A Sensor System for Autonomous UAV Landing

A senior design project submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Electrical Engineering

at

Harvard University

by

Ivan Alexis Cisneros

S.B. Degree Candidate in Electrical Engineering

Submitted by . . . .

Ivan Alexis Cisneros
December 8, 2016

Supervised and Evaluated by . . . .

Scott Kuindersma
Assistant Professor of Engineering and Computer Science
Faculty Advisor

Harvard University School of Engineering and Applied Sciences
Cambridge, MA
December 2016

A Sensor System for Autonomous UAV Landing

by

Ivan Alexis Cisneros

Submitted to Harvard University School of Engineering and Applied Sciences
on December 8, 2016, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Electrical Engineering

Abstract

Unmanned Aerial Vehicles (UAVs) have recently surged in popularity and are now seen as viable tools for use in commercial delivery, search and rescue operations, and planetary exploration. A challenge in utilizing UAVs for these purposes is the limited battery capacity: an average-sized drone has 25 minutes of flight time, which results in limited flight range and payload capacity. Implementing an array of charging stations would extend UAV flight range, but GPS navigation would not provide the resolution necessary to reliably find and land on these stations. This project focuses on developing a vision-based sensor system that would provide onboard sensing mechanisms and path planning in order to accurately detect and land on a charging pad. This solution aims to be lightweight, unobtrusive, self-contained, and more accurate than relying on GPS and IMU alone. This proof of concept sensor system can be adapted to the form factor of mission-specific UAVs.

Faculty Advisor: Scott Kuindersma
Title: Assistant Professor of Engineering and Computer Science

Contents

List of Figures

List of Tables

Introduction

Unmanned Aerial Vehicles (UAVs) – colloquially known as "drones" – have risen in popularity amongst hobbyists, filmmakers, and technology enthusiasts in recent years due to advancements in miniaturization brought about by the mobile technologies sector. UAVs are now affordable and powerful enough to be used for non-recreational applications; various new uses for autonomous unmanned aerial vehicles are currently being developed. As has been shown with Amazon's pilot project "Amazon Prime Air", one function for this new technology is the implementation of rapid short-range commercial delivery[1]. While there are a myriad of applications for autonomous drone technology, the biggest limiting factor is battery life. Because most UAVs are fully electric, they must carry heavy lithium ion polymer batteries. But since UAVs are actively fighting gravity to stay airborne, they require the optimization of their thrust-to-weight ratio, meaning that there are diminishing returns in increasing battery size.

Creating a UAV-specific charging infrastructure similar to the infrastructure necessary for electric vehicles can be a means to solve this issue. Creating a network of charging or battery swapping platforms will allow for autonomous drones to stop and "refuel" – without having to return to their origin – in order to fly in trajectories beyond the limits of their battery's charge.

[1]More details found here: http://www.amazon.com/b?node=8037720011

1.1 Problem

Various companies and institutions are exploring different applications for autonomous UAVs that would increase efficiency and/or avoid putting people in dangerous situations. These include:

- Short and medium distance commercial delivery
- Search and rescue
- Real-time traffic monitoring
- High resolution mapping and surveying (both terrestrial and non-terrestrial)

An obstacle towards the realization of these applications is the poor flight range and battery life of a typical UAV. An average sized multi-rotor can achieve between 20 and 25 minutes of flight time without additional payload weight [1] , and with a nominal speed of 10 m/s, this translates into a flight distance range of 12 km (7.45 miles) to 15 km (9.32 miles). Amazon – a company which is currently exploring UAV delivery applications through their Amazon Prime Air division – hopes to achieve air delivery service within a 16 km (10 miles) radius of a distribution warehouse [2]. This means that for delivery applications, the limited flight range means that there is a limited service area, and perhaps extra costs in creating more distribution centers. Smaller retailers would not be able to implement additional distribution centers, and so this would limit the amount of retailers than can provide this rapid delivery service. For the non-delivery applications, such as search and rescue, the limited flight time would mean that the search and rescue missions would be interrupted or require a large amount of UAVs to be constantly swapped into usage. Thus, relying solely on the charge of a single UAV is prohibitive towards the efficient implementation of the above mentioned applications.

A cause of this limitation is the relatively low energy density of lithium ion polymer (LiPo) batteries. While LiPo batteries are the preferred power source for UAVs (compared to other battery variants) because of their low weight, quick discharge time, and relatively high energy capacity, they are still heavily limited in the flight distance that they can provide. Additionally, there is an inverse relationship between flight time and battery weight. Using the following figure, we can calculate a "Flight time cost":

$$FTC = \frac{1min}{71grams} \tag{1.1}$$

a finding which suggests that all UAV configurations have diminishing returns with regards to battery size (the extra battery capacity does not outweigh the cost of the added weight).

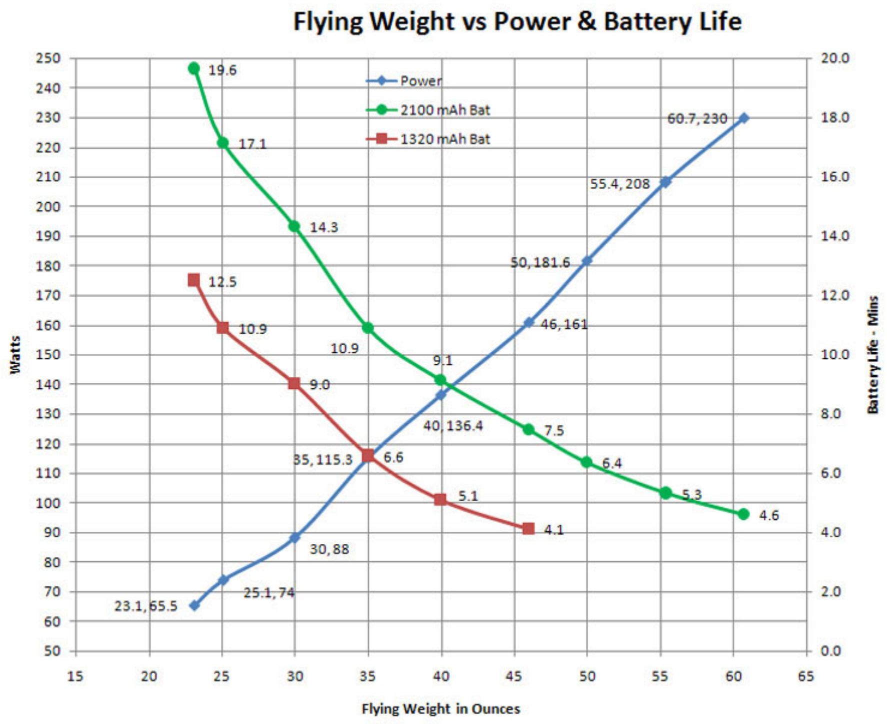


Figure 1-1: The relationship between the flying weight and time of flight of a UAV with batteries of two different capacities[3]

[3]Figure found here: https://code.google.com/archive/p/ro-4-copter/wikis/QuadcopterPerformance.wiki

1.2 Motivation

Electric vehicles are range-limited in a similar way and for the same reasons. In order to advance the adoption of electric vehicles, Tesla Motors created the "Tesla SuperCharger Network", which is a distributed network of high current DC power sources that allow Tesla vehicle owners to rapidly charge their vehicle while on a trip [3]. These charging stations are placed along commonly trafficked routes, and allow owners the ability to drive further than their vehicle's battery capacity would normally allow them to travel. In fact, multiple owners have driven across the continental United States by relying solely on this SuperCharger network. One can imagine that a similar, albeit closer range, system can be deployed for the specific use of UAVs. Virtually all autonomous UAVs have the ability to fly along planned trajectories by using low-cost GPS receivers and following GPS waypoints. As long as sufficient satellite signals can be accessed during the entire UAV flight path, GPS navigation techniques can offer consistent accuracy [4].

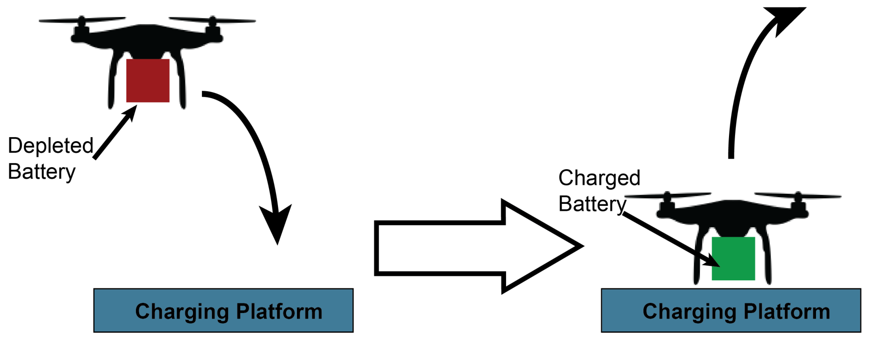


Figure 1-2: A UAV can locate a charging platform, land on it, charge, and continue on its route.

This UAV-specific charging network can be composed of several landing platforms that would charge the UAV, and allow it to continue on its trajectory to its final destination. If a GPS waypoint were placed at every charging platform location, the UAV would be able to find the closest one and navigate towards it. While civilian level GPS is fairly accurate, it can have a location offset of up to 5 meters [5], which means that locating a charging platform requires more than just a GPS coordinate. In order to land on a charging platform of moderate size the UAV would require localized information about the location of the platform and its relative position to the UAV. Onboard sensing would allow the autonomous UAV to react to unpredictable environments, and to overcome the limitations of GPS resolution.

1.3 Prior Art

There exists some work on the topic of autonomous UAV landing. A vision-based approach seems to be common because it allows the flexibility of landing in complex environments and in not relying on a GPS network if the need arises. Fixed winged UAVs, for example, can use an angled camera (forward facing) in order to identify potential landing strips by searching for areas of land that are relatively flat and devoid of hills [6].

At JPL, Johnson, Montgomery and Matthies have implemented a similar terrain-analyzing vision-system for multi-rotors, which identifies non-hazardous landing areas amongst hazardous terrain. In this case, a downward facing camera is used to identify the topology of the area and create a 3D terrain map for identifying the lowest and flattest areas [7] . This solution is specifically designed for use in planetary exploration, and so it relies on the assumption that very little human intervention is available.

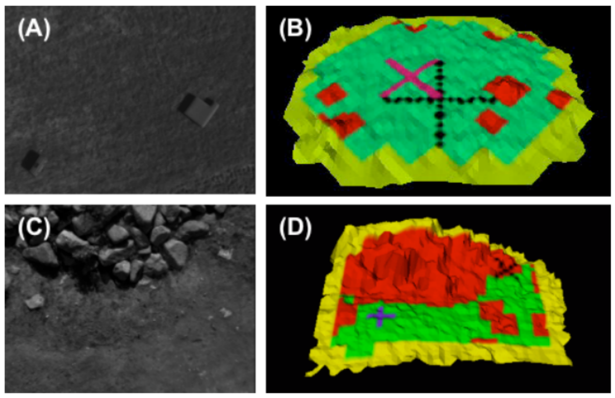


Figure 1-3: Visiual information from the UAV landing system developed at JPL.

For this project, limited human intervention is assumed; a landing platform has to be preemptively placed in an ideal location. For a multi-rotor, a landing platform with a large fiducial marker can be used to not only identify the specific area on which to land, but also allows the UAV to make estimates about its relative location and distance relative to the landing platform using preprogrammed information about the landing platform's size and the size of its representation in an frame of a video feed. Lange, Sunderhauf, Protzel, implemented such a system; their system uses a camera to identify a landing pad with a large "bull's-eye" marker and process this visual information to extract the multi-rotor's relative position and orientation [8]. They note that with a processing rate of 25 Hz they are able to stabilize the UAV's position over the center of the landing pad with a standard deviation of 3.8 cm and a maximum deviation of 23 cm over 5 minutes.

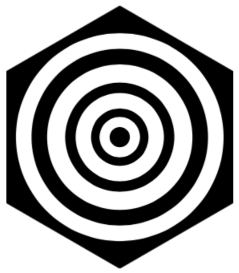


Figure 1-4: The landing marker used in the system developed by Lange et al.

But while they achieve accurate information about the lateral offset position of their UAV, they do not seem to investigate the recognition altitude or recognition lateral offset limitations of their system (they tested with a maximum altitude of 70cm and only with the UAV hovering directly above the landing pad). Additionally, because of the symmetry of their fiducial marker, it is not easy to obtain yaw angle estimates, which could potentially be important in a charging platform (if metal contacts are used, the UAV and charging platform must be aligned with the same yaw orientation).

1.4 Approach

Due to the robustness of a vision-based landing sensor system, we elected to develop on a computer vision platform as well, with specific attention paid to large recognition distances and the availability and accuracy of position and orientation information. Because we assume an ideal landing area, we can define the size and design of the landing platform as well and thus our design will also rely on a fiducial marker system to mark the area of the landing platform and provide all of the localization estimate information.

We thus design a self-contained sensor and landing system that is lightweight, vision-based, and mountable on average sized UAVs. We want this system to be generalizable and adaptable to different UAV platforms, so it must be as low weight and low power as possible.

1.5 Customer/Market

This sensor system is designed to be adaptable to different multi-rotor UAV platforms that are made for different uses. We designed with the applications that were considered in the motivation section in mind, and thus we can assume that distribution and logistics companies, emergency responders, and scientific institutions would be interested in incorporating this product and design into their specific autonomous UAV solutions.

Design

2.1 Design Scope and Assumptions

The idea of extending an autonomous UAV's flight range by using a distributed charging network is a big one with lots of interplaying technologies. To fully implement the idea, a number of solutions must be designed, namely:

- The best charging strategy
- Optimal placement distances and locations between each charging platform
- Optimization of flight paths so that the UAV makes minimal amount of stops
- UAV (and battery) identification so that charging is tailored correctly
- Platform identification and search strategies
- The control system for landing

We must therefore define the scope of this particular project in order to define the goals.

A process path for an autonomous UAV using this network may work like so:

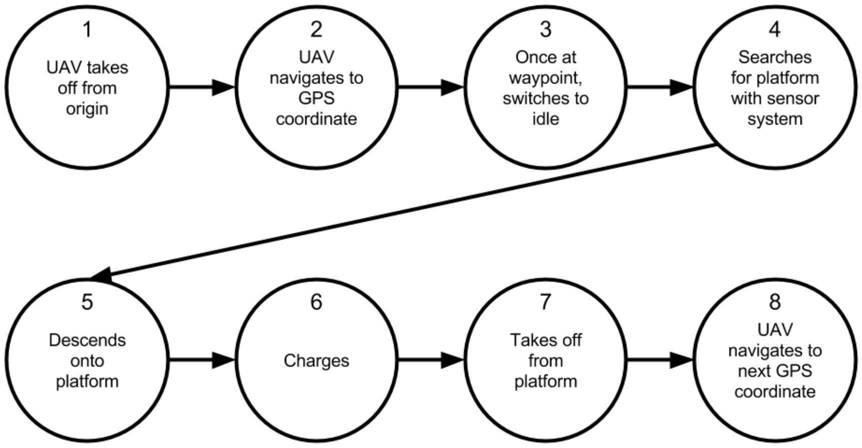

1
UAV takes off from origin
2
UAV navigates to GPS coordinate
3
Once at waypoint, switches to idle
4
Searches for platform with sensor system
5
Descends onto platform
6
Charges
7
Takes off from platform
8
UAV navigates to next GPS coordinate

Figure 2-1: A potential process path for an autonomous UAV using a charging network.

This project focuses primarily on the 4th state in this process path, namely how the UAV will identify the target landing platform and use this information.

A few assumptions that were made when defining the design goals:

- UAV is already at GPS waypoint, so it has a general idea of where the landing platform is located
- Laterally located within 5 meters
- Landing area is clear of obstacles
- Favorable weather conditions (not stormy)
- UAV will not be connected to a network (besides GPS) – the system will have to be self-contained

2.2 Design Goals

Design of this product focused primarily on the platform recognition via visual methods with specific attention paid to large recognition distances and the availability and accuracy of position and orientation information. The designed system must be a localized system that supplements GPS information so that an autonomous UAV can identify the landing platform and extend its functionality beyond the limitations of GPS and the sensors that the UAV already is expected to have. Additionally, in order to minimize the amount of extra payload weight it must also be lightweight and low power.

