# Senior Design II - Summer 2012

Reconnaissance and Demolition Super Attack Tank

Group 7 -

Jeff Hildebrandt

Dylan Lambe

Mick Muzac

Bradley Raley

6 August, 2012

# Table of Contents

# 1.0 Executive Summary

The Reconnaissance and Demolition Super Attack Tank, also known as RADSAT, is a small, autonomous, robotic tank that has hardware and software capabilities allowing it to locate a target, track it, and accurately fire plastic BB's at the target. The tank supports two separate modes of operation, the ability to be manually controlled via a laptop as well as the ability to be set to autonomous mode. These functions are supported by video imaging as well as close object detection via infrared and ultrasonic sensors for obstacle avoidance. RADSAT itself contains a voltage source to power all of its components, a microcontroller to execute all of the commands, and a wireless chip that can receive commands from the laptop via WiFi. While in autonomous mode, the tanks movement decisions and route planning algorithm are dictated by the sensors and video image processing.  These computations are processed and transmitted in real-time and allow RADSAT to successfully locate a target. For complete control, the user is provided with a convenient GUI, as well as software that can accept voice commands via an Android phone.

The primary motivation for the design and implementation of RADSAT came from the group's desire to develop a robot.  Mainly focusing on the software development side, most of the research and design went into the functions that were to carry out RADSAT's mission of autonomously locating a target. Additionally, the software design that went into RADSAT can be adapted into a real world design for military applications, as autonomous vehicles keep human lives out of danger.  An artist's rendering of RADSAT can be seen in figure 1.0.1 below.

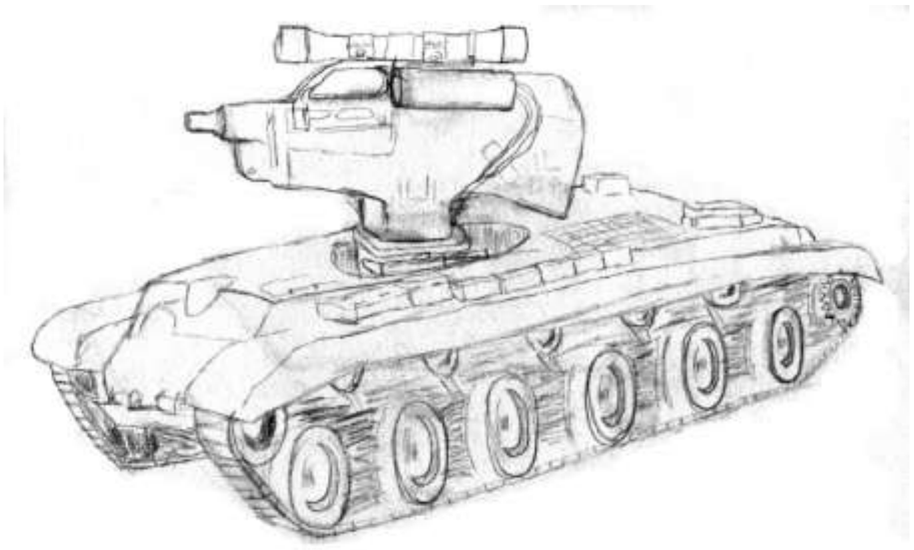


*Figure 1.0.1: Drawing of the complete tank.*

# 2.0 Project Description

## 2.1 Project Motivation and Goals

The motivation behind the RADSAT was a combination of wanting to make something fun as well as something that would look good on a resume. Because the group was founded upon the idea of making a voice controlled robot, the group was excited to combine ideas and to make an interesting and exciting robot. The idea was the RADSAT. The RADSAT is a mostly autonomous, WiFi controlled, sensor equipped tank that will shoot air-soft bullets at colors specified through voice control, as well as return a video feed. The motivation and goals for the individual features of our project are as follows:

Autonomous – the reason an autonomous robot was chosen is mainly for the challenge behind it. An autonomous robot is far more interesting than a manually controlled robot, as it requires not only lot more software design, but also the addition of sensors. Becoming autonomous will give our project a greater depth and will give the group ample experience with software and hardware integration. Even though becoming autonomous will be a greater challenge, the reward of experience will be more than enough to make up for it.

WiFi Controlled – WiFi seemed like the obvious choice for the project. WiFi has a fairly far range and can easily be extended with repeaters. This will allow the RADSAT to roam freely, with only a small chance of losing a signal. WiFi can also travel through walls, so even if the RADSAT leaves the room it will not lose signal. The laptops being used for the project are also pre-equipped to communicate with WiFi, so extra parts will not have to be bought. In addition, microcontrollers can easily be equipped with parts that will make them WiFi compatible, and several exist which already have WiFi chips built into their system. This will allow us to easily communicate signals back and forth from the tank to the laptop, at very high speeds. The most important decision for making the RADSAT WiFi, is because there aren't any existing projects involving sending a video feed through either Bluetooth or radio waves that had anywhere near as good of results as WiFi.

Sensors – the initial decision for the RADSAT was to not equip the robot with sensors, and have all obstacle detection be solely within the video processing software. However, through research, it was discovered that the RADSAT will be far more efficient if sensors are used in parallel with the video processing software. The sensors will be able to preemptively detect when the robot is going to hit a wall or object, thus creating a smarter robot that will be able to find its way through any sort of obstacle.

RC Tank – the initial decision was to use a regular RC car as a base, but when it was decided to make the robot shoot at targets, the decision was changed and an RC tank was decided upon. One reason was because the RC tank comes pre-equipped with an air-soft rifle and rotating turret. Although, the rifle and turret are most likely going to be modified, that route is far easier than creating everything from scratch. The RC tank was also chosen because the motion controls are simpler than going with an RC car. The tank only has two controls: left track move and right track move. This will make programming for obstacle avoidance much simpler, because we will not have to concern ourselves with the hassle of turning a wheel. The tank's base will also be large enough to mount a camera, as well as whatever power (battery, breadboard) and controllers that will be needed.

Color Specified Targets – Adding color specified targets added an extra layer of difficulty to the RADSAT project, and after successfully being able to autonomously track certain colors using a coding algorithm, the groups' programming ability will greatly increase. Colors were chosen over shapes or anything else, because color recognition seemed more reliable than shapes, and movement would be extremely difficult because the robots movement would have to differentiate between the RADSAT's movements between the target's movements.

Voice Control – a voice controlled robot was the basis for the group. This was the very first feature the group thought the RADSAT should include, so of course we had to include it within the project. Nowadays, it seems with phones and computers everything is able to be controlled with voice. Voice control technology is becoming a very relevant technology, so it was figured having voice control experience would be beneficial. Having voice control will also allow the robot to still be autonomous while at the same time allow the user to control it.

Returning a Video Feed – the motivation behind returning a video feed is to have a robot which can operate effectively even when not in direct line of site. Even if the robot leaves the site of the user, the user can still easily see where the robot is and what it is seeing. The video feed is also the most important thing for the RADSAT to function correctly. The video feed will be used for almost all tasks, such as obstacle avoidance, color recognition, and targeting. Without the video feed the RADSAT would have to be completely redesigned.

So, the idea for a robot was first conceived through a joint group mentality to design and build a robot. Although, the idea started from the idea of simply making a voice controlled robot, several ideas grew from there. From watching a YouTube video featuring a WiFi controlled robot, the group decided to expand to different interfaces. Also, through brainstorming and research it was decided to add a video feed and color recognition. Then finally, with a discussion to make the RADSAT more challenging, it was decided to make it fully autonomous. The RADSAT started as a novel idea at first, but has evolved into something that will truly become great.

## 2.2 Objectives

The RADSAT robot is going to have some objectives that it will have to achieve. Each component of the RADSAT robot is going to try to achieve a certain number of standards that will be put forth. While thinking of objectives it is very important to think of the group's experience levels as well as the amount of extra money the group is going to have on hand. The following are the objectives of each different section of the RADSAT:

WiFi Controlled – The objective here is to have a communication from a laptop to the robot via a WiFi connection. The computer will be able to send signals to the robot which controls various functions. For instance, if the laptop sends a signal of 10, then the RADSAT will know to move forward. The signal send and receive rate should be almost instantaneous, enough so, that the user doesn't notice any lag between the signal being sent, and the signal being received.

Return a Video Feed – The video feed which is returned should have a fairly decent picture, without much lag. Preferably no lag, however that may be very hard to achieve because of the amount of editing the video feed will have to go through, prior to being displayed on the GUI.

User Friendly GUI – The GUI should be very intuitive, with only a few buttons and only a few changeable settings. The GUI should intuitively be able to be used by anyone who wishes to control the robot. Also, the GUI should have a professional look and feel, as though it could be marketed.

Voice Controlled – The voice control feature will probably be one of the hardest things that is going to be implemented for the robot. This feature needs to be able to distinguish several different commands that will be input by the user. For example, when a user says the word "red", the program must recognize that the word was a color, and know that the target color is now red. The following words will hopefully be able to be recognized by voice command: fire, red, green, blue, stop, lock.

Sensors – The sensors should be able to sense all objects in its way, including but not limited to smaller objects, such as chair legs. Three sensors planned to become implemented onto the RADSAT; two infrared sensors, as well as one sonar sensor. The infrared sensors should be able to detect when the RADSAT becomes close to any walls, and the sonar sensor will be able to sense any objects within the direct path of the RADSAT. Whenever the sensors sense something, the RADSAT should be able to react accordingly. For example, if the RADSAT is about to run into a wall, the sensors should be able to recognize the event about to occur and correct its path.

Autonomous – The objective here is to make the RADSAT mostly autonomous, and react to only to a few simple commands. When a certain color is given, the

RADSAT should be able to search, and find the color, then target the color with its airsoft rifle. The sensor implementation will allow for free autonomous movement without the fault of running into objects.

Color Recognition – The RADSAT should be able to distinguish at least three different colors. When a certain color is specified, the GUI will display to the user the objects which are being recognized. The whole screen should turn grey, except for the color which was previously specified. By doing this the user will be able to fully recognize exactly what the RADSAT is seeing and the available targets to shoot.

Shoot at Targets – The targets in question will be the color that the user specifies. The RADSAT should be fairly accurate with the chosen targets. In that, the RADSAT will find the exact center of the chosen target, and because the rifle will be powerful enough to shoot in a straight line, when the RADSAT shoots it should hit in the direct center of the target. On the GUI display, a crosshair will be displayed on the current area that the RADSAT is targeted at. This way the user doesn't shoot at an unwanted target.

Power Consumption – The RADSAT will be powered by a single rechargeable battery. The RADSAT should remain in operation for at least fifteen minutes off of a single charge. However, since so many things will be added to the RC tank to make the RADSAT operational, having extremely high hopes for low power consumption didn't seem like a logical choice. There will be several high power devices on the RADSAT, such as a camera, several sensors and a microcontroller.

Relatively low cost – with every part that is purchased, the different components will be compared, and the cheapest option will be chosen. For instance, when choosing a microcontroller, we the less expensive, less powerful, fully integrated model was chosen, rather than the more expensive, more expansive model, and through this process saved $30.

Sturdy – The RADSAT will be fairly sturdy. It should be able to survive a several collisions and still be completely intact. Bumpers will be placed on the side of the RADSAT to ensure any collision will be cushioned, and the delicate components will remain untouched.

Responsive - The RADSAT will be very responsive to any commands given. In that, there will be little, to no lag time between different commands, and the RADSAT will do only what it is commanded to do.

## 2.3 Project requirements and Specifications

Once broad objectives and goals have been established for RADSAT, hardware and software requirements and specifications were slowly introduced as research progressed. Outlining requirements and specifications became necessary not only to facilitate research, but to also organize and lay the groundwork for all of RADSAT's future design work.

## 2.3.1 Software Requirements and Specifications

One of the most important requirements inherently embodied in the idea of a reconnaissance vehicle able to recognize vocal commands is that it should have autonomous features, and a number of different algorithms to control it. Figure 2.3.1 summarizes various major software components of the project and the different inputs and outputs each will be able to interface with. One aspect that is left open in the figure is the manifestation of each depicted block. Certain blocks may certainly be consolidated or otherwise simplified, and almost all blocks are likely to be divisible.

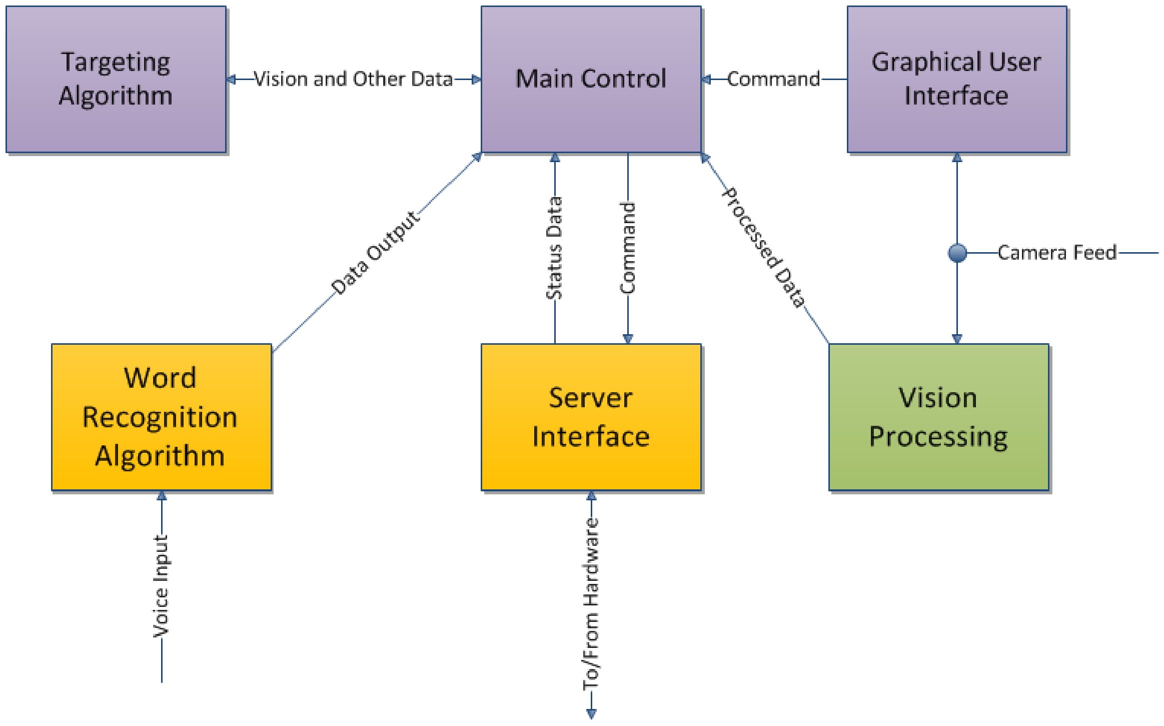


*Figure 2.3.1: Software block diagram displaying general software requirements (colors denote person administratively responsible).*

The main control block is considered the brains of the software portion of the entire system. It is the only block that must have some method of communicating with every other block. Its main requirement is to process several pieces of information received, make an informed decision based on several different factors, and to send a resulting command to RADSAT. Its decision and checking process also includes verifying that it does not attempt to send contradicting commands to RADSAT. Such commands could have arbitrary results if RADSAT is not explicitly programmed to disregard contradictions. The word recognition algorithm's job is to process vocal input into a format understandable by the control block. The GUI's functionality must include displaying a video stream to the user and accepting direct user commands. Although vision processing and targeting go hand in hand, the two blocks have very different functionality. Vision processing only has to accept a camera feed, and convert that into something more usable. The targeting algorithm must be able to process that information further and determine if the target is present. If the target is present, it must determine where in the image it's located and the correct angle to set the turret. The server interface, simply provides an interface to facilitate communication between RADSAT and the controlling computer. Detailed software specifications for RADSAT are concisely represented in Table 2.3.1.

| Spec. # | Specification |
|---|---|
| 1 | The ability to recognize a set of at least 10 spoken words and to successfully translate them into a usable format. |
| 2 | Must be able to accept a verbal command whose length is a maximum of 5 seconds. |
| 3 | Verbal commands must be translated within 5 seconds from the end of the command. |
| 4 | Must recognize the differences between a minimum of 5 different colors. |
| 5 | Must have the ability to view a live video stream transmitted from RADSAT at a frame rate of at least 20 frames per second. |
| 6 | The targeting algorithm must identify a 12x12in target from a minimum distance of 15 feet. |
| 7 | All non-vocal commands must be processed and sent to RADSAT within 500ms of issuance. |

*Table 2.3.1: Software specifications for RADSAT.*

### 2.3.2 Hardware Specifications and Requirements

For reference as this project is developed and designed, and for guidance as to the end goal of this project, the following requirement list and specification table have been devised. They are meant to be comprehensive, yet concise. Ideally, the following statements are realistic and achievable within the bounds of our budget and time resources. As this project progresses, the requirements may have to change.

