class Command():
    """
    Class containing a command to send to a copter

    Creates either a translation command, in which case vector should
    be set, or a rotation command, in which case
    degrees should be set

    :param command_kind: kind of movement, either a translation or
    rotation an attribute of an instance of CommandKind
    :param frame: frame of reference, an attribute of an instance of
    Frame
    :param vector: desired movement vector, an instance of Vector,
    command_kind must be a translation
    :param degrees: desired yaw rotation in degrees, command_kind must

```

```

be a rotation
    :param speed: desired speed in cm/s
    """

    def __init__(self, command_kind, frame, vector=None, degrees=None,
speed=10):

        if command_kind in ([CommandKind.XY_TRANSLATION,
CommandKind.Z_TRANSLATION, CommandKind.YAW_ROTATION]):
            if frame in ([RefFrame.BODY, RefFrame.NED]):
                self._command_kind = command_kind
                self._frame = frame
                self._speed = speed

                if vector is not None and command_kind not in
([CommandKind.XY_TRANSLATION, CommandKind.Z_TRANSLATION]):
                    raise ValueError('Error: Tried to pass in a vector
to a non-translational command kind')
                self._vector = vector

                if degrees is not None and command_kind not in
([CommandKind.YAW_ROTATION]):
                    raise ValueError('Error: Tried to pass in a degree
rotation to a non-rotational command kind')
                self._degrees = degrees

            else:
                raise ValueError('Invalid reference frame passed in')
        else:
            raise ValueError('Invalid command kind passed in')

# Set translation parameters
def set_translation_params(self, vector=Vector(0,0,0), speed=10):
    """
    Set vector and speed, command_kind must be a translation

    :param vector: desired movement vector, an instance of Vector
    :param speed: desired speed in cm/s
    """
    if self._command_kind in ([CommandKind.XY_TRANSLATION,
CommandKind.Z_TRANSLATION]):
        self._vector = vector
        self._speed = speed
    else:
        raise ValueError('Translation params function called for
invalid or non-translation command kind: '
+ self._command_kind)

def set_rotation_params(self, degrees=0, speed=10):
    """
    Set degrees to rotate

    :param degrees: desired degrees of yaw rotation
    :param speed: desired speed in cm/s
    """
    if self._command_kind in [CommandKind.YAW_ROTATION]:

```

```

        self._degrees = degrees
        self._speed = speed
    else:
        raise ValueError('Rotation params function called for
invalid or non-rotation command kind: '
                          + self._command_kind)

@property
def command_kind(self):
    """
    Get Command Kind

    :return: command kind
    """
    return self._command_kind

@property
def frame(self):
    """
    Get Frame of Reference

    :return: frame of reference
    """
    return self._frame

@property
def get_vector(self):
    """
    Get the move Vector

    :return: vector
    """
    if self._command_kind not in ([CommandKind.XY_TRANSLATION,
CommandKind.Z_TRANSLATION]):
        raise ValueError('distance value is a translation parameter
but has been requested by invalid or '
                          'non-translation command kind: ' +
self._command_kind)
    return self._vector

@property
def degrees(self):
    """
    Get the rotation degrees

    :return: degrees
    """
    if self._command_kind in [CommandKind.YAW_ROTATION]:
        raise ValueError('degrees value is a rotation parameter but
has been requested by invalid or '
                          'non-rotation command_kind: ' +
self._command_kind)
    return self._degrees

@property
def speed(self):
    """

```

```

        Get the move/rotation speed

        :return: speed
        """
        return self._speed

    def transform_body_to_NED_yaw_only(self, yawd):
        """
        Transforms Commands in the camera (body) frame to coordinates
        in the NED frame but assumes that roll and pitch
        are 0. The transform_body_to_NED_yaw_pitch_roll method should
        be used instead since the yaw pitch roll method
        is more accurate and the speed difference is negligible

        :param yawd: current yaw in degrees
        """
        if self._frame == RefFrame.NED:
            # Already in NED reference frame
            return
        elif self._frame == RefFrame.BODY:
            # convert from degrees to radians
            yawr = radians(yawd)

            self._vector = Vector(self._vector.x * -sin(yawr) -
self._vector.y * cos(yawr),
                                self._vector.x * cos(yawr) -
self._vector.y * sin(yawr),
                                -self._vector.z)
            self._frame = RefFrame.NED
        else:
            raise ValueError('Transformation from given reference frame
to NED reference frame not supported')

    def transform_body_to_NED_yaw_pitch_roll(self, yawd, pitchd,
rolld):
        """
        Transforms a Command in the camera (body) frame to a Command in
        the NED frame. Use this method
        instead of transform_body_to_NED_yaw_only since this method is
        more accurate. Though this method is
        theoretically slower, the difference is negligible in practice

        :param yawd: current yaw in degrees
        :param pitchd: current pitch in degrees
        :param rolld: current roll in degrees
        """
        if self._frame == RefFrame.NED:
            # Already in NED reference frame
            return
        elif self._frame == RefFrame.BODY:
            # convert from degrees to radians
            y = radians(yawd)
            p = radians(pitchd)
            r = radians(rolld)

```

```

        # The full transformation matrix, incorporates the yaw,
        # pitch, and roll matrices all multiplied together
        # This method is the fastest
        ypr_mat = array([[cos(y)*cos(p),      cos(y)*sin(r)*sin(p)-
cos(r)*sin(y),      sin(r)*sin(y)+cos(r)*cos(y)*sin(p)],
                        [sin(y)*cos(p),
cos(r)*cos(y)+sin(r)*sin(y)*sin(p),      cos(r)*sin(y)*sin(p)-
cos(y)*sin(r)],
                        [-sin(p),      cos(p)*sin(r),
cos(r)*cos(p)]])