The following quantitative design criteria for the sensor system and landing area were established:

Table 2.1: Quantitative design criteria

	Target	**Unit**
Landing Area	0.6m x 0.6m	meter^2
Weight	< 600	gram
Current Draw	< 0.5	ampere
Physical Area	11.4cm x 14cm	centimeter^2
Height	9.5	centimeter
Max Height of Recognition	10	meter
Max Lateral Offset of Recognition	5	meter

The decisions for each of the criteria for the sensor system and landing area were made for the following reasons:

- Landing area: big enough to accommodate an average sized drone, but small enough to not be bulky
- Weight: the UAV's max payload is 2400g (including the UAV weight). UAV weight is 1080g, and so 1320g are left for use, but want to limit the flight time costs as much as possible. A 600g limit results in: Flight time cost = 8.5mins
- Current Draw: minimal power draw, while still allowing for sufficiently fast processing
- Physical area: restricted by the bottom of the UAV's frame

- Height: restricted by the UAV's vertical clearance
- Max lateral offset of recognition: from the expected GPS accuracy error
- Max height of recognition: the necessary height to achieve the max lateral offset condition

2.3 Design Approach

The two primary qualitative design goals for this sensor system were that it be self-contained and that it be generalizable so that it works with different UAV systems.

A self-contained system would be robust because it would be able to work in unpredictable environments and not rely on the unpredictability of incompatible hardware and wireless networks. This sensor system was thus designed with the assumption that the UAV would not have any other wireless receivers beyond a simple GPS receiver, and that the sensor system itself would not introduce any new wireless receiving capability. All sensing and processing would be localized.

Additionally, the generalizability of the sensor system requires that it contain its own processor and power supply. The reasoning behind this is that we cannot guarantee that the UAV onto which this sensor system is mounted will have a processor powerful enough or have sufficient memory for the computer vision tasks. By having a dedicated processor in the sensor system we can guarantee two things: that the hardware is suitably powerful for the computer vision tasks, and that we have a standardized software environment to work in.

Thus, the system was designed with an assumed division of labor: the onboard processor would handle computer vision, sensor querying, and high level decision making, while the UAV's own flight controller processor would handle any time-sensitive tasks, as well as the motor control and stabilization algorithms. This requires that there be some form of communication between the two processors, but this would be simpler to implement than would a sensor system that was agnostic of processor type.

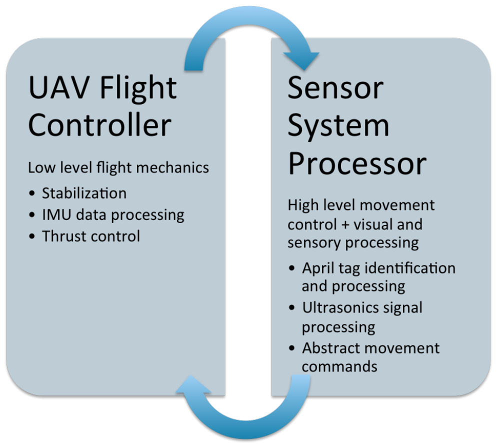

UAV Flight Controller
Low level flight mechanics
• Stabilization
• IMU data processing
• Thrust control
Sensor System Processor
High level movement control + visual and sensory processing
• April tag identification and processing
• Ultrasonics signal processing
• Abstract movement commands

Figure 2-2: The separation of processing tasks between the sensor system's processor and the UAV's processor.

2.4 Design Details

Our design's high level system diagram takes all of these considerations into account. For sensing, we have a camera and ultrasonic sensor. Additionally, it needs a processor and power supply. For communication with the UAV's processor, there should also be a port for a serial bus.

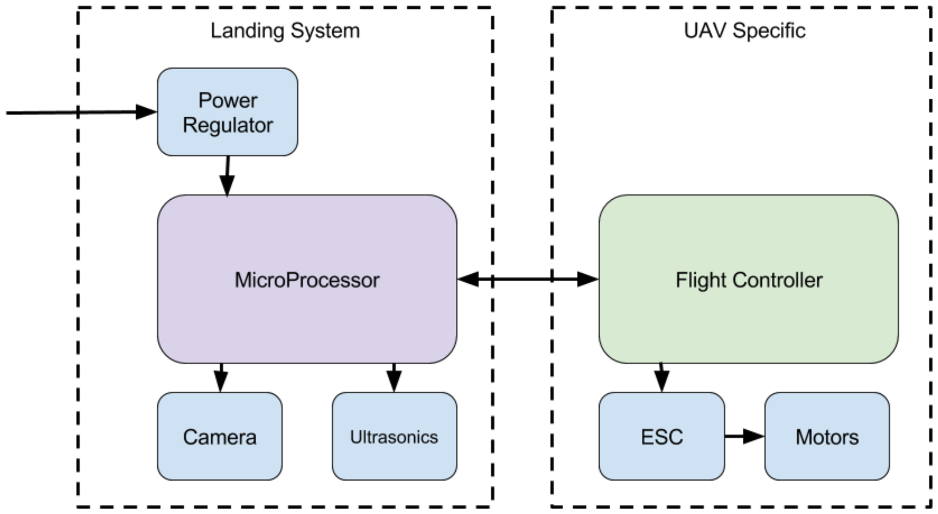


Figure 2-3: A high level overview of the system.

The individual components were chosen as follows:

2.4.1 Test UAV

The UAV for this project was a custom made quadcopter. Rather than buying a prebuilt one, we elected to build one ourselves in order to minimize costs and to have a customizable platform that was both programmable and easy to modify. This quadcopter was built on the DJI F450 frame (lightweight and cheap), and uses DJI 2312 motors with 420 Lite Electronic Speed Controllers (well-made and reputable) [1].

There were many options for the flight controller, but we chose to use the Pixhawk – specifically the HKPilot32 variant – because it is an open source hardware system that is compatible with the open source ArduPilot software system. The ArduPilot software system is an environment that allows modulation of the flight controller settings as well as flight path planning.

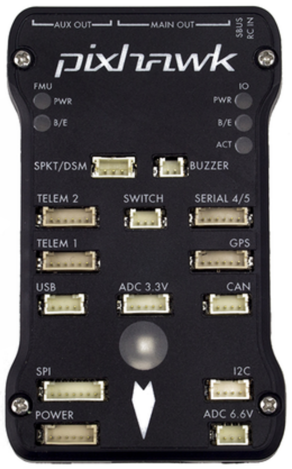


Figure 2-4: The Pixhawk flight controller.

For power, we used a 4-cell 14.8V LiPo battery. This would provide a quadcopter of this weight about 25 minutes of flight time. We also used some lightweight quadcopter legs in order to give the quadcopter enough verticle clearance to mount a device on the bottom side of the body. Lastly, we used a Futaba R6303SB radio receiver because it was cheap and compatible with the Futaba T14SG radio transmitter that we had access to.

[1]More details about propulstion system can be found here: http://www.dji.com/e305

This configuration of the UAV weighed 1155 grams, had a vertical clearance of 14.5cm, and had a total payload capacity of 2400 grams.

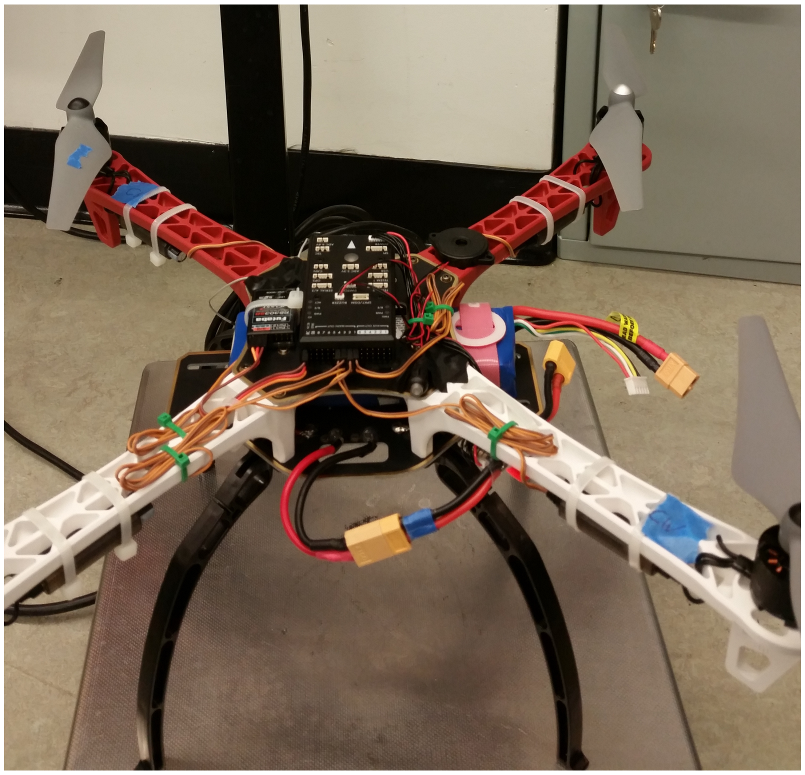


Figure 2-5: Version 1 of the UAV.

2.4.2 Optics

Choice of the downward facing camera was made by considering the following factors:

- weight
- hardware compatibility
- focal length
- aperture

Specifically, the characteristics of the optics were important in determining the limitations of the camera and whether these limitations were outside of our design goals. We paid special attention to the depth of field (DOF) and field of view (FOV) parameters because these parameters gave us insight as to whether we would be able to detect the landing platform at our design goal limits. The depth of field of the camera is the distance range where an object was in clear focus, and so this parameter would give us the altitude limitations of our system. The field of view is the lateral area that is within the camera frame at a certain distance away from the camera, and so this parameter would give us the lateral offset limitations of our system.

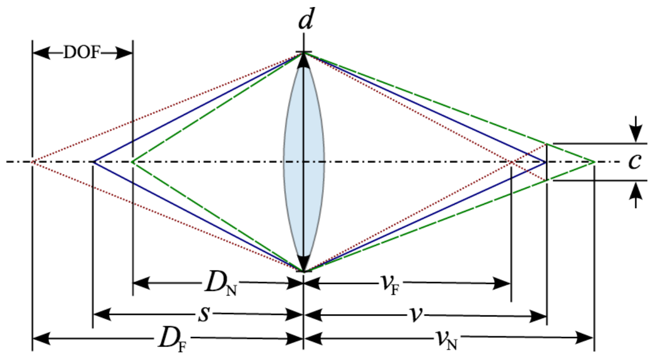


Figure 2-6: The depth of field defines the distance range where the camera image is in focus.

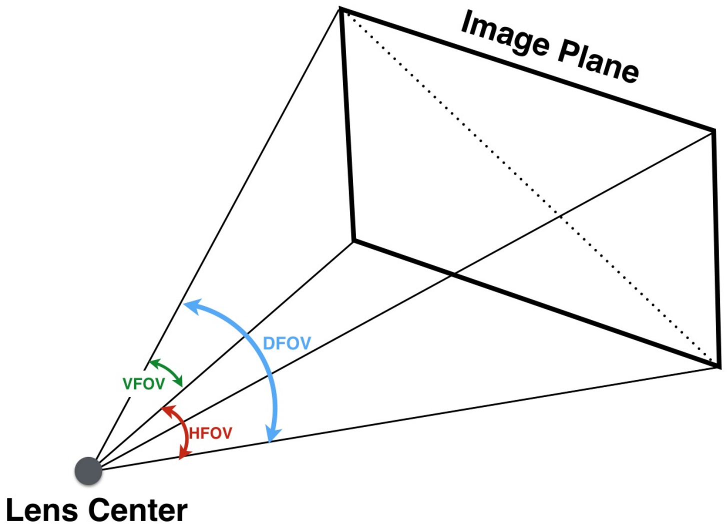


Figure 2-7: The field of view defines the area within the camera's frame at a specific distance away.

In analyzing these characteristics for the Logitech c920 webcam, we found that we were able to meet these goals, and so that is the camera that we elected to use. The Logitech c920 had the following characteristics [9]:

- Weight = 72.6g
- 30fps video
- Focal Length = 3.67mm
- F stop = 2.9mm
- Diagonal FOV angle: 78°
- Horizontal FOV angle: 70.42°
- Vertical FOV angle: 43.3°
- Circle of Confusion: 0.0042mm

Figure 2-8: The logitech c920 camera.

In order to determine the depth of field of our camera we used the following formulae [10]:

$$d_n = \frac{d_{pof} * f^2}{f^2 + kc_{coc}(d_{pof} - f)} \qquad (2.1)$$

$$d_f = \frac{d_{pof} * f^2}{f^2 - kc_{coc}(d_{pof} - f)} \tag{2.2}$$

where:

$$d_n := \text{near DOF distance}$$
$$d_f := \text{far DOF distance}$$
$$d_{pof} := \text{point of focus}$$
$$c_{coc} := \text{circle of confusion}$$
$$k := \text{aperture}$$
$$f := \text{focal length}$$

Plotting this relationship in MatLab with a point of focus distance of infinity, we obtain the following graph which depicts the DOF range: Where the blue curve

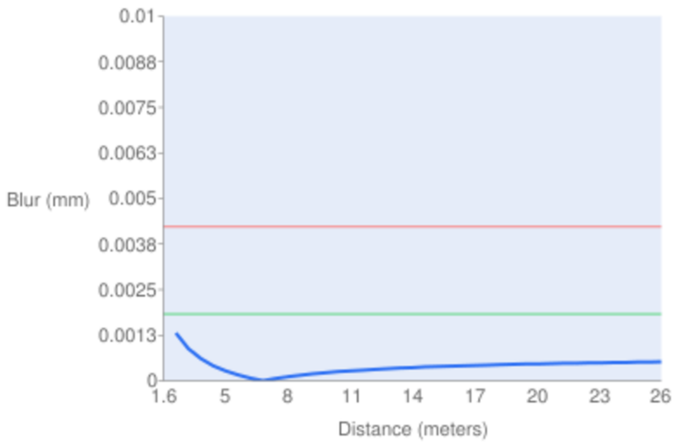


Figure 2-9: Focus vs. distance, defining the depth of field range using the Logitech camera's characteristics.

is the camera focus at different distances, and the green horizontal line depicts the point where image blur begins to becomes significant, and the red horizontal line is the point where the image becomes unrecognizable.

For our camera parameters, we obtain:

$$d_n = 0.91m$$

and

$$d_f = \text{infinity}.$$

Meaning that this camera can clearly see anything up to 0.91m from the camera itself. This suggests that we can theoretically see anything up to and beyond our 10 meter maximum recognition requirement, but anything within 0.91 meters of the camera is not guaranteed to be in focus.

In order to meet the 5m lateral offset design goal, we need to calculate the horizontal field of view (HFOV) and see whether a 5m HFOV is achievable within a reasonable vertical distance. To do this, we use the folowing formula to calculate 1/2 of the entire horizontal distance that should be viewable by the camera:

$$d_V = \frac{d_H}{2 * tan(\frac{HFOV^\circ}{2})} \tag{2.3}$$

where:

$$d_V \coloneqq \text{vertical distance (altitude of the UAV)}$$
$$d_H \coloneqq \text{half of the total horizontal view (10m total)}$$
$$HFOV^\circ \coloneqq \text{horizontal field of view angle}$$

Solving this formula with $d_H = 5m$ and $HFOV^\circ = 70.42$ results in $d_V = 3.5m$, which means that we should have a horizontal field of view of 5m at an altitude of 3.5m. Thus, at an altitude of 10m we will have a theoretical $d_H = 14.1m$ (a total horizontal field of view of 28.2 meters).

2.4.3 Rangefinder

In addition to the position and orientation information obtained by the camera, the UAV would benefit from having a localized sensor for altitude estimation. For altitude, GPS can give an estimate, but like position information this estimate is prone to offset error.

We investigated several distance sensors: Due to the need to remain cost-efficient

Table 2.2: Distance sensor comparison

Technology	**Pros**	**Cons**
IR Proximity	Cheap	Prone to noise/inefficiency outdoors
Radar	Long Range	High power requirement, expensive
Time of Flight	Long range	Expensive
Ultrasonic	Cheap	Medium range

and because the distance sensor would only really be needed when the UAV is within a few meters of the ground, we elected to go with the ultrasonic rangefinder. Our specific component of choice was the HC-SR04 rangefinder:

Figure 2-10: The HC-SR04.

Though we had no specific design criteria for the rangefinder, this choice fits into our system well because of its low power (<2mA quiescent current draw), small size, and high resolution: it is accurate to 0.3cm between the distances of 0cm and 400cm [11]. The rangefinder it most accurate within distances of 4 meters, and so it is most useful in our system at close range, when the UAV approaches the d_n distance of its depth of field range calculated in the previous section and the visual system begins to lose focus because of this close distance.

2.4.4 Processor

Visual computing tasks require more RAM, storage space, and processing power than a simple microcontroller can provide. Thus, a microprocessor is the better option for this sensor system.