Requirements:

-The vehicle shall have one battery to drive all systems; two batteries maximum.

-The vehicle shall be able to quickly move in a straight line and turn while moving or while still.
-The vehicle shall have sufficient sensors to avoid real-world obstacles.
-The vehicle shall be equipped with a camera able to relay video information.
-The vehicle shall be able to aim and fire an airsoft gun.
-The airsoft gun shall shoot rapidly and accurately.
-The vehicle shall be appropriately sized.
-The vehicle shall be able to operate in various types of terrain.

Specifications:

| Item | Spec1 | Spec2 | Spec3 |
|---|---|---|---|
| Battery | >7 Volts | >2000mAh | >30 min. life |
| Chassis | >20in. Length | >10in. Width | <30 lbs |
| TurretServo1 | >720° range | Standard size | >3 kg/cm torque |
| TurretServo2 | 90° range | Standard size | >4.6 kg/cm |
| Drive system | ≈5 mi/h | >15° climbing | <2A draw |
| Airsoft Gun | ≈150 fps | ≈200 BB mag. | 100 rounds/min. |

*Table 2.3.2.1: Specifications for hardware. By chance, each item has three specifications.*

It should be noted that the relationship between the battery life, the battery capacity and the total amp draw on the battery is denoted as ____mAh * 1A/1000mA * 60 min./1h * 1/___A = ___minutes. To make sure the specifications involved in this calculation are realistic it will be tested with hypothetical values: Selecting a battery with a 2,000mAh capacity, and assuming a 3A average total draw on the battery from all systems, and assuming no energy is lost to heat and inefficiency, it should be able to operate for 40 minutes. This number is above the minimum of 30 minutes. If the anticipated parameters are correct, battery life will not be an issue.

The other specifications rely mostly on the parameters of the parts bought and steps will be taken to ensure they are met, at the time of selecting the part. For example, in the section discussing servos the torque exerted on the servo is calculated and a servo is selected to meet the torque specification. Other specifications are not readily derivable now, such as the speed the tank will travel, but will be easily attainable after prototyping is under way. At that time, steps can be taken to alter or correct for areas that are not on par with specifications. Similarly for requirements, if during prototyping a facet of a component falls short of its requirement, it will be remedied using data collected pertaining to that particular shortcoming.

# 3.0 Research

## 3.1: Existing Similar Devices

Looking at similar vehicles one might notice that most of them fall into one of two categories: pre-made toys and home-made hobbyist projects. The majority of the toy radio controlled airsoft shooting tanks are modeled after real tanks such as the Panther or M1. A typical example of what is on the market can be seen below:

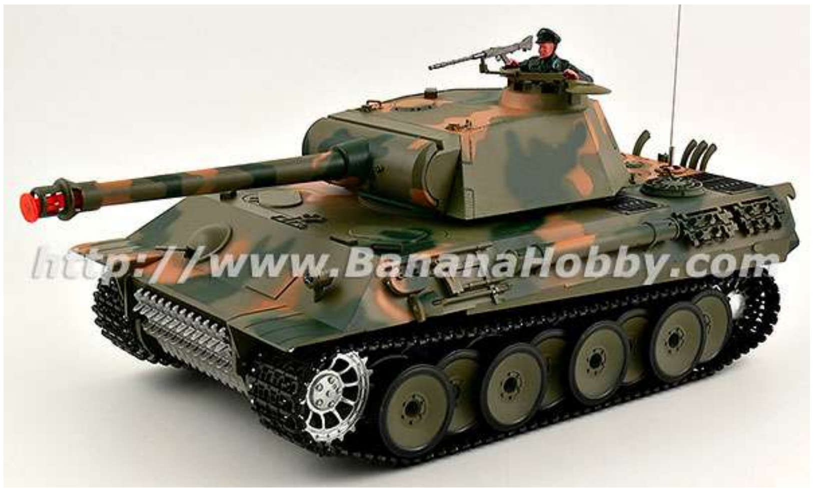


*Figure 3.1.1: A radio controlled 1:16 scale German Tiger I Panzer. With permission from Bananahobby.com*

These tanks usually cost around one-hundred to one-hundred fifty dollars. Smaller, 1:24 scale tanks are lower in price. These tanks typically have rotating turrets and can shoot arisoft BB's. Many of them have smoke and sound features for a more realistic effect. They can drive over rough terrain and some can even drive through shallow puddles. According to a handful of customer reviews, these tanks are sometimes lacking in quality, breaking after only a few uses.

The other class of tanks are those built for fun. These are usually completely custom and for the most part every one of them is different. People sometimes buy treaded bases, make their own, or buy one of the above mentioned tanks and strip it down to use the parts. Anyhow, this approach is very interesting for this project because it will be most similar to them. An example of a wifi camera equipped tank is shown below.

This particular robot has four important things in common with the project at hand: It has a wireless camera to transmit video, it has the ability to pan and tilt the camera, it has wireless communication between the PC and the robot, and it is a treaded vehicle. The write-up about it is not exactly a wealth of information, but it does say that the camera cannot be on the same frequency as the Xbee wireless module. In this case a 2.4 GHz Xbee and a 900 MHz wireless camera were used. The main differences are that RADSAT will have an airsoft gun and a PCB instead of an ardiuno board.

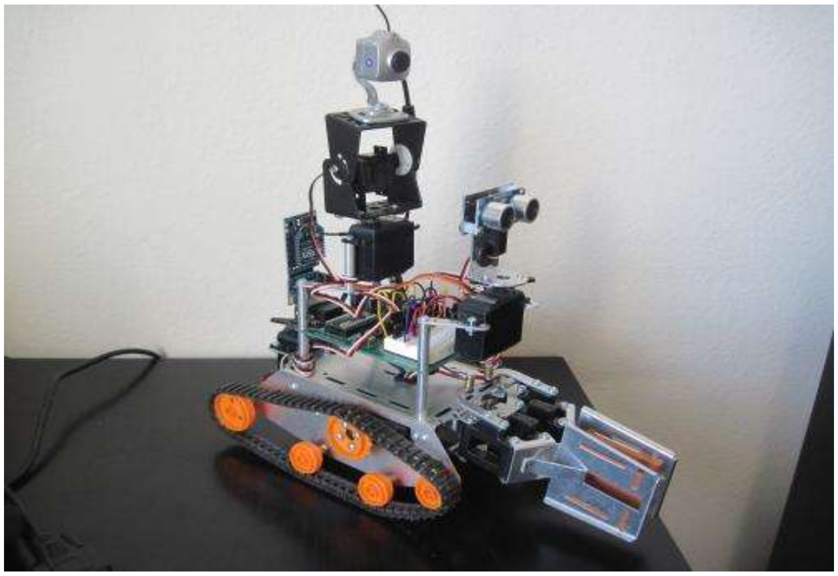


*Figure 3.1.2: Boe-bot robot with Xbee.*
*With permission from JeffBr from letsmakerobots.com*

Another project worth mentioning is a senior design project found on the same website. It is called RES-Q-ME and is designed for rescuing lost or trapped people in disaster situations. It is a treaded tank that interfaces to a computer wirelessly and has a camera. This robot has a camera and infrared navigation and can be controlled by an iPod touch. It uses an Arduino Mini Pro as its processor. The robot can be seen in the figure below:

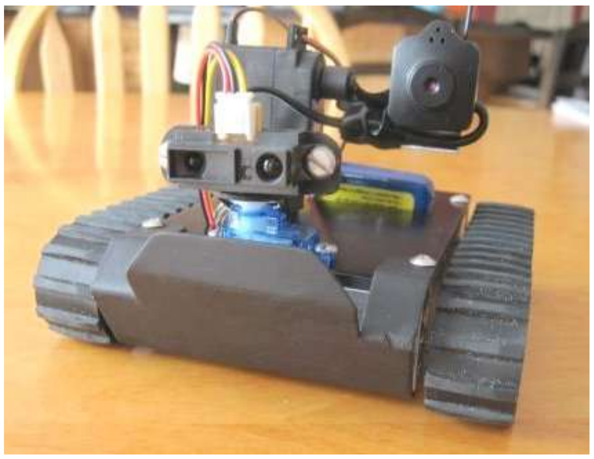


*Figure 3.1.3: The RES-Q-ME with permission from Jacob Frieda*

It shares similarities with RADSAT in that it is a senior design project that focuses on a tank which has a camera and drives autonomously while communicating with a computer via wireless. It uses an arduino and servo drivers for the motors instead of an fully custom PCB. RADSAT will differ in that it will have a PCB with the microcontroller, wireless shield and power distribution and control components all on it with no superfluous components. It will also have automatic camera tracking systems and a gun to fire at targets.

## 3.2 Relevant Technologies

With the goals and objectives of this project decided upon, it is necessary to spend some time researching other models and projects that implement similar technologies to this project.  Although there does not seem to be a design that has implemented all of the technologies for the specific mission of RADSAT, there does exist a plethora of projects that implement one or more of the technologies that are to be implemented by RADSAT.  The research done on these projects will be extremely helpful to the team when it comes down to the development and implementation of RADSAT.

The technologies that will be focused on during this research include those that will have to be programmed by the team in order for RADSAT to carry out its objectives.  This includes video processing, color recognition, an autonomous search function, and voice command recognition, as well as controlling the camera and turret on the tank.  Furthermore, it will be necessary to research other projects that have implemented WiFi communication between a wireless chip and a laptop.

One similar project that was of interest to the team was a tennis ball collecting robot named Autonomous Ball Collector, or A.B.C., designed by a UCF Senior Design group from Summer 2010.  This project sparked the team’s interest because of the similar technologies that the A.B.C. robot implemented that are planned to be implemented by RADSAT.  The A.B.C. robot was designed to search out tennis balls autonomously, scoop them up, and return to a home base.  It did this using a color recognition system similar to the one that is planned to be implemented in by RADSAT.  This goal was accomplished by the A.B.C. robot by using a Blackfin camera that was capable of recognizing colors and shapes.  Once the image of a tennis balled was discovered, the A.B.C. robot used video processing to determine the location of the ball so that it could determine which way to move.  The A.B.C. robot also had the capability to communicate with a laptop via an XBee adapter.  This allowed for remote control of the A.B.C. robot by a user up to 100 feet away.  The similar desires for RADSAT to be able to communicate with a laptop might make the XBee technology something that the team can further research and implement in the final design.

Another project that was looked at by the team was a WiFi robot developed by a hobbyist electrical engineer as a “spy” robot. This robot was controlled by a laptop and was capable of streaming live video via a webcam back to the computer. Movement was also controlled remotely by the user, and the robot had a range of 500 meters from the user.

This project has a lot of relevance to RADSAT and could serve as a useful resource tool. The video streaming and WiFi control are both technologies that the team hopes to implement with RADSAT. The method that the engineer used to control the spy robot was a router mounted onto the robot. This allowed the user to control the robot over the Internet using a wireless access point or with a WiFi enabled laptop.

While researching voice command robots, the team came across a simple project that had a detailed tutorial on how to build a voice controlled robot for roughly $224. This robot, powered by an Axon II microcontroller, responded to the commands “left”, “right”, and “forward”, as well as a few other commands. Being similar to the commands that are to be used in the implementation of RADSAT, this project should prove as a useful resource for the team during the development stage. A neat and interesting aspect of this robot, being programmed by a Thai engineer, was its ability to also switch its voice recognition software over to the Thai language and respond to the same commands.

Upon researching the voice command robot above, the team happened to stumble upon a website that should be an extremely helpful source of information in the development of RADSAT. The site, www.societyofrobots.com, is loaded with resources for beginners who are interested in developing their own robot projects. As this is the first time any team member of RADSAT has ever attempted to develop a robot, this website will be extremely helpful as a resource tool. The tutorials, forums, user support, and frequently asked questions section can assist with the debugging of any issue that the team might come across during the development of RADSAT.

Finally, the team looked at another senior design project from a class back in 2001at an unspecified school that designed what was called the Autonomous Search Robot. This robot used an autonomous algorithm to search an entire room in an attempt to locate and recover a predefined object. Once the object was found, the robot would use a robotic arm to scoop up the object and place it inside its body.

This autonomous search robot has some unique technologies that are relevant to RADSAT and could possibly be further researched and developed by the team later on. First, the search algorithm that the robot implemented stored knowledge of everything it processed on its webcam, not just the object it was searching for. The robot was able to distinguish between its objective, other objects, and walls within the room, and map them relative to one another by their distance apart. Once the robot had identified all of the walls and searched the entire enclosed space, it was able to deem the mission complete. The idea of storing the processed images as a map inside the robots “brain” is an interesting concept that may be worth looking into as the search algorithm implemented by RADSAT.

Another interesting technology with the ASR was its ability to pick up its objective using a robotic arm. In order to accomplish this, the robot needed to know the exact location of the objective and its position relative to the robot in order to accurately aim the arm. This was accomplished by using an infrared sensor that was able to precisely determine the distance of the object. As RADSAT should be able to fire accurately at its target, knowing the distance of the target from RADSAT will be of much use.

