Autonomous Turret

EEL 4914
Group 17
Fall 2010

Hector Colon
Adam Horton
Kyle Steighner
Nicholas Yielding

Table of Contents

Table of Figures

Table of Tables

Section 1: Introduction / Definition

1.1 Executive Summary

In today's age, one cannot go through a typical day without coming into contact with some form of electrical device to help in their daily life. Electrical Engineering and Computer Engineering is a vital importance to everyday life whether it is a laptop computer in one of the wealthiest countries or a pay as you go phone in one of the poorest countries in the world. The point of this realization is that there is a need for the specialized knowledge and experience of these engineers to design and test systems that will provide a service or usefulness to the human race. One of the most necessary services in the world is defense, more specifically military defense. The U.S. DoD budget for 2010 fiscal year was $663.8 billion this was the second largest expense of the U.S. budget behind social security spending. An image of the 2009 net cost spending is pictured in figure 1. This is why our group is proposing a project that is relatable to a military device.

There are military applications of automated turrets and autonomous turrets. They can protect groups of people or assets of the military that require protection that is quick, accurate, precise, and difficult to trick. One example of the application of autonomous turrets/automatic turrets is the use of the systems on aircraft carriers in defense if the carrier is being shot at with AAM or ASM. This is just one example of the numerous uses of these complex turret systems in military defense.

Our group is comprised of two electrical engineering students and two computer engineers. The two computer engineering students are Adam Horton and Hector Colon. The two electrical engineering students are Nicholas Yielding and Kyle Steighner. Nicholas and Kyle have backgrounds in both military operations and technology. Adam, Hector, and Nicholas have experience working with robotic systems and Nicholas is presently a member of the UCF Robotics Club. From our specialized knowledge, we came up with the idea of creating a robotic autonomous turret that is heavily dependent on both hardware and software components.

Our group is proposing to develop a prototype of an autonomous turret. One of the main goals of our project is the turret to function as a simulation to a real defense turret. We are proposing a turret based weapons system capable of being operated on the system's own judgment of its surroundings and environment. Our system will be heavily dependent on both hardware and software accuracy. We want to design the defense system with the highest accuracy and functionality that is possible with the lowest cost possible of

components that make up the system. We have concluded the most important specifications of the system will be detecting, tracking, and targeting of the system.

The system will also have an off-board server included in the design of the system to record engagement history and will also put a snapshot of what the turret sees included in the engagement history, position, and live feed from the system. The server will also allow for manual control of the turret for situations where manual override is desirable. The server will host a web application that will allow users to access all of its services remotely.

The turret that is included in the system will be a paintball or airsoft gun that will simulate the real effect of being hit by a military defense turret. The autonomous turret will stand on a base; the base will be tall enough for the turret to tilt between -45 degrees below the horizon to 45 degrees above the horizon. The base will have a low profile such that the system does not tip over. The turret will run on a mixture of a 12 volt external battery and various internal batteries that will power the PC, the three servo motors, and the microcontroller that comprise the autonomous turret.

When the turret system is not connected to a wall outlet (120 nominal volts) the battery life will be greater than 30 minutes in duration. Two servo motors will control the pan and tilt of the turret and the third will directly control the firing of the system by controlling when the trigger is pulled. The turret system will be able to maneuver a yaw of 180 degrees by one of the three servo motors and will employ web cameras for the field of vision. The maximum range that the autonomous turret will be able to detect an intruder is 100 feet. The accuracy of the range finder will be within +/- 6 inches. When an intruder is within the warning range of the turret there will be an audible warning. If the intruder breaches the firing range of the turret the intruder will be fired upon. The turret system will be connected to a PC for data acquisition, processing of the live feed from the web-cameras that will be placed in use, and connections to the server that we will be utilizing. The delay in the image processing for the turret system will be less than 100 ms to provide the best accuracy and precision. The response time of target acquisition and appropriate response will be less than 2 seconds.

The turret system will allow customizable perimeters for the audible warning and firing up to our maximum range that is supported. The server that will support our autonomous turret system will support multiple turrets but for our prototype we will only be utilizing one. Our server will also provide real time radar for the connected turret thus to increase the overall utilization of the turret. It will also allow for the manual control of the turret system along with full control of the system. There will also be a security portion to our autonomous turret system. If there is a necessity in the case that someone has accidentally entered the range of the turrets' range then we will have a wireless shutoff switch routine. We want our turret to be open-ended when we are completed in the design of the

prototype such that there is more room for improvement in a second version prototype, decreasing the percentage of error, and increasing the amount of accuracy.

1.2 Motivation

The motivation behind our system came mainly from the specialized expertise in the field of robotics and computer/electrical engineering. Our group is comprised of two computer engineering students Hector Colon and Adam Horton. Both of these group members have superb experience in software, computer, and electrical engineering. Adam also has experience in developing web applications and online databases. The other two group members are electrical engineering students Kyle Steighner and Nicholas Yielding. Both of these members are experienced and have specialized knowledge in electrical engineering backgrounds and Nicholas has continued knowledge of robotics. An autonomous robot was a perfect project since it incorporates equal parts of electrical engineering and software development. The motivation of creating an autonomous turret as opposed to another form of robot was, as previously mentioned, the amount of spending on defense for the U.S. In the 2009 fiscal year spending on defense was close to a trillion dollars. This amount of spending is realized by **Figure 1** below. This is why our group proposed a robotic oriented autonomous turret that incorporates the components of software, computer, and electrical engineering.

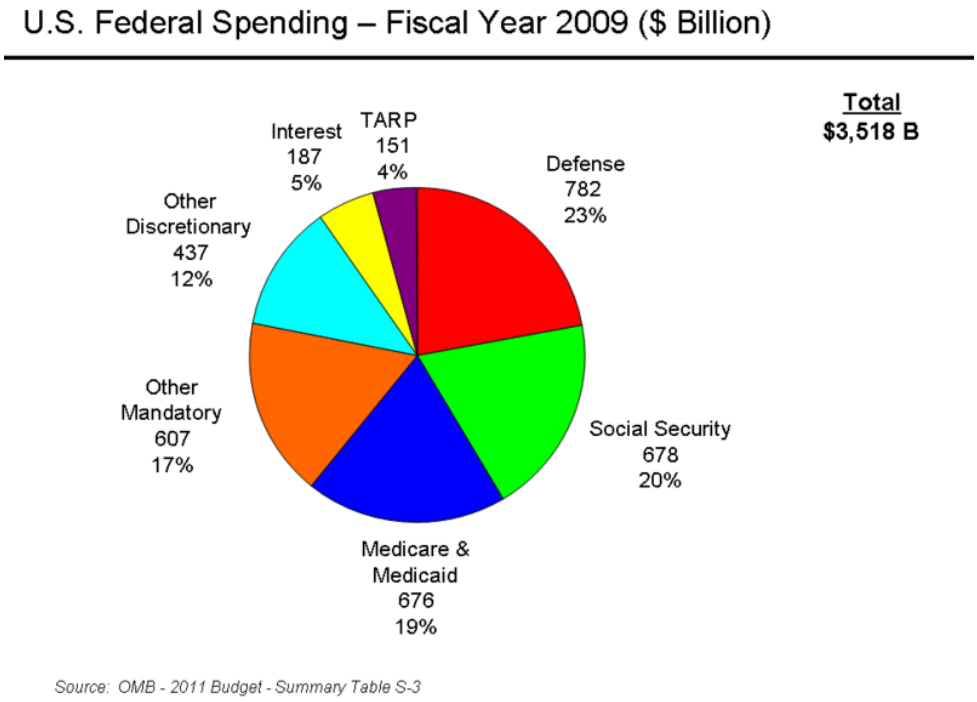

Figure 1. 2009 net cost by category for the United States
Reprinted under GNU Free Document License

1.3 Objectives

There are several objectives our group must accomplish in order for the autonomous turret to be considered a success. This is a list of the basic functionality of the turret system:

- Sense a moving target approaching the turret via the web-cameras and range finder/sensor and send it to a PC for processing.
- Respond to threats and targets appropriately and take action based upon the set parameters given by our group.
- The system is required to be set up easily and quickly. The time it takes from when the base is set and the PC is powered up to when the turret is fully operational should be as short as possible.
- The server system that allows manual operation of the turret must be wirelessly controlled such that it is a separate system from the autonomous turret system but is able to be turned on or off to prove that it is truly an autonomous turret.
- The web application must be accessible from any internet connected PC and be able to access engagement history and radar.
- The system must employ appropriate safety features such that even though this is a simulation system no one will be harmed if something unexpected occurs.
- The turret system excluding the server components shall be small enough such that it can be carried by one person.
- The gun mounting should be able to support a variety of paintball and airsoft guns.

1.4 Specifications

The objectives in our project would be easy to meet if we didn't have any baseline set for performance. The turret should be practical and useful in real world situations so there are several hardware and software specifications we would like to meet:

1.4.1 Hardware Specifications

- The turret weighs approximately 20 lbs.
- The turret is 17" tall.
- The turret is able to fire upon targets 30 feet away.
- The turret is able to determine the distance of targets 60 feet away.
- The battery lasts 5 hours on a full charge.
- The battery is a commercial off the shelf (COTS) 12 volt battery that powers the system

- The rest of the system components are powered by their own internal batteries
- The 12 volt lead-acid battery is rechargeable and has a capacity of at least 8000 mAh
- The rangefinder has a range of at least 75 feet, with an accuracy of ± 3 mm
- The microcontroller provides PWM for at least 3 servo motors
- The webcams are COTS available and provide progressive scan video of resolution greater than or equal to 640 x 480 pixels
- The audible alarm is louder than 50 dB in sound

1.4.2 Software Specifications

- The motion detection is able to detect motion of at least 3 ft/s at 40 feet away
- The turret tracking is able to follow targets moving 5 ft/s at 30 feet away.
- The engagement history database is able to store and display at least 100 prior engagements
- The web application works on all modern browsers such as Mozilla Firefox and Google Chrome
- The manual control of the web application does not have more than 1 second of latency between getting a command and executing it.

1.5 Roles and Responsibilities

1.5.1 Division of Labor

Since there are four members in our group it would be unwise if the group did one topic at a time, thus we set out on our own to do separate parts of our project and will piece all of our work together when the time comes. There are 2 Electrical Engineering students in our group and 2 Computer Engineering students in our group. This is an excellent combination for a group of 4 members max. Where there is equal division of research and design of the electrical and computer components of our autonomous turret. Although we are subdividing the project into parts, when a group member comes upon an issue with a component and he does not know how to resolve it the group will convene and look for solutions. We will describe who is responsible for each component of the turret and then there will figures following these descriptions showing a simple overview of what is expected.

Hector's main responsibility in construction of the prototype will be centered on the development of firing calculations, parameter detection, and calculation of target distance. Both Hector and Adam will research the topic on motion detection to apply it to the prototype. Most of the components that Hector is responsible for are software oriented but also include some hardware

components. The firing calculations will be conducted on the laptop provided by Nick and will be done in a compiler program to tell the turret where to fire, when to fire, etc. and will do so via the microcontroller. The parameter detection and calculation of target distance will include hardware and software components. We will use webcams to visually detect objects moving in the field of vision but it will be the software that calculates that an object is moving and tracks it. The calculation of target distance will be done by the range finder that we have acquired and the data will be read in by the computer and used appropriately.

Adam's main responsibility in construction of the prototype will be centered on manual operation of the turret, and server development of components. The development of the server components is important to our prototype in the essence that it tests our wireless connectivity from the server to our turret and accurately and precisely does what is expected. Also, the manual operation of the turret via the server is an excellent test for the prototype. The manual operation of the turret will be a separate system such that our turret stays truly autonomous. The manual operation of the turret will be at a terminal and will switch over to manual operation when the operator or user decides to. Adam has also acquired much of the hardware that is required for the software systems to operate on so this is another major responsibility on his part to make sure every component is working.

Kyle's main responsibility in construction of the prototype will be around the manufacturing of the base, and servo's. This responsibility that he is taking on is a majorly crucial part of the construction. Without the stability and correct mounting of the components that make up the base the prototype would be doomed. That is why he will verify that the torque and force requirements of the components causing movement or resistance are in check. He is responsible for attaining the mechanical machinery required to move the base and/or turret. He will also be responsible for the electronic system that makes up the base. He is also responsible for the mounting of the servos to the base. One of the small side responsibilities that he is also responsible for is maintaining a spreadsheet of the budgeting of the prototype.

Nick's main responsibility in construction of the prototype will be around the PCB/Microcontroller, providing the network router, and providing the laptop for the operation of the autonomous turret. These tasks that he is taking on are also extremely important to our prototype. His tasks are extremely important to the electrical aspect to the entire prototype system. He is taking on the task of which Microcontroller our group will choose and he is providing his laptop as well as network routers for the prototype to use. He will be providing the wireless routers but the turret will be using a wireless transponder to communicate across the network to the server and laptop. He will also be providing his laptop for use of the control of the autonomous turret. He will not be required to create the software that will be controlling the turret only provide the computer for use.

So our prototype really consists of 3 major components the weapon (paintball/airsoft rifle), the electronic control apparatus (laptop), and the network (server). Most of the components will be acquired in the spring of 2011. For the components of our system that we have already acquired we will working on integrating them together in preparation of the second semester of senior design II. A high level visual diagram of our system with responsibilities linked is illustrated below in **Figure 2**.

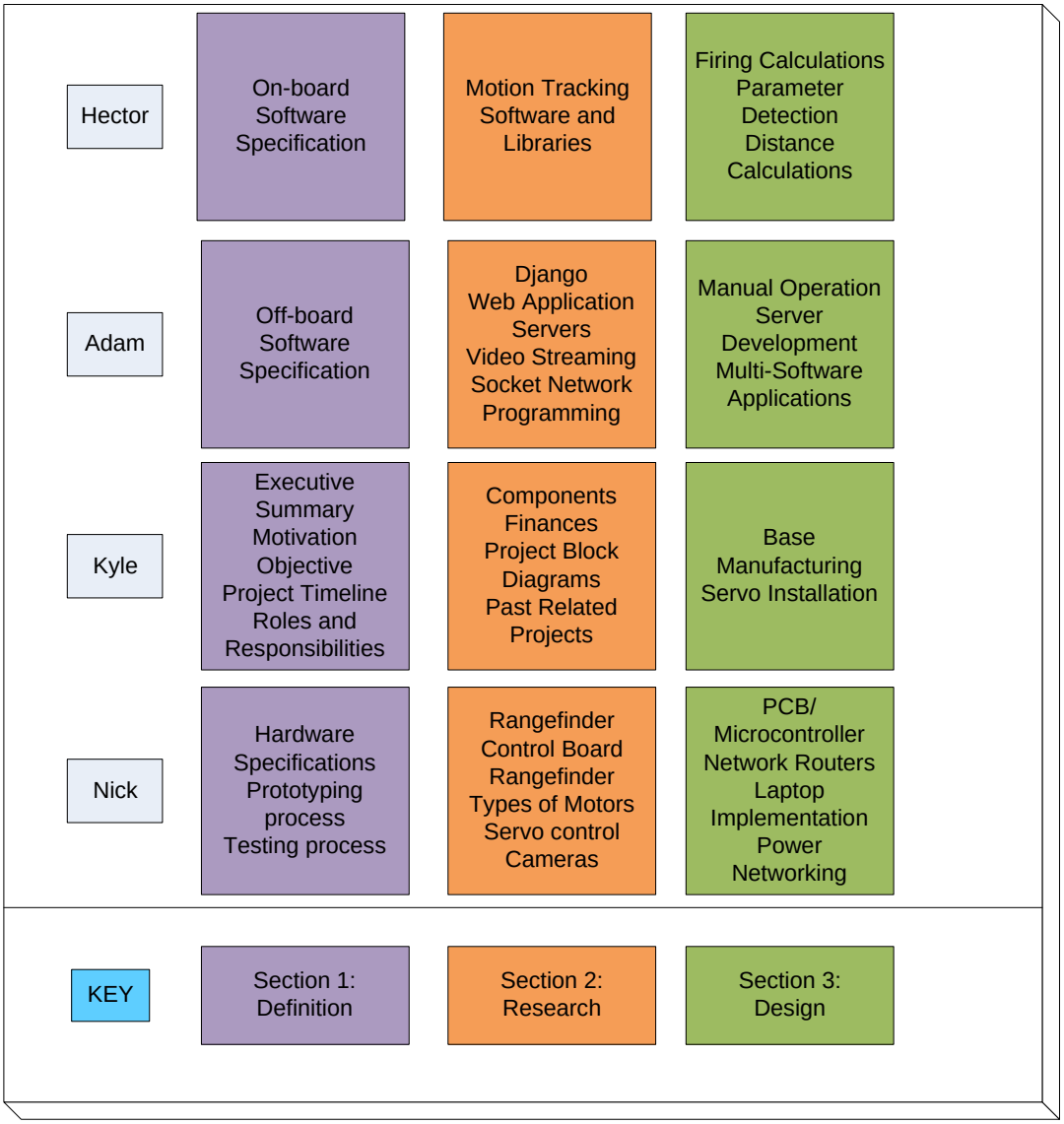

Figure 2. High Level organization of Roles and Responsibilities for prototype

1.5.2 Coding Breakdown

Adam and Hector will be the main coders of turret software. They will handle a majority of the software design and development. The coding of the turret software will be split into two distinct sections. The first part is the local software that runs on the PC controlling the autonomous functions of the turret and the second part is the remote software of the web application that runs on our server.

Hector will be coding the local software which includes the motion tracking system and the automated movement and firing systems. He will also be responsible for creating the targeting and tracking system which uses the web cam and the motion detection to identify targets and track and locate their position. Basically, any code that runs on the local PC will be his responsibility.

Adam will be focusing on the web application aspect of the program. This code sits on the web server and serves the website of our web app. The web app will allow users to view the status of the turret, view engagement history, and allow for remote manual control of the turret. This means that he will be in charge of the web site development, database management and programming, as well as the manual control aspects of the turret design.

The two programmers will share responsibility on developing the system that lets the PC communicate with the microcontroller and move the servos. This section will be the foundation of both the automated and manual controls and will require assistance from the two electrical engineers as well. Getting the software side and the electrical side to work together will be a group effort and will constitute one of the few areas where the expertise of all of our different backgrounds will be used.

It is important to note thought that these roles will be fluid throughout the development of the turret software and both coders will be involved in all aspects of the turret software development as needed. This is especially true as development of both the client and server side becomes more mature and collaboration between the two becomes necessary.

1.6 Project Overview Block Diagrams

The software block diagram illustrated below in **Figure 3** shows in detail the software aspects in each component of the autonomous turret. There are 2 main components in which software is heavily dependent upon. These are the turret and server which we are using for our system and this is the high level components that use software. The rest of the blocks inside of these two high level components are detailed to the low level components. The turret system is going to require a software program to handle the calculations for firing, distance, and target distance. There is also the capability of manual control of the turret but this is only when the user requests to turn the system over to him or her. The server software is going to handle the engagement history, and turret data server, as well as the manual control of the turret. Data that is collected and

recorded will be stored on MySQL to be accessed at a later time. As the image illustrates the only component that our computer engineering group members need to do detailed research on is the motion detection of the turret. The other components are either acquired or need to be developed.

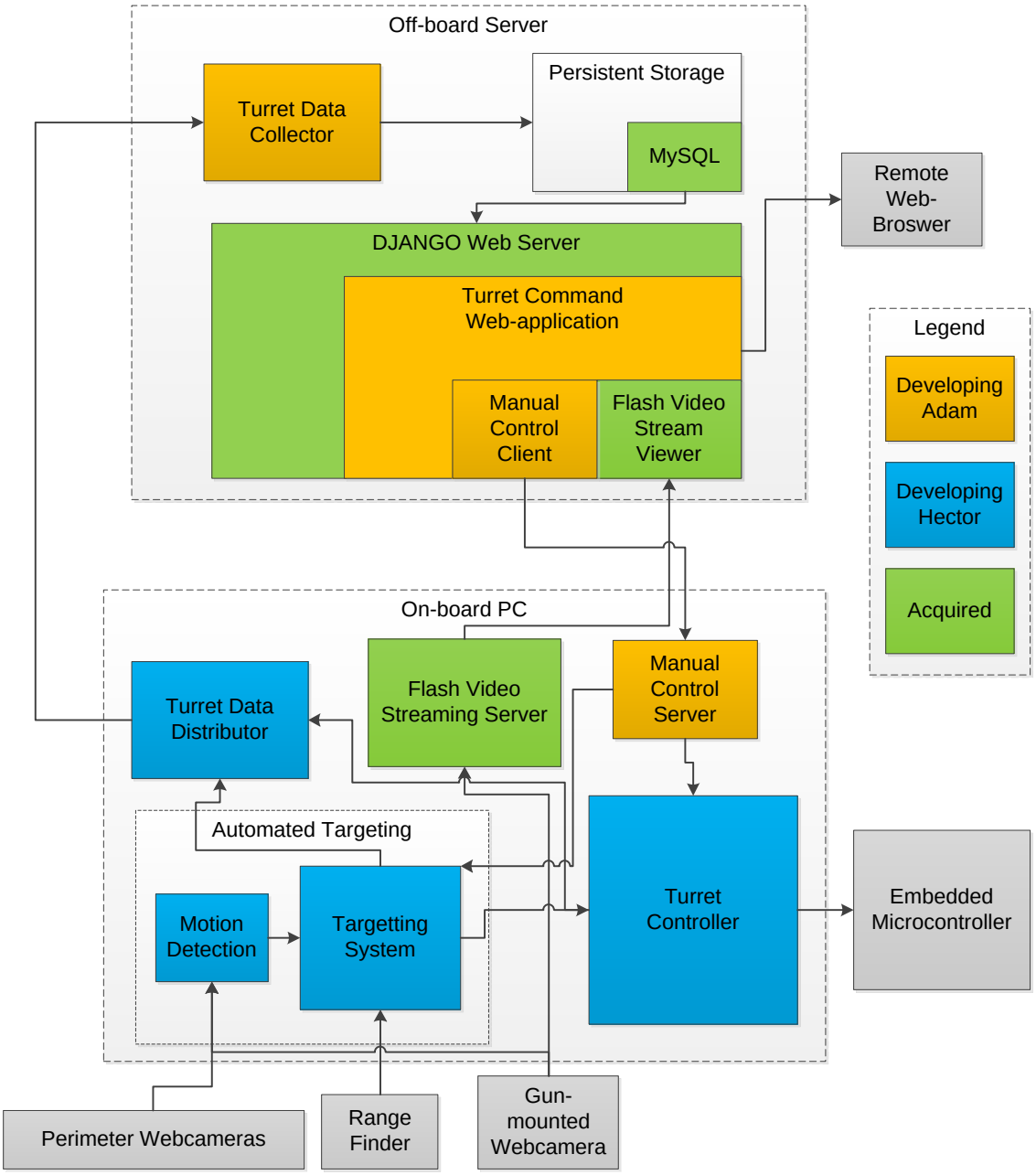

Figure 3. Layout of the software block diagram for our autonomous turret

The hardware/electrical/mechanical block diagram layout can be almost compared to the complexity of the software block diagram. The high level components of the hardware/electrical and mechanical block diagram are the base of the turret, electronic control systems for the turret, and the connection to the computer system. The low level layout of the main components contains a

wide variety of components in them. The base of the turret will have to be designed mechanically. The control of the pivoting of the turret will be controlled by the 2 servo motors and the other servo motor will control the operation of the trigger.

The gearing ratio that we are looking to achieve is a 1:1 ratio. Now on the electrical side of the base operation the servo motors will require electricity as well as the servo that controls the trigger but will have to be an analog voltage signal.

The electronic systems of the autonomous turret include the operation of sensors and the PCB. In these 2 subcategories there are many other components that are electronically controlled. The microcontroller that we have selected will be connected from the turret to the laptop of one of our group members. There will be sensors for the laser finder and motion tracker. The computer system lower level components will include a wireless network that that is linked between the laptop and the server that our group will be using. The computer system components are linked between the hardware and software block diagrams but have different subcategories.

The components that need to be developed are labeled respectively in the image. Once every software component is developed and assembled together it will be ready to integrate it with the hardware and electrical components of the turret system. The hardware block diagram is illustrated below in **Figure 4**.

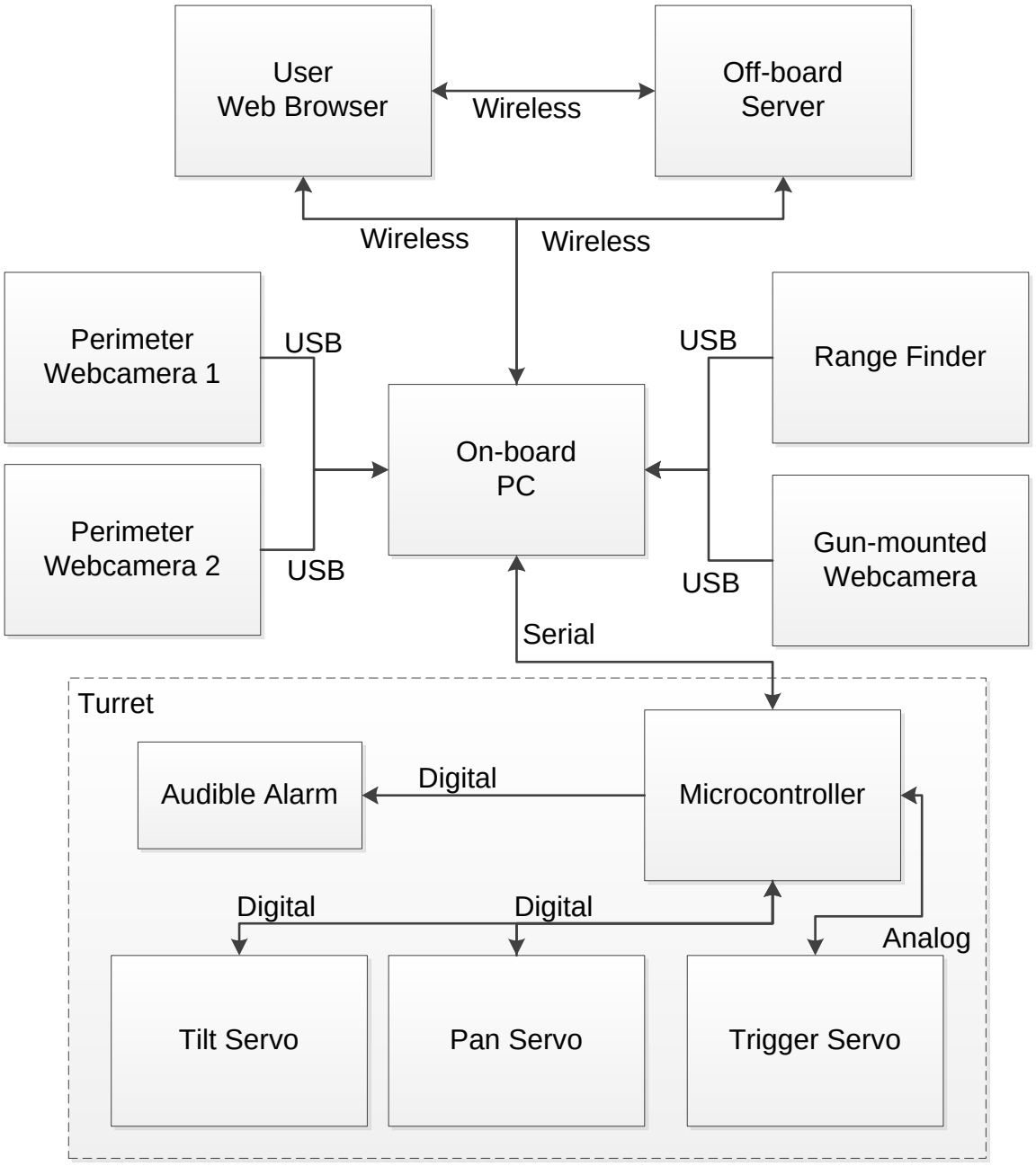

Figure 4. Hardware block diagram for our autonomous turret

1.6 Project Timeline

As you can see from table 6 our group started 3 weeks after the class began which is one set date and the other set date is the date of submission for the paper. The majority of the projects' paper is research/design and actually writing the paper. Thus also seen below in **Table 1** is that the research and writing of the paper spans over almost 2 months.

Fall 2010	61 Days	9/13/2010	12/6/2010
Group Organization	5 Days	9/13/2010	9/17/2020
Identify Components Needed	5 Days	9/20/2010	9/24/2010
Research	28 Days	10/4/2010	11/102010
Writing Project Sec Document	33 Days	10/6/2010	11/19/2010
Proofreadign/Reviewing Final Document	10 Days	11/22/2010	12/5/2010
Submission	1 Day	12/6/2010	12/6/2010

Table 1. Timeline of Senior Design I paper

Our timeline has changed from the initial timeline we submitted in the beginning of the class. Initially we had the research set at a specific time and then once we were done with all of the research then we would write the paper. Although in reality time is of the essence in any real world application and thus once you have completed a research topic you should write about what you have researched as soon as you are done. The proofreading and format applications will take place towards the end of the semester and are estimated to last for 1 week.

Spring 2011	80 Days	1/5/2011	4/26/2011
Acquire Materials	14 Days	1/5/2011	1/24/2011
Build Project	21 days	1/25/2011	2/22/2011
Test	14 Days	2/23/2011	3/14/2011
Fixing bugs and iterative development	20 Days	3/15/2011	4/11/2011
Demonstration/Submission	6 Days	4/12/2011	4/19/2011
Website	5 Days	4/20/2011	4/26/2011

Table 2. Timeline of Senior Design II prototype

The prototype that we are going to design in the second semester of Senior Design will have a timeline that is illustrated in **Table 2**. Since we are looking into the future with the prototype it is an estimate at the moment and will quite possibly change when we reach this point in time. We estimate that we will have purchased some components at the end of Senior Design I and when we return from winter break we will begin working on acquiring the rest of the components. There is one day set in stone at the moment and that is the beginning of the spring semester. The demonstration of the prototype is will occur at a date set by the professor of the course Dr. Samuel Richie. The rest of the timeline for the

prototype will be connecting all of the components together to realize what we originally had hoped to design and testing it. There will obviously be issues that arise from the design of any prototype because it is the first time you are designing something that you have never done before.

That is why our group has set aside almost a month to fix any issues that arise in the design. Towards the end of the second semester of senior design we will have to design a webpage and we have allotted 5 days maybe 7 to complete this task. I believe that our computer engineering group members will definitely know how to design a webpage and the electrical engineering group members will work alongside Hector and Adam to complete this task in the time that we are estimating.

Section 2: Research on Past Related Projects

There have been many past related projects that are related to our autonomous turret. These projects include individual projects that can be found online and past senior design projects with similarities to our own. Researching past projects will allow us to view the success and failures of previous attempts and build upon the experience of others.

Searching for autonomous paintball turrets online leads to a plethora of websites that contain personal projects from many different talented individuals. There you can find summaries of their methods, pictures of their final design, and video demonstrations. One of the websites we most look at during the beginning phases of development was The Sentry Project **[1]**. There are many prototypes and videos posted on this website and it contains exceptional examples of what we wish to accomplish with our project. The YouTube video that is on the main page contains a system that is very similar to the one we wish to develop. The system will consist of the paintball gun and an external system connected to the laptop via PCB and microcontroller. The turret system will also consist of range targeting along with the use of web-cameras. This concept is also similar to the system in the video in that you can see the use of video streaming and laser targeting on the barrel of the paintball gun.

In a sense there will be similarities but the characteristics between our system and the past projects will be different. Overall though I believe that the turret system portrayed in this video and all of the different types of systems that are on this website are exceptional. However this is a website and business owned by paintball sentry for the use of profit. So in comparison with the types of turrets that are posted on this website and the system that we are proposing should vary quite significantly.

Another excellent source of research were the other University of Central Florida Senior Design projects that are also very similar to the one we are proposing to do. It is helpful to compare our project with the rest of the similar UCF senior

design projects since they are on a budget and are for non-profit. The groups that we looked at for research where the G8 Sentry Gun **[2]**, the Paintball Targeting System **[3]**, and the Motion Tracking Sentry Gun **[4]**. The groups all had various similarities to each other and our own ideas but there were some key differences. One thing that stood out was that the G8 Sentry gun had a system that could distinguish between friend or foe. It accomplished this by using a GPS system. Friendly targets would carry a mobile device with GPS positioning and the information from the GPS would inform the turret that there was a friend nearby and not to shoot. We actually considered a similar idea when brainstorming the features we wanted our project to include. We also considered using color to distinguish between targets and designing the turret so it would not shoot at people with green shirts for example. The biggest difference we have with past senior design projects and what we feel will allow us to improve over them will be the fact that it seems all previous projects tried to do their video processing on an FPGA. By instead doing this work on a PC, it will free us to improve our performance and include functionality that will differentiate our project from the past.

By reviewing past project we can see many different ways to solve the same problems. One of the more abstract and difficult parts of our turret to construct was the mount that the gun would be connected to. We can see by going through the pictures of past projects that this is a problem that has plagued any group that has attempted to do an autonomous paintball turret project. Every project used a different approach to securing and pointing the paintball gun. The fact that there were so many variations on what seems like a simple problem let us know that it was an aspect of the design we should focus on. Not only does the mount have to be sturdy enough so it can withstand the kickback from the gun and keep it properly calibrated, but the mount must be designed in such a way so that two servos could turn and point it in two different axis. We compared the different mounting from the past projects and weighed their pros and cons and found one design we liked that we are basing our design on. This method of reviewing past projects and comparing their solutions to the solutions of others was applied to several aspects of our turret design and proved to be a useful way of using the past experience of others to help our own design.

When looking at all past paintball turrets as a whole there are some big differences with our design as well as smaller differences that are expected. The small differences meant that we cannot rely totally on past research and must rely on our own knowledge and expertise. We will not be using the same hardware components and quite possibly not even the same software components that are incorporated into past projects. Also the behavior of the turret and the algorithm that dictates how it selects and engages targets is for a large part personal preference in how the turret is to behave.

There is one major difference in our project that we were unable to find in any past projects that we feel really sets our design apart and keeps it from becoming

just another paintball turret. That is the networked web application that will allow for manual control of the turret as well as provide an off-board radar and serve engagement history to users. There are a couple of projects that we were able to find on the web that allow for manual control, but nothing comes close to the seamless integration of automated and manual controls and the suite of functionality in the web app that we wish to achieve.

There will be one exact aspect of similarity between all of our projects however; the projects will be open ended. There will always be room for improvement on accuracy, efficiency, quicker processing speeds, quicker response speeds (motors), etc. Although since the fact remains that these previous projects have already have completed their turrets and we know what type of components they used we know what to do and what not to do in order to make our project top of the line. We are going to try and optimize cost and value to surpass the last 3 groups in the characteristics that we want to be great such as some of the ones listed above in this paragraph.

2.1 Range Finding Method

For our turret, we plan to have the ability to track range of targets. In order to set up a perimeter, our range finding capability will need to be a distance greater than our planned firing range. For this, we will put a need to find a solution that has fairly high accuracy, and has a range of at least 75 feet. We have several options in technology to accomplish our task. The most feasible solutions to range finding are LIDAR, Infrared, ultrasonic, and pointed laser. Each of these has advantages and disadvantages in their application.

LIDAR are laser devices that perform a sweep across a horizontal plane, sweeping many times a second. With the data received, it is possible to create a two dimensional map of an area in front of the LIDAR, with a wide angle of view that updates in real time. Some LIDARs can have great distances, wide angles, and high accuracy. The downside, is that LIDARs are much more expensive than our project can afford, being several thousand dollars in cost and up. Although great for our application, this will not be achievable due to budget constraints, unless one can be loaned out, which is not foreseeable at this time. Infrared range finding uses a pointed IR beam that can be set up in a sweep if needed. They are inexpensive, but have very low range, and poor accuracy. Often IR range finders are limited to such short distances as 3ft, which will not meet our needs. Ultrasonic range finders have greater range, but have issues with size detection, and are more difficult to interface with than our other choices. This leaves pointed laser range finders. These are used for a wide range of applications such as golfing and construction work, and have price ranges that our within our budget. They have ranges that can reach well over 100ft, and accuracy as precise as less than an inch. They are not setup in a sweeping mode, so they can only track one target at time; however, this will be fine for our application. Because of these reasons, we find this to be the best solution.

2.2 Motor Options

In order for the turret to aim and pull the trigger, a system of three motors will be needed for full motion and action. These include a motor to aim across the X axis of view with a pan motion, a motor to move across the Y axis of view with a tilt motion, and a motor to pull the trigger. For each of these jobs, an electric motor will need to be chosen that can best suit the need. AC motors are very powerful and useful, but since they have price tags that are far beyond the budget of the project, these shall immediately be eliminated as an option. Each motion can be solved with a rotational action, so for electric motors we can choose between a standard DC motor, a stepper motor, and a DC servo motor. The advantage of a standard DC motor is that the largest amount of torque can be achieved for the price, but there is no linear control system built in. The motor is either on or off, and an external feedback control system would need to be created separately to work with the motor, adding many mechanical problems to overcome. The advantage of a stepper motor is that there is a control system built into the motor, but it is an open loop system. Positional commands can be fed to the motor, but no position can be read back, and the motor also contains more vibration than the other two options. The DC servo motor has less torque than the other options, but has a closed loop control system built into it, letting it move to precise positions with much less error. This factor is very important in the design, and extra money can be spent to get a servo with high enough torque to make up for its weakness. This is why servo motors will be chosen for the pan and tilt actions. The trigger action will take less precision and torque, but a cheaper servo motor will be used in this application, to keep uniformity in the overall system design.