Specifically, a microprocessor will need to fulfill the following requirements:

- Have a USB port (for connecting the c920)
- Have several GPIO ports (for interfacing with the HC-SR04)
- Able to run some distribution of Linux (so that we can run virtually any software environment)
- Have a small enough profile to not exceed our area design goal of 11.4cm x 14cm
- Have all of the major serial communication buses (for communication to the UAV flight controller)

In looking at different single-board computers, we elected to use the Raspberry Pi 2 Model B, because it was cheap, well documented, and fulfilled our requirements listed above. The Raspberry Pi 2 has a 32-bit quad-core ARM Cortex-A7 processor, 1GB of RAM, and unlimited storage space (storage is done via micro SD card), every major communication bus, and more GPIO pins than we require [12].

2.4.5 Fiducial Marker

In order to meet our recognition design goals and maximize the amount of information we can achieve from our visual system, we thought it be best to use a fiducial marker to identify the landing platform. Like in the Lange et al. paper, with a fiducial marker we can gather position and orientation estimates, and we can reduce recognition false positives since they are easier to identify outdoors due to their non-natural appearance.

For this, we investigated various different implementations of augmented reality (AR) tags. AR tags were originally created for augmented reality and virtual reality applications. They are optimized for low-latency recognition in 3D space, and by design include information about the observer's relative position and pose [13]. Thus, they are a very good option for this project. The specific implementation that I chose to work with was a ROS package called AR Track Alvar, which is a ROS wrapper of the Alvar variant of the AR toolkit implementation. Alvar is more advanced than the original ARtoolkit. Alvar features adaptive thresholding to handle a variety of lighting conditions, optical flow based tracking for more stable pose estimation, and an improved tag identification method that does not significantly slow down as the number of tags increases. Multiple AR tags can be identified simultaneously if all of the markers are unique. I did not need a large amount of tags, so I elected to use AR tags that were 5bit x 5bit in size; this tag resolution size had 209 unique tags, which was more than I needed.

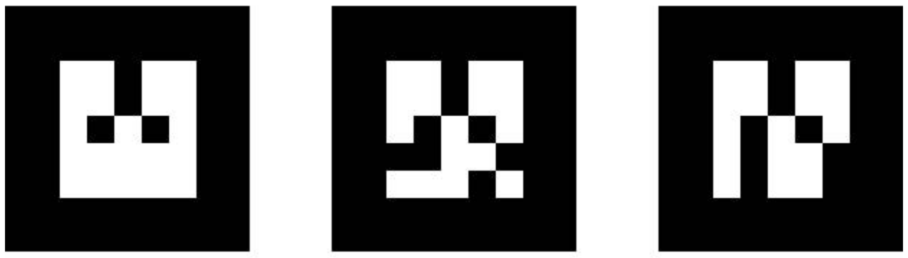


Figure 2-11: AR tags 0, 1, and 2. They are 5bit x 5bit.

For my implementation, I needed to know the UAV's relative x, y, and z position in meters, as well as the Euler angles (yaw, pitch and roll). The raw information obtained from the Alvar node included the identified tag id number, x, y, z position, and the tag's quaternions.

If we define the UAV and AR tag Euler angle reference frames of like so:

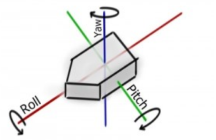


Figure 2-12: The UAV's reference frame.

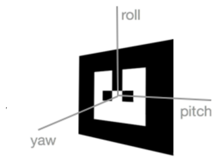


Figure 2-13: The AR tag's reference frame.

we can convert the raw quaternion information into the easier to work with roll (ϕ), pitch (θ), yaw (ψ) in radians using the following formula [14]:

given:

$$\boldsymbol{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T \tag{2.4}$$

then:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0 q_1 + q_2 q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan \frac{2(q_0 q_3 + q_1 q_2)}{1 - 2(q_2^2 + q_3^2)} \end{bmatrix}$$

Given this calculation and the raw data returned by the Alvar ROS node, a vector defining the state of the UAV with respect to an AR tag i can be written like so:

$$\boldsymbol{t_i} = \begin{bmatrix} x_i & y_i & z_i & \phi_i & \theta_i & \psi_i \end{bmatrix}^T \tag{2.5}$$

where:

$x :=$ lateral offset along the same axis as the pitch (in meters)

$y :=$ lateral offset along the same axis as the roll (in meters)

$z :=$ altitude distance along the same axis as the yaw (in meters)

2.4.6 Software Stack

In order to use the AR tag libraries mentioned in the previous section we needed to use the Robot Operating System (ROS). ROS is a software framework commonly used in robotics. It allows for the modular representation of sensors and actuators into abstractions called "nodes". A node has topics associated with it, and a programmer can either subscribe or publish to a topic in order to either gather data or make the robot perform an action. For example, our camera would be able to publish video frames to a \webcam node, which the \ar-track-alvar node would subscribe to and process, and then publish to the \ar-pose topic, which my script could then subscribe to and query for the raw position and pose data. Thus, because it is intended for this purpose, we elected to use the ROS framework in our sensor system's software stack.

A Raspberry Pi can easily install a plethora of different operating systems. For our device, we wanted an operating system that was lightweight but versatile, and was able to run ROS. This narrowed down our choices considerably, as most operating systems that supported ROS were fairly large. One operating system that did fulfill our requirements was Ubuntu 14.04 Trusty for ARM processors. At a size of $<$ 1GB it was a prime candidate.

On top of our Ubuntu OS, we had to also install Python and OpenCV (an open computer vision package for C++ and Python), as we were most familiar with Python, and OpenCV was a requirement for the AR Alvar ROS package.

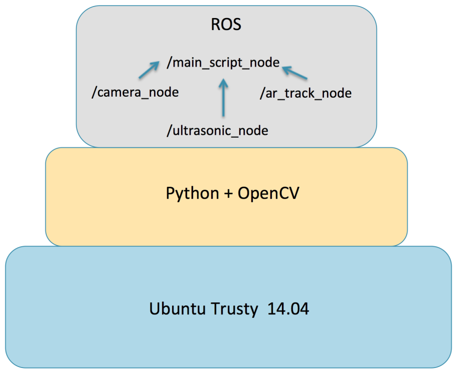


Figure 2-14: Abstract representation of the software stack.

The \ar-pose topic relied on a set of xml files (which can be found in Appendix

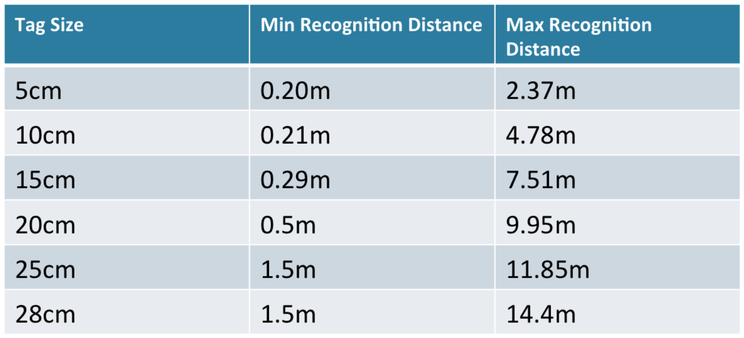
D) which defined the size associated with each tag ID. In this way, the proper transformation could be made between pixel and meters. The main sensory system script created a ROS node called \ar-tag-node, which subscribed to the \ar-pose node and the \ultrasonic-node. Thus, all of the raw sensor information data was available in a single place, and could all be accessed and processed when convenient. The final iteration of the script ran at a speed of 10Hz; this was because of the \ar-pose node bottleneck. While the camera captured video at 30fps, the visual processing of the AR markers took longer than 1/30th of a second, and so the state data was published to the \ar-pose node at 10Hz. Fortunately, by design the processor on this system was not responsible for any time-critical tasks (such as balancing and stabilization of the UAV itself), and while 10Hz seems rather slow it is enough time for a process to decide on high-level trajectory commands based on the visual and ultrasonic data.

2.4.7 Landing Platform / Fiducial Cluster

In creating the landing system, one design goal was to have a high recognition altitude. For this we could have just created a very large AR marker and placed it on the 0.6m x 0.6m platform since larger markers can be recognized further than smaller markers. One problem that arises from this is that at lower altitudes it is possible that the entirety of the marker would not be in the camera frame, and so there would be a section in the descent path where we would not have any visual information to rely on. Additionally, another requirement was to have optimal and continuous recognition of the platform during the entirety of the descent path. Therefore, it was a better idea to find a set of markers of different sizes – sequentially decreasing in size – such that the sensor system would recognize and rely on different markers as it descends on the platform. In this way, the continuity problem is solved.

To accomplish this, we first needed to identify the relationship between marker size and recognition distance, as this information is not available in the Alvar ROS package, nor is it a relationship that can be characterized by a simple equation. This we created tags of different sizes (measured at one side) and used our previously mentioned ROS topics to find the distance limitations of each tag size. For this test, we define the "recognition rate" as the number of times a tag is recognized over 10 samples. We used a cutoff of 60%, so if a tag were recognized less than 60% of the time at a certain distance that distance would be the cutoff point. Our findings should the following limitations for each tag size:

Table 2.3: Tags of different sizes and their recognition distance limitations



Tag Size	Min Recognition Distance	Max Recognition Distance
5cm	0.20m	2.37m
10cm	0.21m	4.78m
15cm	0.29m	7.51m
20cm	0.5m	9.95m
25cm	1.5m	11.85m
28cm	1.5m	14.4m

From this we chose the tag sizes 10cm, 15cm, 20cm, 25cm, 28cm since their recognition ranges overlapped best, and they gave use the maximum and minimum recognition distances that comply with our original design goals. From these choices we created a fiducial cluster where the markers were all as close to the center of the platform as possible and are all have a yaw orientation in the same direction (in the direction of the arrow) so that we can designate a side to be the "Northern" direction:

In this configuration, the tags were given IDs 0, 1, 2, 3, and 4 in order of decreasing size. The sensor system recognition algorithm that we wrote took into account this variation in size based on the tag ID, so that position estimates were calibrated properly based on which ID was recognized. It also took into consideration the fact that any of the lateral position estimates would be offset from the true center by a few centimeters.

Another important aspect of our implementation of the final cluster recognition algorithm was a weighted average of the vectors of the seen vectors. At any point in the descent (except for at the extrema) it is possible for more than one AR marker to be recognized simultaneously. Thus, we could create a weighted average estimate vector of all of the recognized tag estimates, with higher weight given to the largest

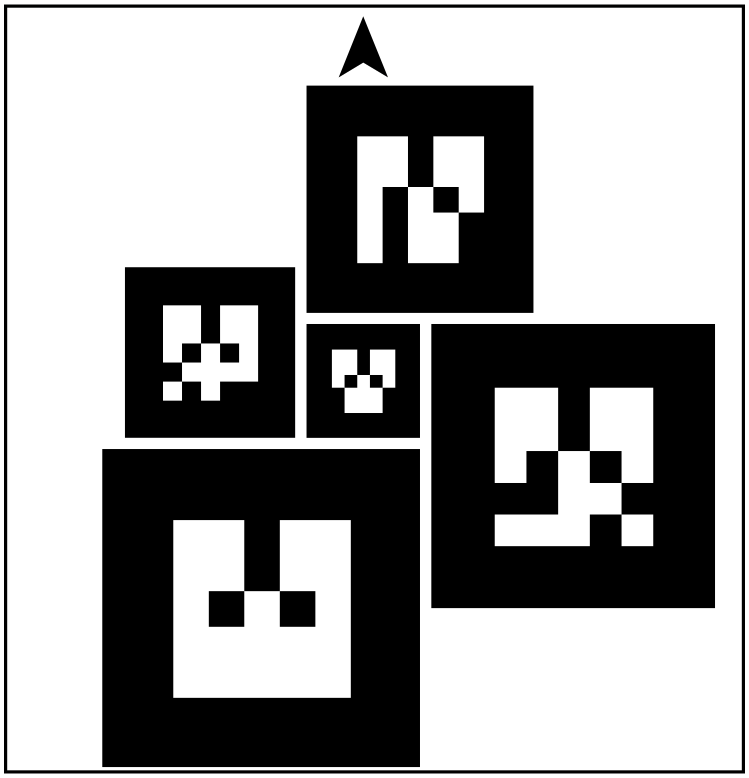


Figure 2-15: The final platform AR marker design.

tag that is recognized in that given instance. The idea behind this is that the weighted mean of multiple state estimates would provide the maximum likelihood state estimate [15]. If we take the vector (2.5) to be a state estimate for an individual tag i, then we define the weighted mean over n number of tags in the entire cluster at any given time to be:

$$\boldsymbol{t_c} = \sum_{i}^{n} \boldsymbol{W_i t_i} \tag{2.6}$$

where $\boldsymbol{W_i}$ is the diagonal weight matrix for vector t_i. The weights are normalized over i, such that:

$$\sum_{i}^{n} \boldsymbol{W_i} = \boldsymbol{I} \tag{2.7}$$

that is, the element-wise addition of all weight matrices $\boldsymbol{W_i}$ would result in a square matrix of dimensions n x n and with a diagonal of 1's like so:

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

This property is held for any combination of observed AR markers – that is, $\boldsymbol{W_i}$ is not static, and depends on how many other AR markers are seen in that instance of time; when there is only one marker spotted, $\boldsymbol{W_i}$ is the identity matrix of size 6 x 6 (the size of a single vector t_i) . Because the state estimate from the larger tags is more likely to be correct, the larger tags are weighted more heavily in the combined estimate. The weights for lower numbered i tags are bigger than the weights for higher numbered i tags (since the lower i's correspond to larger tags).

2.4.8 Design Evolution

One improvement made to the UAV was to exchange the off-the-shelf landing gear for a set of 3D-printed landing gear that improved mounting area at the bottom of the UAV body, prevented obstruction of the camera's vision, and were lighter than the previous set. This replacement had the effect of reducing vertical clearance, but this was not a significant consequence as there was still sufficient clearance for our device.

The first iteration of our device was made to primarily test vision. It used the Raspiberry Pi Camera v1, which is a 5MP CSI camera specifically made for interfacing with Raspberry Pi devices. While lightweight and extremely small, this camera did not meet our distance recognition requirements, and so it was later replaced by the c920.

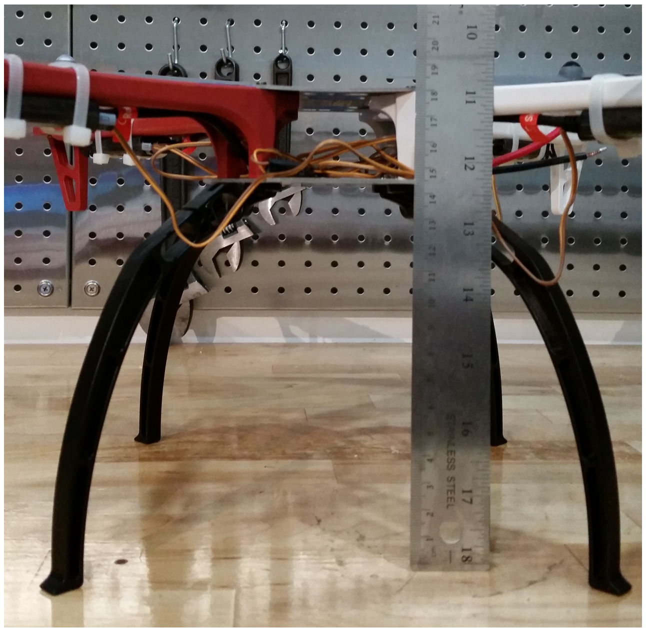


Figure 2-16: The UAV with taller landing gear.