        # directions: N = -y, E = x, D = -z
        in_vec = ([-self._vector.y],
                  [self._vector.x],
                  [-self._vector.z])
        out_vec = dot(ypr_mat, in_vec)

        # out_vec is a 3x1 vector so N = [0][0], E = [1][0], D =
[2][0]

        self._vector = Vector(out_vec[0][0],
                              out_vec[1][0],
                              out_vec[2][0])
        self._frame = RefFrame.NED

    else:
        raise ValueError('Transformation from' + self._frame +
'reference frame to NED reference frame not supported')

    def transform_body_to_NED_yaw_pitch_roll_in_rads(self, yawr,
pitchr, rollr):
        """
        Transforms a Command in the camera (body) frame to a Command in
        the NED frame. Use this method
        instead of transform body to NED yaw only since this method is
        more accurate. Though this method is
        theoretically slower, the difference is negligible in practice

        :param yawr: current yaw in radians
        :param pitchr: current pitch in radians
        :param rollr: current roll in radians
        """

        yawd = yawr * 180 / pi
        pitchd = pitchr * 180 / pi
        rolld = rollr * 180 / pi

        self.transform_body_to_NED_yaw_pitch_roll(yawd, pitchd, rolld)

    def __str__(self):
        return_string = "Command Kind: \t" + self._command_kind
        return_string += "\nFrame of Reference: \t" + self._frame

        if self._command_kind in [CommandKind.YAW_ROTATION]:

```

```
        return_string += "\nDegrees: \t" + str(self._degrees)
    else:
        return_string += "\n" + str(self._vector)

    return_string += "\nSpeed: \t" + str(self._speed)

    return return_string
```


CommandToMAVLink.py

```
"""
Contains methods that convert directions to actionable MAVLink commands
for use with droneAPI
"""

__author__ = 'Kirby'

from numpy import multiply
from numpy.linalg import norm
from time import time, sleep
from geopy.distance import vincenty
from droneapi.lib import VehicleMode
from pymavlink import mavutil
from constants import command_wait

def change_mode(new_mode, vehicle, api):
    """
    Repeatedly attempts to change the mode of the vehicle until the
    desired mode is achieved.

    :param new_mode: Name of the mode to switch to
    :param vehicle: A droneAPI vehicle object whose mode is to be
    changed
    :param api: A droneAPI instance that can be monitored for exit
    requests
    """
    while vehicle.mode.name != VehicleMode(new_mode).name and not
    api.exit:
        vehicle.mode = VehicleMode(new_mode)
        vehicle.flush()
        sleep(command_wait)

def move_to(N, E, D, vehicle, api):
    """
    Relative move. Switches to GUIDED mode to make the move and
    switches back to LOITER mode after the move is
    completed.

    :param N: North movement component in m
    :param E: East movement component in m
    :param D: Down movement component in m
    :param vehicle: A droneAPI vehicle object to be moved
    :param api: A droneAPI instance that can be monitored for exit
    requests
    """
    move_method = "VEL"

    change_mode("GUIDED", vehicle, api)

    if move_method == "VEL":
        # TODO: Dynamically pass in velocity. It's currently set to 0.5
        vel_move(N, E, D, 0.5, vehicle, api)
```

```

elif move_method == "POS":
    pos_move(N, E, D, vehicle, api)
else:
    change_mode("LOITER", vehicle, api)
    raise ValueError("Move method must be either velocity (VEL) or
position (POS) based.")

# IMPORTANT NOTE: SWITCHING TO LOITER MODE SETS THE VELOCITY BACK
TO ZERO
# set mode to loiter mode
change_mode("LOITER", vehicle, api)

def vel_move(N, E, D, speed, vehicle, api):
    """
    Move accomplished by setting the velocity. This detection method
    relies on using time to figure out how far it's
    moved since the GPS we are using is too inaccurate for precise
    movements (our GPS' precision is ~ 1.5 m).

    :param N: North movement component in m
    :param E: East movement component in m
    :param D: Down movement component in m
    :param speed: The movement speed
    :param vehicle: A droneAPI vehicle object to be moved
    :param api: A droneAPI instance that can be monitored for exit
requests
    """

    # Calculate the component velocity vector magnitudes using the
overall speed and the position vector
    # The equation used is: vel vector =
(speed/pos_vector_magnitude)*pos_vector
    pos_vector = [[N], [E], [D]]
    pos_vector_magnitude = norm(pos_vector)
    if pos_vector_magnitude == 0:
        # All components are zero
        vel_vector = pos_vector
    else:
        vel_vector = multiply(speed/pos_vector_magnitude, pos_vector)

    # Movement speed capped at 1 m/s
    velocity_x = vel_vector[0][0] # North direction, in m/s
    velocity_y = vel_vector[1][0] # East direction, in m/s
    velocity_z = vel_vector[2][0] # Down direction, in m/s

    # Update delay in seconds. (This is how frequently the program
checks to see if it should have travelled an
# appropriate distance)
    update_delay = 0.01

    # Predicted travel time, distance (as the crow flies) / speed
    travel_time = norm(pos_vector) / speed

    msg = vehicle.message_factory.set_position_target_local_ned_encode(
0, #
time_boot_ms (not used)

```

```

target system, target component                                0, 0, #
mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame                0x01C7, #
type_mask (ignore pos | ignore acc)                          0, 0, 0, # x,
y, z positions (not used)                                    velocity_x,
velocity_y, velocity_z, # x, y, z velocity in m/s           0, 0, 0, # x,
y, z acceleration (not used)                                0, 0) #
yaw, yaw_rate (not used)

# send command to vehicle
vehicle.send_mavlink(msg)
vehicle.flush()

# Initial time
t0 = time()

while time() - t0 < travel_time and not api.exit:
    sleep(update_delay)

def pos_move(N, E, D, vehicle, api):
    """
    Move accomplished by setting the velocity. Currently this method
    waits to complete until the velocity reaches close
    to 0.