Aside from torque, accuracy is also a concern in motors. The goal of the system is to have a range of 100 feet, and it is at this range that the accuracy will come into play the most. Since the motors are turning the direction of the turret on a central swivel, each degree of error creates a larger inaccuracy at higher distances. This error can be related to a right triangle, with the base representing the distance to the target, and the height representing the margin of error the target is missed by. The **Equation 1** below represents the calculation of this error. In order to reliably hit a human sized target at a range of 100 feet, at least one foot of accuracy is needed. From **Table 3** below, it is shown that at least 0.5 degrees of accuracy on the targeting servo motors are required to meet this goal.

$$D \times \tan \theta = E$$

D = Distance to target, θ = Angle of Error, E = Off target error

Equation 1. Calculation of aiming error

Angle / Distance	50 feet	100 feet
2 degrees	1.74ft error	3.49ft error
1 degrees	0.87ft error	1.75ft error
0.5 degrees	0.44ft error	0.87ft error
0.25 degrees	0.22ft error	0.44ft error

Table 3. Aim error according to distance and angle

2.3 Motor Control Method

In order to properly use the motors in our design, there will need to be a method for controlling them. First, there needs to be a linear control system with a way to provide feedback of the motor's position compared to the desired position. The most clear and efficient way to tell the position of the motor is using a potentiometer. If the potentiometer rotates with the motor, the resistance across the potentiometer will change as they rotate. This resistance can be translated into the position of the motor in degrees. Using this as a sensor as the feedback loop in the control system, the motor can be accurately moved to exact positions. Servo motors can include this system with a potentiometer assembled within the casing. The potentiometer is a limiting factor in the range of a servo motor, with most unable to turn 360 degrees. The gearing is also a limiting factor, with built in stops to prevent damage to the potentiometer. A servo that is geared to rotate 180 degrees will be required for both the pan and tilt motions.

In order to control the Servo motors, a message system must be established. Digital Servo motors are able to be programmed, with the method of control messages able to be customized. For our application, however, the standard Servo message system will serve well. The message system used is a Pulse Width Modulation. The input is a digital square wave, where zero and one are set voltages. The length of each in the wave determines the input. For example, a servo that rotates over an angle of 180 degrees may move to the 0 degrees position at a pulse length of 600 µs, and the 180 degrees position at a pulse length of 2400 µs.

The input connection of a Servo motor uses three leads: positive voltage power, negative voltage power (ground), and the control signal of the type described above. In order to generate this signal, and intermediary between the user input command and the motor input is needed. For this a microcontroller is used. Code can be written to the microcontroller that generates pulse signals to the motor upon receiving commands from its own input. In this fashion, the microcontroller can be connected between the motors and the computer, so that the software can give direct commands that will be translated by the microcontroller into useful inputs for the motor. **[5]**

2.4 Microcontroller Research

2.4.1 PIC microcontroller option: Design and coding

While we plan to use an Arduino board in the developmental phase for motor control, we also plan to design our own microcontroller circuit to use, and to build it for the specific purpose of the project. While more difficult and less expansive of features than a premade Arduino, this would add more design to the motor control portion of the hardware design. Also, circuit created specifically for the turret's motor system would streamline the design and the way it is interfaced with the rest of the turret system.

There are two main options for choosing a microcontroller chip to design the motor control circuit with. Each have their advantages and disadvantages. The first, is to use an Arduino microcontroller chip, and design and build a custom arduino board around it. This would allow the programming of the microchip to be done in the convienient arduino environment. The code can be written in a language similar to C, and use the arduino libraries allowing for a well supported, robust environment to program in, with great expandability.

The second option for use in the design is a PIC microcontroller. The schematic for this configuration is shown in **Figure 5**. These microcontrollers have less of a robust boot loader and interface than the Arduino microcontrollers, but are well known in their use for controlling motors. Lots of information is available on the programming of PICs for servo motor control, and they have been proven in the application in the past. PIC microcontrollers come in 8 bit, 16 bit, and 32 bit varieties. An 8 bit microchip should be sufficient for use with this design. The method for controlling the servo with a PIC is the use of an oscillator in conjunction with the microcontroller. For example, an 8 bit timer is used to generate the Pulse Width Modulation of the correct period, and the other outputs on the PIC are used for the power and ground of the motor.

For the use of a PIC microcontroller, a circuit can be made specifically to control servo motors, allowing for a simpler design but a less open and expandible one as well. The most simplistic circuit to control servo motors with a PIC requires three parts: an oscillator, the PIC chip, and a power circuit. The oscillator is connected to the PIC on it's oscillator pins, and the power circuit is used to regulate the power input for the microcontroller and the motors. Below is an example circuit that can be used to control one servo motor in a simple way.

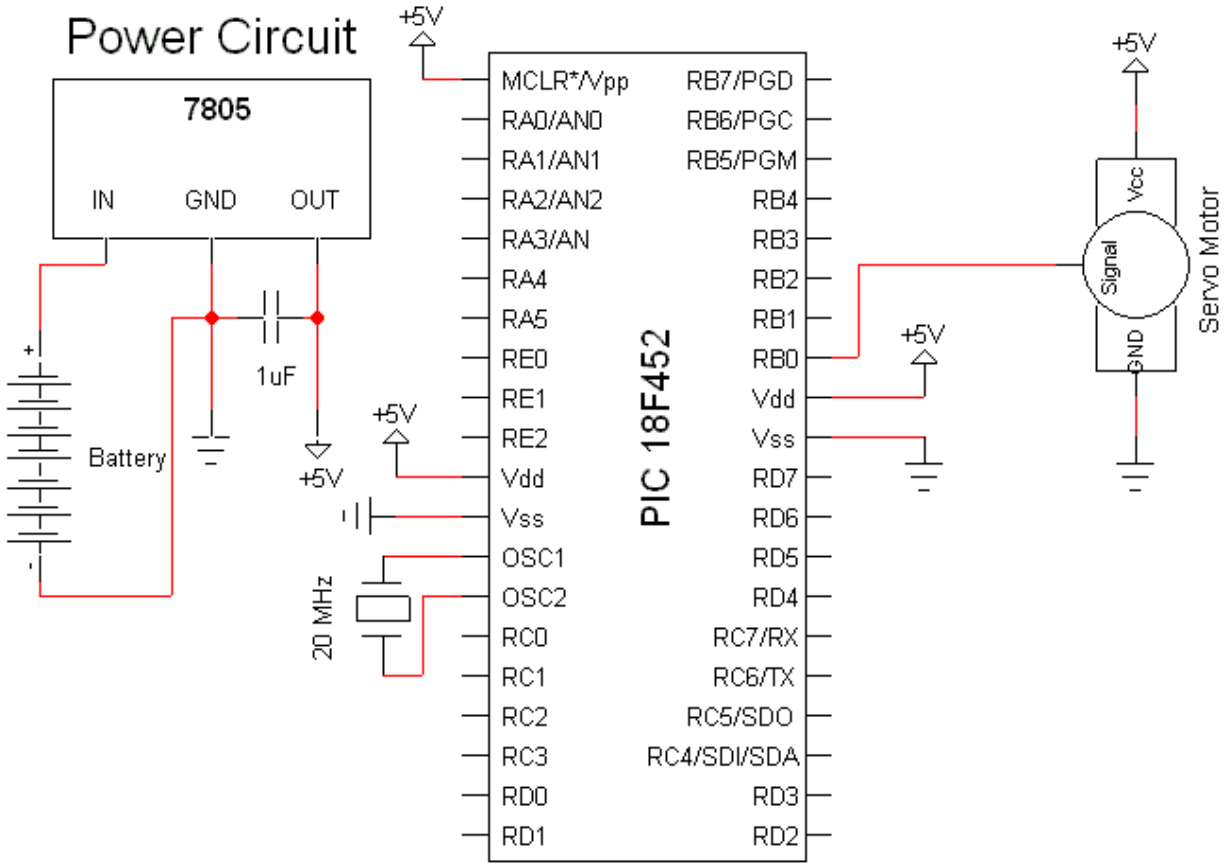

Figure 5. Example schematic of PIC controlling one servo
Copyright Pyroelectric.com. Reprinted under Fair Use

There are two main programming methods with a PIC to create the PWM necessary to control the servo motors. The first is by using delays, the second is by using timers and interrupts. The following is pseudocode that contols one servo with delays. The instruction speed and oscillator speed are used to calculate the delay needed to create the desired pulse length to create each position command. For a general servo command, the signal frequency is at 50 Hz; this makes the period of the signal 20 ms. The positive peak of the square wave can last from 0.9 ms to 2.1 ms, leaving the zero voltage trough of the wave to range from 17.9 ms to 19.1 ms.

```
start
        while motor to stay at position 1
                raise voltage on signal port to high (logic 1)
                delay for pos1 ms
                lower voltage on signal port to low (logic 0)
                delay for 20ms - pos1 ms
                //end position 1

        while motor to stay at position 2
```

```
                raise voltage to signal port on high (logic 1)
                delay for pos2 ms
                lower voltage on signal port to low (logic 0)
                delay for 20ms - pos2 ms
                //end position 2
    end
```

The second method to controlling servos with a PIC microcontroller is to use timers and interrupts to create the pulse signals. This is more advantageous to control multiple motors. Below is pseudocode that controls two servo motors at the same time.

```
    start
        //calculate timer hex for 20ms = timer20ms
        //calculate position timer in hex for servo1 = timerservo1pos
        //calculate position timer in hex for servo2 = timerservo2pos
        while servos to stay at positions
            begin timer20ms for servo1
            begin timer20ms for servo2
            begin timerservo1pos
            begin timerservo2pos

        timerInterruptFunction()
            interrupt timer when complete
            lower voltage to 0 (logic 0)
    end
```
[6]

2.4.2 ATmega Microcontroller Option

The ATmega chip is the second option for a microcontroller in our design. This chip is the line of microcontrollers that are used in Arduino designs, and would allow us to work in the Arduino format. The 8 bit ATmega microcontroller line has digital in/out pins, and analog in pins. One of the advantages it has over the PIC microcontrollers is the inclusion of specific digital out lines that have the ability to produce PWM without extensive software simulation. Since we are mainly controlling servo motors with the microcontroller, this is a big advantage. Also, the microcontroller chips come with a bootloader installed, allowing the chips to be programmed without the need to buy a programmer, saving on costs.

The Arduino coding environment is done in a extentionless file format called sketches. The language is based on Wiring, which is very familiar to the syntax of C. There are many libraries that can be included in the coding, including a servo library that allows the control of many servo motors. By choosing an ATmega chip, we can design the board to fit the pin spacing of a pre-made Arduino, allowing our board to be compatable with all of the Arduino accessories on the market, for future use.

Below is the pin layout of the ATmega328 shown in **Figure 6**, a 28 pin microcontroller that is used in arduino boards. This chip has 32 working registers, 23 general purpose I/O lines, 3 timer/counters, and a 2 wire serial interface, among other features. The chip also features 32K bytes of programmable flash, 1K bytes of EEPROM, and 2K bytes of SRAM. It has 6 Analog input lines, and14 digital I/O lines, 6 of which can be used for Pulse Width Modulation. Since we will be using only 3 servo motors, these 6 PWM lines will be more than enough.

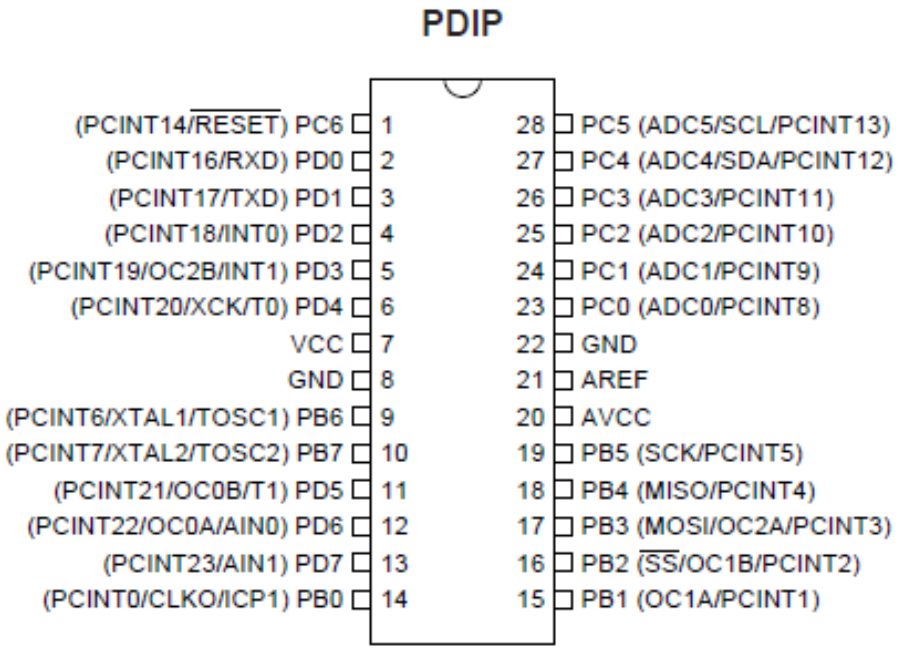

Figure 6. Pin configuration for ATmega328 chip
Copyright Atmel Corporation. Reprinted with permission.

2.4.3 Angular Torque Calculation

To begin with the calculation of the minimum torque needed to maneuver the turrets yaw and pitch we will begin with the angular quantity instead of the linear F=ma form. Our formula that we will utilize is the Newton's second law for rotation which is T=ıα. The moment of inertia is the formula $(m(O_D^2+I_D^2))/2$. M is the mass of the paintball/airsoft gun which is .062 slugs. O_D is the outside diameter of the metal cylinder which is 2" and I_D is 1.875". Knowing this plug in the values ı=(.062 slugs)$[(2^2+1.875^2)/2]$ and the moment of inertia is ı=.2336 lb-s2-in. Then we need to calculate the angular acceleration which is the formula α=(ᘐ2-ᘐ1)/(ΔT). Thus, α=(5.236 rad/s – 0 rad/s)/(.3 s)=5.236 rad/s. Now to find the total torque τ=(5.236 rad/s)(.2336 lb-s2-in) is equal to 1.2231 N-m which when expressed in oz-in is 173.206 oz-in. We wanted the torque converted to oz-in because the manufacturer of our servos expresses their torque in oz-in. Thus

based on our estimate of 173.206 oz-in we still will go with the 250 oz-in servo for the fact that it is more expensive but it has a faster response time. There is also the fact that the servo below the one we have chosen is below this estimate so therefore we have went ahead with a little more expensive servo but may have definite benefits when we design the prototype.**[7]**

Section 3: Design

3.1 Hardware Components

3.1.1 Airsoft Gun

Our original plan was to use an UTG-MP5, but unfortunately it was permanently jammed. We attempted to disassemble the gun to repair it, but our efforts were successful. A picture of our intended gun disassbled can be seen in **Figure 7.** The airsoft gun that we ended up using was the Crosman Mini Carbine. The airsoft size that we will be using along with the gun is a 6mm caliber. This airsoft gun is full automatic in nature. The mechanism is also designed to allow single fire by manually pulling a bolt spring. The power of the gun comes 4 AA type batteries in the handle. A side view of the Crosman Mini Carbine is pictured below in **Figure 8**. Our group is looking to use the paintball gun as well as the airsoft rifle for the following reasons. With the airsoft gun you have an overall lighter weapon but you have decreased accuracy from the lighter ammo. With the airsoft rifle you have an overall lighter weapon but you have a decreased from the lighter ammo.

Figure 7. Disassembled UTG MP5

Figure 8. Side view of the Crosman Mini Carbine Airsoft gun. Reprinted under Fair Use.

3.1.2 Base and Mounting

Our turret system will be mounted on a single arm that is mounted or attached to a circular base that can pivot in 180 degrees left and right and have a minimum of 45 degree pitch downwards. This section of this paper addresses how our team will attach the paintball gun or airsoft gun to the base that we will be using to support the force of our turret. The paintball/airsoft gun will actually be suspended in the air by the use of aluminum parts that appropriately support the force when the turret is fired off. The base and mounting support will be controlled with a total of 2 motors to turn the turret in the appropriate directions by adjusting the point on the x axis or pitch and y axis or yaw. For our turret system to operate in the way that we are proposing, the turret needs to be mounted on a circular base. The paintball/airsoft gun will not be welded or permanently fixed to the circular base because our group is trying to make the system compatible with any paintball/airsoft that is mounted onto the turret. Quite possibly even a real gun may be mounted to the system that we will be designing but we are not going into that area of discussion.

The main supporting aluminum beam that will be supporting the weight of the paintball/airsoft gun in place will be machined to do this. The wood support base that we built will have custom U bolts that the paintball/airsoft gun will rest securely on. Though, we will not make the U bolts that only fit the paintball gun since we are looking to allow any type of simulated or real gun to fit on the mounting system. So in order for the type of gun to securely rest on the arm there may be a need for tie-wrap or aluminum O-ring fasteners. We doubt that

the paintball/airsoft guns will have enough force to kick back to cause the turret system to be unstable. There are some airsoft guns however that imitates actual gun movements and our group must compensate for this issue. There will be one motor underneath the circular base that will have enough torque and force that enables the base to pivot at the time response we want.

The motor that we will be placing under the base will provide enough torque such that the motor does not burn out under the stress of quick response tracking. One of the other motors that we will be utilizing in our turret design is the one that will be controlling the firing of the trigger. The paintball/airsoft triggering motor will simulate automatic firing since the guns are semiautomatic. In order for this to happen the trigger will have to be controlled electronically so that when there is a pulse of electricity the gun is fired. All of the above components will also be connected to the aluminum support beam that stands up vertically on the right side of the paintball/airsoft gun.

There will be some thickness and length associated with the support beam. We are going to incorporate 2 aluminum gears into the support beam. One gear will be placed underneath the support beam connected to the HS-5745MG Digital ¼ Scale motor to control the pitch of the paintball/airsoft gun and the other gear will be placed near the top of the paintball/airsoft gun to control the yaw of the paintball/airsoft gun. In any practical sense it is better and becomes less problematic when you use less components, thus in the design of our support arm it will be an altered aluminum cylinder with dimensions to be determined at a later time.

The turret base is pictured below in **Figure 9** to show the final design of the base

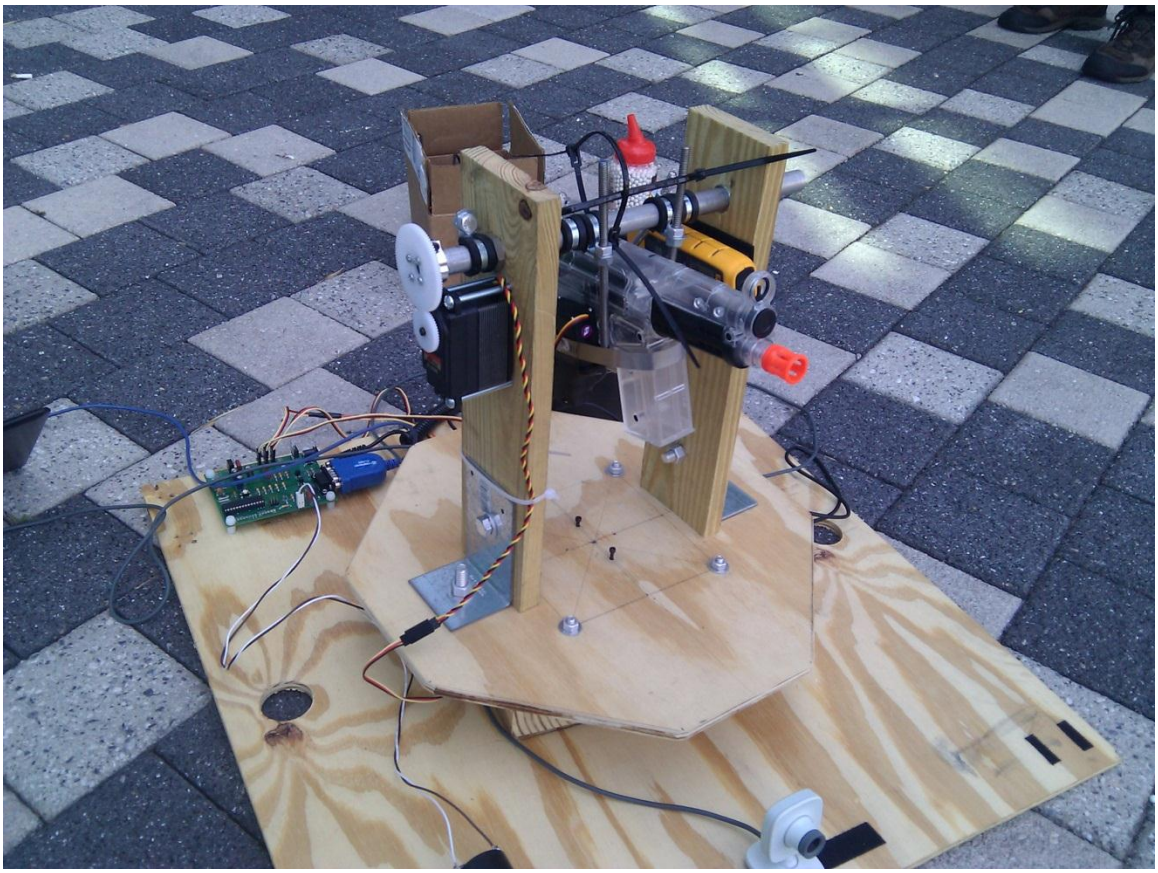

Figure 9. Photograph of turret base design

3.1.3 Servo Motors

To determine which servos we will need for the implementation of our system we needed to know how much torque and how many seconds per degree the servo operates at. Other factors that contributed to our selection of the servos were cost, similarities to other servos, and in some instances there were actually servos that cost more but delivered less performance. Out of the many sources on the internet 2 websites stuck out for the best selection of servos. These websites were servocity.com and hobbyhorse.com. In reality any business that sells remote control hobby devices were found to carry servos but servocity.com and hobbyhorse.com had some of the same servos but cost less. So after we found some sites that carried servos we determined which ones we should choose. We determined that we need a total of 3 servos; 1 that controls the rate of fire by control of the trigger mechanism, and 2 that control the yaw/pitch of the paintball/airsoft gun. We also determined that since both guns require a low amount of pressure to pull the triggers we would only need a low torque and quick responsive servo for the operation of the trigger. On the other hand the control of the yaw and pitch we determined that there will be a great amount of torque needed in comparison to the one for the trigger. Knowing these

parameters and the websites that provide servos we started researching and we concluded the following.

The servo that we decided to choose for the operation of the trigger is the HS-485HB from servocity.com. We came to this decision because it was the cheapest out of all of the other Analog servos ($17.99), it has the fastest response time in seconds per 60 degrees (.11) for the price, and it has a reasonable amount of torque that is required. The torque of this servo is 76 oz-in which is approximately .4 lb-ft. The other Analog servos that we chose this one out of were out of another 7. The servos when placed in increasing price increased up to $39.99 and the maximum speeds decreased as well which sticks out like a sore thumb because you would be paying more for less performance. The torque increases with the price increase but we are not looking for a servo with overkill for the operation of the trigger. The most expensive Analog servo that we found that we considered using for the trigger is called HS-755MG ¼ Scale. It costs $39.99 had an overkill torque of 200 oz-in or 1.042 lb-ft and had a maximum speed of .23 seconds per 60 degrees. We did not select this servo for the reasons of cost and excess amount of torque. **[8]**

The servo that we decided to choose for the operation of the yaw and pitch of the turret was very limited to a selection of 3 servos. Again we went with a servo that was provided by servocity.com the name of the servo is HS-5745MG Digital ¼ Scale. The reasons why we chose this servo for the operation of the turrets yaw and pitch is because they are digital servos. There were only 3 servos that stuck out for our required needs of torque and maneuvering speed as well as cost. The other 2 servos included one from hobbyhorse.com that was the same exact motor, specifications, and cost as the one we selected. The other servo was from servocity.com and had torque of 343 oz-in or 1.787 lb-ft, maneuvering speed of .14 seconds/60 degrees, and a cost of $78.99.

The servo that cost $78.99 was obviously quite expensive and the torque that is provided seems to be overkill for the operation of the axes. The maneuvering speed is also only .01 seconds/60 degrees faster than the servo that we chose so it does not provide an interest to select it as a choice. Our servo that we selected costs $69.99, has a maximum torque of 250 oz-in or 1.3 lb-ft, and has a maneuvering speed of .15 seconds per 60 degrees. **[9]**

3.1.4 Range Finder

For a rangefinder we have decided on using pointed laser type. There are many of these out there made for a variety of uses, ranging from measuring distances while playing golf, to measuring in construction. The devices are handheld, but use an LCD or sited output to display the information. This is convenient for a person using it, but is of no use for our turret system.

In order to make use of the range finder as a sensor, we will need a way to network the range finder with our system, and receive the data in a stream that can be interpreted and useful. This is our largest factor in the choice, and two options have been discovered to interface with the computer. The first is to simply find a range finder that has USB or serial out. These tend to be higher end devices, having specifications above our needs, and higher in price.

The second option is to find an inexpensive rangefinder of acceptable specifications, with a way to modify the interface to connect to a computer. These are hard to find, but one product is shown below in **Figure 10**, the Fluke 411D. With a 100ft range, continuous measurement, accuracy of plus or minus 3mm, small size, and light weight, this range finder meets all of our specifications. Retails of the device are about $100.

Technical specifications	Fluke 411D
Range (for extended distances, use a target plate)	0.1 m to 30 m (0.33 ft to 100 ft)
Measuring accuracy**	± 3 mm (0.118 in)
Units displayed	00.000 m, 000 ft 00 in 1/8, 000.00 ft
Laser class	II
Laser type	635 nm, < 1 mW
Automatic power off	after 180 seconds
Continuous measurement	•
Addition/subtraction	•
Battery life	up to 3,000 measurements
LCD illumination	–
Data locations	–
Min/Max	–
Audible feedback	–
Pythagoras (Indirect measurement)	Simple
Ingress protection	IP40
Dimensions	123 mm x 50 mm x 26 mm (4.84 in x 1.97 in x 1.02 in)
Weight	150 g (5.29 oz)
Temperature range Storage Operation	 -25 °C to 70 °C (-13 °F to 158 °F) 0 °C to 40 °C (32 °F to 104 °F)
Operating altitude (ISO 9022)	up to 3500 m
Storage humidity (at 35 °C)	maximum 85 % for 24 h
Batteries	AAA (2)

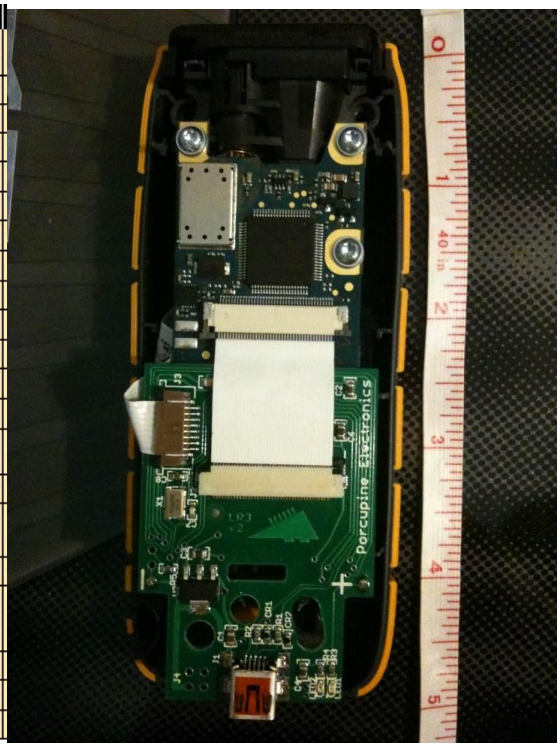

Figure 10. Specs of the Fluke 411D laser distance meter. Copyright Fluke Corporation. Reprinted under Fair Use

A control board has been made by an company name Porcupine Electronics that allows the Fluke 411D to interface with a computer through usb. This control board is installed in the device to replace the LCD screen, after the device has been removed from its case. The resulting connection is displayed in **Figure 11**. The board will allow us to receive measurements from the range finder continuously as a Human Interface Device (HID), receiving 10 measurements per second. The control board adds $150 to the cost of implementation. **[10]**

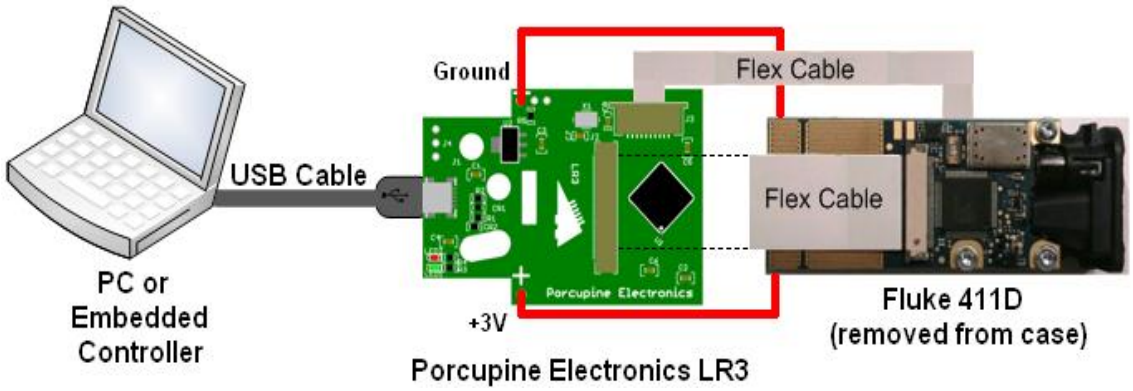

Figure 11. Porcupine Electronics Interface diagram
Copyright Porcupine Electronics. Reprinted with permission

Several other options have been researched in case the acquisition becomes difficult. The rangefinder is the most costly and difficult to find component of the project. **Table 4** lists some of these other options, but the other lasers have specifications above and beyond the requirements, and are thus more costly.

	Price	Range	Accuracy	Sampling Rate	Interface
FLUKE411D	$100 (+150) = $250	0.1-30m	3mm	10 Hz	Control Board - USB
Optilogic RS100	$470	4-100 yards	1m	10-200 Hz	R232 Serial
Acuity AR1000	Inquiring	0.1-30m	2.5mm	6-50 Hz	R232 Serial
Leico D330i	$380	0.05-100m	1mm		Bluetooth

Table 4. Pricing for potential range finders

3.1.5 Audible Warning Alarm

As we are designing an autonomous turret and we expect it to goes off when it sees a moving target either real or simulation there needs to be a warning to the moving target (intruder) that they will be fired upon if they do not leave the area within a certain amount of time. Thus, our group has is going to implement an audible warning system into our prototype to accomplish just this. Our group chose the AudioLarm II from Floyd Bell Incorporated. We based this on one of the group members experience with the alarm sensor and the fact that it costs only $17.40 for a quality alarm sound.
The audible decibel sounds and certain voltages are 85±5 dB at 5 Vdc or 95±5 dB at 30 Vdc, so in other words the sound that this alarm puts out ranges from busy city traffic to the sound of hair dryers. We are expecting to mount this to

base of the turret of our prototype since the width is only about 2 inches and the length is about the same. An example of the audible warning alarm that we will be using on our turret is pictured below in **Figure 12**.

Figure 12. AudioLarm II from Floyd Bell Incorporated
Copyright Floyd Bell Incorporated. Reprinted under Fair Use

A chart of the dB vs the input voltage and current is pictured below in **Figure 13**.

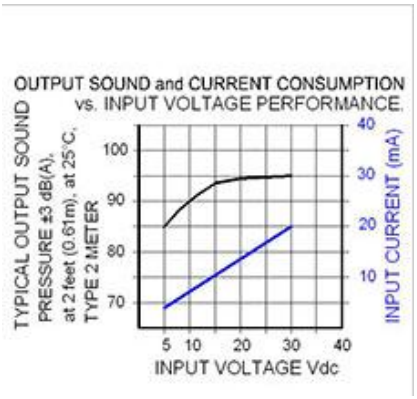

Figure 13. Voltage and Current relationship to output dB
Copyright Floyd Bell Incorporated. Reprinted under Fair Use

The exploded view of the siren that we will be using is pictured below in its respective parts in **Figure 14**.

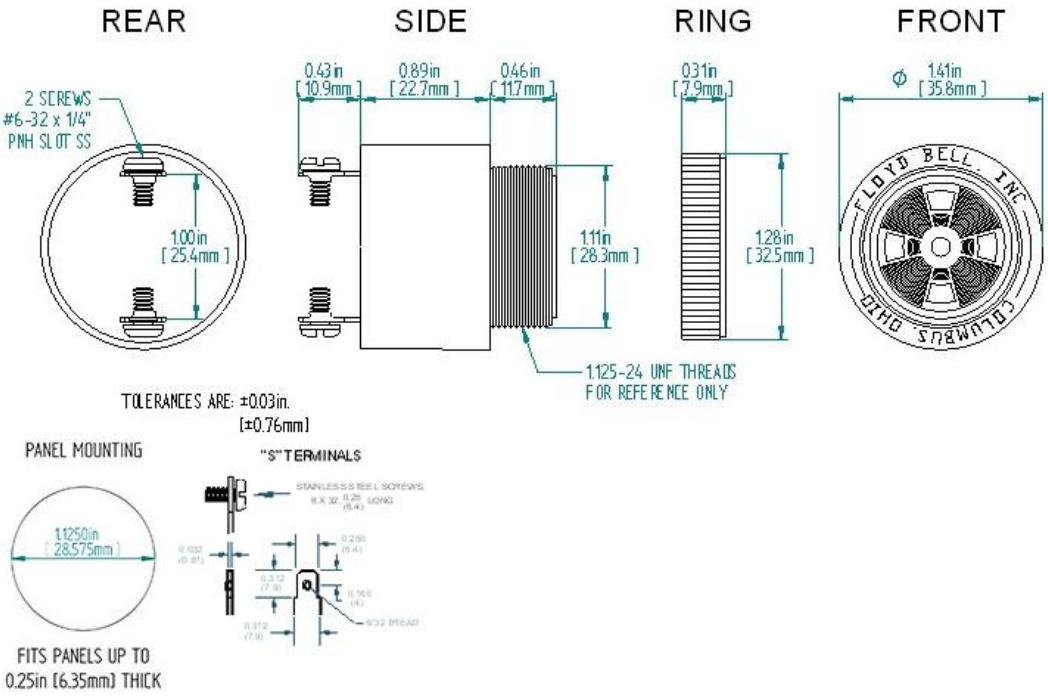

Figure 14. Exploded view of the AudioLarm II from Floyd Bell Incorporated Copyright Floyd Bell Incorporated. Reprinted under Fair Use

The operating voltage is 3-30 Vdc which excellent since our microcontroller operates on a 5 Vdc voltage as well. The operating current that the siren requires is 2 mA at 3 Vdc or 20 mA at 30 Vdc. Based on the specifications listed on the product homepage this siren is definitely compatible with our microcontroller and alternate power sources. The siren will be controlled by the microcontroller so that it will warn the intruder to exit the area and may be powered by the microcontroller or an alternate power source that supplies the required amount. **[11]**

3.1.6 Web Cameras

For the visual recognition system of our autonomous turret we will be utilizing basic webcams that you use for everyday life as well as HD webcams. Our group will be utilizing one HD webcam and 2 low definition webcams. Our HD webcam will be the Logitech 2 Megapixel Webcam Pro 9000 with built in microphone **[12].** The perks of our HD webcam is that it has a Carl Zeiss lens system that provides ultra-smooth video by autofocusing, it provides 2 megapixel; 16x9 widescreen videos and 8 megapixel photos. The HD webcam also has RightLight2 technology that adjusts brightness even in dim light for HD video and photos. The other webcam that we will be implementing into our system will be a USB 6 LED PC Webcam Camera **[13].** Our group needed one HD webcam and two LD webcam and thus this is the LD webcams that we decided to choose. The perks of the LD webcam is that it is cheap for our group, there are six bright white LED's for night vision, and it is plug and play compatible.

The HD webcam will be attached to the barrel of the paintball/airsoft gun for precise and accurate acquisition of the target. The two LD webcams will be located on the base for overall acquisition of targeting to show the difference of images taken. One webcam will focus on the yaw acquisition targeting and the other LD webcam will focus on the pitch acquisition targeting. There will be a total of 3 web cameras implemented into our system one mounted directly on the barrel of the turret and two stationary that are not located on the base. There will be code written by Adam and Hector that computes the difference between an image taken and the next image and calculates where the turret should fire. The webcams will be either tie-wrapped to the barrel of the turret or mounted by some other means. The other 2 webcams will also be tie-wrapped in appropriate locations when the base is built. The webcams will be connected to the netbook via fast USB. The webcams will feed real-time visual images into the netbook, calculations will be done and then the calculations for the yaw and pitch will be sent to the microcontroller. From there, the microcontroller will control the servos and shoot at the target accordingly.

3.1.7 Computer

For the computer component of the design, flexibility and modulation is a goal. On the software side of this, Microsoft Windows will be used as the operating system, covering the widest range of personal computers. For the sake of portability, the system requirements will be low enough to be run on a laptop. The specific computer that will be implemented in the design and tested is the Asus 1015PN, a 10 inch netbook. The netbook has a 1.5GHz dual core processor and dedicated Nvidia Ion 2 graphics. The RAM is 1GB DDR3 standard, but has been upgraded to 2GB. The specifications of this computer are below in **Table 5**.

