Reactionary Targeting Defense System (RTDS)

Group 19

Joseph Musante - Computer Engineer

Chance Reimer - Computer Engineer, Electrical Engineer

Calvin Sands - Computer Engineer

Table of Contents

Table of Figures

Table of Tables

1. Executive Summary

The Reactionary Targeting Defense System (RTDS) is an automatic drone-targeting turret. Using computer vision and a targeting algorithm which can be seen in Figure 3, the RTDS will look for and identify flying common quadcopters (or more generally, drones) in the nearby vicinity using a high-resolution camera. Upon identification, the RTDS will use a simple laser pointer to mark the target. The camera and laser pointer will be mounted on top of two servos providing a dual-axis range of motion, allowing the RTDS to pan the view of the area all around it. The general hardware layout for this design can be seen in Figure 2. It will also be capable of following the drone as it flies by, keeping the drone in the center of its vision and maintaining the laser point marked onto the target as it continues to move.

This document explores the research and development processes we have undertaken in order to make possible the implementation of our turret design. The following section defines much of the goals and specifications the RTDS intends to meet, including the motivation behind such a design in the first place and how our design will accomplish those goals. Those discussed items will direct and motivate all the decisions we will make regarding the actual implementation of the RTDS.

Following the design layout, this document enumerates all of the research we have performed and gathered in order to select the components and software that will make up the physical RTDS. Research performed on previously made designs, similar products that exist in the industry, and their comparisons to our intended design that helped to guide our decision-making process is provided. Additionally, each of the components – both physical and digital – used in the design needed to be compared against many different options. Most notable among the researched components include the types and model of servos to move the camera, the microcontroller board intended to control the servos, the development board that will house all the processing for the artificial intelligence for the image recognition, and finally the image recognition algorithm and software itself. All of these components have a variety of communications standards that they could all use, as well as written software languages of different types and styles, that are all compared and contrasted with one another in the context of the hardware they are meant to be implemented on. Finally, some special mention of the artificial intelligence itself is made, regarding how it is trained, what it is trained with, and how these final decisions are impacted by and will impact the selected hardware.

Design specifics, including more detailed constraints and industry standards that are relevant to the RTDS, are enumerated in order to bound the scope of the RTDS and to increase reproducibility by using well-known design practices. By following these constraints and explicitly detailing our turret design, we hope that future engineers will be able to follow our implementation closely and be able to reproduce our results using either our exact methods and components, or be able to find similar or equivalent components and similar or equivalent industry standards with which to complete the RTDS turret design.

Finally, the integration of all the components and the actual construction plan of all the designs is detailed. The most notable of these construction plans is the design and layout of the printed circuit board to be used as the connection hub and power manager for the servos. Alongside the printed circuit board however are the mechanical plans to house all of the components together into a turret body, and the software architectural layout plan for the artificial intelligence algorithm and its use on its respective development board. All these systems have testing plans created from our previously defined project specifications and constraints that will enable us to determine the success of the implementation after assembly and integration is completed in the Spring.

All designs and plans enumerated in this document are done according to the overarching administrative constraints provided by the requirements of the Senior Design Project. This document addresses the schedule on which we intend to reach major milestones in our work pipeline, as well as the budget that our team has allotted for the RTDS.

2. Project Description

The goals set forward by this document for the RTDS will help to guide the design process in such a way as to help us maximize its effective utility. We hope to transmit these goals and plans through narrative description, diagrams, and tabular data, as displayed in this section.

2.1. Motivation

Unmanned Aerial Vehicles (UAVs), or drones, have quickly leapt from the defense industry and aeronautics research labs to the commercial sector. The great demand for maneuverable, light-weight drones such as the quad-copter in particular has led to many incredible advancements in their design. In turn this has massively increased their availability and capabilities, and subsequently decreased their cost. They can be equipped with live-feed cameras with a wide selection of image qualities, and the controls can be refined to make the drone extraordinarily agile. Nowadays, drones such as these are purchased by remote-control ("RC") hobbyists, photographers and journalists. An entire sport has even been conceived in the past couple of years around high-speed drone racing. These common quad-copters—equipped with live-feed cameras and capable of close to a half hour of flight time—can easily be purchased for under a hundred dollars. Conversely, more sophisticated models, such as those commonly used in agriculture for pesticide spraying, can cost tens of thousands of dollars.

The prevalence of such an accessible technology raises many questions regarding the ethics surrounding their use, especially since up to this point their regulation is largely unstandardized. It's clear that oftentimes they are restricted from being flown over private property (as per the wishes of the property owner) or certain public facilities such as airports and schools; Nonetheless, the means of enforcing these restrictions are very limited, mostly relying on hopefully finding the drone's owner and confronting them personally.

The defense industry is currently pouring tons of money into research that will help protect property and civilians from drone attacks. Being that drones are being used to carry out attacks on foreign countries it's only a matter of time until this problem is at our front door. The drones the defense industry is concerned about isn't the type of drones our project will be targeting, however the software and overall concept for such a device will be similar.

2.2. Project Goals and Objectives

To help enforce flying restrictions, answers to the actual drone itself exist, most notably in circumstances where the property owner or law enforcement is not even aware yet of the intrusion flying overhead. The design of a system that could be used to spot and disable drones in restricted airspace would help to act as a "first-responder" to their unwanted or illegal presence.

A device—such as a form of autonomous anti-drone turret—capable of this would create a virtual protective bubble over the property, preventing a commercial quad-copter drone from entering the space. Should a drone enter the protected air, the turret could be equipped with any variety of detection tools to identify and track the drone, communication tools to notify the operators of the turret, and launchers or emitters to interfere with or halt the flight of the drone. Furthermore, given a significantly large market presence, the potential of the presence of this turret would further act as a deterrent, much in the way that modern surveillance psychology works.

Existing forms of drone detection systems exist currently, with one such product, the "Drone Detector", capable of video identification of a drone from up to 100 meters away. This product focuses on simply detecting the drone, however, and includes other detection methods as well such as audio detection from up to 40 meters away, and even radio frequency detection up to 1000 meters away.

Additionally, many methods for manually disabling a drone exist. The goal of Battelle's DroneDefender device is to jam the GPS and Industrial, Scientific and Medical (ISM) radio frequencies of the drone, causing it to fall from the sky (this technology is still awaiting authorization from the FCC, however). Although it is mired in legal complications, civilians shooting down intruding drones with privately-owned firearms is another method that state courts have had to deal with recently as well.

Our goal is to pioneer a platform on which the drone-detection and drone-disabling technologies could be fused together, with high-powered machine learning at its core. We intend to accomplish this by demonstrating the unison of a rudimentary detection system autonomously providing identifying information about a drone to an integrated targeting system.

2.3. Functionality

Our project proposes the design and implementation of an intelligent anti-drone turret.

The turret will be equipped with a high-definition camera atop a double-jointed articulation, allowing it to scan its immediate surrounding airspace for intruding quad-copters. The turret would be trained to identify drones at different altitudes and distances using machine learning. Upon detection, the turret's camera will "lock on" to its target, maintaining the drone in the center of its vision using a reactive tracking algorithm, following its path as it continues to fly. To demonstrate its high precision targeting, a low-powered laser would be used to mark its target and would be signaled to turn on or off by an operator. The turret would be relatively small, with a base area approximately the size of a shoe-box, making it both discrete and portable.

The focus of the system is responsiveness, and its ability to rapidly interpret sensor data to acquire targets, simultaneously creating range estimates for engagement. The system should send updates to an operator specifying what current mode it is in, and battery level. The system will be capable of displaying its video feed to an operator during all three modes and will show the post-processed image with any possible or engaged targets enclosed within a color-coded box. The modes that should be supported by the drone are: Idle, Engaged, and Possible Target.

During Idle mode, the operator should only receive relevant data such as the system status including current battery level, and orientation of the camera feed including azimuth and degrees of rotation.

During Engaged mode, the operator should receive the processed video data enclosing the target in a red rectangle, with the hardware's estimation for distance, and its confidence rating for a successful shot. The hardware will also estimate the movement of the drone, displaying velocity and direction. Upon firing of the target with laser, the system will also notify the operator, with number of shots, including metrics of successful hits. The system will also notify the operator if the drone was downed.

During Possible Target mode, the system should notify the operator, and display the target enclosed with a purple box. The operator will have an opportunity to send a command to the system to engage or to go back to an idle state. The system will maintain a lock on the specified target until it is no longer in view, an object with a higher confidence rating of being a target comes into view, or until an operator sends the idle/attack command.

2.4. Specifications and Requirements

The requirements given to the RTDS are done so with the intent to shape the narrative of how we envision the turret to operate. By narratively defining them, we are able to generate individual specifications by extrapolating the most important details from the vision of the functioning design.

Ensuring that our project has set engineering requirements is arguably one of the most important steps in ensuring that our project is successful. The requirements below in Table 1 are all of the technical needs that our project must abide by to ensure we have achieved our goals. It is very important that our requirements were thought out and reasonable for they must be demonstratable in order for success in senior design 2 next semester.

Due to the main targeting sensor being a camera, the targeting system is required to be operational during the day, and without obstruction of view of target. System must not be exposed to any inclement weather such as rain. Due to high data bandwidth needed to send live feed from system, it is assumed that the system is able to connect via a wired connection, or a wireless connection with sufficient speeds to view camera feed in real time. The drone will only have to target a maximum of one drone within its confidence range.

These known facts about the operational conditions of the RTDS turret help to enumerate the requirements that will be followed in the continued design. A few example details implicated by the operational conditions include a minimum communication rate between components, and a maximum number of targetable drones.

The four major sub-categories of our defined requirements are for the tracking algorithm, the display and communication between components, the functionality of the firing apparatus, and the power that will be designed for the custom printed circuit board to enter the system to power the servos and firing apparatus.

Description	Value	Unit
Tracking Algorithm		
Hardware must be able to target one drone	-	-
Hardware must be able to track targets within a specific radius confidence range	5	ft
Hardware must be able to track targets within a set field of view	180	degrees
Hardware must be able to follow target movement for accurate firing within a specific range	6	in
Display and Communication		
Hardware must display the camera feed and contain target within a digital rectangle	-	-
Displayed feed must show when system will fire	-	-
Displayed feed must show processed sensor data in real time	<10	ms
Hardware must support ethernet or wireless connectivity for display feed	-	-
Firing Apparatus		
Firing apparatus must be able to track targets within a set field of view	180	degrees
Power		
Hardware must connect to AC power	<15	watts

Table 1: Requirements

2.5. House of Quality

One important element that has greatly affected the design and planning of the RTDS turret is the fact that not all requirements and specifications are created equal. That means that, while all requirements must be planned and accounted for in the design, some requirements will be more important than others and will thus by necessity carry a greater weight with them into our planning. By identifying these most critical requirements in the earliest stages of design, this helps to maximize the most identifiable attributes – those that will define the success of the RTDS – and to minimize time spent on less important attributes that would have a minimal effect on the design's success.

The house of quality helps to identify which of our requirements and specifications are those which are most critical. By creating cross-sections of each of the customer's needs with that of the controllable requirements and specifications that we have defined in our preparatory design stages, we create intersections in which the voice of the customer regarding each of those specifications is able to echo whether or not it is of great value or importance. Figure 1 is the house of quality for the RTDS design defined here in this document.

The three specifications highlighted in red are those sought to be specifically demonstrable and of most critical impact through our implementation of the RTDS. Specifically, the RTDS will be able to perform rapid target selection of a drone in under 10 milli-seconds; It will be able to target one drone within its viewfinder; And it will successfully identify a drone at least 75% of the time.

Compared to the remaining four requirements, the three red-highlighted requirements are quite evidently those which will not only be the most demonstrable, but also those through which we will be able to measure the success of our design. For example, while maintaining a fixed budget is critical for the successful design of just about anything, exceeding the requirement for cost by a factor of 10% would not be an indication for a lack of success. On the other hand, if our rapid target acquisition regularly took 10% longer to identify any targets, or our target accuracy was 10% less accurate than intended, these would be glaring inadequacies in the face of our design results. By understanding that these three requirements are those that are most important, we simultaneously understand where we must focus most our efforts, as well as where we can safely save time and resources without negatively impacting the final operational success of the RTDS.

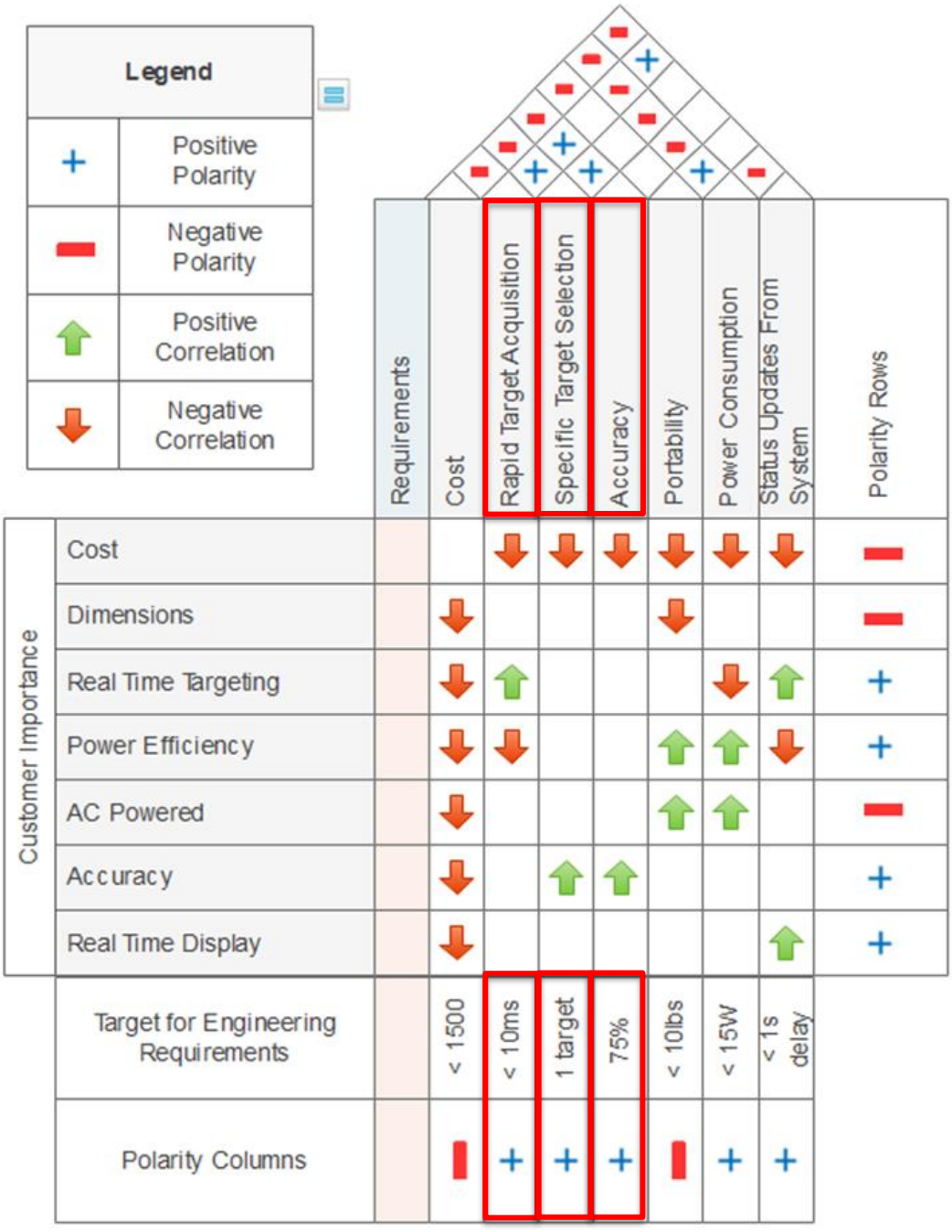


Figure 1: House of Quality

2.6. Hardware Diagram

Below is the general design for our hardware can be seen that goes over the main hardware components. What's worth noting is that there's a lot of software that goes into each of the components. The general layout for the hardware is quite straight forward where the sensor system outputs data to the development board to process the image. The development board determines whether or not a drone is in the image and then sends data about the position and velocity to the PCB for the sensor system.

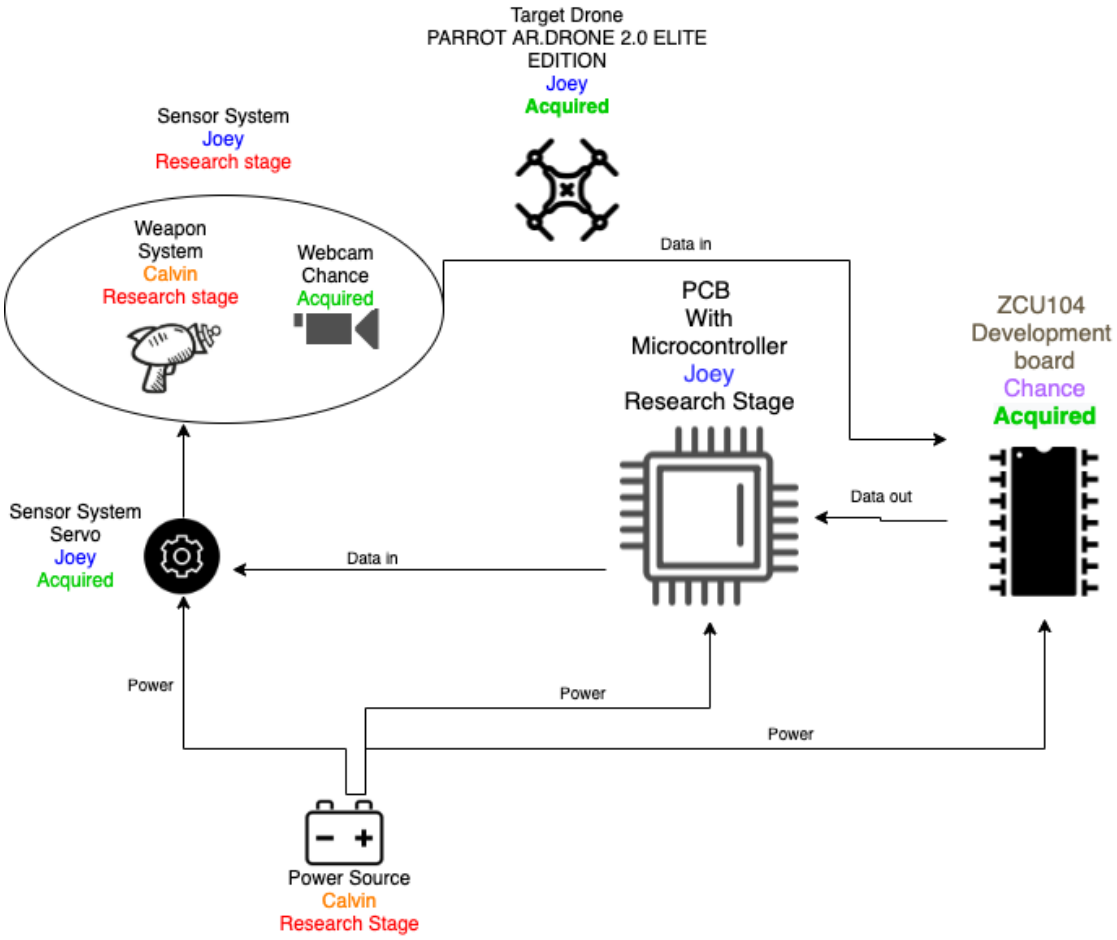


Figure 2: Hardware Diagram

2.7. Software Diagram

Below is a very simple software diagram that is a general overview upon how our project will function. From the very beginning where it starts processing sensor data to determining whether or not it's a drone and taking the appropriate actions. Each one of the blocks on the software diagram require multiple steps such as predicting the drone's location, however the general software flow can be understood by viewing Figure 3 below.

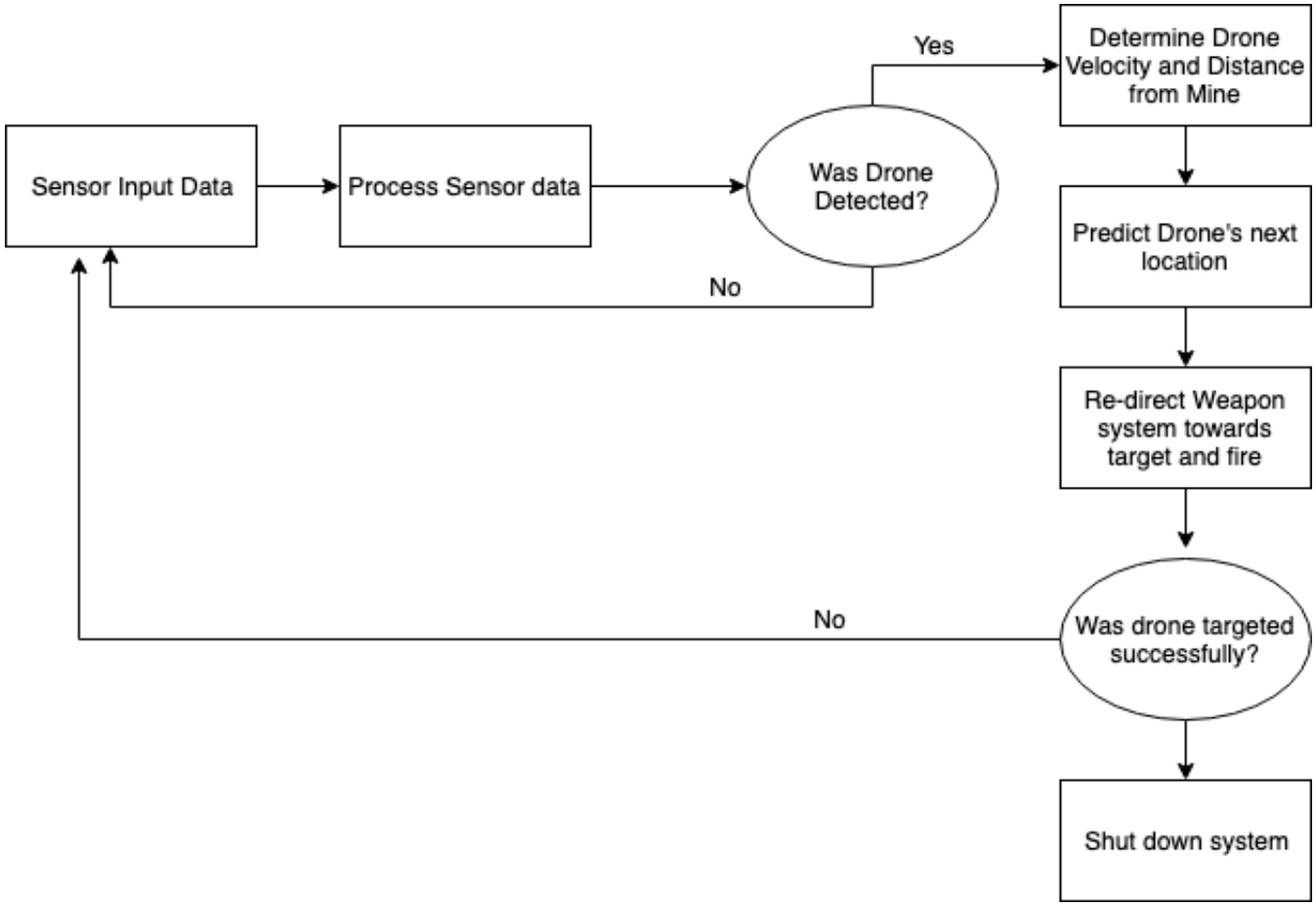


Figure 3: Software Diagram

3. Project Research & Part Selection

The first and one of the most crucial steps of implementing this project is doing enough research to ensure we pick not only the most affordable parts, but also parts that will lead to the most effective design. By reviewing previous similar projects and comparing some of the advantages and limitations to their design we will be able to ensure that our project will have the best of both worlds. Choosing the correct components is crucial for a successful project and each component we choose will have various advantages and disadvantages. By reviewing the requirements for our project, we will determine which parts will give us a good balance of cost to functionality.

Another aspect to consider is compatibility between parts, for some parts will simply not be compatible with others. Ensuring proper compatibility between each part that is picked is crucial. This means some of the parts we think would be the best fit must be limited out for we wouldn't be able to use them.

For instance, say we wanted to use UART as a communication protocol between the microcontroller and the development board so we could support a higher baud rate. Although this sounds good on paper and perhaps the development board supports it the microcontroller may not. On top of the microcontroller possibly not supporting it, perhaps the microcontroller has a cap set on the baud rate that would make using the UART communication protocol a waste of resources. By looking at the requirements and finding software and components that will support them while binge compatible with each other we can ensure success.

Significant research will be required for every bit of hardware we decide to use and every bit of software we decide to implement. As previously stated almost every component will have some sort of interaction with others and a choice for one may limit the choices for another.