With the research that has been completed on these technologies, the team has a much broader knowledge base that will be used to develop RADSAT. The projects that were the scope of the research each had unique qualities that may further assist the team during the design stages. Though the technology implemented by RADSAT will be unique and developed by the team for RADSAT’s specific purposes, these technologies can be used as a basis for what has been developed and implemented by other engineers in past projects and designs.

## 3.3 GUI Research

As the tank will be controlled using a laptop, a graphical user interface will need to be researched for the project. This GUI will need to include all of the commands that the tank is expected to respond to and carry out. These commands will be transmitted via a wireless connection to the microcontroller on the tank.

The commands and operations that the tank should respond to for this project are listed in the *Table 3.3.1* below:

| **Command** | **Description** |
|---|---|
| Connect | Creates a connection between the tank and laptop |
| Tank: Forward, backward, left, right | Moves the tank in the specified direction |
| Turret: Up, down, left, right | Rotates the turret and camera in the specified direction |
| Auto-Search | Sets the tank to autonomously seek out a target |
| Fire | Fires a shot at the target |
| Exit | Breaks the connection between the tank and laptop |

*Table 3.3.1: GUI Commands*

Given that the tank is within the specified range of the laptop, the tank should be able to respond to these commands immediately.  This will be accomplished by transmitting specific signals to the tank every time a command is used.  Once the signal is received, the tank should respond and act according to the above descriptions.

As well as the commands that the GUI must implement, it must also be able to stream live video from the tank.  This video should be captured by the webcam mounted on top of the tank, and should be available in real time to be viewed on the laptop.  This will allow the user to control the tank remotely, as well as monitor the tanks progress when the auto-search function is enabled.

## 3.4 Camera Research

Through research, a way to stream video from the RADSAT to a laptop must be found.  There of course exist several ways to do this, and the goal is to find the method which is best suitable for the project.  There are a few issues that the camera must be able to overcome.  They are: the camera must have a color video feed, it must be able to send the video feed back to a laptop, and it must be able to operate at an appropriate speed.

<u>Idea 1: Mount a router and wired IP security camera directly on the tank.</u> - This idea stemmed from the iAndroBot.net, WiFi Robot project. This project involved hacking a router, plugging a network security camera directly into it, then returning a video feed to a server on the laptop.  This idea was the original intention for the project, and at first glance, it seemed as though it was a good idea.

First of all, the implementation wouldn't be very hard, and the robot would essentially be creating its own wireless network.  Because the robot had its own wireless network, there would be no additional hardware required in order for the laptop to communicate with the robot.  Then, in order to make the router able to communicate with the laptop, new firmware and hardware would have to be added.  Tomato, which is an open source, Linux based firmware for routers, would have to be installed to overwrite the router's existing firmware.  The Tomato firmware would allow the router to be reprogrammed to fit the robot’s needs.  For the hardware addition of the router, a serial port would have to be added, which will have certain signals applied to each pin, and the robot would receive and apply accordingly. The security camera would be very easy to install. The security camera would simply plug into the router, as it would do in a normal situation, and no addition firmware or hardware would have to be added to it. Once everything has been properly installed, the router would place the video feed on a server which would be read by the laptop.

This idea meets the design's target goals; however, there are just a few problems. For one, mounting an entire router on the robot would take up a lot of space, space which could be used for additional power, micro-controllers, or sensors. It would also look very unsightly to have a giant router mounted on top of a robot. Another problem with this design is its lack of real world application. This project can only exist inside of the hobbyist world, as no company would actually hack a router for one of their projects. The company in question would most likely use a cheaper, less troublesome method.

Idea 2: Bluetooth Web-Camera - The Bluetooth web-camera is a relatively new product on the market, and would give the group experience for any Bluetooth development that might want to be done in the future. Bluetooth has several advantages. For example, the Bluetooth web-camera can operate at around 2.1 mega-bytes per second, which would allow for a fairly decent video stream, and connectivity between the camera and laptop would also be very simple. Bluetooth also requires very little power to run effectively, one battery charge can last up to four hours. However, Bluetooth has several downsides as well.

The main deterrent from using a Bluetooth web-camera is: the camera has a max range of only thirty feet. Thirty feet is not an incredibly short distance, and would still be okay for a robot as long as the user was going to always be in the same room as it, but the RADSAT is designed to search, which might result in going to additional rooms. Although range extenders can be applied, the native ability of only thirty feet, and the only other option of purchasing additional hardware is lacking, when compared to other options (i.e. WiFi). Also, a Bluetooth webcam is around three times the price of a comparable WiFi camera, which would be a waste of money for such limited options. So, even though Bluetooth is an option which will succeed for the project, it simply is not the preferred option.

Idea 3: 2.4 GHz Wireless Camera - The second place option for the project, is a 2.4 GHz wireless camera. The 2.4 GHz wireless camera has some very nice options. For instance, the camera has a long range (about 450 feet), is fairly inexpensive, (some) can communicate directly to a USB receiver, and it can efficiently send data which allows for good video resolution.

However, despite all the positives of this set-up, there is one thing that prevents the 2.4 GHz wireless camera from being the optimum choice for this project; the fact that it just hasn’t been done before. Through the research for this project, there wasn’t a single example of someone being able to receive and manipulate a video feed from a 2.4 GHz wireless camera. Since the group is still at the beginning stages of programming, and know nothing about grabbing a USB video feed, trying a concept which there are absolutely no guidelines for, would be an unnecessary challenge. Especially since the video feed is one of the most important aspects of the project. So, even though the 2.4 GHz wireless camera has all the specs we desire, it would just be too difficult and too time consuming to implement.

Idea 4: WiFi Security Camera - After some extensive research, the WiFi security camera became the optimum choice for this project. The WiFi security camera has long range, is inexpensive, streams in a digital format, is wireless, and displays the video on an easy to access local server. The camera's range is limited only by the range of the router it is hooked up to. So, potentially with the use of repeaters, the range can be as far as necessary. The camera also was around half the price of any other type of camera, with just as many, if not more features. Also having the camera stream in a digital format eliminates the hassle of converting an analog signal to a digital one, which means that the computer can easily read it, and it can immediately be altered as necessary. The camera is able to submit wireless signals on its own, without the need of relying on any other devices, which is a huge bonus as video processing is one of the most important aspects of the project. The camera displays the image it is capturing onto a local server, with any IP address of your choosing, this makes receiving the stream quite easy to implement.

*Figure 3.4.1: WansView WiFi Security Camera*
*Permission Pending From Amazon.com*

There a two main options to choose from when deciding on a WiFi security camera. There is the JPEG format, and the MJPEG format. The JPEG format is far simpler than the MJPEG format. The camera simply places a JPEG picture on the server at its designated frames-per-second speed. This allows anyone to easily access the JPEGS by infinitely sending requests for the JPEG on the server. The MJPEG format is much more difficult to implement, because there is a constant stream, which has to be broken apart to be able to work on it. However, the MJPEG format is superior despite the difficulty involved. This is because, instead of an infinite stream of requests to the camera server, there only needs to be one. This results in more processing power left for other functions. So, the WiFi security camera seems like the best choice for the project, as no other camera type has all the necessary features at an affordable price.

| | Router with IP Camera | Bluetooth Camera | 2.4 GHz Camera | WiFi Camera |
|---|---|---|---|---|
| Speed | 11 mbps | 2.1 mbps | Comparable | 11 mpbs |
| Video Location | Server | USB Interface | USB Interface | Server |
| Data Type | Digital | Digital | Analog | Digital |
| Distance | 230 ft | 30 ft | 450 ft | 230 ft |

*Table 3.4.2: Camera Research Summary*

## 3.5 Video Processing Research

The process chosen for the video stream was the MJPEG format. This research is used to find a way to take a video stream (which exists on a server that was created by a WiFi security camera) and convert it into usable code within C#.

As with anything, there are many ways to approach processing of an MJPEG video source. There are several libraries and programs which already exist that allow for editing and streaming of a MJPEG video stream within a program. The goal is to find the simplest and most reliable way to accomplish all that is necessary for the project. The following are the different sources that were discovered to help with this process:

AForge.Video Libraries - This is a collection of libraries done in C# which has libraries for all types of video formats. Out of all the classes within this library, the class which is most relevant for the project is MJPEGStream. The MJPEGStream library allows the input and viewing of a URL address for an MJPEG stream (such as the address for a wireless security camera). The MJPEGStream class has instances for username and password, as well as several different functions for starting and stopping receiving the feed, and can also return a bitmap image. From this research, it seems that this library is the core for all video processing for C#, and almost every MJPEG stream project found, uses these libraries. So, these libraries will most definitely be used within the RADSAT project.

OpenCV Libraries – These libraries exist for C#, and offer everything that might be needed in order to complete an MJPEG stream. It can read in, and display an MJPEG stream and has tons of different examples where this has easily been implemented. These libraries seem very good and may be used for the RADSAT if there is any trouble with getting AForge.Video Libraries working. However, many of the programs which have been proven to work for the WansView WiFi video camera stream were using the AForge.Video Libraries. Therefore, for that reason alone the OpenCV Libraries were not chosen.

Camera Vision - video surveillance on C# - This is an open source video surveillance project developed by Andrew Kirillov. The purpose behind this project was to stream several different IP surveillance cameras, from different sources and different manufactures into one convenient split screen view.

Through the advice given from this source, it was determined that going with an MJPEG stream was the optimum choice for video surveillance streaming. His reasoning was, because an MJPEG stream required only one attempt to access the stream, where with a JPEG stream, infinite attempts to access the stream would have to be implemented.

On his website he gives a general overview on the steps to acquire an MJPEG stream, which are:

1. To discover the type and retrieve the boundary
2. Read in the initial portion of the stream to search for the boundary
3. To continuously read in data until you receive a new boundary
4. Extract the image
5. Process and edit the image
6. Steps 3-5 continue in a loop until commanded to stop

This was a good source for information on how to retrieve an MJPEG stream in C#. However, when running his program and attempting to view the video feed, the video feed did not show up. So, his program will be used as a reference only.

<u>VLC Media Player</u> - The VLC media player is an open source media player that can play virtually any type of video format. It has support for DVD, audio CD, VDC, most multimedia files such as AVI and MKV, as well as various streaming protocols such as MJPEG. It is widely regarded as the best media player that is available today.

While trying various methods and programs to extract the video feed from the WiFi camera server, VLC media player was the first source that was able to successfully view the video feed. This was a huge step which allowed the project to progress, since it was now known that there was some way to successfully extract the video feed, and that the location of the video feed was correct. It was discovered that it was possible to retrieve the video stream, and that the correct address was chosen.

Since VLC media player is open source everyone has access to its extensive libraries. Within these libraries there are several classes which allow for the extraction and manipulation of MJPEG video feeds, as well as various methods to delay and manipulate the streams. Although VLC media player is written in C++ the libraries should still be compatible with C# and will still be used as a good resource.

<u>Java MJPEG</u> - Java is the programming language which most of the group members are most comfortable with. Therefore by looking at the Java MJPEG

project’s code, the process of retrieving an MJPEG video was understood better. It seems the process isn’t too different for both Java and C#.

The Java MJPEG project uses the java.net, java.io, java.awt, and java.imageio libraries. The main class extends java’s existing class “InputStream”. Java’s InputStream works by continuously reading in bytes from some source of data, in this situation it would be for reading in bytes from the MJPEG stream. The Java MJPEG project first establishes the stream, then tries to connect to a specified password protected URL. Following that, it reads in the data from that URL and, if the location is correct, then only the data for the MJPEG stream should be located there. It then reads in every pixel, byte by byte, and returns a completed image.

The process for Java MJPEG is almost identical to the process of retrieving an MJPEG stream with C# code. However, java.net is just not up to the standards as C# for this type of operation. Even though this program works, the quality of the stream was just not of the same caliber as many of the MJPEG stream projects that were done in C#. One reason that was speculated for the lack of quality of the Java stream, was because of the lack of thread usage. The threads used in the C# libraries enable processing of different sections of the stream, at different times, which allows multitasking and allows for faster speeds. Also, because this is the only MJPEG stream project that was found in Java, it proves that C# is most likely the optimum choice.

iSpy - iSpy is an open source project for maintaining and manipulating several different streams, from several different WiFi security cameras. This project’s libraries seem to be the best choice for the project. It is the best because it uses the VLC media player’s libraries as well as the libraries from AForge.Video, as well as many of its own.

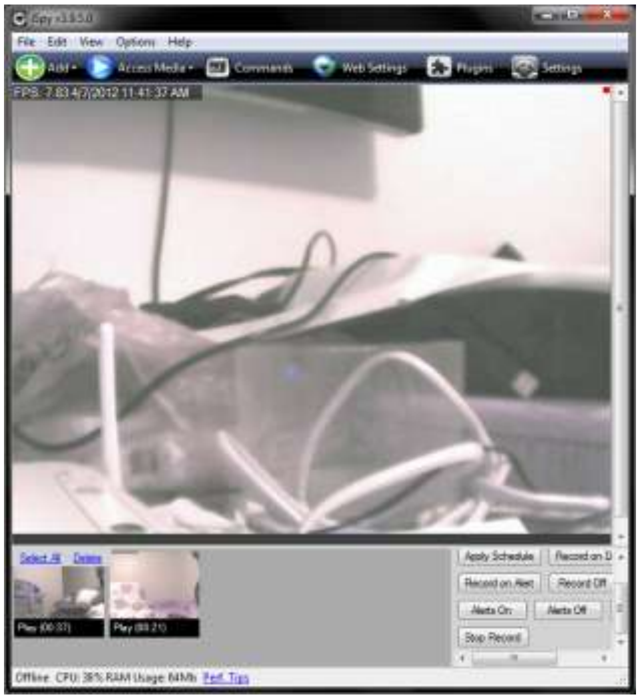


*Figure 3.5.1: iSpy MJPEG Stream*

The video feed from the WiFi camera ran the smoothest within this program as compared to any other source. iSpy also has several examples of manipulating the video feed that it receives, with options such as contrast control, and motion detection. Since the project relies heavily on the manipulation of the MJPEG video feed, having a few examples on how to successfully do it will be very helpful. So, because of the success of this program and the several examples it provides, it will be used as our primary source for information.

## 3.6 Microcontroller Research

The microcontroller that will be used in the RADSAT project needs to have a certain number of specific needs in order for it to be used for the project. Most importantly it needs to be cheap. Since there are no plans on using the actual microcontroller in the final stages of the robot build. Therefore, finding the cheapest microcontroller possible is the optimal choice. The microcontroller will only be used to give an idea of what needs to go onto the RADSAT's PCB board.

Also, the microcontroller of course has to be WiFi enabled, or at least have the option of adding a WiFi module to it. Without this part, there will be no way for the laptop to communicate with the RADSAT. Also, the WiFi part doesn't need to be extremely powerful since the data being sent and received from it will be in bytes, or kilobytes at most.