Specifications

System Model	ASUS 1015PN
Operating System	Windows® 7 Ultimate Express Gate
Architecture	32 bit (x86 based)
Display	10.1" LED Backlight WSVGA Screen (1024x600)
CPU & Chipset	Intel® Atom™ N550
	1.5 GHz 2 Core(s), 4 Logical Processors (Hyperthreaded)
Memory	SO-DIMM 2GB DDRIII
Wireless Data Network	WLAN 802.11b/g/n @2.4GHz Bluetooth 3.0

Display Card(s)	Next-Gen. ION N11M-PT1 with 512MB VRAM
	Integrated Intel Graphics
	NVIDIA Optimus Graphics Switching
Storage	250GB HDD
Camera	0.3M Pixel
Audio	Hi-Definition Audio CODEC Stereo Speakers Mic.
Input / Output	1 x VGA Connector 1 x HDMI port 3 x USB 2.0 1 x LAN RJ-45 2 x Audio Jack (Headphone / Mic-in) 1 x Card Reader: MMC/ SD(SDHC)
Battery	6-cell Li-ion Battery Pack- 6cells
	10hrs (6cells, 56W/h) battery life.
Dimensions	262x178x23.6~36.4mm
Weight	1.25Kg

Table 5. Specifications of ASUS 1015PN Netbook

The advantage of this computer is the maximum portability that is achieved, while maintaining the ability to use an operating system. Unlike a small desktop computer, the netbook has an LCD screen, pointer device, and keyboard built in. This makes interacting with the computer need no additional peripherals. It also has internal 802.11n wifi, as well as bluetooth 3.0, and 3 USB ports. These again lead to less external accessories needed for the computer. The netbook also requires no external battery, with the internal battery lasting 6-10 hours depending on usage. With all of these features, the longevity and integratability of the computer will not be a weakpoint of the system.

The goal is to make the software able to run on this level of computer, so that nearly any computer could be put in place to satisfy the need of this component. If the netbook can run the software side of the system, then any lapop or desktop computer could satsify the same need for computing power. This greatly increases the modularity and expandability of the system.

3.1.8 Server Specifications

A server will be needed for parts of the software integration. Any computer can be used, but the computer will need more power than the computer that is

onboard the turret system. Also, a wireless card will be needed so that the server can connect to the turret system. This will be done either through the router on a DHCP network, or in Ad-Hoc mode directly to the turret. The computer will handle all server functions of logging data, as well as provide an interface to view and control the turret system through the network. Below in **Table 6** is the computer we have on hand that will be used for the server.

Specifications

System Model	Dell XPS m1730
Operating System	Windows® 7 Ultimate
Architecture	64 bit (x64 based)
Display	17" UltraSharpTM Wide Screen WUXGA (1920x1200) display with TrueLifeTM
CPU & Chipset	Intel® Core™2 Duo Processor T8300 (2.4GHz, 3MB Cache, 800Mhz FSB)
Memory	4GB Dual Channel 667MHz DDR2
Wireless Data Network	WLAN 802.11b/g @2.4GHz Bluetooth 2.0
Display Card(s)	NVIDIA® SLITM Dual GeForce® 8700M GT with 512MB GDDR3 Memory
Storage	120GB HDD
Camera	2M Pixel
Audio	5.1 Channel Digital HD Audio Stereo Speakers Mic.
Input / Output	1 x Dual-Link DVI-I port 1 x S-video out port 1 x Firewire (IEEE 1394a) port 4 x USB 2.0 1 x LAN RJ-45 3 x Audio Jack (2x Headphone / 1x Mic-in) 1 x Card Reader: MMC/ SD(SDHC) Consumer IR sensor ExpressCard 54 mm slot DVD+/-RW with Dual Layer DVD+R write capacity
Battery	9-cell 85WHr Li-Ion Battery
Dimensions	406x50.7x302.6mm
Weight	4.81 kg

Table 6. Specifications of Server Computer

3.1.9 Wireless Network Component

In the turret system, a wireless network is needed in order to let the computer communicate statistical information with a server for later data analysis. Also, with a wireless network implemented, the turret can have the option to not only be controlled autonomously and through the main computer, but also from a secondary computer through the network. Opening up the option for remote operation adds a great deal of expandability to the system. This will allow the turret to be controlled from a distant location, as well as settings modified and autonomous systems activated and deactivated remotely. Not only this, but with this system, multiple turret systems may be controlled from a central location, allowing for the system to be embedded in the central security system of a complex. The system could also be expanded to a mobile system in the future, andcontrolled remotely from a cenral locaton like modern military UAV systems.

For the wireless network, a router will be needed to create a DHCP server with the ability to uplink the connections to a WAN. Also, the computers in the system will need the ability to connect to the router through the IEEE 802.11 standard. This is already included in the ability of the netbook, which has a 802.11n card built in. For the router, at least 802.11g should be used, but wireless g should not be necissary. 802.11g is capable of speeds of 54Mbps across the network, which is faster than most internet connections, as well as faster than needed for the software in the system. All modern routers have a least wireless g technology, even inexpensive models. Since wireless n is backwards compatible with wireless g, wireless g is the standard that will be used, as opposed to the faster, yet more expensive wireless n.

The router that has been chosen for the design is the Linksys WRT54G wireless g router. This is a very common, inexpensive router. The greatest advantage in this choice is the Linux based firmware that the router uses. Because of the open source nature of Linux, the standard firmware included on the router by Linksys can be replaced by more powerful custom firmware. This allows a inexpensive router to be upgraded to the equivalent of a much more powerful and expensive router, with a greatly expanded feature set. To the right is a picture of the router, which is a common and easily recognized model. The custom firmware that will be use in place of the standard Linksys firmware is DD-WRT, an open source firmware originally designed for the WRT54G series of routers. **[14]**

3.1.10 Microcontroller

Pre-made Developmental Microcontroller

Our group has chosen an Arduino Uno microcontroller. Our group has made this decision based on the previous work with robotic devices and electronic

machinery. This microcontroller combines all of the necessities of our turret prototype into it. It has a USB I/O terminal for computer communication, linking of the laser rangefinder, and could be used for other purposes. It has both Analog and Digital terminals which we need four our prototype. It has EEPROM, SRAM, and Flash Memory located right on the board as well. The operating software of the Uno microcontroller is also very user friendly and not extremely complicated to implement the commands our group needs to. We chose this microcontroller based on the price and the functionality of the board that we needed in order for our turret to operate. Again there were some overkill microcontroller boards that were over $50 or more and there were too many ports, faster processor, more memory, etc. that we necessarily do no need for this type of application.

We visited over 20 different websites that offered the Arduino Uno microcontroller and the overall pricing of the microcontroller was approximately $25-$30 or more. We have decided to go with the microcontroller from the modern device website which was the lowest price at $24.95. It has all of the same specifications as the one listed on the Arduino distribution website. There were about 10 websites that offered the microcontroller for about $25 but in the end the one we chose was the lowest. There are analog terminals that will be suitable for our trigger servo and any other devices that require an analog signal. There are many digital terminals that will be suitable for our digital servos and the other electronic devices that require a digital signal. The microcontroller also has a 16 MHz crystal oscillator and a physical reset button that one could press to reset the software on the board. There is also an extra added layer of protection for the USB port, there is a fuse that breaks the connection automatically if there is more than 500 mA is applied.

On the main Arduino website we were able to obtain an image, the detailed circuit diagrams, and the software that is required to run the Uno. We were also able to obtain a spec sheet of the microcontroller which is also helpful in the electrical design of our prototype. The Arduino boards are great in a sense that they allow expansions of other components onto their boards. For instance, if we would like to add a wireless connection to the Uno there is a compatibility with the XBee RF module. **[15]**

3.2 Hardware Component Acquisition

3.2.1 Servos

The servos that we will be implementing into our prototype design will be obtained from servocity. The servo that we will be utilizing for the trigger of our gun will cost $17.99 plus additional charges. The two other servos that we will be utilizing for our turret will cost $139.98 plus additional charges; the price is adjusted to include 2 of the servos. The website states that shipping of any product is expected to take 5 days to process and ship.

3.2.2 Rangefinder

The rangefinder that we will be implementing into our prototype design will be obtained from Amazon. The rangefinder we will be utilizing for our turret is the Fluke 411D Laser Distance Meter. At the time that this is being written the current price of this meter is $99.95. The shipping on Amazon products usually vary depending on which vendor you buy it from. Usually an expedited option is available, but not always and there is 2 and 5 business day shipping speeds. Sometimes there are specials where if you buy it at a certain time you get one day shipping for free.

3.2.3 Rangefinder Control Board

The rangefinder control board that we will be implementing into our prototype design will be obtained from Porcupine electronics. The rangefinder control board as of right now costs a total of $150. Our group has contacted the vendor and requested an estimate of how long would it take to process and ship an order. The vendor replied if the order is received by 5 P.M. Central Time it will ship the next business day assuming the items are in stock.

3.2.4 Audible Alarm

The aduible alarm that we will be implementing into our prototype design will be obtained from Floyd Bell Incorporated. The aduible alarm that we will be utilizing is the AudioLarm II. The cost of the alarm as of right now is $33. The vendor stated that from the time of purchase the shipping would be one business week.

3.2.5 Webcams

The webcams that we will be implementing into our prototype design will be obtained from Amazon. The webcams that we will be utilizing for our turret is one HD Logitech 2 MP Webcam Pro 9000 with microphone and there will be 2 low cost LD webcams. The HD webcam currently on Amazon costs $59.99 and the two low cost LD webcams currently cost $6.47. The shipping on Amazon products usually vary depending on which vendor you buy it from. Usually an expedited option is available, but not always and there is 2 and 5 business day shipping speeds. Sometimes there are specials where if you buy it at a certain time you get one day shipping for free.

3.2.6 Turret Arm Base Materials

For the base of the turret, we will be purchasing metal to machine the main pieces ourselves. In order to get a price that is within our budget, we will want to use what is known as drop material. When companies place orders with metal suppliers, the orders are cut from large pieces of materials to exact length, and

short pieces are leftover that cannot be used to fulfill other orders. Certain suppliers will offer these pieces for very low prices, since they are otherwise only left to be disposed of. In addition to this, there are also surplus companies that offer pieces in this manner in large supply, and sell by the weight. We have found two companies that offer such services, and will personally visit each during the prototyping phase to see which can best suit our needs.

The first of these two companies is Alro Metals, which is located locally in Orlando in the Winter Park area. This company is known through networked sources of a group member, and sells drop pieces of metal. We discussed our needs with this source, and received an estimated price of a maximum of $40 to meet our needs, with the possibility of having the metal donated to our project all together. Since no members of our group are mechanical engineers, consulting on our design is something we are very interested in, and our source to Alro Metals is willing to help with this.

The second company is Acme Industrial Surplus. This company is also local, although, being located in Sanford, is away than Alro Metals. Acme is a surplus provider that sells metal by weight, and is an inexpensive option for our group. Our group contacted Acme for a price quote on the estimated amount of metal that we will need, and received an answer of $25. If metal can not be acquired through donation through another company, this will be our cheapest alternative, and where we will purchase the metal for the base.

3.2.7 Arduino Uno Microcontroller

The Arduino Uno Microcontroller that we will be implementing into our prototype design will be obtained from SparkFun. The microcontroller SKU number is DEV-09950 and as of right now the cost of the board is $29.95 and will require shipping costs. Their website states that you first have to create an order and then you have to fax your purchase order to their fax number we have to reference our SparkFun order number as well. The second constraint for shipping the microcontroller is that they will not ship your products until the vendor has received both the purchase order and order number. It will take three business days from receipt before any processing is completed and the vendor will be able to ship.

3.2.8 Fabricated PCB

The Fabricated PCB that we will be implementing into our prototype design will be obtained from Advanced Circuits. We will have to provide the design of the PCB to the vendor; the vendor will construct the PCB, and will ship the PCB back to our group. As of right now the cost of the PCB design is estimated at $33.00 for every PCB and we are going to obtain 2 to work with for a total cost of $66.00. **[16]**

Arduino Board Code Design

We remain open to using a PIC if needed in prototyping due to the simplicity of the circuit and the well documented history of using them to control servo motors. However, we still plant to build a custom made Arduino for our design. While the hardware is more complex, the coding is much simpler in the Arduino environment. Powerful libraries exist to help with various tasks, and for our design we will be using the Arduino Servo Library. With our microcontroller, up to 12 servos can be controlled using this library, which is more than adequate for the needs of the design. Below is a brief overview of the Servo Library's functions that we will need.

Functions

- attach() : Attach servo variable to a pin
  - Syntax: *servo*.attach(pin)
  - Alternate: *servo*.attach(pin, min, max)
- write() : writes value of angle (int) in degrees to servo
  - Syntax: *servo*.write(angle)
- writeMicroseconds() : writes a value of angle(int) in µS to servo
  - Syntax: *servo*.writeMicroseconds(µS)
- read() : Read the current angle of the servo variable
  - Syntax: *servo*.read()
- attached() : Check if servo variable is attached to a pin
  - Syntax: *servo*.attached()
- detach() : Detach servo variable from its pin
  - Syntax: *servo*.detach()

Figure 15. Arduino Servo Library Functions [17]

The above library is the only thing needed to control servos in the Arduino environment, unlike the complex delay or timer/interrupt sequences in the PIC pseudocode previously discussed in **Figure 15.** The method for controlling a servo with this library is to simply follow these steps: declare each servo as a variable, attach each servo, write positions to each servo.

To communicate with the computer using the Arduino, a second library is needed. The Serial library allows communication between the microcontroller and serial port on the computer, thus allowing the computer to control the servos, alarm, and any other attached devices, as well as read the battery voltage. Meet this requirement, the Serial Library will be used, which lets us send and receive serial data on the Arduino as character data or in bytes. Below in **Figure 16** is a brief overview of the Serial Library's functions.

Functions

- begin() : Sets the data rate in baud for serial data transmission
    - Syntax: Serial.begin(speed)
- end() : Disables serial communication
    - Syntax: Serial.end()
- available() : gets the number of bytes available in the serial buffer (serial buffer holds 128 bytes)
    - Syntax: Serial.available()
- read() : Reads incoming serial data
    - Syntax: Serial.read()
- flush() : Flushes all data in the serial buffer
    - Syntax: Serial.flush()
- print() : Prints data to the serial port as readable ASCII text
    - Syntax: Serial.print(char)
    - Alternate: Serial.print(byte(ASCII value))
    - Alternate: Serial.print(char, format)
        - Format specifies the number base, or number of decimal places displayed
- println() : Prints data to serial port as readable ASCII, followed by a carriage return
    - Syntax: Serial.println(char)
    - Alternate: Serial.println(char, format)
- write() : Writes binary data to the serial port, sent as a byte or series of bytes
    - Syntax: Serial.write(val)
    - Alternate: Serial.write(str)
    - Alternate: Serial.write(buf, len)

Figure 16. Arduino Servo Library Functions [18]

Microcontroller Coding IDE

To program the microcontroller, the Arduino IDE, version 0021, will be used. This IDE is free from the Arduino website, and open source. It runs in Java, and can be used on Windows, Linux, or Mac operating systems. This IDE allows for simple, straightforward programmig of our Arduino board, without the use of a special programming tool. Although the IDE is still in an alpha stage of development, it is well documented and reliable. It comes built in with a compiler, and a variety of examples covering many of the most common Arduino applications. The code used on the board is an implementation of Wiring, which is similar to C. This makes for a a comfortable coding environment, where mistakes can be minimized. Software written in this IDE are called “Sketches”, and create files with no extension. There are options choose the board type that is being used in the setup, as well as monitor the data being sent over the serial connection to the board. Once the code is complete, it can be compiled and programmed to the board using the buttons on the toolbar. Below is a screenshot

of the Arduino environment, with the version information displayed in **Figure 17** over the toolbar, text editor, and message window.

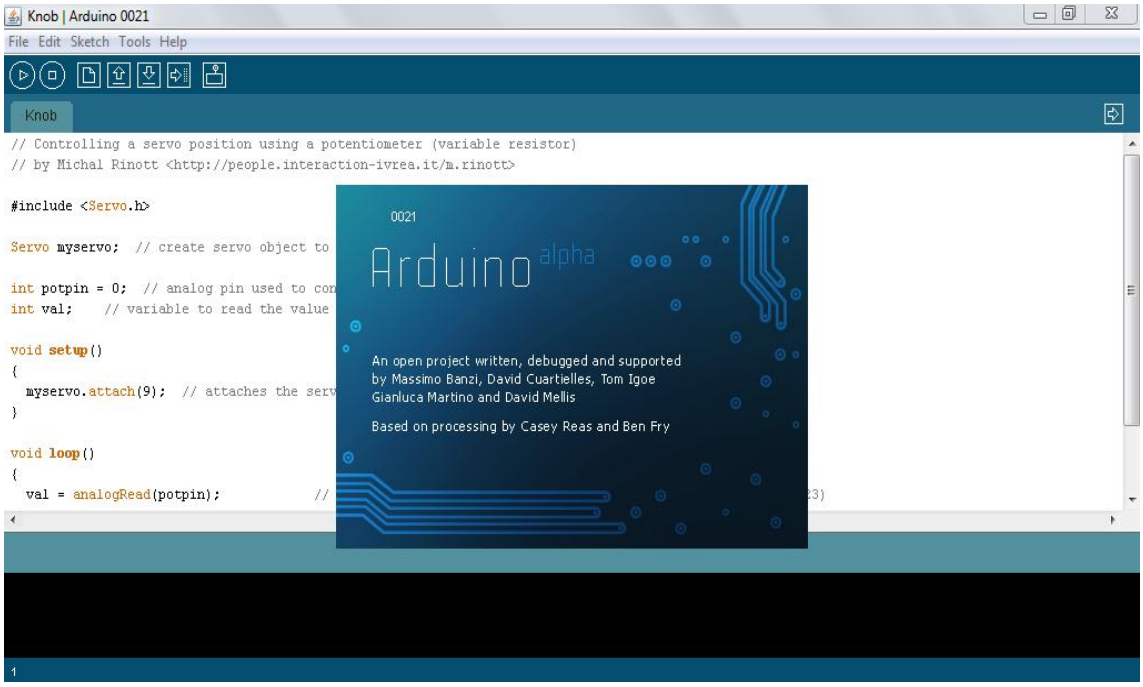

Figure 17. Interface of Arduino IDE
Copyright Arduino. Reprinted under Fair Use

3.3 Custom Arduino Board Design

Below in **Figure 21** is the schematic for the Arduino microcontroller board we will build. It uses the ATmega328 chip that is explained in further detail above. The board uses two voltage reglators of 5 volts and 6 volt. These voltage inputs are for the board power and servo power, respectively. The board power regulator is a standard LM7805 regulator (Cost of $6.00), which can take an input of 7V to 20V, and output 1 Amp. Selected details from the datasheet for this part are below in **Figure 18.** This regulator might not support 3 servo motors with only 1 Amp output, since each servo draws up to 600 mA; however, 1 Amp will be more than sufficient to power the board. The 6 volt regulator is a LM338T, which is an adjustable voltage regulator capable of output 5 Amps. Selected details from the data sheet for this part are below in **Figure 19.** The input voltage can be between 4.2 and 40 volts, but should be above 10V for our application. This will be able to drive the 3 motors, as well as other devices that may need to be added in the future. It's adjustability also lets the output voltage to be changed, should different motor voltages be needed at a later time. This voltage regulator is not a standard part, but can be aquired from National Instruments in a quantity of 5 by sampling the part

Electrical Characteristics (LM7805)

Refer to the test circuits. -40°C < T_J < 125°C, I_O = 500mA, V_I = 10V, C_I = 0.1µF, unless otherwise specified.

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
V_O	Output Voltage	T_J = +25°C	4.8	5.0	5.2	V
		5mA ≤ I_O ≤ 1A, P_O ≤ 15W, V_I = 7V to 20V	4.75	5.0	5.25	

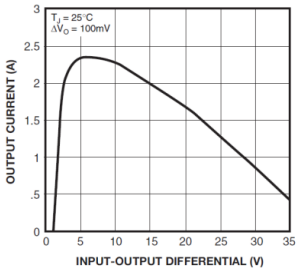

Figure 18. I/O voltage parameters and graph of peak output current for LM7805
Copyright Fairchild Semiconductor Corporation. Reprinted under Fair Use

SYMBOL	PARAMETER	CONDITIONS	NOTES	PIN-NAME	MIN	MAX	UNIT
Rline	Line Regulation	3V ≤ Vdiff ≤ 35V			-3.5	3.5	mV
		3.3V ≤ Vdiff ≤ 35V			-14	14	mV

Figure 19. voltage parameters for LM138/LM388
Copyright National Semiconductors Corporation. Reprinted under Fair Use

Below in **Figure 21** is the schematic for the Arduino microcontroller board we will build. It uses the ATmega328 chip that is explained in further detail above. The board uses two voltage reglators of 5 volts and 6 volt. These voltage inputs are for the board power and servo power, respectively. The board power regulator is a standard LM7805 regulator (Cost of $6.00), which can take an input of 7V to 25V, and output 1 Amp. This regulator might not support 3 servo motors with only 1 Amp output, since each servo draws up to 600 mA; however, 1 Amp will be more than sufficient to power the board. The 6 volt regulator is a LM338T, which is an adjustable voltage regulator capable of output 5 Amps. The input voltage can be between 4.2 and 40 volts, but should be above 6V for our application. This will be able to drive the 3 motors, as well as other devices that may need to be added in the future. It's adjustability also lets the output voltage to be changed, should different motor voltages be needed at a later time. This voltage regulator is not a standard part, but can be aquired from National Instruments in a quantity of 5 by sampling the part.

The V_SENSE line in the schematic is the output of a circuit that is a simple, cost effective way to sense the output voltage of the battery input, and therefore also

estimate the charge of said battery. The technique used is a simple voltage divider, and using Ohm's Law, the value of the two resistors can be calculated. The first parameter for these values is to use a total resistance that drains the least current possible, while still overcoming noise. For our circuit we will use 0.1 mA. For the sake of these example calculations, we will choose the max output of the battery to be 12V. To achieve 0.1 mA current drain, the two resistors R_{10} and R_{11} should equal 120 kΩ total. Next, the ratio of the two resistors need to be calculated. The output of the voltage divider is attached to an analog input pin on the microcontroller, so this voltage must be below 5V, the maximum allowed on this pin. To be safe, we choose the output to be only 4V. By the voltage divider rule, the values of R_{10} and R_{11} should be 40 kΩ and 80 kΩ respectively. This creates a ratio of 3.33 between the output voltage and the battery voltage. Knowing this, the voltage of the battery can be sensed using code on the microcontroller. The following pseudocode shows the technique used to measure the battery voltage, and thus also estimate the battery charge.

```
start
        int pinVolt
        int battPin      //number of analog pin V_SENSE uses
        use Serial.begin(baudrate) to start comm to PC
        loop
                use analgoRead(battPin) to get pinVolt    //this is an integer ratio
                multiply pinVolt by 0.00488 //ratio of integer to real voltage on
analog pin
                multiply answer by 3.33 to get battery voltage
                send value to computer by serial
                delay() to slow sampling rate
        end loop
end
```

The POWER label in the schematic is a 4 pin header that is meant to be used for developmental purposes. This pinout gives the user access to all parts of the board power. The +5V can be used to power external devices for testing, as long as their total current drain does not take too much current away from the microcontroller. The VIN pin can be used to power the board from an alternate power supply if needed, and leads into both voltage regulators. Two ground pins are used so that two pin connetors can be used on this pinout if wanted. The POWER1 pinout serves a similar purpose to the POWER pinout, being used for development purposes corresponding to the servo power.

The pinout section ICSP is a special pinout meant as an alternative means of programming the ATmega328. ICSP stands for In Circuit Serial Programmer. These pins are configured so that they can be hooked directly to an AVR-ISP (in-system programmer), USBtinyISP, or a self made Parallel Programmer. The reason for using this is it gives the ability to program the microcontroller without having the bootloader on it. This frees up about 2 KB of flash memory, and

allowing for more code to be put on the chip. We should have plenty of room on the ATmega328 chip, so memory will not be a concern. However, this provides an alternative to programming through serial, should problems arise. An AVR-ISP and USBtinyISP are both purchasable options that are inexpensive, and Parallel Programmers are not too complex to build ourselves. The ICSP will most likely not be needed in our design, but if the designed were to be changed in the future to use a different ATmega chip then it will prove useful. This is looking towards future possibilities, and a fallback safety**[19]**.

Q1 is an oscillator. This is connected to both oscillator inputs on pin 9 and 10, is used for the clock, and needed to create the PWM in our motor control. The oscillator needs to be 16 MHz crystal, with a load capacitance of 18-20pF. This is a very common part, and any that fit these specifications may be used. These cost between $0.50 and $1, and we will use one that is through hole mounted (for ease of assembly) and small form factor. The specific part we will use will be the FOXSLF/160-20 16MHz 20pF capacitor, because it is known to be compatable with our ATmega328 chip.

X1 is the DB9 serial connector that leads to the computer. The main method of programming the ATmega328 is through this serial connector onto pins 2 and 3. Also, communication of commands to and from the computers will be through this serial connection. While the standard for Arduino boards is serial communication, most modern computers, including the netbook we will be using, do not have serial ports. For this reason, a conversion from serial to USB will be needed. Most Arduino boards have a serial to USB conversion on board, using a FTDI chip or equivalent. We will not do this, to save board space and cut down on costs. Off board serial conversion is the best method for our use, and can be done with an inexpensive USB to serial cable. These cables cost between 10 and 15 dollars, and are easy to aquire.

J1, and J3 are the Digital I/O pins for the microcontroller, and J2 are the Analog In pins. These pin headers are an Arduino standard, organized in the form they are displayed. When the board layout is designed and built, the spacing of these pin headers will be set to the standard of regular sized Arduino boards. This will allow the board to be expandable, with Arduino accessories able to be attached to it in the future. This gives access to every usable pin on the microcontroller, allowing additional devices to be attached in the future if the design is expanded.

SERVO1-3 are breakout pins for the attachment of our three servo motors. This will make attaching the servos to our control board much neater and more clearly labeled. Each servo has 3 pins: one for the PWM signal, one for motor power, and one for GND. These pins on a normal microcontroller board are inconveniently spaced apart, while the servo uses a single 3 pin connector. The ony disadvantage of these breakout pins is the fact that they occupy their corresponding digital out pins when the servo loads are connected, and the user must have knowledge of this to avoid conflicts. The Alarm connector is of similar

use, and is a common molex connector. This connector will be created and soldered to the alarm, so that it can be controlled like the servos. Below in **Figure 21** is the schematic of our microcontroller design, followed by **Table 7** which labels the parts for reference.

ATmega8 is the microcontroller chip that is the center of the design. For our design we will be using an ATmega328, but any 8 bit ATmega microcontroller may be used in its place, with the proper pin connections made. Below in **Figure 20** is the pin layout of the ATmega328, a 28 pin microcontroller that is used in arduino boards. This chip has 32 working registers, 23 general purpose I/O lines, 3 timer/counters, and a 2 wire serial interface, among other features. The chip also features 32K bytes of programmable flash, 1K bytes of EEPROM, and 2K bytes of SRAM. It has 6 Analog input lines, and14 digital I/O lines, 6 of which can be used for Pulse Width Modulation. Since we will be using only 3 servo motors, these 6 PWM lines will be more than enough. The first 3 PWM pins are on pins 15, 16, and 17 where it is labled OC1A, OC1B, and OC2a respectively. These will be the pins we will attach the servo motors to.

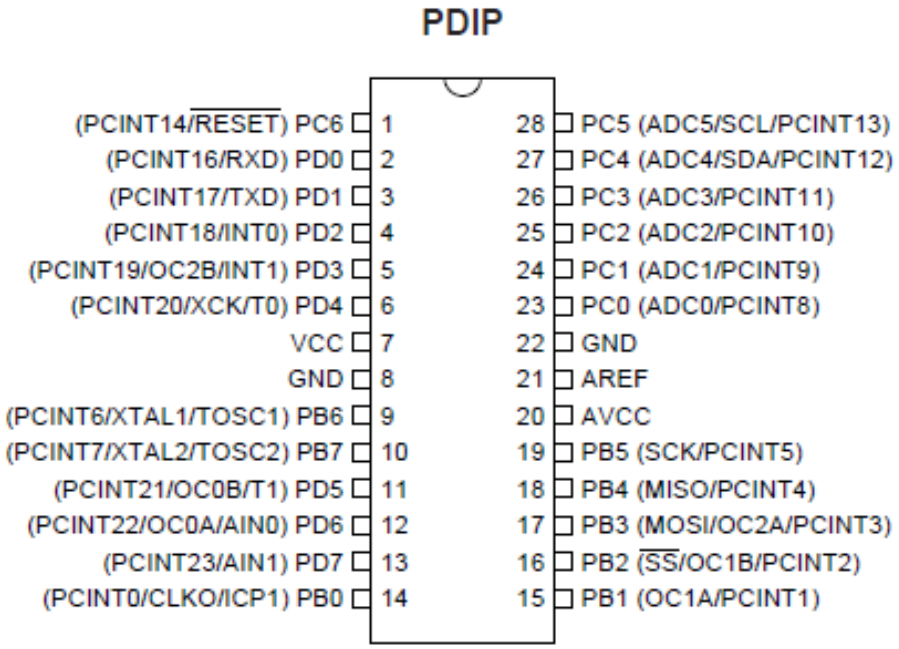

Figure 20. Pin configuration for ATmega328 chip. (From atmel.com) Copyright Atmel Corporation. Reprinted with permission.

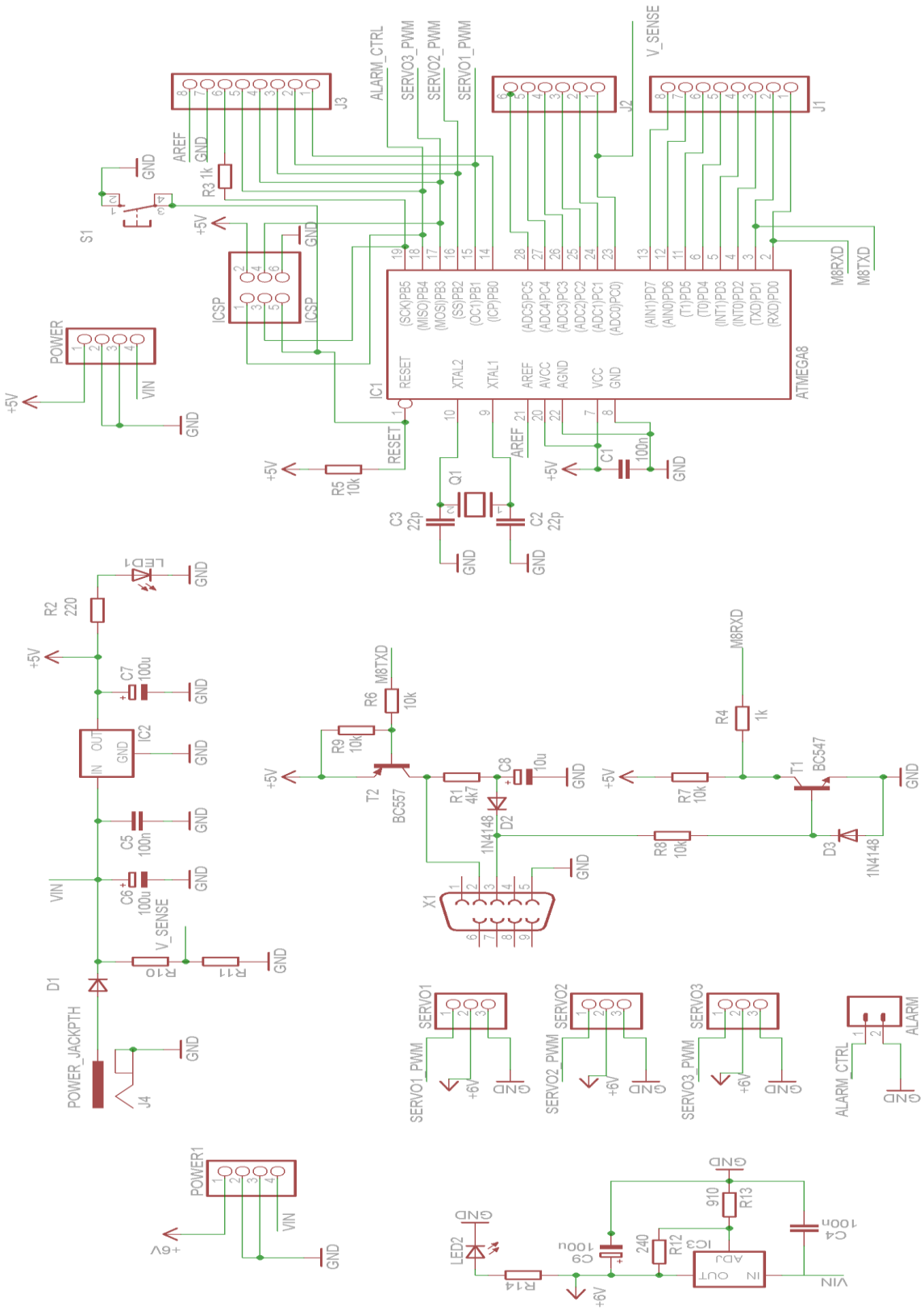

Figure 21. Schematic for custom Arduino board

PART CODE	Description
POWER_JACKPTH	Standard 5.5mm barrel jack for main power from DC converting wall wart or battery
D1	Diode to prevent negative current if power is hooked up incorrectly
V_SENSE	Voltage divider circuit to sense battery voltage
IC2	7805 Voltage regulator to obtain 5V board power at 1A max
LED1	LED to indicate board power is on
IC3	LM338T Voltage regulator to obtain 6V servo power at 5A max
LED2	LED to indicate servo power is on
S1	Reset Button
SERVOx	Breakout pins for servo motor connections
X1	Serial connection from computer
Q1	Oscillator
ICSP	In Circuit Serial Programmer
ATMEGA8	8 Bit ATmega microcontroller. Any 8 bit chip will do, but we will use a ATmega328

Table 7: Key for labels in Schematic Figure 21.

3.3.1 Printed Circuit Board Layout

For the layout of the printed circuit board, we used the free version of Eagle. Because of this, the size of our board was restricted to 4 x 3.2 inches. We found this to be plenty of space for out schematic design, and chose to use all through hole parts so that the board could be assembled by hand quickly. Because of the possibility of 5-7A current in certain traces on the board, space became a problem in certain sections. To remedy this, a power plane was used, as well as a ground plane. For developmental purposes, all of the microcontroller pins have pin outs, as well as the pin outs for the ICSP if programming via the DB9 port fails. Eagle's auto-route function was used for most of the traces, with the remaining traces routed manually. Below is the layout of the board.

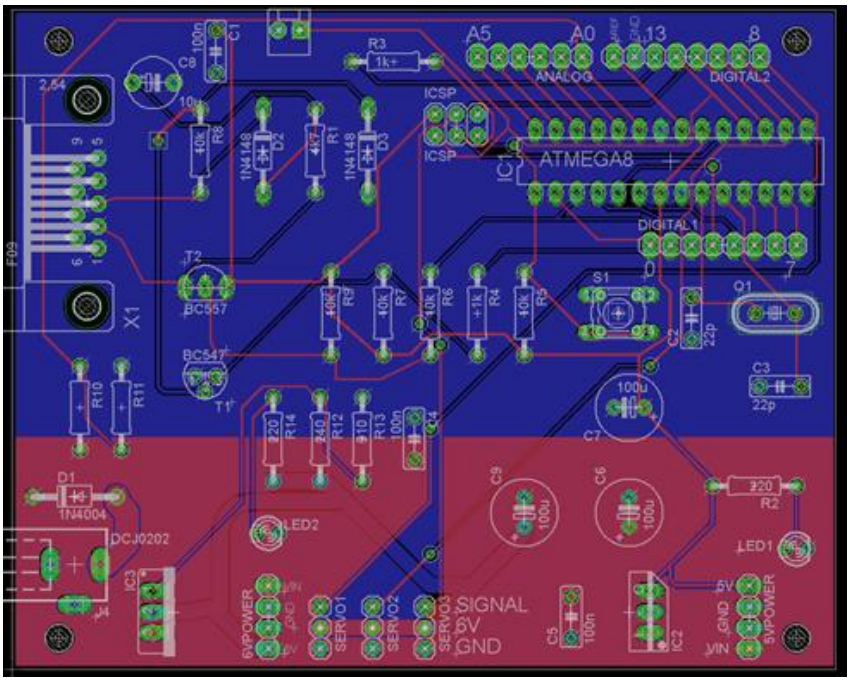

Figure 23. Layout of printed circuit board

3.3.2 Hardware Power

To power our hardware systems, several power sources of different voltages will be needed. The computer, control board, motors, rangefinder, alarm, and cameras will all need power from battery sources, to allow the system to be set up in remote locations. The computer and the rangefinder each have their own battery sources built into their systems, with long battery lifes that will satisfy the requirements. The cameras will draw power through USB from the computer, needing no external battery source. This leaves the control board, servo motors, and alarm to be powered by a separate battery system.

The servo motors and alarm are powered through circuits embedded in the control board, with the need for an external battery. The power circuit on the board has two regulators that power the microcontroller, alarm, and servo motors. These regulators need a minimum voltage input to function. The first regulator that leads to the board power line is a LM7805, which outputs 5V at a max of 1 Amp. This voltage regulator requires an input between 7V and 25V. The second voltage regulator leads to the motor power line, and is an LM338. This is an adjustable voltage regulator, that requires the input voltage to be at east 3V higher than the output voltage. Our output from this regulator will be 6V at a maximum of 5A, and thus the input voltage must be at least 9V. With the requirements of both regulators in mind, a batter above 9V is needed. Another factor that must be taken into consideration is the nature of battery outputs. At full charge, batteries output at a voltage above their specified rating, and when low on charge they can fall below their specifications. Also, depending on the battery

capacity and maximum discharge rate, this output can be very unreliable under a heavy load. For this reason we will need a battery strong enough to not only meet the voltage requirements, but also the current requirements.