For this research we will be considering different microcontrollers, servos, devolvement boards, communication protocols, NEURAL NETS, computer vision algorithms, and other various components.

3.1. Existing Projects and Products

Being that a lot of the similar products are built for the defense industry most of that is classified and there isn't much documentation on it. However, there are some similar projects that will be discussed which have some of the properties that our project is aiming for. By examining similar products, it will help guide us in the right direction upon what works and what doesn't.

Our goal is to not only exceeded our own expectations, but also of the similar products out there except for maybe some of the military ones for we simply don't have the budget for that. Our hope is that with an efficient, reliable, and effective design we will be able to help others in the future of protecting themselves from rogue drones.

3.1.1. The Phaser

Being that this is a product which is built for the US Air Force by Raytheon the exact details of this design is certainly classified. However, Raytheon has built a device which is about the size of a storage container. This device is very sophisticated and uses a microwave-based weapon to take the drone down. It fires its blast of radiation directly towards the drone at the speed of light so it can bring down the drone in a split second. One of the many benefits of using an RF based weapon is that it doesn't require "Ammunition" Per say. This means that as long as the device isn't physically damaged and has access to power it will be able to continue shooting down drones forever. The beam that is sent from the phaser is 100 meters broad and can target drones from the distance of 1KM. The Phaser is also said to use an electro-optical sensor to help track drones. One thing it doesn't mention is if this system is fully autonomous or not, it may require an operator to confirm a target before firing.

Figure 4 Phaser System. Image used courtesy of Cal Jeffery.

3.1.2. Drone Defender

This solution to the problem isn't nearly as advanced or pricey as the Phaser mentioned above. Drone Defender is a handheld weapon which claims to use GPS disruption and remote-control disruption. With an operating time of 2 hours of continuous use it sure can take out some drones, however being that it weighs a full 15 pounds someone's going to have a hard time pointing this to the sky that long. On top of that being that this requires human interaction it means the sky will only be protected as long as someone is watching it. This isn't ideal for a continuous solution would mean more protection, and less time wasted. Also, this is relying on a human to detect the drone whereas our project will be fully autonomous and will be able to find and target a drone all by itself.

3.1.3. Net Gun

A product made by a company called the net gun store was originally used to help safely catch animals from a distance. However due to the increased nuisance of drones in the past year their product is being used to catch drones. Their product uses compressed gas to shoot a net from a distance of up to what looks like to be around 10 feet away. What separates

this product from the two above is that being it uses compressed gas to fire a net, it's commercially available for anyone to buy. Whereas the two products above are only sold for government and military applications, the base model of the net gun can be purchased for around $900. The main limitation for this product is the range, being that it's only using compressed gas the net can't be fired over long distances. As we have seen above there are many other ways of taking down a drone, if they were to implement them their product would no longer be available to the general public. Apart from the distance this product also lacks from the fact that it requires a person to not only find a drone, but to also take it down with the one shot it has loaded. When compared to the other two products above which have a number of shots only limited by power consumption, a single shot to take down a drone many not be enough.

3.2. Microcontroller Research

Our project will require the use of a single microcontroller to control the servos which will be used to pan and tilt our sensors to detect drones. The microcontroller will also receive commands from the development board such as when a drone has been detected and instructions on how to track it. This microcontroller will need to have a fast-enough processor to be able process commands to rapidly switch from a sweeping mode to a tracking mode and also control the servos at the same time. On top of that the microcontroller will need to have enough GPIO's to power both servos as well as send them control signals.

3.2.1. Microcontroller options

There are dozens of microcontrollers currently on the market to choose from, all having their own benefits and drawbacks. Our first pick would be a generic Arduino Uno for its widely popular and thus has a plethora of community support. On top of that we all have experience with this microcontroller outside of school where we just used the MSP430. Being that some microcontrollers are built for a specific purpose such as built in wireless communications and an abundance of GPIO's we will need to critically evaluate each microcontroller to find out which one will meet our needs most effectively.

3.2.1.1. TI MSP430G2553

Being that we all have used this board in a couple of our classes this was defiantly a board to consider. One of the downsides to choosing this particular boards compared to an Arduino per say is lack of community support. Being that an Arduino is much more widely used, when it comes to implementation of controlling servos and inputting information, we may have a more challenging time. With that said some of the benefits is that it is ultra-low power only consuming 230µA when the CPU is operating at 1MHz and only 0.5µA in standby mode. Although the small power consumption is inviting being that were using AC power for our project power consumption isn't much of a concern for now. The fact that this board has 23 GPIO pins is certainty helpful however we won't need nearly that many to control two servos.

3.2.1.2. Raspberry Pi Zero

Choosing this microcontroller would no doubt be overkill, however using one would allow us to easily add some more functionality. Say we wanted to have a display showing the current pan/tilt angle for the servos supporting the camera and light, implementing this with a Raspberry Pi would be much easier than of the other choices. With that said, using this would be unnecessary being that they run an entire GUI biased OS with built in Wi-Fi and Bluetooth. Not that it's very important since we're using AC power, but it would require much more power being that idle draws 90mA. All we need a microcontroller to do is take in some simple commands from the devolvement board and rotate some servos. With that said this board does have much more community support when compared to something like the TI MSP430G2553 board. For these reasons and many more we decided to pursue other options for a microcontroller.

3.2.1.3. Arduino Uno

Finally, we get to the breadwinner that is the Arduino Uno, with its simplistic design cheap parts and great community support this seems like the best choice for our project. This microcontroller is based on the ATmega328P chip which has an 8-bit microprocessor with up to 16MHz throughput. The Arduino Uno has 14 digital input/output pins and6 analog input pins, which is more than enough to control two servos but also not complete overkill. The ATmega328P processor cost's less than $3 so it will also be very cheap to build compared to the processor for the Raspberry Pi Zero. Also being that some of us in our group have already implemented servo control using an Arduino Uno, we're already off to a good start compared to some of the other boards that will be compared next.

3.2.1.4. ATtiny86

The ATtiny85 is a board which we are all rather unfamiliar with so right off the bat that would make implementation a bit more challenging. With that said it is a high-performance low power 8-bit microcontroller with considerably good specs. Only having 8 GPIO pins wouldn't be an issue being that we only need to control two servos. Another benefit of this board that's relatively unnecessary for us is its size, this microcontroller is tiny and if we wanted to try and make our design stealthier this would perhaps make it a better option over something huge like the Raspberry Pi Zero

3.2.1.5. ATMega32U4

The ATmega32U4 is the "Mega" version of the ATtiny86 and it goes without saying supports more functionality such as 32Kb flash memory instead of 8KB in the ATTiny86. Another benefit of the "Mega" is its support of UART, extra SPI port and 44 GPIO pints which for our project is 100% overkill. With all of this said this is a great microcontroller for the money and if we were using multiple cameras or other sensors may have been a more appropriate choice.

3.2.1.6. TI CC2540

Next up is the CC2540 which to be honest I've never even heard of before beginning this research so that's mildly concerning. However, it's a board that's somewhat similar to the M24LR Discovery being that it has some built in wireless components built in such as

Bluetooth 4.0. Also, when compared to the other microcontroller's it has highest clock rate, most flash memory, and RAM. However, considering our needs for a microcontroller all of this would be overkill.

3.2.1.7. Beagle Bone Black

The last microcontroller that we considered was the Beagle Bone Black which is a low-cost, community-supported board which packs quite a punch. Hardware wise it has 512MB DDR3 RAM, 4GB flash storage and a dedicated 3D graphics accelerator. Connectivity wise the Beagle Bone Black supports Ethernet, HDMI, and 2x46 pin headers. The board claims to boot Linux in under 10 seconds and is only $25! For the money, this is a really great microcontroller, however much like a lot of the other ones, it's way overkill for our specific application. With that said, given its very affordable price and speed compared to the others this is still an option for it would help cut down time in processing data to and from the Devolvement board.

3.2.2. Microcontroller Comparisons

After reviewing all the microcontrollers, we narrowed it down to just two boards, the Arduino Uno, and Beagle Bone Black. The main factors in this decision were community support, speed, cost, and GPIO pin count. Table 2 lists the comparison's where the color green means good, yellow decent, and red poor.

3.2.2.1. Community support

There is no doubt that the Arduino Uno is one of the most widely used microcontrollers and with that said would have the most documentation and tutorials available online. However, since we only need to control two servos one for panning side to side and another to move up and down implementation shouldn't be too challenging. The only main issue comes into play when considering the fact that were not allowed to use an off the shelf microcontroller and have to build our own into a printed circuit board. With that said community support is crucial if we are to have any issues with communication or just overall building problems. Using a much more popular microprocessor such as the ATmega328P will expediate any troubleshooting issues that will most likely occur at some point. It's estimated that back in 2013 around 700,00 Arduino Uno's are being used and considering that was from six years ago there are definably more being used now. When compared to the 100,000 BeagleBone Blacks sold back in 2013 that's significantly less people that may have ran into the same issues that we may come across.

	MSP430	Arduino Uno	ATtiny85
Power consumption	0.851 mW	0.740 mW	0.370 mW
Max Clock	16 MHz	20 MHz	20 MHz
GPIO pin count	16	23	8
Current output per pin	48 mA	100 mA	40 mA
Flash memory	16 KB	32 KB	8 KB
RAM size	512 B	2 KB	512 B
Cost	$2.32	$24.95	$1.20

Table 2: Microcontroller Overall Comparison, with the chosen Arduino Uno highlighted in red.

The values above for power consumption were calculated by using the formula P=VI, where V is the voltage assumed to be 3.7V and I is the current at 1MHz. Due to the application of simply moving some servos back and forth we assumed that we would be at a low clock rate and normal operation temperatures. Although overall power consumption isn't much of a concern since were using an AC power source, every engineer is always focused on efficiency of their design. When looking at Table 2 above its quite clear that for the most part all three microcontrollers are about equivalent, however the community support for the Arduino Uno is hard to beat.

3.2.2.2. Cost

Luckily cost isn't too much of a concern for our applications being that we only need a single microcontroller to suit our needs. With that said being that we don't have any sponsor, we'd defiantly like to keep the cost low if possible. However, we're not going to possibly make our project more challenging just to save $10 or so.

As shown in table 2 its quite clear that the ATtiny85 is the cheapest microcontroller that we are considering coming out to $1.20 per unit. The MSP430 is also clearly significantly cheaper than the Arduino Uno coming out to only be $2.32 per unit. Given these prices if we were going to try and implement this project with multiple "mines" all controlling their own servos then using one of these cheaper microcontrollers may have been necessary. Being that the Arduino Uno is able $25 is defiantly significantly more expensive but when it comes to troubleshooting any issues, we occur we will end up saving time which in turn is money.

3.2.2.3. Memory Size

Although memory size isn't crucial since we won't be storing any data on the microcontroller and the sweeping algorithm for the servos will be relatively simple maximizing our data size is beneficial in case, we end up adding more features down the road. Now there are two types of memory to take into consideration here, flash memory and RAM. Flash memory is where the program telling the microcontroller to sweep its servos will reside, so even though our program shouldn't be too hefty it's always nice to have some extra space. Although this won't make much of a difference in the grand scheme of things, flash memory performs fastest when it's not completely full, so with that said the more extra space we have in flash storage means the faster the program will execute. Now RAM is where executing the program and any other external calculations occurs.

From Table 2 one can clearly tell that when the comes to flash memory size the Arduino Uno comes out on top with a twice as much memory as its competitors that only have 16KB. As previously mentioned, there's no way our simple program will use up that much storage, but in order to ensure a future of expanding the functionality of our project more storage is key. When it comes to RAM the Arduino Uno starts really showing why it's worth the money supporting a full 2KB compared to the tiny 512 B that the MSP430 and ATtiny85 support. The use of this more RAM will come into play when the microcontroller is performing its routine sweep and is told by the development board to switch into a tracking mode and is given specific orders on where to direct its servos. Since the drone will by flying through the air, we and this microcontroller to respond to input as fast as possible to ensure an accurate and quick response time.

3.2.2.4. General Purpose Input/output

The number of GPIO pins isn't a crucial factor of deciding what microcontroller to use being that we only need to control two servos. However, in order to leave room for future expansion with possibly more sensors or an upgraded weapon system more GIOS is crucial. Each servo requires the use of 3 pins on a microcontroller, two of which are for power and the third needs to be connected to a digital GPIO pin. Since we are going to be implementing a pan and tilt functionality, we need two servos. These servos will need to be independently controlled meaning that we won't be able to use the same digital pin to control both servos. So, the use of at least two GPIO pins is a must have in order to complete our project. Once again, the Arduino Uno comes out on top with a total of 23 GPIO pins while the ATtiny85 has only 8 and MSP430 has 16. When comparing the three of these microcontroller's the Arduino Uno's 23 GPIO pins is defiantly overkill and we could comfortably settle for the MSP430's 16 pins. Being that we want to design this project with a possibility for future improvements the ATtiny85 8 GIPO pins just isn't enough.

3.2.2.5. Clock Frequency

Clock rate is rather important for our microcontroller when it comes to controlling the servos. Being that servos rely require motion with precise control and swift response to commands a higher clock rate is better. However, being that all the microcontroller's that we are comparing have a clock rate higher than 16MHz this shouldn't be much of a concern at all. But when it comes to executing instructions such as telling the servo to turn to a

specific point having a higher clock rate can defiantly be helpful. We want our sweeping algorithm to run as fast as possible, and more importantly when the development board send a signal to switch from sweeping mode to tracking mode we want as close to a real time response as we can get. This being said having a higher clock rate means more instructions executed per second which in turn increases the execution speed of when an instruction is sent to the microcontroller. The clock rate will also somewhat dictate power consumption of the microcontroller but being that we're going to use an AC power source instead of a battery this shouldn't be much of an issue at all. All in all, a higher clock rate will result in not only a faster time switching between sweeping and tracking mode it will also provide a stronger platform for future improvements that could require a faster processor. This is why the Arduino Uno's 20MHz is a solid choice for our project being that it will leave plenty of room for future improvements to our project.

3.2.3. Microcontroller Choice: Arduino Uno

We chose this specific microcontroller for many reasons, apart from it meeting all of our requirements we mainly chose it for the abundance of community support available. The Arduino Uno is unarguably one of the most widely used microcontrollers thus finding support for implementing our project will be as straightforward as possible. On top of that there are many libraries available for the Arduino Uno for controlling servos which is key when it comes to ensuring we are sending command signals as quickly and reliably as possible.

3.3. Servo Research

Selecting the correct servo is one of the key parts for implementing our project correctly. A servo which moves too slowly or too fast would render our sensor system useless. On top of the speed of the servo we will need to select one which will have enough power to pan and tilt our sensor system without being complete overkill. One of the first factors to consider will be the weight of the load the servo is expected to move. This will be a crucial factor for not only the pan functionality but mainly the tilt. When we tilt the sensor system up for instance, the servo will have to remain in a constant position which requires it to have enough torque to support the weight sensor system. Choosing a servo that can't support the weight of the sensor system would result in a broken servo for it wouldn't be able to exert enough torque to support the sensor system. The next factor will be application speed or velocity and how far the load will need to travel. As for speed or velocity we can favor torque for we won't need the servo to be moving too quickly otherwise we won't be able to gather accurate data from our sensor system.

3.4. Servo Choices

As for servo choices there are DC brushed servos along with AC brushless servos which although being more expensive have a longer life expectancy. On top of the difference in longevity, although AC brushless servos generally have more torque, they are harder to apply being that computation is done electronically and not mechanically. We will also need to consider how much voltage or current the servo will need to operate, for we will be powering it from our Arduino we will have a maximum voltage we will need to abide

by. As previously mentioned, before we make a choice of which servo best fits our needs will need to break out some physics and calculate the worst-case scenario for the peak torque. This generally occurs when the fastest acceleration is occurring with the full weight of our sensor system. Regular RC servos only have feedback on position to the internal controller which isn't ideal for we can't confirm its working as expected. This problem is solved with Analog servos for they provide feedback with an extra wire that can be connected to the analog input pin on the microcontroller. Whether or not we need this extra feedback to ensure correct positioning will determine what class of servo motor is needed for our project. Another crucial factor will be Jitter and is best described as twitches while trying to hold a steady position. This could be a very big issue for if the twitching is too much the sensor system won't be able to accurately focus. Depending on how significant the jitter is may mean we have to look into using a stepper motor.

3.4.1. Hitec HS-422 Servo Motor

This is a relatively cheap servo coming out around $13 that doesn't have too much to offer compared to some of the other pricier ones that will be compared. However, for the price it does pack quite a punch, the company Hitec claims it's one of the most durable and reliable servos they have. It has dual iron-oilite bushings, high impact resin gear train and high-performance circuity. The output torque at 4.8V is 45.82 oz/in, which would be enough to support our camera system. With an operating voltage of 4.8V to 6.0V the Arduino Uno shouldn't have too much of an issue powering it. However, the max output is only 5V so we may need to look into finding other means of powering the servo if they all have this voltage range.

3.4.2. Hitec HS-645MG Servo Motor

This is a bit more of an expensive choice coming out at $29 per servo, and considering we need two of these would be quite pricey. However, Hitec claims this is their top selling high torque metal gear servo. It claims to have a strong three-pole Ferrite motor and supports a heavy-duty metal gear train. It also has a dual ball bearing-Supported output shaft which should make for smooth transitions. The output torque at 4.8V is 111.09 oz/in, is significantly more than the Hitec HS-422 Servo Motor which would be enough to support our camera system. With an operating voltage of 4.8V to 6.0V the Arduino Uno shouldn't have too much of an issue powering it. However, the max output is only 5V so we may need to look into finding other means of powering the servo if they all have this voltage range. However, for the money this servo by far outperforms the HS-422 servo compared above.

3.4.3. SG90 Digital Micro Servo

This is defiantly going to be the cheapest servo we consider using for our project coming out at only $5 per servo. This is a very light servo weighing in at only 9 grams, although weight isn't a factor were considering that's still quite impressive. This servo also operates on the same voltage range from 4.8V to 6V, so it's becoming clearer that if we're going to be operating the servo at the higher end of its voltage range, we will need another way to power it. A rather concerning bit of data on this servo is that it doesn't list a maximum torque, instead it has what's called a stall torque. This means that it would be able to hold

an object at a set position with a rotational speed of zero with only 18oz-in. Although this may sound like enough, over time it may wear on the servo motors and render them ineffective or break them entirely.

3.4.4. HS-755HB Giant Scale Servo Motor

As its name implies this servo is giant, however comes out at a reasonable price of $25 per servo. Which is defiantly more than we would like to spend however may be a necessity. The HS-755HB servo has Karbonite gears which are four times stronger than standard nylon gears. This servo has a stall torque of 152.75oz-in which is much higher than any servo we have seen so far. The HS-755HB will also operate just fine at 4.8V which means we wouldn't have to implement any external power supply apart from power for the microcontroller which would be ideal. The only drawback of this servo is its size, it is for lack of better words giant. We haven't compared size yet so far because it wasn't too important of a factor, however coming out at 59x29x50 mm this servo is large. Being that the pan servo will have to support the weight of the tilt servo its weight of 110g is also a factor worth considering.

3.4.5. Servo Comparisons

Before we make a final choice on what servos we will use for our project we must calculate the max torque that will be applied to our sensor system. This is crucial because if our servo doesn't have enough torque to be able to hold up our sensor system all of this will be for nothing. The main factors in this decision were max torque, speed, weight, and cost. Table 3 lists the comparison's where the color green means good, yellow decent, and red poor.

	HS-422	HS-645MG	SG90 9g	HS-755HB Giant
Cost	$13	$29	$5	$25
Max Torque (oz/in)	45.8256	111.0946	25	152
Speed (sec/60o)	0.21	0.24	0.12	0.23
Weight (g)	45	55	9	110
Manufacture	Hitec	Hitec	Tower Pro	Hitec

Table 3 Servo Overall Comparison, with the selected SG90 servo highlighted in red.

3.4.6. Max Torque Calculations

In order to calculate the max torque that will be applied to our sensor system we must first take into consideration the weight of our sensor system. One factor that needs to be taken into account before we calculate the torque is what type of object, we will use to hold up our sensor system. Being that we will be using some specifically sized servo motors we will have to custom make something to hold onto the sensor system. If we go with some

type of metal mount for the sensor system this will add excess weight and thus, we will need more powerful servos which in turn would cost more. Speaking of cost, getting a metal mount built for our sensor system would be unrealistically costly. Luckily 3D printers are widely popular and quite simple to use. We will custom make some type of structure that will hold the servos which will allow us to pan and tilt our sensor system. The benefit of this is it will be light weight and cost efficient.

Being that we already had the SG90 servos on hand we went ahead and printed our sensor system mount and attached the servos to it. We ran some tests with the 50-gram camera and 2.2-gram laser attached to it and had no issues whatsoever when it came of either the pan or tilt functionality. That being said the only servo we really needed to worry about when it comes to torque is the tilt servo. Being that it will be responsible for lifting the camera vertically and holding it there. However, the saving grace is that the distance from the axis of rotation and the mass which is our camera and laser is very minimal being that the servo responsible for this tilting is attached directly to the unit. Our tests concluded that we shall have no issues supporting our camera and laser using the SG90 servo's that we already have on hand.

3.4.7. Servo Selection

Being that we are college students are sponsoring this project ourselves we can't quite afford too expensive of servos. However, we need to ensure that the servo we select will have enough power to get the job done. With that said we already had two of the SG90 servos and ran some strength tests and they seemed to do the job. The other servos that we compared would defiantly have more power and would be better fitted for upgrading the weapon or sensor system in the future. But being that we already had the SG90 servos on hand from previous projects it just made sense to use them.

3.5. Sensor system mount

We will need some sort of object that will allow our servos to give our sensor system the pan and tilt functionality we desire. This will require looking into different means of constructing a mount and looking at their pros and cons. One servo will have to rotate left and right on the x axis between -90° and +90°. This servo will have mounted to it the other servo which will support the tilt functionality for the sensor system. Depending on the type of material we choose to use for this structure will determine if we will need some type of support system to take some weight off the servo doing the pan functionality. When it comes to the tilt, this servo will be mounted sideways so the sensor system will be able to pan up and down.

3.5.1. CNC Machine

There is no doubt that if we were to have this mounting system built out of solid piece of metal it would look amazing. However, this would cost a fortune, on top of the cost to have this done it would also end up making our sensor system even more heavy which in turn would require more powerful and expensive servos. One benefit to using an CNC machine would that we wouldn't have to worry about making the structure strong enough since it would be made out of solid metal and our camera only weighs around 50g. The main

downside to this design would be the cost, we simply wouldn't be able to afford to have something like this built for us.

3.5.2. 3D Printer

Being that 3D printers are widely popular and quite simple to use this defiantly seems like the best choice we have available to us. One among many benefits to using a 3D printer would be how cost efficient it would be, we would be able to design a rudimentary mount and print it out and give it a test. This would allow us to ensure we have the best possible design which will put the least amount of stress on the servo motors as possible. Another key benefit would be how light the mount would be, the lighter the mount is the less the servos will have to work which means the less expensive of servos we need to use. 3D printing software is also much more simplistic and would require less time spent on learning a new software suite.

3.5.3. Sensor System Mount Comparisons

Given the many drawbacks of using a CNC machine to make our sensor system mount it's pretty clear we made the right choice of using a 3D printer. This will not only allow us to save a lot of money, but it will also give us ease of mind knowing that if our first design needs improvements we can easily learn from our mistakes and afford to just print another. It is crucial that we really think about the design of this mount for if done properly it will take a lot of the stress off the servos and make their job easier. However, the CNC machine would not only look a lot cooler, but the part it's self would last a lot longer. We must take into consideration how crucial the longevity of our device is to us for it's an important factor when choosing between a plastic or metal part.

3.5.4. Sensor System Mount Selection

Even though it's clear that building our sensor system mount on a 3D printer would not only be cheaper but also a lot easier we must still consider the CNC machine if were going for project longevity. Being that the 3D printed plastic part is made out of layers of plastic it could end up breaking down over time but given the light weight of the sensor system exposure to particular elements like the sun would really take a toll on the sensor system mount. Over a set amount of time outside there is no doubt that the 3D printed part would begin to deteriorate due to sun exposer. When compared to a piece of metal we wouldn't have any concern of the sun detreating the mount over time. There is no doubt that the sensor system or the servos themselves would fail before we had an issue with the metal detreating. Given that our project is more of a prototype for a much larger and more expensive final product it makes sense to just use a simple 3D printed part. Also, when considering how much more expensive it would be to get a part built using a CNC machine, we would have no room for mistakes and it's quite likely our first design would have some sort of issues. For this reason and many more we have defiantly chosen to use a 3D printed sensor system mount and will begin the design of it shortly.