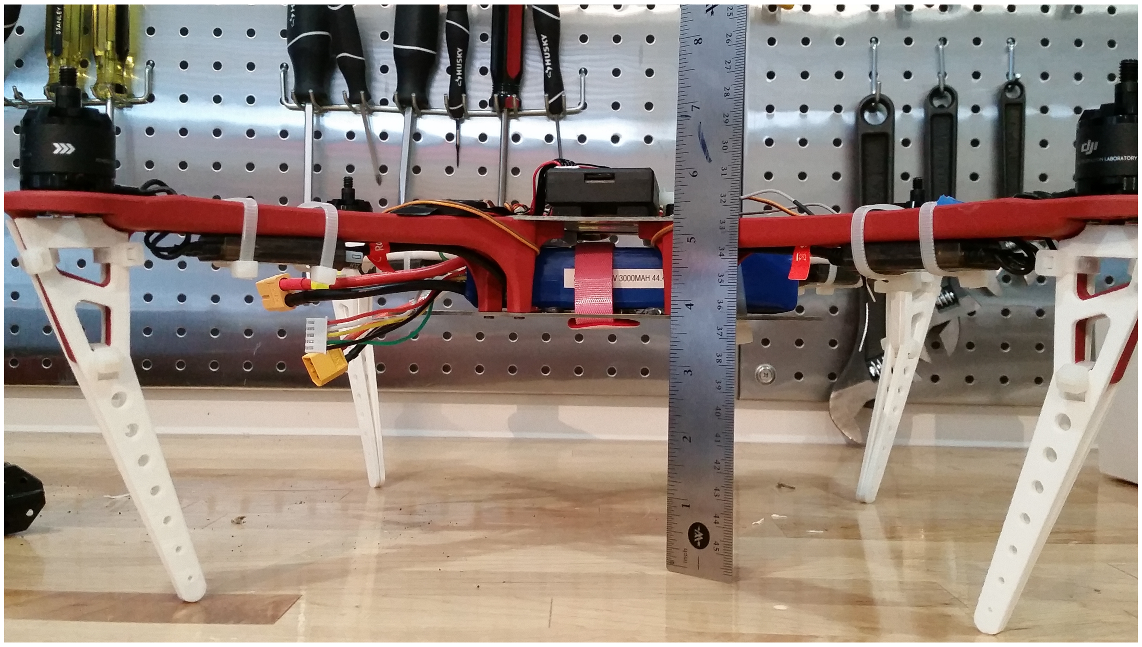


Figure 2-17: The UAV with shorter and more spread out landing gear.

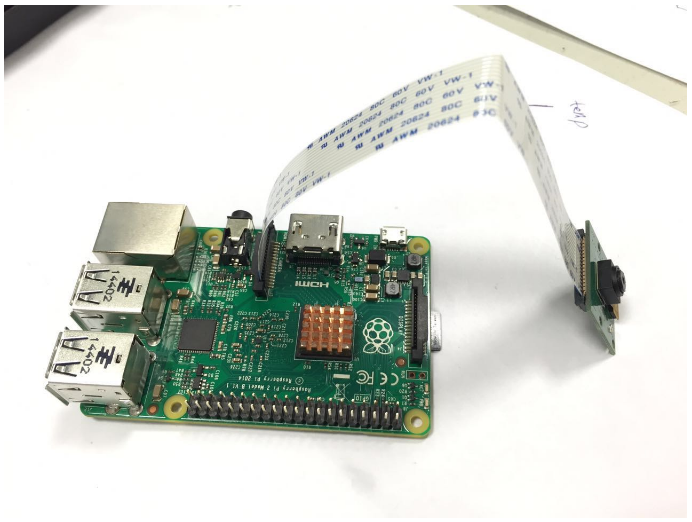


Figure 2-18: The first iteration of the device. It used a RPi camera v1.

Our device was prototyped on a breadboard and testing fixture, powered by a portable power bank:

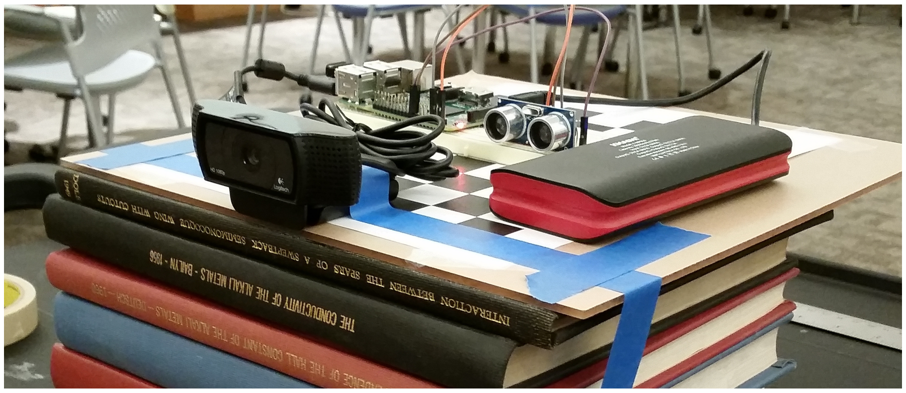


Figure 2-19: The second iteration of the device.

After designing an enclosure that met our design goal specifications, the final prototype was made:

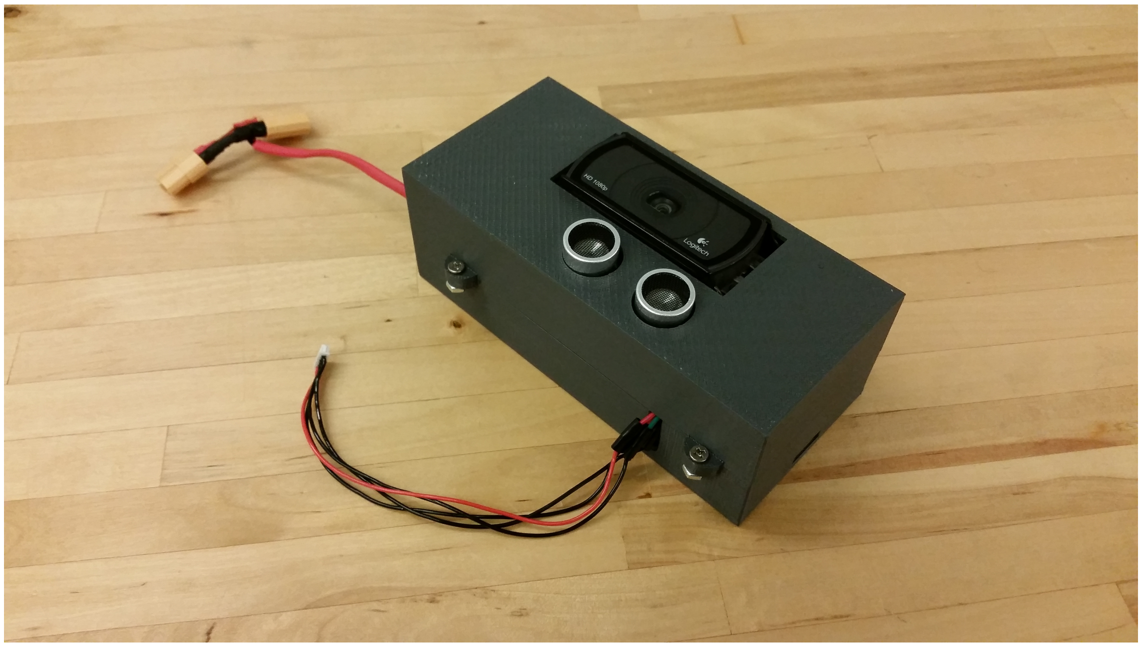


Figure 2-20: The final iteration of the device.

An exploded view of this enclosure better illustrates the integration of the components:

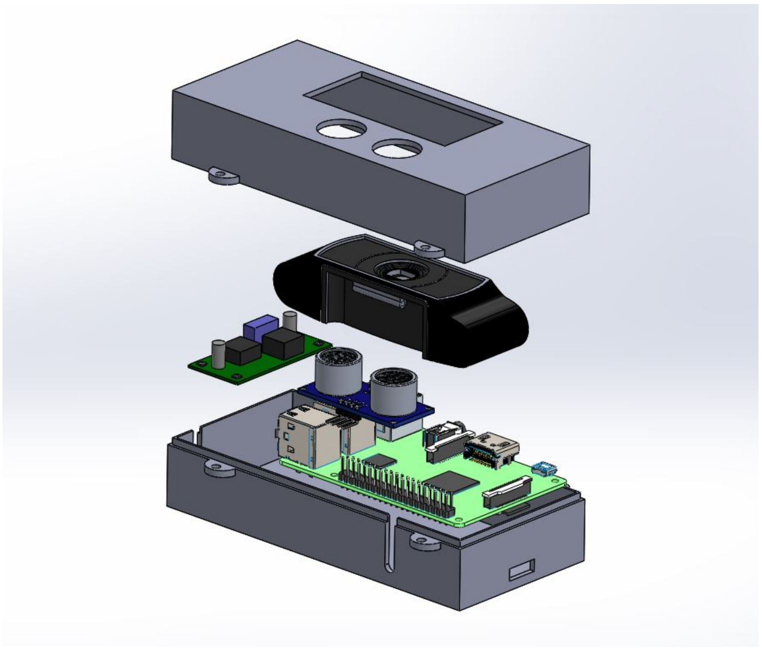


Figure 2-21: An exploded CAD view.

Vibration reduction foam and velcro was used for mounting to the bottom of the UAV. This was essential to obtaining clear visual, and in improving recognition.

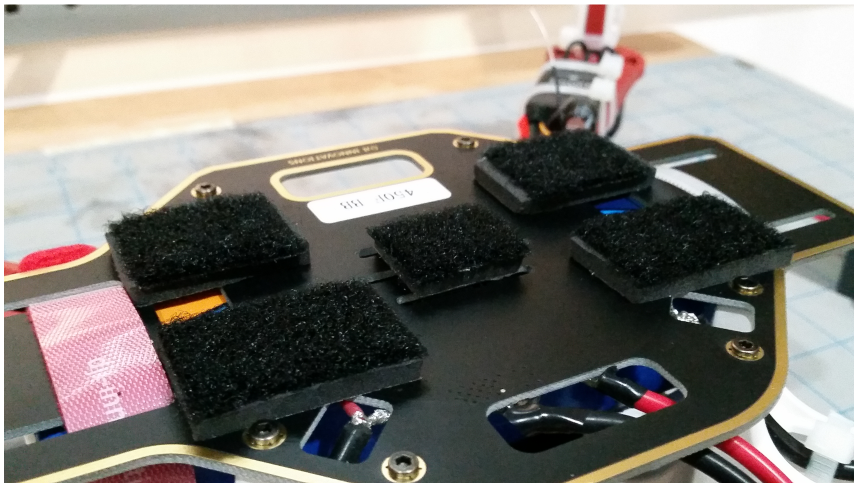


Figure 2-22: Vibration reduction foam on the bottom side of the UAV body.

The final integration of the device onto the UAV. The device siphoned power from the UAV's LiPo battery, and also had a serial port for communication with the Pixhawk.

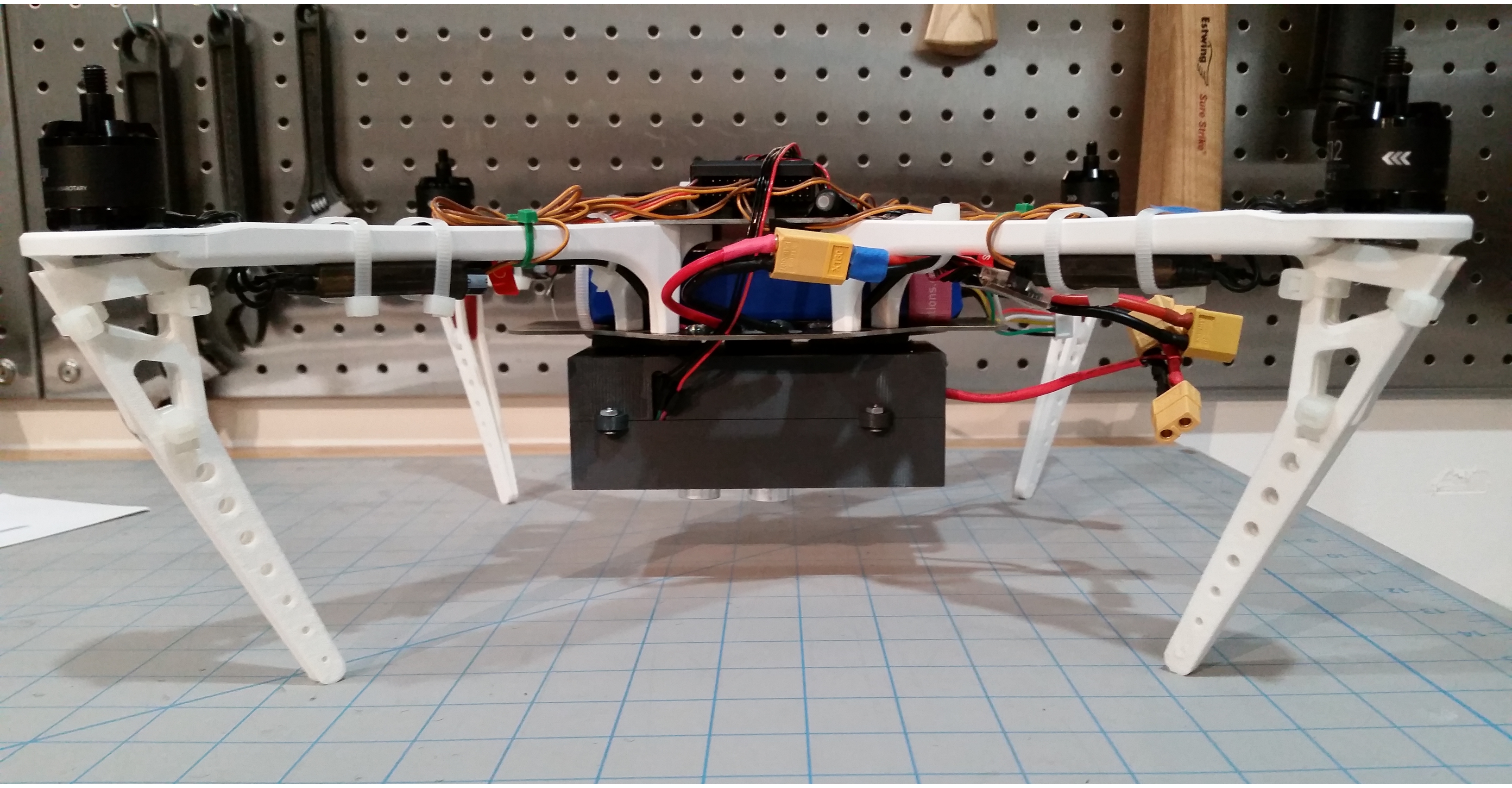


Figure 2-23: Integration of the device and UAV.

Evaluation/Verification

In order to control for different lighting conditions and different distances throughout these tests, the camera settings were fixed to:

- Auto white balance off
- Autofocus off
- Focus at infinity

First, we performed confirmation tests to see how well the raw data from individual tags performed for AR markers of several sizes. There are six main variables in a state, and so to test one we had to fix the other five. For example, while testing for distance estimation accuracy in the z direction we had to fix the x and y positions, as well as the yaw, pitch, and roll. The three main variables that we tested were accuracy in the z direction (altitude), accuracy in the x direction (lateral offset), yaw accuracy (rotation around the axis normal to the tag). We found that while keeping all other variables constant, at fixed increments the state estimation by using a single tag was ,on average, off by 3cm or less. For distances of more than 2 meters, this error was so insignificant that it is likely attributable to user error and/or error in test setup. From this we concluded that for fixed setups, state estimation was almost completely accurate, and did not vary with tag size (the error was constant for the ranges in which a tag was recognizable).

For the fixed variable tests that follow, our test setup consisted of a tag (or multiple tags) that were held static, distance markers on the ground which were used for reference, and a cart with our sensor system prototype held rigid by tape.

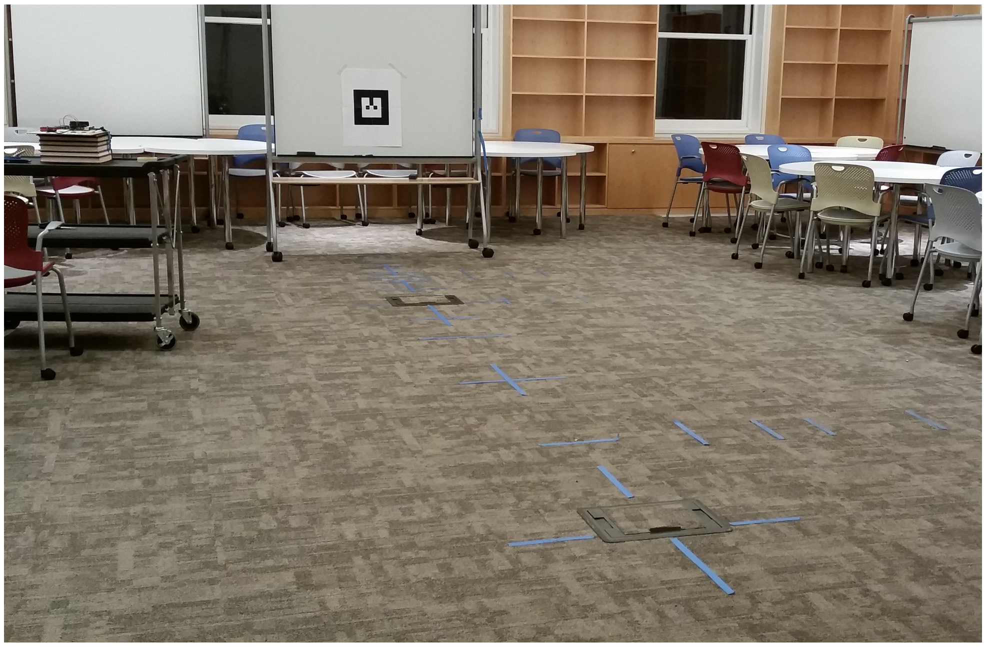


Figure 3-1: The test setup for fixed variable testing.