Although there are two inputs, it would be more convenient to have only one battery for the system. This means that both requirements of power input need to be met with one battery. The first thing to consider is the voltage of the battery to use. Since the first input needs at least 7V input, and the second needs at least 9V input, a standard battery that is above those ratings should be used. The upper bounds of the input voltages for these regulators are 20V and 30V, respectively. This means that the voltage input must be in a range between 9V and 20V, but because of the nature of batteries, the input should not be near the minimum and maximum values. The most common battery type that fits these voltage specifications is 12V, which come in a variety of powerful batteries. The next specification that needs to be considered is the capacity of the battery, measured in Amp-hours. This means how long a given current can draw from the battery, which also means how high of a current can be drawn before the battery no longer can meet output specifications. Our circuit at maximum output is estimated to draw 2 to 2.5 Amps.

Given a standard, inexpensive 12V battery with a rating of 4 to 5000 mA-hours capacity, the motors should last a minimum of 2 hours. Since 12V batteries come in a wide range of capacities, the system can be greatly expanded to meet higher longevity needs. With a very inexpensive battery the system could run at full speed for 2+ hours, and therefore with normal use 3-5 hours. Car batteries output 12V and could be easily used as a power source for the system, and have extreme longevity. Car batteries often have a capacity of around 100 Amp-hour, and could therefore be used at maximum draw for almost 50 hours. This would let the motors run in normal use for days without the need of charge. This expandability leads to another possibility for implementing the turret system: vehicle attachment. With the computer and sensors able to be run off of wall power that can be obtained through car compatible power inverters, and the motors able to run off of 12V, the system could be easily attached to a automobile or any other vehicle type with a suitable 12V battery. Our group has found a 12V lead-acid battery with a capacity of 8 Amp-hour on Amazon for under 20 dollars which would perfectly suit the needs of our prototype.

3.4 Software Components

3.4.1 C# 4.0

For the client side of the project it is necessary to write something that is fast and lightweight and can handle video processing in real time. C# was chosen because of its relative speed compared to Java and its open libraries for video processing. These libraries allow for real time video processing and provide tools for motion detection and tracking. Other lower level languages such as C were considered, but ultimately passed on because of their relative difficulty of use, their lack of support for object oriented programming, and their lack of open source video processing libraries. C# provides a robust object oriented programming language similar to Java. Because of this, high-level, modular code can be written in a way that promotes code readability and reusability.

C# is part of Microsoft's .Net framework and is available through Visual Studio. C# 4.0 is the latest version of the language, and will be used for development. **[20]**

3.4.2 AForge.NET Framework

The Aforge.NET framework will be used for video processing to locate and track targets using the onboard webcam. AForge.NET is a framework for C# that contains many useful open source libraries for image and video processing, computer vision, and machine learning. The AForge.Video library will be utilized to create the motion detection algorithms.

3.4.3 Visual Studio 2010

There are not too many options if one is looking for a full fledged .Net IDE for C# programming. The two main options are Microsoft's own Visual Studio and MonoDevelop, based on the open source C# compiler Mono. While MonoDevelop is open source and now is fully supported on Microsoft Windows, Visual Studio has better integration with the .Net Framework and is much more widespread. Because of this, the group members have much more familiarity with Visual Studio. On top of that, Visual Studio is better supported and has more documentation online. For these reasons, Visual Studio 2010 was chosen as the .Net IDE for the project.

Microsoft's Visual Studio 2010 is the latest release of Microsoft's IDE package. It provides support for numerous programming languages such as Visual C++, Visual Basic, Python, Ruby, and others. For the purposes of the project, it will be utilized to write C# code which comes standard with Visual Studio. Since C# and Visual Studio are both developed in close conjunction at Microsoft, using Visual

Studio to write C# code is extremely easy and streamlined. It provides the basic functionality of a modern IDE such as syntax highlighting and an excellent auto completion tool named IntelliSense. It also includes a form designer for easily making GUI interfaces and a class designer for designing and creating object oriented code. **[21]**

3.4.4 Java 6.2 SDK

In order to satisfy the server side software requirements and maintain a high degree of independence from the configuration of the server, the JAVA programming language will be used. The latest version of Java 6 SDK (currently 6.22) will be used for development in order to keep discovered security holes to a minimum and maximize functionality.

An advantage of using Java is a wide array of expandable optimized built-in functions and methods that are not included in lower level programming languages such as C or C++. Because Java is widely used a large number of 3rd party software libraries are also available both as paid libraries and open-source libraries at which can be used to decrease the development time and cost of the server software. Java is also available on any major Operating System meaning that it can run on either a Linux or Windows server.

Java also allows the use of Object Oriented programming. Object Oriented programming provides writing code that is modular and simplistic for manipulating independent data pieces. Object Oriented software design also can be designed in class diagrams where the data pieces and functions correlate to the object that they correspond to and provide an easier to understand codebase.**[22]**

3.4.5 MySQL

A persistent storage system is necessary to maintain the data received from the turrets. This requires some sort of database that may be queried remotely. Options included MS Access, an implementation of SQL, or a NoSQL database system such as Cassandra. In the end, SQL was chosen because it is an open format and because it is the most popular database system for web applications in the world. Once we had settled on SQL there were many implementations that had to be chosen from. MSSQL, ProstgreSQL, and MySQL where all considered. MySQL was chosen because of its open source nature and its ubiquitous nature on the web (it is used by websites such as Facebook and Wikipedia).

MySQL offers a well known and widely used database application by which the persistent information can be stored. It uses SQL in order to query the data and functions to perform a number of aggregate functions that are optimized for large number of results.

By using SQL it allows the use of 3rd party software libraries in order to simplify its integration into the software that is developed. These software libraries also include database design as well as Object relational mapping libraries if necessary. **[23]**

3.4.6 Python 2.2

Python is a commonly used web-development programming language and allows the use of some highly developed web-app frameworks. Python is similar to Java in that it is executable on common Operating Systems. Python's syntax however is more attributed to its strict formatting style and significant whitespace unlike Java. Python uses an open-source library meaning it's free to use and distribute. Python's standard library is also very extensive and powerful so we will be able to focus on writing usable software instead of rewriting helper methods. **[24]**

3.4.7 Django

Web application development is commonly time consuming. Because of the time-constraints of the development time-frame, Django was selected as the web-app framework for the project. Django offers a high-level framework with a wide array of functionality to decrease development time of web applications. Django web applications are written in Python. It is open-source meaning that it is free to use. Django also offers a set of functionality for developing and maintaining a SQL database design. The design is specified in its object relational mapping and generates the SQL in order to generate the database tables and columns. **[25]**

3.4.8 Java Eclipse

Since Java is such a popular programming language these days and is an open standard, there are a variety of integrated development environments to choose from. IDEs that were familiar to the group, free, and well supported where reviewed. NetBeans is a Java IDE that is written by Oracle. It provides support for other languages such as Ruby and Python. jGrasp is another IDE that was considered. It is lightweight and can automatically create Control Structure Diagrams for programs written in Java, C, C++, and Objective-C to easily visualize the flow of complex programs. The Java IDE Eclipse was eventually chosen for its familiarity and its status as one of the most popular Java IDEs.

The Eclipse IDE for Java offers an excellent editor for Java projects. The built-in functionality for refactoring and generating source code simplifies the development and maintenance of large code projects. Another advantage to using Eclipse allows for easy maintenance of the classpath, an ant plug-in for simplifying build procedures. Eclipse also offers plug-in support to extend the basic IDE functionality. Search functionality is provided for basic text search for specific types of files, and a java specific search to search for references of specific methods.

The highly customizable nature of Eclipse provides functionality to set up code formats and syntax to maintain the same syntax and formatting very easily between different users for all source files. The editor provides syntax highlighting in order to make the source files easier to read by differentiating between constants, variables, classes, operators, etc. Besides Java files, Eclipse also provides support for XML, JavaScript and html files.

The PyDev plugin for Eclipse makes it an excellent IDE for writing python code as well. The plugin features Django integration, allowing one to easily create Django applications and run/debug programs as Django applications. It also features syntax highlighting, code completion, and other basic features one would expect from a Python IDE. **[26]**

3.4.9 Arduino IDE

The microcontroller was programmed in the Arduino IDE using the Wired programming language. Wired is a custom microcontroller programming language based on Processing that is similar to C++ and easy to use. There is built in support for servo controls which makes reading the current location of servos and sending them new instructions trivial. Writing to the analog and digital pins is just as simple. As an extra bonus, programming the PCB and the Arduino UNO is identical, so early development was able to proceed before we received the final PCB. **[27]**

3.4.10 Apache Subversion

Source and revision control is an important tool when dealing with any large project with multiple members. The ability to work on files and code concurrently and upload them to a centralized location removes a lot of the hassles from collaboration using more antiquated techniques such as mail groups. The ability to go back to previous revisions and track progress make it an invaluable tool. The two most popular forms of source control today are Subversion and Git. The main difference between the two is that while Subversion is a centralized revision control system, Git is distributed. This means that on Subversion there is one canonical version of a file on a central server and on Git there are many working files that each are considered to be their own repository and can be exchanged in a system similar to peer-to-peer networks. Subversion was chosen because of the small number of students in the group means that the client-server model of Subversion works best. Also, Subversion has built in support for the IDEs we are going to use such as Eclipse.

3.4.11 TortoiseSVN

Configuration management is a useful tool for maintaining a number of files between multiple users and tracking the history of files. TortoiseSVN offers a subversion implementation to maintain the baseline files on a web server, and allows the users/clients to download the baseline, make changes and commit the changes to the server. TortoiseSVN also offers automatic merging in order to handle simultaneous updates from different users. TortoiseSVN also provides resolving conflicts when automatic merging is not possible.

At the time of committing new or updated files the user can add a comment which is available to all users viewing the history of updates on files. These comments can aid understanding the reason and nature of the changes being made to the files. **[28]**

3.4.12 WinMerge

As part of configuration management, it is commonly helpful to view the changes between one version and the next. Default comparison tools are often inadequate and hard to understand. WinMerge offers an easy to use editor to display changes between different versions and conflict files. WinMerge also offers an implementation directly into TortoiseSVN in order to be used when comparing different versions. It can also be manually set to be used as the default comparing application in Eclipse for all different file types. **[29]**

3.5 Software Design Process

For the coding of the software side of the autonomous turret, a form of agile software development that emphasizes rapid prototyping and revisions was utilized. While there was significant planning on the overall structure of the program and data structures that will be used, emphasis was on getting working code operational as soon as possible and making quick revisions and new versions that improve performance and functionality. This required the coding team members to work together and communicate frequently as changes and new ideas are made. As soon as updates are made, testing was done on any new features and the changes were tweaked until satisfactory. A simple overview of the agile development philosophy can be seen in **Figure 22**:

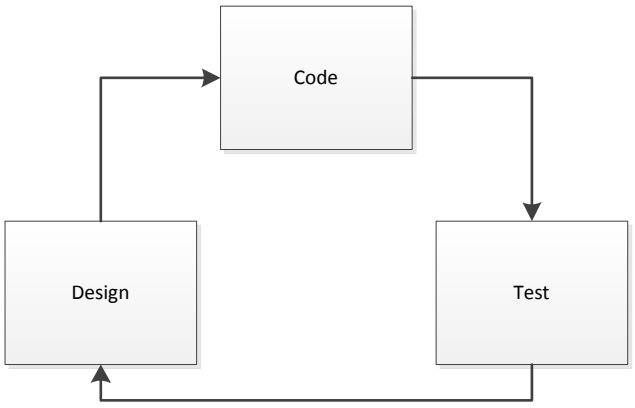

Figure 22. Agile Software Development

3.6 On-board Software Design

The on-board server software is responsible for controlling the turret, automated targeting and offering a manual control override for the turret. To accomplish these tasks, the system is broken down into 4 primary components. The first component is the Turret Controller which provides a C# interface to the embedded microcontroller for directly controlling the state of the turret. This component is also responsible for parsing the data from the microcontroller in order to keep track of the active state of the turret. The second component is the Turret Data Distributor which is a C# thread that connects to an active socket connection on the server in order to send the current and historical information of the turret to the server to be persisted. The third major component is the automated targeting system. This component is responsible for utilizing the cameras and range finder that are attached on the turret to locate moving targets determine if they are either in the warning range or firing range and act accordingly. This component then sends commands to the Turret Controller to move and fire the Turret. The last major component is the Manual Control Server which offers an active socket connection for the off-board server to connect to when it wishes to take manual control of the server. Coupled with the Manual Control Server is the Flash Video encoder and streamer which encodes the gun-mounted video stream and offers it to the server to view what the turret is seeing when assuming manual control of the server. This interplay of components is shown in **Figure 23**.

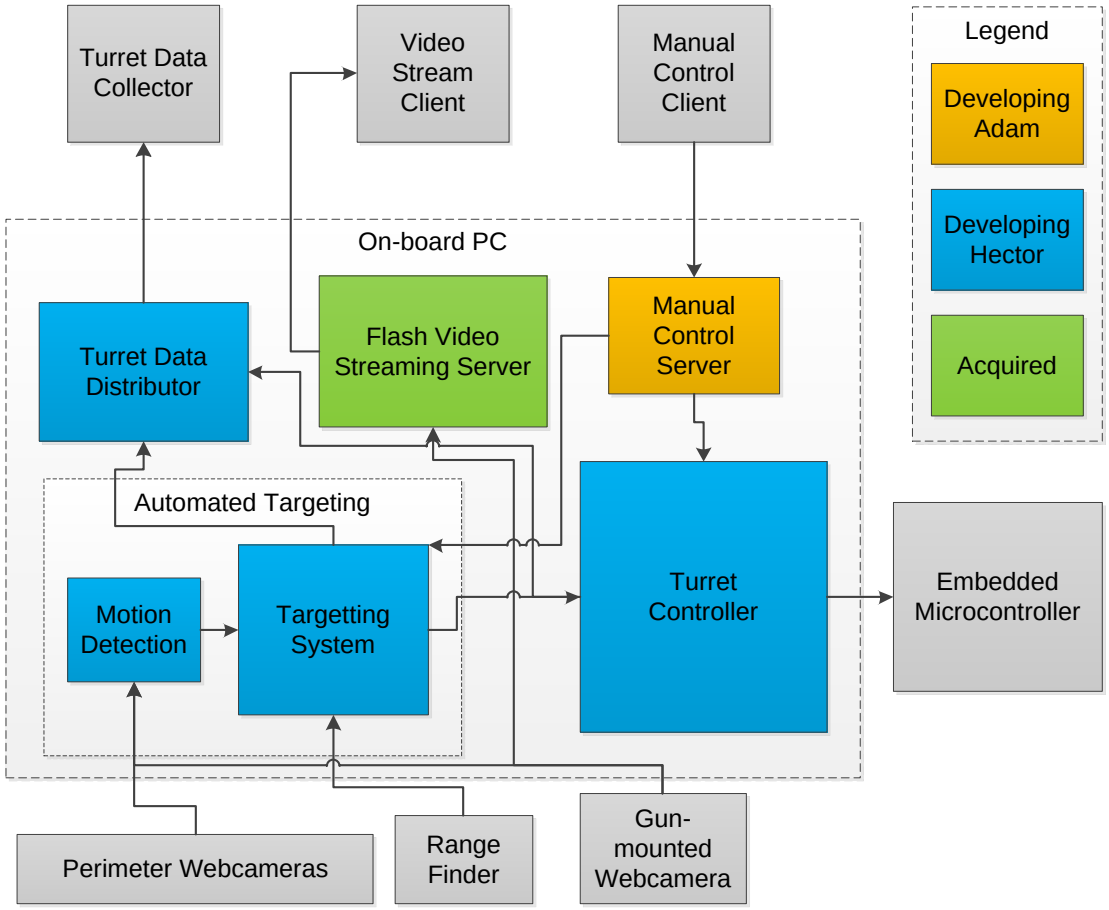

Figure 23. Block diagram of the general on-board software design.

3.6.1 AForge.Vision.Motion Library

The AForge.NET library we used contains many helpful methods for video processing. These libraries were called extensively for our motion detection algorithm and targeting system. A list of the classes we will use to build our modules follow:

BlobCountingObjectsProcessing
Motion processing algorithm, which counts separate moving objects and highlights them as rectangles on screen.

CustomFrameDifferenceDetector
Motion detector based on difference with predefined background frame.

GridMotionAreaProcessing
Motion processing algorithm, which performs grid processing of motion frame.

MotionAreaHighlighting
Motion processing algorithm, which highlights motion areas.

MotionBorderHighlighting
Motion processing algorithm, which highlights border of motion areas.

MotionDetector
Motion detection wrapper class, which performs motion detection and processing.

SimpleBackgroundModelingDetector
Motion detector based on simple background modeling.

TwoFramesDifferenceDetector
Motion detector based on two continues frames difference.

3.6.2 Motion Detection and Target Acquisition

The motion detection and targeting component of the turret‘s design is responsible for taking the input from the camera and rangefinder and using that information to acquire targets and their location. The motion detection algorithms identify targets by using video processing to calculate differences between frames, while filtering out any background changes. It also estimate the targets center of mass. Then, by using the targets size and position in the video feed as well as the viewing angle of the proximity camera, it estimates the target’s location relative to current crosshairs of the turret. It then sends this information to the turret controller module, which handles turret movement.

The activity diagram in **Figure 24** below tracks the process that the motion detection and target acquisition modules go through when detecting a new target. The Motion tracking first detects motion and sends this data to the targeting module. If it is a new target, the alarm will sound to warn the target and the turret will begin tracking the target. The turret will always track the largest target in sight. If after 5 seconds the target is still in view, the turret will fire upon its location. The turret will continue tracking the target and firing every 5 seconds until the target leaves the vision of the turret or stops moving. If the target is outside the range of the motion tracking of the turret, it will assume that the target has fled or is neutralized and will release the target. Any motion that it detects afterward will be treated as a new target and the cycle repeats.

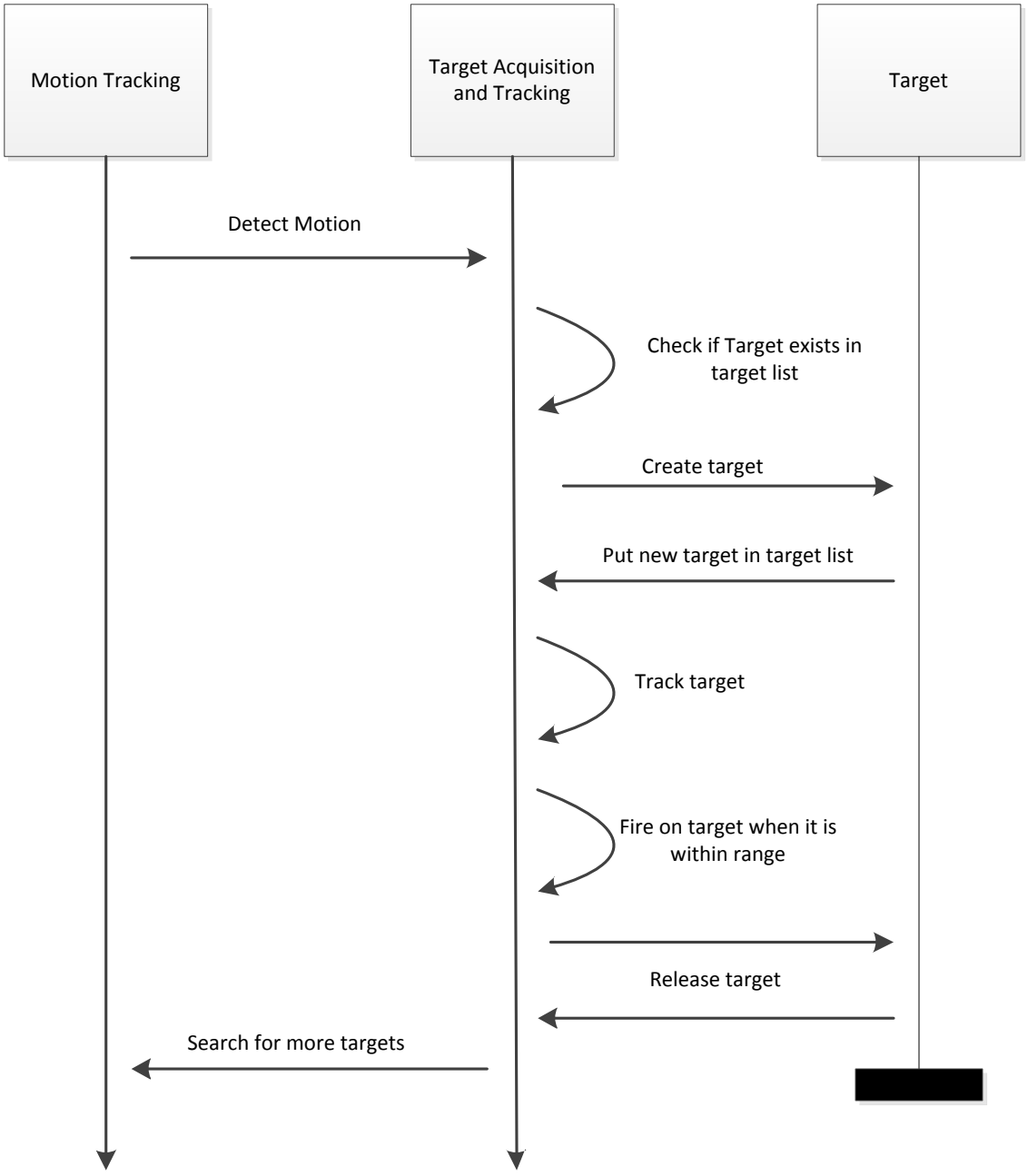

Figure 24. Target Acquisition Activity Diagram

More complicated cases include situations when there is more than one target being tracked. In this case, the largest target will always be tracked and the turret will classify them as being the same engagement.

3.6.3 MotionDetector Class

The first part of the module is the motion detector class. It does the video processing and calculations to determine if there are any targets available. It is

also the main Form class of the program and is responsible for displaying the onboard GUI. Below is a summary of the methods that will be implemented. The class diagram is displayed below in **Figure 25**.

3.6.3.1 Variables

stream : videoStream
The current video stream of the web camera.

background :bitmap
The previous frame of video

currentFrame : bitmap
The current frame of video

target : Rectangle
The highlighted rectangle that appears on screen. It represents targets found by the motion detection and blob detection.

hasTarget : boolean
A Boolean flag that keeps track if there are any targets currently on screen.

target : Rectangle
The highlighted rectangle that appears on screen. It represents targets found by the motion detection and blob detection.

turretpos : double
The current position of the turret as previously instructed by the last frame.

3.6.3.2 Methods

NewFrame(sender : Object, image: Bitmap) : void
This method does the most of the work for the automated controls. It is called whenever there is a new frame from the video. The first time it is called, it just saves the current frame as the backgroundFrame and returns. Afterwards, it compares new frames to the backgroundFrame and detects the motion. It then updates the background Frame by merging it with the current frame.
The motion detection works by first converting the image to black and white. Then, it applies a pixel filter to group similar pixels. Then a difference filter is applied to highlight the areas where the pixels have changed and then they are filtered by a threshold filter. A blobdetection object is then used on the image to draw rectangles on the effected areas. The blobs are ordered by size and we take the largest one. By calculating where the rectangle appears on screen compared to the viewing angle of the camera, we then call the TurretController to move to the target's location. Checks are preformed to see if the target is new or not and the appropriate actions are taken.

Start_click() : void
Is called when the user clicks the start button on the GUI. It takes the selected cameras and begins streaming the video to the program. It also takes the current battery life from the PCB and displays it on screen

Motion Detector	
stream : videoStream background : bitmap currentFrame : bitmap target : Rectangle hasTarget : Boolean turretpos : double	NewFrame(sender : object, image: Bitmap) : void Start_click() : void

Figure 25. The Motion Detector Class

3.6.4 Target Class

The motion detecting and target acquisition uses a data structure named Target. Target will be a public class that may and will have numerous instances. The purpose of the class is to keep track of not only position and location relative to the turret, but statistics and other information such as the engagement history of the target. What follows is a listing and summery of the public variables and their data types that will be contained in the Target class. The class diagram is displayed below in **Figure 26**.

3.6.4.1 Variables

ID : int
The identification number of the target. This number differentiates between targets and is useful for the engagement history.

angleLocation : int
An integer representation of the targets angular location on the horizontal axis.

pitchLocation : int
An integer representation of the targets angular location on the vertical axis.

distance : int
An integer representation of the targets distance from the turret.

timeAcquired : int
The time that the target was acquired. This is useful to track how long the target has been tracked for the tracking algorithm and also to calculate total tracking time for the engagement history after the target has been released. There is no

helper setTimeAcquired method as the **timeAcquired** is set during the creation of the Target Object by the constructor.

timeReleased : int
The time that the target was released from tracking. Useful for calculating total tracking time.

shotsFired : int
An integer that keeps track of how many shots have been fired at the target. Usefull for determining if the target is still a threat and can be released as well as for engagement history for the web application.

active : boolean
A flag that signifies that the target is the current target the turret is presently tracking. Only one target object at a time may have this flag set to true.

3.6.4.2 Methods

Target() : void
The constructor of the Target class. It sets the ID and time acquired variables and initializes the newly created object.

getAngleLocation() : int
This method returns the angular location of the target

setAngleLocation(int) : void
This method updates the angular location of the target.

getPitchLocation() : int
This method returns the vertical location of the target.

setPitchLocation(int) : void
This method updates the vertical location of the target.

getDistance() : int
Returns the last updated distance of the target. This method will be used by the server to create the engagement history.

setDistance(int) : void
Updates the target's distance from the turret

getTimeAquired() : int
Returns the time that the target was first targeted by the turret. This method will be used by the server to create the engagement history.

getTimeReleased() : int

Returns the time that the target was no longer tracked by the turret. This method will be used by the server to create the engagement history.

getShotsFired() : int
Returns the number of shots that was taken at the target.

shotFired() : void
Increments the number of shots fired by one. This method is called each time the turret fires at the target.

release() : void
Called when the turret no longer is tracking the target. It sets the timeReleased integer.

Target	
ID : int angleLocation : int pitchLocation : int distance : int timeAcquired : int timeReleased : int shotsFired : int active : boolean	Target() : void getAngleLocation() : int setAngleLocation(int) : void getPitchLocation() : int setPitchLocation(int) : void getDistance() : int setDistance(int) : void getTimeAquired() : int getTimeReleased() : int getShotsFired() : int shotFired() : void release() : void

Figure 26. Targeter Class Diagram

3.6.5 Turret Controller Class

The Turret Controller module of our turret system sits between the target acquisition and the microcontroller. It sends the instructions to the servos to move to the desired locations and it gets input from the turret in regard to its position. The automated controls of the turret will be written in the object oriented language C#. It was an intentional decision to use a high level language not only to make code easier to write, read, and maintain, but the modularity of object oriented design allows for a level of robustness and reuse that would be harder to recreate using a lower level language. In fact, the design of the automated controls should allow for a variety of control schemes with little rewriting and many of the methods could be reused for varying functionality. Below is a summary of the methods that will be implemented. The class diagram is displayed below in **Figure 27**.

3.6.5.1 Variables

panpos : byte
The horizontal angle of the turret.

tiltpos : byte
The tilt angle of the turret

battery : byte
The charge on the battery as reported by the microcontroller.

isMoving : Boolean
A Boolean flag that keeps track if the turret is currently in motion.

3.6.5.2 Methods

getAngle() : int
Checks the turret controller and gets a value that represents the current angular position of the turret. It then returns this value as an integer between 0-1800.

setAngle(int) : void
The integer that is passed in is a value between 0-1800 (900 = 90.0 degrees). This value is sent to the turret controller which moves the turret to the specified location.

getPitch() : int
Checks the turret controller and gets a value represents the current angular pitch of the turret. It then returns this value as an integer between -200-200.

setPitch(int) : void
The integer that is passed in is a value between -200-200. This value is sent to the turret controller which moves the turret to the specified pitch.

isMoving() : Boolean
Returns true or false depending if the turret is currently in motion.

Alarm(int) : void

Turns the onboard siren on for a given amount of seconds. If the siren is already sounding, it will stop the current siren and start the new signal.
.
Fire(int) : void
Sends message to turret controller to fire the gun. The integer that is passed in is the requested amount of shots to take.

isFiring() : Boolean

Returns true or false depending if the turret is currently firing.

OnRecived(object, SerialDataReceivedEventArgs)
Automatically called whenever the microcontroller sends data to the onboard computer. It decodes the byte packets and updates the onboard computer as to the status of the turret.

Turret Controller	
panpos : byte tiltpos :byte battery : byte isMoving : Boolean	getPitch() : int setPitch(int) : void isMoving() : boolean alarm(int) : void Fire(int) : void isFiring() : boolean getAngle() : int setAngle(int) : void Onrecived(object, SerialDataReceivedEventArgs) : void

Figure 27. Turret Controller Class Diagram

3.6.6 Microcontroller Communication

The turret controller acts as the interface between the messages sent from the PC and the microcontroller that actually powers the servos on the turret. Since speed is a factor and memory is more limited than on a PC, messages are sent and received by the microcontroller as packets. Each packet consists of 5 bytes. These 5 byte packets of 8 bits each are both output and received as input by the microcontroller.

Outbound packets from the PC to the microcontroller tell the servos to activate and move. They also control the alarm system of the turret. The breakdown of the packet can be seen in **Figure 28**.

Figure 28. Outbound Packet Structure

The PC will also be constantly receiving input from the microcontroller every loop. The packet has a similar structure, but contains information on the current status of the servos. The current position and battery status is sent to the onboard computer. The packet structure breakdown can be seen below in **Figure 29**.

Figure 29. Inbound Packet Structure

These data packets are responsible for telling the Turret Controller in what position the turret is in and that information is fed to Automated and Manual Controls. The flag byte will contain some one bit flags that will let the PC know the status of some systems as well as the turret's current state. There are some bits left over which may find some use during development or future iterations of this project. The flag structure can be seen in **Figure 30**:

Figure 30. The Flag Byte Structure

The first two bits are flags that let the Turret Controller know if the turret is currently moving. This is useful for the firing methods as our current plans disallow firing of the turret while it is in motion. The third byte is for keeping track of when the firing servo is being activated. We do not want to try to fire the mounted gun while it is already being fired. The last byte keeps track of whether the alarm is currently sounding.

3.6.7 Microcontroller Code

The microcontroller is written in the Arduino IDE using Wired and as such, is not programmed in an object oriented manner. It is written in a manner such that the code could be reusable in similar projects though. Arduino programs are separated into three main parts. First, are the variable declarations such as Servos and integers that are going to be used in the program. Second is the setup() method that is run once as soon as the microcontroller is programmed. In our setup() method we initialize the serial communication port, attach the servos to their respective pins as well as the alarm and battery pins, and initialize all the variables. The last and most important part of the Arduino program is the loop() method. As its name suggests, this method is called over and over again for the lifetime of the microcontroller's life. This is where we program all of the microcontroller logic.

In our implementation we first read the current position of all the servos. Then we check if there are any packets ready to be read from the onboard controller. If there are, we read the serial port and save the packets. If the packet is a movement instruction, we update the current desired position of the turret. If the packet is an instruction to fire or sound the alarm, we start to perform the action and then save the current time in milliseconds. We then take a look at the desired pan and tilt of the turret and move one degree in that direction. Also, we look at the current time and if the trigger or the alarm have been active for a specified amount of time (half a second) we stop those actions. Finally, we read the battery voltage from analog pin 1 and preform some calculations to convert the value in to volts. We then multiply the value by 100 to get rid of any decimals so we can send it as a byte. The last thing we do is insert a 300 millisecond delay. This delay is useful because it slows the speed of the turret, making it more accurate and reducing strain on the servos and base. We tried to keep the total delay of each loop as small as possible so that we could send and receive updates to the turret as fast as possible.

3.6.8 Manual Control Server

This class is designed to handle all incoming commands from the **ManualControlApplet** class when connected. It runs using two threads. One thread handles the incoming commands and executes immediate actions to the **turretController** while the other thread updates the **turretController** using persistent actions like continuous rotation and pitch. The class diagram is presented in **Figure 31** and explained below.

3.6.8.1 Command Constants

ROTATE_CMD
This constant is the string expected from the **ManualControlApplet** in order to command the turret to rotate. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

PITCH_CMD
This constant is the string expected from the **ManualControlApplet** in order to command the turret to specify the offset for the vertical angle set its pitch to the vertical angle to a specific degree. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

SIREN_CMD
This constant is the string expected from the **ManualControlApplet** in order to command the turret to execute a normal occurrence of the siren. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

FIRE_CMD
This constant is the string expected from the **ManualControlApplet** in order to command the turret to fire once. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

SET_ROTATE_CMD
This constant is the string expected from the **ManualControlApplet** in order to command the turret to turn on or off a constant rotation in either the left or right direction. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

SET_PITCH_CMD
This constant is the string expected from the **ManualControlApplet** in order to command the turret to turn on or off a constant change in pitch either up or down. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

3.6.8.2 Direction Constants

UP, **DOWN**, **LEFT** and **RIGHT** are the constants used to specify what to expect from the **ManualControlApplet** in the web application in order to specify the direction when using the **SET_ROTATE_CMD** and **SET_PITCH_CMD**. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

3.6.8.3 State Constants

ON and **OFF** are the constants used to specify what to expect from the state in the **SET_ROTATE_CMD** and **SET_PITCH_CMD** commands. The value of the constant matches the value expected in the **ManualControlApplet** class in the web-application.

3.6.8.4 Other Constants

PASSCODE
This constant holds the value expected to be received upon receiving a connection from the **ManualControlApplet** class, any other value received will be considered an unauthorized connection.

name
This is the common name of the server and it originates from the configuration file in the turret software.

port
This is the port the thread will listen on for connections from the **ManualControlApplet**.

turretController
This is the **TurretController** instance that the turret is using to send commands to the micro-controller to control the motors and other devices of the turret. It is also reference in order to find out the current state of the server.

socket
This is the socket connection to a **ManualControlApplet**. Only one socket instance exists at a time meaning a **ManualControlServer** can only have one simultaneous connection at a time.

targetAngle
This holds the current angle the turret is expected to move toward. This value may be different than the actual angle of the turret however when new commands are received from the **ManualControlApplet** the new target angle is calculated using an offset from this value rather than the actual turret angle.

targetPitch
This holds the current pitch the turret is expected to move toward. This value may be different than the actual pitch of the turret however when new commands are received from the **ManualControlApplet** the new target pitch is calculated using an offset from this value rather than the actual turret pitch.

rotatingRight
This variable indicates whether the turret is currently maintaining right rotation. If this value is true, **rotatingLeft** must be false.

rotatingLeft
This variable indicates whether the turret is currently maintaining left rotation. If this value is true **rotatingRight** must be false.

pitchingUp

This variable indicates whether the turret is currently pitching up. If this value is true **pitchingDown** must be false.

pitchingDown
This variable indicates whether the turret is currently pitching down. If this value is true **pitchingUp** must be false.

3.6.8.5 Methods

ManualControlServer()
This method initializes all the variables to their default values.

run()
This method first waits for a connection to be established with the **ManualControlApplet**. After verifying that the connection is authorized from the **validate()** method, the method will spawn a new thread to process data traffic on the socket using the **processCmds()** method. In parallel, this method will loop and check to see if it needs to apply any continuous rotation or pitch changes. Once manual control is stopped, the method will return to waiting on a connection.

calcTargetAngle(offset : Integer)
This method calculates the new target angle based on the current turret angle, current target angle, and the offset. If the difference between the turret's actual angle and target angle is opposite from the direction of the offset, the new target angle is based on the turret's current angle and the offset. Otherwise the new target angle is based of the current target angle and the offset.

calcTargetPitch(offset : Integer)
This method calculates the new target pitch based on the current turret pitch, current target pitch, and the offset. If the difference between the turret's actual pitch and target pitch is opposite from the direction of the pitch, the new target pitch is based on the turret's current pitch and the offset. Otherwise the new target pitch is based of the current target pitch and the offset.

setAngle()
This method calls the **setAngle()** method in the **turretController** using the current target angle when the current target angle is modified.

setPitch()
This method calls the **setPitch()** method in the **turretController** using the current target pitch when the current target angle is modified.

validate(passcode : String)
This method compares the passcode sent from the **ManualControlApplet** against the **PASSCODE** value hardcoded in the class. If the strings match the

turretController is notified to switch to manual control and ignore the automated targeting system.

processCmds()
This method loops waiting on receiving input from the **ManualControlApplet**. It processes all immediate commands once received including **ROTATE_CMD**, **PITCH_CMD**, **SIREN_CMD**, and **FIRE_CMD**. However **SET_ROTATE_CMD** and **SET_PITCH_CMD** control the **rotatingRight**, **rotatingLeft**, **pitchingUp**, and **pitchingDown** commands which are used by the **run()** method for continuous control of the turret.

terminateControl()
This method cancels the lock on the **turretController** and allows it to resume to automatic control. This occurs when the socket connection from the **ManualControlApplet** is closed or lost.

ManualControlServer :: Thread	
+ ROTATE_CMD : String + PITCH_CMD : String + SIREN_CMD : String + SILENCE_CMD : String + FIRE_CMD : String + SET_ROTATE_CMD : String + SET_PITCH_CMD : String + UP : Integer + DOWN : Integer + LEFT : Integer + RIGHT : Integer + ON : Integer + OFF : Integer - PASSCODE : String - name : String - port : Integer - turretController : TurretController - socket : Socket - targetAngle : Integer - targetPitch : Integer - rotatingRight : Boolean - rotatingLeft : Boolean - pitchingUp : Boolean - pitchingDown : Boolean	+ ManualControlServer() + run() : void - calcTargetPitch(offset : Integer) : Integer - calcTargetAngle(offset : Integer) : Integer - setAngle(angle : Integer) : void - setPitch(pitch : Integer) : void - soundSiren() : void - silenceSiren() : void - fire() : void - validate(String passcode) : Boolean - processCmds() : void - terminateControl() : void

Figure 31. C# class diagram for the ManualControlServer class.