3.6. Sensor system weapon

We need to attach some type of weapon to the sensor system as a proof of concept so that we can ensure that if our system was to be used to take down a drone it would be effective. There are many possibilities that we could choose from such as an air cannon or nerf gun, however implementing something like this would enhance the complexity of our project. We would need to be able to confirm that the weapon of choice actually targeted the drone successfully, which limits our choices of a weapon system. Another factor to take into consideration is number of shots we would have at targeting the drone. If we go with something like a nerf gun, we would have a limited amount of shots to take before we have to reload the weapon system again.

3.6.1. Nerf Gun

Implementing a nerf gun to fire at the drone wouldn't be too complicated, however it does raise a few issues. The most significant concern is size, many of the spring loaded nerf guns are quite large being that they need to house a spring large enough to shoot a little dart. This is a real issue with size is weight, being that our sensor system will support the weapon system weight is a real concern. Being that we chose to go with micro servos our sensor system needs to be as light as possible. Another issue with the nerf gun is that it only has a limited number of shots before it would need to be reloaded. Being that we may not be as accurate as we'd like it would be very inconvenient to have to keep reloading the nerf gun. The last issue regarding the nerf gun would be confirming that we hit our target. Finding and confirming that what a camera is looking at is a drone in the air is hard enough, confirming that this drone now has a nerf dart attached to it is a whole other issue. Supposedly you could say that the nerf dart may knock the drone out of the air and that could be confirmation that we hit the target but if it ends up not doing that or not even sticking to the drone we would have no way of knowing that we hit the target other than replaying the video or by human visual confirmation.

3.6.2. Air cannon

Implementing an air cannon would be a much lighter solution compared to the nerf gun being that it only be a little pipe with some compressed air. However, it does suffer from some of the same issues as the nerf gun. Reloading the air cannon would be an issue being that we would either have to use a manual air pump like a bicycle pump or something like an air compressor. An air compressor would enable us to have as many shots as we'd like but air compressors are not only large and heavy, they are also expensive. We would also have to learn about Numatics to know how much air to shoot. The main issue that this poses is it requires us to buy more stuff and that would be out of our budget. The next issue is confirmation that we hit the target, being that we are only shooting air it would be very complicated to see that we hit the drone. The only way would to see if we hit the drone off course but even then, how would we know if the drone wasn't just flying in that direction.

3.6.3. Laser

Implementing a laser would be the easiest solution for many reasons. First it would enable us to fire at the drone as many times as we'd like without having to reload or worry about

running out of ammo. Another benefit of using a laser is that it's a very cheap and light solution weighing in at only 2.2 grams we have no issue mounting it to our sensor system. The final reason using a laser is that confirming that we hit the target would be a lot straighter forward. We would simply have to look for a red dot on the drone to confirm that we hit the target.

3.6.4. Weapon system comparison

Taking into consideration the different options that we have for mounting a weapon on our sensor system it's clear that we should go with the laser. We have chosen to use the KY-008 laser for its very cheap and light weight. The laser operates at a voltage of 5V which can easily be supplied from our microcontroller so the set up will be very easy. Another benefit of using the laser is that we will be able to track our drone while shining the laser on it to indicate that we have successfully targeted it. All in all, the when compared to our other options the laser is not only the easiest to implement but also the best when it comes to a proof of concept being that it will give us an unlimited amount of shots and ability to prove our tracking functionality is effective.

3.7. Wireless VS wired communication

In order to switch the Arduino Uno between sweeping and tracking mode we will need some type of commutation between the two boards. There will be many factors to consider such as speed, reliability, cost, and simplicity. On top of switching between sweeping and tracking we will also need to be able to send other commands to the Arduino such as where to direct the sensor system. The only question that remains will be whether we use wireless communication such as Bluetooth, or Wi-Fi, or use a wire to connect the two. Wired and wireless communications will both have their benefits and downsides we will just have to consider what will most accurately meet our requirements.

3.7.1. Wireless communication

Obviously, the main benefit of using wireless communication would be that we wouldn't have any wires between the main development board and the sensor system. This would be a real game changer for we would be able to possibly place the much cheaper sensor system outside and the more expensive development board in a safe location. However, the reason this most likely won't work for us is that we need to be able to transfer data from the sensor system to the devolvement board as possible. Although Wi-Fi would most likely be fast enough for this it would make this project significantly more challenging. Being that we don't have a requirement to have the communication line between the sensor system wireless so implementing this defiantly isn't on the top of our priority.

3.7.1.1. Bluetooth

Bluetooth is inherently not as fast as Wi-Fi and although we're not sending much data from the devolvement board devolvement board to the sensor system, we will be sending a constant video feed from the sensor system to the devolvement board. Being that the newest protocol for Bluetooth 4.0 only transfers at 25Mbps this would most likely not be quick enough. On top of 25Mbps possibly not being fast enough this speed is achieved under

ideal conditions. This means that due to real world conditions derogating transfer speeds we would most likely not see this. On top of that, distance between the two boards will also implicate the transfer speed so we may have to put the boards so close to each other it may as well be wired.

3.7.1.2. Wi-Fi

With speeds being plenty fast enough to transfer some video feed between the devolvement board and the Arduino Uno, the only real problem is implementation. Some Arduino's have an ESP8266 Wi-Fi chip already built in, however being that were going to be building the Arduino on a PCB it would require us to build one in. The real problem is that the ZCU104 devolvement board doesn't support Wi-Fi. Buying or building a chip that would work with this devolvement board wouldn't be too complicated being that there is an Ethernet port on the board already but it's an extra step we don't necessarily have to take.

3.7.2. Wired Communication

We will need two types of communication that will be used for two different pieces of hardware on our project. The first and most straight forward would be the connection between the 1080p Camera and the devolvement board. The camera is already set up to use USB 3, so changing that to Wi-Fi would be unnecessarily complicated. As for the connection from the devolvement board to the Arduino since we're using a wired connection already for the Camera we may as well do the same. This means we would need to have a USB port on our PCB that would connect to the Arduino's built in Serial port. A wired communication is also ideal for this connection because both the devolvement board and the Arduino support UART. This means the set up would be far less cumbersome and would only require some basic research. The devolvement board supports a FT423H which is a highspeed USB 2.0 to UART IC, at a speed of 480Mb/s. This will be plenty fast enough and if anything will be bottlenecked at the Arduino UART port.

3.7.3. Wired Vs Wireless Comparison

For simplicities sake we're going to go with a wired connection between the Arduino and devolvement board. Along making implementation much more straightforward we will also avoid many issues that could come into play with wireless communication such as interference or signal strength. The only downside here is that the devolvement board will have to be physically connected to the sensor system. However, being that our project is mainly a prototype and should only be used indoors we wouldn't really benefit from a wireless connection anyways. We will need to implement a UART connection between the devolvement board and the sensor system, to control its direction. To transfer a live feed from the sensor system to the devolvement board we will use the already built in USB 2.0 connection.

3.8. Serial Communication

Choosing between a wired or wireless communication is only half of the issue, the next part is what protocol we are going to use. When considering what type of serial communication, we could use we must take into consideration what the microcontroller

and devolvement board supports. On top of that given the type of data we will be transferring between the microcontroller and devolvement board will also help determine what type of serial communication we should use. This section will break down each of the different types of serial communication the microcontroller supports and go into detail about how they work and why choosing one over the other could have a great impact on the successfulness of our project.

Communication Standard	**UART**	**SPI**	**I2C**
Speed	Up to 115.2Kbps	10Mbps+	Up to 3.4Mbps
Complexity	Simple	Difficult	Moderate
Scalability	Only 1 device per UART port	Theoretically unlimited	Limited by address size
Required Wires	2	4	2

Table 4 Serial Communication Comparison, with the selected UART communication standard highlighted in red

3.8.1. UART

The UART acronym stands for Universal Asynchronous Receiver Transmitter and being that it's built for two wire communication between two devices it's one the of simplest ways to implement a serial connection. There are two ports on the microcontroller to support UART, one called TX for transmit and the other RX which is for revive. Being that UART is asynchronous we don't have to worry about getting the clock rates synchronized between the sender which would be the development board and the receiver which would be the microcontroller. Thus, UART uses a different method to determine how the data on the receiving end is processed. By the use of a start and stop bit which are agreed on prior to sending data the sender and receiver can know when to start and stop sending and receiving the data. The next crucial factor for UART is what's known as the baud rate. The baud rate is most simply defined as how many bits per second can be sent or received using the UART connection. Measured in bits per second the standard baud rate is at 9600 but can also be as high as 115200. An important factor to consider here when picking baud rates is the amount of noise the UART connection will be able to overcome. With a higher baud rate, you may be transferring data faster, but the connection will be more prone to noise, thus possibly less reliable. Speaking of reliability while there is clearly potential for error, UART uses parity bits which helps the receiver correct up to a specific number of bits depending on how many parity bits there are. One of the main reasons to use UART over any other type of serial communication is the fact that it's relatively easy to set up compared to other protocols especially when the two devices communicating have different clock rates which is the case between the microcontroller and development board. The only real downside of UART is the number of devices which will be able to communicate. Being that our project only requires communication between the between the microcontroller and development board this factor shouldn't be an issue. The sensor system will use its own USB cable to send data to the development board. With all of this considered implementing

a UART connection between the microcontroller and development board seems like a good fit so far.

3.8.2. SPI

The next serial communication protocol to consider for our project is a synchronous type called Serial Peripheral Interface. One of the main benefits for using this type of serial communication is the ability to communicate between more than just two devices at a time, which is all we could do with the UART serial connection. However, being that SPI is synchronous, the lock rates between the microcontroller and development board must match up when reading the data. Luckily this isn't support hard to do because of the way SPI connections are set up using 4 wires between the devices. Two of the four communication lines are used for sending and receiving data while the third is used for device selection and the last is for saying when to read each bit. The two data lines are dependent upon which one is the master, and which is the slave. In this context the master is most generally the microcontroller while the slave would be any devices connected to it.[14]

SPI allows for fully duplex communication which means that data can be sent and received at the same time. Also given the way it's set up; it can interface as many devices as needed. With all of this said some of the main benefits for using SPI don't quite apply to our project. Figure 5 below shows a low-level diagram of how SPI is generally implemented and how the master and slave devices communicate

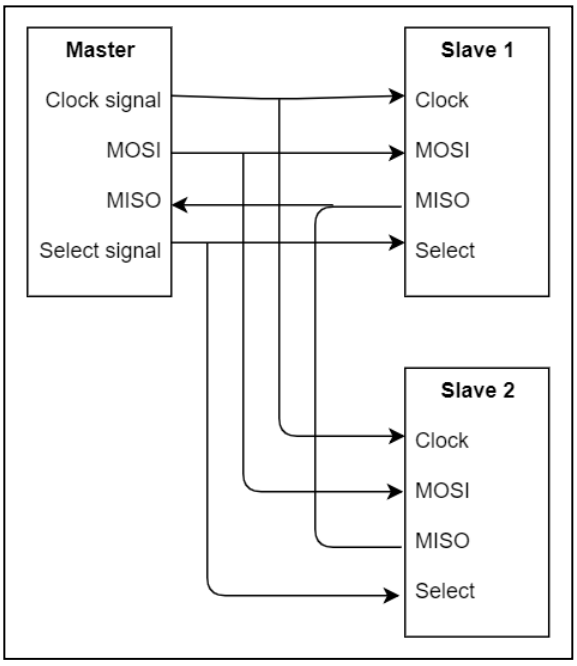


Figure 5 SPI Communication diagram

3.8.3. I2C

The last type of serial communication to consider implementing for our project is known as Inter-Integrated Circuits but is more commonly just known as I2C. This protocol is quite similar to both UART and SPI, except doesn't suffer from some of the downsides of them.

Similar to SPI, I2C also uses a clock signal to synchronize the transfer of data been the master and slave devices. What's fascinatingly different between I2C and SPI is how it selects which device it's communicating with. By sending a start condition somewhat similar to the start bit in UART the receiver can know when to start processing its data. On top of that, after the start condition but before any data is transferred the address of where the data is supposed to go is sent. This ensures the correct device gets the data without the use of an extra wire just for device selection. I2C also has some other tricks to ensure that two devices aren't going to be trying to communicate at the same time, which would render any data transferred processable. Only limited by a number of addresses, the I2C protocol can be used to communicate between way more devices then our project needs to.[10]

The main take away here is the I2C would be overkill for our project being that it's mainly used when multiple devices need to communicate with each other. Figure 6 below shows a general implementation for I2C.[9]

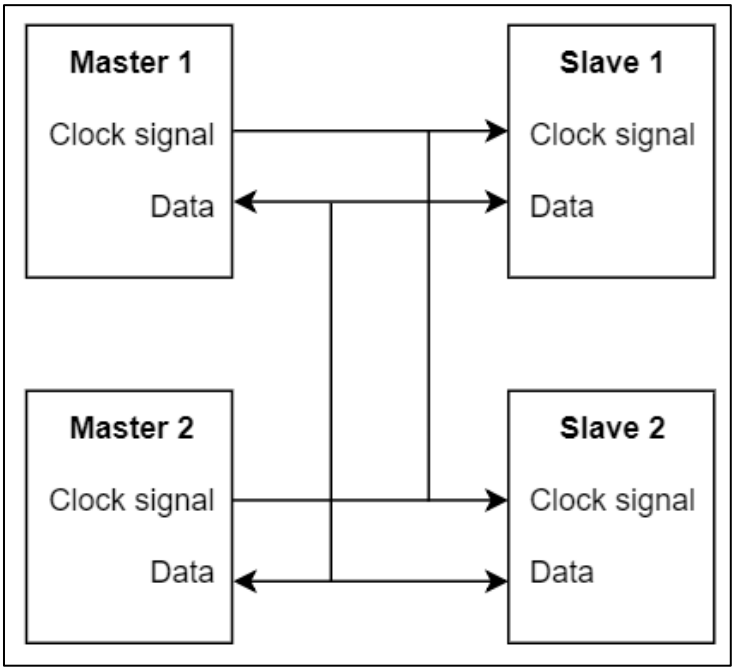


Figure 6 I2C Communication

3.8.4. Serial Communication Comparison

When considering the microcontroller, we have a few different serial communications that could be implemented. However as previously stated each one has its benefits and downsides. We must take into consideration not only the microcontroller but also the development board when making our choice.

Looking at the advantages that a communication protocol like SPI has to offer it's quite clear that we wouldn't make use of its scalability. SPI could be used to communicate between a number of different devices and being that we only need to send data from the development board to the microcontroller this would be unnecessary. Although the 10Mbps transfer rate is inviting, this protocol significantly more complex when compared

to something like UART and we wouldn't benefit from its advantages. With this said although this protocol has many uses in the real world, we may have to rule it out as a choice for our project.

The next protocol to consider is I2C which is quite similar to SPI when it comes to having a lot more functionality then UART. Being that it's widely popular, community support would be significantly easier to find and available compared to SPI. Another benefit to SPI is its ease of implementation to connect multiple devises. Being that only three wires is used instead of four like SPI uses it's a bit easier to implement hardware wise. The main issue with using I2C is exactly the same as SPI, we would be implementing a more complicated protocol and not benefiting from it.

The last and arguably best communication protocol for our project to consider is UART. The sheer simplicity of this type of connection is one of its main benefits. Although its max data rate isn't that fast it doesn't really matter because we won't be sending much data at all between the microcontroller and development board. The fact that it only requires two wires to be connected between the two devices makes implementation much easier when it comes to hardware. On top of all of this, both the microcontroller and development board both openly support UART so we wouldn't have to implement some hacky solution to get something like SPI to work between the two. All and all UART is possibly the only protocol to really consider for our project given its simplicity. Table 4 shows the breakdown of each serial protocol. In the table the colors indicate the effectiveness where the color green means good, yellow decent, and red poor.

3.8.5. Serial Communication Selection

When considering all the information above about the various types of serial communication above it's quite clear that choosing UART would make the most sense. Being that our project only requires communication between two devices, implementing anything else other than UART wouldn't be necessary. After having ensured that both the development board and microcontroller both support UART, implementation should be relatively straightforward even though the two boards will have different clock rates and such. Also, when considering the ease of implementing this type of communication protocol the choice doesn't require much thought at all. Considering the microcontroller, we choose which is the Arduino Uno has great community support, implementing the UART should be relatively straight forward on it. However, although the development board supports UART it will be more complicated to implement due to the lack of documentation and community support being that it's such an expensive product.

The only issue with a UART connection is knowing when to check for new data on the microcontroller. The easy way to do this would be to continuously check if any new data has been received. There are many problems with this method, the first being that were wasting processing power to keep checking if a new command has been received. On top of that the main issue is that once a command is received, we will need to be able to break out of executing the command if a new one is sent. For instance, when we enter tracking mode we will need to be reading and executing commands from the development board in as close to real time as possible. With this in mind we would have no time to keep checking if a new command has been received.

The solution to this problem which we have chosen to use is implementing hardware interrupts. By having the development board send an interrupt to the microcontroller whenever a UART command is going to be sent we can ensure a much more reliable and efficient functionality. Not only will this enable us to implement effective tracking by having the microcontroller track while also being able to accept new commands, we will also not have to keep checking if a new UART command has been sent. Since the only time we will have to check for a UART command is when we receive an interrupt, we will have a more seamless switch between scanning and tracking mode as well. Luckily the development board and also the microcontroller both support interrupts and have dedicated pins implementing this should be rather straight forward.

The only real decision left to make is what baud rate to use. Even though the development board supports a higher baud rate then the Arduino Uno, we won't be able to use it because of the limitations on the microcontroller. With that said a higher baud rate means less reliability. Since our entire project relies on the development board being able to reliability communicate and control the sensor system, we need reliability. With all of this said choosing the default of 9600 bits/second seems appropriate. This should allow for a fast-enough data transfer rate between the development board and microcontroller while also being reliable enough.

3.9. Machine Learning Development Boards

To meet the specification requirements of accurate targeting and timing, the board must be able to quickly parse data from a sensor and determine if a target has entered field of view. The focus of selection for development boards is the ability to process images in real time, and the ability to support a pre-trained neural network for accurate classification of targets in the picture. Additionally, the development boards need to be able to have communication ports to connect to the created PCB to communicate effectively to extraneous hardware that will be needed for real time tracking and targeting.

3.9.1. Development Board Options

Development boards for computer vision fall into two categories, Graphics Processing Unit (GPU)/Embedded based development boards, and Field Programmable Gate Array based boards. The Graphics Processing Unit boards are mainly fielded by the NVIDIA Jetson line, and only Xilinx was viewed for FPGA development boards, since no member on the team is familiar with Altera chips.

3.9.2. FPGA Development Boards

Xilinx produces a wide variety of Field Programmable Gate Arrays, however, the research focused exclusively on Multiprocessor System on Chip solutions, due to the sole support of the Xilinx Revision stack, and Deep Neural Network Development Kit (DNNDK) solution provided by Xilinx. The Zynq Ultrascale chips are the most advanced of the SOC chips. Of these, three boards are evaluated: the ZCU102, ZCU104, and Ultra96.

3.9.2.1. ZCU102

The ZCU102 is a Xilinx made development board, targeted for multiple applications including video processing and machine learning. It is one of the target boards for the proprietary Deep Neural Network Development Kit (DNNDK). The board features the ZU9EG model of the Xilinx Zynq Ultrascale line of Multiprocessor System on Chip, which houses a quad core Cortex ARM processor, a dual core Cortex-R5F real-time processor, and a Mali-400 MP2 graphics processing unit. It has on board DDR4 SODIMM connectors for the Programmable Logic, and Programmable System sides of the Multiprocessor System on Chip. The board has connectivity for multiple communications, including Peripheral Component Interface Express (PCIe), Ethernet, USB3, Display Port, and SATA. The board has extensive Input/Output (I/O) expansions, including FPGA Mezzanine Card (FMC) interfaces, and 64 user-defined I/O signals.[27]

3.9.2.2. ZCU104

The ZCU104 is a Xilinx made board targeted specifically for embedded vision design. The board is supported by the Xilinx proprietary Deep Neural Network Development Kit (DNNDK), and by the Xilinx Revision Stack, which has support for a Xilinx developed OpenCV. The board is built around the 7U7EV model of the Xilinx Zynq Ultrascale line of Multiprocessor System on Chip, which also houses a quad core ARM Cortex processor, dual core Cortex-R5 real-time processor, and a Mali-400 MP2 graphics processing unit. The board also houses a 4KP60 H.264/H.265 video codec. The board has 2GB of DDR4 RAM connected to the Programmable System (ARM Cores), and a SODIMM connector for any choice of RAM for the Programmable Logic (FPGA fabric) side of the ZU7EV chip. The board's peripherals include common video outputs including HDMI, Display Port, Ethernet, PMOD connectors, and one Low Pin Count FPGA Mezzanine Card for I/O expansion.[28]

3.9.2.3. Ultra96

The Ultra96 is a development board for the Xilinx Zynq Ultrascale ZU3EG Multiprocessor System on chip. The board is not developed by Xilinx, but by a third-party manufacturer. The board has only 2GB of Low Power DDR4 Memory, and a mini DisplayPort. The board has a 40-pin header for low speed I/O, and 60 pins for high speed I/O. The board also has support for USB connections, and Wifi. The board is supported by Xilinx's Deep Neural Network Development Kit, and Revision stack.[26]

3.9.2.4. FPGA Comparisons

The chosen FPGA will perform a critical role in the artificial intelligence of the RTDS turret. Ensuring that it meets proper criteria given our constraints will help the RTDS identify and differentiate drones in real-time.

3.9.2.4.1. Comparison of MPSOC Chips

are taken from the Xilinx provided selection guide of their Zynq Ultrascale Products. As seen from them, the Processing System (PS) portion of each of the chips is the same. The main difference between the different Multiprocessor Systems on Chip is the difference

between the sizes of the Programmable Logic (PL), and access to hard IP blocks on the die.

The EG line does not have a video codec for quick processing of video, alleviating spacing constraints needed to implement the Deep Learning Units (DPUs) instrumental in providing real time computer vision from camera inputs.

Note, the above table is only comparing the items located only on the die of the Zynq MPSOC chip, not of what is connected on the board. For example, the ZU7 chip has a hard core dedicated for PCIe connection, which is needed to run Xilinx's IP core for PCIe, however, there is no PCIe port for the ZCU104 board that houses the chip. The same argument holds for PCIe regarding the Ultra96.

Comparing the Programmable Logic sizes, it is seen that the ZU9EG has the highest amount of logic elements, and RAM available on the chip, with the highest communication speeds available. The ZU7EV sits relatively close behind, with 500K logic blocks, and comparable RAM size. It is of note, that the type of RAM between the EV and EG line chips is not the same. Xilinx has introduced an "UltraRAM", which serves as a larger capacity, more flexible memory block (REF ULTRARAM). The UltraRAM, coupled with the on-die video codec unit found on the ZU7EV, outweighs the increased fabric space of the ZU9EG.

Note, the amount of logic elements the FPGA has is a critical constraint for the project, since the proprietary Deep Neural Network Development Kit (DNNDK) provided by Xilinx is implemented on the Programmable Logic (PL) side of the chip, on either the logic elements or on the DSP blocks available (DPU Xilinx Guide).

	Device Name		ZU3EG	ZU9EG	ZU7EV
Processing System (PS)	Applications Processing Unit	Processor Core (APU)	Quad Core ARM Cortex A53 up to 1.5GHz		
		Memory w/ECC	L1 Cache 32 KB/Core, L2 Cache 1MB, on chip Memory 256KB		
	Real-Time Processing Unit	Processor Core (Real Time)	Dual-Core ARM Cortex-R5 MPCore up to 600MHz		
		Memory w/ECC	L1 Cache 32KB, Memory 128KB per core		
	Graphics and Video Acceleration	Graphics Processing Unit	Mali-400 MP2 up to 667MHz		
		Memory	L2 Cache 64KB		
	External Memory	Dynamic Memory Interface	x32/x64: DDR4		
		Static Memory Interface	NAND, 2x QUAD-SPI		
	Connectivity	High Speed Connectivity	PCIe Gen2 x4, 2x USB3.o, SATA 3.1, DisplayPort, 4x Tri-mode Gigabit Ethernet		
		General Connectivity	2xUSB2.0, 2x SD/SDIO, 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO		
	Integrated Block Functionality	Power Management	Full / Low / PL / Battery Power Domains		
		Security	RSA, AES, and SHA		
		AMS – System Monitor	10-bit, 1 MSPS – Temperature and Voltage Monitor		
PS to PL Interface			12 x 32/64/128b AXI Ports		

Table 5: Zynq MPSOC Model Comparison

	Device Name		ZU3EG	ZU9EG	ZU7EV
Programmable Logic (PL)	Programmable Functionality	System Logic Cells (K)	154	600	504
		CLB Flip Flops(K)	141	548	461
		CLB LUTs (K)	71	274	230
	Memory	Max Distributed RAM (Mb)	1.8	8.8	6.2
		Total Block RAM (Mb)	7.6	32.1	11
		UltraRAM (Mb)	-	-	27
	Clocking	Clock Management Tiles	3	4	8
	Integrated IP	DSP Slices	360	2,520	1,728
		Video Codec Unit (VCU)	-	-	1
		PCI Express Gen 3x16	-	-	2
		150G Interlaken	-	-	-
		100G Ethernet	-	-	-
		AMS – System Monitor	1	1	1
	Transceivers	GTH 16.3GB/s Transceivers	-	24	24
		GTH 32.75GB/s Transceivers	-	-	

Table 6: Zynq MPSOC Model Comparison (continued)

3.9.2.4.2. Comparison of Peripheral On-Board Components

As seen in Table 7, the ZCU102 has more peripheral components focused on connectivity than the other two boards. The project requires high processing power for camera inputs, however, does not expressly need high throughput communication protocols with multiple FPGA Mezzanine Connectors and PCIe ports. The excessive communication abilities of the ZCU102 are superfluous for the project.