The microcontroller also needs to have a USB port plug-in. This is necessary so the board can be easily programmed. Although, there are other means of programming a board, programming a board via a USB port is what have been taught within UCF's classes, and will make programming much easier, also no solder connections will have to be made onto the board.

Finally, the board needs to have a minimum of seven I/O ports, and four analog inputs. The seven I/O ports is the exact number that is needed in order to control every part of the RC tank. The four analog inputs is the exact number that is needed in order to receive a value from every sensor that is going to be equipped to the RADSAT. The following are the microcontrollers that were researched for use with the RADSAT:

<u>ConnectCore Wi-9P 9215</u> - This board is essentially a dream board for the RADSAT project, it does everything the RADSAT could possibly ever need to do and more. It has a very powerful 150 MHz ARM core processor, but for the purposes of moving around only bytes of data, it is a little bit overkill. It also features eclipse-based programming and debugging, which is a platform that the group is very familiar with. The ConnectCore is Linux based, has tons of existing libraries, has 80 I/O pins, and can receive data from a cloud server, which would enable operation of the RADSAT, even if the RADSAT wasn't connected to the same network as the laptop.

All of these features that the ConnectCore has are great, but are outside the scope of the RADSAT project. Also, it was priced at a staggering $499. Although, the ConnectCore did give a good idea of what was out on the market, and what possible future project can be capable of, it was simply too much to handle.

<u>WiFly Shield</u> - This is an add-on part, which can be attached to any Arduino board. This is a nice piece of equipment which meets many of the RADSAT's needs. It's small, takes very little power (has a sleep mode), can communicate with 2.4GHz IEEE 802.11b/g, has ten I/O pins, and eight analog inputs. It also has an attached breadboard for development. With all that, this WiFi shield could easily be a solution for the RADSAT. However, if possible, a totally integrated WiFi shield would be preferred, and at $89.95, this is not the cheapest option available.

<u>RedBack Arduino Compatible WiFi Microcontroller</u> - The RedBack is a WiFi module which is about the size of a quarter. The small size would make the RADSAT lighter, and would enable the RADSAT to either go faster, or have additional parts added to it. It can attach to any Arduino board which has five I/O pins available. This allows for a huge selection of Arduino boards that could be chosen. It can also connect at 1megabyte to 2 megabytes per second, which is more than enough speed for the RADSAT. The only problem with this set-up is that the chip itself costs $75, and because it is not integrated with the board, it will take up five I/O pins. For the price of $75, we can purchase both the development board and shield from a cheaper, slightly larger model.

Arduino Duemilanove and WiShield - The Arduino Duemilanove and WiShield in combination are the ideal microprocessor for the RADSAT. It has fourteen digital I/O pins, twelve analog input pins, USB connectivity, has a 328P processor (slow by normal means, but perfect for what we'll be using it for), and to buy both, it costs only $75, which when all put together equals exactly what we're looking for, for our project. The only problem though, is again the WiShield is separate from the microcontroller itself. The WiShield uses seven of the Duemilanove pins to operate, which would leave no extra pins for development, in the case that a pin breaks or expansion is needed, additional parts would have to be purchased, which is not a desired scenario.

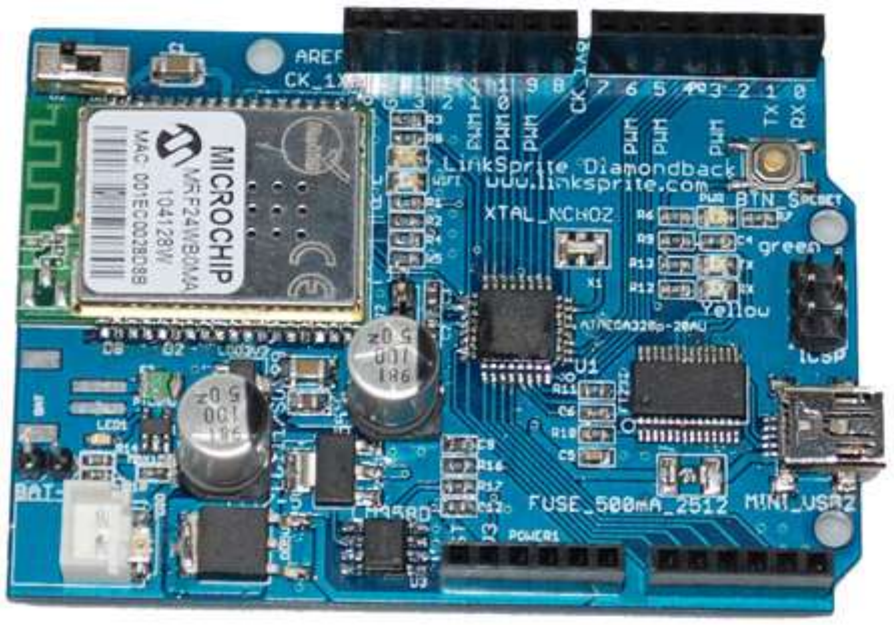


*Figure 3.6.1: DiamondBack*
*CuteDigi.com Permission Granted*

DiamondBack Arduino Compatible WiFi Microcontroller - This is the board the group ended up going with. Essentially it is exactly the Arduino Duemilanove and WiShield, only they're already both together on the same chip, opening all digital I/O ports. Therefore the Diamondback is the perfect microcontroller for the project.

## 3.7 Wireless Communication Research

In order to communicate with the RADSAT from a laptop computer, several methods of communication will need to be researched to figure out which method will best serve the groups purposes. Through initial research, several possibilities for communicating have already been discovered. One such method would be to mount a wireless router onto the robot. Another would be to use something known as an XBee adapter, with an XBee transmitter connected to the laptop and an XBee receiver connected to the robot.  One final option that will be looked at is the idea of using a microcontroller with Wifi capabilities built into the chip.

As researched earlier, it is possible to setup up a router that can be used for communication between the microcontroller and laptop over a wireless network, or connected directly to a Wifi enabled laptop.  This would require physically mounting the router onto the RADSAT, which may take up more space than desired and would add additional weight to the robot.  If this were done, however, the router would be able to receive signals from the laptop and could forward them on to the microcontroller.  There are many routers out on the market that are easily hackable for this purpose, and there wouldn't be an issue developing software in C capable of running on the router.

As another option seen in use by the team while researching the A.B.C. robot discussed earlier was that of using an XBee adapter for wireless communication.  This would require two pieces of hardware, an XBee receiver connected to the robot and an XBee transmitter connected to the computer.  This would mean that the RADSAT would only be controllable by one laptop that had the XBee transmitter connected to it, as opposed to any laptop with WiFi capability.  Though they would be light weight and wouldn't take up much room on the RADSAT, they would be an additional cost of about $10 each.  The software examples and support provided for the XBee adapters, however, is very abundant, and there would be little problem learning how to setup the wireless connection.

Finally, the team looked at microcontrollers that had Wifi capability built into the board.  Specifically, the team looked at the Arduino Diamondback microcontroller.  This microcontroller, priced at about $75, has 802.11b (Wifi) wireless capability at 1 and 2 Mbps.  This would allow the RADSAT to connect to any Wifi enabled computer with no additional hardware needed.  As the Wishield used for connectivity would be built onto the microcontroller, it would not require the use of much additional space, and would not add much weight to the overall design.

Looking at all of the options, the Table 3.7.1 below was created to compare the different methods that the team would be able to choose from in order to create a wireless connection between the RADSAT and a laptop for control over the robots functions:

| Option | Router | XBee | Diamondback |
|---|---|---|---|
| Pros | Easy to hack, already owned by team, easy to program | Small, affordable, easy to setup, a lot of support offered | Less setup required, easy to program, connection to any wifi enabled laptop |
| Cons | Bulky, extra weight | Limits control to one computer | More expensive option |

*Table 3.7.1: Wireless communication comparison*

In the end, the team decided upon using the Diamondback microcontroller from Arduino as a basis for designing a unique PCB to create the wireless connection. Though this microcontroller will be used in the initial design of the RADSAT, it will be the PCB that is used in the final design. This will allow the team more opportunities for design. Set up and coding for the diamondback microcontroller will be looked at in later sections, and will be used as the basis of the final coding for the PCB.

## 3.8 Communications

As specified in the software requirements and requirements section, RADSAT's many different functionalities and algorithms must have some method of relaying information to and from both the software components and the hardware components.

### 3.8.1 Inter-process Communications

RADSAT's various software components must either be able to communicate with one another, or be consolidated into one program. In the case of GUI and the main control components, it is natural to think that they would indeed be one in the same program (in fact, virtually all C# components will be within the same program). A number of major conflicts arise when considering any number components written in different programming languages. Although the components may not need to directly communicate with one another, eventually, down the line, indirect communication must happen. There are a few different ways to implement inter-program communications, all with their advantages and disadvantages. This section will only cover three of the methods: file system (or a central database), sockets, or using sub-processing.

Using a file system to enable communications between two different processes is a rather simplistic solution, but can still be effective. If an intermediate file is used between processes, then each process would be required to have read and write access to the file. To ensure consistency, virtually all reputable operating systems will restrict a file from being accessed if another process is accessing the file in such a way that the contents could be changed. To help mitigate the effects of being denied access to a file, each process could alternate turns with a timer. Alternatively, the same concept could be applied to a database. Most databases are equipped to handle concurrent access on the order of hundreds of users (and many are able to handle thousands), so finding one capable of handling two processes would be trivial. However, the biggest disadvantage is the fact that the additional overhead of a database would add to the time needed to process information sent to and from RADSAT.

Another option could be to use sockets to communicate over IP between two separate processes on the same system. This approach has the advantage of being a hugely platform independent solution. The latency time would be contained because communication over IP between two systems on the same network is minor, but when communicating with two processes on the same system is very negligible. While C# has very developer friendly classes to handle communication over sockets, C is not as simple to use. It would likely add another level of complexity in the form of having to use a library like Winsock to handle the communication.

Sub-processing is by far the most robust option that will be considered. It essentially allows for direct communication between a host process and its slave process.

## 3.9 Autonomy and Sensors Research

### 3.9.1 Introduction

Implementing autonomous path finding is already a challenge in and of itself, but when coupled with the fact that RADSAT will not possess a method of knowing its exact absolute position at any given point in time (because of the lack of GPS), the feat starts to seem even more insurmountable. One advantage that RADSAT has is that its objective isn't to go from point A to point B with minimal cost, but to start at point A and find a potentially hidden point B. Because finding the most optimal path without first knowing the location of point B is impossible, minimizing cost should be done by ensuring that the same area is searched the fewest possible number of times. To adequately search a given area, RADSAT only needs two pieces of information.  The first necessity  is the relative distance between it and any obstructions that are within range, both in front and to the sides of its body. A way to identify the color that it is searching for once an instruction has been given to it is the second. Another life saving advantage of RADSAT is that sensing obstructions and identifying colors are independent and one can be researched and designed without regards to the other.

There are generally two basic types of obstructions. The first type consists primarily of walls and objects with large surface areas in both the X-Z and Y-Z planes. The next type of obstruction are objects that are generally "thinner" than the former type and have relatively small surface areas. The most notable example of this kind of obstruction is a chair's leg, which is also known as an autonomous robot's arch-nemesis because of the high level of difficulty that exists in detecting them. In order to achieve an acceptable level of autonomy, RADSAT will need methods of detecting both type 1 and type 2 obstructions, as well as obstructions that are derived from the combination of both types (e.g. a thin structure with a large surface area on one side). Ideally, in a perfect world, there would be one sensor that would be able to identify all types of obstructions with near perfect accuracy. In the real world, different types of obstructions need to have different methods of detection to minimize costs and maximize efficiency.

### 3.9.2 Sensing Type 1 Obstructions

During the very early researching phases of developing RADSAT, the team initially had the idea of using a number of flex/force sensors, each as a type of robotic feeler, effectively transforming them into proximity sensors (referred to as feeler sensors in this section). This idea was quickly dismissed as research progressed because of their limited usage and because of the inability to *directly* generate an output voltage (due to the sensors being based on the concept of variable resistance). It is also generally impossible to use multiple feeler sensors to consistently detect obstructions that are smaller than the minimum distance between sensors. This claim is easily verifiable when viewing sensors that are mounted, and as such, feeler sensors will not be covered to a great extent in this section. They will, however, be mentioned sporadically for comparison purposes.
Infrared sensors are only able to detect obstructions that are sufficiently wide because, like feeler sensors, they can only detect type 2 obstructions in very specific circumstances. However, ultrasonic sensors (sometimes referred to as sonar sensors), are slightly more complex than their infrared counterparts and generally cost more. Both work using the same general concept: send some kind of a signal and take note of the time it takes to receive a reflection, or the angle at which the reflection is returned. Although the concept is very simple and intuitive, there are complications that arise when trying to incorporate either type of sensor in RADSAT.

There are generally two relevant types of proximity sensors; digital sensors and analog sensors. Both types are able to detect obstructions within their respective ranges, but the major difference is in the method used to deliver that information to the microcontroller. Digital sensors are sensors that are only able to inform of the existence of obstructions that are within range. This type of sensor is able to output a high voltage level when something is detected, but neglects to include information regarding the distance to the obstruction. Analog sensors are those that output an analog voltage that somehow relates the output voltage to the distance to the obstruction. Of course, there are some types of digital sensors that are able to give distance information using multiple output lines, though these sensors are very expensive.

| **Sensor** | **GP2Y0D02YK** | **GP2Y0A02YK** | **FSL0095103ST** | **MB1210** |
|---|---|---|---|---|
| **Method** | Infrared | Infrared | Physical Contact | Ultrasonic |
| **Output** | Digital Voltage | Analog Voltage | Variable Resistance | Multiple |
| **Real Distance** | No | Yes | Yes | Yes |
| **Range (Min)** | 20cm | 20cm | 0cm | 20cm |
| **Range (Max)** | 150cm | 150cm | 11.43cm | 765cm |
| **$V_{cc}$ (V)** | 4.5 to 5.5V | 4.5 to 5.5V | N/A | 3.3 to 5V |
| **$V_o$ (V)** | -0.3 to ($V_{cc}$ + 0.3) | -0.3 to ($V_{cc}$ + 0.3) | N/A | 0 to 3V |
| **Approx. Price** | $12 | $15 | $15 | $45 |

*Table 3.9.2.1: Comparison of GP2Y0D02YK, GP2Y0A02YK, FSL0095103ST, and the MB1210 proximity sensors.*

The GP2Y0A02YK (figure 3.9.2.2) and GP2Y0A02YK infrared sensors both use triangulation to determine the distance between it and the obstructing object by calculating the angle between the infrared signal sent and the one reflected back. The voltage output is generally between 0.4 and 3.0V when it detects the incoming reflection and generally has a minimum usable range of about 20cm. More importantly, it is able to detect obstructions up to a maximum of 150cm which makes it one of the most suitable sensors for detecting type one obstructions.