3.1 Accuracy of Recognition for Altitude

In order to test for the accuracy of recognition with respect to altitude, we fixed the sensor system at several increments between 0 and 11 meters along the z-axis while keeping all other variables constant. The sensor system was placed at an y distance that was approximately aligned with the platform's y axis; the roll and pitch angles were fixed at 0, and the yaw was fixed at +/- π.

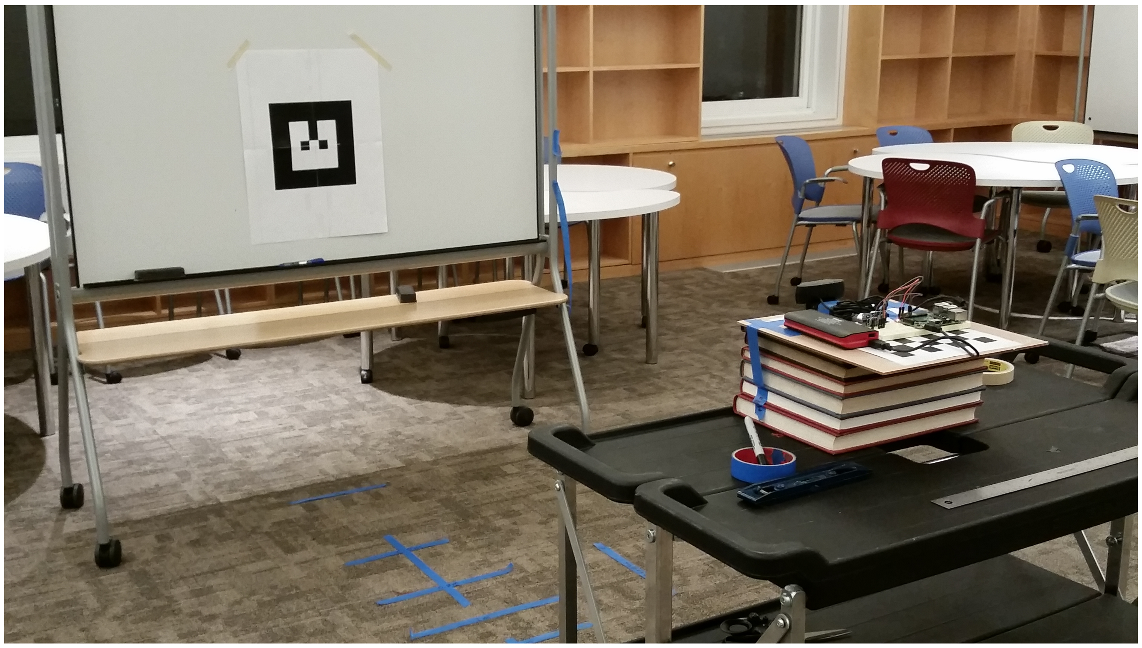


Figure 3-2: The prototype being used for a test.

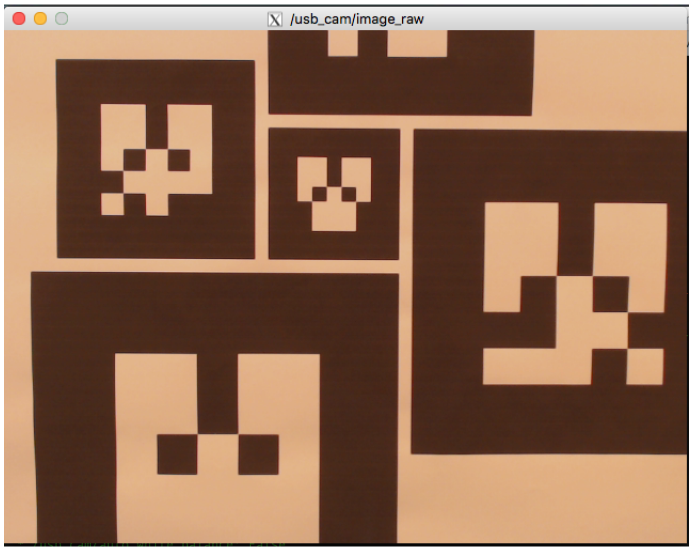


Figure 3-3: A raw image from the camera's perspective.

We then modified our script to output a log file that included the state information from the cluster and from every individual tag that was identified at that particular distance. We then compared the average of multiple samples at each distance with the actual distance using the following formula:

$$EE = \frac{M - A}{A} * 100 \tag{3.1}$$

where:

$$EE := \text{estimation error}$$
$$M := \text{measured distance}$$
$$A := \text{actual distance}$$

The following plot shows this comparison. The red bar is the cluster error (estimation error from the weighted mean vector). Tags that were not spotted at a particular z distance were omitted.

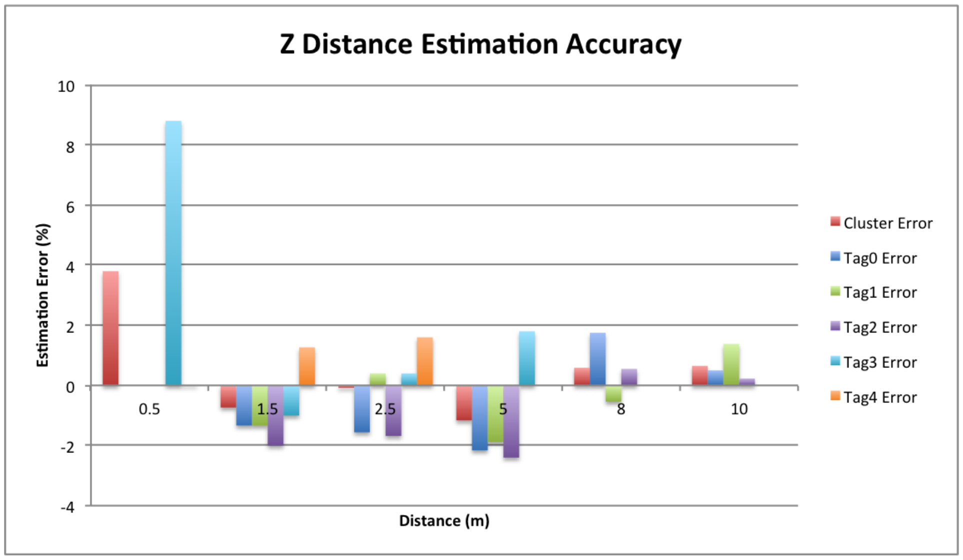


Figure 3-4: Z distance estimation comparision between the cluster and individual AR tags.

We can see that for just about every z distance increment, the estimation error of the cluster was better (closer to 0) or equal to any individual tag estimate error, which signifies that the mean is actually slightly more accurate than a single estimate.

3.2 Accuracy of Recognition for Lateral Offset

Similarly we conducted estimation accuracy tests while modulating lateral offset distance at fixed lateral offsets and different fixed z distances in order to compare the estimated lateral offset with the actual lateral offset. We varied the z distances in order to see whether the accuracy trend continued for further z distance. The z distances used were: 5m, 8m, and 10m. At these fixed z distances we shifted the x-axis offset by: 0.5m, 1m, 2m, 3m, and 5m (where possible).

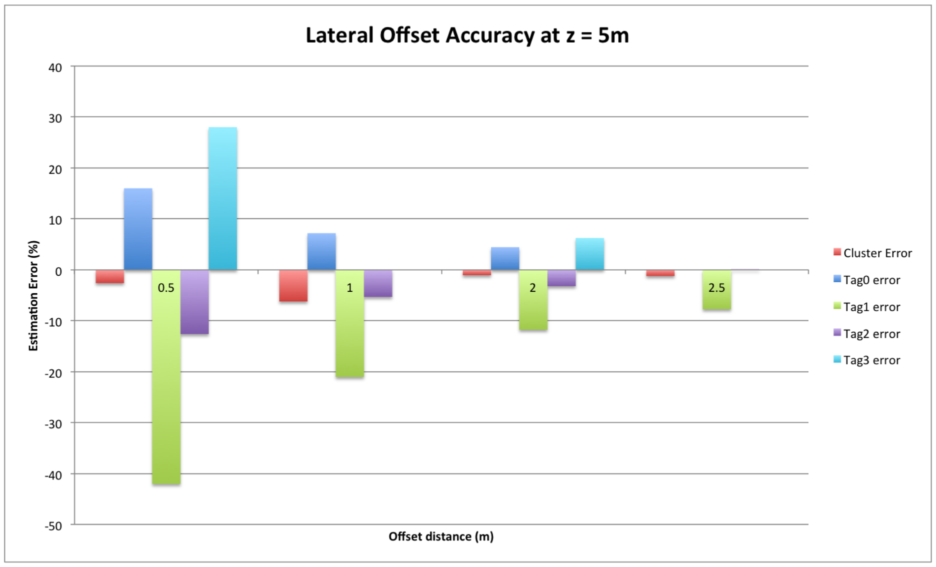


Figure 3-5: Z distance estimation comparision between the cluster and individual AR tags at various lateral offsets.

Similarly, in all of these tests, the the estimation error of the cluster was better (closer to 0) or equal to any individual tag estimate error.

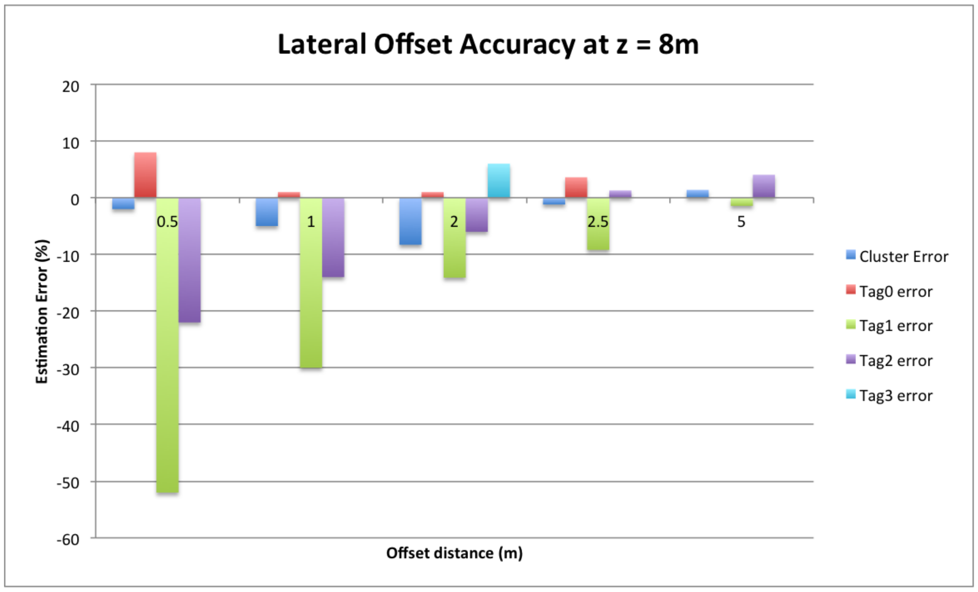


Figure 3-6: Z distance estimation comparision between the cluster and individual AR tags at various lateral offsets.

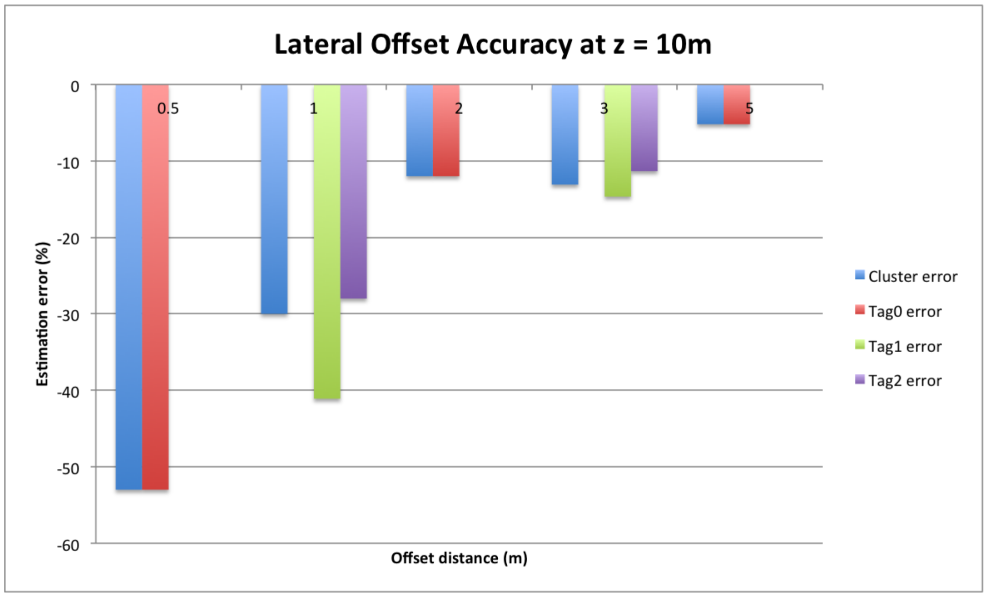


Figure 3-7: Z distance estimation comparision between the cluster and individual AR tags at various lateral offsets.

3.3 Recognition at Different Illuminance Levels

Another test that we conducted was recognition rate vs. illuminance. Illuminance is a measure of how much luminous flux is spread over a given area. One can think of luminous flux (measured in lumens) as a measure of the total "amount" of visible light present, and the illuminance as a measure of the intensity of illumination on a surface. Thus, testing recognition vs. illuminance would give us insight into how well the system would perform in various different lighting conditions and times of day.

For this test we used the following commonly accepted lux values [16]:

Table 3.1: Illuminance levels for our simulated conditions

	Illuminance Condition	**Illuminance (lux)**
🌙	Night	20
☁	Dark Overcast Day	100
🌅	Sunrise/Sunset	400
⛅	Overcast Day	1000
☀	Really Bright Day	32000

In order to simulate these brightness levels, we used a dual head 1000-Watt halogen tripod work light and placed it at different distances from the test platform in order to modulate the lux (the work light brightness could not be modulated otherwise). We measured lux using our smartphone's ambient light sensor and a Google app called "Science Journal" which allows access to the raw data from the smartphone's sensors.

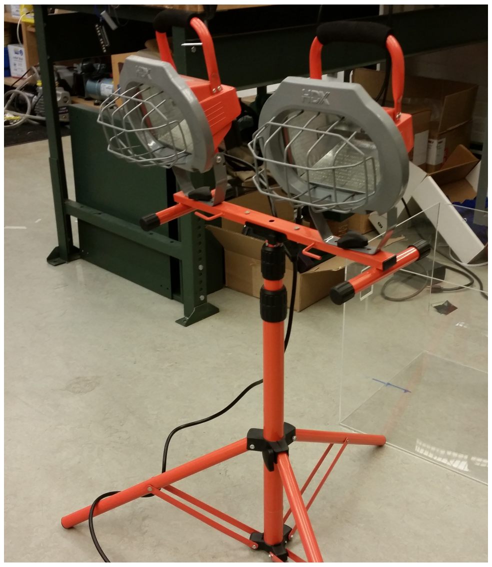
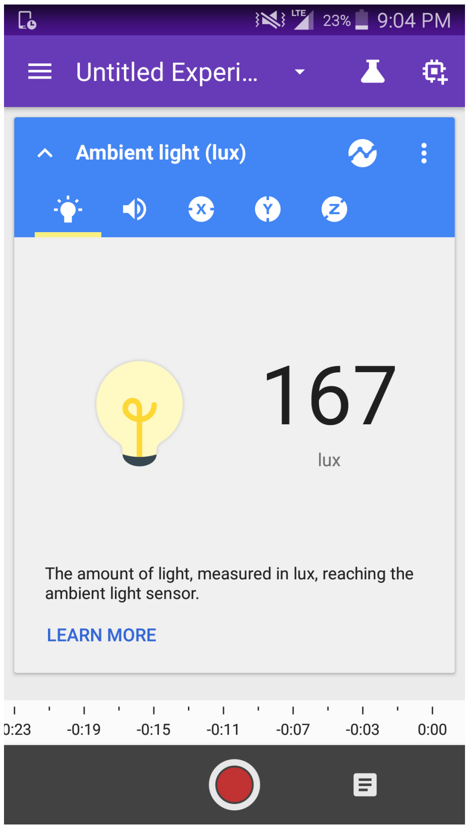


Figure 3-8: The halogen work light used for our brightness level simulations.