Peripheral on-Board Component	Ultra96	ZCU102	ZCU104
External Memory (RAM)	2GB LPDDR4	Yes (SODIMM for PS)	Yes (SODIMM for PL, 2G for PS)
Storage (SD card)	Yes	Yes	Yes
USB	Yes (3.0 & 2.0)	Yes (3.0 & 2.0)	Yes (3.0 & 2.0)
HDMI	No	Yes	Yes
Wifi Module	Yes	No	No
Display Port	Yes	Yes	Yes
PCIe	No	Yes	No
SATA	No	Yes	Yes
FMC port	No	Yes (2 FMCs)	Yes (Low Pin Count)
PMOD Port	No	Yes	Yes
Headers for Easy I/O pinouts	Yes	No	No

Table 7: FPGA Dev-Board Components

The ZCU104 has all the needed connectivity for the project, with multiple GPIO accessible ports through PMOD connectors and Low Pin Count FPGA Mezzanine Connectors. The USB3.0 Port can be used for the USB camera, to provide 60FPS input. The SODIMM connector allows the user to place any size DDR4 memory for connecting to the Programmable Logic (PL) side, however, has a fixed 2GB DDR4 RAM module for the Processing System (PS).

The Ultra96 only has a display port module for displaying information to the screen, and the target output for screen display is currently HDMI. Additionally, the Wifi module is extraneous to the project. The headers are nice for quick prototyping to servos; however, the project will already be fabricating an I/O board to serve the purpose of quick connectivity to the main development board, making the perk less impactful.

3.9.2.4.3. Comparison of Cost

Development Board	Price (USD Dollars)
ZCU104	895
ZCU102	2,495
Ultra96	249

Table 8: Cost of Researched Zynq Development Boards

As seen in the above table, the ZCU102 is more than double the price of the ZCU104. The ZCU104 is approximately three times the cost of the Ultra96 board. For pricing constraints, the Ultra96 is the cheapest option.

3.9.2.5. Chosen FPGA Based Development Board

The major constraints for the development boards in respect to the project are the capabilities of the on board Zynq MPSOC Processor, and its ability to implement the DNNDK workflow needed to create the machine learning based tracking algorithm. Due to a desire for quick movement, the focus for selection of a development board also depends on the Zynq MPSOC's speed in processing the video. The ZCU104 is the only of the investigated boards to have a video codec unit. Cost is also a concern for purchasing a development board. The ZCU102 has more logic elements than the ZCU104, however, is more than twice the cost for only a 20% increase in logic elements. The Ultra96, is the cheapest option, however, has one fifth of the logic elements available to the ZCU104, and one sixth of the ZCU102. Additionally, the ZU3EG chip does not have a variety of features available with the chips utilized by the other boards. Due to the above points, the ZCU104 is chosen to be the most cost-effective FPGA solution for the project.

3.9.3. GPU/Embedded Development Boards

A Graphics Processing Unit will perform much in the same way as a Field Programmable Gate Array will, but with the specialized hardware that it is equipped with it is more designed strictly for artificial intelligence and neural network processing, as opposed to the versatile logical capabilities of a powerful FPGA. Finding the GPU with the most efficient qualities that fit within our constraints will allow us a fair comparison against the candidate FPGA already selected.

3.9.3.1. NVIDIA Jetson Series

The NVIDIA Jetson is a solution comprising a System-n-Module development board with the all necessary components of the system (CPU, GPU, RAM) already on the development board. The appeal of the Jetson is quick development times, building around a preexisting solution. The Jetson line comes with three notable development boards, the Jetson Xavier, Jetson TX2, and Jetson Nano.

The Jetson Xavier has a 512-core Volta GPU with Tensor Cores, an 8-core ARM CPU, 16GB of memory, and 32GB of eMMC storage. The Xavier is capable of 30 Trillion

Operations Per Second (TOPS) for deep learning tasks. The Xavier has a power consumption of 10W/15W/30W depending on selected power mode.

The Jetson TX2 is a 256-core NVIDIA GPU. It has 8GB of memory, and 32GB of eMMC storage, with an operation power consumption of 7.5/15W depending on the power mode selected. The device has a performance of 1.3TFLOPS (Trillion floating-point calculations every second).

3.9.3.2. UP Development Kits

The UP-development kits all rely on Intel solutions for their development boards. It was difficult to navigate the UP website to find benchmarks for their solutions for different neural networks, and thus was difficult to compare. The boards offer an option to offload CPU and GPU processing to an Intel developed "AI Core", which houses the Movidius Myriad X Vision Processing Unit.

3.9.3.3. VIA Edge AI developer Kits

The Via Edge Developer AI uses a Qualcomm Embedded processor. The claimed usage cases for the developer board is facial recognition, object detection, and other intelligent robotic solutions. The VIA Edge board also has a Wifi module, a Bluetooth module, and GPS. It supports Linux BSPs.

3.9.3.4. Comparison

The lack of performance statistics for the VIA Edge AI board and the UP-development kit made NVIDIA's boards the only verifiable option. NVIDIA is also the leader in GPU based acceleration of artificial intelligence, and it is expected that their development boards would be highest in quality. Exhaustive comparison of chip components was not done; however, a perfunctory inspection of price and on-board components shows that NVIDIA solutions are superior to the other investigated edge AI solutions.

Board/Software Support	NVIDIA TX2	VIA Developer Kit	UP Development Kit
HDMI port	Yes	No	Yes
YOLO Support	Yes	Unknown	Unknown
TFLOPs/TOPS	1.3 TFLOPs	Unknown	Unknown

Table 9: Comparison of GPU/Microprocessor Boards

As seen in the above table, the NVIDIA TX2 is compared to the two other boards with desired characteristics. Not much supporting information was found regarding what neural networks the other boards support, nor their optimal AI performance. Due to the lack of documentation, and the known support that NVIDIA offers, the NVIDIA line was the optimal selection of the GPU chips.

3.9.4. GPU vs. MPSOC

Having selected both a candidate GPU and FPGA, it is now important to make an educated decision between which of the two board types will be most efficient for the purposes of our design, as well as remain within the constraints of the design.

3.9.4.1. AI Performance

The DPU cores available to the ZCU104 allow it to run at a peak performance of 2.24TOPS. The speed of the cores, however, are not the major selling point of the DNNDK package available by Xilinx, it is the DeePhi pruning of the input network for higher performance of the neural network. The pruning allows for the neural network to be smaller, with higher performance. The ZCU104 has lower wattage consumption, with higher FPS for the same neural network on the Jetson TX2.

3.9.4.2. Cost

The ZCU104 costs approximately 30% more than the Jetson AGX. Comparing the available on-board resources and value is difficult, since the essential hardware components driving the ability to process high amounts of data differ, the ZCU104 uses the Zynq Ultrascale chip, and the Jetson uses NVIDIA's GPU architecture.

Board	Cost
Jetson Xavier	699.99
ZCU104	895

Table 10, Development Board Cost Comparison

3.9.4.3. Development Boards Support/Software

The supporting software will be the primary means of interfacing with the developer board. Choosing a board with a powerful and robust integrated software will help avoid unnecessary down-time while attempting to learn its use and debugging unexpected errors.

3.9.4.3.1. Xilinx DNNDK and ReVISION Stack Solution

The ReVISION stack is a solution for Xilinx supported boards to use in conjunction with the xfOpenCV library created by Xilinx for hardware acceleration of commonly used OpenCV functions. The ability to use common functions between different work environments allows for developers to create proof of concept structures using native OpenCV architectures (e.g. developing with Python), and then recreate the algorithm in the SDSOC environment. The ReVision stack supports the ZCU104 board.

Additionally, the ReVision stack is built in conjunction with the DNNDK solution. The currently supported convolutional neural networks include Resnet50, Googlenet, VGG16, SSD, Yolo v2-v3, Tiny Yolo v2-v3, and Mobilenet v2.

3.9.4.3.2. NVIDIA Jetpack

The Jetson development boards are pre-installed with a multitude of development tools. The tools include the Nsight Eclipse Edition, CUDA-GDB, CUDA-MEMCHECK, Nsight

Systems, nvprof, and Nsight graphics. The Nsight Eclipse Edition is a for development of GPU accelerated applications and operates on the GPU modules that are on board. The CUDA-GDB and CUDA-MEMCHECK are for debugging applications, and memory, respectively. The Nsight Systems profiles for the GPU and CPU on the system.

3.9.4.4. Chosen Development Board

Due to the lower cost, greater power, and access to state-of-the-art integrated YOLO support, the Xavier was the chosen board for the project. The software environments of both the NVIDIA and Xilinx are comparable in support, and both are targeted for optimization of neural networks.

3.10. Deep Learning Networks related to Object Detection and Classification

The chosen machine learning network will need to be optimized for object detection, specifically in respect to size of the network, and latency. The surveyed neural networks include GoogleNet, ResNet, and YOLO. Many other architectures exist, however, due to these networks support on the ZCU104, the available literature for the specified networks, and team familiarity, only the named architectures are specified.

3.10.1. GoogleNet

The GoogleNet architecture is a highly optimized network when compared to AlexNet and VGGNet due to the removal of fully connected layers and instead using global average pooling. The network is used for object detection and classification.

The GoogleNet architecture was one of the first Convolutional Neural Networks that utilized network-in-network modules. These allowed for greater generalization of the feature sizes needed to be trained by the network, allowing for greater productivity since the person training the network did not need to specify the exact filter size for each convolutional layer. This module is known as an “inception model” and was considered a breakthrough in machine learning.

Due to the limited size of an embedded system, smaller sized weight networks are desired, making GoogleNet an attractive choice for training drone tracking. The GoogleNet network is supported by the DNNDK solution.

3.10.2. ResNet

The ResNet architecture also utilizes the network-in-network type of structure that GoogleNet uses. ResNet uses a residual module, which trains convolutional neural network to extreme depths, making it a novel architecture. ResNet can also be used for object detection.

The ordering of the ResNet is of note, since it has skipping connections, which allow the network to become deeper, while still maintaining lower complexity in respect to other convolutional neural networks.

3.10.3. Yolo V3

YOLO is a novel approach to object detection due to its speed in finding multiple objects in an image. The algorithm achieves this by splitting the image into multiple boxes, and then analyzing each for a specific object. Due to the speed of the network, and its high rates of classification accuracy, this network was chosen by the group.

3.11. Computer Vision Packages

The task performed by the Reactionary Targeting Defense System requires the consideration of a software package chosen to handle the computer vision and machine learning algorithms. A number of previous considerations can be examined before looking deeply into any of the available packages.

First, this is an examination of the considerations from an exclusively software perspective. Other constraints may exist with regards to the hardware chosen to be used – some FPGAs or other processor devices may be bested suited for certain computer vision packages, or perhaps even designed with a particular package or algorithm in mind. Ignoring these constraints momentarily allows for an unbiased investigation of the available market of computer vision packages.

Additionally, we can rule out any significantly low-level packages that would require us to effectively build our computer vision algorithms from the ground up. The purpose of this design is to act as a proof of concept of the integration of well-known computer vision principles with that of well-known robotics and autonomous movement. This eliminates the need for any considerations of certain software packages that could be used to implement computer vision software, but not necessarily having the capability natively. Some examples of this include the NumPy and SciPy packages, which have built in analyses and computational capabilities, but do not perform machine learning by themselves.

3.11.1. OpenCV

One of the most popular and well-known computer vision libraries is OpenCV, an open source computer vision library used in most all facets of the market: Individual use, such as this RTDS senior design project; Corporate use by companies such as Google, Honda, and IBM; Public use like that of public surveillance and face recognition.[1]

The most obvious benefits of OpenCV are its cost and portability. Being open source, OpenCV is free to use, and as a BSD-licensed product this extends even to private, for-profit corporations – certainly we will have no legal troubles using it for our private educational purposes. OpenCV also runs on any machine that can compile and run C code, making it incredibly portable across popular operating systems, while having interfaces for C++, Python, Java, and MATLAB.[15]

Most notably, OpenCV also maintains a Video Analysis module, which is relevant to our project's goal of being able to track a flying drone in real-time across the camera's vision.

3.11.2. SimpleCV

SimpleCV is a self-described "framework" built for simplifying the application of computer vision and lowering the skillset needed in order to begin applying it and learning with it. *"This is computer vision made easy"* is proclaimed in bold text on their website's front page. Lower technical requirements may make using it easier for the purposes of our project, however it may lack some of the higher-powered object classification demands of watching for a flying drone in real time through a live camera feed.[21, 23]

Evidence from the main SimpleCV website would imply that it is not currently supported. The last blog post made to its website was from February 2014. The linked "Help Forum" from the core website is also deprecated, only displaying a "Service Unavailable" message when attempting to access it. Finally, the nail in the coffin might be that the last commit pushed to its GitHub repository was from April 7th, 2015. [22]

Despite the fact that the core library has received no updates in nearly the past five years, this is not necessarily grounds for immediate disqualification from our considerations. The library is still functional with its contemporary releases of scripting languages, and no modern image file format is significantly different from what it was back in 2015. Additionally, there are 49 active pull requests – one of which is as recent as August 2019 – and 712 forks of the GitHub repository. Should there be a significant update or feature necessary or desired to use the library, it is certainly reasonable to believe that one of the pull requests could bear the feature or capability we require.

3.11.3. scikit-image

Scikit-image is a self-described "collection of algorithms" native to python designed for computer vision, image processing, and object detection. It is a free-use, open source library that is an extension of the SciKit toolkit for SciPy. It makes heavy use of python's NumPy library, by representing all images being examined by it as custom NumPy datatypes "ndarrays." Being written in Python, a scripting language as opposed to a compiled language like Java, comes with its own advantages, most notably with regards to its runtime speed. [24]

The documentation for scikit-image makes a particular point to enumerate the difficulties of performing object detection on video file formats and is not native to the library. The most basic workaround suggested by the documentation is to segment any input video into individual images, performing the desired task on each frame of the video individually, and then regathering the images as a post-process. For the purposes of the RTSD processing power, this may be a feasible option assuming that this is the least expensive of the potential solutions; Otherwise, other video-reading libraries are suggested, including PyAV, MoviePy, Imageio, and OpenCV.

3.11.4. You Only Look Once (YOLO)

The "You Only Look Once" (YOLO) algorithm is a modern approach at computer vision and object classification. Its focal designer has a developed package known as "Darknet" that is a free and open-source software package. It is primarily written in C, which is a

lightweight language, meaning that unlike other languages such as Java, it has very little background activities built into the language and thus will execute uninhibited by unexpected functions. This allows the expectation that YOLO has a fast and efficient execution. YOLO is run as a compiled executable, therefore making it accessible from any language capable of running on your given operating system.[20]

YOLO natively only supports object classification in static, .jpg format images, and only performs these operations using the processor. YOLO is designed to be compiled specifically with OpenCV for video processing on video files or real-time video streams from a webcam.

3.11.5. Package Comparisons

Among the listed computer vision packages considered to perform our operations, the decision for which to consider using on our chosen FPGA is based on a couple of critical requirements: Speed, or how fast is the package is able to classify an object in our target image; Video processing capability, or whether the package is capable of natively handling video inputs, and what the computational cost of importing a second package would be; Processing requirements, or how much computational power the package needs overall in order to effectively perform the object classification in real-time. With these considerations, our choice ends up falling to the You Only Look Once (YOLO) algorithm.

Most importantly, it is faster than the alternatives, despite even in the particular case of using OpenCV for a partial amount of its computation. The reason this is because of the YOLO algorithm's novel practice of applying a convolutional neural-network across the whole of the image, and only needing to actually scan, or look at, the image one time. The other packages, when performing the actual object detection, use older, filter-based methods, in which filters are applied one-by-one across the image, and features are identified by the likelihood of particular features appearing in certain patterns across the image. OpenCV specifically uses a cascading classifier, which, while a relatively fast algorithm amongst its peers, still suffers from the drawbacks of a multi-pass filter algorithm – that is, having to scan the image many times, which can be computationally costly.

The YOLO algorithm, again, does not contain its own video processing capabilities, and herein lies its unique setback of depending on third-party resources. By relying on the OpenCV libraries to perform its video processing, it increases the amount of computational resources required to perform its object classification. OpenCV itself specifically requires the use of the system's dedicated graphics-processor unit in order to efficiently perform analysis on any video-format file. In this way can these packages reduce the strain of computational cost by offloading the task of video processing to a dedicated graphics-processing unit.

Finally, since computational resources are potentially among the most constraining limitations we will have on our RTDS turret, it will also require special consideration. It is important to note that the Darknet YOLO implementation also contains a Tiny YOLO implementation, which is a computationally less expensive YOLO model to be used on constrained systems such as our RTDS turret. That aside, YOLO's use of a convolutional neural net and its avoidance of rescanning the target image multiple times creates an offset in the amount of processing power it needs. This combined with the smaller neural network

used by Tiny YOLO results in YOLO being a sufficiently light-weight computer vision package for our constrained turret system.

3.12. Software Languages

Given that a portion of our operations will need to be run autonomously outside of the chosen computer vision package, a primary software language must be considered. This language will be used with the YOLO algorithm in order to call its execution and pass it relevant parameters based on the external stimuli provided by the camera of the RTDS turret. The chosen software package will very likely also play a critical part in the testing portions of our design process, as we test the RTDS tracking algorithm off of the actual turret hardware en lieu of an actual personal computer. Having the right language for any of these jobs will ensure the least amount of time is spent trying to implement our design due to language limitations.

3.12.1. Python

Python is a popular scripting language with large support within the computer vision community. Many of the core computer vision packages are written in Python, such as OpenCV, and therefore makes interfacing directly with that package very straightforward. It runs on most every operating system, most specifically the most popular ones like Windows, Mac OS, and most all of the popular Linux distributions like Ubuntu and Redhat.

Being a scripting language, Python stands out from most of the rest of the languages demonstrated in this list. Scripting languages are unique in that they are never compiled, but instead are translated into machine code and executed at runtime. It is for this reason that Python does not necessarily count as a programming language. True programming languages are compiled into machine-code executables before runtime, and then are simply executed without the need for any additional processing to be performed on them. Since scripting languages are translated into machine code during the actual execution, they can take longer than compiled languages to perform the same task, assuming the executable was pre-compiled.

Being a scripting language comes with its advantages however. Changes are able to be made very easily to scripting language code, since the executables do not need to be re-compiled. This could even mean that changes could be made during an operation to the script immediately before calling it. Ultimately, with Python being a scripting language, it makes implementation and maintenance very easy, but comes at the cost of some execution speed.

3.12.2. C

C is the most long-lived programming language of those considered for our experimentation and design. It is able to be compiled and executed on every major operating system—Windows, Mac OS, Linux—and for that is greatly versatile for any machine that a developer may wish to run it on.

Due to its elder nature, it is also perhaps the least computationally expensive of the languages, having the least amount of overhead for any of its statements. This means that

very little occurs in the background of C – what you write is almost exactly what will be performed on the hardware making it capable of running “very close to the metal.” This also makes C very capable of manipulating direct memory addresses and performing bitwise operations on variables natively, without accessing unusual libraries or having to manipulate functions that are otherwise protected from the user at the surface level in other programming languages. Short of writing directly in Assembly Code, C will execute the fastest given the same code and the same resources as its competition.

The limitation that comes with C is the very same as what makes it the fastest: no hidden quality-of-life background operations, and no hyper-powerful native functions, modules, or packages. This in turn requires that everything we may want to accomplish in C will have to be performed using a manually crafted implementation. Being a compiled language also means that any change to the source code must be recompiled and the executable replaced, making the language a little bit less agile whenever we wish to make slight adjustments to our RTDS turret.

3.12.3. C++

C++ bears many of the same characteristics of C. However, its differences are enough that examining them separately is worth the consideration. C++, much like C, is lighter-weight, has little if any uncontrollable background operations, is a compiled programming language, and overall has a pretty efficient runtime compared to other programming and scripting languages. One of the primary differences that C++ has over C though is its object-oriented design and larger, more sophisticated native background capabilities and functions.

Being a compiled programming language again presents itself as mostly a boon for C++. This allows it to directly execute commands without any of the preprocessing, translation, or compilation steps that scripting languages require while executing, saving it much time during the actual program runtime. C++ also inherited the “close to the metal” attribute from C—C++ has relatively close translations to operating system’s Assembly/machine code, and thus is less abstracted from what the computer will actually read during execution—which can not necessarily be said about all other programming languages. After all this, the extended list of native modules and functions gives C++ a benefit over C, making it easier to implement more complex operations, and easier to understand and manipulate more complex structures through its object-orientated nature.

The same capabilities that make C++ a better choice over C are at the same time the same things that might drive us away from using it. The increased power of the language and its functions also comes with the background baggage necessary to maintain it. With objects existing too, that means that simple variables may now no longer be raw types, but rather are now each individual objects themselves. All this combines to cause C++ to be slightly slower and less efficient on average, which may cause enough of a difference for our real-time image capture and processing of our flying drones that it might not be the best choice for the RTDS’s design.

3.12.4. Java

Java is another object-oriented programming language, which is also based off of C at its core. If C++ is a C-based language that slightly adds native quality-of-life methods and functions to make C easier to use, Java does the same thing, multiplied by ten. Java is both well-loved and notorious for its vast quantity of deep, immense libraries that cover most every capability imaginable. This gives the language a great range of power to perform any task needed by it, at the cost of being bloated with requirements in order to maintain all the standards that the different libraries require.

By using Java, our design would be at no shortage of pre-built tools for whatever testing or implementation task we would wish to perform. Interacting with the YOLO Darknet executable would be simple and manipulating the return data would as well. If needed, it would even be extraordinarily simple to piece together a custom object class using Java that would hold the return data from the Darknet operations, making them easily parsed for the following tracking algorithms. Java has a limitless range of utility, and like our other considered languages is very versatile and runnable on any major operating systems or operating system distributions.

With the great wealth of utility though comes the cost of unpredictability and inefficiency. While still more efficient and faster at execution than Python in virtually all regards, Java is last amongst its other compiled programming language peers. Much of this is due to its uncontrollable background processes. Most infamously is Java's wonderful "garbage collection" tool. Garbage collection is the automatic action Java performs in order to "clean up" dereferenced variables in memory in order to help mitigate memory leaks from occurring. It's a wonderful attribute of Java and is of particular good use for extremely large projects in which a memory leak might go unnoticed. But because we have no unprotected method to disable the collection or to have it only be enacted it manually, it is possible for it to begin performing garbage collection during our video processing, which would slow down the execution of our tracking algorithm, if but for a moment.

3.12.5. MATLAB

MATLAB is the other non-compiled scripting language being considered for testing and limited implementation of our drone-object classification and tracking. It is a privately maintained mathematical language by MathWorks and is designed to provide simple solutions to otherwise extremely complicated mathematical operations. It does have an interface to Python, thus allowing the use of specific Python modules for certain tasks that MATLAB is not able to do for itself natively.[11]

A lot of MATLAB's reportedly powerful capabilities comes from its access to diverse and unique packages and libraries. One of these libraries is MathWorks's Simulink, which is a code visualization tool. There are also MathWorks's APIs which allow interoperability not only with Python, but C and C++, as well as directly with computer vision libraries as well like OpenCV.

All of the background weight that MATLAB bears for all of its powerful capabilities also falls to the same judgement as Java and Python: it is slow, even slower than Python. Part of this can be reasoned away by the fact that MATLAB is not necessarily a production-level language, but rather a research language, designed for research and experimentation,

whose solutions would then be exported to a productionizable language such as those mentioned previously. MATLAB is also the first language considered to not be a free and open-source language. As a product of MathWorks, MATLAB requires a license to use, which suddenly has the unique consideration of a monetary constraint. This is not a negligible monetary constraint either—the MATLAB license has its own price tag, while each subsequent supplementary "toolbox," such as the Image Processing Toolbox, or the Computer Vision Toolbox, and every other utility, each have an additional not-insignificant price.