3.7 Server Side Software Design

The off-board server software is responsible for collecting and storing the history and current state of all connected turrets, providing manual control for the connected turrets and displaying the history and current state of the turrets to users via a web-application. To accomplish these tasks, the system is broken down into 3 primary components. The first component is the Turret Data Collector which is a background java process that provides an active socket connection for turrets to connect to in order to send their current and historical information to be persisted. The second major component is a Django Web-server which provides the web-application to display the current and historical data to users. The third major component is a java applet which is accessible in the web-application that connects to the specified turret in order to override the automated targeting and take manual control of the turret. As part of the manual control a flash video streaming client is used in order to view what the turret is looking at while controlling it manually. This interplay of components is shown in **Figure 32**.

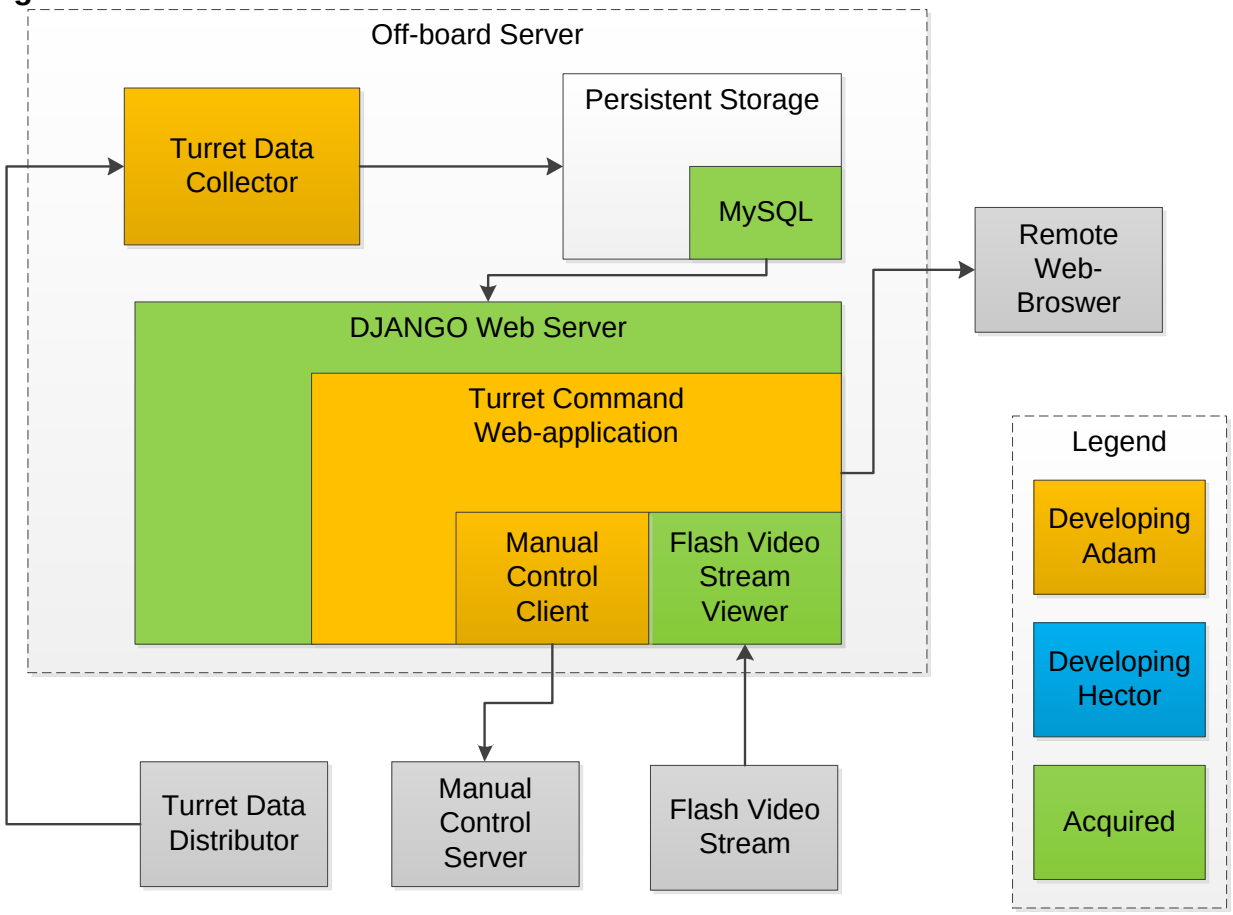

Figure 32. Block Diagram of the general off-board server software design.

3.7.1 Turret Command Web-Application

The Turret Command Center is a web-application that provides historical records, live-radar, and manual control of the connected servers. The web-application is programmed in Python using the Django development library. Each method in the class is designed to be processed when a url pattern is recognized. The server processes the page template and then returns the page html to be displayed.

3.7.1.1 Methods

index()
This method loads the template for the home page and the latest set of engagements and compiles the html and sends it to the browser.

engagements()
This method loads the template for the engagement list and all the recorded engagements ordered by date/time descending and compiles the html and sends it to the browser.

filteredEngagements(turretId : String)
This method loads the template for the engagement list and only passes the engagements for the specified turret ordered by date/time descending and compiles the html. The compiled html is then sent to the browser.

retrieveEngagementDetails(engagementId : String)
This method loads the template for a single engagement information pane and all the relevant information to the specific engagement with the **engagement_ID** matching **id**. The template is then compiled with the added information and the resulting html is sent to the browser.

turrets()
This method loads the template for the page that views all the turrets and retrieves all the recorded turrets from the database. The template is then compiled and the resulting html is sent to the browser.

turret(turretId : String)
This method loads the template for viewing the detailed page of a single turret specified by the **Turret_ID** matching the **id** and all relevant information for a specific turret. The template is then compiled and the resulting html is sent to the browser.

turretRadar(turretId : String)
This method loads the template for a **Radar** pane and all the relevant information about the turret and targets. The template is then compiled and the resulting html is sent to the browser.

getRadar(turretId : String)
This method retrieves the latest state of a turret and returns a JSON serialized object of all the relevant data to the browser to be processed to update a **Radar** object.

watchRadar(turretStr : String)
This method loads the template for viewing multiple radars of all specific turrets with **Turret_ID**s listed in by the id list. The template is then compiled and the resulting html is sent to the browser.

3.7.2 Turret Class

This class represents the relational object to the **Turret** database table. The class diagram for the Java object is presented in **Figure 33** and explained below. The object is also reflected in Python however only uses the properties, but due to python's use, no getters or setters are required.

3.7.2.1 Variables

turretId : String
This is the unique identifier for a specific turret instance.

name : String
This is the common name to identify a specific turret instance

latitude : Double
This is the relevant latitude position of the turret to the other turrets in a common space. (Optional)

longitude : Double
This is the relevant longitude position of the turret to the other turrets in a common space. (Optional)

orientation : Integer
This is the orientation to north of the center of the turret. (Optional)

angle : Integer
This is the current angle relevant to the center of the turret that the turret is currently facing.

pitch : Integer
This is the angle between the pitch of the turret and the level plane of the turret that it is currently at.

warningRange : Integer

This is the distance the turret is specified to warn the target.

firingRange : Integer
This is the distance the turret is specified to fire at the target.

turretIP : String
This is the IP address of the turret. This is used for passing the IP address to the manual control applet when controlling the specific turret.

cameraIP : String
This is the IP address of the IP camera associated with the turret. This is used for passing the IP address to the live video stream player during manual control.

Cameras : List<FieldOfVision>
This is the list of cameras associated with the turret.

Turret()
Constructor for the Turret class.

getTurretId() : String / setTurretId(String)
These provide the ability to get and set the **turretId** property.

getName() : String / setName(String)
These provide the ability to get and set the **name** property.

getLatitude() : Float / setLatitude(Float)
These provide the ability to get and set the **latitude** property.

getLongitude() : Float / setLongitude(Float)
These provide the ability to get and set the **longitude** property.

getOrientation() : Integer / setOrientation(Integer)
These provide the ability to get and set the **orientation** property.

getAngle() : Integer / setAngle(Integer)
These provide the ability to get and set the **angle** property.

getPitch() : Integer / setPitch(Integer)
These provide the ability to get and set the **pitch** property.

getWarningRange() : Integer / setWarningRange(Integer)
These provide the ability to get and set the **warningRange** property.

getFiringRange() : Integer / setFiringRange(Integer)
These provide the ability to get and set the **firingRange** property.

getTurretIP() : String / setTurretIP(String)
These provide the ability to get and set the **turretIP** property.

getCameraIP() : String / setCameraIP(String)
These provide the ability to get and set the **cameraIP** property.

getCameras() : List<FieldOfVision> / setCameras(List<FieldOfVision>)
These provide the ability to get and set the **cameras** property.

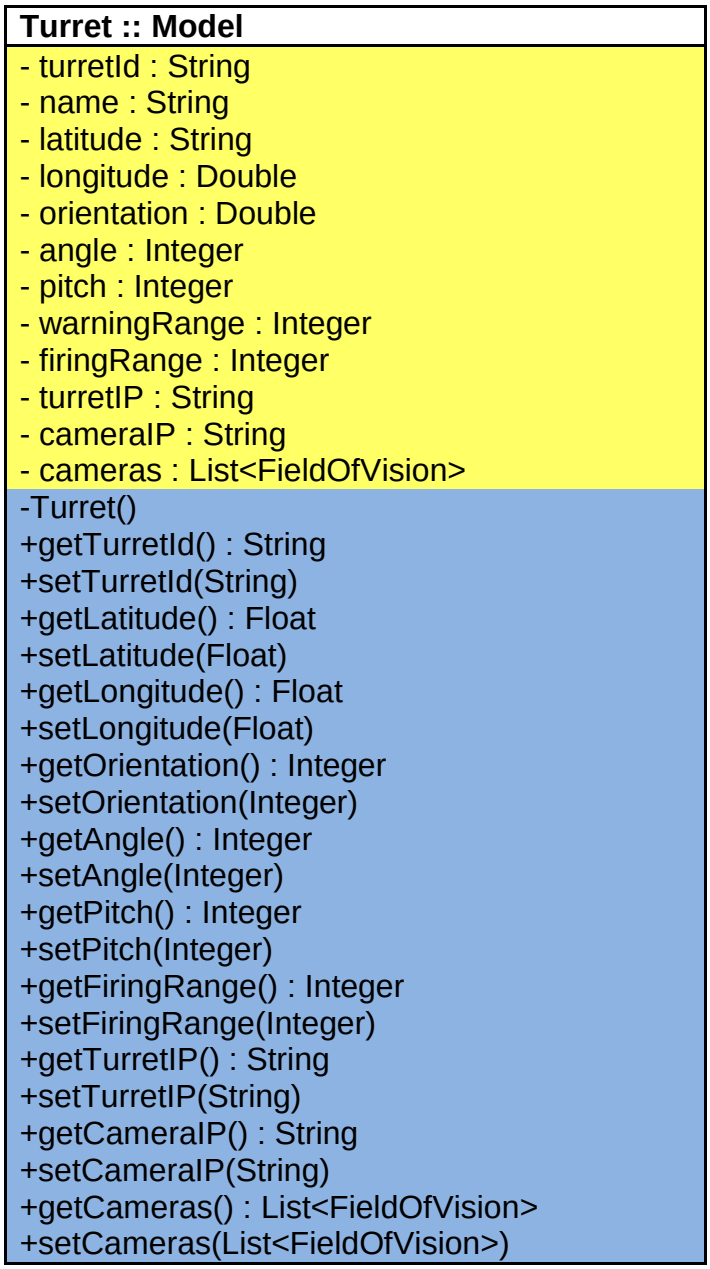

Figure 33. Python class diagram for the Turret class.

3.7.3 TurretTarget Class

This class represents the relational object to the **Turret_Targets** database table. The class diagram is presented in **Figure 34** and explained below. This object was only used in the python code.

3.7.3.1 Variables

targetId : String
This is the unique identifier for a target.

turretId : String
This is the link to the turret that identified the target.

distance : Ineger
This is the distance from the turret to the target. When this distance is not null, it means that a moving object was identified but has not been measured or is outside of the warning range.

angle : Integer
This is the angle the target was identified at relevant to the center view of the turret.

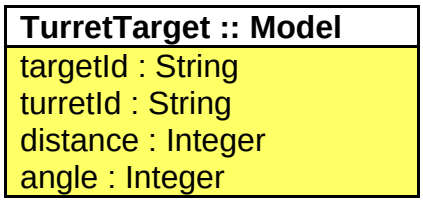

Figure 34. Python class diagram for the TurretTarget class.

3.7.4 FieldOfVision Class

This class represents the relational object to the **Field_Of_Vision** database table. The class diagram is presented in **Figure 35** and explained below.

3.7.4.1 Variables

cameraId : String
This is the unique identifier of the a camera.

turretId : String
This is the link to the turret that the camera is connected.

fovOrientation : Integer

This is the angle in regards to the turret that the center of the camera's field of vision.

horizontalFOV : Integer
This is the angle width of the field of vision of the camera in the horizontal plane.

verticalFOV : Integer
This is the angle height of the field of vision of the camera in the vertical plane.

FieldOfVision()
Constructor for **FieldOfVision** object.

getCameraId() : String / setCameraId(String)
These provide the ability to get and set the **cameraId** property.

getTurretId() : String / setTurretId(String)
These provide the ability to get and set the **turretId** property.

getFovOrientation() : Integer / setFovOrientation(Integer)
These provide the ability to get and set the **fovOrientation** property.

getHorizontalFOV() : Integer / setHorizontalFOV(Integer)
These provide the ability to get and set the **horizontalFOV** property.

getVerticalFOV() : Integer / setVerticalFOV(Integer)
These provide the ability to get and set the **verticalFOV** property.

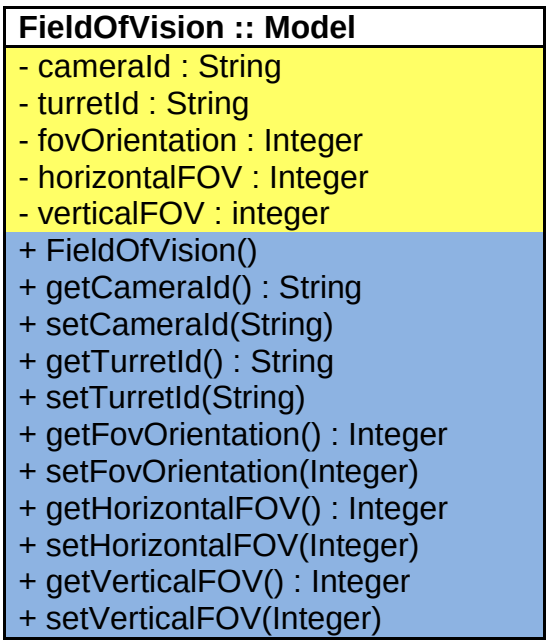

Figure 35. Python class diagram for the FieldOfVision class.

3.7.5 Engagement Class

This class represents the relational object to the **Engagment_History** database table. The class diagram is presented in **Figure 36** and explained below.

3.7.5.1 Variables

engagementId : String
This is the unique identifier of the engagement.

turretId : String
This is the link to the turret that experienced the engagement.

engagementDttm : Long
This is the date and time that the engagement occurred.

distance : Integer
This is the distance between the turret and the target at the time of the engagement.

angle : Integer
This is the angle from the center of the turret to the direction of the target at the time of the engagement.

latitude : Double
This is the latitude position that the target was at when engaged.

longitude : Double
This is the longitude position that the target was at when engaged.

screenshot : String
This is the filename of the screenshot from when the target was engaged from the gun-mounted camera.

Engagement()
This is the constructor for the Engagement class.

getEngagementId() : String / setEngagementId(String)
These provide the ability to get and set the **engagementId** property.

getEngagementDttm() : Long / setEngagementDttm(Long)
These provide the ability to get and set the **engagementDttm** property.

getDistance() : Integer / setDistance(Integer)
These provide the ability to get and set the **distance** property.

getAngle() : Integer / setAngle(Integer)
These provide the ability to get and set the **angle** property.

getLatitude() : Float / setLatitude(Float)
These provide the ability to get and set the **latitude** property.

getLongitude() : Float / setLongitude(Float)
These provide the ability to get and set the **longitude** property.

getScreenshot() : String / setScreenshot(String)
These provide the ability to get and set the **screenshot** property.

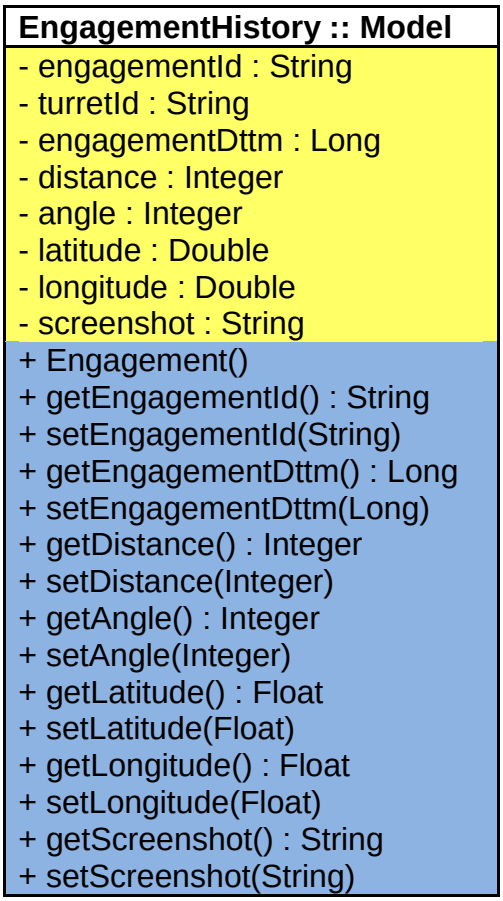

Figure 36. Python class diagram for the Engagement class.

3.7.6 DataCollectorServer Class

The **DataCollectorServer** class is the thread that handles accepting all incoming connections from turrets and spawns DataCollectorWorker threads to continue

processing the data. The class diagram is presented in **Figure 37** and explained below.

TCP_PORT : Integer
This specifies the port to run the ServerSocket for accepting connections from the turrets.

processStatus : StatusCollectorWorker
This is the StatusCollectorWorker thread instance that is created to process incoming UDP status packets.

listen : Boolean
This is specifies whether the DataCollectorServer should be listening for new connections or not.

main(String[])
This method is the main method to indefinitely listen on a TCP socket at port **TCP_PORT** for new connections from turrets and then sends them into a **DataCollectorWorker** instance to be processed.

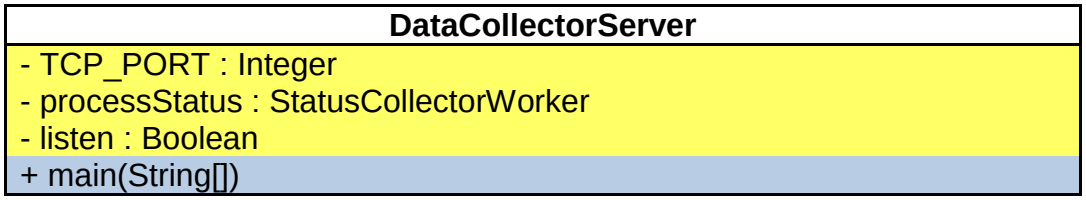

Figure 37. Java class diagram for the DataCollectorServer class.

3.7.7 DataCollectorWorker Class

The **DataCollectorWorker** class is the thread that processes all the data coming from a single turret to add the data into the database for use in the web-application. The class diagram is presented in **Figure 38** and explained below.

turret : Turret
This is the static data for the turret that is received at the start of the connection.

dbUtil : DatabaseUtil
This is the instance of a DatabaseUtil that handles all inserts/updates into the database.

INITIAL_MEDIA_DIR : String
This is the directory that all initial camera set-ups images are stored in.

ENGAGEMENT_MEDIA_DIR : String
This is the directory that all engagement images are stored in.

DataCollectorWorker(Socket, DatabaseUtil)
This is the constructor for the DataCollectorWorker class.

run()
This is the main method that is kicked off from the DataCollectorServer which handles all the TCP data transmissions coming from the server.

receiveImage(String, String)
This handles grabbing a current screenshot from the IP camera associated with the turret when necessary.

DataCollectorWorker :: Thread
- turret : Turret - dbUtil : DatabaseUtil - INTIAL_MEDIA_DIR : String - ENGAGEMENT_MEDIA_DIR : String
+ DataCollectorWorker(Socket, DatabaseUtil) + run() : void - receiveImage(String, String)

Figure 38. Java class diagram for the TurretDataCollector class.

3.7.8 StatusCollectorWorker Class

This class handles the UDP status messages sent from all turrets and updates the **TurretTargets** table and the **Turret** table for angle and pitch. The class diagram is presented in **Figure 39** and explained below.

port : Integer
This specifies the port that the **StatusCollectorWorker** listens.

dbUtil : DatabaseUtil
This handles all the leg work for updating the **TurretTargets** table and the **Turret** table.

running : Boolean
This controls when the **StatusCollectorWorker** listens for UDP packets.

turretLookup : Map<String, String>
This holds the mapping of the **turretIP** to the **turretID** for when performing status updates.

StatusCollectorWorker(DatabaseUtil)
This is the constructor for the **StatusCollectorWorker** class.

run()

This is the main method that runs that processes indefinitely UDP packets and stores the updates for the angle, distance, and turret targets.

registerTurret(String, String)
This enters a turretId, IP address pair into the **turretLookup** map.

stopRunning()
This allows a different process to terminate processing UDP messages.

StatusCollectorWorker :: Thread
- port : Integer - dbUtil : DatabaseUtil - running : Boolean - turretLookup : Map<String, String>
+ StatusCollectorWorker(DatabaseUtil) + run() : void + registerTurret(String, String) : void + stopRunning() : void

Figure 39. Java class diagram for the TurretDataCollector class.

3.7.9 DatabaseUtil Class

This class provides utility class for handling JDBC connections and executing different kinds of MySQL queries. This class is used by the **DataCollectorWorker** and the **StatusCollectorWorker** for transactions with the off-board MySQL database. The class diagram is presented in **Figure 40** and explained below.

3.7.8.1 Variables

DB_URL : String
This is the MySQL database connection string used for getting a connection.

USER : String
This is the user name for the database connection.

PASSWORD : String
This is the password for the database connection.

SCHEMA : String
This is the schema for the tables in the MySQL database.

STORE_TURRET : String
This is the SQL string for inserting or updating an entry into the **Turret** table.

UPDATE_TURRET : String
This is the SQL string for updating the angle and distance of the **Turret** table.

STORE_ENGAGEMENT : String
This is the SQL string for inserting or updating an entry into the **Engagement_History** table.

STORE_FOV : String
This is the SQL string for inserting or updating an entry into the **Field_Of_Vision** table.

STORE_TARGET : String
This is the SQL string for inserting or updating an entry into the **Turret_Targets** table.

DELETE_TARGETS : String
This is the SQL string for deleting all targets for a specific turretId from the **Turret_Targets** table.

conn : Connection
This is the Java JDBC Connection for performing MySQL statements.

storeTurretStmt : PreparedStatement
This is the **PreparedStatement** for executing the **STORE_TURRET** SQL string.

updateTurretStmt : PreparedStatement
This is the **PreparedStatement** for executing the **UPDATE_TURRET** SQL string.

storeEngagementStmt : PreparedStatement
This is the **PreparedStatement** for executing the **STORE_ENGAGEMENT** SQL string.

storeTargetStmt : PreparedStatement
This is the **PreparedStatement** for executing the **STORE_TARGET** SQL string.

storeFieldOfVisionStmt : PreparedStatement
This is the **PreparedStatement** for executing the **STORE_FOV** SQL string.

deleteTurretTargetsStmt : PreparedStatement
This is the **PreparedStatement** for executing the **DELETE_TARGETS** SQL string.

DatabaseUtil()
This is the Constructor for the DatabaseUtil class.

getConnection() : Connection

This gets a new JDBC Connection using the **DB_URL**, **USER** and **PASSWORD**.

commitChanges()
This explicitly calls a commit command to the database.

storeTurretObject(Turret) : Boolean
This stores a Turret object and its associated FieldOfVision objects.

storeTurret(Turret) : boolean
This stores a **Turret** object into the **Turret** Table.

updateTurret(String, Integer, Integer) : Boolean
This updates the **Turret** table for the angle and distance columns.

storeEngagement(Engagement, Turret) : Boolean
This stores the **Engagement** object into the **Engagement_History** table.

storeFieldOfVision(FieldOfVision, Turret) : Boolean
This stores the **FieldOfVision** object into the **Field_Of_Vision** table.

storeTurretTarget(String, String, Integer, Integer) : boolean
This stores the distance and angle for a new target into the **Turret_Targets** table.

clearTurretTargets(String) : boolean
This deletes all the targets for a particular turretId from the **Turret_Targets** table.

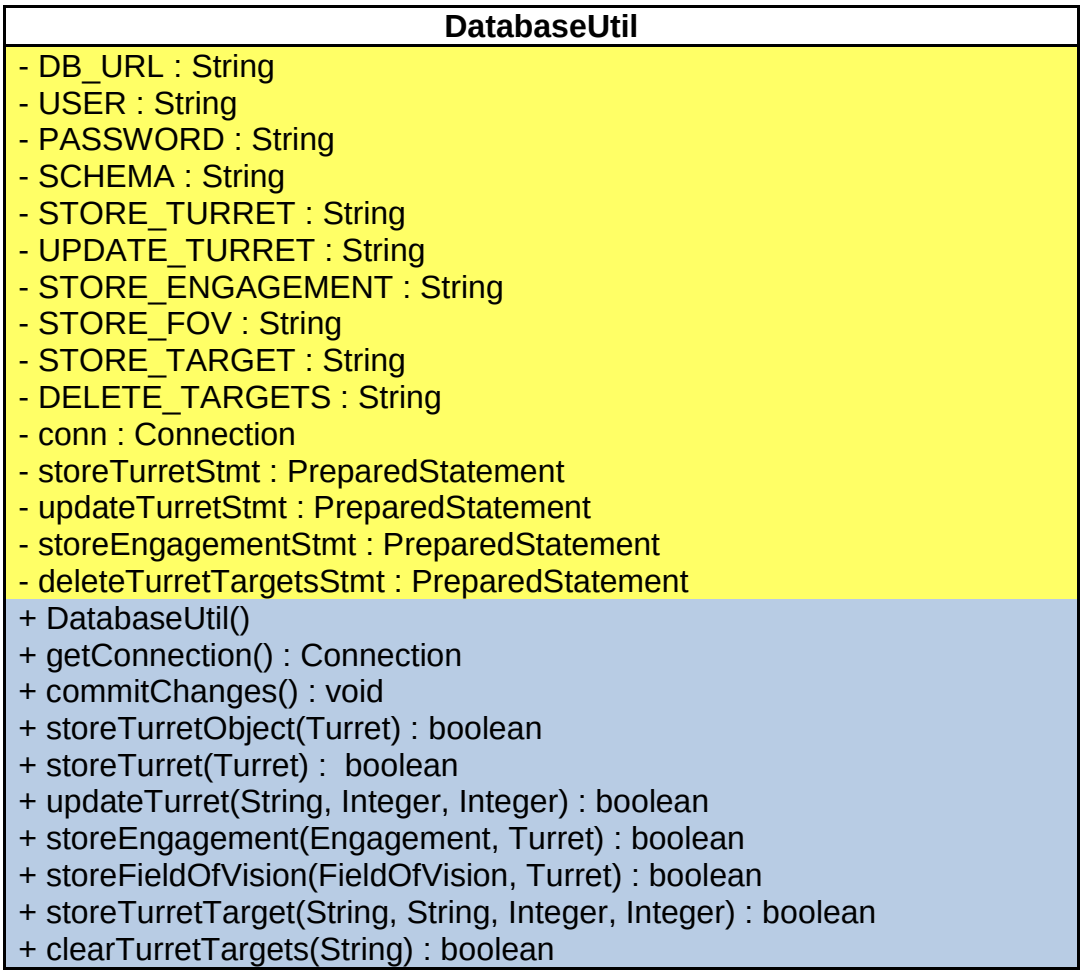

Figure 40. Java class diagram for the DBUtil class.

3.7.10 ManualControlApplet Class

One of the most important features of our web application is the manual control interface for the turret. Users will be able to manipulate the turrets angle and pitch as well as fire the mounted gun. The **ManualControlApplet** class is a java class that extends **java.applet.Applet**. The applet will be provided in the off-board web-application to override the automatic targeting of the turrets connected to the server and grant manual control to the turret. As part of the manual control the user is able to rotate and pitch the turret, fire the firearm, and sound or silence the turret warning siren. To control these features the user is able to control the turret using a combination of the keyboard and mouse. The keyboard alone can perform all the available functions of the turret while the mouse is limited to only rotating, pitching and firing the turret. The class diagram is presented in **Figure 41** and explained below.

3.7.10.1 Command Constants

ROTATE_CMD

This constant is the string sent to command the turret to offset its horizontal angle a specific degree amount. The value of the constant matches the value expected in the **ManualControlServer** class on the turret.

PITCH_CMD
This command is used to command the turret to specify the offset for the vertical angle set its pitch to the vertical angle to a specific degree. The value of the constant matches the value expected in the **ManualControlServer** class on the turret.

SIREN_CMD
This command is used to command the turret to execute a normal occurrence of the siren. The value of the constant matches the value expected in the **ManualControlServer** class on the turret.

FIRE_CMD
This command is used to command the turret to fire once. The value of the constant matches the value expected in the **ManualControlServer** class on the turret.

SET_ROTATE_CMD
This constant is the string sent to command the turret to turn on or off a constant rotation in either the left or right direction. The value of the constant matches the value expected in the **ManualControlServer** class on the turret.

SET_PITCH_CMD
This constant is the string sent to command the turret to turn on or off a constant change in pitch either up or down. The value of the constant matches the value expected in the **ManualControlServer** class on the turret.

3.7.10.2 Direction Constants

UP, **DOWN**, **LEFT**, and **RIGHT** are constants used to specify the direction when using the **SET_ROTATE_CMD** and **SET_PITCH_CMD**. The values of these constants match the values expected in the **ManualControlServer** class on the turret.

3.7.10.3 State Constants

ON and **OFF** are constants used to specify the state in the **SET_ROTATE_CMD** and **SET_PITCH_CMD** commands. The values of these constants match the values expected in the **ManualControlServer** class on the turret.

3.7.10.4 Key Constants

The following constants are used to hold the values of the keycodes of the individual buttons on the keyboard when processing keyboard events:
LEFT_ARROW
RIGHT_ARROW
UP_ARROW
DOWN_ARROW
S_KEY
Q_KEY
SPACE_KEY
ESCAPE_KEY

3.7.10.5 Other Constants

PASSCODE
This constant holds a hardcoded string that represents the phrase to validate that the system is authorized to talk to the **ManualControlServer** class on the turret.

ROTATE_SPEED
This specifies the rate at which mouse movement in the x direction correlates to horizontal angle displacement.

PITCH_SPEED
This specifies the rate at which mouse movement in the y direction correlates to vertical angle displacement.

3.7.10.6 Variables

hostname
This variable holds the host name and connection string to attempt a connection to the **ManualControlServer** class on the turret.

port
This specifies the port that the **ManualControlServer** class on the turret is listening.

oldXPosition
This holds the previous x position of the mouse.

oldYPosition
This holds teh previous y position of the mouse.

socket

This holds the **Socket** java class which corresponds to the TCP connection to the **ManualControlServer** class on the turret.

inputStream
This is the **InputStream** java class which corresponds to the data traffic from the **ManualControlApplet** to the **ManualControlServer** class on the turret.

commandStream
This is the **OutputStream** java class which corresponds to the data traffic from the turret control server to the **ManualControlApplet** which is used to transport the current state of the turret. This includes the angle, pitch, and status flags.

3.7.10.7 Methods

ManualControlApplet()
This is the constructor for the applet which will set up the visual presentation of the applet including all labels, buttons, and listeners.

start()
This method is called when the manual control is selected to begin by the user. Before this any input will be disregarded and the turret will remain on automatic targeting. Once this method is executed the applet will assume control over the turret. This method is called when the Start button is pressed in the applet.

stop()
This method is called when the manual control is selected to stop by the user. Once this method is executed manual control of the turret will cease and the automatic targeting will resume. This method is called when the Escape key is processed by the **keyPressed()** method.

keyDown(e : Event, key : Integer)
This method is called whenever the user presses a key down on the keyboard. In this method, the applet will determine which key is pressed by comparing the key parameter against the relevant key code constants listed above. The keys this method processes are the left, right, up, and down arrow keys.

keyUp(e : Event, key : Integer)
This method is called whenever the user releases a key on the keyboard. In this method, the applet will determine which key was released by comparing the key parameter against the relevant key code constants listed above. The keys this method processes are the left, right, up, and down arrow keys.

keyPressed(e : Event, key : Integer)
This method is called whenever the user presses and releases a key on the keyboard. In this method, the applet will determine which key was pressed and released by comparing the key parameter against the relevant key code

constants listed above. The keys this method processes are the 'S', 'Q', Escape, and Spacebar keys.

mouseClicked(e : MouseEvent)
This method is called whenever the mouse is clicked when the event listener is enabled. This method will be used to allow the turret to be controlled by the mouse where clicking the mouse button will fire the turret.

mouseMoved(e : MouseEvent)
This method is called whenever the mouse is moved when the event listener is enabled. This method will be used to rotate and pitch the movement. Using the mouse coordinates of the **MouseEvent** parameter and the **oldXPos** and **oldYPos**, the rotation and pitch changes will be calculated and sent to the turret.

moveLeft()
This method will be called when the turret needs to start rotating left until stopped which is when the left arrow is pressed down.

moveRight()
This method will be called when the turret needs to start rotating right until stopped which is when the right arrow is pressed down.

moveUp()
This method will be called when the turret needs to start pitching up until stopped which is when the up arrow is pressed down.

moveDown()
This method will be called when the turret needs to start pitching down until stopped which is when the down arrow is pressed down.

shiftLeft(offset : Integer)
This method will tell the turret to rotate from its current horizontal angle to shift left in tenths of degrees. The number of tenths of degrees is specified by the offset parameter. This is called after left mouse movement is detected.

shiftRight(offset : Integer)
This method will tell the turret to rotate from its current horizontal angle to shift right in tenths of degrees. The number of tenths of degrees is specified by the offset parameter. This is called after right mouse movement is detected.

shiftUp(offset : Integer)
This method will tell the turret to pitch from its current vertical angle to shift up in tenths of degrees. The number of tenths of degrees is specified by the offset parameter. This method is called after up mouse movement is detected.

shiftDown(offset : Integer)

This method will tell the turret to pitch from its current vertical angle to shift down in tenths of degrees. The number of tenths of degrees is specified by the offset parameter. This is called after down mouse movement is detected.

stopLeft()
This method will tell the turret to stop rotating left if it was previously set to rotate left from the **moveLeft()** method.

stopRight()
This method will tell the turret to stop rotating right if it was previously set to rotate right from the **moveRight()** method.

stopUp()
This method will tell the turret to stop pitching up if it was previously set to pitch up from the **moveUp()** method.

stopDown()
This method will tell the turret to stop pitching down if it was previously set to pitch down from the **moveDown()** method.

fire()
This method will tell the turret to fire the firearm once. This method is called when either the spacebar is processed by the **keyPressed()** method or the left mouse click is processed by the **mouseClicked()** method.

siren()
This method will tell the turret to sound the turret warning siren for a set period of time. This method is called when the 'S' key is processed by the **keyPressed()** method.

silence()
This method will tell the turret to immediately silence the turret warning siren if its currently sounding. This method is called when the 'Q' key is processed by the **keyPressed()** method.

setPort(portNum : Integer)
This configures the applet to attempt to connect to the turret using the port number **portNum**.

setHostname(hostname : String)
This configures the applet to attempt to connect to the turret using the address **hostname** parameter.

sendCommand(String cmd)
This method provides thread-safe access to the **commandStream** to avoid concurrent modification from different threads.

ManualControlApplet :: Applet	
'+ ROTATE_CMD : String '+ PITCH_CMD : String '+ SIREN_CMD : String '+ SILENCE_CMD : String '+ FIRE_CMD : String '+ SET_ROTATE_CMD : String '+ SET_PITCH_CMD : String '+ UP : Integer '+ DOWN : Integer '+ LEFT : Integer '+ RIGHT : Integer '+ ON : Integer '+ OFF : Integer '+ SPACE_KEY : Integer '+ S_KEY : Integer '+ ESCAPE_KEY : Integer '+ Q_KEY : Integer '+ LEFT_ARROW : Integer '+ RIGHT_ARROW : Integer '+ UP_ARROW : Integer '+ DOWN_ARROW : Integer '- PASSCODE : String - ROTATE_SPEED : Integer - PITCH_SPEED : Integer '- hostname : String ' - port : Integer - oldXPos : Integer - oldYPos : Integer - socket : Socket - commandStream : InputStream - outputStream : OutputStream	'+ ManualControl() '+ start() : void '+ stop() : void '+ keyDown(e : Event, key : Integer) : void '+ keyUp(e : Event, key : Integer) : void '+ keyPressed(e : Event, key : Integer) : void '+ mouseClicked(e : MouseEvent) : void '+ mouseMoved(e : MouseEvent) : void '- moveLeft() : void - moveRight() : void '- moveUp() : void '- moveDown() : void - shiftLeft(offset : Integer) : void - shiftRight(offset : Integer) : void - shiftUp(offset : Integer) : void - shiftDown(offset : Integer) : void - stopLeft() : void - stopRight() : void - stopUp() : void - stopDown() : void - setPort(portNum : Integer) : void - setHostname(hostname : String) : void

Figure 41. Java class diagram for the ManualControlApplet class.