    :param N: North movement component in m
    :param E: East movement component in m
    :param D: Down movement component in m
    :param vehicle: A droneAPI vehicle object to be moved
    :param api: A droneAPI instance that can be monitored for exit
    requests
    """

    # The home location appears to be stored as the 0th waypoint
    home_location = vehicle.commands[0]

    geo_home = (home_location.lat, home_location.long)

    # Get distance from home
    Ndist = vincenty((vehicle.location.lat, home_location.long),
geo_home).meters
    Edist = vincenty((home_location.lat, vehicle.location.long),
geo_home).meters

    # Correct for the sign of the distance
    if vehicle.location.lat < home_location.lat:
        Ndist = -Ndist
    if vehicle.location.long < home_location.long:

```

```

    Edist = -Edist

    Nloc = Ndist + N
    Eloc = Edist + E

    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0, #
        time_boot_ms (not used)
        0, 0, #
        target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        0x01F8, #
        type_mask (ignore vel | ignore acc)
        Nloc,
        Eloc, D, # x, y, z positions in m
        0, 0, 0, #
        x, y, z velocity in m/s (not used)
        0, 0, 0, #
        x, y, z acceleration (not used)
        0, 0) #
    yaw, yaw_rate (not used)
    # send command to vehicle
    vehicle.send_mavlink(msg)
    vehicle.flush()

    # TODO: Put a conditional check here. I can think of three
possibilities:
    #     1) Use distance calculated via gps
    #     2) Use distance calculated via velocity and time
    #     3) Wait until velocity is about 0
    while vehicle.velocity.vx > 0.01 and vehicle.velocity.vy > 0.01 and
vehicle.velocity.vz > 0.01 and not api.exit:
        sleep(1)

```

LogFlight.py

```
"""
Contains methods used to log important flight information along with
images from the landing stage
"""

__author__ = 'Kirby'

import datetime, errno, subprocess
from os.path import join, dirname
from os import makedirs
from cv2 import rectangle
import droneapi.lib
from configure_path import SYSTEM
from DetectObject import save_image, overlay_targets
from EdisonSDCardSetup import setup_sd_card

# Creates a csv file with the specified header in the specified
location of the form log_CURRENT-DATETIME.csv
# Opens the file in append mode
# Returns the log_file
def setup_log_file():
    if SYSTEM == "Edison":
        sd_directory = setup_sd_card()
        log_folder = join(sd_directory, "Logs")

    elif SYSTEM == "Kirby's Mac":
        log_folder =
"/Users/Kirby/Desktop/Senior_Design/SVN/sw/branches/UAVino-
1.0.0/UAVino_Python_Code/UAV_Logging/Logs"

    else:
        raise "Error: no SYSTEM set or unsupported SYSTEM: " + SYSTEM

    # create a new folder for each specific log entry
    log_folder = join(log_folder, datetime.datetime.now().strftime("%Y-
%m-%dT%H_%M_%S_%f"))
    try:
        makedirs(log_folder, 0755)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise

    # double checks that the log folder now exists
    if subprocess.call(["ls", log_folder]):
        raise "Error: Log folder not found at this location: " +
log_folder
    else:
        print ("Logging to " + str(log_folder))

    csv_header = "Targets Detected, Command Kind, Frame of Reference, X
distance, Y distance, Z distance, Degrees, Speed, " \
"Latitude, Longitude, Altitude (meters), Altitude
Relative?, Pitch (radians), Roll (radians), " \
"Yaw (radians), Ground Speed (m/s), Flight Mode,
Time, Date"
```

```

    filename = "log_" + datetime.datetime.now().strftime("%Y-%m-
%dT%H_%M_%S_%f") + ".csv"
    file_path = join(log_folder, filename)
    log_file = open(file_path, "a+")
    log_file.write(csv_header + "\n")
    return log_file

def close_log_file(log_file):
    log_file.close()

def log_flight_info(log_file, drone, targets_detected=None,
command=None):
    drone.flush()

    # get value of variables to log
    if command is not None:
        command_kind = command.command_kind
        ref_frame = command.frame
        x_dist = command.get_vector.x
        y_dist = command.get_vector.y
        z_dist = command.get_vector.z
        degrees = command.degrees
        speed = command.speed

    else:
        command_kind = ""
        ref_frame = ""
        x_dist = ""
        y_dist = ""
        z_dist = ""
        degrees = ""
        speed = ""

    latitude = drone.location.lat
    longitude = drone.location.lon
    altitude = drone.location.alt
    relative = drone.location.is_relative
    pitch = drone.attitude.pitch
    roll = drone.attitude.roll
    yaw = drone.attitude.yaw
    ground_speed = drone.groundspeed
    flight_mode = drone.mode.name
    time = datetime.datetime.now().time()
    date = datetime.date.today()

    # This is what controls the order of the csv file, make sure it
    matches up with the csv header string
    log_vars = [targets_detected, command_kind, ref_frame, x_dist,
y_dist, z_dist, degrees, speed,
                latitude, longitude, altitude, relative, pitch, roll,
yaw, ground_speed, flight_mode, time, date]

    for var in log_vars:
        log_file.write(str(var))

```

```

    if var != log_vars[-1]:
        log_file.write(",      ")
    else:
        log_file.write("\n")