The amount of light, measured in lux, reaching the ambient light sensor.

Figure 3-9: The Google Science Journal app.

We tested all five of these lux values at z distances of 2m, 5m, 8m, and 10m in order to compare how illuminance affects recognition rate as the UAV increases in altitude.

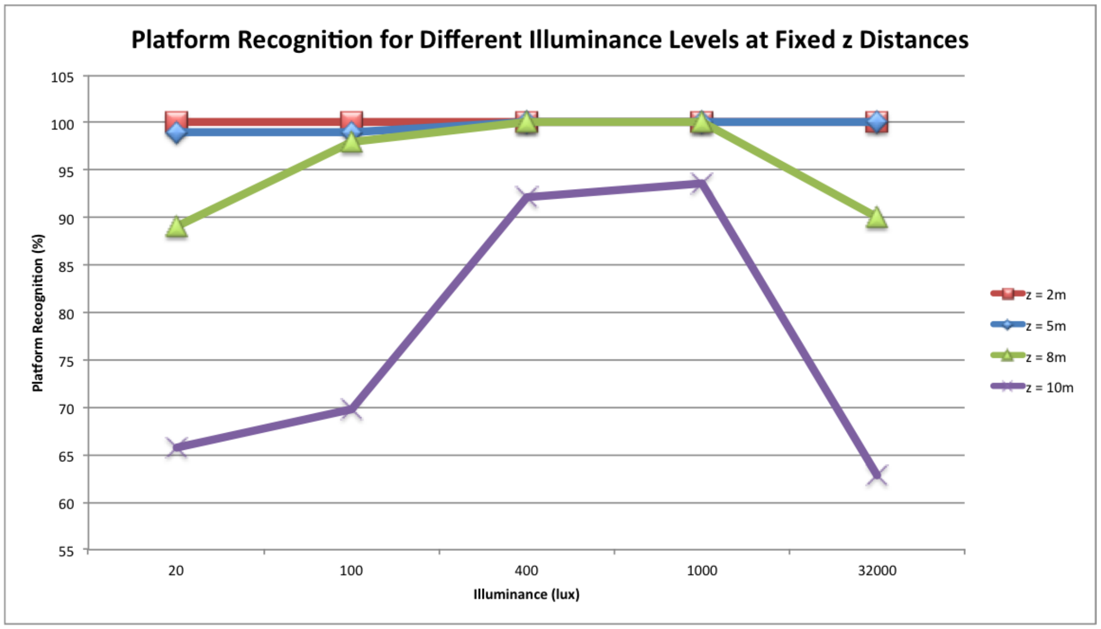


Figure 3-10: Recognition rate vs. illuminance at different z distances.

Recognition rate was defined as the number of times out of 10 samples that the AR cluster was detected (and then converted into a percentage). From this plot, we can see the further z distances dip in recognition at the lowest lux values, and at the very highest one. This is to be expected, since darkness and light saturation both impede recognition. For all distances, the optimal brightness levels were between 400 lux and 1000 lux. It was observed that lower recognition rates affected recognition accuracy; >60% accuracy was acceptable but not ideal.

3.4 Recognition While Moving

In order to see whether recognition would be reliable while the UAV was on its descent path, we tested recognition rate vs. distance while the sensor system is not static. To do this, we mounted the sensor system on our test UAV and manually piloted it over the AR cluster platform along a descent trajectory that it would be expected to take under ideal circumstances. It was not possible to fix variables such as lateral positioning or descent speed, so those were controlled as much as was possible.

For this test, the UAV started at ground level and was flown upwards while centered at the lateral origin of the platform. It moved upwards at approximately 0.2m/s (the speed that it would be expected to descend at if autonomous) and logged the recognition rate along with a vertical distance (with recognition rate defined as in the previous test) until the algorithm could no longer identify the platform.

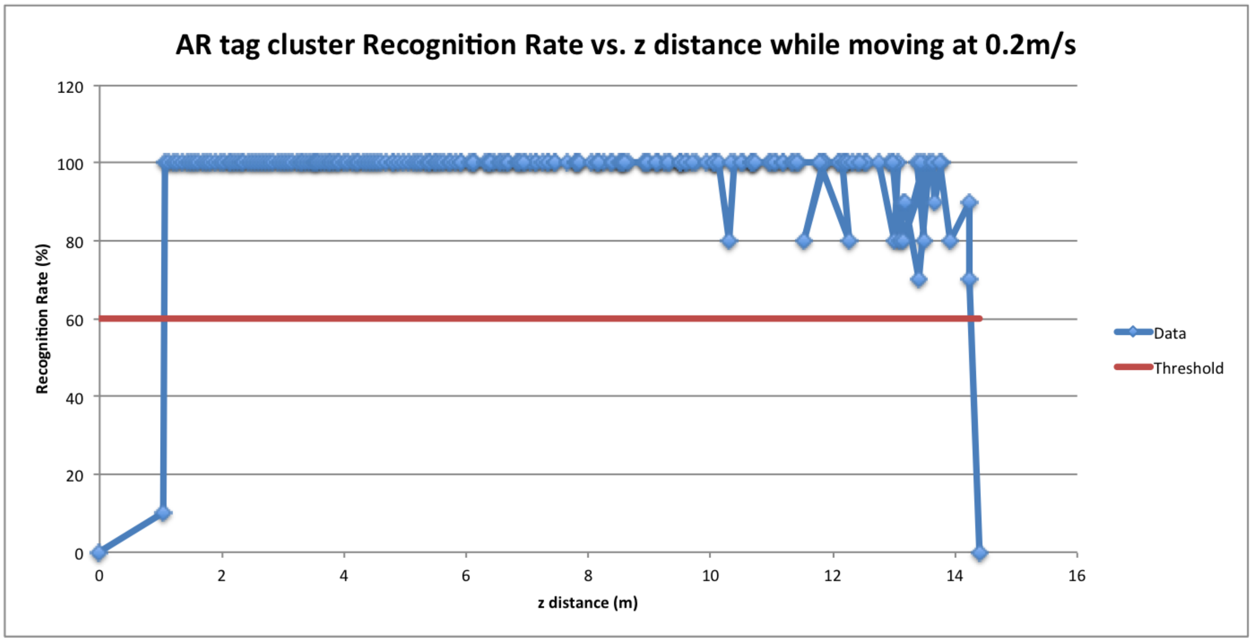


Figure 3-11: Recognition rate vs. vertical distance while the UAV is moving.

A few conclusions from this plot:

- Recognition is unreliably until a 1 meter vertical height is achieved. This connects back to our findings about the camera blur region and depth of field (which detailed a near distance of 0.91m). We see that in practice the depth of field does have an inhibitory effect on recognition at close distances.

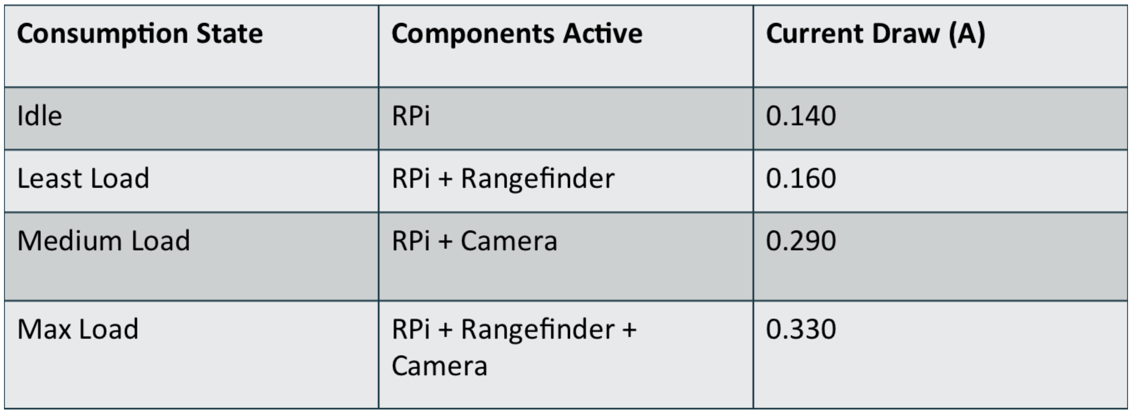
- We achieve a recognition rate of 100% between 2 meters and 10 meters of altitude, meaning that the cluster of AR tags are working more effectively as a combination than any one tag would work alone. The optimal recognition ranges (see table 2.3 for these recognition ranges) for the chosen AR tag sizes overlap enough to achieve reliable recognition along the descent path.

- Recognition becomes noisy and unpredictable at altitudes greater than 11.5 meters. Referring to table 2.3 we see that there is only one AR tag which is still recognizable at this distance (the 28cm tag) and so recognition of the platform at this altitude is actually just recognition of the 28cm tag. While the recognition rate jumps around, it still maintains a rate of $>$60% (which is the point when estimation error becomes greater than the 3cm error described at the introduction of the Evaluation/Verification section).

3.5 Quantitative Design Goal Tests

Current draw was measured with a supplied voltage of 14.8V (the voltage of the UAV LiPo battery), which was stepped down to 5V by the sensor system's buck converter. The buck converter had an efficiency rate of 80%, so some power was lost for all of the following measurements. We tested four different states that the sensor system would undergo during normal operation. These were: Idle (only the Raspberry Pi is active), Low Load (only the rangefinder is active), Medium Load (only the vision system is active), and Max Load (all components are active).

Table 3.2: Current Draw Measurements



Consumption State	Components Active	Current Draw (A)
Idle	RPi	0.140
Least Load	RPi + Rangefinder	0.160
Medium Load	RPi + Camera	0.290
Max Load	RPi + Rangefinder + Camera	0.330

While 0.33A is not negligible, that accounts for a <1 min flight time reduction (assuming the device is operating at max load for 5 minutes). Additionally, we were able to meet our current draw design goal and not surpass that limitation.

We also weighed the components of the system individually before combining them into the final prototype. We see that the combined device is slightly heavier than the individual components, but that is a combination of the 4-40 screws, wiring, and the XT-60 power connector.

Table 3.3: Measured Weights of Components

Component	Weight
Raspberry Pi	42.05 grams
Power Regulator	13.06 grams
Camera	72.65 grams
Rangefinder	8.43 grams
Enclosure total	65.6 grams
Device Total:	225 grams

Even at 225 grams, our device does not exceed our design limitation of 600 grams. The total flight time cost due to this additional weight is 3.17 minutes.

In tallying our measurements detailed here and in previous sections, we can compare our design targets with our actual characteristics:

Table 3.4: Design Target Comparison

	Target	**Actual**
Landing Area	0.6m x 0.6m	0.65m x 0.65m
Weight	< 600g	225g
Current Draw	< 0.5 A	0.33 A
Physical Area	11.4cm x 14cm	5.84cm x 13.2cm
Height	9.5cm	4.5cm
Max Height of Recognition	10m	14.4m
Max Lateral Offset of Recognition	5m	5m

We see that we managed to meet 6 of the 7 design targets, with the "Landing Area" target being exceeded only slightly.

Budget

A detailed bill of materials with component prices can be found in Appendix A. In total, this project cost \$548.11, which is slightly higher than the allotted budget of \$500. The device itself cost \$100 while the rest of the budget was used for the components of the UAV.

This project was formed in the context of a one-off proof of concept prototype. For this reason, the project budget was considered only as a design constraint and not a design factor; that is, for the most part the project prototype was not built to be the most economical. However, we did try to cut costs wherever possible because the \$500 budget for the project was relatively small; UAV parts are notoriously expensive, and so cutting costs would allow more components to be purchased.

A design that would leverage the economies of scale would require using fewer off-the-shelf components. In fact, if this product were intended for a consumer or business market, it would be completely custom made. Everything from the microprocessor board to the camera to the enclosure would require more time to design and test. However, the design of a completely custom made sensor system of this sort for mass production would require a much larger up front investment of both R&D time and money than the amount that is allowed through ES100.

Conclusion

In this paper we have demonstrated the design and evaluation of a proof of concept prototype of a device that would aid an autonomous UAV in landing at a specific location, with the intention that such a device could be used in extending an autonomous UAV's performance by allowing it to locate and land on a charging platform.

For this we used a custom-designed fiducial marker system built using the Alvar markers system, which gave state information about the UAV's relative x,y,and z positions, as well as the relative Euler angles. Using the Robot Operating system we were able to devise a state estimation algorithm that uses visual data from an on-board camera and the processed information from the ar-track-alvar ROS library in order to identify and estimate the UAV's relative state by using the information from one or more identified markers on the platform. An ultrasonic rangefinder provides supplementary altitude measurements for altitudes where the visual system becomes unreliable due to the limitations of its depth of field. It is relatively low power device that uses a Raspberry Pi 2 for processing and inter-processor communication. All of this is packaged in a lightweight and mountable enclosure.

In terms of performance, we were able to meet all but one of our quantitative design goals. The sensor system prototype is capable of identifying the target landing platform at a maximum altitude of 14.4 meters and with a maximum x lateral displacement of up to 5 meters, with very little error for both quantities. Additionally, the design of the landing platform fiducial cluster allows for the reliable recognition of the landing platform for a large range of altitudes.

Further research and development would be required to decrease the device's component cost of $100. A custom-made solution devoid of off-the-shelf components would be cheaper, more lightweight, and easier to mass-produce.

Future Work

Beyond the implementation of the rest of the charging system described in section 2.1, the product can be improved in the following ways:

Form factor: The final prototype had dimensions of 5.84cm x 13.2cm x 4.5cm. These dimensions describe the size of the enclosure, and were as minimal as possible given the size of the components that had to be housed in the enclosure. While these dimensions did not exceed our design constraints, they are not optimal. The form factor of the sensor system and its housing can be reduced in three ways: using a smaller camera (non-USB), designing all of the components to fit on a single custom-made PCB, and creating a slim ABS enclosure for injection molding. There are a plethora of small but versatile processors, cameras, and rangefinders that if incorporated into a custom-made system would result in a product that was about the size of a modern smartphone.

UAV Identification: While it is possible to modify the landing platform to use different AR marker combinations in order to allow a landing UAV to distinguish between different platforms, a landing platform has no way of knowing what kind of UAV has landed on it. One possible solution is to add an RFID tag on the sensor system in order to give a contactless way for a UAV to identify itself to the landing platform. The RFID tag on the UAV sensor system would include identifying information about ownership, information about the type of battery it is powered with, and charge that the battery is currently at, and even information about the route that it is taking.

This feature would require a standardization of RFID readers on all landing platforms on a network.

Autonomous flight: While this sensor system was designed from the beginning with compatibility with the Pixhawk through a serial bus, it was not achieved within the time frame of the project. Thus, another aspect to work on is implementing a high-level guidance algorithm that uses the position and orientation information that is currently acquired from the visual and ultrasonic sensors.

Acknowledgments

I would like to thank my advisors for both academic terms: Professor Scott Kuindersma and Professor Radhika Nagpal. Additionally, all of the ES 100 and Teaching Labs staff were instrumental in helping me in completing this project. I would also like to thank the SEAS Faculty as a whole for helping me solve unprecedented problems. And lastly, I would like to thank my friends and family for their support.

References

[1] D. James, "10 Drones with the best flight times." http://www.dronesglobe.com/guide/long-flight-time/, 2015.

[2] N. Lavars, "Amazon to begin testing new delivery drones in the US." http://newatlas.com/amazon-new-delivery-drones-us-faa-approval/36957/, 2015.

[3] "Supercharger." https://www.tesla.com/supercharger, 2016.

[4] "Using UAV GPS." http://www.terrisgps.com/how-is-gps-used-in-uav/, 2016.

[5] W. T. Team, "Global positioning system (gps) standard positioning service (sps) performance analysis report," *Federal Aviation Administration*, 2016.

[6] G. Anitha and R. G. Kumar, "Vision based autonomous landing of an unmanned aerial vehicle," *Procedia Engineering*, 2012.

[7] A. Johnson, J. Montgomery, and L. Matthies, "Vision guided landing of an autonomous helicopter in hazardous terrain," *IEEE International Conference on Robotics and Automation*, 2005.