3.12.6. Language Comparisons

Among the listed software languages that will be used to test our design and potentially integrate as a medium between the tracking algorithm and the servo instruction output for the turret, many of the same considerations exist as those that existed for selecting the computer vision package. Those considerations were speed of program execution, how well the language will be able to integrate itself seamlessly with the YOLO Darknet package, and how computationally demanding it is to execute the particular language. These factors, while important, and still viewed as considerations, fall subject to one final requirement that we the developers have for the language, and that is ease and speed of use. It is for all these reasons that we have chosen Python as our primary programming language, despite its initial misgivings.

After examining our considerations of each of the listed programming languages, one might conclude that C would have been the ultimate optimal choice for our RTDS turret; And they would not necessarily be wrong. We can explore each of these reasons first, before discussing why those reasons do not necessarily outweigh the qualities that Python offers.

In September of 2017, Jane Elizabeth reported on results found by a team of Portuguese researchers, demonstrating C is among the fastest and most energy efficient programming languages that exist, by comparing equivalent operations across 27 different languages. These languages included C, C++, Python, and Java each individually. It is worth noting that MATLAB was not a part of the experiment, but it can be confidently assumed that it underperforms all other languages we are measuring.[7]

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Table 11, Language comparisons against energy consumption, execution time, and memory use

What is most noteworthy about their results, seen in Table 11 for our explanation here is how greatly C outperforms the other languages being considered. You could say that this is “by design”—perhaps C was not originally designed in such a way to be so efficient in 2019, but it certainly is maintained to that standard nowadays. It is for this reason that under these factors and in a vacuum, C would objectively be the best language for the RTDS turret algorithm to ensure it runs as quickly and as efficiently as possible. However, the lack of powerful libraries, high-level function abstractions, and a necessity to recompile code make C more unwieldy to use for the rapid development than other languages. So even if we “upgraded” C to another compiled language, such as C++ or even Java, the necessity to recompile code slows down developmental trials enough to warrant wanting a scripting language.

That is not to say that C will have no place in our implementation. RTDS is already using C simply by implementing YOLO: C is the language that YOLO is written and compiled in already. This fact is certainly attributing in part to its impressive benchmarks. Additionally, many of the standard serial communication formats, such as UART, SPI, and I2C, require an extension of C in order to establish its connections.

3.13. YOLO Training and Implementation

Training a new model for the YOLO library is the process in which an annotated dataset is provided to the YOLO algorithm, which it uses to create the model it will use against all other test images that it will be provided, including the actual implementation of tracking a drone.

3.13.1. Overview

Training begins with an annotated dataset. The dataset is a collection of various images that in some form pertain to the image or shape of the object that we wish to track or identify. The annotations are typically parallel text files that reference bounded areas on the corresponding images that indicate where an object of interest exists on the image, and of what class that object is. For our design, we only require our model to be able to identify one class, quadcopter-style drones, however YOLO's capabilities enable it to identify many different classes of objects simultaneously.

Figure 7 on the left is a trial image we provided to the YOLOv3 trained model during the initial tests performed on a commercial computer. Figure 7 on the right is the new predicted image returned by YOLO v3 model, with the bounding classifier boxes of all the objects that the model was able to identify.

Figure 7, base image passed to and predictor image returned from the YOLOv3 trained model

The pretrained model trained on the COCO dataset is surprisingly robust, giving credit to both the quality of the database the model was trained on, as well as to the strength of the YOLO algorithm itself. COCO is a large-scale object detection dataset that contains 123,287 unique images, 80 unique classes, and 886,284 instances of those objects within those images.

Given the documentation on YOLO bears bold claims on the algorithm's ability to generalize non-photograph images [YOLO paper, page 8], we provided our pre-trained model with a few non-photographic scenes in order to test the strength of these claims.

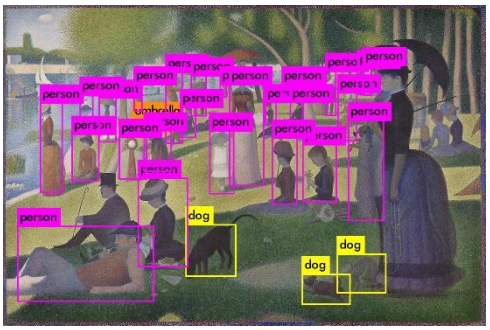


Figure 8, Generalizability of the YOLO algorithm

Figure 9, two misclassified objects in the painting of "A Sunday on La Grande Jatte"

While not perfect, the predictor clearly does a remarkable job of positively identifying the vast majority of people and animals in the image. A few notable mistakes are the monkey in the bottom right that is classified as a dog, and what appears to be a short lamp-post on the left that is classified as a person.

One of the most noteworthy takeaways from this initial experiment, however, is a verification of a claim made by Redmon et. al [YOLO]:

"Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background."

We see this both in our initial photograph and in the painting. The "more localization errors" can be understood to be the bounding boxes either misclassifying an object, not wholly bounding an object, or not classifying a positive object at all. In our street photograph, we can see a few localization errors: namely, the frontmost motorbike is not bounded in its entirety, and there is a second traffic light that is not bounded at all. In our painting, the examples provided in Figure 9 are localization errors, as well as a number of people in the foreground who evaded classification in their entirety.

The next claim that YOLO "is less likely to predict false positives on background" is clear. Very rarely, if ever, did any of our trials cause a random shape on the wall or background to cause YOLO to misclassify it as an object of interest. This will be a valuable trait for the RTDS turret as it attempts to identify and track a drone flying nearby, since a false positive on the background would cause our camera to center its vision on that one point and lock itself to a non-target.

3.13.2. The Algorithm

To get a better understanding of how the YOLO algorithm works and distinguishes itself, it would be best if we understood how its peers operate. Some of these other systems typically involve taking multiple unique patterns and that resemble the classifiers being searched for a scan the whole image. At certain intervals the patterns are looking for shapes in the image that most closely relate to noteworthy portions of their target classifier. This will create a temporary bounding box that will be refined by later processes in the algorithm.[6] The problem with approaches like this one seen in Figure 10 is that there are many parameters that need fine-tuning, which can result in needing to run trials for upwards of exponential orders of magnitudes of different parameter combinations in order to find your absolute minimum error.

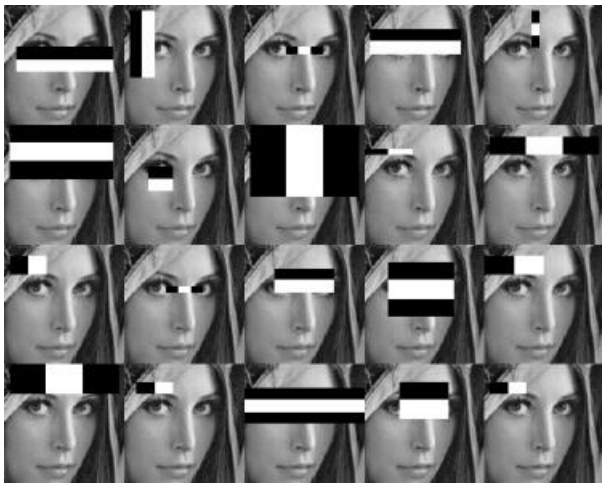


Figure 10, patterns being used to identify a face. Image used courtesy of Greg Borenstein via Creative Commons license.

YOLO's approach is one that reduces steps taken and reduces the number of times the image needs to be scanned and rescanned to only one time – hence, "You Only Look Once." A single neural network is used to on the image as a whole, which is divided into regions. The neural network then makes simultaneous predictions for various classifiers and their boundaries within these regions. Finally, non-max suppression ensures that objects identified on or across the borders of multiple regions are not classified multiple times.[19]

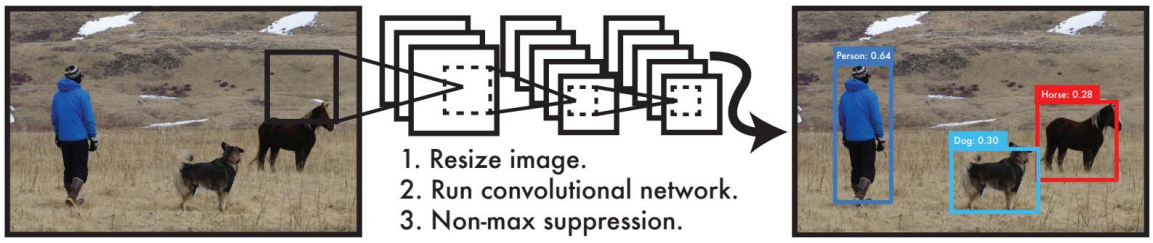


Figure 11, The simple logic behind the "You Only Look Once" computer vision algorithm. Courtesy of Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.

One way that the major difference between YOLO and previous image detection algorithms makes itself evident is how YOLO trains. Other algorithms will oftentimes require cropped images where the portion that should be bounded is the only portion supplied to the trainer. In Figure 10, the cropped image of a face is the type of data that would have multiple patterns (seen as the white and black boxes of various shapes overtop of the images) trained overtop of it to identify which patterns most closely resemble the general shape of a face. YOLO is capable of training simultaneously for multiple different classifiers in an image because it trains on whole images, using the pattern as described visually in Figure 11.

The YOLO algorithm uses a convolutional neural network, as in Figure 12. The initial portions of the network are designed to parse the image and identify the noteworthy features within the image, while the end of the network performs the classifier predictions and creates the final bounding boxes. While this style of convolutional neural network is

not novel by itself, the use of the one network over the whole of the image simultaneously is what makes YOLO stand out as faster, more accurate, and more easy to tune.

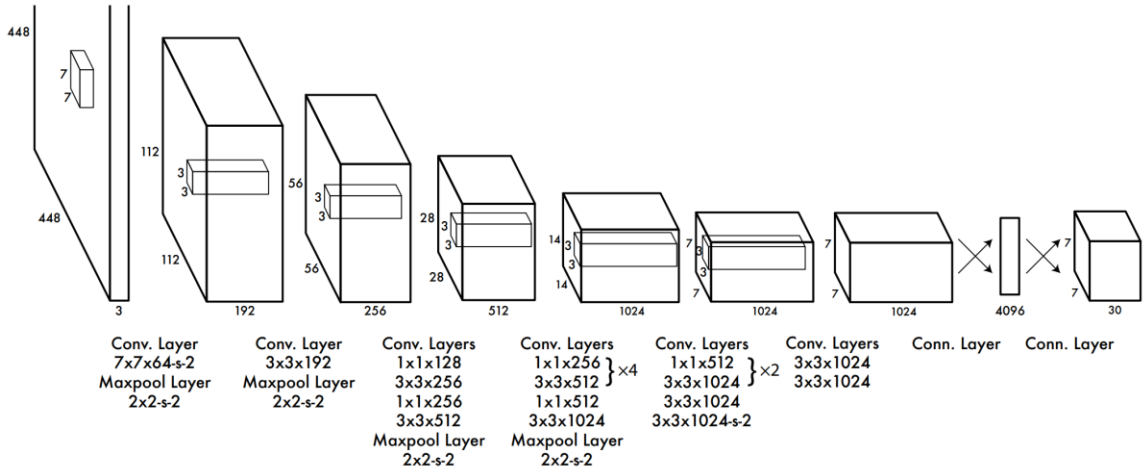


Figure 12, The YOLO architecture. Courtesy of Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.

From here, we must now take this generalized understanding of the function and implementation of the YOLO architecture and apply it to our RTDS turret, which is searching in real-time for a quadcopter drone using a streaming video feed.

3.13.3. Dataset & Dataset Preparation

Our dataset will be a collection of images of various objects in multiple perspectives that the YOLO algorithm will use as a baseline to learn from. As recommended by Alexey, a dataset can be of any number of images, but the larger and more diverse the set is, the lower the error that can be expected. Alexey recommends a size of at least 2000 images.[2]

Our dataset must be a very particular one given our target object that we hope to classify: a quadcopter style drone. For this reason, none of the pre-trained models that we have so far tested or experimented with will be suitable for our RTDS turret. Fortunately, an open-source dataset does exist that we will be able to take advantage of to train on, which features the recommended number of images of the drones we will be classifying.

The dataset we use must be manually prepared such that each image has a parallel text file that indicates the class (or classes) of object that is in the image, and the position of the bounding box around that object. Specialized toolsets exist that help to facilitate the creation of these annotated text files, but regardless, each image needs to be manually prepared so as to ensure that each training image is correctly classified and correctly bounded. One such tool is called LabelImg, which is a free and open-sourced image editor that allows for bounding boxes to be classified hand-drawn, and then are saved in a designated new directory.

3.13.3.1. Dataset Size

The size of a dataset will greatly affect the quality of the results of a machine learning problem and solution. The more information provided to a machine learning algorithm, the

more comprehension the learning model can gain regarding the features of the information, and thus the more educated of a conclusion it can draw.

Google's Machine Learning course gives a very basic rule of thumb, stating that "your model should train on at least an order of magnitude more examples than trainable parameters." While this rule of thumb makes it pretty clear how large a dataset should be for a model training on numerical data – for instance, if your model was training on data consisting of three numerical values per input, then you would likely want at least 30 examples for each class you expect to be identifying for your dataset – it still remains a little uncertain how large a dataset of images should be for a computer vision problem like the one the RTDS is solving. In our particular case for our RTDS turret project, we will have two types of physical drones that we will be classifying, and therefore at minimum two different classes. Therefore, if we were to assume that the same philosophy of dataset size as Google suggests applied in the same way to our computer vision problem, then we would only need about 20 images for each class.

The problem with this line of thinking is that this is equating features as classes, which is an erroneous assumption. Features are the points of data that each unique example piece of input bears. The class is the resulting definition, or conclusion, that our model should draw from those input data points. So, if the classes in the case of our RTDS turret training model are the two different types of drones we will be identifying at minimum, then the features will be any number of data points that can be gleaned from the input images. This, theoretically, could be an infinite number of features, if our training methodology has not enumerated them explicitly! Realistically this won't actually be infinite, though, because an image will have a limited number of data points due to its resolution size/number of pixels. Additionally, in practice we will want the model to develop an "eye" for the most important parts of the images provided. This is because a computer vision algorithm that constantly considers every single pixel as an "important" feature could end up being dreadfully inefficient and would very likely suffer from overfitting. So over time, the model would become aware of which parts of the images provided to it are the most important parts, and which attributes of those parts are the most likely to truly be the important parts. In order to provide the model with a suitable amount of data to be able to learn these patterns and draw these conclusions though, enough unique images would need to exist within the dataset.

This leads us to the next logical question: What number of images per classification would be sufficient for a computer vision dataset? Here we can look to some of the more particular recommendations surrounding the YOLO algorithm training methods to find a suitable response to this question. We learn from the README document of Alexey's forked repository of Redmon's DarkNet YOLO implementation the recommendations for how to improve object detection. Among these recommendations is the most basic, simplest requirement that the dataset will require that Alexey provides: "for each object which you want to detect - there must be at least 1 similar object in the Training dataset with about the same: shape, side of object, relative size, angle of rotation, tilt, illumination."[2] So the extreme bare minimum size that our dataset could be for our two different classes would be two images, one per class, with each of those two images bearing many of the same attributes that we would expect our prediction environment to have.

The very immediate and obvious problem with this bare minimum is the lack of additional examples to demonstrate to our model what the full three-dimensional shape of our two classes looks like. With only one image for a given class, there would be so little information that it would almost certainly be incapable of classifying any other image correctly outside of near replicas of the original training image – and the training image itself. Among other things, the model would have no context as to the depth of the image of the drone or the negative space surrounding it in the image. It would not recognize where the shape of the drone ended, and the negative space began. As your dataset size increased to include more examples of the same class, the model would be able to more completely understand what important attributes – or features – the drone has, versus the changing background data.

With this in mind, Alexey continues: "So desirable that your training dataset include images with objects at diffrent (sic): scales, rotations, lightings, from different sides, on different backgrounds - you should preferably have 2000 different images for each class or more." Immediately Alexey makes it clear the sheer caliber of size that is required for an ideal dataset. For the RTDS turret YOLO model, 4000 images total would be needed, with 2000 each being of the two different classes of drones that will be identified.[2]

This recommendation is, however, viewed both by Alexey himself and by commenters on his YOLO implementation as an ideal, and not as a hard-and-fast rule. Alexey uses the word "preferably" to explain how requisite it is that the dataset be of this size. A user on StackOverflow by the name gameon67 adds "1000 per class is not bad also. Even with hundreds of images per class you can still get decent (not optimum) result. Just collect as many images as you can."[8]

Our intention for the RTDS turret is to train it for initially 300 unique images for each class, for a total of 600 images. Should this prove a substantial quantity enough to garner confidence in the model's prediction accuracy, then the additional 2000 images of random, other drones that already exist in the DroneNet dataset will be added to the total dataset. The model will be retrained for three classes: Our two unique drone models, and a third, ambiguous drone type that embodies all other drones that do not match specifically our two drones. At any point in the process, the size of the dataset that includes images of just our two drones can be added upon, from the initial 300 images each, in order to bolster the effectiveness of the model to differentiate our two drones from one another, and from other drones.

The important information to keep in mind is that as the portion of the dataset including just the images of our two drones grows, a certain threshold will be reached in which adding more images to the dataset will not add any functional improvement to the classification capabilities of our model. This is an observable phenomenon true for most all machine learning models and algorithms.[5] Up until that point however, there can be an expectation that the improvement will be dramatic as the learning set size increases, up until it hits the vertical threshold.

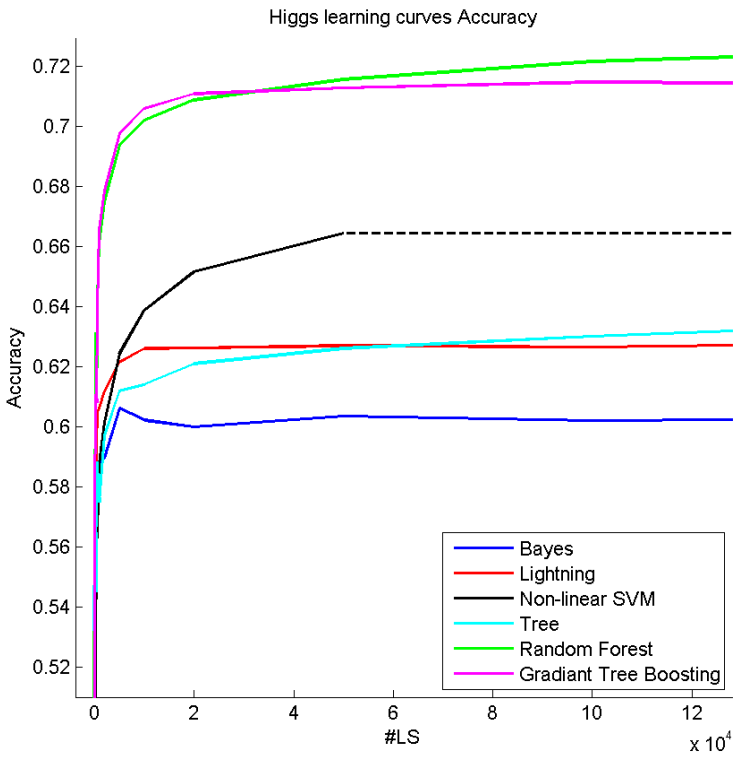


Figure 13, Accuracy of various machine learning models versus learning set sizes. Image used courtesy of AdrienNK via Creative Commons license.

As demonstrated in Figure 13, the accuracy of our YOLO machine learning model can be expected to grow only up to a certain threshold, at which point adding further images to our dataset (or learning set) will not improve its prediction accuracy.[18] Part of the RTDS implementation will include building the dataset and discovering the optimal minimum size dataset necessary in order to achieve high prediction accuracy during run-time.

4. Design Constraints and Standards

This section is an overview of all the project constraints and standards that our project will abide by. From design constraints put in place by budget being that were college students, to legal constraints there to keep us away from going to jail. Being that our initial design was going to be weaponized to “Take Down” the drone that was one aspect of our project that was already constrained. Constraints will also detail the expected functionality for when our project is completed. They will go over how if our project were to be scaled up to a full-fledged weaponized system how it could in the future protect us from attacks by drones. Lastly standards will go over the general protocols and procedures that our project will abide by when its fully functional.

4.1. Realistic Project Constraints

Depending on our we program our software, our project could be used to target more than just drones. Given the danger of this type of application we will limit it to only being able to be trained to recognize different varieties of drones. With that said there are still a number of different use cases for our project in the real world. The main such application would be for the everyday person to not want pesky drones creeping around their property or spying on their everyday actions. Being that privacy is almost a thing of the past in 2019, people would like a way to ensure there aren’t any drones looking into their bedroom windows at night. Next up would be more of a military application, to protect solders or people in danger from weaponized drones. In the next coming years, it’s quite likely that smaller and more available drones could be used to carry out attacks. The research we do now for how we could combat this could provide very useful in the future.

4.1.1. Economic Constraints

Many senior design projects are either sponsored by either UCF of some other large company looking to have some students do the research and planning for a future project for them. A variation of this very project was actually offered as a sponsored project however due to the requirements and constraints put on by the sponsor, we choose to take our own direction with it. With that said, computers for image processing are quite pricey given their needed power and speed. On top of that if we were going to be able to detect drones from more than ten feet away, we would need a much more expensive sensor system. Being that our project is internally sponsored by the members of our group, we must stay within a reasonable budget which significantly limits the design of our project.

It’s hard to compare the overall price of our project to similar other projects due to the fact that the first two are only available for military applications the price isn’t available. The net gun is just under $1000 but it lacks from any type of automation which really changes its price point. Being that our project won’t have a weapon but will have the automation which could quite easily be weaponized a price point of under $2000 is quite reasonable.

4.1.2. Environmental Constraints

Given the current state of climate change and pollution that we are currently facing on our planet ensuring that we choose to use as few parts as possible is key. Limiting the amount

of waste is a design factor that could be used to help advertise our project in the future for we could claim to be environmentally friendly. With that said the choice to use AC power instead of a battery is one way our project is being environmentally friendly for not only are lithium-ion batteries hard to recycle, they would be a waste of energy in the long run. There's no way to make our entire project environmentally friendly but we will do whatever we can.

When it comes to the main parts of our project such as the development board, microcontroller, and PCB these parts won't be able to be recycled. Thus, ensuring that we choose quality parts that won't break overtime and need to be replaced is another way we can be environmentally friendly.

4.1.3. Social and Political Constraints

Our product would defiantly get some bad publicity if it were to be able to be used to target other object such as birds or even people. Ensuring that our software isn't able to be adapted to target such objects is crucial to our design. Apart from that our product is not going to be cheap and therefore will not be in the budget of the majority of people. If our project were to be continued it would be key to look into how to go about making it more affordable.

Being that we are only doing a prototype design for our project it will lack a means of actually taking down a drone. With that said, if a product like this were to be sold to the public, we would have to ensure we are being the initial creators wouldn't be sued down the road for the type of weapons users decide to implement on it. For there are many ways users could quite easily retrofit various types of weapons to make our harmless project into a lethal weapon.

Once we decided that we weren't going to weaponize our project we initially choose to use a laser as a proof of concept. However due to the dangers of blinding people with lasers we chose to use a simple bright flashlight. Although a flashlight won't cause any harm to a drone, if someone was using it to film a bright enough flashlight pointed directly at the drone's camera would render the video useless. This is socially accepted for there is nothing wrong at all with shining a flashlight at some pesky drone that's creeping around your property.

4.1.4. Legal Constraints

Due to the FAA having such tight regulations on weapons that could be used to harm airplanes we choose to not weaponize our project. Abiding by all the various laws regarding what is and isn't a weapon could get very tricky. This is why the first two similar products are only to be sold for military purposes, for use by the public could prove to be very dangerous. Our initial thoughts for weaponizing our project was some type of RF jamming device. However, The Communications Act prohibits any use of a Jammer in the United States with penalties of prison and $100,000 fines.

With all of this said, there technically wouldn't be anything stopping us from implementing some type of compressed air-based net that could be shot to take the drone down. However due to the previously mentioned budget constraints we choose against implementing this type of functionality. This will also ensure we are able to demo our project at the senior

design showcase for there's nothing wrong with waving around a flashlight. For the demo on UCF premises we've decided to use a fishing pole to demonstrate the drone flying since flying a drone is prohibited.

As previously mentioned, our software will need to ensure that a user won't be able to change the computer visions target from a drone to anything else. Giving users the ability to change the computer visons target would make our project exponentially more dangerous. Therefore, we need to take every step possible to ensure that our software is locked down from users input of other types of targets. Doing so will ensure that no legal action can be taken against us in a court of law.

4.1.5. Health and Safety Constraints

Being that our final project will be just about fully autonomous it's key that we take precautionary measures to ensure the safety of both people and animals. Ensuring our project doesn't end up being used to harm any people or animals down the road is crucial. Apart from the legal concerns, this is the other main reason we chose to not weaponize our project.