# img_suffix should be a unique identifier (0, 1, 2, ... is easiest)
# logs an unmodified image and calls log flight info
def log_landing_stage(log_file, drone, image, targets_detected,
command, img_suffix):

    image_file_name = "landing_img_" + str(img_suffix) + ".jpg"
    image_file = join(dirname(log_file.name), image_file_name)
    save_image(image, image_file)

    # save flight details for context
    log_flight_info(log_file, drone, targets_detected, command)

# log two images, one with the object drawn on and one without, then
calls log_flight_info
# These images are used for debugging by a human, and further training
a classifier, respectively
def log_landing_stage_for_training(log_file, drone, image,
detected_targets, command, img_suffix):
    image_file_name = "landing_img_" + str(img_suffix) + ".jpg"
    image_file = join(dirname(log_file.name), image_file_name)
    save_image(image, image_file)

    overlaid_image = overlay_targets(image, detected_targets)
    pic_size = overlaid_image.shape
    pic_height = pic_size[0]
    pic_width = pic_size[1]
    rectangle(overlaid_image, (int(pic_width/2 - 40), int(pic_height/2
- 40)),
                (int(pic_width/2 + 40), int(pic_height/2 + 40)), (255, 0,
0), 8)

    image_file_name = "overlaid_landing_img_" + str(img_suffix) +
".jpg"
    image_file = join(dirname(log_file.name), image_file_name)
    save_image(overlaid_image, image_file)

    log_flight_info(log_file, drone, len(detected_targets), command)

```

failsafes.py

```
"""
Contains failsafes and safety checks used for UAVino

Note: This functionality has not been fully tested. An error was
occurring when trying to use these methods.
I don't recall what the fix is but it should be easy
"""

__author__ = 'Kirby'

import droneapi.lib
import sys

# Monitored Attributes
# TODO: Consider adding channels 1-4 (roll, pitch, yaw, thrust) to
monitoring
monitored_attributes = ['mode']

def start_external_command_monitor(vehicle):
    """
    Stops the script whenever the drone receives a command from a
    different external source.

    This method prevents multiple control sources from giving the drone
    commands at the same time. This function is
    primarily needed to accommodate RC commands and commands from
    another GCS.

    :param vehicle: The droneAPI vehicle instance to be monitored
    """
    for attr in monitored_attributes:
        vehicle.add_attribute_observer(attr, lambda: sys.exit("Exiting:
Detected external input from source other than "
"DroneKit"))

def stop_external_command_monitor(vehicle):
    """
    Removes the attribute observers set by
    start_external_command_monitor

    :param vehicle: The droneAPI vehicle instance being monitored
    """
    for attr in monitored_attributes:
        vehicle.remove_attribute_observer(attr, lambda:
sys.exit("Exiting: Detected external input from source other "
"than
DroneKit"))
```


pixel distance conversions.py

```
"""
Contains a method that can be used to estimate the distance away a
picture was taken from given the dimensions
of an object and the size in pixels of the object in the picture
"""

__author__ = 'Kirby'

def pix_to_alt(object_width, object_height, detected_width,
detected_height, GSD_constant_x, GSD_constant_y):
    """
    Estimates the altitude a picture was taken from given the width and
height of an object in both ground units
(such as cm, m, feet, etc.) and in pixels using the Ground Sample
Distance constant for an image.

    :param object_width: The width of the object to be detected in
ground units (such as cm, m, feet, etc.)
    :param object_height: The height of the object to be detected in
ground units
    :param detected_width: The detected width of the object in pixels
    :param detected_height: The detected height of the object in pixels
    :param GSD_constant: The Ground Sample Distance constant for the
image
    :return: average estimated altitude
    """

    dist_per_pixel_x = object_width/detected_width
    dist_per_pixel_y = object_height/detected_height

    alt_est_x = dist_per_pixel_x / GSD_constant_x
    alt_est_y = dist_per_pixel_y / GSD_constant_y

    alt = (alt_est_x + alt_est_y) / 2

    return alt
```

centering_demo.py

```
"""
Attempts to center a drone over the docking station

Currently this demo only centers the drone horizontally. It makes no
vertical movements or yaw rotation
"""

__author__ = 'Kirby'

import sys
sys.path.append("/home/root/UAVino_Python_Code")

import cv
import cv2
from time import sleep
from os.path import join, dirname
import droneapi.lib
from pymavlink import mavutil
from CommandToMAVLink import move_to
from DetectObject import detect_targets, overlay_targets
from GenerateDirections import get_next_command
from UAV_Logging.LogFlight import setup_log_file, close_log_file,
log_landing_stage, log_landing_stage_for_training
from failsafes import start_external_command_monitor,
stop_external_command_monitor
from constants import GSD_constant_x, GSD_constant_y

api = local_connect()
vehicle = api.get_vehicles()[0]
commands = vehicle.commands

# Monitor commands from another control source (RC or another GCS) in
order to prevent commands from multiple control
# sources at the same time
#start_external_command_monitor(vehicle)

target_cascade_name = "docking5.xml"

# load classifier
target_cascade = cv2.CascadeClassifier(target_cascade_name)
if not target_cascade:
    raise RuntimeError('Could not load cascade classifier: ' +
target_cascade_name)