[8] S. Lange, N. Sunderhauf, and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments," *International Conference on Advanced Robotics*, 2009.

[9] "HD Pro Webcam C920 Technical Specifications." http://support.logitech.com/en_us/product/hd-pro-webcam-c920#download, 2015.

[10] L. Larmore, *Introduction to Photographic Principles. 2nd ed.* New York: Dover Publications, Inc., 1965.

[11] "Product User's Manual – HC-SR04 Ultrasonic Sensor." https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit, 2013.

[12] "Raspberry Pi 2, Model B Datasheet." https://cdn-shop.adafruit.com/pdfs/raspberrypi2modelb.pdf, 2013.

[13] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system.," *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, 1999.

[14] J.-L. Blanco, "A tutorial on se (3) transformation parameterizations and on-manifold optimization," *University of Malaga, Tech. Rep.*, 2010.

[15] F. James, *Statistical Methods in Experimental Physics (2nd ed.)*. Singapore: World Scientific., 2006.

[16] P. Schlyter, "Radiometry and photometry in astronomy." `http://stjarnhimlen.se/comp/radfaq.html#10`, 1997.

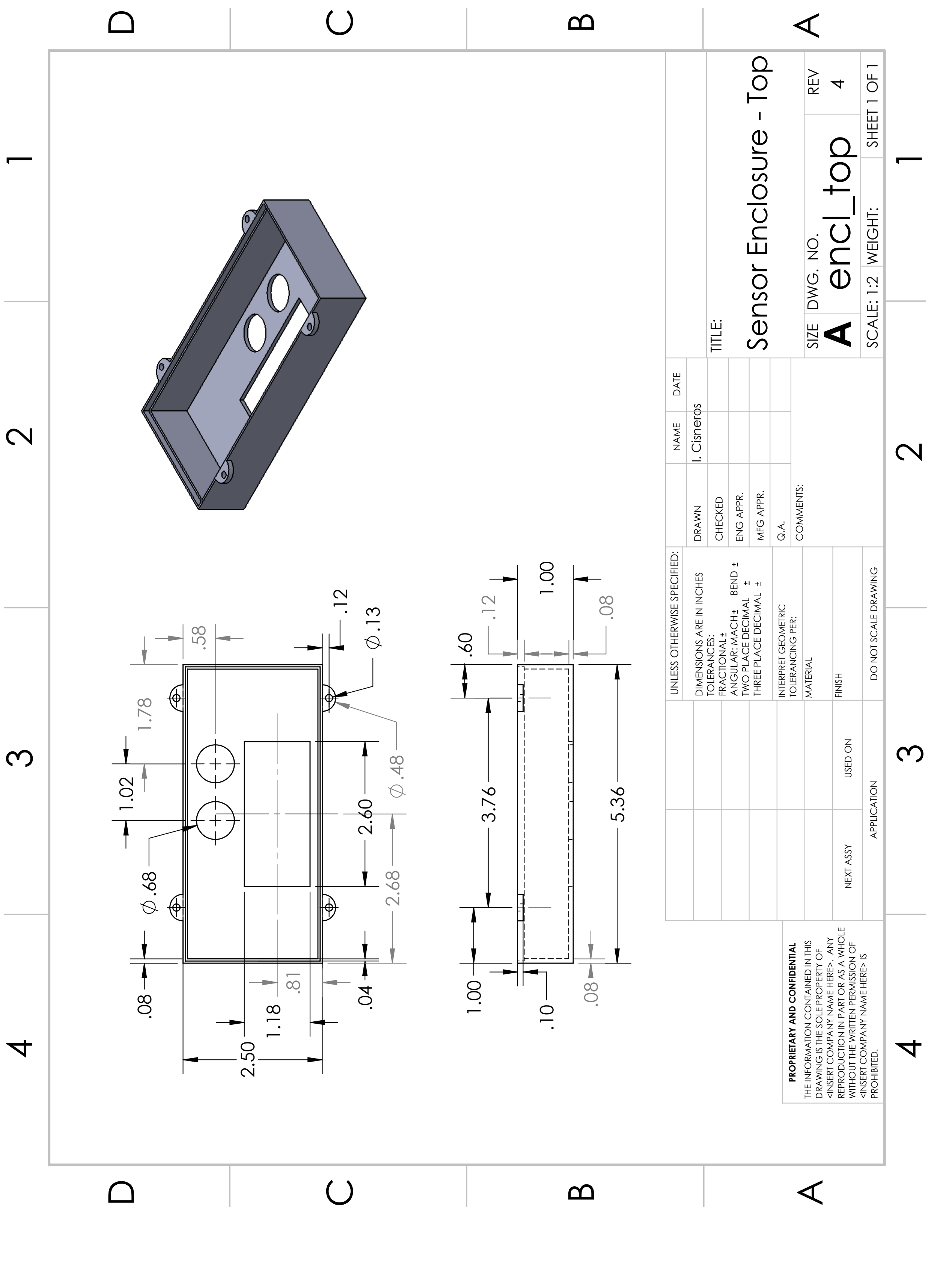
Appendix A - Bill of Materials

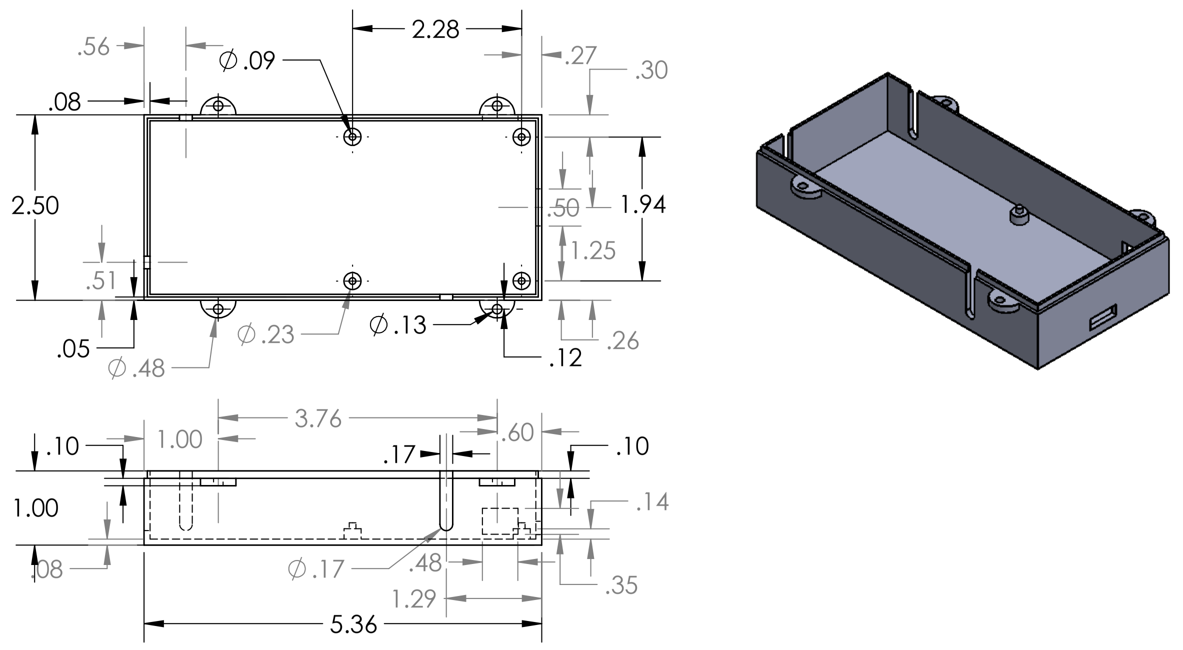
Item name	Supplier	Part Num	PPU	Number of units	Item Total	Link	Purchased through
HKPilot32	Hobbyking	387000050-0	$119.99	1	$119.99	http://www.hobby	ES 100
DJI Flame Wheel F450 ARF KIT	Amazon	B00G4A2RBU	$201.70	1	$201.70	https://www.ama	ES 100
Landing Gear	Amazon	B00NPPCEBA	$7.99	1	$7.99	https://www.ama	ES 100
Propellers	Amazon	B010U0JHXU	$10.98	1	$10.98	https://www.ama	ES 100
Radio receiver	Amazon	B009E95FGC	$59.99	1	$59.99	https://www.ama	ES 100
XT 60 connectors set	Amazon	B01E9HM7NC	$6.99	1	$6.99	https://www.ama	ES 100
battery	Hobbyking	Z30004S-20	$17.85	2	$35.70	http://www.hobby	ES 100
Lipo voltage checker	Amazon	B01DBZK7Y4	$4.99	1	$4.99	https://www.ama	ES 100
DC DC converter	Amazon	B008BHAOQO	$5.90	1	$5.90	https://www.ama	ES 100
Raspberry Pi 2	Amazon	B01ER2SKFS	$14.99	1	$14.99	https://www.ama	ES 100
HC-SR04	Amazon	B00E0NXTJW	$8.99	1	$8.99	https://www.ama	ES 100
Logitech c920	Amazon	B006JH8T3S	$64.00	1	$64.00	https://www.ama	ES 100
Buck Converter	Amazon	B008BHAOQO	$5.90	1	$5.90	https://www.ama	ES 100
				Total Price	$548.11		

Appendix B - Engineering Drawings

Engineering drawings of the enclosure.

The following CAD models were made in SolidWorks.


Sensor Enclosure - Top
TITLE:
DWG. NO.
encl_top
SIZE A
REV 4
SCALE: 1:2
WEIGHT:
SHEET 1 OF 1
DRAWN I. Cisneros
NAME
DATE
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.
.08
Ø.68
1.02
1.78
.58
.12
Ø.13
Ø.48
2.60
2.68
.81
1.18
2.50
.04
1.00
3.76
.60
.12
1.00
.08
5.36
.10
.08
A
B
C
D
1
2
3
4



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN	I. Cisneros	
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

Sensor Enclosure - Bottom

SIZE	DWG. NO.	REV
A	encl_bot	4
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1

Appendix C - Components and Additional Figures

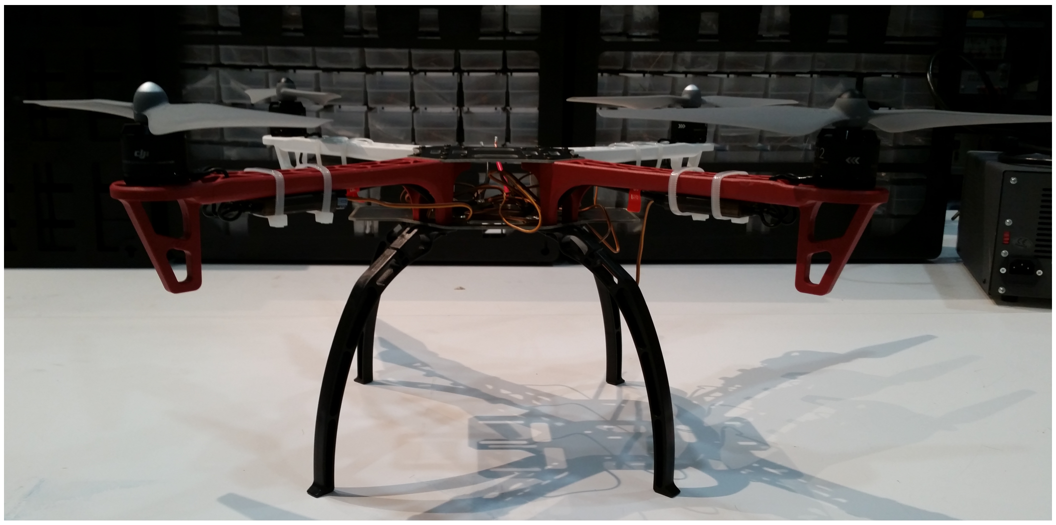


Figure C-1: Test UAV Original Frame.

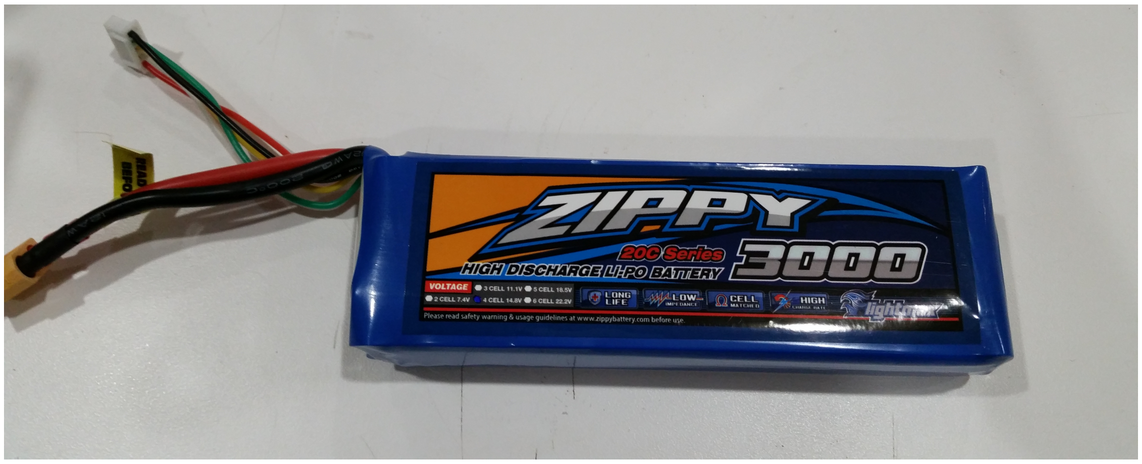


Figure C-2: The LiPo battery that we used.

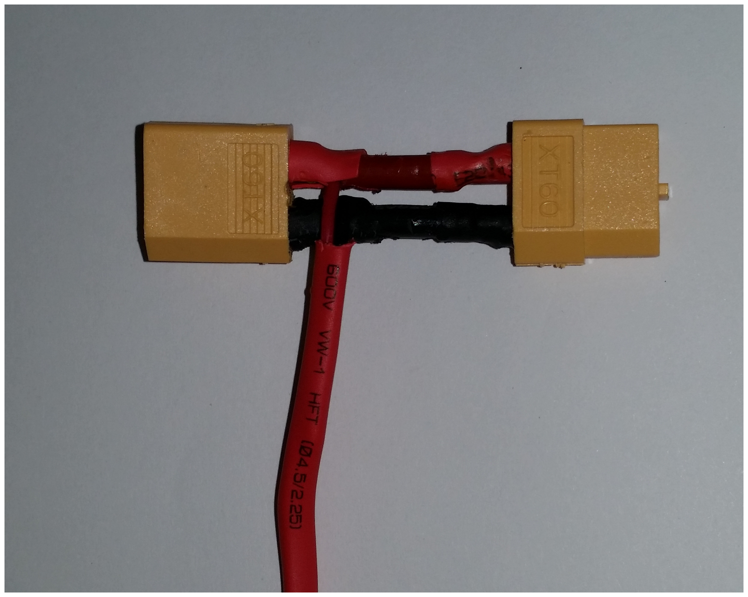


Figure C-3: The voltage splitter that drew power from the LiPo to our sensor system.

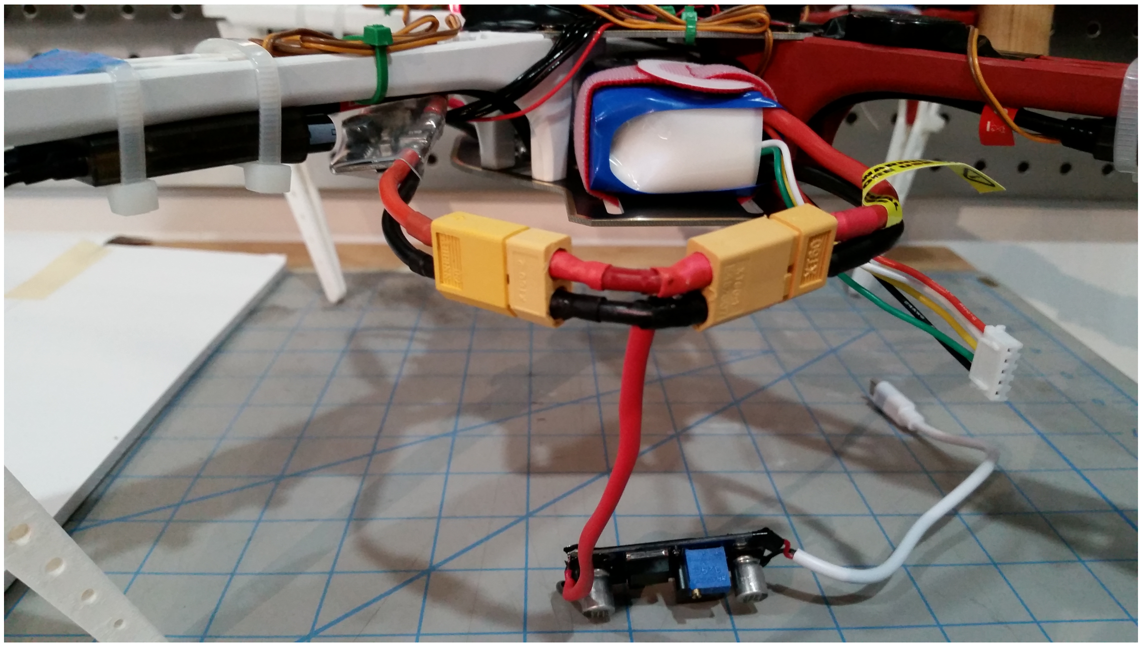


Figure C-4: The entire device power module integrated.