It is our responsibility to do everything that we can to ensure that our product will be only used for its intentions of targeting drones. The computer vision software is of our primary concern for it needs to be locked down to users can't enter different types of targets. Developing our software with this as a primary concern will undoubtably make it more complicated but is a must to ensure the health and safety of both people and animals.

4.1.6. Manufacturability Constraints

It is our desire that the design and implementation of the RTDS's algorithms, training data, and physical turret be fully reproducible. Achieving this goal would ultimately result in the RTDS being cost-efficient for its components, and easily manufactured on a non-individual or -unique scale. The constraints that therefore exist are those on average component cost, individual component availability, source code traceability, and overall component modularity and longevity.

The prototype design of the RTDS is expected to contain some specifically high-priced components, most notably its FPGA that will perform the major share of computations for object classification from a live video camera feed. Other major components will include: the encoded servos to be used as the joints for the mounted camera; the custom designed printed circuit board through which the servos will receive power and communicate with the core FPGA; and the general housing components that will encase the delicate electronics of our turret. While the FPGA will be vastly more expensive than the rest of the components, keeping the average cost of those other components down would help to justify the general manufacturing cost of the design as a whole.

Any design components that are chosen either as whole pieces (such as the selected encoded servo motors) or as parts of a whole (such as individual resistors selected to be used in the printed circuit board) must be available on a long term, or at least modularly replaceable with an equivalent part. Should any physical part of the design become obsolete and fall out of production, and should there not exist any acceptably equivalent parts available on the market, the whole design of the RTDS could immediately become

obsolete. It will therefore be critical for the continued manufacturability of the RTDS that selected components follow common accepted standards, and that any component that is custom-made (such as the printed circuit board as a whole) be easily reproducible both due to clear documentation and the availability of common production services.

The source code, regardless of the language being used or for what purpose, must be traceable, modular, reproducible—basically, everything that falls on the hardware as a manufacturability constraint also falls on the software. Should the software at any level need to be rewritten due to an advancing standard or a new component, it should be modular insomuch that only the code directly affected by the change would need any adjustment. This is achievable through strict adherence to the principle of least knowledge, or "Law of Demeter," and each written software component should have a minimal amount of knowledge of any other component, especially those with which it communicates, in order to maintain full code modularity.

Generally, these manufacturing constraints boil down to a necessity that each physical component and each software component be long-lasting and modular, while minimizing cost. Longevity will inspire confidence in the design, modularity will act as an insurance against unexpected replacements, and cost reduction will increase the realistic reproducibility.

4.1.7. Sustainability Constraints

In addition to environmental constraints dictating our use of not being directly harmful to the environment, sustainability constraints will encourage the use of sustainable sources of materials and resources wherever possible, while still maintaining financially sound manufacturing decisions.

Financial constraints unfortunately prevent the use of extremely high-priced experimental components that are effectively 100% sustainable. It does however put us on the path to strive towards making use of low-power modes on our device, using energy-efficient software and hardware designs, and choosing components that themselves are manufactured using sustainable practices and/or from sustainable resources, such as hardware that is produced from recycled materials, or that is produced using energy efficient methods.

4.1.8. Ethical Constraints

The current ethical constraints mostly rely upon correctly targeting only drones. The targeting system should be built expressly with accuracy in mind when determining targets, so it does not accidentally target people or animals.

The current implementation of the targeting scheme ensures that when below a certain threshold the drone will not lock on to the object. Common flying animals, and manmade structures should be surveyed with the trained network to discern the maximum confidence value produced for such items, evaluating the risk associated with the system mistakenly firing.

4.1.9. Time Constraints

The nature and conditions of this project design will see time as a critical resource that will affect most every decision that is made for RTDS. While every task performed on the progress of the RTDS is constrained by the time remaining before our final presentation, certain tasks that will be performed for the project will require notably more time than other.

4.1.9.1. Dataset Preparation

The dataset we use to train our learning model will shape the success of our design's ability to identify and track a quadcopter through the sky. Without a diverse, robust, and precise range of training images, many risks arise that would threaten the functionality of the turret. Some of these risks include: Identifying false-positives in the surrounding scene and tracking those instead of the drone; Failing to identify the drone at a particular angle, or even at all in the worst of cases; Creating too large of a bounding box around the target drone such that the turret can not accurately deduce where the drone actually is (such as if the bounding box consumes the whole image); etc.

It is for this reason that great care must be taken to ensure that the dataset chosen is large enough to cover every slight variation in position and angle a drone could be seen from, while also being precise such that the inclusion of any training image does not degrade the accuracy of our model. Searching for a dataset like this can, by itself, take a long time. If it needs to be built from scratch, building and preparing that dataset – which is to format or annotate each and every individual image manually so as to be usable by the training algorithm – can take even longer. Many machine learning algorithms stress the importance of a large dataset, which could mean anywhere from a couple hundred unique images – such as in a personally built dataset designed for a custom classifier – to hundreds of thousands, such as with the computer vision benchmarking dataset know as COCO. Regardless of the tasks yet needed to be done to have a prepared dataset, the time required to complete them is not negligible, and it is critical that we allot time for ourselves at the beginning of our design to ensure we have a dataset meeting the criteria recommended for the computer vision library we use.

4.1.9.2. Machine Learning Training Time

Similar to the dataset preparation time constraints, when training a machine learning algorithm there are many variables – quite literally in the form of parameters – that will require many sequences of trial and error. We expect to train many times and often in order to minimize the error of the precise and accurate learning model we hope to create. Training a large machine learning algorithm however is no short task and can very easily cause a day's worth of work to be lost while all that time is spent waiting on a model to finish.

In a machine learning algorithm, the parameters are the changeable values that the algorithm uses to adjust itself to the conditions it is being used for and the images it is trying to learn. Different parameters for a computer vision algorithm can cause it to identify different regions of interest in the images or to resize or alter the images it is learning from. While budgeting our time spent testing the algorithm, it will be necessary to leave ample room for adjustments to the model that will require training and re-training.

4.1.9.3. Labor Hours

Each member of the team bears obligations both to the research, development, and implementation of the RTDS design, as well as to exterior responsibilities outside of the Senior Design process. These exterior responsibilities can include other classes, work, religious responsibilities, and volunteer service, to name a few. Additionally, work-life balance suggests that a healthy relationship between work-related activities and leisure-, hobby-, or family-related activities also be met. These many additional obligations and activities, combined with the relatively short deadline by which the Senior Design project must be completed, assembled, and operational, leaves the team with only a certain number of labor hours left to work with.

Considering all these extra obligations will facilitate that realistic goals be set for the design and implementation of the RTDS turret. This must also require that the scope of the design does not grow outside of the realm of time that we members have to achieve it. Already, many design concepts had been cut from the core requirements of the design in order to ensure that the base capabilities of the RTDS turret would be completable. While these requirements have been removed from the base implementation, they still remain on the backlog of the task-list so that, should the RTDS be completed ahead of schedule and be fully and satisfactorily operational, the team can continue to improve the RTDS's capabilities.

4.2. Project Standards

The project standards define the minimum requirements of quality, safety, accessibility, and universality that the major components of the RTDS turret will be made with. Oftentimes these are industry standards that we have decided to impose onto our project's design ourselves in order to increase the reproducibility of the design by future engineers, as well as to ensure safe and simple compatibility between differing components that might otherwise not be compatible, due to being custom made – such as in the case of our custom designed power supply for our printed circuit board – or due to having been produced by different manufacturers or engineering firms – such as in the case of our two different development boards.

In other instances, some standards may not be a choice, and instead of self-imposing the standard for our benefit, the standard acts as a constraint around which we must work if we are to be successful in our implementation. One such example – although it is detailed earlier in this document in Section 4.3 – is the standard file-type format that the DarkNet implementation of the YOLO algorithm is able to train from. While we are not necessarily electing to abide by this image format standard, we are obliged to do so in order to be able to work with the YOLO algorithm.

Enumerated here are those standards that exist in the computer engineering industry that are being used by our design.

4.2.1. Power Supply Standards

Ensuring the safety and efficiency of our project we need to adopt some type of standard for our power supply. We have chosen to abide by the IEEE standard IEEE 1818-2017

which specifically standardizes in the use of low power AC or DC applications. This standard also goes over reliability and load characteristics for designing low power supplies. We are not designing our own power supply for the microcontroller or development board, however it's still crucial to ensure that the power adapter we do choose to use will be safe and effective. Doing so will ensure a baseline of safety and reliability which will not only help us sleep at night but will also provide beneficial if this project is to ever be continued into a more commercial scale.

4.2.2. PCB Standards

Ensuring that our printed circuit board meets some type of regulated standard is key. For without our printed circuit board controlling our sensor system, our project wouldn't be able to track drones whatsoever. With that said, for our printed circuit board all the standards are maintained by the Institute for Printed Circuits (IPC). Now when it comes to purchasing these standards for our project that is a real issue being that were only college students self-funding this project and have no financial means to purchase these expensive standards. With that said we will use this section of our senior design document to go over the generally accepted standards in great detail for when it comes to designing and building a printed circuit board. In order to ensure not only functionality but also safety we must take a few key factors of our printed circuit board design into account.

The first of which is PCB materials for we may be able to find some cheaper boards, but they could not be as safe as others which are more widely used. The second aspect to take into consideration to ensure reliability and safety is later thickness. Layer thickness is important because using a standard layer thickness will result in faster manufacturing and cheaper manufacturing. While manufacturing and cheaper manufacturing sounds ideal there could be reasons to go about adopting our own custom PCB layer thickness. The last factor to take into consideration when it comes to ensuring reliability and safety is solderability, for although it would be super nice to have a small PCB it would prove to be more of a challenge when it comes to soldering components on. Ensuring we take all of these factors into consideration when designing our PCB is crucial to ensure our project is a success.

When it comes to PCB materials there aren't too many options to consider however there are a few materials that are generally used. The first and most common material that's used is known as FR-4 and is best described as reinforced glass epoxy. RF-4 is not only versatile but is also a common standard for when it comes to material for PCB manufacturing. One of the main benefits of FR-4 is that it is not only a great electrical insulator, it is also very strong when taking into consideration of its weight, and on top of all that it's also flame resistant. For these reasons listed and more FR-4 is a solid choice for our PCB design material.

The only other PCB material to consider is what's known as PTFE and it's generally used for more precise needs. PTFE is generally used when the PCB needs to support high speeds or frequencies while also being very accurate. It's also flame resist and very strong considering its weight.

When it comes to solderability we are referring to how successful we can expect each solder we to be successful. There are many factors to consider such as how much copper coating to use for each pin and heat of the soldier. Another factor to consider is PCB layout, if we have components that are very close together it will be more complicated to solder. We will need to find a happy median between making our PCB too large and thus more expensive and making it too small and hard to solder. Ensuring that we do the best job we can do to solder on the components is key for troubleshooting a bad solder can be very cumbersome. By following some common standards on how to properly and safety solder we can ensure that we will be successful.

The last factor to consider for our PCB design is thickness of each layer on the board. Thinness of each layer is an important factor many reasons, one of which is a thicker layer will support bigger components. Many manufactures state that the standard for PCB thickness is 1.57 mm. It's more of a historical standard for this thickness was used back in the earlier days of PCB design. There is a range of PCB layer thickness that is considered a standard among many manufactures and it's between 0.78mm and 1.57mm. In general, there are many design factors that must be taken into consideration when picking PCB thickness but only a few of them apply to our project

Copper thickness is an important factor for it determines how efficient the current flow will be. Copper thickness can also impact signal flow which could really cause some issues depending on the requirements for the application. Signals are mainly only impacted by copper thickness at high frequencies above 100Mhz. With this in consideration given our project's application we won't be reaching any frequency's nearly that high. With that said if we adopt the standard for copper thickness, we shouldn't have any issues with either current or signal flow.

The other factors to consider are operating environment, number of PCB layers and signal types. Being that our project is more of a proof of concept it will be used in ideal conditions. This means that the operating environment for our project won't be anything crazy, the PCB used for our project won't be exposed to any extremely hot or cold temperatures. The PCB also shouldn't be exposed to any water or other dust or debris. Hopefully in future iterations of this product it will be able to combat harsh conditions, and thus the operating environment will have a larger impact on the design of our PCB. The next factor to consider is number of PCB layers, this is important for if we're trying to have multiple layers the PCB design will be much more complex. Being that our PCB is only used to control the position of two servos we are only going to have the minimum which is two layers and thus this factor isn't anything we have to worry about either. The last factor to consider for our PCB design is signal types, this many has to do with the frequency of the signals. As previously mentioned, our requirements don't need us to do anything over 100Mhz, this adopting a general standard will suffice.[17]

By familiarizing ourselves with the available documentation provided by IPC, we shall be able to use performance class 1 standards since our project would ideally be a general consumer product. These IPC standards will help us ensure our PCB is not only built correctly but also built with the right materials.[18]

4.2.3. Arduino Application Standards

When it comes to software application standards there isn't much to go by for Arduino's IDE. With that said simply adopting some general programming practices will help ensure our code is not only legible but efficient and effective. Programming practices are importing for many reasons, one of which is readability. Being that we would like this project to be continued in the future ensuring some programming standards is key.

4.3. Tiny YOLO and YOLO v3 Video Input Constraints

Using the "You Only Look Once" computer vision algorithm and library will carry with it a number of constraints regarding the type of input data that it will be capable of understanding and tracking. YOLO v3 is the currently accepted, general-use implementation of the YOLO algorithm maintained by Joseph Redmon, one of the lead designers of YOLO. Parallel with YOLO v3 is the light-weight, faster-running Tiny YOLO. While their training parameters and creation of learning models differ between the two, the input and output data the receive and return, respectively, are constrained by the design of the core YOLO algorithm itself, as well as its supporting libraries that it depends on to perform higher functions.

While static images passed to YOLO v3 and Tiny YOLO are constrained by the strict file formats that YOLO was designed to read, object classification using real-time video stream or on a video file is actually constrained by the file format constraints of OpenCV, another computer vision library. When performing real-time video object classification, YOLO v3 and Tiny YOLO must be compiled using the system's installed OpenCV library. OpenCV is then used by the YOLO package to receive and parse the video input. YOLO then performs its convolutional neural net analysis on the current frame of the video as provided by OpenCV.

4.4. Latency Constraints

The focus of the project is implementing a real time targeting system that is dependent on multiple components to work proficiently. The main components involved are the targeting system, the guidance system, and the firing system for the platform.

4.4.1. Master to Slave Communication Constraints

The maximum speed of communication between the development board and the slave board is a potential bottleneck for the system. For the system to work as expected, the optimal latency of communication between commands between the slave and master should be near nonexistent.

4.4.1.1. Dedicated I/O lines for Communication

The quickest proposed methodology of communicating to the slave board would be through a dedicated I/O line. This method would send the entire packet over in one clock cycle, since the number of lines would be the size of the data packet needed for the slave to drive the guidance system.

Currently the proposed word length would be between 24 and 32 bits, with a byte per type of data needed to be directed towards the guidance system. It is the I/O boards purpose to ease the processing on the development board, making the slave translate information from the development board into driving the servo smoothly to reduce screen blurring on the input sensor.

Drawbacks to this implementation include the lack of flexibility in design if changes needed to be made in later stages of the project. Additionally, the size of the word length would be fixed, thus if a design changed necessitated that the word size be extended past the size of the I/O constraint, then the entire design would need to be redone.

The implementation of this method is plausible, as the ZCU104, or the Jetson Nano has FMC connector, or sufficient header space respectively, to allow for 32 bits of direct communication through GPIO lines.[12] Although, due to the fixed word length, the utilization of microcontroller pins is also wasteful, making this implementation rather unsatisfactory.

4.4.1.2. USB Communication

USB is supported on the ZCU104 and the Jetson boards. It is a relatively quick protocol, and is supported universally by most kernels, and development boards. The speed of the protocol is directly dependent on the baud rate of the protocol. The baud rate must be agreed upon by both sides of the communication, as the sampling rate of the receiving side must be able to catch all the bits sent in the communication.

Due to the wide support of the protocol, ease of implementation, and wide documentation, UART/USB communication allows for a robust and relatively quick communication of data between the slave and the master. The data sent by the UART protocol is limited to one byte of data transmission per communication, thus the information must be sent by a set of 3-4 bytes at a time, or four transmissions to give a command between the development board and the slave.

The UART communication could provide problems when reading a stream of information and the frame of the word packet is misaligned. Due to no flow control, and possible overfilling of a buffer of the receiving FIFO on the slave board's I/O, a certain type of rudimentary flow control would need to be implemented to ensure that the data sent to the slave board will not be lost. Other types of robustness will be implemented as needed during testing.

Due to the highest speed of UART being 11520 baud, meaning that 32 bits will be transmitted in 0.003 seconds, which is more than 5 times quicker than the maximum frames per second (0.016 seconds for one frame).

4.4.1.3. I2C Communication

The I2C protocol is robust and implemented by virtually all development boards and is useful when needing to communicate to multiple devices on a data line. The I2C protocol has a multitude of speeds, including 100kbits/s, 400kbits/s, and 1Mbits/s. The communication allows for multiple masters to control the bus and communicate to any other devices on the bus.

Due to only one other device being on the bus, it is not of the highest priority to implement a protocol that is prioritizes the linking of multiple devices on one bus.

4.4.1.4. Chosen Protocol

USB is the most flexible protocol focusing on speed for the parameters specified by the project. Due to the possibility of changing the packet exchange between the two boards, it is more desirable to have a USB communication to send information between the two boards.

Due to speed being imperative, currently the highest baud rate is being chosen. For robustness, oversampling will be performed on the receiving side, to decrease the chance that bits will be missed and that the frame of the data will be upset. Flow control will be implemented as needed.

4.4.2. Guidance System Movement Speed Constraints

The movement of the Guidance System could be a bottleneck for the system's performance, due to the movement of the sensor having a direct impact on the ability of the platform to track targets. Due to the high need for quick smooth movement, it is imperative that the system accept input data from the slave board and move nigh instantaneously.

The slave I/O board supplies the power to the Guidance system and communicates over header GPIO pins to the guidance system's servos. Due to the communication architecture simply being direct wire connections, it is relatively quick in between the command sent from the slave board to actual movement of the actuator.

Due to the existing setup of driving servos, there is no proposed improvement for this information. The setup of the guidance system is a direct connect from GPIO pins to the servo, with all necessary information sent via those channels.

4.4.3. Firing System Communication Constraints

The targeting system must be able to fire at the target near instantaneously, due to the very unstable nature of location of the drone in respect to time. Currently, the proof of concept implementation is instantaneous, since it is a laser directed towards the drone, and the speed of light is currently the quickest speed achievable.

After the proof of concept is completed, more robust implementations will need to be thought of to make the system more robust. These constraints will have to be the range of the system, the speed of the projectiles made by the system, and the speed that the system can rotate, if the system is separate from the camera's guidance system.

With finite range, it is up to the motion detection and range finding system to determine if the drone is within an acceptable range. Of course, this is based on the assumption that the drone's size is known before, and thus is able to be approximated of its distance based solely on the size of the target in the camera screen.

4.4.4. Camera Input Constraints

The type of camera dictates what formats are natively supported that can be sent to the development board. Concerning the video pipeline, it is of importance the format of the video, it's framerate, aspect ratio, and size of the input video. Due to the YOLO constraints of image size and size of bounding boxes, the camera input must match the same information that the model was trained on.

Additionally, the optical flow module that will be used in conjunction with the YOLO module must ensure that the format from the camera source is able to be translated accurately to a usable color domain. The framerate is of concern, due to the dependency that accuracy has on the speed of new images being placed into the classification and augmentation pipelines needed to approximate velocity and classify drones in the camera's view.

5. Project Design

The project consists of four separate components, the chosen development board, the student created slave interface board, the firing mechanism, and the guidance structure for the image sensor. Each system must work together quickly to move from targeting of the drone, tracking the drone's direction, and moving the sensor so that the image of the drone will remain in the sensors view.

For the project to successfully execute, the latency of each system must be minimized. For the chosen development board, the ZCU104, this means ensuring that each frame is processed as quickly as possible. The student created slave board must be able to receive commands using a quick and robust communication protocol, to ensure the validity of data with the real time system. The current implementation of the firing structure is simply a laser.

5.1. Development Board

The development board serves as the master of the entire platform and directs the other two subsystems. The ZCU104 utilizes the powerful Zynq ZU7EV chip to quickly process large amounts of data in real time. Used in conjunction with Xilinx's proprietary DNNDK/Revision solution package, the board can be used for object detection and tracking.

The project will utilize the YOLO architecture due to its quick and accurate classification of objects. The YOLO architecture is further optimized by pruning by the DNNDK DECENT optimization solution, which prunes off unnecessary nodes in the convolutional neural network. Absolute coordinates are given for the object in the image, allowing for tracking of the position of objects in the frame.

The board has enough RAM to enable buffering of image frames as necessary for incoming image data. The speed and efficiency of the algorithm after pruning allows for the board to operate above thirty frames per second.

The ZCU104 is the main decider of what signals are sent to the I/O Slave, and whether to "fire". The ZCU104 will operate in a state machine, determining what signals to send to the slave board and toggling fire commands.

To determine velocity of tracked objects, currently the ZCU104 will utilize a naïve vector implementation, using two points and the time difference between the two points. In conjunction with the points, the size of the object will also be considered when creating the velocity vector.

5.2. I/O Slave Board

The student created slave board will communicate with the guidance system. The slave board is only concerned in quickly queueing information sent by the ZCU104 and

executing those commands by toggling General Purpose Input Output Pins to communicate with the guidance structure.

The I/O slave board will be microcontroller based, and as such, will run in a permanent sleep mode until risen through interrupt by signals from the ZCU104. Word packets from the ZCU104 to the I/O board will be fixed length and communicate information about the speed and direction that the target is moving. It is the slave's responsibility to accurately calculate the necessary outputs to export to the guidance structure to maintain the target within the frame.

A momentum variable should be attributed to movement of the guidance structure, to determine how forcefully the movements should be to maintain focus of the target. If too much power is applied, and the drone makes negative progress in respect to the frame, the momentum variable should reflect this result and slow the intensity of movements in the current direction.

The direction of communication between the slave and development board is one way. All communication will be a fixed word length from development board to the slave. The reflection of received signals will be from the sensor input.

The slave should be able to determine how many servos that it has connected to on the guidance structure, and as such, be able to dynamically modify its response to the development boards commands to accurately adjust the guidance structure accordingly. The minimum number of servos is two for correct operation of the platform. The current maximum number of servos that can be connected to the board is not defined for the project, however, for each added servo, the complexity of the slave microcontroller code will increase, possibly resulting in a decrease of performance for this component of the project.

5.3. Guidance Structure

The guidance structure consists of the servos and sensor of the platform, and the architecture needed to maintain steady focus over an area. The minimum number of servos needed to be supported by the guidance structure is two, however, the maximum amount needed is not yet defined.

The guidance structure will need to be able to smoothly move from one location to another, to reduce the resultant blur on the sensor. It is imperative that the guidance structure be quick, yet not jitter to reduce bad frames entering the master pipeline, causing possibly misconstrued commands to be relayed back through to the guidance structure.

The guidance structure will accept all information from the slave I/O board. There is an abstraction between the guidance structure and the development board. The development board does not know the state of the servos, only its current field of view. During startup of the system, it is imperative that a zero command is available so that the development board's center, and the servos center is aligned, subverting any chances that the development board might overstep the range of the servos.

5.4. Firing Structure

The firing structure is directly tied to the development board, using a PMOD connector. When the development board decides to issue a firing command, it must be immediate, therefore the latency of transactions between the slave board and development board is unacceptable.

The current firing structure is simply a laser connected to a switch. Driving low on the enable pin will ground the power line of the laser, meaning that it will turn on. The lasers power source is independent of the development board and the slave I/O board.

As the project progresses, there is room for modular reassignment to the firing structure, however, firing mechanisms would further complicate the necessary transaction structure, as distance would have to be more accurately determined, and absolute location of the drone will need higher confidence intervals.

The location of the firing system in respect to the sensor is of great importance, as the offset between the two devices must be measured and accounted for to increase the chances of successful hits on the target. The perfect placement of the laser will be found experimentally during the fabrication of the platform. As of right now for testing purposes the laser is mounted directly above the camera. The main benefit of this positioning is that the offset between the center of the camera lenses and the focus point of the laser is almost as small as possible.

For proof of concept the red laser will be used to determine if the drone has been “hit”, as the drone will have a red dot on it, the region of interest can be evaluated, and contrasted in the Red Green Blue spectrum. Analyzing the red spectrum, a comparison can be made between the previous frame to see if a red intensity has gone up. If the intensities have gone up, then a hit is determined.

5.5. Graphical User Interface

The platform will have a somewhat intuitive interface that will demonstrate on screen what the system is currently seeing. The interface will be able to show the approximated velocity of the drone, a rectangular box detecting the drone, and other features.