3.7.11 Radar Prototype

This Javascript class handles all internal workings of a **Radar** element. The radar is accomplished by an html **Canvas** object that has all relevant elements of the **Radar** drawn on it to represent the real-time status of a turret. The class diagram is presented in **Figure 42** and explained below.

3.7.11.1 Variables

canvas : Cavnas
This the Javascript **Canvas** object used to draw the **Radar** object.

turretName : string
This is the common name of the turret assigned to this **Radar**.

turretX : number
This is the relative X position of the turret assigned to this **Radar**.

turretY : number
This is the relative Y position of the turret assigned to this **Radar**.

turretOrientation : number
This is the relative orientation of the turret assigned to this **Radar**.

turretAngle : number
This is the angle from the relative orientation that the turret assigned to this radar is facing.

fovs : Array
This is an array that holds all the **Field_Of_Vision** objects registered to the turret assigned to this **Radar**.

targets : Array
This is an array that holds all the targets within either the warning range or the firing range of the turret assigned to this **Radar**.

objects : Array
This is an array that holds all the detected objects outside the warning range of the turret assigned to this **Radar**.

warningRange : number
This is the configured warning range of the turret assigned to this **Radar**.

firingRange : number
This is the configured firing range of the turret assigned to this **Radar**.

3.7.11.2 Methods

paint()
This is the method to redraw all elements associated with the **Radar** instance. This includes redrawing the turret, field of visions, targets, and objects.

addTurret(x, y, name, orientation, angle, wRange, fRange)

This method registers the turret to the radar to be drawn on the canvas.

addFOV(center, horizontalFOV)
This method registers a **Field_of_Vision** object to the radar to be drawn on the canvas.

setSize(x, y)
This method specifies the size to draw the canvas.

drawTurret()
This method draws the Turret on the canvas.

drawFOV(x, y, center, horizontalFOV)
This method draws all relevant elements for a Field of Vision on the canvas.

drawTarget(x, y)
This method draws all relevant elements for a target on the canvas.

drawObject(x, y, angle)
This method draws all relevant elements for a detected object on the canvas.

processUpdates(updates)
This method processed the JSON serialized data array from the server and updates all values relevant to the radar.

Radar
Canvas : Cavnas FOVs : Array targets : Array objects : Array warningRange : number firingRange : number
paint() : void addTurret(x, y, name, orientation, angle, wRange, fRange) : void addFOV(center, horizontalFOV) : void setSize(x, y) : void drawTurret(x, y, angle) : void drawFOV(x, y, center, horizontalFOV, wRange, fRange) : void drawTarget(x, y) : void drawObject(x, y, angle) : void

Figure 42. Javascript class diagram for the Radar prototype object.

3.8 Database Design

The database design consists of four tables and the contents and relationships are shown in **Figure 43**. The primary table is the **Turrets** table. The **Turrets** table contains a record for each turret connected to the server. Each row in the **Turrets** table has a unique **Turret_ID** which is the primary key and refers to a randomly generated string assigned to each connected turret. Aside from the unique identification string assigned to the turret, the database has a **Name** column, which stores the common name for the turret for display in the web application. The **Latitude**, **Longitude**, and **Orientation** columns are used for the live radar display when relative positioning between different turrets is desired. Otherwise, these values may be null. The **Latitude** and **Longitude** columns are the cardinal offsets from some common origin of all the turrets, and the **Orientation** is the compass direction the center of turret is facing (the center is the middle of the total horizontal motion). If they are set then the radar will be able to provide all local positions of the turrets in a common shared space. If they are not the web application will only be able to provide relative locations in reference to the orientation and position of a specific turret. The **Turrets** table also contains a **Angle** column and **Pitch** column, which are the current angle and pitch of the turret is situated respectively. The **Warning_Range** and **Firing_Range** are also stored in the **Turrets** table. The **Warning_Range** specifies the distance in feet from the turret of the identified targets that are warned using the audio alarm. The **Firing_Range** specifies the distance in feet from the turret to the identified targets that are fired at using the gun.

The **Field_of_Vision** table is used to store the camera configurations of the turrets. The entries in the **Field_of_Vision** are unique by the **Camera_ID** which is a randomly generated string. The turret that the camera is connected to is specified by the **Turret_ID** that matches the turret's **Turret_ID** in the **Turrets** table. These configurations are used to mark on the radar the area at which is covered by the turret. The **FOV_Orientation** column stores the angle between the center of the turret and the center direction that the camera is facing. The **Horizontal_FOV** column holds the angle size of the camera's horizontal field of vision. The **Vertical_FOV** column holds the angle size of the camera's vertical field of vision.

The **Engagement_History** table contains all accounts of identified targets that entered either the Warning_Range or the Firing_Range. Each engagement is unique by the **Engagement_ID** column which is a randomly generated string. The **Turret_ID** holds the matching **Turret_ID** of the turret that identified the target, and the **Latitude** and **Longitude** columns hold the specific relative cardinal position of the turret in respect to the turret. The **Screenshot** column holds the filename of the screenshot of the target from the gun-mounted camera at the time of identification.

The **Turret_Targets** table keeps track of the current targets of each turret. This is used to relay the active targets in the active range of a turret for use in the live-radar on the web-application. Each entry in the **Turret_Targets** table is given a random string value for unique identification of the **Target_ID** column. The **Turret_ID** of the turret that is tracking the target is stored in the **Turret_ID** column. The distance measured between the turret and the target is stored in the **Distance** column. The **Distance** column may be null if the distance to the target cannot be measured or has not yet been measured. The angle from the center of the turret to the target being tracked is stored in the **Angle** column.

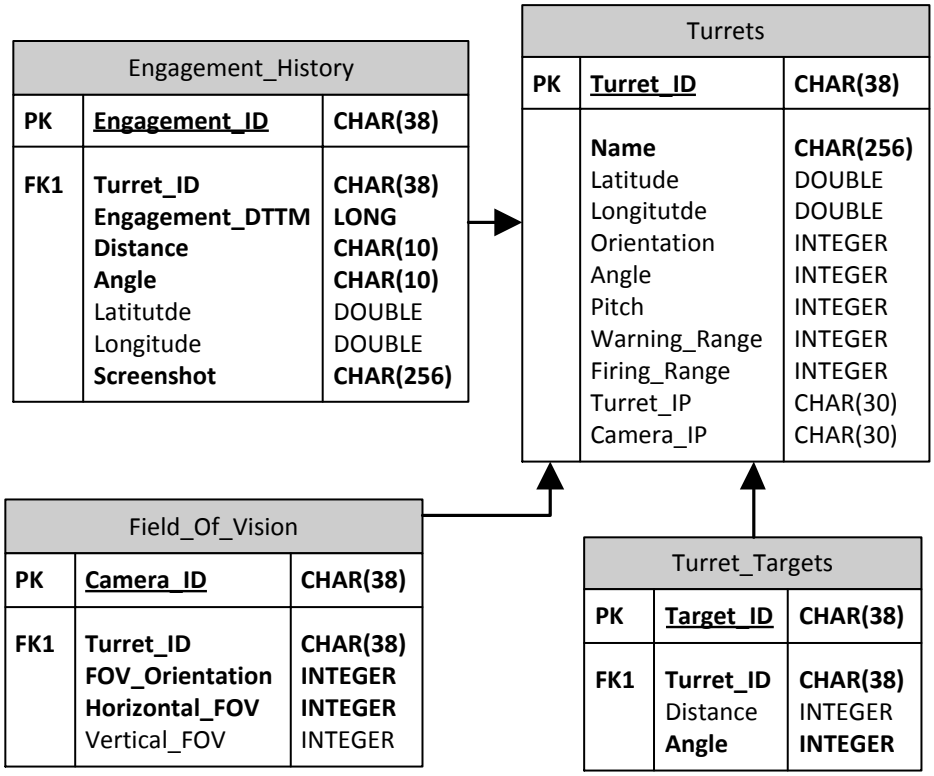

Figure 43. Database Model Diagram of the MySQL database.

3.9 Software Development Planning and Time Management

Since the software aspect of the automated turret is such a complex system with various modules that rely and depend on each other, it is necessary to plan the order and pace of development. Modules that have lots of other modules dependent on it should be developed first in order to avoid development bottlenecks. Modules with few dependencies have much more slack time and have a lower development priority. In order to map the milestones we wish to accomplish and find the critical path for our development we have developed a Program Evaluation and Review Technique chart, or a PERT chart. The PERT chart helps us identify the critical path which is the path with the least slack time.

The critical path determines the minimum amount of time we will spend on the project, and if any component on the critical path gets delayed, the whole project will also be delayed. Below in **Figure 44** is our PERT chart:

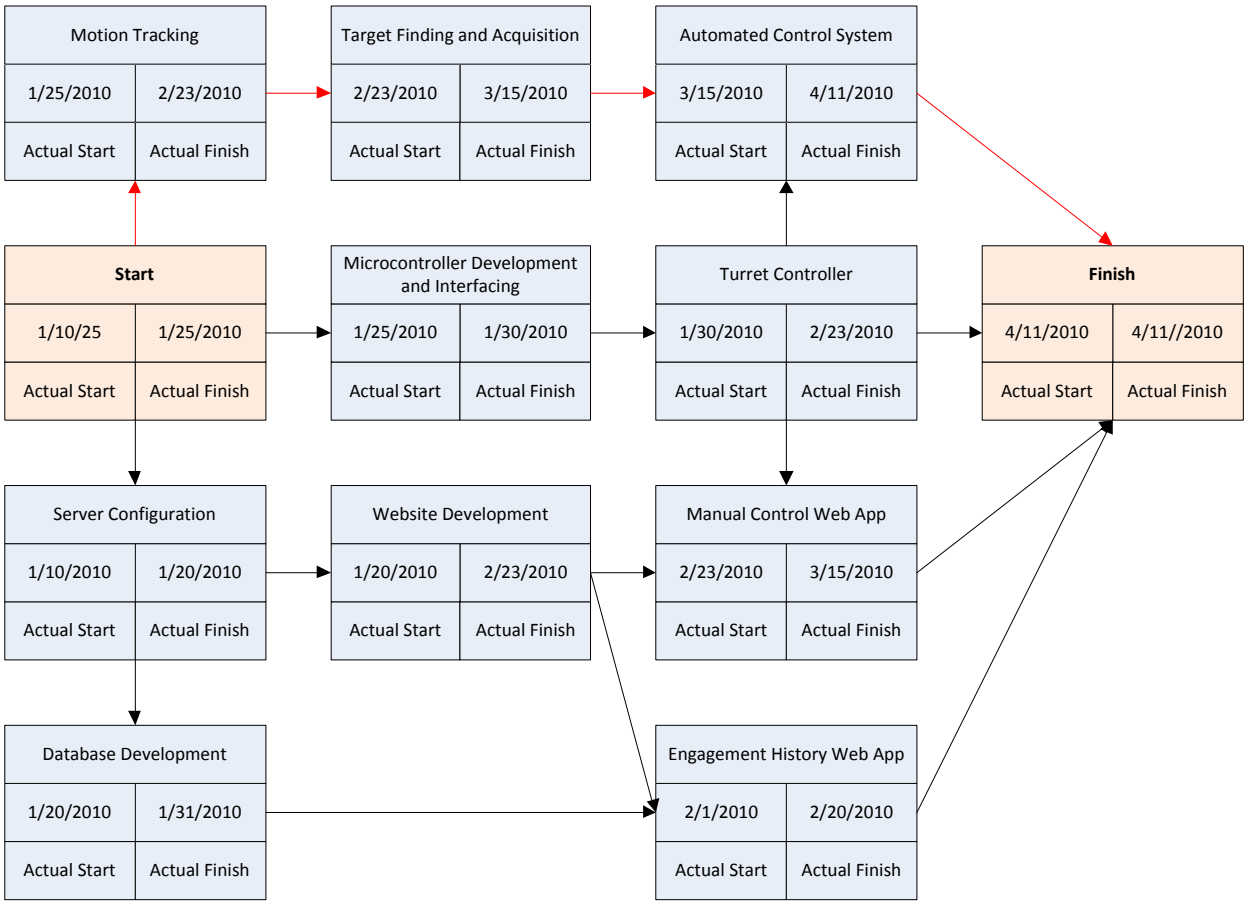

Figure 44. The PERT chart

The critical path in the PERT chart is shown in red. This shows, by our estimates on the duration it will take developing each module, that the Automated Control System will take the longest time to develop and that the motion tracking and targeting modules are along its critical path. It will be vital that these sections of our development do not fall behind schedule as they could push development of the whole system back. On the other hand, the PERT chart also shows us that the Engagement History web application has a rather large slack time that will allow us to use time set aside for it's development for more critical sections if needed.

The PERT chart also contains sections for tracking Actual Start and Actual Finish times. This will be helpful during development to track whether we are ahead or behind of schedule. We will also use this PERT chart to grade our performance after the project has been completed.

3.10 Explicit Design Summary

Below in **Table 8** is the list of components that have been chosen in our design, along with the combined list of important schematics and diagrams for quick reference.

Computer	Asus 1015PN Netbook
Cameras	6LED USB Webcam (x2) Logitech 2 MP Webcam Pro 9000 (x1)
Range Finder	Fluke 411D Laser Distance Meter LR3 Laser Rangefinder Interface Board by Porcupine Electronics
Pan / Tilt Servos	HS-5745MG ¼ Scale Servo (x2)
Trigger Servo	HS-485HB Servo
Microcontroller chip	ATmega328
Gun	UTG Brand CQC MP5A Spyder Shutter Java Edition
Battery	UPG UB1280 12V Lead Acid Battery
Base	Self Machined
Microcontroller Circuit	Self Designed
Development Microcontroller	Arduino Uno

Table 8. Hardware Component List

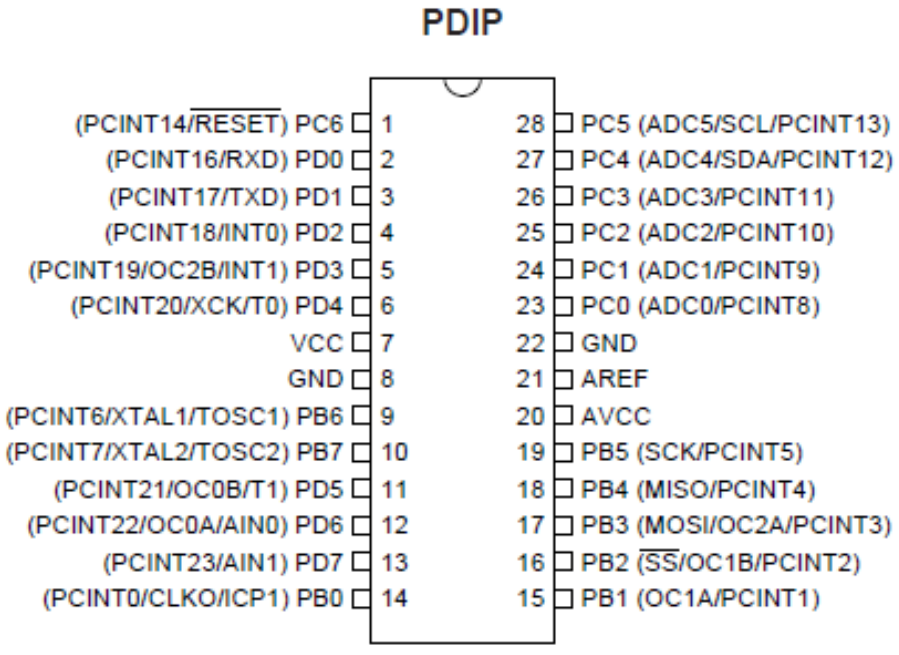

Figure 45. Pin layout of ATmega328
Copyright Atmel Corporation. Reprinted with permission.

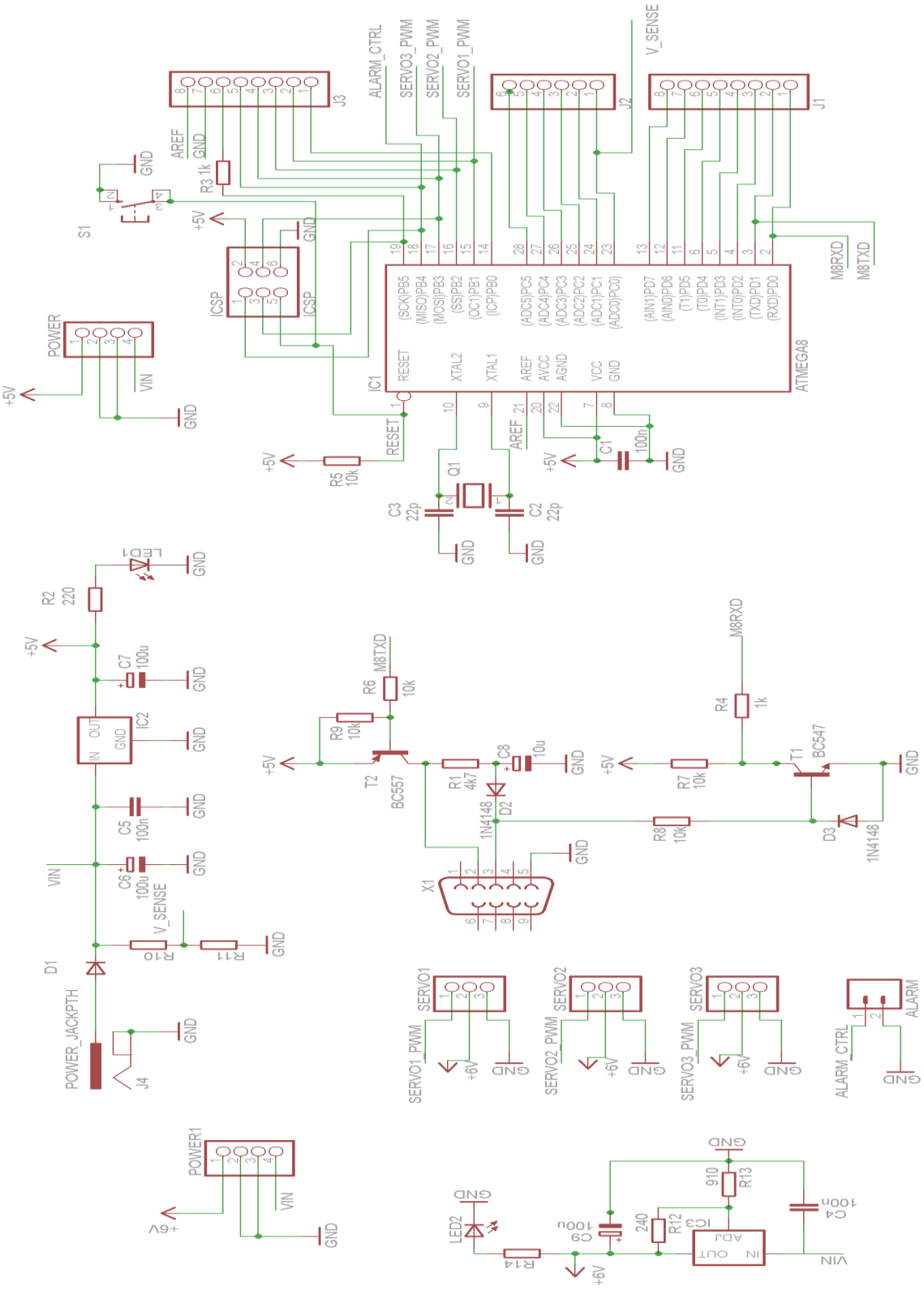

Figure 46. Schematic for our custom Arduino board

PART CODE	Description
POWER_JACKPTH	Standard 5.5mm barrel jack for main power from DC converting wall wart or battery
D1	Diode to prevent negative current if power is hooked up incorrectly
V_SENSE	Voltage divider circuit to sense battery voltage
IC2	7805 Voltage regulator to obtain 5V board power at 1A max
LED1	LED to indicate board power is on
IC3	LM338T Voltage regulator to obtain 6V servo power at 5A max
LED2	LED to indicate servo power is on
S1	Reset Button
SERVOx	Breakout pins for servo motor connections
X1	Serial connection from computer
Q1	Oscillator
ICSP	In Circuit Serial Programmer
ATMEGA8	8 Bit ATmega microcontroller. Any 8 bit chip will do, but we will use a ATmega328

Table 9. Key for labels in Schematic Figure 46.

Technical specifications	Fluke 411D
Range (for extended distances, use a target plate)	0.1 m to 30 m (0.33 ft to 100 ft)
Measuring accuracy**	± 3 mm (0.118 in)
Units displayed	00.000 m, 000 ft 00 in 1/8, 000.00 ft
Laser class	II
Laser type	635 nm, < 1 mW
Automatic power off	after 180 seconds
Continuous measurement	•
Addition/subtraction	•
Battery life	up to 3,000 measurements
LCD illumination	–
Data locations	–
Min/Max	–
Audible feedback	–
Pythagoras (Indirect measurement)	Simple
Ingress protection	IP40
Dimensions	123 mm x 50 mm x 26 mm (4.84 in x 1.97 in x 1.02 in)
Weight	150 g (5.29 oz)
Temperature range Storage Operation	 -25 °C to 70 °C (-13 °F to 158 °F) 0 °C to 40 °C (32 °F to 104 °F)
Operating altitude (ISO 9022)	up to 3500 m
Storage humidity (at 35 °C)	maximum 85 % for 24 h
Batteries	AAA (2)

Figure 47. Datasheet of our rangefinder component
Copyright Fluke Corporation. Reprinted under Fair Use

Specifications

System Model	ASUS 1015PN
Operating System	Windows® 7 Ultimate Express Gate
Architecture	32 bit (x86 based)
Display	10.1" LED Backlight WSVGA Screen (1024x600)
CPU & Chipset	Intel® Atom™ N550
	1.5 GHz 2 Core(s), 4 Logical Processors (Hyperthreaded)
Memory	SO-DIMM 2GB DDRIII
Wireless Data Network	WLAN 802.11b/g/n @2.4GHz Bluetooth 3.0
Display Card(s)	Next-Gen. ION N11M-PT1 with 512MB VRAM
	Integrated Intel Graphics
	NVIDIA Optimus Graphics Switching
Storage	250GB HDD
Camera	0.3M Pixel
Audio	Hi-Definition Audio CODEC Stereo Speakers Mic.
Input / Output	1 x VGA Connector 1 x HDMI port 3 x USB 2.0 1 x LAN RJ-45 2 x Audio Jack (Headphone / Mic-in) 1 x Card Reader: MMC/ SD(SDHC)
Battery	6-cell Li-ion Battery Pack- 6cells
	10hrs (6cells, 56W/h) battery life.
Dimensions	262x178x23.6~36.4mm
Weight	1.25Kg

Table 10. Specifications of ASUS 1015PN Netbook

Specifications

System Model	Dell XPS m1730
Operating System	Windows® 7 Ultimate
Architecture	64 bit (x64 based)
Display	17" UltraSharpTM Wide Screen WUXGA (1920x1200) display with TrueLifeTM
CPU & Chipset	Intel® Core™2 Duo Processor T8300 (2.4GHz, 3MB Cache, 800Mhz FSB)
Memory	4GB Dual Channel 667MHz DDR2
Wireless Data Network	WLAN 802.11b/g @2.4GHz
	Bluetooth 2.0
Display Card(s)	NVIDIA® SLITM Dual GeForce® 8700M GT with 512MB GDDR3 Memory
Storage	120GB HDD
Camera	2M Pixel
Audio	5.1 Channel Digital HD Audio
	Stereo Speakers
	Mic.
Input / Output	1 x Dual-Link DVI-I port
	1 x S-video out port
	1 x Firewire (IEEE 1394a) port
	4 x USB 2.0
	1 x LAN RJ-45
	3 x Audio Jack (2x Headphone / 1x Mic-in)
	1 x Card Reader: MMC/ SD(SDHC)
	Consumer IR sensor
	ExpressCard 54 mm slot
	DVD+/-RW with Dual Layer DVD+R write capacity
Battery	9-cell 85WHr Li-Ion Battery
Dimensions	406x50.7x302.6mm
Weight	4.81 kg

Table 11. Specifications of Server Computer

Figure 48. Crosman Mini Carbine with fully automatic capability.

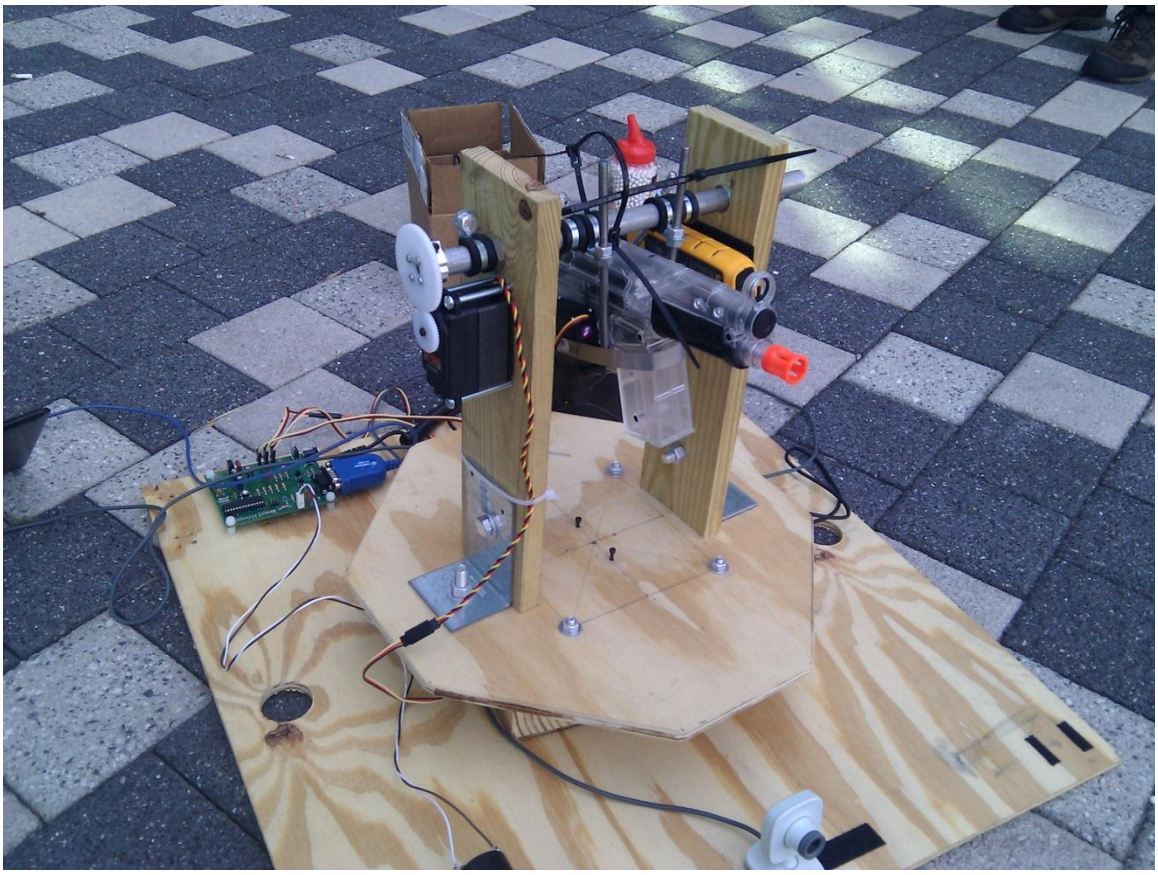

Figure 49. Final base construction

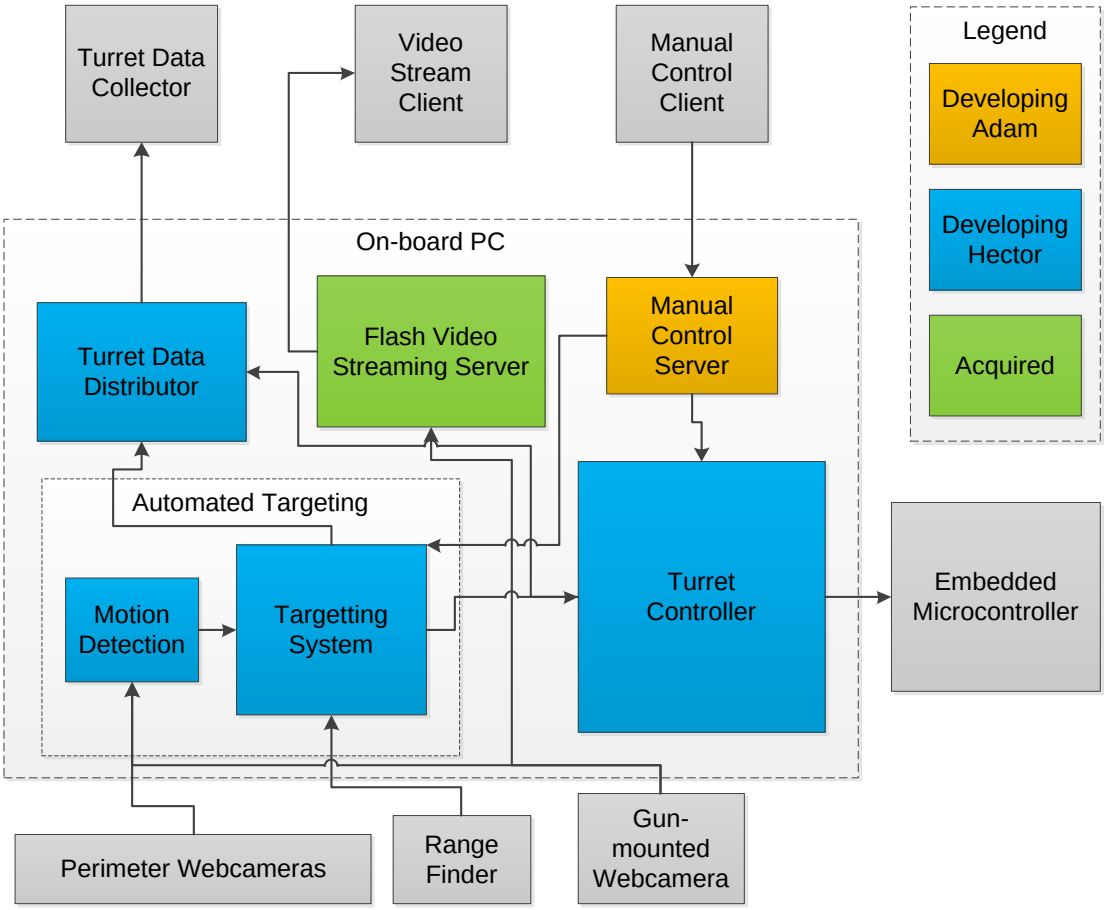

Figure 50. Block diagram of the general on-board software design.

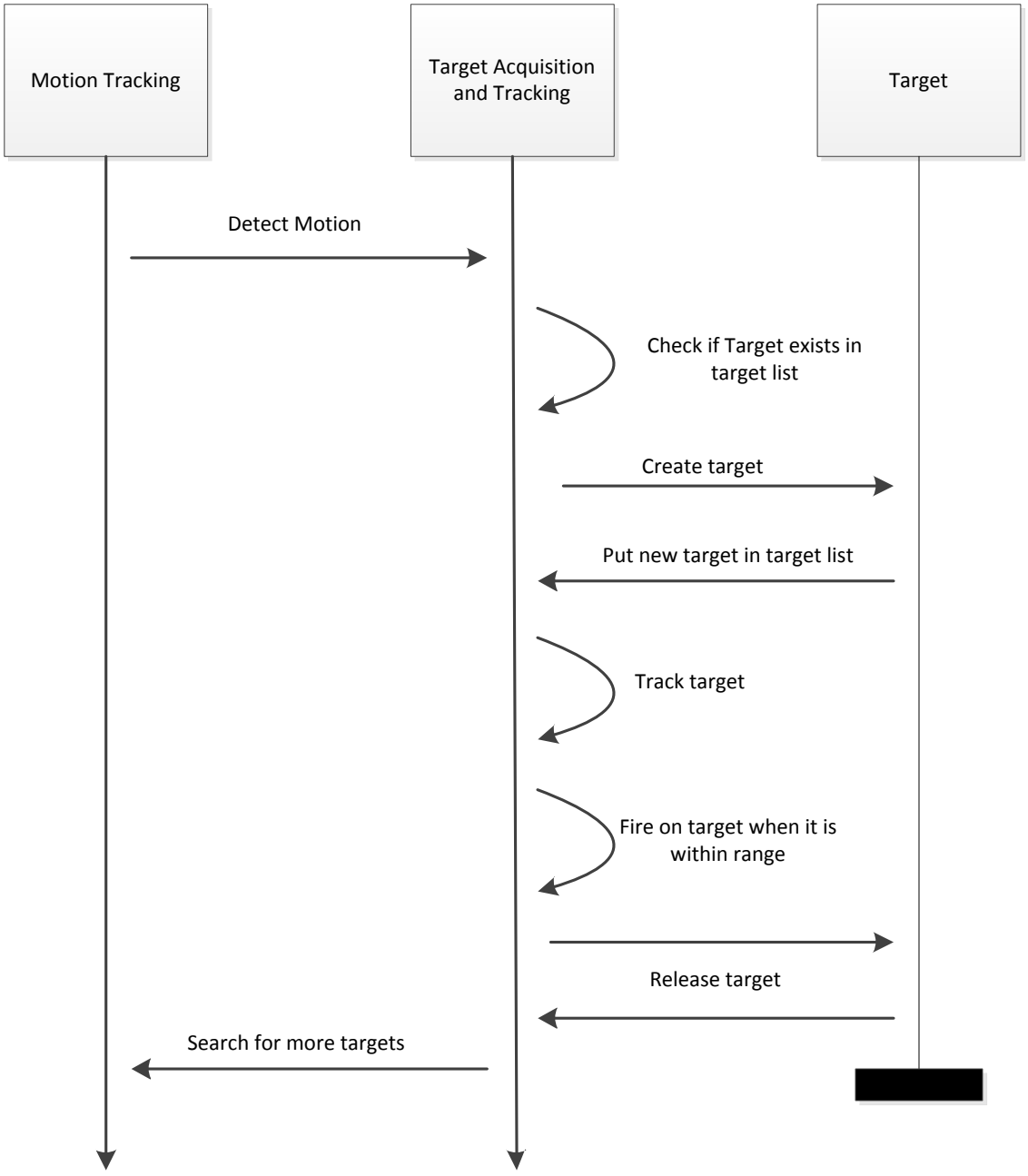

Figure 51. Target Acquisition Activity Diagram

Motion Detector	
stream : videoStream prevFrame : frame currFrame : frame difference : int	detectDifference(previous : frame, now : frame) : void motionDetected() : int

Figure 52. The Motion Detector Class

Targeter	
targetList: target[ ] curretTarget: Target	aquireTarget() : target trackTarget(target) : void releaseTarget(target) : void getDistance(target) : int hasTarget() : boolean

Figure 53. Targeter Class Diagram

Target	
ID : int angleLocation : int pitchLocation : int distance : int timeAcquired : int timeReleased : int shotsFired : int active : boolean	Target() : void getAngleLocation() : int setAngleLocation(int) : void getPitchLocation() : int setPitchLocation(int) : void getDistance() : int setDistance(int) : void getTimeAquired() : int getTimeReleased() : int getShotsFired() : int shotFired() : void release() : void

Figure 54. Targeter Class Diagram

Automated Control
getAngle() : int setAngle(int) : void getPitch() : int setPitch(int) : void isMoving() : boolean startSiren(int) : void StopSiren() : void isSiren() : boolean Fire(int) : void isFiring() : boolean

Figure 55. Automated Control Class Diagram

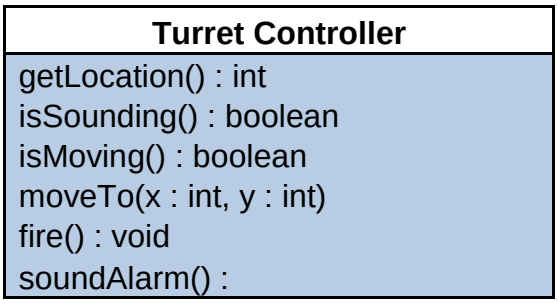

Figure 56. Turret Controller Class Diagram

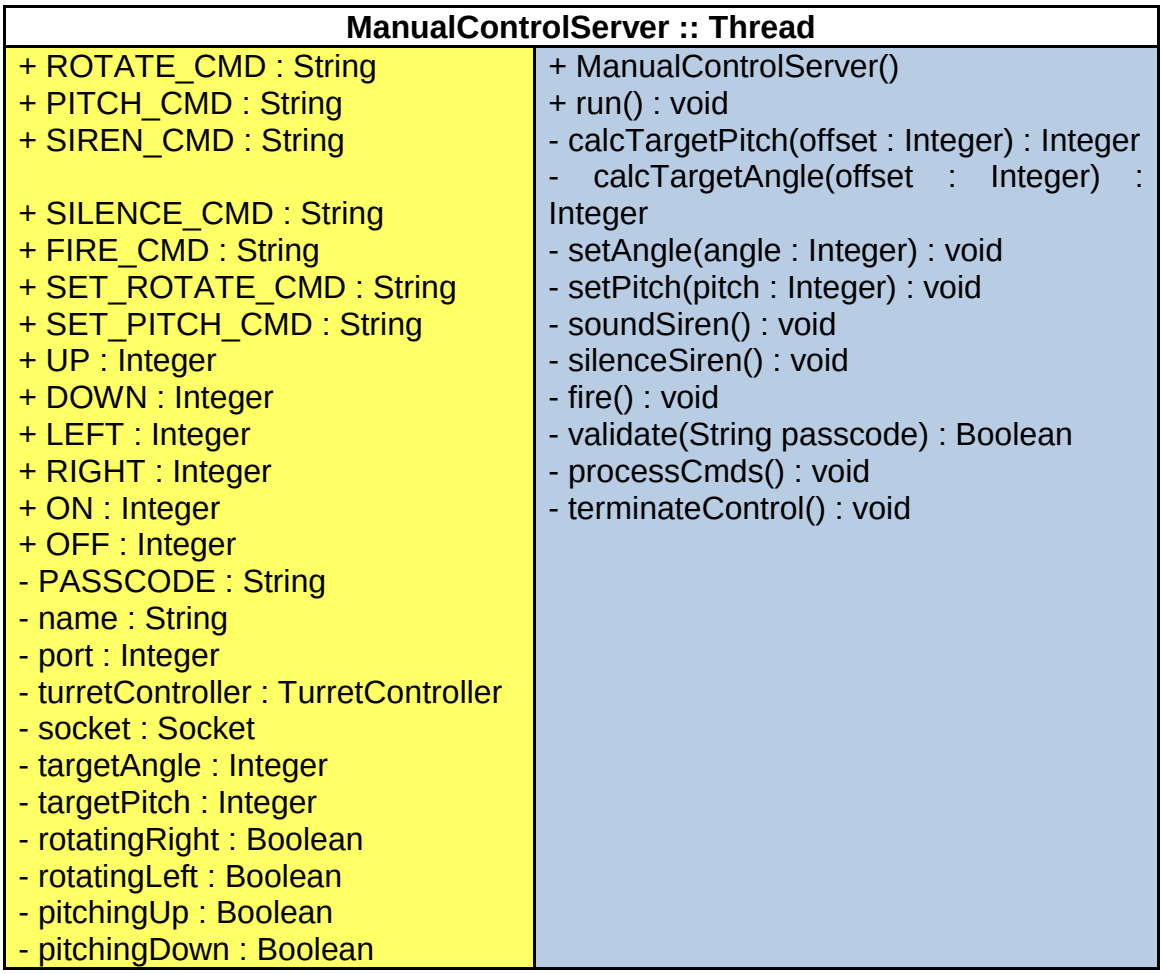

Figure 57. C# class diagram for the ManualControlServer class.