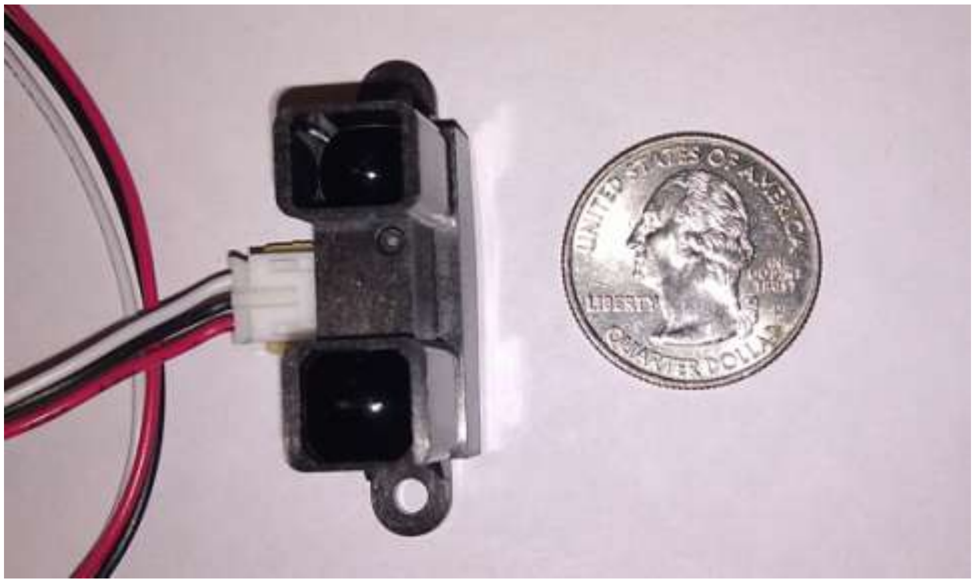


*Figure 3.9.2.2: The Sharp GP2Y0A02YK infrared sensor with the Japan Solderless Terminal (JST) connector.*

A problem with using an analog sensor is while the distance is supposed to theoretically be proportional (or inversely proportional) to the output voltage, that isn't necessarily true during its actual application. The Sharp sensor's output voltage *increases* exponentially as the distance *decreases* down to a certain minimum sensing range (which is marginally below the documented minimum). At that point, a reversal happens and the output voltage *decreases* linearly as the distance *decreases* with both eventually approaching 0. This awkwardly surprising relationship between the output voltage and the sensing distance (figure 3.9.2.3) actually introduces two major problems in the design of RADSAT. Because the usable output voltage has an exponential relationship with the distance, solving for the distance isn't as simple as just multiplying the voltage by some constant. The other problem is that every usable output voltage detected at a distance *greater* than the reversal distance has a matching and presumably unusable output voltage that could be the result of sensing an obstruction at some distance *less* than the reversal distance. The first problem can potentially be solved by developing a mathematical function to relate the distance and the output voltage, but it is much too cumbersome of a process, and can only achieve approximate results at best. Because of this inconvenience, a rather unintuitive approach of developing a lookup table will be used to solve the problem.

Developing a lookup table is quite unintuitive because it requires taking the continuous, right portion of the graph in figure x and translating those values to their respective distances at discrete intervals (i.e. sampling the signal). The length of the interval will depend on the minimum resolution that would be necessary to meet or exceed the requirements and specifications. Assuming the maximum interval length is agreed to be an inch, then it would be necessary for the lookup table to have a minimum of 52 entries (150cm MAX - 20cm MIN, converted to inches). Specifying an interval smaller than an inch would require more entries and anything larger than an inch would require fewer entries in the lookup table.

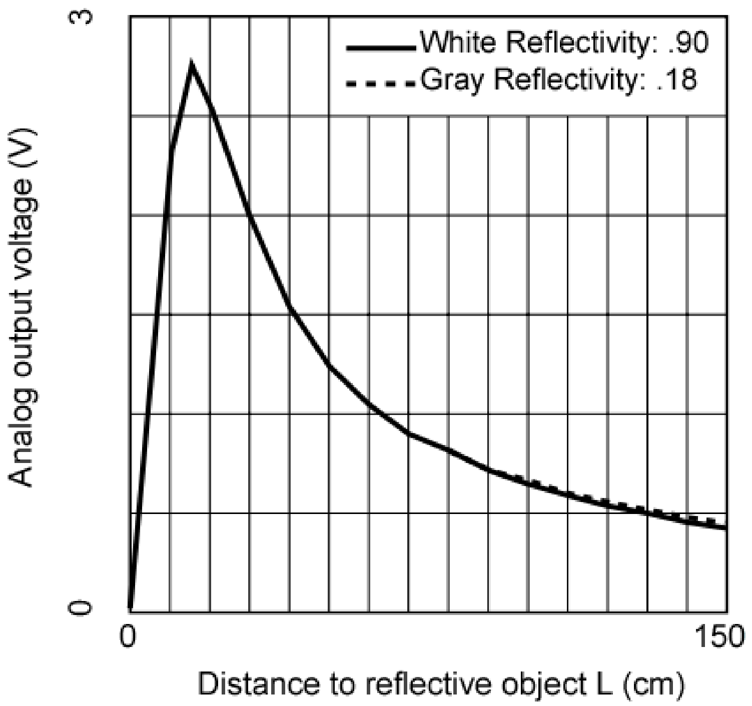


*Figure 3.9.2.3: Representation of the relationship between the output voltage and obstruction distance.*

Mapping each voltage to its distance according the graph in figure x is possible, but doing the inverse and mapping the discrete steps in distance is much simpler because the change in distance remains constant. Calibrating the sensors is absolutely necessary because it is very unlikely to receive a sensor that will follow the manufacturer's output voltages exactly (due to numerous different factors). The simplest way to calibrate the output of a sensor and to build the lookup table is to measure the output of the sensor and compare it to the distance between it and the obstruction. This process eliminates any potential error associated with the manufacturer's ratings and gives a lookup table that is guaranteed to be fairly accurate.

Recall that the second problem had to do with a sensor being unable to tell whether an obstruction was closer than the minimum range or further because each valid output voltage is related to two different distances. For example, an output of 2V on the Sharp GP2Y0A02YK sensor could potentially be achieved by sensing an obstruction at both 10cm (invalid) and at 30cm (valid). Fortunately, this problem can be solved with clever thinking and using a very intuitive solution. If the set of invalid values occurs only at distances approximately less than 20cm, then mounting every infrared sensor in such a way that all possible invalid values must fall within the tank's outside perimeter (Figure 3.9.2.4) would be an ideal way to solve the problem of output voltages being associated with more than one distance.

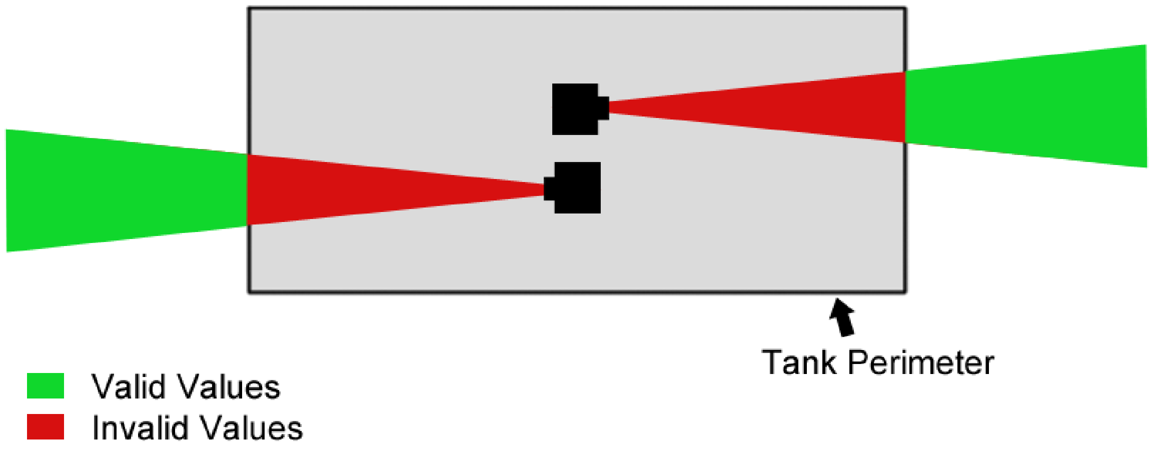


*Figure 3.9.2.4: Infrared sensor alignment image depicting valid and invalid output ranges. Not to scale.*

## 3.9.3 Sensing Type 2 Obstructions

The team initially thought that sensing very thin type 2 obstructions was generally impossible given the resources available and the scheduled project completion date. A sensor able to detect this type of very elusive obstruction should logically be able to sense other types of obstructions as well, because other types of obstructions are harder to detect. This type of sensor would most likely be expensive and relatively hard to find, but would probably work extremely well.

Using a feeler sensor to consistently detect these obstructions has already been established as generally being impossible, so the focus of this section will be to compare infrared and ultrasonic sensors. Recall that the Sharp GP2Y0A02YK infrared sensor uses triangulation to determine the distance between it and an obstruction. An infrared signal is sent from one end of the sensor, and the receiver on the opposite end receives the signal at some angle (and this angle is used to calculate the distance). This approach may seem ideal, but after considering the beam size of the infrared sensor (which is only a few millimeters in radius), using it to detect type 2 obstructions becomes increasingly difficult. There exists infrared sensors that are more capable of more broad detection, but again, these sensors are prohibitively expensive. A solution could be achieved by simply mounting the sensor on a motor capable of bidirectional rotation in very small increments (possibly a servomotor). Rather than simply being static, the infrared beam would instead rotate with the motor, which in turn would allow the sensor to effectively scan a relatively wide area. This process would eliminate the problem associated with the lack of a wide beam. Mounting the sensor on top of the wireless camera (which is required to rotate) could be one possible configuration. While this solution is perfectly achievable within the allotted timeframe, it would introduce many layers of additional complexity that could have otherwise been avoided if RADSAT utilized a suitable sensor.

Although very possible, the prospect of using light as a viable medium to successfully detect type 2 obstructions is very dim. Alternatively, the MaxBotix MB1210 is a self-calibrating, ultrasonic sensor (figure 3.9.3.1) that uses the effects of reflecting sound to detect impeding obstructions. Under ideal conditions, this sensor could accurately detect the existence of obstructions within a distance of 0cm to a maximum of over 760cm. The sensor has a very convenient and compact design (very apparent when compared to the Sony sensor), allowing it to easily be incorporated into the design of RADSAT.

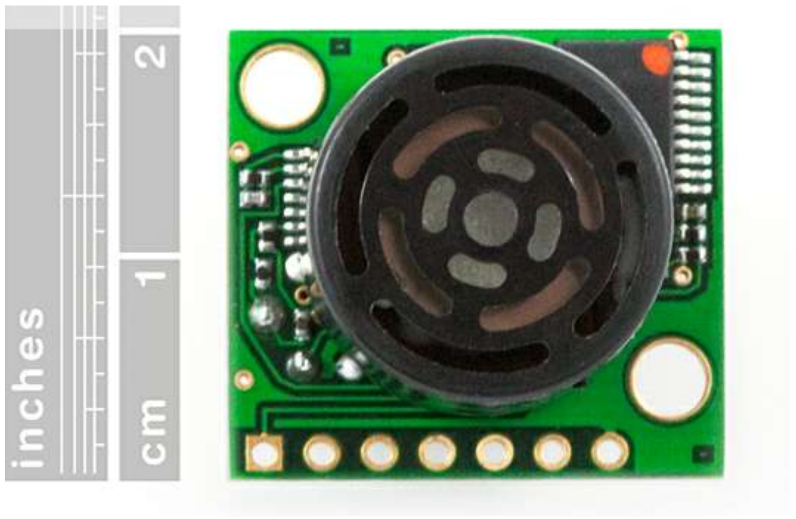


*Figure 3.9.3.1: The MaxBotix MB1210 ultrasonic sensor,* reprinted under license (CC BY-NC-SA 3.0) from Sparkfun.com.

The sensor itself has numerous other advantages which justify the increase in cost over Sharp's GP2Y0A02YK. One of these advantages is that although both sensors are unable to accurately sense obstructions below a certain minimum distance (around 20cm in both sensors), the MB1210 does not have the problem of associating virtually every output voltage with more than one distance. Instead, if an obstruction is detected at a distance below 20cm, the sensor will output a voltage equal to what would have otherwise been the output for an obstruction detected at exactly 20cm. In more concise terms, the sensor's output voltage remains constant at any distance less than or equal to 20cm. This feature greatly simplifies the problem of choosing a mounting location for the sensor because although 20cm is the minimum range, any voltage output values at a distance below 20cm *aren't* considered invalid (unless the obstructions are located between 0mm and 1mm, which is nearly impossible). The value of all analog output voltages on the MB1210 follow this general equation:

$$V_{cc} / 1024 \text{ per cm, where } V_{cc} = 5V \text{ or } 3.3V \quad (1)$$

Using equation (1), it can be found that a $V_{cc}$ equal to 5V will result in an output voltage of approximately 4.9mV per cm, while a $V_{cc}$ of 3.3V will result in an output voltage of about 3.2mV per cm. It becomes immediately apparent that another benefit with using this sensor is that the analog output voltage is linear and is very neatly defined. As such, it becomes unnecessary to construct a lookup table to handle all of the possible valid output voltages, as is necessary with the GP2Y0A02YK infrared sensor. Although the maximum range is listed as over 760cm, hardware limits the maximum reported voltage to approximately 700cm when using 5V and 600cm when using 3.3V.