The detection region on the screen will be limited by the camera view size, however, will be able to display in real time the drone that is being tracked. The GUI shall display the approximate direction, motion, and predicted movement that the platform should attempt to take in order to track the drone. Additionally, the classifying type of drone will be used to specify if the drone is of a certain type.

The detection system will attempt to handle blurs as well as possible by maintaining the control of the motors, so that the GUI will be able to show more pleasantly. Additionally, the GUI will allow for users a rudimentary way to interact, and if there is a certain target that is under a certain threshold, it will track but allow for the user to deselect if necessary.

The GUI will display the color of the encircling box of the target with a green color. Next to the green color will be the name of the current targeted information, including how much it thinks it is a drone (the confidence interval). The GUI will only show up for confidence intervals more than 30 percent, and anything below 50 will have the user intervene.

The GUI will have both the targeting information, and the untargeted information as contrast. The video will stream at 30FPS and be able to show in three channel RGB color. Additionally, the GUI will have a way for the user to view the tracking path of the drone, if desired, and also view the function of momentum sent to the servos. The power sent to the servos will also be shown, demonstrating the relationship between the speed and the power consumption of the servos. This will show the responsiveness of the system on variable different inputs.

The GUI will have a method of determining where it is looking in its current 180-degree domain. The platform can only operate in a 180-degree pattern, as constrained by the servos, and so, must be able to display where it is in that domain. When the domain is being infringed upon, an error will pop up on the screen telling the user that the drone has surpassed its domain.

The GUI will also be able to show to the user when a successful hit has been determined or not, and also update the current status that the system is in. The status will be color coded as green for idle, yellow for tracking, and red for firing. The system will have a bar on the side with the text name, and color coding so that the user can determine what state that the system is in. The state is also important in determining whether or not the drone is functioning correctly.

5.6. Log System

The platform must have a way for debugging functionality to occur, and as such will have a debugging system that will print out errors and current states of the system. It is imperative to have good system environment awareness when dealing with the tracking system, as it is complex. To mitigate the complexity, each module in the system will have an associated designated text file that will notify the user what system functionality is being infringed upon. The files include the visual error log, the processing error log, the communication error log, and the latency error log.

The visual error log will have everything that is related to camera functioning and problem handling. As the platform will be running a Linux operating system, it must have certain privileges and rights when accessing the camera. The information printed in this log is only concerning boot up, and any errors that have to do with the camera processing module. These errors will refer to the line in code that they are derived from, if it is code based. The camera is ported into the scripts by accessing “dev/video0” as there are no other cameras on the platform, however, this is possible for issues to occur if there are multiple cameras attached to the system. To increase debugging potential, a list of all possible input camera devices is posted on bootup, associated with their selected resolution, and type of video input. Another possible mismatch is the type of the video input into the image processing pipeline, as the pipeline is formatted for a very specific type of video input that must not be given the wrong input, lest the error output be displayed in the pipeline. The input error is the only error that is displayed in two files, the visual error log, and the processing error log.

The processing error log is used to dictate any possible errors that may occur in the image processing system, and can include, input mismatch, and possible mistakes involving handling of the image data. Most of the mistakes in the processing error log have to do

with receiving and outputting data. The processing side will start a pipeline between the camera and the output with a few buffers and threading to run optical flow for velocity direction and YOLO for object detection. The processing error log also outputs any generic Linux error outputs that may or may not have to do with the software that is running on the hardware.

The communication error log is of interest, as it will be the most densely packed involving thousands of data transfers, per second. The error log will consist only of the types of errors involved with communication, which is UART in the project. The error will include the baud rate, the type of error associated, and the current state of the system. The error log will also output the buffers present in the hardware to see if the project is indeed sending messages over UART to the application, as well as the port used. The port will also be shown as being open or closed in the error log, determining if there is an error with the port. Permission problems that involve the UART communication will be written to this file, not the general processing error log.

Finally, as latency is a large component of the project, the latency will be hard set in the code to determine what is an acceptable latency. Low frame rates, or too high frame rates, including too delayed processing of either the pyramidal optical flow pipeline or the other YOLO pipeline will be written out as an error log to the latency error log. The latency error log is useful in determining problems with overall performance of the system and can be used to determine hardware issues.

6. Overall Integration, PCB Design and System Testing

Taking the combined knowledge gathered and specifications and constraints specified in the previous sections, the full hardware and software implementation of the RTDS can now be defined.

6.1. PCB Design

One of the main requirements for passing senior design 2 is building a solid PCB. Due to the nature of our project size of our PCB which is one of the main constraints for PCB design doesn't really apply to us. Of course, a smaller PCB would look better but when it comes to implementation, we are going to air on the side of making the PCB larger in order to ensure success of our design. Our main concern for out PCB will be ensuring that our power supply is significant enough for our two servos.

6.1.1. Schematic and PCB Software

In order to ensure that our PCB design is going to be effective building it on a breadboard will be very helpful. By building our same design on a breadboard we will be able to test each and every component before we solder them on the PCB. On top that we will be able to ensure that our design is effective and reliable before we go and order the PCB and end up wasting a bunch of time and money.

When it comes to a PCB IDE, we have chosen to use EasyEDA for it's a free software which has so far been quite user-friendly. There are many benefits to using a widely popular software, one of the most important ones is community support. Being that EasyEDA is a free software there are not only many YouTube video tutorials on how to use it but also many online forms to help troubleshooting. On top of that an account can be made to remotely save your PCB design online to ensure it doesn't end up getting lost. Another benefit to storing the PCB design online is that it's available for our whole group to easily see and modify. EasyEDA also has footprints for many components built into libraries which ensures that all the components for our PCB will fit as expected. This is ideal for it makes the PCB design less of a headache leaving only routing as the main issue to consider.

6.1.2. PCB Manufacturing and Soldering

As for getting our PCB built there are many different options but considering the simplicity of our design, we are only going to choose from two different manufactures. Once our design is finalized and we export it as a gerber file, we will need to send this file over to one of these manufacturing companies. A factor to consider when choosing a PCB manufacture is cost and time to build and ship. As expected, all the PCB manufactures based out of USA are significantly more expensive. As long as our project stays on schedule, we shouldn't have to worry about shipping times and may as well order our PCB from china and save our hard-earned money. Many manufactures won't even print a small number of PCB's for they are in it for the big purchases so we must ensure the company

we chose is okay with only making 3 of them. The two OCB manufactures that we are considering are JLCPCB and PCBWay and are both quite cheap and fast considering how long shipping is going to take. Table 12 shows the standard quote for getting a PCB built from each of these companies. Being that we don't have our design finalized we can't get any exact quotes just yet.

	JLCPCB	PCBWay
Size	?	?
Layers	2	2
Country	China	China
Price	?	?
Shipping	$16.67	$17

Table 12, PCB Manufacturer Comparison

Being that the prices are about the same we have chosen to go with PCBWay because their website was more user friendly and that's key in something that your unfamiliar with. Another benefit for choosing this manufacture is that many other groups have use them and haven't had any issues as far as we are aware.

Once the board is built and shipped to us, we will still have to solder all of the various components on. Being that I have very little experience with soldering we may have to either reach out to a friend for a hand or practice a bit more before we end up messing up one of our three boards. Being that shipping is going to take a while we need to ensure that we don't end up breaking one of the boards due to a faulty soldering job. We could reach out to a company to have them professionally solder all of our components on but being that it would be quite pricey this will only be used as a last resort. As long as we don't wait until the last minute to solder everything together, we should have plenty of time to practice and ensure everything works on the first try.

6.1.3. PCB Layout and Design

As previously stated, due to the nature of our project we don't have any real size requirements. However, with that said there's no reason why our PCB should have to be any larger than 3 x 3 inches. In order to get the most of our money we will need to ensure we make use of the top and bottom layers of our PCB design. Making use of both sides of our PCB will also making the process of routing easier being that we won't have to worry as much about overlapping. Being that we are going to have two different layers we will need to make use of a PCB design concept known as vias. The propose of vias is to enable different layers of the PCB to transfer signal between them. They are implemented by drilling holes through the PCB so that the routing can take place between them. Then it comes to thickness as previously stated we will just adopt whatever is a standard recommended by the manufacture. PCBWay has standard thicknesses of anywhere between 0.1mm up to 0.2mm for free so we will probably go with 0.2mm so that the PCB will be as durable as possible for free.

When considering routing and soldering with PCB design the majority of components will be surface mount devices (SMD). When considering copper traces as previously stated we should be fine with going with a standard size for we shouldn't be pulling more than an amp at any point. With all of this said being that we only really need three jumpers on our PCB this design should be relatively simple and hopefully we won't run into any major issues. The general design for our PCB is below in Figure 14, while some of these aspects many change the general layout should remain the same. Both servos will have jumper cables that will be ran from the PCB to the servo, that will come off of headers on the PCB. The main benefit of this is that if we end up breaking one of our servos, we will be able to easily just put a new one on without having to solder everything apart and then back together. As for the UART connection between the PCB and development board we will just have a header so that the PCB will be able to easily be programmed and tested while also being able to receive data from the development board. Part of the UART jumper will also include a separate cable for the interrupt pin connected to the Arduino. This will enable us to quickly stop the Arduino from its scanning mode and tell it to start accepting serial data from the UART pins.

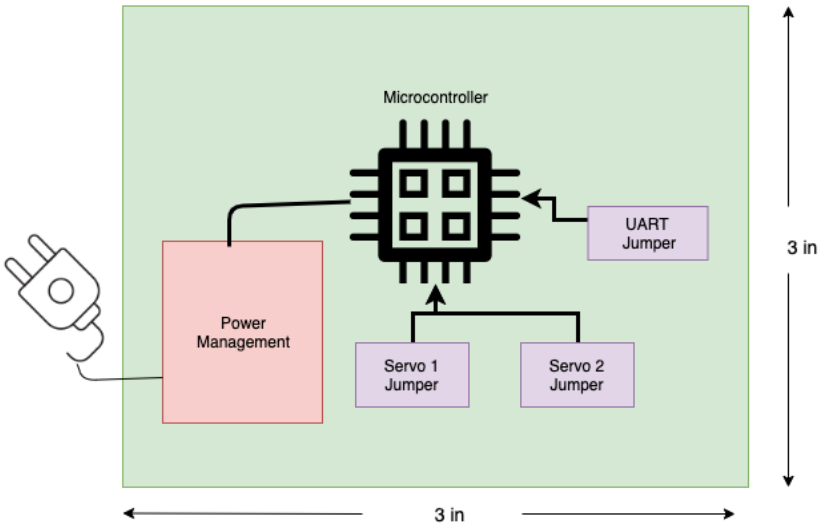


Figure 14, General PCB Design

6.1.4. Sensor System Housing

As for the housing for the sensor system we 3D printed a very simple and modifiable design which holes both of our SG90 servos. This design works for now but may have to be modified a little bit when it comes to mounting the camera and laser diode. The main benefit to this design is that its three different parts. This enable us to easily change the sensor system mounting plate while not affecting the main pan and tilt design. In order to protect the servos our scanning algorithm will have to limit its sweeps to what the pan and tilt 3D printed part allows. For instance, the servo may have a full range of 180 degrees however when placed in the 3D mount can only move 45 degrees in each direction. If we don't write software to stop the servo from trying to go the full 180 degrees, it will defiantly shorten the longevity of our servos and our overall design. The sensor system will also have to be mounted to a hard-flat stationary surface in order to stop it from tipping over and falling. Being that we choose to 3D print our sensor system housing we will have no

issues when it comes to modifying it so that the camera and laser diode will fit. All in all, our sensor system housing prototype design is just about fully functional, and we have a great start on it so far.

6.2. Software Design

The software design is the logical layout the RTDS source code will follow in order to make any duplicate hardware implementation reproducible. Here we focus on the implementation of the YOLO algorithm libraries, and the software environment that exists on our selected hardware.

6.2.1. Testing on Static Images

Initial tests were performed on the pre-trained YOLO model provided with the dataset of drone images, by Chuan-en Lin.[3] These tests were to test the capabilities of YOLO to classify drones generally, as well as classify drones that may not bear all the expected features of a quad-copter.

Figure 15, Classifying our testing drone

Figure 15 is one of the drones we intend to use ultimately for our RTSD implementation. It bears a standard quadcopter shape that was very clearly classified by the pre-trained model. Various angles and different distances of the photos of the drone show that YOLO is able to understand the general shape of the drone that we are seeking to classify. Even photos with many distracting details in the background and foreground demonstrate that we will be able to have confidence with our own trained model's predictions.

Additional experiments were performed in Figure 16 and Figure 17 in order to test the robustness of the model. Figure 16 is a prediction performed on a drone with six propellers (as opposed to the typical four of a quadcopter). Despite that the drone-trained YOLO model contains no images of six-copter designs, it is still able to abstract the shape of a drone enough to recognize our subject is still technically a drone. From the way the bounding box is drawn, however, it may be reasonable to conclude that what is actually being identified is the "quadcopter in the six-copter." Or rather, that the YOLO model has identified that shape of the four propellers in the middle of the drone and is effectively ignoring the other two propellers. While our actual implementation of the RTDS will not be tested on any six-copter style drones, it does help us to understand – for the purposes of presenting our design as a proof-of-concept – that our turret will be capable of classifying irregularly shaped drones. This adds a degree of surety to the

design, knowing that a minor change of drone body shape or number of propellers will not be able to evade detection.

Figure 16, a non-quadcopter drone correctly identified by our trained drone model. Image provided for non-commercial use courtesy of StickPNG.com.

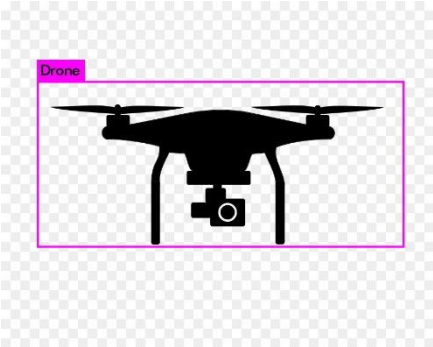


Figure 17, generalized, non-photograph images of a drone correctly identified. Image provided for non-commercial use courtesy of JeongGuHyeok.

Experiments were also run using cartoon generalizations of quadcopters, as another means of testing the limitations of YOLO's and our model's classification abilities. Much like the painting used previously, the cartoon generalization demonstrates the ability of the algorithm to understand the object as a whole and its inherent shape.

6.2.2. Testing on Video Stream

Initial successful tests were performed using OpenCV and the YOLO algorithm to demonstrate their ability to be integrated, as well as to demonstrate successfully using the pre-trained YOLO weights on the drone dataset. All test proved to be fruitful. Figure 18 shows a photograph of our live test of using an image of a drone being identified by YOLO from a live webcam video feed.

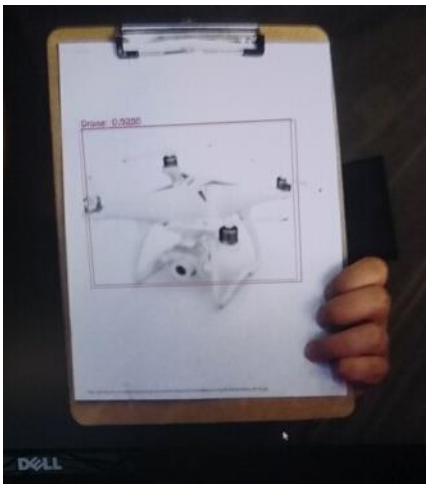


Figure 18: Successful test of identifying an image of a drone on a video stream

One of the immediate results learned from this test however was regarding the relatively high computational demands the standard YOLO v3 algorithm requires. While the YOLO v3 algorithm runs smoothly on a computer with a large dedicated graphics card, initial tests

on the Nvidia Jetson showed that even with its raw, dedicated processing power, we may have difficulties maintaining a high framerate from our artificial intelligence. It was for this issue however that Tiny YOLO was developed. In an attempt to assist developers with tighter hardware constraints to implement the learning algorithm, a shallower, less robust new version of YOLO was created.

6.2.3. YOLO v3 vs Tiny YOLO

YOLO v3 is the name of the currently most optimized implementation of the YOLO algorithm as originally discussed by Redmon in his published work "You Only Look Once: Unified, Real-Time Object Detection." This implementation is optimized for a fair balance amongst a number of different constraints: These include training time, resource requirements, and prediction confidence. According to Alexey, the YOLO v3 model will require approximately 4GB GPU-RAM. Conversely, other pre-trained models provided by Alexey's YOLO implementation have different requirements. While there is another model that requires 4GB GPU-RAM, called YOLO9000, this model detects and classifies 9000 different objects, as opposed to the standard model trained on the COCO dataset's 80 different objects.[4] Tiny YOLO is a more lightweight model that requires only 1GB GPU-RAM.

Model	Trial Image	Average Classifier Accuracies	Time	GPU-RAM Requirements
YOLO v3	giraffe.jpg	98%	44.57 seconds	4 GB
Tiny YOLO		53%	1.81 seconds	1 GB
YOLO v3	dog.jpg	97%	44.80 seconds	4 GB
Tiny YOLO		57.2% (42% for failed classification)	1.81 seconds	1 GB
YOLO v3	person.jpg	99.67%	47.92 seconds	4 GB
Tiny YOLO		90.25% (67.75% for failed classification)	1.83 seconds	1 GB

Table 13, Comparison of YOLO Training models

A comparison was made using the same input images to evaluate the accuracy vs. time spent when predicting classes, as seen in Table 13. One of the immediate, most noticeable differences between the two models is the time required to predict, with Tiny YOLO averaging just short of 2 seconds, and YOLO v3 averaging about 45 seconds. This puts Tiny YOLO at having a runtime that is about 25 times faster than that of YOLO v3. While these runtimes are measured without GPU processing – and thus are much slower than realistically what their run times would be if they were – the same 25-times difference can still be expected between the two.

Now where Tiny YOLO has a massive advantage for speed, YOLO v3 wins back that advantage with accuracy. YOLO v3 consistently performed at twice the accuracy of Tiny YOLO, and more importantly, never misclassified in any of its predictions. Tiny YOLO, on the other hand, performed relatively well, but with some glaring inaccuracies and mistakes. One glaring example of the shortcomings of Tiny YOLO's accuracy is in Figure 19. On the left image the zebra can be seen as totally selected by its bounding box. Tiny YOLO however makes the error of mistakenly missing the back half of the zebra, as it is partially obscured by the giraffe. Here, YOLO v3 would not have made that mistake because of the extra time it takes to reverify its objects of interest, and the higher accuracy threshold and lower error permissibility that it has trained to in its model.

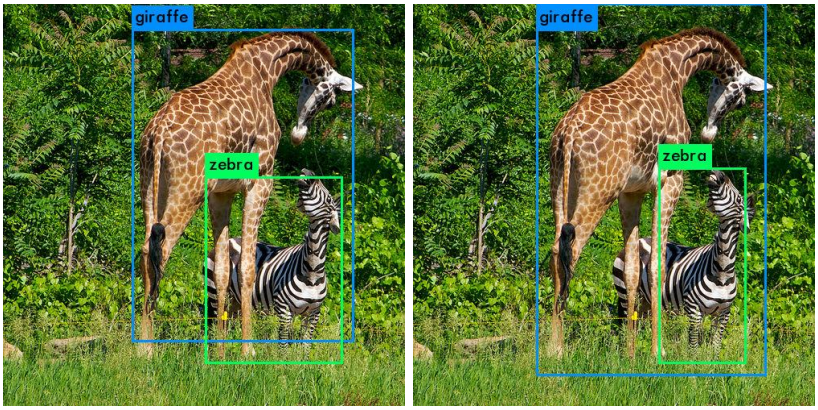


Figure 19, giraffe.jpg. Left: YOLOv3, Right: Tiny YOLO

Despite these inaccuracies however, Tiny YOLO will almost certainly find its use with RTDS. While YOLO v3 would certainly be preferable to use on the turret for its high accuracy, the time it requires to make predictions is extremely long and would certainly cause the turret to become dysfunctional. While preparing to use Tiny YOLO, however, this experiment will continue to help identify what shortcomings to expect from its implementation. Sometimes the drone might not be perfectly centered in the bounding box during any given video frame, or the algorithm loses sight temporarily of the bounding box of the drone. In these events, our supporting software will need to be prepared to handle these irregularities and make adjustments in real-time if we are to keep the drone in the center of the image feed.

6.2.4. SDSOC Environment Coding

The software-defined system-on-chip, or SDSoC, Environment is the toolset used by Xilinx for programming and developing on their products' embedded systems. It is important to understand the context of this environment so that we can make best use of its capabilities for our artificial intelligence and computer vision software.

6.2.4.1. DNNDK

The neural network must first be pruned before it can be loaded into the SDSOC environment by using the DECENT environment developed by Xilinx. The YOLO network must first be trained with selected images using the DarkNet compiler, and then

ported over to the DECENT optimizer, which will replace the floating-point weights into eight-bit unsigned integer weights, which will greatly optimize performance.

The project requires that the platform must be able to process the images, and the act on the returned information in relatively real time, with specification in communicating to a slave board, necessitating the use of the SDSOC + DNNDK solution, meaning that the binaries must be compiled on a Linux system with a minimum of thirty-two Gigabytes of RAM.

A secondary solution to the SDSOC compilation is using strictly Vivado block diagrams, running PetaLinux on the chip. The above solution requires the developer to appropriately connect all the ports of different blocks, and program the desired functionality in an embedded environment, which is not currently a desired path for the team.

6.2.4.2. Classifier Input, and Frame Referencing for Intelligent Tracking

The current design workflow involves the use of the ZCU104 development board as the main hub of information processing from the single camera sensor. The development board reads all the needed files to boot from an SD card on the board that is partitioned as FAT32. The boot files are generated by the previous DNNDK solution that is packaged with the SDSOC environment, and they are placed on the SD card once compiled. The compile times can take a very long time, since SDSOC must package the C++ files into RTL with respect to the ZCU104 chip, the ZU7EV, which has five hundred thousand logic elements. The routing algorithm will attempt to synthesize and implement the most optimized routing, using an algorithm similar to stochastic gradient descent.

The YOLO implementation currently used will classify the image, and then return the absolute coordinates of the drone in the image. The input image for YOLO is resized from the actual dimensions received by the 3.4 MP USB 3 See3CAM, however, the resizing operation will not impact the accuracy of the algorithm. The classification algorithm will be able to successfully classify drones from any angle, including birds' eye and person to zenith views. The training will successfully eliminate bias of classification by separating all birds eye, and ground views into equal categories in respect to the other possible view angles. Once classified, the system will have to start tracking the drone, by sending movement commands to the slave board that controls servos that turn the camera.

The slave board is communicating over Universal Asynchronous Receive Transceiver, using the RS232 protocol of one start bit, eight data bits, a parity bit, and a stop bit. The line is idle high. The agreed data packet gives information comprised of four word lengths, one for the direction that the platform believes the object to be moving in, one for the angle in that direction (always positive angles that are in between two direction types, i.e. if the direction was moving up and to the left, the angle would be from 0-90 degrees in the second quadrant of the viewport), and one for a velocity that the ZCU104 believes the drone to be moving at. The platform will also be able to tell the slave board if the drone is moving closer, by determining if the size of the drone in the pipelined picture is increasing or decreasing. The ZCU104 will be trained with specific drones in mind and will have the relative sizes of these drones to provide a naïve implementation of range finding.

The platform is only scanning an area in front of it, with a one-hundred-and-eighty-degree sweep and is normally in an idle state. When in the idle state, the platform will sweep in between this range, waiting for a classification.

The next state is the targeting and tracking phase. Once the platform has targeted a drone, it will attempt to discern movement of the drone, and calculate the necessary movement to fire on the drone. All targeting will be displayed over HDMI to a screen for monitoring. Once the platform has successfully targeted, and calculated movement needed to successfully "hit" the drone with a laser, it will attempt to discern if the laser hit the target.

During the hit detection phase, the platform will keep the laser on, and discern in the red channel if certain intensities have increased in the histogram. Once a certain spike is detected, the platform will count this as a successful hit. If no spikes are seen in the red channel when compared to the previous histogram, the drone will declare a miss.

The platform will continue to attempt to fire at the drone until it leaves the Region of Interest for the drone, which is a one-hundred-and-eighty-degree sweep in front of it, and a ninety degree to negative ninety degree sweep vertically, assuming that the platform is at zero degree point vertically. The variable for vertical placement can be trimmed as needed if the platform is not elevated.

For determining the velocity of the drone, one possible method being reviewed is determining the absolute pixel location of the drone in a previous frame, and then finding the new pixel location in the current frame. Knowing the Frames Per Second of the camera, and classification pipeline (~30FPS), the system can determine the amount of time passed in between the pixel movement, and calculate a rudimentary velocity vector, based on the distance between the two pixels. Additionally, the bounding boxes can again be used rudimentarily to create a size comparison, to understand if the drone is moving parallel, orthogonally, or in any other direction. The naïve implementation described above will most likely be a proof of concept and will be used unless the secondary research item bears fruit.