# read the video stream
capture = cv2.VideoCapture(-1)
if not capture.isOpened():
    raise RuntimeError("Could not open video stream")
pic_width = capture.get(cv.CV_CAP_PROP_FRAME_WIDTH)
pic_height = capture.get(cv.CV_CAP_PROP_FRAME_HEIGHT)

logfile = setup_log_file()
```

```

counter = 0

while not api.exit:
    ret_val, frame = capture.read()
    current_altitude = vehicle.location.alt
    current_attitude = vehicle.attitude

    if ret_val:
        detected_targets = detect_targets(frame, target_cascade)

        # Currently we handle multiple targets being detected by simply
        # not generating a command
        if len(detected_targets) == 1:
            for (x, y, w, h) in detected_targets:
                next_command = get_next_command(x+w/2, y+h/2,
pic_width, pic_height, current_altitude,
                                                GSD_constant_x,
GSD_constant_y)

                log_landing_stage_for_training(logfile, vehicle, frame,
detected_targets, next_command, counter)

                # Transform the command to the NED frame so it can
                # guide the octo, and send it to the octo

                next_command.transform_body_to_NED_yaw_pitch_roll_in_rads(current_attit
ude.yaw,

                current_attitude.pitch,

                current_attitude.roll)

                move_to(next_command.get_vector.x,
next_command.get_vector.y, 0, vehicle, api)

            else:
                log_landing_stage_for_training(logfile, vehicle, frame,
detected_targets, None, counter)

        else:
            print "No captured frame"
            log_landing_stage_for_training(logfile, vehicle, frame, None,
None, counter)

        counter += 1

        print "Completed one move"
        #sleep(0.5)

close_log_file(logfile)
#stop_external_command_monitor(vehicle)

```

capture_images.py

```
"""
Continuously writes images captured from a camera to a file using
OpenCV

This technique is useful for gathering positive and negative training
images to train or improve a classifier
"""

__author__ = 'Kirby'

from cv2 import VideoCapture
from os.path import join, dirname
from time import sleep
from DetectObject import save_image
from UAV_Logging.LogFlight import setup_log_file

if __name__ == "__main__":

    capture = VideoCapture()
    capture.open(-1)
    if capture:
        log_file = setup_log_file()
        count = 0
        while True:
            ret, frame = capture.read()
            if ret:
                # save with a 6-digit suffix
                image_file_name = "landing_img_" +
"{0:0>6}".format(str(count)) + ".jpg"
                count += 1
                image_file = join(dirname(log_file.name),
image_file_name)
                save_image(frame, image_file)

            else:
                print "No captured frame"

            sleep(0.5)
```

landing_algorithm_demo.py

```
"""
Runs a demo of the vision recognition used for UAVino.

This program is meant to be run on a laptop or desktop. It displays the
detected targets (using a Haar Cascade
classifier) and the center of the camera frame. This demo is great for
presentations and demoing the vision
capabilities.
"""

__author__ = 'Kirby'

import cv
import cv2
from DetectObject import detect_targets, display_targets
from GenerateDirections import get_next_command
from pixel_distance_conversions import pix_to_alt
from constants import GSD_constant_x, GSD_constant_y

if __name__ == '__main__':
    target_cascade_name = "docking5.xml"
    target_width = 15.5      # in cm
    target_height = 17      # in cm

    # TODO: Calculate altitude
    current_altitude = 130

    # load classifier
    target_cascade = cv2.CascadeClassifier(target_cascade_name)
    if not target_cascade:
        raise RuntimeError('Could not load cascade classifier: ' +
target_cascade_name)

    # read the video stream
    capture = cv2.VideoCapture(-1)
    pic_width = capture.get(cv.CV_CAP_PROP_FRAME_WIDTH)
    pic_height = capture.get(cv.CV_CAP_PROP_FRAME_HEIGHT)

    if capture:
        while True:
            ret_val, frame = capture.read()

            if ret_val:
                detected_targets = detect_targets(frame,
target_cascade)
                for (x, y, w, h) in detected_targets:

                    # Get estimated altitude from detected object
                    current_altitude = pix_to_alt(target_width,
target_height, w, h, GSD_constant_x, GSD_constant_y)
                    print "Estimated Altitude: " +
str(current_altitude)
```

```

        next_command = get_next_command(x+w/2, y+h/2,
pic_width, pic_height, current_altitude,
GSD_constant_x,
GSD_constant_y)
        print next_command
        print "\n\n"

        # mark the center of the image for easy visualization
when testing
        cv2.rectangle(frame, (int(pic_width/2 - 40),
int(pic_height/2 - 40)), (int(pic_width/2 + 40),
int(pic_height/2 + 40)), (255,0,0), 8)
        display_targets(frame, detected_targets)

    else:
        print "No captured frame"

    c = cv2.waitKey(0)
    if c == 'c':
        exit(0)

```

test_takeoff.py

```
"""
Causes the vehicle to takeoff in the simulator. Should not be used
outside of a simulator since it overrides
the RC channel input
"""

__author__ = 'Kirby'

import datetime, os
import droneapi.lib
from time import sleep
from pymavlink import mavutil

print "IMPORTANT: DO NOT RUN THIS PROGRAM UNLESS USING THE SIMULATOR.
IT OVERRIDES THE RC CHANNEL INPUT."
SIMULATING = True

api = local_connect()
vehicle = api.get_vehicles()[0]
commands = vehicle.commands

vehicle.mode = droneapi.lib.VehicleMode("STABILIZE")
if SIMULATING:
    # This command should only be run in a simulator since it overrides
    RC channel 3
    vehicle.channel_override = {"3" : vehicle.parameters['RC3_MIN']}
    vehicle.flush()
    sleep(1)
vehicle.armed = True
vehicle.flush()

while not vehicle.armed and not api.exit:
    print "Waiting for arming..."
    sleep(1)

vehicle.mode = droneapi.lib.VehicleMode("GUIDED")
vehicle.flush()

print "Taking off!"
if SIMULATING:
    # This command should only be run in a simulator since it overrides
    RC channel 3
    vehicle.channel_override = {"3" : vehicle.parameters['RC3_TRIM']}
    vehicle.flush()
alt = 50
vehicle.commands.takeoff(alt) # Take off to 20m height
while (vehicle.location.alt < (alt-.1) or vehicle.location.alt >
(alt+.1)) and not api.exit:
    sleep(1)
    vehicle.flush()

vehicle.mode = droneapi.lib.VehicleMode("LOITER")
vehicle.flush()
```


test_alt_descent.py