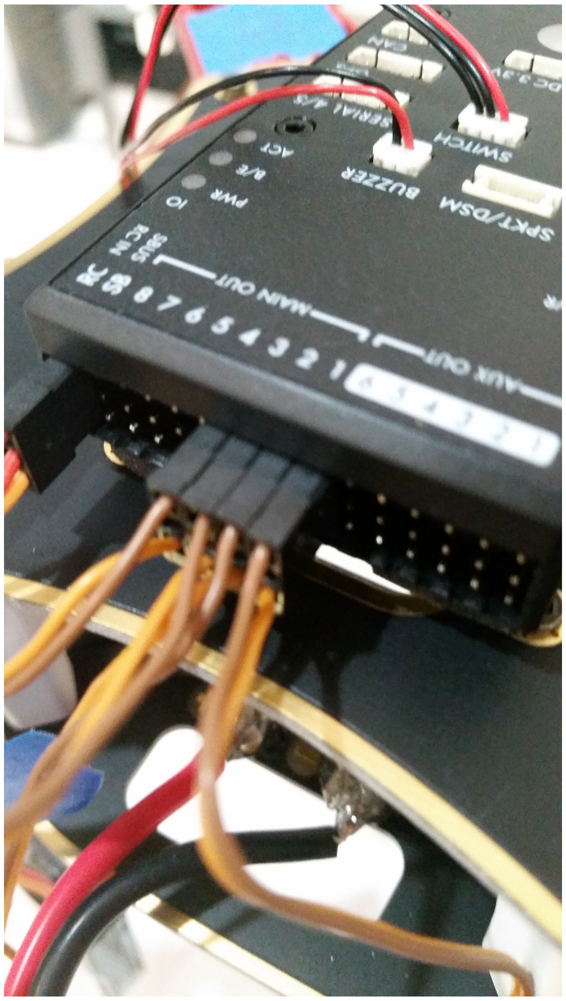


Figure C-5: Connections to the Pixhawk flight controller.

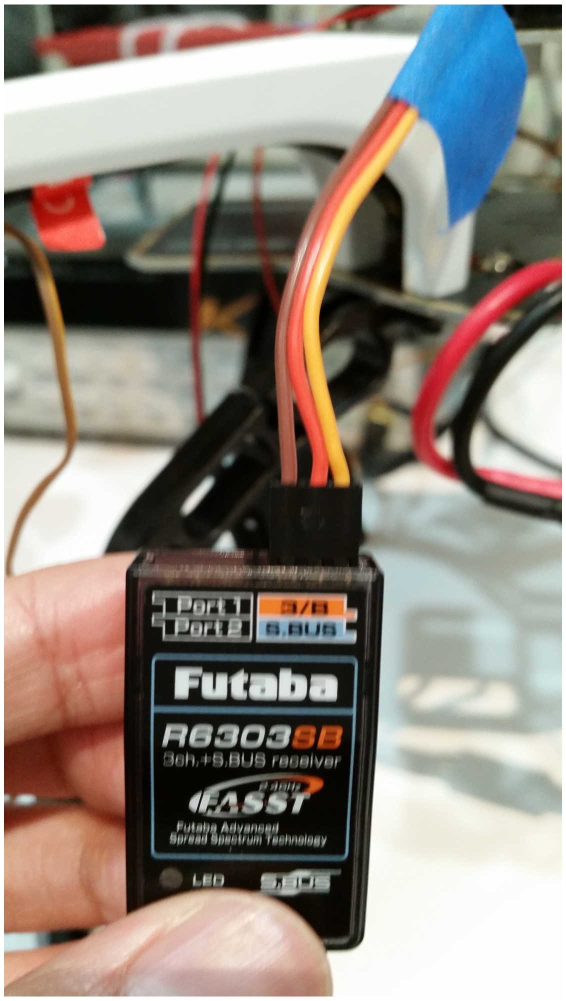


Figure C-6: The Futaba radio receiver.

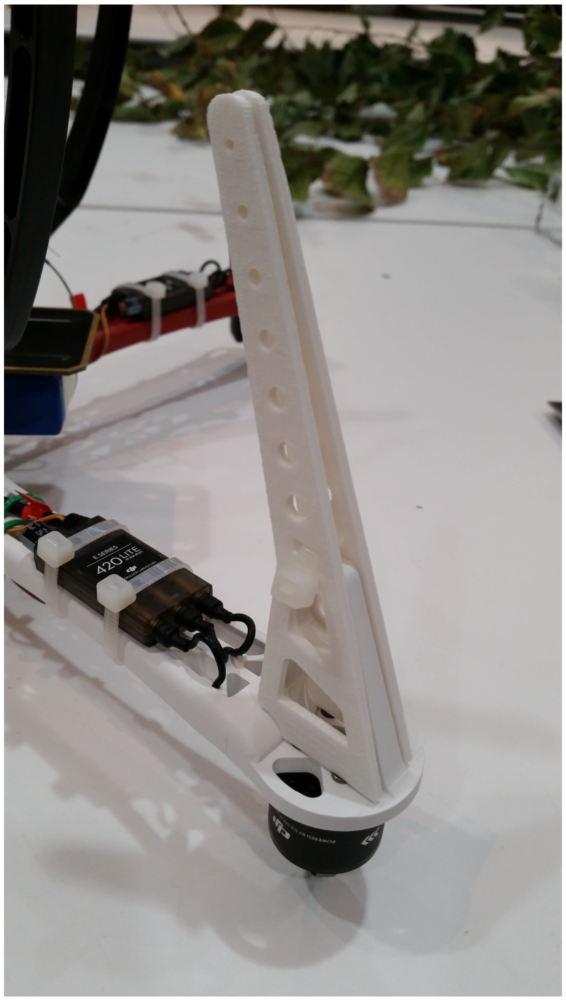


Figure C-7: The new 3D-printed landing gear attached to the UAV frame.

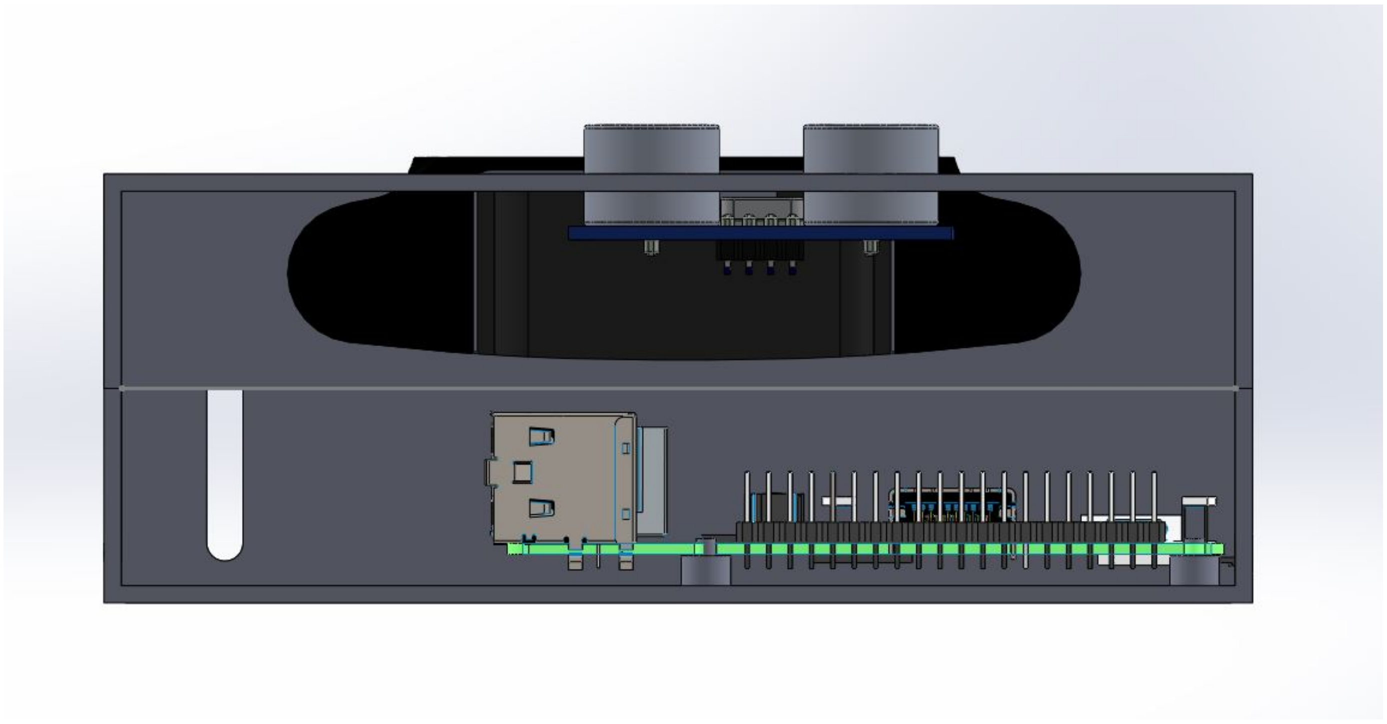


Figure C-8: A side view of the device in order to demonstrate the component integration. The space on the left hand side is for the power module and cabling.

Appendix D - Code

For a complete set of the code used in this project, please see the publicly available github repository found here: `https://github.com/icisneros/uav_landing`. A few smaller code samples are given here; the bigger, more complex scripts are thousands of lines long, so they are too lengthy to include in this appendix.

An example XML file which details how to handle a particular AR tag. This particular file defines a marker with ID = 1, and a size of 4.4cm x 4.4cm.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<multimarker markers="1">
    <marker index="1" status="1">
        <corner x="-2.2" y="-2.2" z="0.0" />
        <corner x="2.2" y="-2.2" z="0.0" />
        <corner x="2.2" y="2.2" z="0.0" />
        <corner x="-2.2" y="2.2" z="0.0" />
    </marker>
</multimarker>
```

The USB camera launch file (for activating the web-cam camera node). It incorporates some setting controls (like deactivating the autofocus):

```
<launch>
```

```
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
     output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
    <param name="camera_info_url" type="string" value="file
       ://$(find usb_cam)/usb_cam.yaml" />
    <param name="autofocus" value="false" />
    <param name="autowhitebalance" value="false" />
    <param name="focus" value="0" />
  </node>
  <node name="image_view" pkg="image_view" type="image_view"
     respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

The ar-tracking launch file (for activating the ar-pose node/topic):

```
<launch>
        <arg name="marker_size" default="4.4" />
        <arg name="max_new_marker_error" default="0.08" />
        <arg name="max_track_error" default="0.2" />

        <arg name="cam_image_topic" default="/usb_cam/
```

```
            image_raw" />
        <arg name="cam_info_topic" default="/usb_cam/
            camera_info" />

        <arg name="output_frame" default="/usb_cam" />
        <arg name="bundle_files" default="$(find
            ar_track_alvar)/bundles/tag0.xml $(find
            ar_track_alvar)/bundles/tag1.xml $(find
            ar_track_alvar)/bundles/tag2.xml $(find
            ar_track_alvar)/bundles/tag3.xml $(find
            ar_track_alvar)/bundles/tag4.xml $(find
            ar_track_alvar)/bundles/tag5.xml $(find
            ar_track_alvar)/bundles/tag6.xml $(find
            ar_track_alvar)/bundles/tag7.xml $(find
            ar_track_alvar)/bundles/tag8.xml $(find
            ar_track_alvar)/bundles/tag9.xml $(find
            ar_track_alvar)/bundles/tag10.xml" />

        <node name="ar_track_alvar" pkg="ar_track_alvar" type
            ="findMarkerBundlesNoKinect" respawn="false"
            output="screen" args="$(arg marker_size) $(arg
            max_new_marker_error) $(arg max_track_error) $(
            arg$
</launch>
```

The ultrasonic rangefinder class:

```
import RPi.GPIO as GPIO
import time
```

```
class RangeFinder:

    def __init__(self):
        GPIO.setmode(GPIO.BCM)
        self.TRIG = 23
        self.ECHO = 24
        self.calib_dist = 0  # in cm

    def initialize(self):
        """Assign the GPIO pins and delay for 2
           seconds while it settles
        """
        print "Distance Measurement In Progress"

        GPIO.setup(self.TRIG,GPIO.OUT)
        GPIO.setup(self.ECHO,GPIO.IN)

        GPIO.output(self.TRIG, False)
        print "Waiting For Sensor To Settle"
        time.sleep(2)



    def calibration(self):
        """Sets the calib_dist variable, which is a
           distance offset in cm.
           Should be called at the very beginning
              before any distance_read calls.
        """
        GPIO.output(self.TRIG, False)
        time.sleep(2)
        self.calib_dist = self.distance_read()  # in
           cm


def distance_read(self):
        """Sends ultrasonic pulse, times how long it
           takes to detect an echo, and
           then calculates the distance based on the
              speed of sound in air at sea level.
           Uses the offset distance to zero out the
              distance reading.
        """
        # send ultrasonic pulse
        GPIO.output(self.TRIG, True)
        time.sleep(0.00001)  # 10uS pules
        GPIO.output(self.TRIG, False)
        pulse_end = 1980458336

        # listen for the echo
        while GPIO.input(self.ECHO)==0:
                pulse_start = time.time()

        # echo was heard
        while GPIO.input(self.ECHO)==1:
                pulse_end = time.time()

        # time differential; time.time() returns the
           time in seconds
        pulse_duration = pulse_end - pulse_start
        distance = (pulse_duration * 17150) - self.
           calib_dist  # 17150 = (34300 cm/s) / 2
        distance = round(distance, 2)  # distance is
           in cm

        # thresholds for accurate measurements
        if 0 < distance < 400:
                distance = distance
        else:
                distance = 0.0

        # distance in meters
        distance_m = distance / 100  # 171.50 = (343
           m/s) / 2

        # return distance # return in cm
        return distance_m # return in m



def wait(self):
        """To be called in between distance reads in
           order to allow the sensor to settle.
           Introduces a 1 second delay.
        """
        GPIO.output(self.TRIG, False)
        time.sleep(0.2)

def finish(self):
        """Called when done using the ultrasonic
```

```
            sensor.
            Uses GPIO.cleanup() function
        """
        GPIO.cleanup()
        print "Measurement stopped"
```

The AR tag processing function in any of the cluster identification scripts. This takes the raw data from the ar-pose topic and creates a state vector (which is stored in a dict where the tag ID is the key and the vector is the value) :

```
def processTag(self, data):
    """index into tag data (if available):
        data.markers[0].pose.pose.position.x
        data.markers[0].pose.pose.position.y
        data.markers[0].pose.pose.position.z
    """
    global Tag_Detected
    global Tags_Detected_List
    global Tags_Dict
    global Multiple_Tags

    if data.markers:  # make sure data is not empty

        Tag_Detected = True

        len_of_dict = len(data.markers)
        if len_of_dict > 1:
            Multiple_Tags = True
        for tag in range(len_of_dict):

            tag_id = data.markers[tag].id

            # position (in meters) rounded to the nearest
                cm
            tag_x = round(data.markers[tag].pose.pose.
               position.x, 2)
            tag_y = round(-data.markers[tag].pose.pose.
               position.y, 2)  # inverted in the camera's
                reference
            tag_z = round(data.markers[tag].pose.pose.
               position.z, 2)

            # pose angles (in radians).  range restricted
               : -pi to +pi
            tag_angles = self.quaternion_to_euler(data.
               markers[tag])

            # populate the global dict
            Tags_Dict[tag_id] = [tag_x, tag_y, tag_z] +
               tag_angles

        rospy.loginfo("tags_dict = ")
        rospy.loginfo(Tags_Dict)
    else:
        Tag_Detected = False
        Multiple_Tags = False
        # remember to empty the dict every loop
        Tags_Dict = {}
```