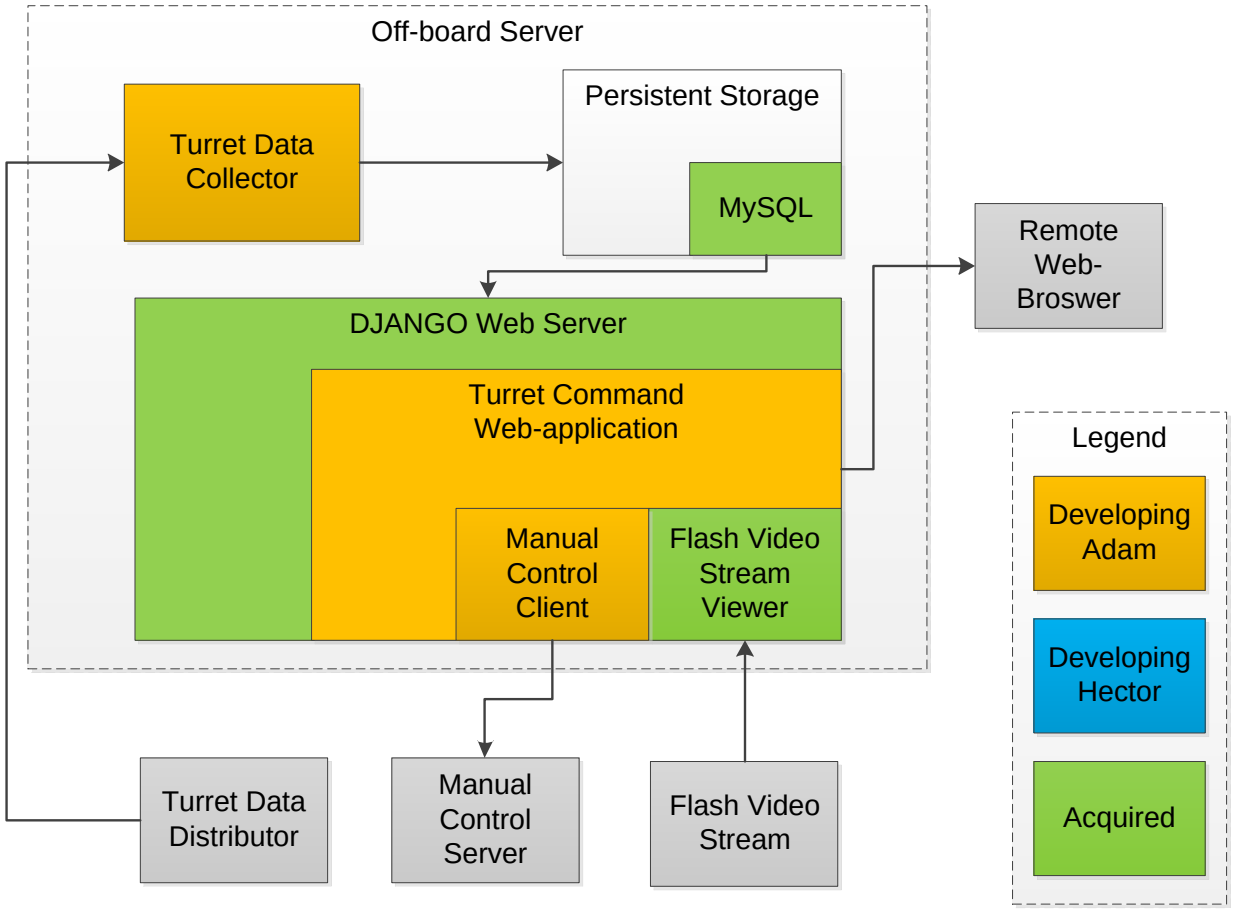

Figure 58. Block Diagram of the general off-board server software design.

Turret :: Model
turretId : String name : String latitude : String longitude : Double orientation : Double angle : Integer pitch : Integer

Figure 59. Python class diagram for the Turret class.

TurretTarget :: Model
targetId : String turretId : String distance : Integer angle : Integer

Figure 60. Python class diagram for the TurretTarget class.

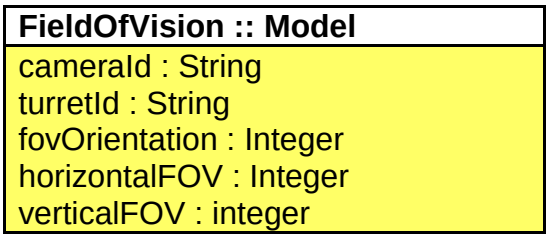

Figure 61. Python class diagram for the FieldOfVision class.

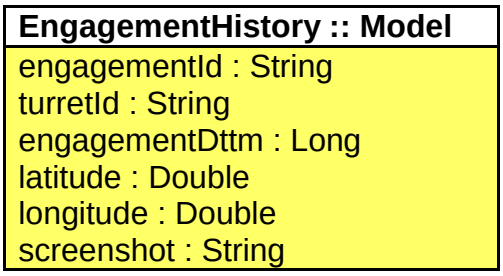

Figure 62. Python class diagram for the EngagementHistory class.

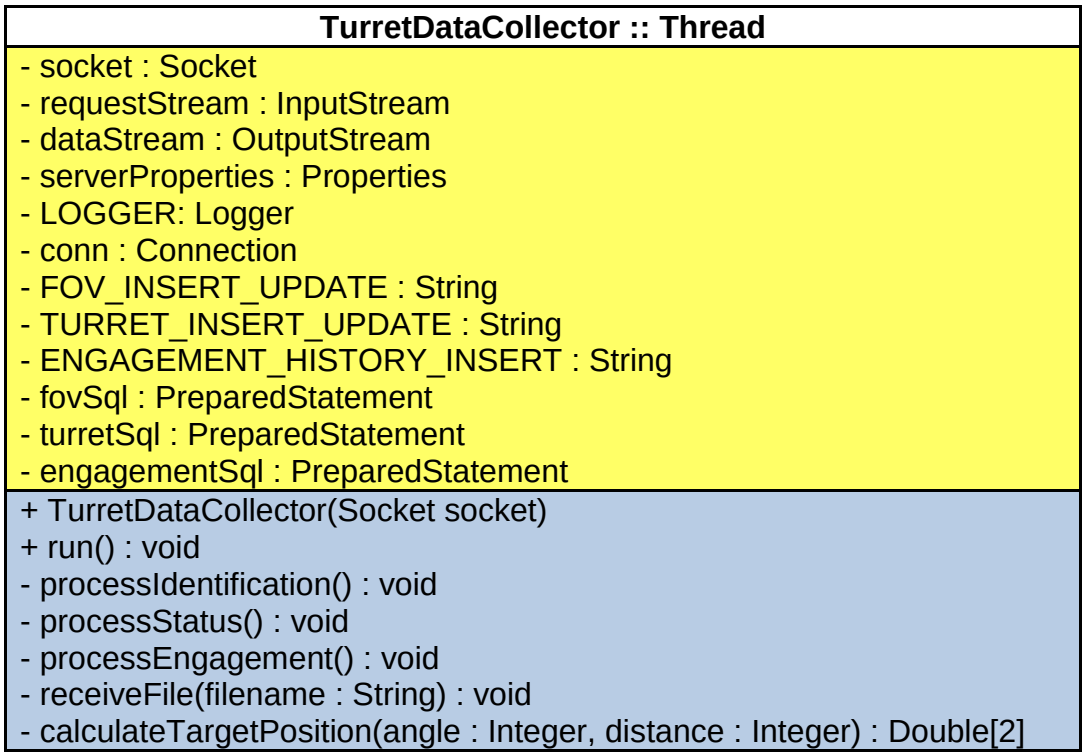

Figure 63. Java class diagram for the TurretDataCollector class.

DBUtil
+ getConnection(username : String, password : String, connectionString : String) : Connection + <?>executeAggregate(conn : Connection, query : String) : ? + executeQuery(conn : Connection, query : String) : List<Map<String, Object>> + executeStatment(conn : Connection, query : String) : Integer

Figure 64. Java class diagram for the DBUtil.

ManualControlApplet :: Applet	
'+ ROTATE_CMD : String	'+ ManualControl()
'+ PITCH_CMD : String	'+ start() : void
'+ SIREN_CMD : String	'+ stop() : void
'+ SILENCE_CMD : String	'+ keyDown(e : Event, key : Integer) : void
'+ FIRE_CMD : String	'+ keyUp(e : Event, key : Integer) : void
'+ SET_ROTATE_CMD : String	'+ keyPressed(e : Event, key : Integer) : void
'+ SET_PITCH_CMD : String	'+ mouseClicked(e : MouseEvent) : void
'+ UP : Integer	'+ mouseMoved(e : MouseEvent) : void
'+ DOWN : Integer	'- moveLeft() : void
'+ LEFT : Integer	- moveRight() : void
'+ RIGHT : Integer	'- moveUp() : void
'+ ON : Integer	'- moveDown() : void
'+ OFF : Integer	- shiftLeft(offset : Integer) : void
'+ SPACE_KEY : Integer	- shiftRight(offset : Integer) : void
'+ S_KEY : Integer	- shiftUp(offset : Integer) : void
'+ ESCAPE_KEY : Integer	- shiftDown(offset : Integer) : void
'+ Q_KEY : Integer	- stopLeft() : void
'+ LEFT_ARROW : Integer	- stopRight() : void
'+ RIGHT_ARROW : Integer	- stopUp() : void
'+ UP_ARROW : Integer	- stopDown() : void
'+ DOWN_ARROW : Integer	- setPort(portNum : Integer) : void
'- PASSCODE : String	- setHostname(hostname : String) : void
- ROTATE_SPEED : Integer	
- PITCH_SPEED : Integer	
'- hostname : String	
' - port : Integer	
- oldXPos : Integer	
- oldYPos : Integer	
- socket : Socket	
- commandStream : InputStream	
- outputStream : OutputStream	

Figure 65. Java class diagram for the ManualControlApplet class.

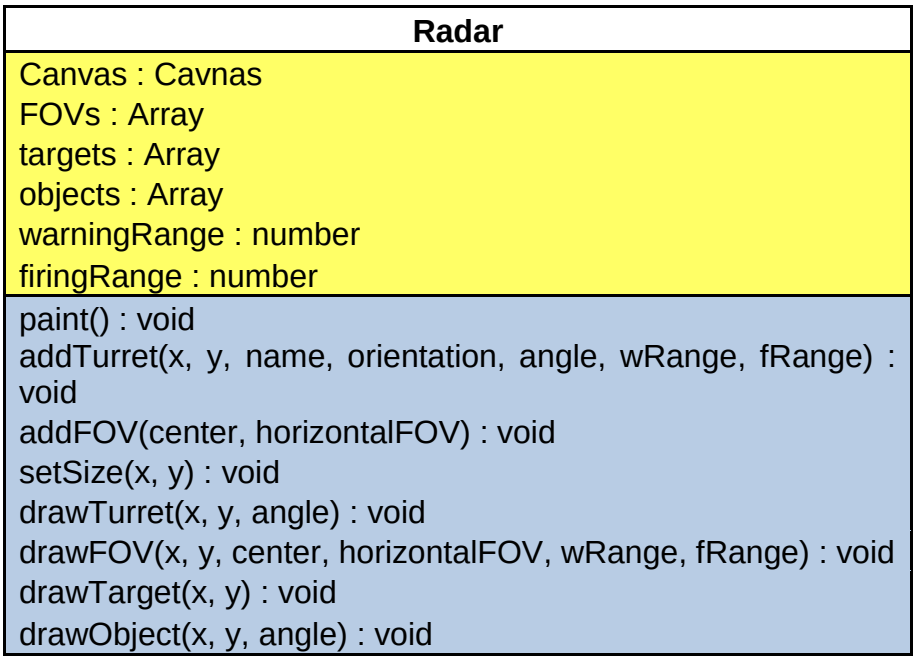

Figure 66. Javascript class diagram for the Radar prototype object.

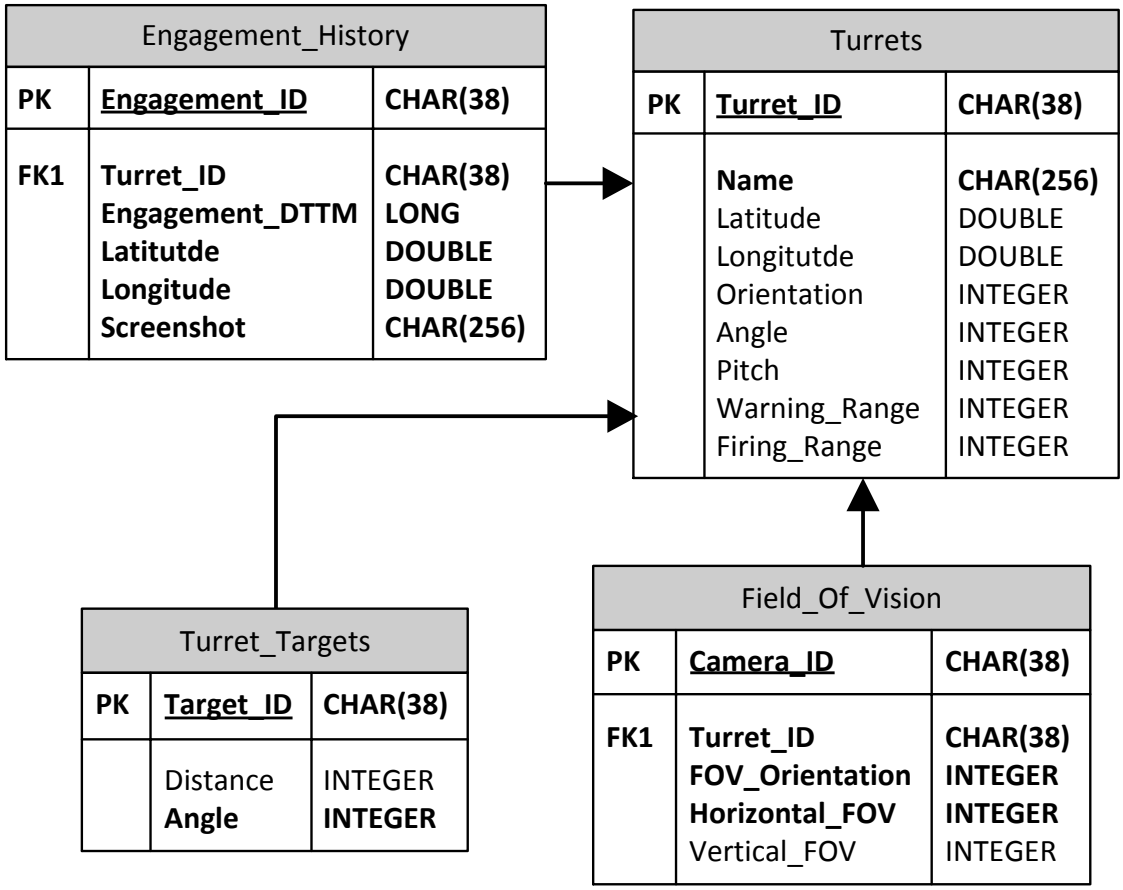

Figure 67. Database Model Diagram of the MySQL database.

Section 4: Prototype

4.1 Costs and Financing

The cost of the paintball gun is unknown because it was acquired about 7 years ago. So the paintball gun was at no cost to the group. Some items needed to be acquired for the gun to become operational once again though. Kyle bought a replacement barrel for the gun which was a 14” length longer than the generic 9” barrel that came with the original kit. There will also be a bottle of gun lubrication provided for use for the duration of project. He also is providing 500 paintballs and a kit of O-rings for the group at no cost to the group as a whole since the paintball gun will continue to be the property of Kyle at the end of the prototype testing.

The cost of the 500 paintballs was $15.99, and the 10 pack of O-rings were $4.99. These prices do not include taxes. The barrel cost a total of $28.49. The lubrication cost a total of $4.49.

Item	Cost
Servo Motors	$170.00
Rangefinder	$100.00
Rangefinder Control Board	$150.00
Test Webcam	$6.00
Turret Arm Base Materials	$50.00
Arduino Uno Microcontroller	$35.00
(2) Fabricated PCB	$110.00
Audible Alarm	$33.00
Base Materials	$60.00
Airsoft Rifle	$20.00
Battery	$43.00
Sum of Components	$727.00

Table 12. Cost of components

4.2 Stand and Servo Assembly

After our group has ascertained all of the components that our autonomous turret required as deemed by this prototype design document they will have to be properly assembled together. First our stand will be required to be completed for us to take any further actions in assembling our turret. Once the stand is complete, our group will take the servos that we will have obtained and place them in their appropriate place. One of the HS-5745MG Digital ¼ Scale servos will be in direct contact to a gear of the same size on the base of the stand and will have a 1:1 gear ratio which will control the pitch of the turret. This assembly

of the servo is similar to the way we will have the yaw adjustment connected but it will be located towards the top of the stand.

The stand will also have a machined portion out where the gun will sit and our group will attach the gun by any means necessary. There will be a third servo mounted on the gun itself around the trigger and will control the pulsating of the trigger. All of the servos will be connected to our microcontroller the Arduino Uno that we have adapted to our use of our turret. The microcontroller will be connected via a power source cable. So the summary of the servo and microcontroller connection is just that the servos are directly connected to the microcontroller and the microcontroller has to be powered by the power cable that comes with the board.

Our netbook will be connected via fast USB and the code that our group members have written will be written to the board. The rangefinder will be connected to the Arduino Uno through the ports available on the board. The use of a tripod or some other elevating stand will be used if necessary but will most likely be left out of this design since an intruder will not be able to get within reaching distance of the turret and thus there seems to be no need for an elevating type of stand.

4.3 Camera and Sensor Assembly

The web cameras that we will have obtained for this design prototype is one that is HD the Logitech 2 Megapixel Webcam Pro 9000 and two others that are LD. The HD web camera will be directly affixed to the barrel of either the paintball or airsoft gun. Since web cameras are meant to be mounted on laptop monitors or desktop monitors we will have to alter the web cameras and attach them to our turret in their respective places by any means necessary. The two other LD web cameras will be affixed to the base.

Since the web cameras only have one cord coming out and it is a USB connection, they will have to be connected to our netbook that we will be using. The laser rangefinder that we will be using along with our autonomous turret will be taken out of the casing such that we can attach it via our Arduino Uno microcontroller. Since we will be taking the rangefinder out of the casing we will have to take care in not damaging the circuitry on the board when attaching it to the turret. The rangefinder will also be attached to the base and will determine the range at which the target is located.

4.4 PCB Fabrication Companies

When chosing a fabrication house to create our printed circuit board, there are several good options. 4pcb.com, expresspcb.com, and pcb123.com all offer discount prices on low quantity orders and orders for students. When the layout

is complete, our board will most likely need only 2 layers. 4pcb has flat rate pricing on individual boards for students, with the price at $33 dollars per 2 layer board, with a board area of up to 60 square inches. Expresspcb has a more complex pricing scheme for individual board sales, with the cost being equal to this formula: $55 + ($0.65 * NumberOfBoards * BoardAreaInSquareInches) + ($1.00 * NumberOfBoards) + Shipping. **[30]**. Pcb123 also offers single order boards, but the pricing is not competitive with the previous two manufacturers in our case.

We plan to buy two copies of our designed board, so that a spare will be ready should any problems arise. With this in mind, buying for 4pcb will be the most cost effective solution for any reasonable sized board we may need, from 3x3 inches, to even as large as 6x10. This will bring the board to $66 dollars, plus the cost of components.

4.5 PCB Design Software

For PCB layout software, there are several choices. The most used software for PCB design is EagleCAD, which offers a freeware version with limitations. The software contains extensive libraries with many parts from numerous manufacturers, making it a solid choice for the layout environment. The schematic of our self-designed microcontroller board was created using EagleCAD, and it is a somewhat familiar environment. However, there are limitations. The size of the board can only be a maximum of 100 x 80 mm (4 x 3.2 inches)**[31]**, and the board can only be two layers. If these limitations need to be broken, different software will have to be used. In the board layout developer, each component of the schematic is displayed as it's corresponding physical dimensions. Each part is placed on the board, and traces are drawn to meet the necessary connections. Each layer of the board is shown in a different color. There are options to attempt auto trace placement, but most often the board will be designed by manually placing the traces, and then editing the layout to fix errors in trace rules. Below in **Figure 68** is a screenshot of the EagleCAD environment.

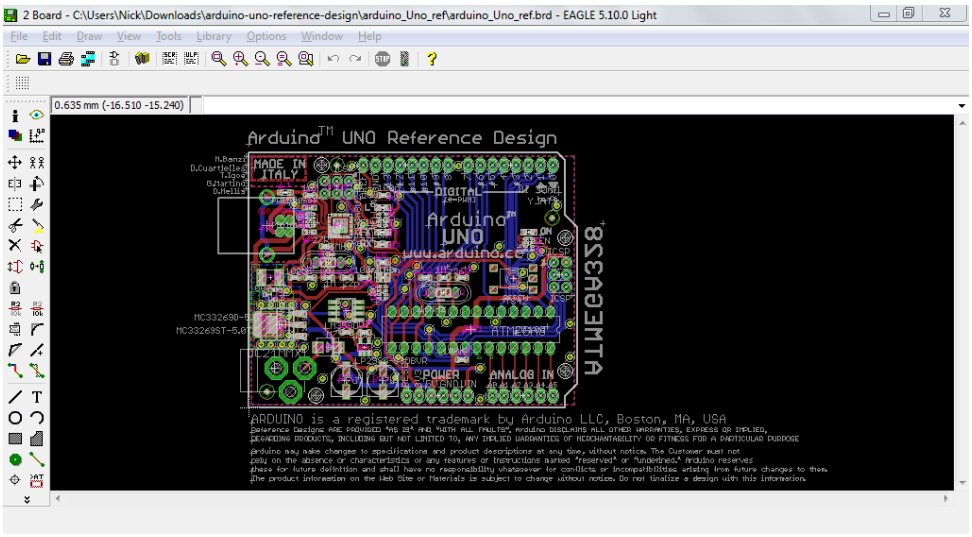

Figure 68. Screenshot of EagleCAD Board Layout Environment Copyright Cadsoft Computer. Reprinted under Fair Use

While EagleCAD is a known industry standard for a PCB Layout environment, it is not without its flaws. The interface is aging, and has not been majorly updated in many years; this shows with its lack of task to task flow in certain areas. The main drawback of the EagleCAD software is the steep limitations of the Freeware version. The professional license has no limitations, but costs thousands of dollars and is not within our group's reach. This brings us to look for a new alternative with less restrictions, that is still free. What we found is the PCB Artist Layout Software, by Advanced Circuits. This software is made by the same company that we plan to have our PCB manufactuered by. This gives it several advantages. The first is that it is well supported by customer service of Advanced Circuits. The company has a technical support line dedicated to the software, and this number is displayed clearly at the top of the software window. The second advantage is that files created with PCB Artist are the prefered type to be manufacturered by 4pcb.com. The company also provides file checking and rule checking for free before designs are submitted for manufacturing, to make sure that layout mistakes have not been made. The libraries are not as expansive as EagleCAD, but it is still very robust. Finally, the interface of the IDE is much more modern and flows easier than EagleCAD, which is convenient to work with. The biggest disadvantage of the PCB Artist software is it is unable to import EagleCAD files. This means that our group will have to remake the schematic of the PCB from scratch in PCB Artist, which will take more time. Although EagleCAD has been used for the original schematic of our PCB, we will use PCB Artist for the rest of the PCB design and layout. Below, **Figure 69** shows the PCB Artist environment.**[16]**

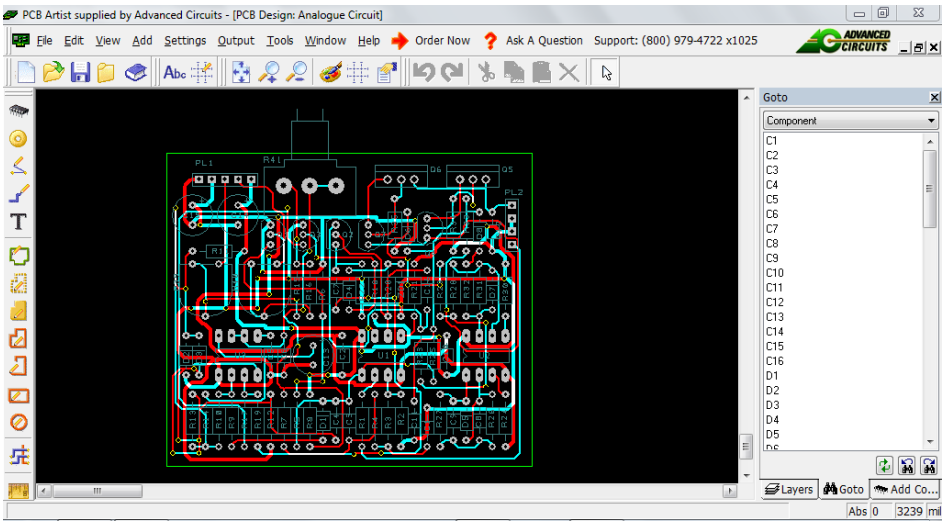

Figure 69. Screenshot of PCB Artist Layout Software environment Copyright Advanced Circuits. Reprinted with permission

4.6 Printed Circuit Board Assembly

A printed circuit board will need to be assembled and implemented in the design for use as a microcontroller circuit. The majority of the use of this circuit will be as a motor control board. Since the design will be only two layered and not largely complex, two identical boards will suffice for the prototyping process. The first board will be populated and used in the prototype, while the second will be stored as a backup in case of manufacturing flaws or part malfunction. Each component of the board design will be obtained in a set of five, as extra protection against individual component failure.

The circuit board itself will be designed in a PCB CAD software such as EAGLE, and submitted to a PCB manufacturing company. To obtain a reasonable price for only two purchased boards, the board will be manufactured as a piece of a pallet, along side other individual boards. This causes constraints in dimensional area, as well as the limitation of two layers, which have both been considered in the design process. Having the PCB manufactured will allow for a much more dense and precise design than any feasable home made techniques can create.

There are two accessible techniques to populating the PCB. The first is to populate it by hand soldering each component to the board, and the second is to solder the components using a stencil technique. Using a stencil with solder paste is a very fast and efficient technique to solder many boards of the sametime, however since only one or two boards will need soldering, using this technique will not be efficient. One of the group members Nick has solid soldering experience, including surface mounting. With this capability in mind, hand soldering will be the most reasonable approach to PC population.

4.7 Web Application User Interface and Flow

The web-application provides three primary functions: engagement history, live radar, and manual control of the servers. These functions are available through a primary layout of five different pages. These pages include a home page, filterable engagement history page, turrets page, manual control page and radar page.

4.7.1 Home Page

The application begins at the home page. **Figure 70** displays the prototype for this page. On this page the user is able to access the turrets page and the engagements page. Also presented on the page is a grid view of the 6 latest engagements across all connected turrets. The latest engagement is located in top left and the 6th latest engagement is located in the bottom right. Each engagement in the grid displays the screenshot at the time of the engagement from the gun-mounted camera, the turret's common name that fired the at the target, and the time of the engagement. Clicking any of the engagement tiles will route the user to the engagements page with that specific engagement selected to view additional details.

Figure 70. Prototype of Home page's user interface and look.

4.7.2 Engagements Page

The filterable engagements page as shown in **Figure 71** is a two pane setup. On the left pane, all the engagements after the filter are listed with the name of the turret and the time of the engagement. By clicking on a item in the left pane the details of the engagement are displayed on the right pane. In addition to

repeating the turret name, date and time of the engagement, the details pane includes the screenshot of the target at the time it was fired upon. Also in the details pane is a radar representation of the relative positioning of the target. The engagements page is filterable on turret Id in order to only view the engagements encountered to a specific turret.

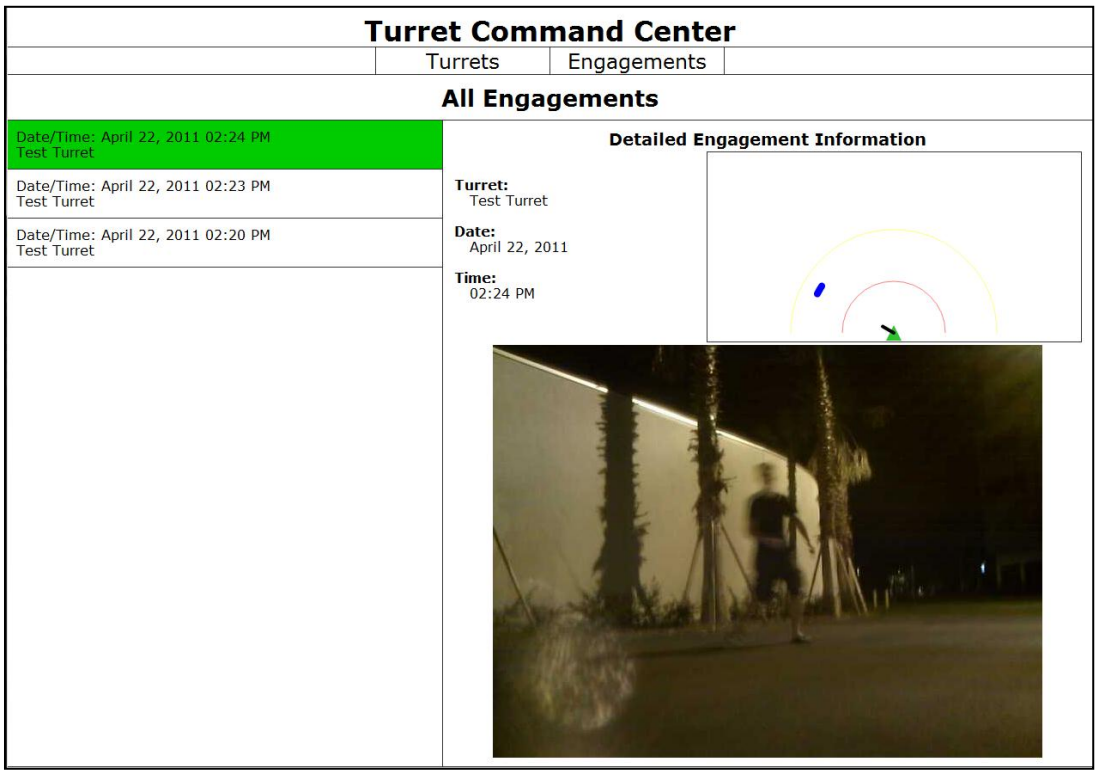

Figure 71. Prototype of Engagements page's user interface and look.

4.7.3 Turrets Page

The turrets page provides the gateway to the turret-specific engagement history page and the manual control page for an individual turret. **Figure 72** shows what is visible on this page. The page presents all the turrets that are connected to the server in a grid view. Each tile in the grid displays a screenshot from the initial set-up time of the turret, the turret's common name, a checkbox, and a link to view the turret-specific engagement history. Clicking on the turret name or screenshot will navigate the user to the manual control page. A user may also check any number of turrets and click the Watch selected button in order to navigate to the Radar page.

Figure 72. Prototype of Turrets page's user interface and look.

4.7.4 Manual Control Page

The manual control page, shown in **Figure 73**, displays the radar of the turret and a video stream of the gun-mounted camera. The **ManualControlApplet** provides the ability to take manual control of the turret. Once manual control is started, the user is able to control the turret using either the mouse or keyboard. The **ManualControlApplet** also displays information about the turret's current state. All throughout the time on the page, the user is able to view a live-video stream of from the turret. Also during the manual control the radar will still be partially functional. The radar will display the current positioning of the turret in the total range possible, and display suggested objects displaying motion, however because the turret is not able to determine the distance, the targets can never be verified if they are in the warning or firing range.

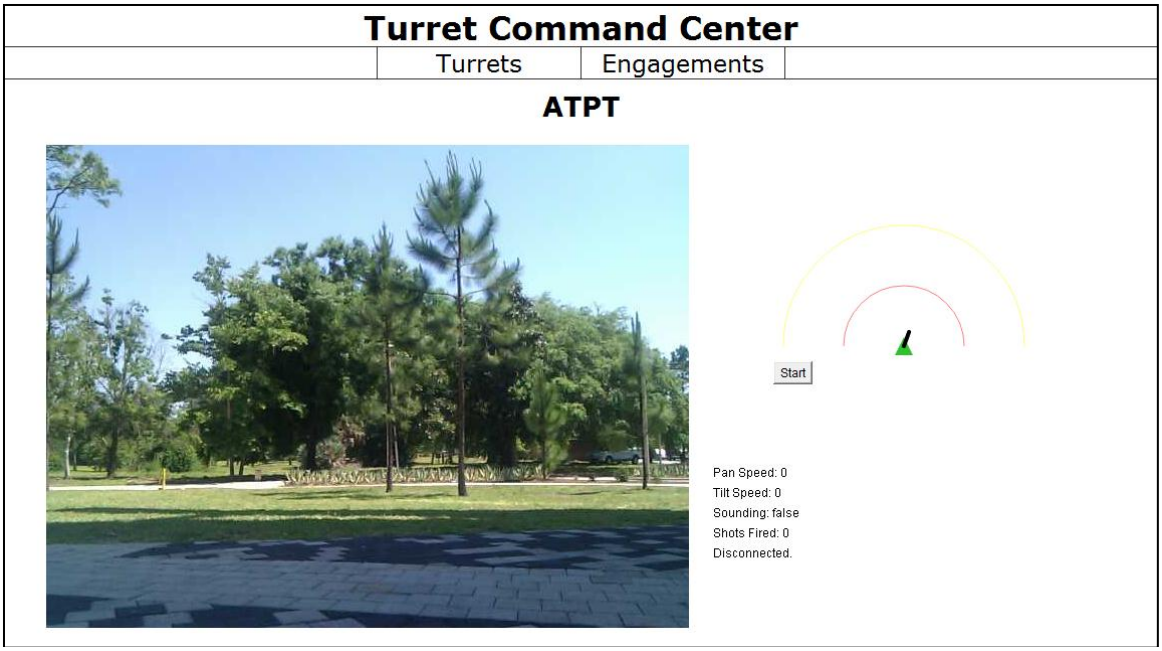

Figure 73. Prototype of the Manual Control page's user interface and look.

4.7.5 Watch Page

The watch page offers a simultaneous view of multiple turret Radar objects. **Figure 74** displays the layout of the page. The Radars are displayed in a grid. The turrets to view on the radar are specified on the Turrets page. All the turrets will be updated in real-time to reflect the actual state of the turret and all identified targets.