```
"""
Causes the vehicle to change altitude

Simply changing the altitude ended up causing the octocopter to also
change its yaw. To avoid this, use the
SET_POSITION_LOCAL_NED Mavlink message instead
"""

__author__ = 'Kirby'

import datetime, os
import droneapi.lib
from time import sleep
from pymavlink import mavutil

api = local_connect()
vehicle = api.get_vehicles()[0]
commands = vehicle.commands

# TODO: CHECK TO SEE IF THIS LOOP INTERFERES WITH CONTROL VIA THE
# MISSION PLANNER, RADIO CONTROL, OR FAIL-SAFES
while vehicle.mode.name != droneapi.lib.VehicleMode("GUIDED").name:
    vehicle.mode = droneapi.lib.VehicleMode("GUIDED")
    vehicle.flush()
    print "Switching modes"
    sleep(1)

target_location = vehicle.location
start_location_alt = target_location.alt # used for debugging

if vehicle.location.alt > 10.0:
    target_location.alt = target_location.alt + 5
    commands.goto(target_location)
    vehicle.flush()
    print "Moved up five meters from " + str(start_location_alt)
    print "to " + str(target_location)
    print "Vehicle now at " + str(vehicle.location)

elif 2.0 < vehicle.location.alt <= 10.0:
    target_location.alt = target_location.alt + 5
    commands.goto(target_location)
    vehicle.flush()
    print "Moved up five meters from " + str(start_location_alt)
    print "to " + str(target_location)
    print "Vehicle now at " + str(vehicle.location)

else:
    print "Altitude at less than a meter, too close to ground"
    exit(0)

while (vehicle.location.alt - target_location.alt < -.1 or
vehicle.location.alt - target_location.alt > .1) \
    and not api.exit:
    print "Ascending"
    sleep(1)
```

```
vehicle.mode = droneapi.lib.VehicleMode("LOITER")  
vehicle.flush()
```

test alt descent v2.py

```
"""
Causes the vehicle to change altitude without changing yaw
"""

__author__ = 'Kirby'

import datetime, os
import droneapi.lib
from time import sleep
from pymavlink import mavutil

api = local_connect()
vehicle = api.get_vehicles()[0]
commands = vehicle.commands

while vehicle.mode.name != droneapi.lib.VehicleMode("GUIDED").name:
    vehicle.mode = droneapi.lib.VehicleMode("GUIDED")
    vehicle.flush()
    print "Switching modes"
    sleep(1)

target_alt = vehicle.location.alt
start_alt = target_alt # used for debugging

def change_alt(h):
    # Note: all distances are in relation to the home location
    pos_z = -h # Down direction, in m
    if h > 0:
        vel_z = -0.5
    else:
        vel_z = 0.5

    # Note: Currently only velocity, position, and acceleration can be
    # masked as a whole. You can't simply mask out one
    # axis.
    MASK_XYZ_YAW_YAW_RATE_ONLY = 0xF3F8
    MASK_XYZ_ONLY = 0xFFF8
    MASK_Z_ONLY = 0xFFFB
    MASK_VXVYVZ_ONLY = 0xFFC7
    MASK_POS_VEL_XYZ = 0xFFC0

    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0, #
        time boot ms (not used)
        0, 0, #
        target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        MASK_VXVYVZ_ONLY, # type_mask
        pos_z, # z positions in m, x and y ignored
        0, 0,
        vel_z, # x, y, z velocity in m/s
        0, 0, 0, #
```

```

x, y, z acceleration (not used)                                0, 0)    #
yaw, yaw rate (not used)
    # send command to vehicle
    vehicle.send_mavlink(msg)
    vehicle.flush()

if vehicle.location.alt > 10.0:
    target_alt = target_alt + 5
    change_alt(target_alt)
    vehicle.flush()
    print "Moved up five meters from " + str(start_alt)
    print "to " + str(target_alt)
    print "Vehicle now at " + str(vehicle.location)

elif 2.0 < vehicle.location.alt <= 10.0:
    target_alt = target_alt + 5
    change_alt(target_alt)
    vehicle.flush()
    print "Moved up five meters from " + str(start_alt)
    print "to " + str(target_alt)
    print "Vehicle now at " + str(vehicle.location)

else:
    print "Altitude at less than a meter, too close to ground"
    exit(0)

while (vehicle.location.alt - target_alt < -.1) and not api.exit:
    print "Ascending"
    sleep(0.1)

vehicle.mode = droneapi.lib.VehicleMode("LOITER")
vehicle.flush()

```

test_pos_move.py

```
"""
Causes the octocopter to move using the position arguments of the
SET_POSITION_LOCAL_NED Mavlink message
"""

__author__ = 'Kirby'

from time import sleep
from droneapi.lib import VehicleMode, Location
from pymavlink import mavutil

api = local_connect()
vehicle = api.get_vehicles()[0]
commands = vehicle.commands
vehicle.mode = VehicleMode("GUIDED")