Although considered unnecessary for RADSAT, another advantage with this ultrasonic sensor is that it enables reading from a total of three different sensor outputs (analog voltage, serial digital, and pulse width). If there happened to be a major design change anywhere down the line which rendered the analog voltage out port unusable, then there would still exist two other output lines to read values from. Figure 3.9.3.2 outlines the MB1210's beam pattern and

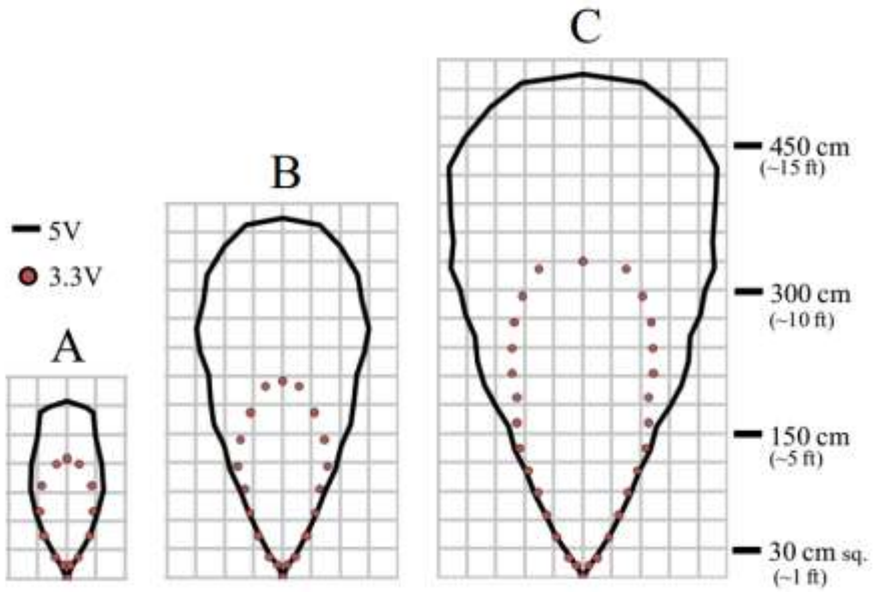


*Figure 3.9.3.2: Beam pattern of the MaxBotix MB1210 ultrasonic sensor. Reprinted with permission pending from MaxBotix.*

## 3.1.4 Heading and Searching

During the initial brainstorming phases, a debate ensued internally within the team regarding whether or not it was necessary for RADSAT to have a sensor that could explicitly calculate its heading. This question is directly related to and could be directly answered by first determining the kind of searching algorithm RADSAT would use to find its target. One potential method is to simply move along the walls of a room and to mostly rely on RADSAT's camera to find its target before a full lap is complete. If a full lap is completed before the target is found, from a third person's perspective, another lap would be started. A viewer would take notice of RADSAT performing rounds around the room an infinite number of times (worst case), but as far as RADSAT and its inputs are concerned, they are just continuing the search. The problem would generally occur if the target is in the middle of a rooms so large, that the camera would be unable to actually find the target or if the target is located in the middle of two obstructions in the middle of a room. "Around the room" searching could theoretically work, but it would produce insufficient results and would contain too many cases where it would simply fail. It is possible to modify the search by travelling in a random direction if it has been moving free from obstruction for a certain length of time (essentially lowering the probability that only travelling along walls would not happen). Although the modification is an improvement, it is still not as effective as the team would prefer.

A more effective method is for RADSAT to search a room with full knowledge of its relative facing direction. There are a few different magnetometers and compass sensors out there, but the one that seems most compatible and easy to use is Honeywell's HMC6352 Compass module (figure 3.9.3.3). The module gives has a heading resolution of 0.5 degrees, meaning it can detect a total of 730 different possible headings. Considering the fact that even 50 different headings would lead to a reasonably precise number, concluding that the sensor is accurate enough for our purposes can be justified. The biggest drawback this sensor (and most other inexpensive magnetometers) has is that because it uses the Earth's magnetic field as its basis for the direction north, if it enters another magnet's magnetic field, then the HMC6352 will measure north from that magnet's north pole.

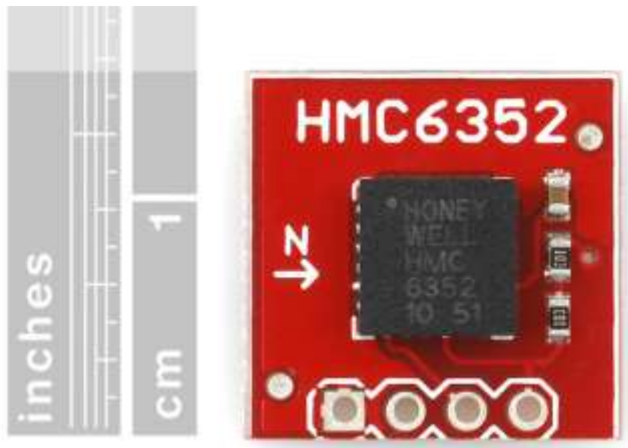


*Figure 3.9.3.3: Honeywell HMC6352 Compass module, reprinted under license (CC BY-NC-SA 3.0) from Sparkfun.com.*

With RADSAT now possessing this newfound ability to determine exactly which direction it is facing, more elaborate searching algorithms can then be to developed to enable its search both more intuitive, efficient, and robust. The most intuitive way to search an area is to start anywhere in a room and to travel in one direction until an obstruction is found, it can then complete a 180 degree rotation in such a way that it avoids obstructions to the left and to the right and then starts to scan that area until either an obstruction is found, or the target is found, as shown in the flowchart in figure 3.9.3.4.

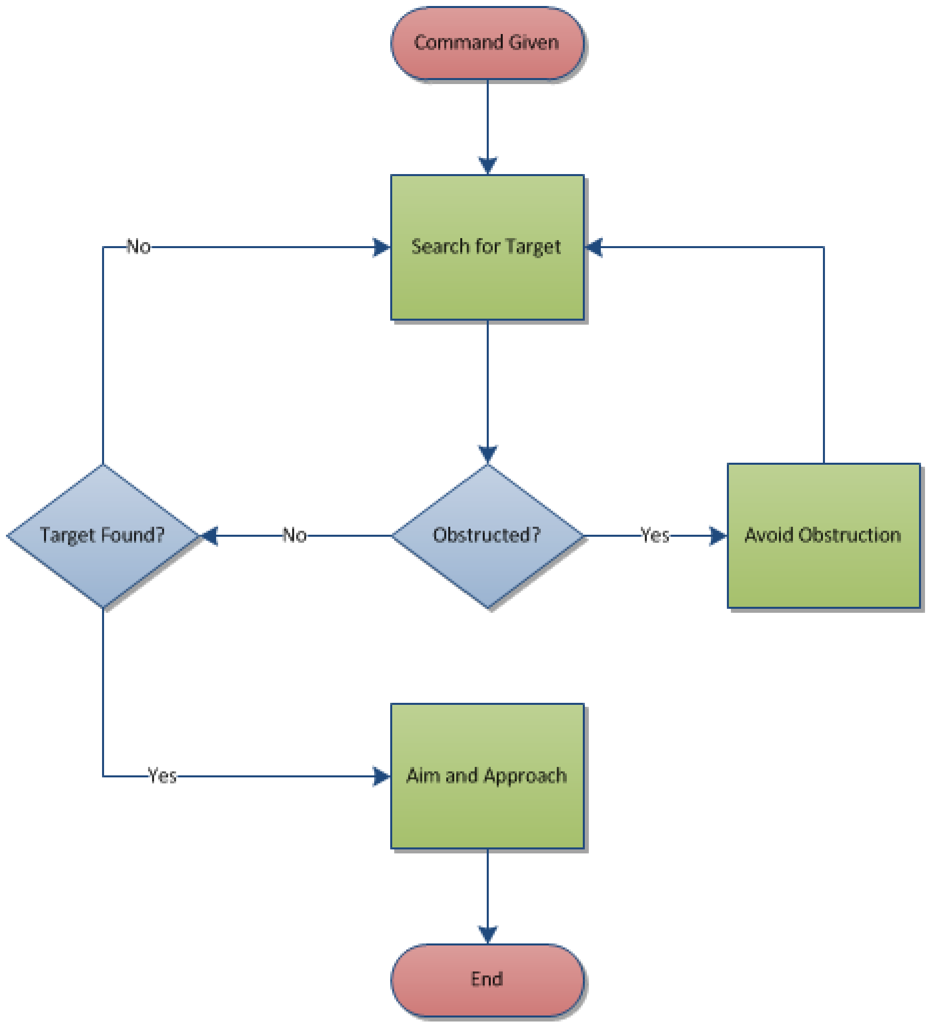


*Figure 3.9.3.4: Flowchart depicting target searching and obstruction avoidance.*

At this point it is both appropriate and necessary to discuss whether the sensor processing will be handled on board, via the micro-controller, or by the computer. This depends on a wide range of factors. These include, cost, difficulty, speed, capabilities, bandwidth, number of sensors. According to the project requirements the speed at which the robot can respond to the sensors is of high importance. Even more important is the accuracy and reliability; in a word, effectiveness. These factors are primary, and factors such as cost, ease of execution and aesthetics are secondary. A decision will be made based on what information can be gathered before the prototype is built.

The parallel for this decision is the human brain. In humans the cerebral cortex processes deep thoughts and can handle complicated situations requiring reasoning, decisions, judgments and planning. The medulla oblongata, on the other hand, is a more basic part of the brain and handles unconscious movements, reflexes, and reactions such as making the heart beat, controlling the muscles used to sneeze, and blinking when something gets near to the eye.

In many ways, RADSAT should process the sensor input like the medulla, automatically producing one preset action for a given stimulus. This is a starting point for this discussion. After reviewing the facts, the decision will be made whether to modify this hypothesis. The four sensors together provide the following information: The compass will provide directional orientation which will be useful for determining where the tank has been and is going to be in the very near future. The two infrared sensors are useful for sensing if the side of the tank is approaching or close to a wall or obstruction. Finally, the ultrasonic sensor provides information as to what is in front of the vehicle. These give the tank spatial relation to its immediate surroundings.

For a given sensor to relay information to the computer or to the microcontroller it takes such a short time it is almost instant, and therefore, both about the same. Thus, reaction time is not an issue. The difficulty of programming the microcontroller to give a response to a sensor input is not high. If, for example, the data from the sensors was to be recorded and used to construct a virtual map for memorization of obstacle location, then it would need to be relayed to the computer. Because that is not within the scope of this project, handling sensor feedback will be done on the microcontroller. It is the best, easiest and most reasonable way. This is discussed further in 4.6.3 Microcontroller Design.

## 3.10 Vocal Command Recognition Research

### 3.10.1 Introduction

One of the main objectives of RADSAT is the ability to respond to vocal commands in an effective manner. As such, the ability to analyze an incoming stream of sound is an absolute necessity. Researching different voice and speech recognition libraries led us to a plethora of different approaches to solving the problems associated with designing RADSAT. Most speech recognition libraries use a general purpose, one size fits all approach to tackle the problems that they were designed to solve. While these libraries are very useful for reference purposes, we specifically sought out solutions that have a more targeted purpose.

The deliberation process for choosing a library to implement to verbal command recognition software was comprised mostly of the team asking, "which set of tools made the most sense and allowed for a relatively large amount of potential expansion later on?". Making the most sense implies that it should probably be free, easy to integrate the resulting program into the system, and relatively easy to interface with the actual library. We narrowed it down to using either Android, Port Audio, Sphyinx-4, or general Windows API/.NET libraries. We compared and contrasted the most relevant aspects of each respective library (Table 3.10.1.1). Three are very plausible solutions that could be used in the design and development of RADSAT, but using the various Windows APIs (including .NET and DirectX libraries) for word recognition are included for comparison purposes and will not be discussed in detail in this document.

| **Library** | **Android SDK** | **Port Audio** | **Windows API/.NET** | ***Sphinx-4*** |
|---|---|---|---|---|
| **Platform** | Android (Linux based) | Cross Platform | Windows | Cross Platform |
| **Language** | Java | C/C++ | .NET | Java |
| **Integrability** | Low | High | High | High |
| **Difficulty** | Low | Medium | Medium | Low |
| **Recognition** | Yes | No | No | Yes |
| **Cost** | Free | Free | Free | Free |

*Table 3.10.1.1: Audio Library Comparisons.*

## 3.10.2 Sphinx-4

Like Android's speech recognition library, Sphinx-4 is also a Java library able to translate entire words and phrases (both discrete and continuous) to English strings. This library is notable not simply because it's another easy to use and open source Java library, but because it was made in collaboration with many different educational institutions and research labs. Sphinx-4's documentation includes the resources needed to use its set of classes effectively and, more importantly, it includes an overview of the mechanics that power the library. The most useful part of the overview was their description of HMM (Hidden Markov Model). HMM (figure 3.10.2.1) is a simple statistical model that essentially has hidden states with a probability to enter each one of the states. A phoneme in HMM is the smallest unit of a word and is represented in the figure by the circular states. Each phoneme has certain properties that are included in the model, and based on those characteristics, a decision can be made whether or not the algorithm will enter that state.

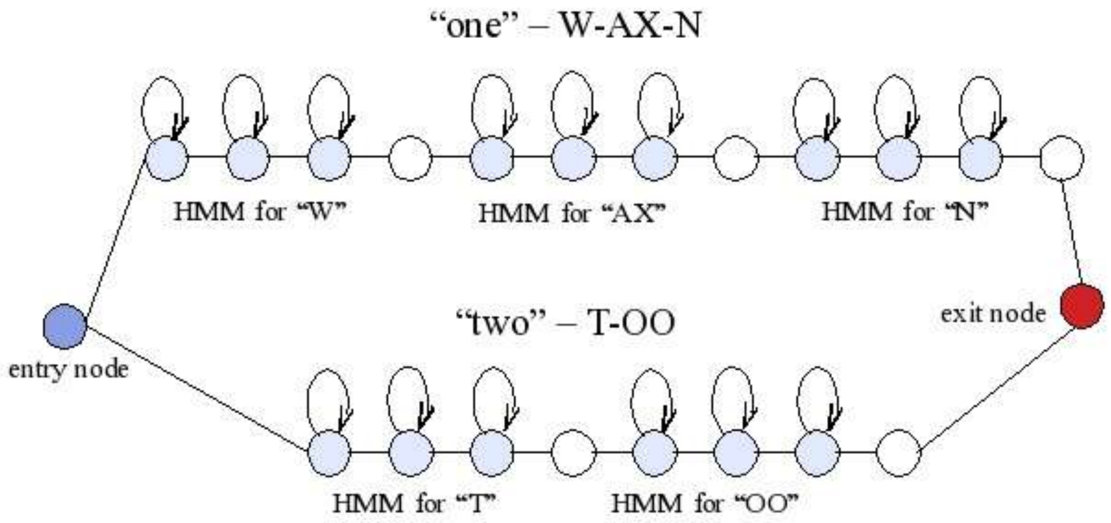


*Figure 3.10.2.1: Sphinx-4 search graph. Reprinted with permission pending from the Sphinx-4 Foundation.*

Although Sphinx-4 is open source and is a great tool to use, it requires very little actual design, but is very easy to actually implement. It is included in this section as a reference tool for RADSAT and future development.

## 3.10.3 Port Audio

Port Audio is a cross platform and relatively low-level audio library used to interface with recording devices and is able to provide access to an incoming audio stream for further processing. It is written in C and allows the programmer to write programs in either C or C++. Its biggest advantage lies with its ability to be extremely powerful while enabling a great deal of flexibility. Port Audio gives the developer the ability to specify the recording time, the sample rate, and other criteria at will. It returns floating point values that represent the level of sound it has recorded.

Audacity is a free, open source and cross platform program used to record audio and edit all types of audio files. There are numerous of programs that do the same thing and can handle the same operations, but what makes Audacity truly remarkable is that it uses Port Audio to handle the overwhelming majority of its audio I/O and processing. As shown in Figure 3.10.3.1, Audacity puts many of Port Audio's features on display as it allows you to record a sound and it generates a real-time visualization of that sound, which is exactly what the RADSAT project will need to do.