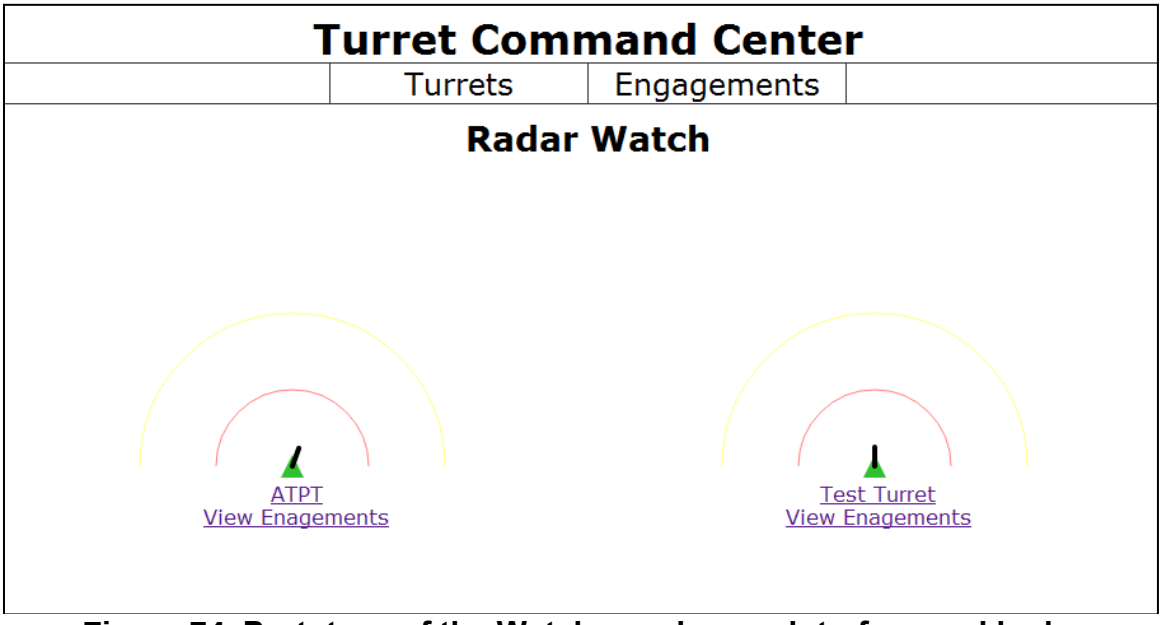

Figure 74. Prototype of the Watch page's user interface and look.

Section 5: Testing

5.1 Hardware Testing

5.1.1 Controller Development and Build Testing

Testing the microcontroller took place in two stages. The first stage was using the pre-assembled Arduino Uno to test the platform, connecting the motors one by one using a breadboard. This allowed us to begin coding and testing the microcontroller before we finish assembling the Arduino of our own design. The Arduino Uno is similar ot the one our group designed except it has onboard USB, so it will be even simpler to connect to a PC and begin programming.

The first thing we tested on the Arduino Uno was the Arduino IDE and it's ability to program the board through USB. Even though the connection was USB, the communication was still serial communication, with the device showing up on the computer as a COM port. The first thing we tested was programming the example programs that are included in the IDE, such as the LED control program. These programs can be used to test the compiling and programming features on the IDE, as well as the serial communication monitoring feature.
After the basic developmental tests were complete, the board was ready to move into tests that were directly related to the project. The Servo Library and Serial Library were both be tested individually with simple implementations, such as controlling one motor with the Servo Library using the built in servo examples, and controlling one LED with the Serial Library. After this was complete, we moved to more complex tasks, eventually controlling servo motors with the computer using a simple GUI. At this point the board was ready to be used in our design as a temporary component while our PCB design was manufactured and assembled.

The second stage in microcontroller testing was testing the group designed Arduino board, eventually bringing it to replace the pre-assembled Arduino. The same testing process used on the pre-assembled board was used on the group made board, with extra testing as needed. Since there is no on board USB, the serial connection needed extra testing. Serial to USB conversion cables are known to sometimes be inconsistent in compatability, so cautious testing involving the serial connection was needed. Also, since the board was assembled by the group, testing of each component was also needed.

First, each component and connection was tested with a multimeter to check that the board is true to design. In testing the resistors, we found that we had accidently soldered an incorrect resistor to the voltage divider of our circuit. We also tested the pins for each of the servos as well as the alarm.

5.2 Servomechanism Testing

5.2.1 Yaw/Pitch Servo Testing

Testing of the servos that control the overall operation of the turret system was a major aspect of the design of the prototype. The first test that was performed on the servos was testing the claims of the specifications from the manufacturer. You can easily use a stop watch and change the degree of the servo from one maximum to the other maximum degree angle and see if the results are near the claim of the manufacturer. After that test is performed the real test of any system is when the load is attached to the system. The premade Arduino was not able to power all three of the servos under a load so making sure that our PCB was able to handle the load was very important. The load of this test is the forces on the paintball/airsoft gun and the aluminum support arm. It is obvious that the servos will perform less when the load is attached but the performance change should be minuscule and this can also be measured in the way mentioned above.

The servos can also be electrically tested such that meet the requirements. The voltage that is required for the HS-65HB servo is in the range of 3-5 volts and has to be a peak to peak square wave. This can be verified on an oscilloscope and when the system is completely done, we can verify that our system is in this range. The current drain when the servo is not connected to a load corresponding to their respective voltages 4.8 or 6 will be 400 or 500 mA. This can be easily verified by a simple voltmeter.

5.3 Trigger Servo Testing

The testing of the HS-5745MG Digital ¼ Scale servo can be electrically tested to see when the load is placed on the servo and seeing whether or not our system is provide a 3-5 volt peak to peak square wave that is required for the pulse. The operating range of the voltage of this servo while the load is on should be between 4.8 and 6 volts.

The current drain while there should be no load placed on the servos for the corresponding voltages of 4.8 and 6 volts should be from 700 to 840 mA respectively. We also tested the specifications that our group wants to meet. This testing was accomplished by determining if the servo motors reacted fast enough (milliseconds) from the data that was provided via the visual and software components. This can also be done on a regular stopwatch or a phone stopwatch. Thus the servo testing was done in terms of mechanical and electrical aspects. If any of these test procedures are violated in terms of the test being completed then the component has failed.

Once we determined that the trigger functioned correctly, we had to mount it on the gun and test if it could pull the trigger. We kept the default value of the trigger

server at 90 degrees and slowly increased the value of the angle the trigger would go to when fired. At each iteration we checked if the trigger server pulled the trigger far enough and stopped once we were happy with the result.

5.4 Rangefinder Testing

An extensive multilevel testing procedure will be needed to evaluate the performance of the rangefinder. The first step will be to test the Fluke 411D in its unmodified state, with the control board not yet integrated, and the LCD screen intact. From there, the control board will be embedded into the system, and the range finder will be tested again. Each instance of testing will need a full array of tests that cover every possible scenario that the range finder may be put in during the use of the system.

For the first set of tests without the control board attached will take place inside. The first step will be to test the accuracy of the device at a distance of a few feet, on objects of different colors and shapes. The measurement that is displayed on the LCD screen of the Fluke 411D will be compared to measurements taken with measuring tape. The second step of this phase of testing will be to test the minimum range of the device. After this, a long range indoor test will be performed in a hallway, with several colors and shapes of targets, including a human target. This range will be compared to range with a measuring tape. Several ranges will be tested, in intervals of up to and just beyond 100 feet. After these tests are complete, the continuous measurement function will be tested by measuring in a sweep between multiple targets of known distance. The final indoor test will be to track the distance of a moving target, ultimately testing a moving human target. Ideally, the device will obtain accurate continuous measurements on the moving human target as the user manually tracks it at a variety of distances.

Laser rangefinding devices behave differently in different environments, so indoor testing will not be sufficient for our needs. The turret system should be versatile with satisfactory function in a variety of environments, and the performance of the rangefinder is an important factor in satisfying this goal. Each of the indoor tests need to be performed outdoors, where the laser suffers from interference and lowered reflectivity. Since most of the turret’s use will take place in outdoor environments, the specifications need to satisfy our goals outdoors as well.

For the final round of miscillaneous tests, the device will be put under the most stress in special case instances. The first case will be a night time test. This test can take place at night outdoors, or inside in a dark room. This will test the rangefinder’s ability to perform at night, if the camera array on the system were to be later modified for night vision. The second case will be to test view obstructions. Intense light interference from different angles will be used to see if it affects the performance. Next, a foggy environment will be used to see how

badly the rangefinding is affected. This is expected to be the breaking point of the range finder's capabilities. After this extensive gauntlet of tests is complete, the Fluke 411D will be reassembled with the control board replacing the LCD.

5.5 Web Camera Testing

Testing of the web cameras will consist of a detailed list of many requirements in order for the web cameras to complete this testing procedure. First we need to determine if the HD web camera interaction with the LD web cameras are adequate. When the autonomous turret is completely setup and operating our group needs to test to see if the interaction of the two types of web cameras is interacting in the way we want them to. By this we mean are the LD web cameras doing the overall visual recognition of the turret area and determining if there is any movement calculated with the software that our group has developed. The HD web camera should do the fine visual recognition with the higher megapixels that the camera has to offer. Another portion of the testing procedure for our web cameras is in relevance to the firing of the turret. Our group needs to determine if from the time of visual acquisition to the firing of our turret is in the milliseconds. This can be done with any stopwatch and the autonomous turret can be tested as such.

There is also the testing of when the web cameras has acknowledged a moving target and has signaled the turret to fire. Our group needs to verify that the overall autonomous turret system processing is in the fractions of seconds as well. Our group needs to also verify that most importantly that the web cameras visual, software, and electrical componentry work together without any flaws and issues. This is important because if the components that we use for our autonomous turret are not compatible in these ways than we have to go back and choose other components. Verifying that these components work together will prevent time being lost in having to search replacement components. The last small test our group should perform for the web camera test procedure is the question does the web cameras we chose do what we had hoped it to do. First, for the price of the HD and LD web cameras combined we made good decisions in that we are equipping our turret with medium to high performance components for low to medium pricing. From observing what the web cameras have to provide, it seems as though what our group is looking for. The real test is when we have our autonomous turret system set up and determines if the performance is adequate in the response to an intruder in the turrets view of sight.

5.6 Software Testing

Since an agile development strategy will be employed, software testing will be an important and ongoing process through development. There will be multiple milestones during development that will require testing as the system gets more and more complex. Not only will modules have to work independently, but

unexpected errors may result in adding them with other parts of the project. This is why unit testing and systems testing will be an important part of the project.

5.6.1 Software and Hardware Communication Testing

One of the first and most important aspects of our software testing will to test the microcontroller to see if it can receive instructions and manipulate the servos correctly.

The first test that we will perform will be a base test to determine the basic functionality of the software and hardware to see if we can send messages between the two. To perform this test we will create a program that manipulates the servos to activate in different ways. They should be able to move in both directions as well as in short bursts and in more prolonged movements. Hopefully, the servos comply and react in the desired manner in a timely manner. We also have to perform a similar test with the trigger servos. A program will be written that will activate the trigger servo. The program will include single shot and rapid fire tests. In order to pass the test the servo must not only react to our commands but should be able to squeeze the trigger sufficiently so that the turret's weapon is fired. While the first two tests where designed for our own personal education on sending messages between the PC software and the microcontroller, eventually we will have to implement what we have learned into our software design and write the methods that we will actually use in the final version of our turret. The final testing of this aspect of the turret software will be done when we have written these methods. We will write a routine that calls each of the Turret Controller methods and make sure that the turret responds appropriately. This will let us know that any functionality we build on top of those core methods will have a solid foundation.

5.6.2 Motion Detection and Target Acquisition Testing

The motion detection algorithm very well may be the most complicated aspect of our project. Iterative development with multiple tests as new and improved versions roll out will be important. Standard test cases to measure and grade improvements and compare revisions will be key.

The first test that will be done after any change to the motion detection software will be a test to determine if the algorithm delivers false positives. The test case will be a video stream that contains no movement. The algorithm will be checked against the video stream that does not contain any moving objects. Simple backgrounds such as a blank wall should be tested first. Eventually "busier" backgrounds such as outdoor environments with minimal movement will be tested as well. Hopefully, the motion detection algorithm will not detect any motion and will not identify any targets. This test is useful because it gives us a way to calibrate the sensitivity of our motion detection. Also, it can also be seen as a test of our web camera because if our webcam is of too low quality, the graininess in the picture will set off motion detection even of low sensitivity. The

next step is to test if the software can identify a single target. At first, one target will move in front of the on board camera. As the tests progress, different speeds and sizes of the target will be tested as the algorithm improves. The algorithm should identify the motion of the target. Once tracking is implemented, it should also be able to follow the intended target. The last test we will perform is a test with multiple objects moving in the turrets field of vision. In order to accomplish this test we have multiple objects moving in the camera's field of view. There will be tests with objects starting to move at the same time, and with some objects starting before others. More and more objects will be added to the test as the target acquisition system becomes more robust. Our expected outcome of the test is that the motion detection should identify the biggest threat and track it. It should be able to distinguish between the two targets and only track one of them.

5.6.3 Range Finder Software Tracking Testing

The range finder and its control board and drivers are made by a two different companies so we had to do extensive tests to determine if they work together as advertised and that they each delivered the performance that they promised. To start off, we tested the stand-alone range finder. We tested it on a variety of objects of different sizes and distances away, at different times of the day and compared the range finder's results to our own results using a tape measure. The range finder should be able to position any object that we would like the turret to be able to target, track, and fire upon. Once we were satisfied with the performance of the range finder, we connected it to the control board and connected that with our PC. We performed the same tests on the same objects at the different ranges. This time we not only compared our results with the tape measure, but also with the range finder results to check if the range finder's performance was impacted by the control board.

Once we determined that the range finder and the control board both work as advertised, we then began incorporating the range finder data into our code. We tried a variety of methods to incorporate the datastream of the control board into our code but they all proved unsuccessful. First, we tried running the software the control board came with as a separate process and read in the standard output into our program. We had problems with the buffer getting full and reading in current values. Then we tried asynchronous reading where any updates to the buffer should of updated our program with the latest value. For a reason we still haven't resolved, this also did not work. Finally, we tried spawning the process as its own thread and continuously reading the buffer as it was written. This method still didn't work as we were unable to even start the control board software. In the end, we had to abandon using the range finder in our code.

5.6.4 Off-board Turret Data Collector Testing

In order to ensure that the off-board Server collects and stores the active status and engagement history of the turrets. Logging was implemented in the

TurretDataCollector java class. A simple TCP java console process connects to the server. Once the connection was established, the tester sent the expected PASSCODE to the server in order to validate the tester process. After the passcode is validated, the tester tested out each possible command string. After the command strings have been sent and processed, we validated that they were correctly interpreted by looking into the log files.

The off-board server also had to be able to correctly insert the data it receives into the MySQL server. To validate that the data was entered correctly, the state of the database was observed prior to testing. As each command was processed, the relevant MySQL tables should be queried in order to ensure that the proper data modifications were carried out according to the command that was processed.

In addition to testing the off-board server with one connected turret, the server must also be capable of handling connections from many turrets simultaneously. To test out simultaneous connections, the above process was executed simultaneous from 5 additional computers after the single turret testing was complete.

5.6.5 Off-Board Web Application Testing

The web-application located on the off-board server provides the interface to remotely access and view the state of all connected turrets. The web application consists of five primary page templates with several minor variations. To test these pages, the web-server was started, with several turrets connected and the database was prepopulated with the necessary turret configuration data.

To test the home page the URL of the web-application was visited and it was verified that the page was rendered with the latest 6 engagements based off the current data in the Engagement_History table. The correct screenshots from those engagements were also verified to be displayed on the main page. Once verified, all links exiting the page were visited to verify that they linked to the correct resulting page.

To test the engagements page the Engagements link was visited from the home page and the URL explicitly and correctly typed in the address bar of the web-browser. The engagements page was also tested to see if it properly displayed all the engagements in descending order from the date/time of the engagements and the correct turret names corresponding to the engagements were listed along with them. Several engagements were clicked on to verify that the right pane correctly displayed the turret representation of the engagement, the snapshot of the engagement, and the proper time of the engagement were listed. These were all verified off of manually calculating the expected results from the MySQL database. All links leaving the page were verified to be working properly and ensured that they result in the proper page. The URL for viewing the

engagements page with a filter on a specific turret was also visited and ensured that the engagements listed only belong to that specified turret.

To test the turrets page, the Turrets link was visited from the home page and the URL explicitly and correctly typed in the address bar of the web-browser. The turrets page also displayed all actively connected turrets with their proper common-names listed below the snapshot of their initial configuration screenshot. The turrets were also checked to see if they were ordered from the common-name. All links leaving the page were properly visited to ensure that they result in the proper page.

To test the watch page, the turrets page were checked to have more than one checkboxes marked and we pressed the “Watch Selected” button and the URL was verified to be explicitly and correctly typed in the address bar. Once at the watch page, the Radars were verified that they match the radar content with the turret that they are listed as. One of the radars was tested extensively by manipulating the data in the database to simulate the new state data received by the turret. The changes on the Radar matched up with the changes made to the state data.

To test out the manual control page, a turret was selected from the turrets page and the URL was explicitly and correctly typed in the address bar. The manual control page correctly displayed the video-stream and radar of the turret. The manualControlApplet also correctly initialized. The video-stream was validated that it was coming from the correct turret by verifying the view from the turret that is selected to control. The radar was also validated by manipulating the data in the database to simulate things happening at the turret.

Once all these pages were working, it was concluded that the web-application was correctly working as intended. However these tests did not include testing the storage of the turret data, and checking if the manual control applet worked as intended.

5.6.6 Off-Board Radar Testing

Testing of the off-bard radar began with an offline prototype developed for the local PC. Targets were displayed in the field of vision of the turret while in manual control. Single targets and multiple targets of different sizes and distances were used. The off-board radar was able to display potential targets in real time to the user. It also was able to add and remove targets from the radar as they became available. The targets were shown in the correct position relative to the turret.

After testing had been completed on the offline prototype, development started on adding the radar to the online web application. Testing the radar in the web application was a similar process. There was extra emphasis on making sure that the radar was updated in real time, or as fast as possible considering latency

issues between the server and the turret. Also, special care was made so that the radar did not interfere with the performance of other sections of the web application.

5.6.7 Off-Board Manual Override Testing

Manual controls also went through some iteration throughout the project. Testing the basic movements and the control schemes was our first priority. Only after mastering those did we allow it to be incorporated with the web application and begin remote testing.

The first test case was just a preliminary test to check the manual control methods. In order to do this a local prototype of the manual controls was made for preliminary testing. It had all of the functionality of the final version except it did not work remotely and instead was running entirely on the local machine. The test covered both horizontal and vertical movement as well as firing. In order to pass the test the turret had to respond quickly and accurately to the manual commands given. It had to work with all the available keyboard commands. Once the local prototype was working we were able to port that to the web application once development has progressed to that point. The testing of the remote manual controls in the web application was very similar to the prototype testing. The turret was able to respond to all commands given in a reasonable amount of time considering the latency between the server and turret.

5.6.8 Off-Board Engagement History

Testing the on-board engagement history began with creating some engagements. At first, these engagements were be manually generated and input to the MySQL database. We tested that the database correctly stores and delivers the information for each engagement. Once the engagement history view in the web application was developed, we tested that it correctly laid out and displayed the correct information.

Once the engagement history functions were implemented in the automated and manual controls, we tested that the turret sends its engagement history to the server's database. Everything from the engagement time and location was displayed appropriately in the web application.

5.6.9 System Integration Testing

Even if all the modules work well separately, there can be glitches when everything is put together. In complex systems with multiple parts, it is hard to visualize and predict how one system my inadvertently effect another. Because of this, even after all the modules have been tested and shown to work, system integration testing had to be done before the turret was ready for deployment.

The first systems that were coupled were the motion tracking and the autonomous movement modules. Testing was done to ensure that they are able to communicate well and preform their intended functions. Tests were done that activated the motion detection with potential targets and we checked if the turret successfully targeted them and was able to move and track the targets. Testing the tracking at different speeds and distances was done to check the limits of the automated turret.

Testing the website once the individual modules had been completed was also important. On top of unexpected errors that may arise in integrating all the different systems that are calling the same database, problems could of arisen from bandwidth problems. With multiple modules calling the turret PC for information, there was also the possibility of latency issues. Web browser performance was also measured to check that all the modules on each page do not overload it.

Finally, when both the online and offline components were working perfectly, they were combined to complete the automated turret design. The final round of testing was done just as vigorous as previous iterations. Every bit of functionality outlined in our design document plus any additional functionality that had been added or modified was tested under a variety of conditions.

Section 6: Reflections

6.1 Features Left Out

There were some features that our group left out from our original proposal at the beginning of our project. There were one or more reasons why our group decided to opt out these features from our turret system it might have had an effect on our price, time, or operation of the system in general. The main feature that comes off the top of our groups' head that we left out is the operation of the turret to operate at a 360^{o} field of view. This seemed like a great idea at first but then our group started thinking if the turret operated at a 360^{o} field of view how would it differentiate between friends and foes that are standing behind the turret. So in the end we did decide to leave that feature out but it could have been adapted to our system. Another feature that our group left out from the start after reviewing our initial design review with our professor Dr. Richie is the ability for our turret to operate at night. We all agreed that our proposal for our turret to operate in night vision is a feature that our system did not need to incorporate because our course takes place during the day and our prototype testing will be during the day as well.

Another feature that our group did not necessarily leave out but our turret does not have is the feature of automatic firing. Although this may be true we will simulate automatic firing with the use of a servo that controls the trigger and the

servo is controlled by pulsing of electronic voltage. Another feature that we left out of our system because of complexity in software coding is the proposal of color targeting/distinguishing between enemy and friend. Our components and parts that make up our autonomous turret system are to date relatively lite as one could make it on a controlled budget. An improvement on this feature is not left out again; it would just be an upgrade to our system to look for smaller system components to make it more lightweight. At the end of our prototypes completion there will be room for improvement on the data acquisition (percent error, accuracy, etc.).

Another initial goal that has been left out of this design is friendly target recognition. This adds and extensive amount of design to the project, and has a questional feasability in realistic practice. The incomplete design outlined the use of a separate gps module that communicated wirelessly through an Arduino board to the turret, relaying the location of the person in possession of the device. The system could then triangulate the position of the friendly target relative to the base system, and mark it in the vision as a friendly target. This can be done with the Arduino Uno which is already in our project for test purposes, with the addition of a GPS module, two wireless modules, and proper interface boards. The cost for this addition is steep, reaching at least $150 dollars, which is another reason this has been chosen to be left out. Also, it is known that GPS accuracy is only around ±3 meters, which makes getting accurate measurements and calculations less feasability.

Other features that we left out from our original proposal are the option that our turret would be able to vertically adjust and compensate if the Carbon Dioxide pressure is declining. This was left out because the Carbon Dioxide limit will not decrease enough to require this feature. Another great feature that was left out of our turrets' design is due to the fact that it would require more of a budget that our group has provided to the prototype. It is the triangulation of someone firing back at the turret or near it and the turret reacts accordingly. As always if our turret prototype was sponsored or had a greater budget to work with we could have possibly incorporated all of the original features that we proposed.

6.2 Future Improvements

There are many future Improvements that we could incorporate into our prototype design given a higher budget to work with and more time to do so. The main concept behind including more features or improvements into a design is the fact that when there is more of a budget to work with you or able to incorporate better components and more advanced technology. As it is stated in this design document we are working on our own budget and thus are incorporating components that efficiently and precisely construct the turret to operate in the way we see fit. Also, we are incorporating into our design the technology that was available at the time of this design document being written and therefore there will always be improvements available since technology is

always evolving ever rapidly. On the topic of future improvements, in reality we would also like to include any feature that we left out on our final prototype design to be included in the next prototype design.

The future improvements that we would like to incorporate into the design after our first prototype is the ability for the turret to operate in a 360^{o} range of motion. This can be done with the alteration of the servos to operate in a range of 360^{o} and a wireless safety shutoff switch to shut down the turret. This feature would also have to work along with the distinguishing of friend or foe. The two other features that should be incorporated into the design of the prototype after the one we design should be the ability for the turret to distinguish between friend or foe and the ability of the turret to operate using night vision. Another small feature that would be nice on a future prototype is an automatic paintball/airsoft gun that way our group would not have to pay for an extra servo, it would put less strain on our microcontroller and in general the system. We would also like to take our efficiency, accuracy, and precision data from our system and make the next prototype better.

The overall general concept that we want to improve on in our second prototype is the overall speed and responsiveness of our turret system. Knowing this fact and the other fact that there will be more technologically advanced components by the time our prototype is built. Our group would equip our prototype with a possibly different design and faster technology. First we would require a faster processing microcontroller, which would increase the processing speed of the inputs and outputs that are connected to the microcontroller. This would have a huge impact on the turret system design since the microcontroller controls most of the components of our system. If our group had more of a budget we would also buy a more advanced computer/netbook which would also increase the processing speed of other components in our system. Improving on these two major areas would increase the overall speed and responsiveness of the second prototype. Another hardware component that could be changed but would possibly be overkill for the system is the type of servos that are used with more torque and faster angle change per second. There are also software and networking improvements that could be incorporated into the second prototype and they are mentioned below.

6.3 Setup and Operation Manual

6.3.1 Turret Setup

1. Place the base system on flat solid ground such as concrete, or in a grass/dirt area where stakes can be used.
2. If the base is placed on soft ground, stakes can be used in the 2 inch diameter holes on the left and right of the large square bottom to secure the base in place

3. Make sure that the gun is secured in the U bolts and the trigger servo is in the correct position. The gun should not move in the U bolts, and the arm of the servo motor should be near the trigger but not pulling it.
4. Make sure that the printed circuit board is properly secured in the raised mount at the back of the base.
5. Plug each of the servo motors into the respective servo pin headers as follows
    a. Servo 1: Pan motor
    b. Servo 2: Tilt motor
    c. Servo 3: Trigger motor
6. Make sure the USB to serial cable is plugged securely into the PCB, and plug it into the computer. On the computer, make sure that the COMM port is set to COMM8 using Device Manager.
7. Connect the battery to the PCB using the 5.5mm jack.
8. Connect the web camera to the PC via USB
9. Connect the range finder to the computer via USB
10. Open the ATPT executable
11. Start Django webserver
12. Start data collector server (Java process)
13. Select camera in ATPT, hit start button

6.3.2 Turret Operation

Onboard turret Control:

1. Make sure all the components are connected correctly as in the setup above.
2. To use automated control, take the following steps:
    a. Make sure the Automated box is selected
    b. In the ATPT software, select the webcam from the center drop down list.
    c. Select start at the top to start automated control
3. For on board manual control testing, take the following steps:
    a. Select the Manual box
    b. Use the Up, Down, Left, Right, FIRE, Alarm buttons to test and control the turret.
4. Below in Figure # is a picture of the ATPT application for reference

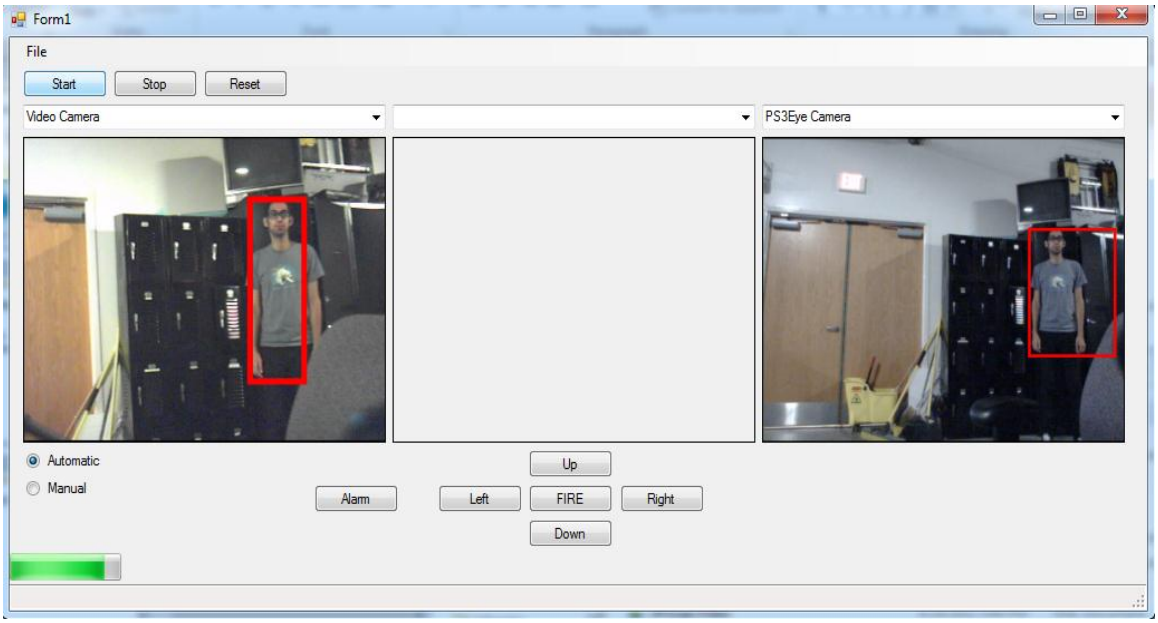

Figure 75: Reference screenshot of ATPT software

Offboard Turret Control:

1. Load web applet and load Welcome Screen.
    a. A grid with all the turrets along with a screenshot of their last engagement will be listed
2. Select Turret to manually control.
3. Manually control turret
    a. Turret movement can be controlled by the arrow keys or the 'WASD' keys
    b. Space bar fires the turret
    c. Pressing shift sounds the alarm
    d. The radar and turret information can be seen on the right of the page.
4. Check engagements by visiting the engagement page
    a. Engagements will be listed along with a screenshot of the engagement
5. The Radar Watch page will show the radar of each turret at the same time

7 Appendix

7.1 Works Cited

1. *The Sentry Project*. Web. 03 Dec. 2010. <http://www.paintballsentry.com/>.
2. "Home." *G8 SENTRY GUN*. Web. 05 Dec. 2010. <http://eecs.ucf.edu/seniordesign/fa2008sp2009/g08/index.htm>.
3. ""Paintball Targeting System." *Department of EECS, UCF*. Web. 03 Dec. 2010. <http://eecs.ucf.edu/seniordesign/fa2007sp2008/g11/>.
4. "Spring 2007 - Senior Design - Group 4." *Department of EECS, UCF*. Web. 03 Dec. 2010. <http://eecs.ucf.edu/seniordesign/sp2007su2007/g04/>.
5. ServoCity. "How Do Servos Work?" *Servocity*. Web. 03 Dec. 2010. <http://www.servocity.com/html/how_do_servos_work_.html>.
6. "PyroElectro.com » Servo Motor Control - Introduction." *PyroElectro News, Projects & Tutorials*. 2010. Web. 03 Dec. 2010. <http://www.pyroelectro.com/tutorials/servo_motor/index.html>.
7. "Torque Calculation." *Physics Help and Math Help - Physics Forums*. Web. 03 Dec. 2010. <http://www.physicsforums.com/showthread.php?t=284466>.
8. "HS-485HB Servo." *Servocity*. Web. 03 Dec. 2010. <http://www.servocity.com/html/hs-485hb_servo.html>.
9. "HS-5745MG Digital 1/4 Scale." *Servocity*. Web. 03 Dec. 2010. <http://www.servocity.com/html/hs-5745mg_digital_1_4_scale.html>.
10. "Laserbotics." *Porcupine Electronics*. Web. 03 Dec. 2010. <http://porcupineelectronics.com/Laserbotics.html>.
11. "Technical Specifications: MW-09-530-S (MW09530S MW09530 MW09 MW 09 530 S) - Floyd Bell, Inc." *Piezo Audible Alarms & Buzzers - Floyd Bell, Inc.* Web. 03 Dec. 2010. <http://www.floydbell.com/products/specifications/MW-09-530-S>.
12. Amazon By. "Amazon.com: Logitech Webcam Pro 9000 / QuickCam Pro 9000: Electronics." Web. 03 Dec. 2010. <http://www.amazon.com/Logitech-Webcam-9000-Built--Microphone/dp/B002M78ECK/ref=sr_1_3?s=electronics&ie=UTF8&qid=1290651822&sr=1-3>.
13. Amazon. "Amazon.com: USB 6 LED PC Webcam Camera plus Night Vision MSN, ICQ, AIM, Skype, Net Meeting and Compatible with Win 98 / 2000 / NT / Me / XP / Vista: Electronics." Web. 03 Dec. 2010. <http://www.amazon.com/Webcam-Camera-Vision-Meeting-compatible/dp/B0015TJNEY/ref=sr_1_5?s=electronics&ie=UTF8&qid=1290651822&sr=1-5>.
14. *Www.dd-wrt.com | Unleash Your Router*. Web. 03 Dec. 2010. <http://www.dd-wrt.com/site/index>.

15. "Arduino - ArduinoBoardUno." *Arduino - HomePage*. Web. 03 Dec. 2010. <http://arduino.cc/en/Main/ArduinoBoardUno>.
16. *Quickturn PCB Manufacturer, Advanced Circuits Printed Circuit Boards Production, Annular Ring Boards, Circuit Board Fabrication, PCB's Fabricator and Distributor*. Web. 03 Dec. 2010. <http://www.4pcb.com/>.
17. *Arduino - Servo*. Web. 03 Dec. 2010. <http://arduino.cc/en/Reference/Servo>.
18. "Arduino - Serial." *Arduino - HomePage*. Web. 03 Dec. 2010. <http://arduino.cc/en/Reference/Serial>.
19. "Arduino - Bootloader." *Arduino - HomePage*. Web. 03 Dec. 2010. <http://www.arduino.cc/en/Hacking/Bootloader>. "Arduino - Servo."
20. "The C# Language." *MSDN | Microsoft Development, Subscriptions, Resources, and More*. Web. 05 Dec. 2010. <http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx>.
21. "Microsoft Visual Studio 2010." *Microsoft Visual Studio 2010*. Web. 05 Dec. 2010. <http://www.microsoft.com/visualstudio/en-us/>.
22. *Java.com: Java and You*. Web. 05 Dec. 2010. <http://www.java.com/en/>.
23. *MySQL :: The World's Most Popular Open Source Database*. Web. 05 Dec. 2010. <http://www.mysql.com/>.
24. *Python Programming Language – Official Website*. Web. 05 Dec. 2010. <http://www.python.org/>.
25. *Django | The Web Framework for Perfectionists with Deadlines*. Web. 05 Dec. 2010. <http://www.djangoproject.com/>.
26. *Eclipse - The Eclipse Foundation Open Source Community Website.* Web. 05 Dec. 2010. <http://www.eclipse.org/>.
27. "Application Programming | Adobe Flash Platform." *Adobe*. Web. 05 Dec. 2010. <http://www.adobe.com/flashplatform/>.
28. *TortoiseSVN*. Web. 05 Dec. 2010. <http://tortoisesvn.net/>.
29. *WinMerge*. Web. 05 Dec. 2010. <http://winmerge.org/>.
30. *ExpressPCB - Free PCB Layout Software - Low Cost Circuit Boards - Top Quality PCB Manufacturing*. Web. 03 Dec. 2010. <http://expresspcb.com/>.
31. *CadSoft Online: EAGLE Layout Editor*. Web. 03 Dec. 2010. <http://www.cadsoft.de/>.

7.2 Copyright Permissions

7.2.1 Advanced Circuits

Nicholas Yielding <nyielding88@gmail.com>
To: Layouthelp@4pcb.com

Dear Sir or Ma'am,

I am with an electrical engineering senior design group at the University of Central Florida that is interested in purchasing a PCB from you in the near future, as well as using your PCB Artist Layout Software. I am requesting permission to use images of the PCB Artist Layout Software in our design documentation, with proper citations to your copyright of course. I look forward to your reply, thank you.

Sincerely,

Nicholas Yielding

Jim Hellmer <Jim@4pcb.com>
To: Nicholas Yielding <nyielding88@gmail.com>, layoutHelp <layoutHelp@4pcb.com>

Nicholas,

We appreciate the courtesy of your request, thank you. As long as there will be no commercial use of these images you have our approval to use them for your educational purposes.

Please let us know if we can be of any other assistance.

Jim Hellmer
Vice President - Engineering
Advanced Circuits
(800) 979-4722 ext 1338

Confidentiality Notice: This e-mail and any attachments are intended only for the use of those to whom it is addressed and may contain information that is confidential and prohibited from further disclosure under law. If you have received this e-mail in error, its' review, use, retention and/or distribution is

strictly prohibited. If you are not the intended recipient, please contact the sender by reply e-mail and destroy all copies of the original message and any attachments. Further, this e-mail transmission may contain information controlled for export purposes under the U.S. International Traffic in Arms Regulations (ITAR); if this applies, no export, sale, transfer, release or other disposition of this information is permitted without prior authorization from the U.S. Government.

7.2.2 Porcupine Electronics

Nicholas Yielding <nyielding88@gmail.com>
To: info@porcupineelectronics.com

Dear Sir or Ma'am,

I am with an electrical engineering senior design group at the University of Central Florida that is interested in buying your Fluke rangefinder control board in the near future. I am inquiring if it would be okay for us to use the images and information on your site in our design documentation, with proper citations to your site of course. I look forward to hearing your reply, thank you.

Sincerely,

Nicholas Yielding

---

info@porcupineelectronics.com <info@porcupineelectronics.com>
To: Nicholas Yielding <nyielding88@gmail.com>

Nicholas,
No problem, feel free to use the images from our web site and LR3 documentation for your project.
Thanks,
Richard

7.2.3 Atmel Corporation

Nicholas Yielding <nyielding88@gmail.com>
To: webmaster@atmel.com

Dear Sir or Ma'am,

I am with an electrical engineering senior design group at the University of Central Florida that is interested using the ATmega328 in our design. I am requesting permission to use sections of the ATmega328 data sheet [1](mainly the pin layout diagram) in our design documentation, with proper citations to your copyright of course. I look forward to your reply, thank you.

Sincerely,

Nicholas Yielding

[1]: http://www.atmel.com/dyn/resources/prod_documents/8271S.pdf

---

De Caro, Michael <Michael.DeCaro@atmel.com>
To: Nicholas Yielding <nyielding88@gmail.com>

Nicholas,

You may use sections of our datasheet with the proper citations as you described.

Regards,
Michael