# Go to 10 meters above home to start
height = 10;
N_dist = 0;
E_dist = 0;

def move(x, y, h):
    # Note: all distances are in relation to the home location
    pos_x = x # North direction, in m
    pos_y = y # East direction, in m
    pos_z = -h # Down direction, in m

    MASK_XYZ_YAW_YAW_RATE_ONLY = 0xF3F8
    MASK_XYZ_ONLY = 0xFFF8
    MASK_Z_ONLY = 0xFFB

    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0, #
        time_boot_ms (not used) #
        0, 0, #
        target system, target component
        mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
        MASK_XYZ_ONLY, # type_mask (ignore vel | ignore acc)
        pos_x,
        pos_y, pos_z, # x, y, z positions in m
        0, 0, 0, #
        x, y, z velocity in m/s (not used)
        0, 0, 0, #
        x, y, z acceleration (not used)
        0, 0) #
        yaw, yaw_rate (not used)
        # send command to vehicle
        vehicle.send_mavlink(msg)
        vehicle.flush()

move(N_dist, E_dist, height)

while (vehicle.location.alt < (height - .1) or vehicle.location.alt >
```

```
(height + .1) and not api.exit:
    sleep(1)

print "Reached altitude of " + str(vehicle.location.alt)

# Note: all distances are in relation to the home location
N_dist = 10    # North direction, in m
E_dist = 0     # East direction, in m
height = 10

move(N_dist, E_dist, height)

for i in range(0,20):
    sleep(1)

# set mode to loiter mode
vehicle.mode = VehicleMode("LOITER")
print "Position move completed"
```

test vel move.py

```
"""
Causes the octocopter to move using the velocity arguments of the
SET_POSITION_LOCAL_NED Mavlink message
"""

__author__ = 'Kirby'

from time import time, sleep
from droneapi.lib import VehicleMode, Location
from pymavlink import mavutil

api = local_connect()
vehicle = api.get_vehicles()[0]
commands = vehicle.commands
vehicle.mode = VehicleMode("GUIDED")

vel_update_rate = 0.1 # in s or about 1/hz, in this case it's about 10
hz
t0 = time()

velocity_x = 1 # North direction, in m/s, in this case 1 m/s * .1 s
= 10cm travelled North
velocity_y = 0 # East direction, in m/s
velocity_z = 0 # Down direction, in m/s

msg = vehicle.message_factory.set_position_target_local_ned_encode(
    0, #
    time_boot_ms (not used)
    0, 0, #
    target system, target component
    mavutil.mavlink.MAV_FRAME_LOCAL_NED, # frame
    0x01C7, #
    type mask (ignore pos | ignore acc)
    0, 0, 0, # x,
    y, z positions (not used)
    velocity_x,
    velocity_y, velocity_z, # x, y, z velocity in m/s
    0, 0, 0, # x,
    y, z acceleration (not used)
    0, 0) #
    yaw, yaw_rate (not used)
# send command to vehicle
vehicle.send_mavlink(msg)

# Currently the velocity update rate doesn't really do anything.
Eventually it will be used to prevent the Edison from
# spamming commands to the PixHawk
while (time() - t0) < vel_update_rate:
    sleep(1)
    vehicle.flush()

# IMPORTANT NOTE: SLEEPING FOR MORE THAN A COUPLE SECONDS CAUSES A
CESSATION OF THE HEARTBEAT MESSAGES WHICH TRIGGERS
```

```
#           AN IMMEDIATE RETURN TO LAUNCH. INSTEAD, SLEEP SHOULD BE PUT IN
A LOOP TO ALLOW FOR CONTINUOUS HEARTBEAT MESSAGES
#           TO BE SENT
print "Pausing"
for i in range(0, 10):
    sleep(.5)