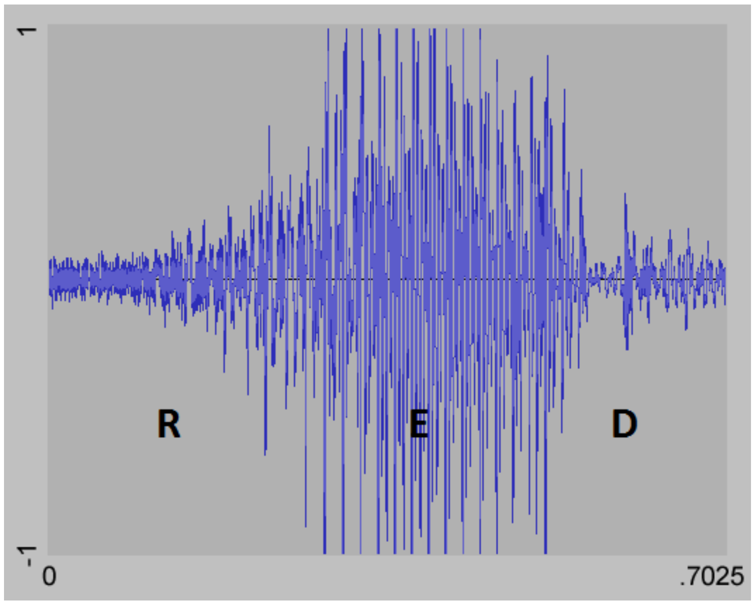


*Figure 3.10.3.1: Sampled vocal visualization of the word "red", recorded using Audacity*

## 3.10.4 Android

Android is a Linux based operating system developed by Google and is largely focused on being compatible with many different devices including different types of phones and tablets made by competing manufacturers. The popularity of Android stems from the fact that it's backed by Google, it runs on many different mobile devices, and it has a very expansive and robust set of libraries and APIs that can handle virtually any task a programmer can imagine. One of these libraries (or packages in Java terminology) includes the code for the *SpeechRecognition* class. This class equips even a novice developer with the ability to recognize incoming sound and to translate that sound into a data string. All a programmer has to do is let the Android operating system know that it should start listening for speech. From there Android's built in speech recognition will handle listening to the user and translating what the user said into something more usable.

One of the core requirements is that a video feed needs be displayed for the user. So, it would make logical sense for RADSAT's controlling device to also be able to display an incoming video feed to allow the user to see what it's seeing *while* controlling it. One big disadvantage with using Android is that streaming video (or even a series of images) from a wireless feed while communicating with another wireless source seemed extremely difficult, if not impossible to do. When that inability is combined with having to process vocal commands, doubts began to arise regarding whether or not an Android app had the ability to process everything that it was expected to handle.

In the beginning, the team was very biased towards using Android as the main operating system for very obvious reasons. Not only would we have been excited to be able to deploy the controlling aspects of the system onto cell phones and tablet devices, but the project itself would be greatly simplified by using Android SDK's built in classes to do general word recognition for us. However, in the end, the team decided against using Android as a solution because of its limited computing capacity and because it would be difficult to integrate all of the features necessary to adequately control RADSAT.

### 3.10.5 Possible Solution

The approach that will likely be used to enable voice commands and recognition in RADSAT is to integrate the open source "Port Audio" sampling C library to interface with the microphone of a personal computer. The library will allow the system to stream and sample any sound recorded from the microphone at some defined frequency and resolution, but does not include any other type of voice related functions. It is impossible for an algorithm to differentiate between random blabber, noise, and an actual command without first making some kind of a comparison. This implies that the first step to verifying a potential command is to pre-record actual commands and to build a dictionary. The recording and verification process can happen in one of two ways.

One method is to record an array of fully playable sounds, with each one representing what an actual command should sound like. Then, when a potential command is sampled by the user, it can be compared to each one of our prerecorded sounds until a match with minimal deviation is found. If no suitable match is found, then the potential command is not considered a verified command and is discarded. An alternative method is to develop a nonstandard, custom file format that will contain the length of the sampled command, and an array of floating point numbers representing sampled points from an already sampled stream. Any potential command will be re-sampled, stored in a data structure, and compared to each one of our verified commands. A deviation will be calculated, and the final steps in this approach will follow those in the first method. It is important to note that in both cases, the deviation is not calculated by simple subtraction, but by comparing the change in frequency between two adjacent (or in some cases, relatively close) points in the sample. This is done to ensure that the differences in each person's natural tone is compensated for and acknowledged in the algorithm.

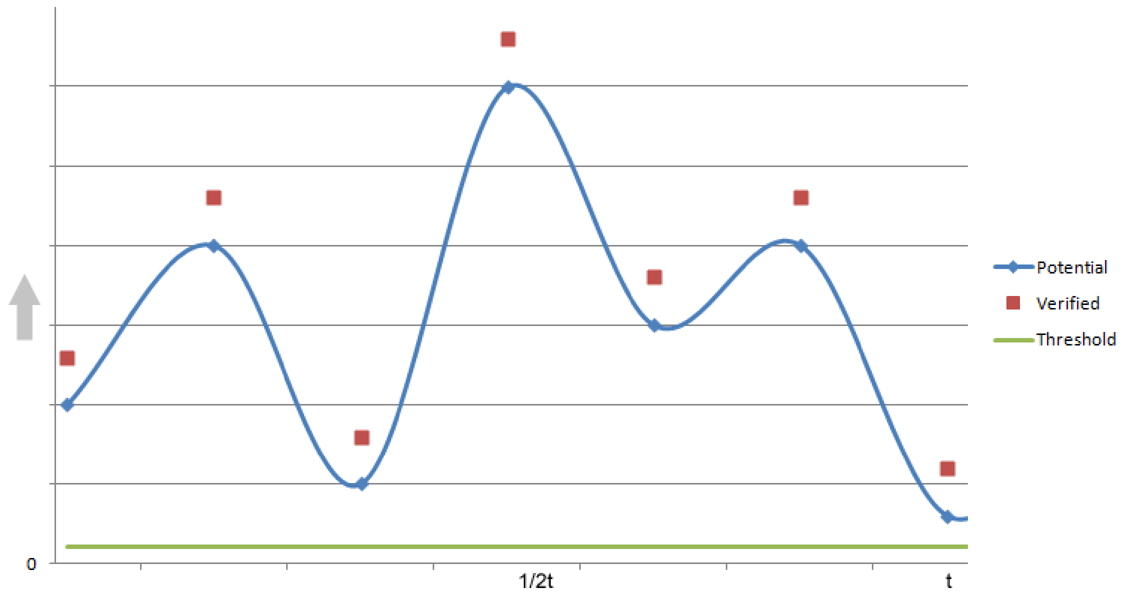


*Figure 3.10.5.1: Representation of a stored sample of a verified command, an incoming (potential) command stream, the threshold, and a normalized time axis.*

Another problem is determining whether or not the system should consistently stream sound to the application on an ongoing basis, or if pauses should be implemented to handle memory and processing limitations. This problem could be easily circumvented by streaming (but not storing on the hard disk) audio samples continuously and "freeing" unused data structures that are stored in memory after receiving samples that are below a certain threshold, consistently, for some length of time, $\alpha$. Doing this may introduce undesirable side effects, especially in uncontrolled environments. The issue is the possibility that constant noise will produce samples that are always above the threshold, and will result in the stream never pausing to free memory. For instance, if the aforementioned low-leveled time limit, $\alpha$, is set to 60 seconds and low-leveled sound is being recorded, consistently for 58 seconds, then the stream will continue recording as expected. However, if a loud sound is heard at 59 seconds, then the time counter will be reset from 58 and will start to approach $\alpha$ from 0, but the recorded stream itself will not be reset and memory consumption will continue. To put it concisely, it's possible for a stream to continue, without pause, indefinitely even though $\alpha$ is set to any reasonable value. This can be solved by having a universal time limit that would automatically stop a stream and free data if the stream has been continuously receiving data for too long.

RADSAT supports two categories of vocal commands: directional and action commands. Directional commands deal with the movement of RADSAT while action commands tell it what it should do. Another major difference is that directional commands do not take precedence over sensory input, while action commands can, only if RADSAT is able to follow the command. For example, it will not fire unless it has found and taken aim at its target. Table 3.10.5.1 lists the description of some of the vocal commands accepted by RADSAT.

| **Command** | **Up** | **Down** | **Left** | **Right** | **Stop** | **Fire** | **Find** |
|---|---|---|---|---|---|---|---|
| **Type** | Direct | Direct | Direct | Direct | Action | Action | Action |
| **Precedence** | No | No | No | No | Yes | Yes | Yes |
| **Desc.** | Moves up | Moves down | Moves left | Moves right | Stops RADSAT | Fires at target | Con't Find |
| **Prereq.** | Sensor | Sensor | Sensor | Sensor | None | Target | None |

*Table 3.10.5.1: Subset of all vocal commands and their effects.*

These commands are only a subset of the full set of vocal commands that RADSAT will be able to support.

# 4.0 Project Hardware and Software Design Details

## 4.1 Hierarchy of Decisions

Throughout the entire project, many choices and decisions were made or revamped in regards to the final design. In this section, these choices, revamps, and final decisions are outlined. The discussion that will follow will also include the teams reasoning for each of the decisions.

At the beginning, the team came together under the idea of building a robot. In the initial stages, it was not clear or defined at all what the functions of this robot would be or how they would be carried out. After a few meetings and deliberations between all of the team members, an initial set of goals was outlined. Though the project would eventually evolve to its more sophisticated final design, the goals and objectives that were outlined in these initial meetings would serve as a basis for the robot that would eventually be dubbed Reconnaissance and Demolition Super Attack Tank, a.k.a, RADSAT.

First, the team deliberated on what the size of the robot would be. Though a large robot was discussed, it was unanimously agreed upon by the team that the robot would be compact and light-weight. With this in mind, the chassis was then discussed to determine how the robot would move around. Several options were examined to see what possibilities the group had to choose from. It was agreed upon at that time that a tank chassis would be best fit for the project and movement would be easily controllable. With the initial framework coming together, the idea of a tank sparked the group's interest of a robot that would be used as a weapon.

With the idea of a tank now as the motivation for the group's future workings, additional goals were laid out. First, specifications for the length, width, height, and weight of the tank were defined. Following this, it was determined that the tank should be wireless and controllable from a laptop computer. A discussion was then held amongst the group of how communication would take place between the laptop and the tank. It was initially determined that a hackable router would be connected to the tank. With this router in place, the team would be able write a program on the laptop that would communicate with the tank by sending signals to the router. With this design, the group would be able to manually drive the tank around in the specified radius of the wireless connection. A webcam was to be hooked up to the router so that it could send a video stream back to the laptop. Using this video feedback, the group would be able to drive the tank to a specified target. Once the target was discovered, the group would be able to issue a command to the tank that would fire the turret at the enemy target. It was at this point that the tank was named RADSAT, in respect to its nature as a weapon.

With the initial design of RADSAT laid out, the team presented this idea to Dr. Richie, the electrical engineering and computer engineering Senior Design instructor at the University of Central Florida. During the group's discussion with Dr. Richie of the functions of the RADSAT, it was agreed upon by both parties that not enough design had been included to satisfy what would be a suitable Senior Design project. A discussion of new design goals that could be added to RADSAT took place, and the team returned to the drawing board.

The team, after performing more research on other wireless vehicles that had been designed by other engineers in the past, then held another meeting together that would outline the goals for the final design of RADSAT. In order to add more design to RADSAT that would ultimately lead to a good Senior Design project, some of the previous decisions that had been made were changed and some functions that would add more room for design by the team were added. These changes and decisions are discussed as follows.

The first major change that was made for RADSAT was how it was going to wirelessly communicate with the laptop. After discussing the idea of a router attached to the R/C tank's body, it was ultimately agreed upon that this implementation would be simply to easy use for a Senior Design project. Furthermore, the use of a router did not seem to apply well to for real-world application. The idea of the router was thusly scrapped, and a new solution to the problem was researched. In the end, it was agreed upon by the group that a WiFi Shield would be implemented on the team's PCB to communicate with the laptop. This offered a functional solution that would offer new opportunities to the team for designing software for RADSAT.

Next, the group looked at how RADSAT would function in its search for a target. Though the group agreed that the R/C tank should still be able to be manually controlled by a human user over via a laptop, this did not seem to be good enough for the project. In addition to manual control, the group decided that RADSAT would have an autonomous aspect in the fact that it would be able to follow a search algorithm to search out and find a target on its own. The team thus had a whole new design aspect that would be implemented by RADSAT.

In addition to the search algorithm, another opportunity for design presented itself by including an autonomous feature in RADSAT. Namely, the team would now be able to research, design, and test the feature of video processing of the images that were to be captured by the webcam. During its search, RADSAT would need to make its own decision as to whether or not an image captured by its webcam was in fact a target. With video processing now an additional design feature that could be added to RADSAT, the project was beginning to take the shape of a project good enough to be completed during the timeframe allowed by Senior Design.

With the video processing, the team discussed what would be determined by RADSAT to be an “enemy” target. Though many options were discussed, i.e. shape, size, etc., it was determined by the team that RADSAT would search out a specified color as its target. Color recognition then became the next design aspect that was to be researched and implemented by the team in the final design of RADSAT.

With the autonomy of RADSAT, it was next agreed upon by the team that collision avoidance would need to be implemented in its search function so that RADSAT would not be crashing into obstacles and objects every time it searched out a target. With this in mind, it was decided that sensors would be researched and implemented by RADSAT to perform this collision avoidance. Once again, another design aspect that RADSAT now had in its arsenal as a good design project.

Finally, the team decided to add one more function to RADSAT that would add to its overall design. It was agreed upon by the team that RADSAT would follow one more method for issuing commands through voice control. Listening to the human user, the R/C tank would respond to a command that was issued vocally. This, combined with all of the other functions to be designed that were discussed above, was then presented to Dr. Richie as the revamped design for the group’s project.

As can be seen, the meetings and discussions that the group held to determine the overall functionality of RADSAT led the team from a relatively easy and uninteresting project into one that would offer the team many opportunities for design. The above functions were all deemed suitable and reasonable for the team to complete in the implementation of RADSAT. It was at this point that the team continued to research all of the implementable functions, and began writing the documentation for all of the design and testing issues that were to be carried out for RADSAT.

## 4.2 GUI Design

The GUI needed for this project is rather simple and does not need to be exuberant at all. It can be programmed in most object-oriented programming languages. As the team researched many different GUI’s and how they were programmed, it was decided that the C# programming language would be used to program the GUI. An example of what the GUI will look like to control the tank can be seen in *Figure 4.2.1* below.