The second possible way of determining the movement of a certain object is through the use of optical flow. Optical flow allows for the detection of the motion of objects by displaying the distribution of through color. The current proposed approach is to wait for movement into the detection state. When in idle, the optical flow will not be computed, thus eliminating unnecessary computations and energy draw. When a target has entered frame, the platform should compute the optical flow, only for the Region of Interest that the drone is currently thought to be in. Since the coordinates are absolute from the YOLO algorithm, the image fed to the optical flow module needs to be the same size.

6.2.4.3. Targeting System Distance and Range Finding

The RTDS must be able to track the target from frame to frame, enabling it to communicate with its slave board in relatively real time. To ensure that the target is within the proper frame of the camera screen, the board need to use an appropriate method of determining velocity of an object per frame. The chosen way for approximating movement from frame to frame is using an implementation of a type of motion detector in conjunction with the YOLO implementation to define the regions of interest for the system, allowing the system to focus on the target's motion.

6.2.4.3.1. Optical Flow

Optical flow is a method of motion estimation based on the change of the pixels in the image. The method generally used minimizes color difference between pixels in the image, and calculates overlapping regions either through a window based, or patch-based approach. The optic flow allows for the generation of relative velocities between an observer and a scene. Due to the stationary aspect of the platform, we can assume that all velocity and motion in the scene is independent of the viewing source.

The principle use of the optical flow algorithm is to determine what pixels have moved in from one frame to another. Therefore, using the YOLO algorithm to define a region of interest, a certain number of regions must be stored in order to generate a comparison of the drone as it moves. The algorithm can be used to estimate the velocity of the pixel, which will be correlated to a vector that will be utilized by the development board to send commands to move the camera.

A weakness of optical flow is with illuminations. Illuminating the image will cause a change in the way the optical flow will detect the velocities of the objects, and thus, there must be constraints on the lighting environment that the drone is flying in. Normalizing lighting will increase the accuracy of the optical flow algorithm, allowing the system to be more accurate.[13, 25]

6.2.4.3.2. Affine Motion Tracking

The affine tracking algorithm attempts to identify patches of pixels in a frame and track those pixels in the sequence of images. The motion tracking dubs its name from using affine machine parameters over an optical flow field.

6.2.4.3.3. Kalman filter

Kalman filter, or linear quadratic estimation, uses measurements observed over time, and produces estimates of unknown variables. The unknown variables in the platform's case would be the velocity and distance away from the platform. The filter will keep track of the system and variance of the estimate, corresponding to the unknown parameters.

The Kalman filter is utilized in tracking algorithms to track objects in a video stream. A large amount of research has been done in regards

6.2.4.3.4. Comparisons

Due to the high computational task of Optical Flow, it is necessary to already have each frame already computed, with the previous frame stored in memory as reference to create the frame for the next iteration. The proposed workflow for the software model is to have a Dense Optical Flow frame created, and then using the values from the YOLO model's detection coordinates to estimate velocity of objects in the subset of the image frame. Thus, both the YOLO module and the Dense Optical Flow module must be created in parallel, with an output stage necessitating that both modules have completed for the given frame, allowing for prediction of the velocity of target(s) in the frame. The data path can be pipelined with only a finite framed FIFO at the end of the structure.

Due to the extensive research found with velocity approximation using optical flow, the current approach for velocity approximation and vector direction of the drone will be using OpenCV's optical flow function.

6.3. System Housing

The housing for the system will have to minimally enclose the development board, the created Printed Circuit Board, and a platform to turn the camera. The current platform will require a minimum of two servos to turn the camera in an area of one-hundred-and-eighty degrees vertically and horizontally. Additionally, the platform will need to holster a laser of some type, which is connected directly to the ZCU104.

The housing will need to allow for access of the development board, and student generated boards ports, while still allowing for the appropriate airflow to ensure the longevity of highly utilized chips (the Zynq ZU7EV chip).

The housing will need to focus on ease of portability, and stability. It is desirable to allow users to easily deploy the RTDS to different locations, and in maintaining a stable point to fire possible armament that might be added to RTDS later.

6.3.1. Servo Specific Housing

The housing unit needs to specifically account for the rotating structure that will be turning the camera within its target space. The rotation is one-hundred-and-eighty degrees, vertically and horizontally, meaning that significant thought must be placed in the cutout room for all servos. Additionally, wiring constraints need to be met so that a thoughtful spacing minimizes the length of the wires when connecting to the student developed slave board.

6.3.2. Development Board Specific Housing

The housing must be large enough to accommodate the target development board, student made slave board, and provide accessibility to all mandatory connections. The current project constraints allow for the board to be powered from a wall outlet, therefore the power outlet of the ZCU104, and the slave board must be exposed in the created housing.

It is plausible for the housing to only enclose the servos and provide a structure that is not congruent to the development and slave boards sizes. This configuration would allow for a more modular approach of placing different sensor constructs next to the processing constructs. The project will most likely take a modular approach in development of the housing required for the boards, as airflow requirements needed for cooling, and required connections will surely differ from the slave board and the ZCU104.

6.3.3. Material Type

The housing material will most likely be 3D printed plastic, created using UCF's on campus 3D printer, or another available 3D printing resource. The model will be created using any CAD software compatible for exporting 3D printable files and will be designed in respect to sizing constraints of the development board, slave board, and servos.

6.4. System Testing

The main objective of the system is to successfully target and hit a drone. The proof of concept implementation uses a laser due to alleviating constraints necessary when determining drop of a projectile, and calculating offset needed to hit a moving object. Currently, testing of the system will be done with the following cases: movement type tests, object type tests, and view type tests.

The focus of each test type is latency, accuracy, and repeatability. It is imperative that the platform can quickly determine a target and track its movements. Arguably, accuracy of what type of target is being tracked is needed to prove the platform's capabilities. Finally, experiments will be repeated with as close to the same conditions as possible.

6.4.1. Lateral Movement Tests (Required)

The platform will need to be quick enough to track a drone within its window, and then fire/tabulate its results. To achieve the previous requirement, the platform will need to relay direction and speed variables to the slave student developed board in order to move the main sensor (the 3.0 USB SEE3CAM camera). The test will ensure that the platform will be able to span across its view space within the specified constraints for maintaining the maximum drone detection speed.

The test is comprised of four stages, a low speed test, a half-way speed test, a maximum speed test, and an above maximum detection speed. The current confidence threshold is not yet known and will be obtained experimentally during the trial period of the project. The stages are self-explanatory, the platform should easily detect a low speed or idle drone, be able to keep track with a half/full speed drone, but expectedly fail when the drone moves quicker than the platform can track. The confidence intervals will be obtained based on distance from the drone as well, as objects that are closer will be difficult to track at higher speeds. The tests success is determined by successful hits of the platform. Each passable test must at least "hit" the drone at least once, and all data for the tests should be saved by the system for ease of tabulation when quantifying the success/failure rates of the RTDS performance.

6.4.2. Vertical Movement Tests (Required)

The platform will need to track a drone as it moves vertically in its view window and be able to fire/tabulate its results. To achieve the previous requirement, the platform will need to be able to detect the bottom, or top of the drone, and be able to form a relationship between the changing images, and its velocity as it progresses.

The test will be comprised as a permutation of two test parameters, viewpoint of the drone, and the velocity that the drone is movement. The max velocity and minimum detected velocity will all be obtained experimentally near the final portion of the project. The view point tests consist of only bottom of drone (descending motion incomplete), only top of drone (ascending motion incomplete), movement from top to bottom (ascending motion complete), and movement from bottom to top (descending motion complete). The speeds that will be tested include low, moderate, high, and above threshold. The tests objective is to acquire the accuracy that the platform will have for each test, with a pass being at least

one successful “hit” from the on-board laser during each of the tests that should be passable, based off or previously gained experimental data. All data will be saved onto the SD card for evaluation purposes.

6.4.3. Lateral Movement Tests (Required)

The platform will need to be able to detect drones that are moving in angles across the viewing boundary, and fire/tabulate. Meeting the previous requirement necessitates that the platform can determine the angle of motion that the target is moving in, allowing the camera to move in an angle across the viewpoint. The platform must be able to successfully determine the direction and movement speed of the drone, and relay that information quickly to the slave board.

The tests will be comprised of the classical trigonometric paths, moving in thirty-degree segments, from zero to three hundred and sixty degrees. The output from the platform will be monitored from each test, with the angle parameter being of highest interest to the tester. The angle should be within two degrees of the attempted flight path, and all such flight paths will be found experimentally when the project proceeds to test stages.

Velocity is the second parameter for each of the tests, with four different levels of velocity tested from slow to idle, moderate, max speed, and above threshold testing. To pass, the platform must perform at least one successful hit for all passable test conditions.

6.4.4. Approaching/Fleeing Tests (Nice to Have)

(optional requirement) The platform should be able to track an image that is moving directly forward or backward in respect to the position of the platform, meaning that the image of the drone would enlarge or minimize as time progressed. To achieve the previous test condition success, the platform must be able to form a relationship between the initial detected size of the object, and its continued decrease or increase in size.

One possible way to achieve the above test is to use the bounding boxes calculated by the platform and determining if the area of the box is increasing or decreasing per frame. The tests will consist of two parameters, the side of the drone, and the speed at which the drone is approaching of fleeing. Successfully completing the test should have the platform identify that the target is only moving away or towards the sensor, and the approximate velocity of the movement. All the speeds will be experimentally acquired during the test phase of the project. A pass also consists of the drone successfully hitting the drone at least one time. All results will be saved to a file.

6.4.5. Object Type Tests (Required)

The platform should be able to differentiate between different types of objects and show with high accuracy that it can detect the generic drone object. Having high specificity is imperative to the project’s success and is heavily correlated to the efficiency of the trained Convolutional Neural Network. The platform at a minimum should be able to specify any type of drone versus a plane, for example.

To test the differentiability of the platform, multiple objects will be placed in a frame next to the drone. Initial tests will have the drone be stationary. The platform must successfully

target only the drone in frame and acquire at least one hit to pass this test. The types of objects will currently include other types of flying objects including, birds, planes, helicopters, tree movement, and thrown objects.

6.4.6. Drone Type Tests (Nice to Have)

The platform should be trained for multiple types of drones and be able to differentiate based on the type of drone. Sub classing the drone types would allow for finer tuning of the previous tracking algorithms, by providing size constraints to the object in view, and then find out the distance away from the sensor based on experimentally gained data using the SEE3CAM USB camera.

To test the above feature, multiple different drones should be presented to the platform, with each drone that has been accounted for accurately being placed in its classification bin. A generic bin will be used and trained for in case drones do not fall into one of the specified types. The types will be chosen based on current drone popularity. In addition to successfully sub classing the type of drone, the platform should score at least one successful hit on the drone using its modified parameters due to the type of drone in frame.

6.4.7. High Accuracy Occlusion Detection (Nice to Have)

The platform should be able to detect an object that is partially not visible due to occlusion by another object. The ability to detect portions of a drone allow for quicker recognition of movement of a drone, or the ability to target a drone attempting to hide behind a material. The ability to detect partially occluded object falls on the training of the neural network, meaning that the machine learning architecture must be exposed to a multitude of images that are partially occluded.

To test the above functionality, a drone will be applied at differing percentages of its size behind an occluding material, like a box. The platform should be able to detect the drone up to half of the drone's size. Additionally, once targeted, the platform should "hit" the drone a minimum of one time.

After immobile tests are successfully passed, the platform will then need to be able to detect the drone as it passes behind objects and maintain a lock. The previous concept can be tested by having the drone pass into the field of view, then pass partially behind a blocking object, like a box. The drone should maintain a lock and be able to target the drone up to the fifty percent mark of occlusion.

6.4.8. Drone size in Relation to Distance and Accuracy (Required)

The RTDS turret must establish a minimum and maximum distance at which a target can be locked on to. The above principle depends entirely on the minimum size of images that were trained on the system. The size of the drone also plays a factor into the above concept, as a smaller drone, will be more difficult to detect than a larger drone at the same distance.

The first approach to accomplish this requirement would be to include more training images that match the relative size of a drone viewed from a distance. While this will certainly help the RTDS identify distant drones to a degree, eventually the resolution of

our camera will no longer be able to differentiate the speck that is a drone in its sights and any other object in the sky such as a bird or plane.

The second approach is to us larger-sized drones in general in order to train and verify the capabilities of the RTDS. While it is unrealistic to assume that all intruding drones that we would wish to neutralize would be of a certain size – let alone that size being larger than average – it might still be safe to assume that all malicious drones would be of at least a certain size, meaning that they would need to be carrying some amount of payload, be that a camera, or a weapon, etcetera.

The above concept will be tested using two drones of different sizes, tested at ranges increasing by ten-foot intervals. The distance of which a lock can not be required will be verified experimentally during the testing portion of the project. A successful pass will be the platform detecting the drone within the confidence interval specified in the constraints of the design.

Hits will not be required for test passing, since the interval of the histogram spike will decrease significantly as the distance away from the drone increases. The lack of confidence in successful hits will attribute to multiple misses at long range. The above artifact is the driving reason for enabling accurate range finding methodologies, so that the platform will determine if it can successfully determine a strike occurred.

6.4.9. Multiple Target Detection (Nice to Have)

The platform should be able to target multiple drones in frame and be able to sequentially place the drones in order of detection. The above constraint is difficult due to needing to allocate a relationship between the initial detection, and correctly identifying that specified drone in the next frame.

One possible solution to the above problem is to look at the overlap of bounding boxes classified by the object detector, and then making predictions based on the percent overlap. This implementation is naïve, however, since closely grouped drone groups will most likely make this method fail.

Testing the platform for multiple detection is simplistic in that each test case will simply depend on two variables: the number of drones in the detection region, and the density of drones in each region. The test cases will exist as a permutation of these two variables, testing the distance between the drones that facilitates mistakes in the trained network, and the number of drones before the communication between the slave board and master development board generates errors.

All successful passes must detect the max number of detectable drones and provide a target sequential list that goes up to the max detection limit. The max number will be found experimentally or set during the writing of software.

6.4.10. Priority Targets (Nice to Have)

The platform could possibly evaluate targets in frame based on associating priority weights to different types of drones. The ability to target multiple drones, and the classification of different types of drones would need to be implement before this type of structure could be implemented and or tested.

Additionally, as described previously, target priority could take advantage of the proximity-detection capabilities of the RTDS and identify which of all the drones is determined to be the closest or furthest. Priority targets based off this quality would have a higher chance of being marked or hit by the turret, as a closer target naturally would have less interference between itself and the turret.

Testing this type of functionality would rely entirely on the creation of a bias after YOLO has classified all the objects in frame, and then begin tracking the type with the highest bias attributed to it. One type of test includes using two different drones starting near each other, then moving in opposite directions. The platform should track the drone with higher priority.

Furthermore, multiple different drones could be placed in the view of the platform, and each should leave the frame of view. The tracking should drop from the hierarchy as specified in the code, ensuring that as the highest priority left, the second highest would be tracked, etcetera.

7. Administration

7.1. Estimated Project Budget and Financing

Component	Price	Status
ZCU104 Development board	$895	Acquired
Parrot AR.Drone 2.0 Elite Edition	$120	Acquired
Servo Motors	$20	Acquired
Power Source	$50	Research Stage
Weapon System	$50	Acquired
Camera System	$50	Acquired
Miscellaneous Parts	$50	Research Stage

Table 14: Budget

Tentative Total: $1235

7.2. Initial Project Milestone

7.2.1. Senior Design I

Task	Due Date
EEL4914 Initial Project Document - Divide and Conquer	September 20th
Initial Project Research	September 30th
Updated Divide and Conquer document (D&C V2)	October 4th
Design Research	October 10th
New Assignment on Standards	October 25th
60-page Draft Senior Design I Documentation	November 1st
100-page submission	November 15th
Final Document Due	December 2nd

Table 15: Milestones I

7.2.2. Senior Design II

Task	Due Date
Order PCB	TBD
Parts Check/Order Parts	TBD
Hardware and Software check	TBD
Assemble Prototype	TBD
Test Final Product	TBD
Final Document Due	TBD
Final Presentation	TBD

Table 16: Milestones II

8. Conclusion

The Reactionary Targeting Defense System seeks to combine the technology of object recognition in the field of machine learning along with robotics integration. This will be done in order to create a proof-of-concept device that will be capable of identifying and tracking flying quad-copters (or simply drones) in the nearby vicinity, and ultimately tagging those drones with a laser marker.

By accomplishing this, the RTDS will demonstrate its capability to create an area of denial to drones in the nearby air-space. Future interpretations of our design could very well implement larger hardware and more robust software solutions in order to increase tracking speed and distance, and identification processing speed. Additionally, following proper and appropriate laws regarding the matter, future designs could replace the simple laser marker with full-scale neutralizing equipment. Examples of these kinds of devices include radio-frequency jammers and high-frequency lasers, as well as projectiles such as nets or actual rocketry or bullets. However, being that neutralizing weapons such as frequency jammers are highly illegal for many reasons by the FAA, we are refraining from any type of weaponization of our project.

The RTDS is comprised of two different, major components: The development board for processing image data using artificial intelligence which will send commands to the printed circuit board with our microcontroller. The second more important component is our printed circuit board with its embedded microcontroller. It will be responsible for controlling the servos which will enable the pan and tilt functionality of our sensor system. The printed circuit board will also be responsible for powering the laser which will be attached to the sensor system.

The RTDS is also comprised of three different and distinct software components: The software controlling the servos and their positioning; The software controlling the tracking algorithm that returns data to the servo-controller; And the artificial intelligence algorithm that receives the video feed and will return where it has identified any visible drones.

All the components comprising both our hardware and software have all been rigorously compared against all relevant alternatives, with those being used being the optimal choices given our constraints and target requirements.

All in all, by completing this project we will not only meet the requirements to pass our senior design project we will also learn a lot about the functionality and implementation of computer vision and machine learning techniques. Our project shall tackle a real-world issue regarding privacy and safety of people in regard to the ever-emerging technology of drones, whether autonomous or remotely controlled.

The design is to be implemented during the Spring 2020 semester, at the end of which the RTDS will have been constructed, tested, and ready for demonstration.

A) Appendix

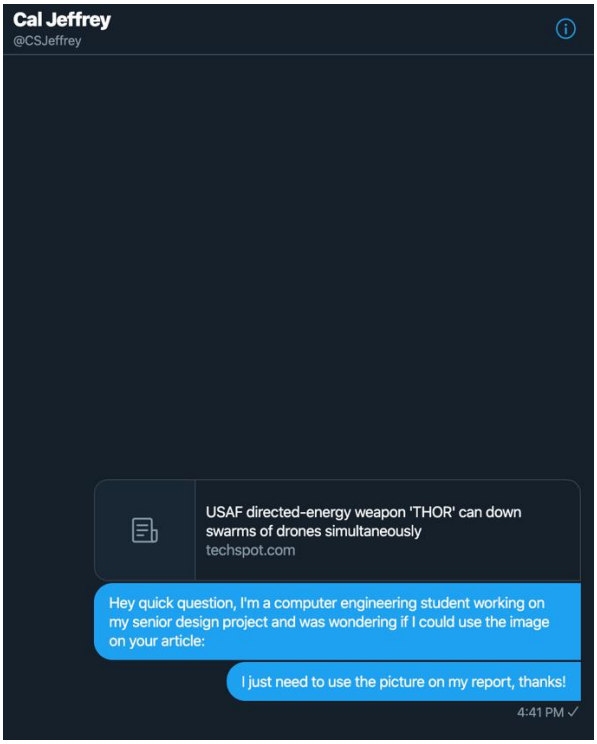
A1) Reference

1. "About". Opencv.Org, 2019, https://opencv.org/about/.
2. Alexey. “AlexeyAB/Darknet.” GitHub, 26 Oct. 2019, https://github.com/AlexeyAB/darknet.
3. Chuan-en Lin. “Chuanenlin/Drone-Net.” GitHub, 25 May 2019, https://github.com/chuanenlin/drone-net.
4. "COCO - Common Objects In Context". Cocodataset.Org, 2019, http://cocodataset.org/#home. Accessed 26 Oct 2019.
5. "Does The Dataset Size Influence A Machine Learning Algorithm?". Stack Overflow, 2019, https://stackoverflow.com/a/25670125. Accessed 15 Nov 2019.
6. Dwivedi, Divyansh. “Face Detection For Beginners.” Medium, Towards Data Science, 27 Mar. 2019, https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9.
7. Elizabeth, Jane. “Java Is One of the Most Energy-Efficient Languages, Python among Least Energy Efficient.” JAXenter, 2 Oct. 2017, https://jaxenter.com/energy-efficient-programming-languages-137264.html.
8. "How Many Images (Minimum) Should Be There In Each Classes For Training YOLO?". Stack Overflow, 2019, https://stackoverflow.com/a/55367914. Accessed 15 Nov 2019.
9. I2C BUS," [Online]. Available: https://www.i2c-bus.org/multimaster/. [Accessed 9 April 2019].
10. "I2C info," [Online]. Available: https://i2c.info/. [Accessed 3 April 2019].
11. “Image Processing and Computer Vision - MATLAB & Simulink Solutions.” MATLAB & Simulink Solutions - MATLAB & Simulink, https://www.mathworks.com/solutions/image-video-processing.html.
12. “Jetson Nano Developer Kit.” NVIDIA Developer, https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
13. Kaehler, Adrian, and Gary Bradski. “Learning OpenCV 3.” - O'Reilly Media, Dec. 2016, http://shop.oreilly.com/product/0636920044765.do.
14. M. Grusin, "sparkfun.com," [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all. [Accessed 31 March 2019].
15. "Opencv: Introduction". Docs.Opencv.Org, 2019, https://docs.opencv.org/4.1.1/d1/dfb/intro.html.
16. “What Is the Standard PCB Thickness?” *Tempo*, 11 Apr. 2019, https://www.tempoautomation.com/blog/what-is-the-standard-pcb-thickness/.
17. PC, "IPC," November 2017. [Online]. Available: http://www.ipc.org/4.0_Knowledge/4.1_Standards/PCBA-Checklist18.pdf. [Accessed 18 October 2019].

18. Pereira, Rui, et al. “Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?” Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering - SLE 2017, 2017, doi:10.1145/3136014.3136031.
19. Redmon, Joseph, et al. “You Only Look Once: Unified, Real-Time Object Detection.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, doi:10.1109/cvpr.2016.91.
20. Redmon, Joseph. Darknet: Open Source Neural Networks in C, https://pjreddie.com/darknet/.
21. Rosebrock, Adrian. "My Top 9 Favorite Python Libraries For Building Image Search Engines - Pyimagesearch". Pyimagesearch, 2019, https://www.pyimagesearch.com/2014/01/12/my-top-9-favorite-python-libraries-for-building-image-search-engines/.
22. "Sightmachine/Simplecv". Github, 2019, https://github.com/sightmachine/SimpleCV.
23. "Simplecv". Simplecv.Org, 2019, http://simplecv.org/.
24. Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) https://doi.org/10.7717/peerj.453
25. Szeliski, Richard. Computer Vision: Algorithms and Applications, Sept. 2010, http://szeliski.org/Book/.
26. “Ultra96.” 96Boards, https://www.96boards.org/product/ultra96/.
27. “Xilinx Zynq UltraScale MPSoC ZCU102 Evaluation Kit.” Xilinx, https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#hardware.
28. “Xilinx Zynq UltraScale MPSoC ZCU104 Evaluation Kit.” Xilinx, https://www.xilinx.com/products/boards-and-kits/zcu104.html.

A2) Copyright

a) Figure 4:



b) Table 11:

Energy Efficiency across Programming Languages

How Does Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SLE'17, October 23–24, 2017, Vancouver, BC, Canada
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5525-4/17/10…$15.00
https://doi.org/10.1145/3136014.3136031

c) *Figure 10:*

Image used courtesy of Greg Borenstein via Creative Commons license.
https://www.flickr.com/photos/unavoidablegrain/6884354914
https://creativecommons.org/licenses/by-nc-sa/4.0/

d) *Figure 11, Figure 12:*

Courtesy of Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi.
arXiv:1506.02640 [cs.CV]

e) *Figure 13:*

Image used courtesy of AdrienNK via Creative Commons license.
https://stackoverflow.com/a/25670125
https://creativecommons.org/licenses/by-sa/4.0/

f) *Figure 16:*

Image provided for non-commercial use courtesy of StickPNG.com.
https://www.stickpng.com/img/electronics/drones/custom-drone

g) *Figure 17:*

Image provided for non-commercial use courtesy of JeongGuHyeok.
https://www.pngfans.com/middle-6b8dc81a04866432-phantom-drone.html