# IMPORTANT NOTE: SWITCHING TO LOITER MODE SETS THE VELOCITY BACK TO
ZERO
# set mode to loiter mode
vehicle.mode = VehicleMode("LOITER")
print "Velocity move completed"
```


APPENDIX M

Senior Design Conference Slides

 SANTA CLARA UNIVERSITY

UAVino



Matthew Belesiu
Aaron Chung
Nathan Carlson

Phillip Coyle
Kirby Linvill
Megan Peekema

www.sclsu.edu SCHOOL OF ENGINEERING 

 SANTA CLARA UNIVERSITY

Presentation Overview

- UAVino Introduction
- How UAVino Works
- Key Innovations
- Results
- Conclusion

www.sclsu.edu SCHOOL OF ENGINEERING 




 SANTA CLARA UNIVERSITY

What is UAVino?

- Drone-based crop inspection system
 - Real-world vineyard customer
- Long Term Vision
 - Implemented as a service to local customers

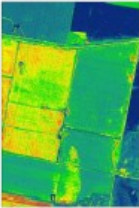



www.sclsu.edu SCHOOL OF ENGINEERING 

 SANTA CLARA UNIVERSITY

Current Crop Inspection Methods

- Sensor Networks
 - On-site, under owner control
 - Labor intensive
- Satellite Imagery
 - Cost-effective
 - Weather dependent
- Drones
 - Both on-site and cost-effective



www.sclsu.edu SCHOOL OF ENGINEERING 

 SANTA CLARA UNIVERSITY

Problem Statement

- Systems Level
 - Develop a low-cost drone crop inspection system
 - Allow for autonomous docking and recharging
- Business Level
 - Deliver value to real-world customer

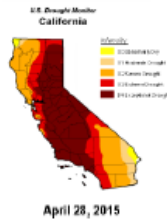


www.sclsu.edu SCHOOL OF ENGINEERING 



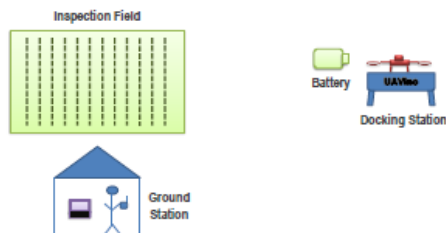
Social Impact

- California Drought
 - Agriculture uses 80% of state's water
- Real World Application
 - State produces 90% of nation's wine



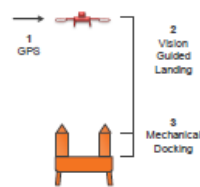
Our Solution

- Technology
 - Multispectral imaging camera
 - Docking and recharging capability
- How it works...

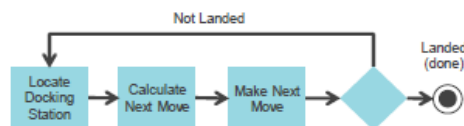


Key Subsystems

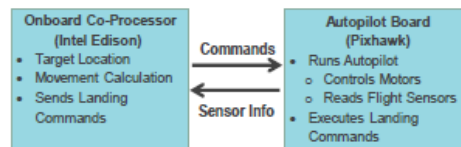
- Autonomous Docking
 - GPS return to launch
 - Vision guided landing
 - Mechanical docking
- Recharging Capability
- Post Processing



Vision Guided Landing Algorithm

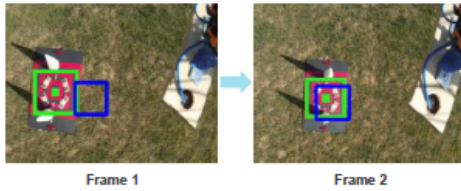


Vision Guided Landing Hardware

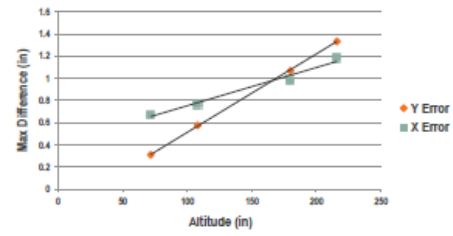




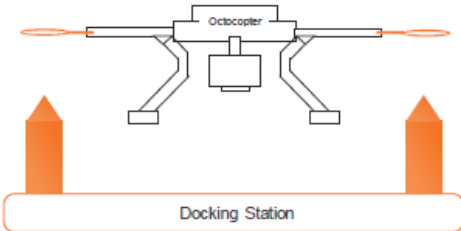
Docking Station Identification



Docking Station Identification

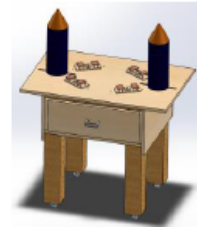


Autonomous Docking Station



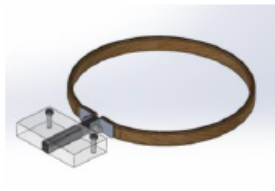
Docking Station Components

- Cones
- Drawer/Platform
- Charging Plates



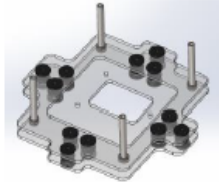
Mechanical Docking Components

- Strong
- Lightweight
- Precise



Camera Mounting

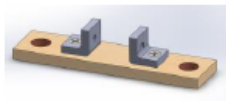
- Multispectral Camera
- Vibration Isolation
- Lightweight





Electrical Contact Feet

- Copper Plugs
- Brackets
- Added Stability

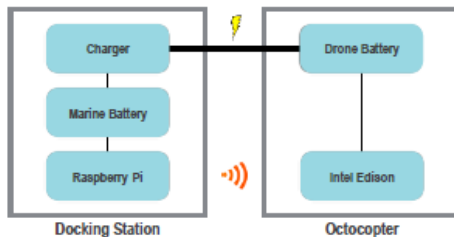


Recharging Capability

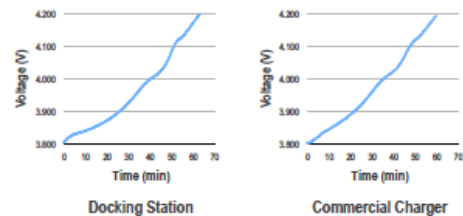
- Precise Landing
- Direct Contact Charging
- Charger Housed in Docking Station



Charging System Schematic



Charging System Performance

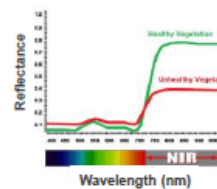


Post-Processing

- Examination of Images
- Photo-mosaic
- Vegetation Index



Normalized Difference Vegetation Index



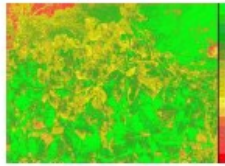
$$NDVI = \frac{(NIR - Red)}{(NIR + Red)}$$



Normalized Difference Vegetation Index



Raw Multispectral Image

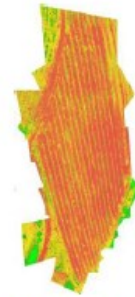


NDVI Processed Image



Long-Term Monitoring

- Aggregation of Data
- Standardization
- Health Tracking



Current State of the Project

- Short Term
 - Foundation for autonomous operation
 - Tuning landing algorithm
 - Implement flight management algorithm
- Future Functionalities
 - Infrared crop imaging
 - Multiple drones



Conclusions

- Agricultural Inspection System
- Real World Customer
- Key Subsystems
 - Autonomous Docking
 - Recharging Capability
 - Post Processing



Thank you!

We welcome any feedback and questions.