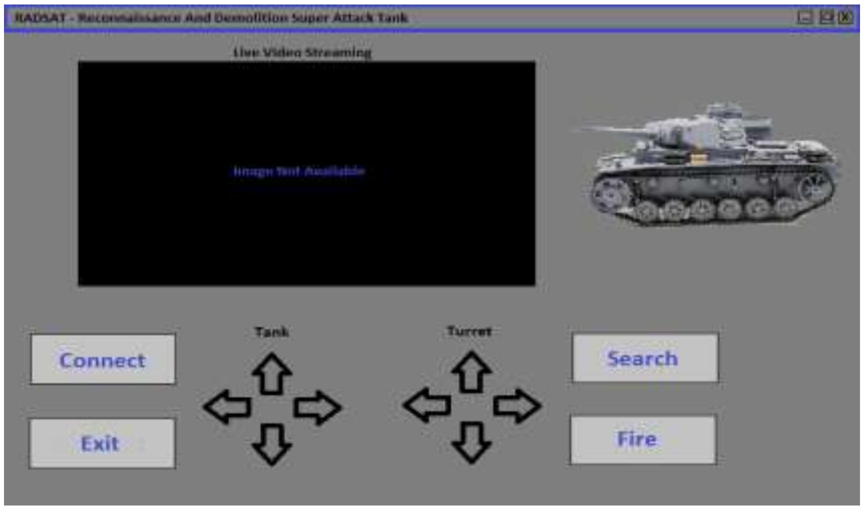


*Figure 4.2.1: GUI Design*

With this GUI, all of the human commands that RADSAT will respond to can be sent to the microcontroller. The “Connect” and “Exit” buttons on the left hand side will respectively establish and break down a connection between RADSAT and the microcontroller. The “Search” button on the right will have RADSAT implement the autonomous search algorithm to search out and locate a specified target. The “Fire” button, also on the right hand side, will be pushed when the target is found and is to be “destroyed”. This command will not be executed by RADSAT autonomously, but only at one of the team member’s discretion.

Two sets of directional arrows were required for the design. The first set of arrows on the left under the title “Tank” will be the arrows that control the movement of RADSAT. The left and right arrows will rotate RADSAT in the specified direction, will the “up” and “down” arrows will move RADSAT forward and backward respectively. The second set of arrows located on the right under the title “Turret” will be used to rotate the camera and turret. The left and right arrows will rotate the camera in the specified directions, will the “up” and “down” arrows will pan the camera upwards towards the ceiling and downwards toward the floor.

Under the “Live Video Streaming” title will be displayed the images processed by the microcontroller from the camera. Images will be processed in real time and displayed on the GUI screen. In this fashion, the user of RADSAT will be able to see what the turret is aimed towards at all time and will be able to make a good judgment call of when RADSAT should fire at the target. A delay in the video display might result in RADSAT firing at an unspecified target, which is an issue that the team would like to avoid at all costs.

## 4.3 Tank Design

It was decided from the start that the tank body or chassis would be bought as a prefabricated part. Building a custom chassis would be time consuming and falls outside the realm of computer or electrical engineering and is therefore somewhat irrelevant to the purpose of this project. Having made the decision to buy a chassis there are two routes one can take. The first is to buy an off-the-shelf RC tank, strip it of its electronics, turret and unnecessary parts, and replace them with our own electronics and turret assembly. The other option is to search for a robotic tank base made especially for robot hobbyists to build onto and build our robot onto it.

If the first option is chosen one of the biggest advantages is that the R/C tank can be disassembled and provide a glimpse into the physical components of the tank were planned and put together. This may help in the construction of the physical aspects of RADSAT by showing how certain problems are resolved. In anticipation of this step in building the tank, attempts will be made to predict design problems and resolve them with hypothetical solutions. Again, this is only in reference to the chassis; electrical and computer engineering design problems are to be addressed thoroughly and intelligently long before the tank is assembled.

Other advantages to using an R/C tank body are that they come in a very large variety of sizes, price ranges, features, and manufacturers. Additionally, there are consumer reviews and videos, which can be useful in determining how high quality the tank is, and what can be expected in terms of capabilities and durability. It is helpful to have an idea of how fast the tank can move and what kind of terrain it can drive over without getting stuck. These tanks come with electronics and salvaging parts such as motors and speed controls could save time and money. Lastly, It could be an advantage or disadvantage, but the visual appearance of RADSAT will be heavily decided by the chassis chosen for it.

A disadvantage to using an R/C tank is the cost of the product. These are in the one-hundred dollar price range and that is incentive to look for an alternate way to obtain a chassis. However, if this option is chosen there is more incentive to get it right the first time, rather than buy a tank and find that it is insufficient for the needs of this project. Adding to this disadvantage is the possibility that advertising could overstate the abilities of the tank or simply make it look bigger or better than it actually is. Also, there is not a wealth of information available about how other people have taken these chassis and modified them to make a custom robot. It has been done, and is certainly possible, but detailed documentation is lacking, and therefore firsthand experience will have to suffice.

On the other hand, a base built especially for robot hobbyists could be a viable alternative. These offer the advantage of having surfaces specifically designed for mounting arduino boards and motors and other necessary robot components.

They are readily available on multiple sites, so finding a low-cost one is possible. People have used these to make simple robots and there are plenty of write-ups and videos demonstrating their uses.

On the down side, perhaps contrary to one might intuitively think, there are not a lot of options to choose from, only three were found that fit the description of the RADSAT project. These bases are between 8 x 4 inches and 9.5 x 9 inches, which is relatively on the small side. A base that is too small could become problematic if the robot is prone to tipping over with the weight of the turret making it top-heavy. The small size may also put limitations on the terrain the tank can traverse, for example sidewalks and short grass will be okay for any tank, but long grass and rocky earth may pose problems for a small tank.

The following table shows tanks that are available for purchase for this project. There is a vast supply of R/C tanks, so a sample was made of the best candidates for this project. These are the first five items in the list. The latter three items are the aforementioned robot chassis. As can be seen, by comparison they are very small. A 1:16 scale R/C tank is best suited to the needs of this project.

| Tank Name | R/C tank (scale) | Price | Size |
|---|---|---:|---|
| SnowLeopard M26 | Y (1:16) | $95.00 | 21 x 9 in. |
| Jagdpanther | Y (1:16) | $113.00 | 21 x 9 in. |
| DAK Pz.Kpfw.IV Ausf.F-1 | Y (1:16) | $89.00 | 22 x 9 in. |
| Heng Long LEOPARD II | Y (1:24) | $50.00 | 16 x 6 in. |
| German Tiger 1 | Y (1:16) | $85.00 | 21 x 9 in. |
| Dagu Rover 5 Chassis | N | $50.00 | 9.5 x 9 in. |
| Boe-Bot Tank Tread Kit | N | $35.00 | 8 x 5 in. |
| Tamiya Track Kit | N | $16.60 | 8 x 4 in. |

*Figure 4.3.1: Abbreviated table of available tank chassis*

Given the project requirements and the available options, it was decided that the SnowLeopard M26 will be purchased for use of its chassis. It is more expensive than some of the others, but it is most attractive because it has a good reputation as a reliable and high quality R/C tank. It has a body that looks like it will be the easiest to work with and it appears to have enough room to fit the necessary parts onto it, with flat surfaces to make mounting easy.

*Figure 4.3.2: The Snow Leopard M26, a stepping stone to greatness*
*With permission from www.bananahobby.com*

## 4.4 – Turret Design

### 4.4.1 – Turret Functionality

This section discusses decisions between continuous rotation servos and 360 degree movement. It also covers possible methods to augment turret direction with the movement of the base. Other options such as fully aiming with base or implementing a wireless firing mechanism are discussed as well. The chosen method will be prototyped and tested. If it fails, the next best choice will be used or a compromise will be made.

In consideration of a continuous rotation servo for the pan function, there are both advantages and disadvantages, limitations and freedoms. A continuous pan would allow the tank to target objects as they circle around it, or as it circles around them, and would allow for search algorithms to simply rotate the turret in one direction while attempting to find and identify a target. The drawback of this system is that wires from the base to the gun and the pan servo would not be able to twist indefinitely. Possible solutions are wireless signals between the turret and base, or brushes that make contact with circular tracks, like the electrical transmission in a slot car track.

For further clarification, see the diagram below. It shows a simplified version of how electrical current can be transferred from the battery on the base, to the turret assembly above, while still allowing the turret assembly to infinitely revolve around the axis through point A.

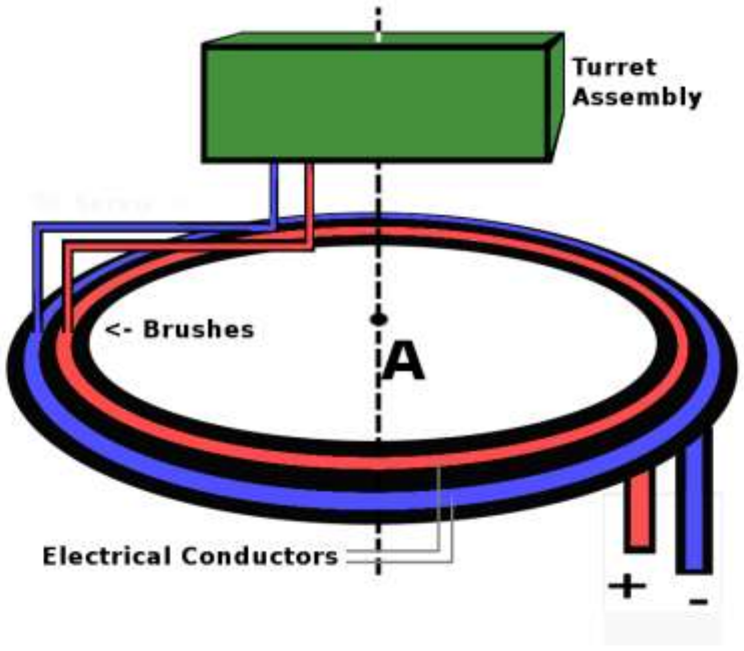


*Figure 4.4.1.1: Rotating Power Supply System with Brushes*

As practical as the diagram makes it look, this approach is less than ideal for a number of reasons. The brushes will experience wear over time and need to be replaced, a bump or jar to the frame may cause momentary open circuit which in turn would cause the servo to behave erratically, and it would simply be a high degree of difficulty to build and implement. This idea must be abandoned in search of more appealing alternatives.

The next choice, if a continuously rotating turret is to be used, is wireless transmission between the base and turret. This would have the same perks as the previous method, but would be simpler in that there would be no need to build an unconventional electrical current transmission system. However, it would also require a power source and wireless receiver on the turret. Because of the added weight, the servos would be under addition strain. Possible drawbacks are sluggish response, broken servos, and loss of communication during electrical storms due to RF interference.

An alternative solution would be to limit the turret rotation to a 360 degree pan. In this case, need for awkward linking between the base and turret is eliminated. It can be simply wired directly to the microcontroller and power source. For this added simplicity, the only sacrifice is that the turret has a limited turning radius and therefore if it needs to turn to a specific coordinate but reaches the limit of its range it must turn all the way back around. For example, in polar coordinates the same direction can be pointed to by rotating 30 degrees or by rotating -330 degrees. The below figure illustrates the example with a possible situation that may arise. In this case, the turret points at a moving target starting at point A and moving to point B, when it reaches 360 degrees, its full rotational capacity, it must turn back and go the long way around.

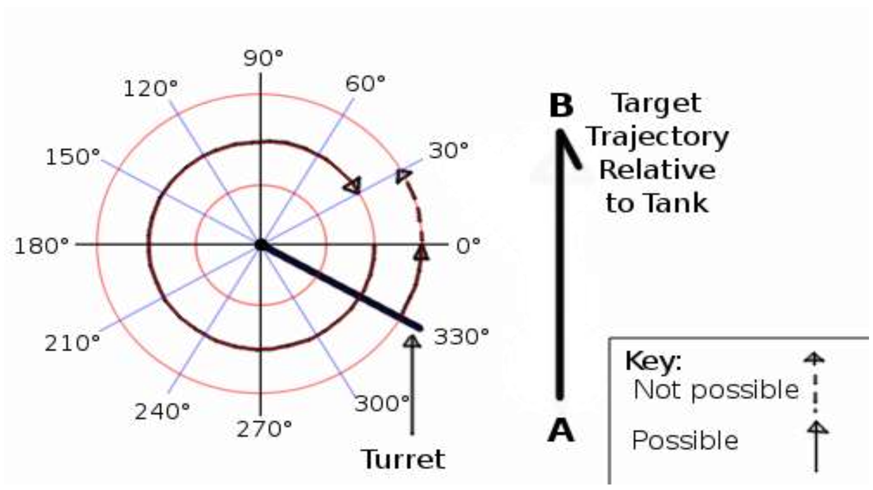


*Figure 4.4.1.2: Original picture used under the Creative Commons Attribution-Share Alike 3.0 Unported. Attribution goes to Mets501.*

This weakness could be highly detrimental to the effectiveness of the tank's attack if the target is sentient and realizes that it can exploit the “blind spot” that the tank now has. More realistically, the problems posed here are loss of time while the turret readjusts, difficulties in coding and refining the targeting system, and waste of energy to make superfluous movements of the turret. Lastly, it might be noted that it simply lacks aesthetic appeal; watching the turret turn all the way back around every time it reaches the end of its range is undesirable.

A possible option to improve the above method would be to augment the rotation of the turret with rotation of the tank base. That is, when the tank is stationary, the base handles lateral tracking, and the 360 degree pan servo is used only when moving. This combination could eliminate the majority of the potentially problematic situations, such as that described above. The case in which the tank is being circled by its target is solved if the tank base rotates in place. The case in which the tank is circling its target is also solved because as it circles, the base turns anyway, and therefore the turret will never reach its “blind spot”. The only remaining case is if the tank were fleeing from or chasing a foe that is moving while circling. This, and similar erratic behaviors, will still be a problem for this configuration, but they are not within the scope of this project.

An alternate choice is to use a servo with 720 degrees of rotation, and enough wire to be able to wrap around the axle of the base servo one full wrap in either direction. In this case the added freedom, along with software designed to keep the servo from turning too far and breaking the wires, can increase the effectiveness without introducing problems. The software also returns the servo to centered or a range near centered periodically to keep it in its most useful position. Despite how fun the other options sound, this option is the one that will be explored and pursued for implementation. It meets the requirements, is inexpensive, and has the fewest complications.

## 4.4.2 The Gun

When deciding on what to use for the tank's gun there are three options: Use the gun it comes with, if it comes with a gun, choose a new gun to buy, or use a already in possession. The problem with using the gun the tank comes with is that it is probably not very high quality. At best, it is unpredictable and therefore not an attractive option. Researching and finding a gun that is well-reviewed and fits the needs of this project is a more attractive option, but this option costs both time and money. The third choice is to use a gun that a team member already owns, and that is the option that is going to be explored as a first choice. The other options have been noted because if the first choice fails at the time of testing, the others will be explored with specific goals in mind, such as less weight or lower current draw.

The gun to be used as first choice for this project is a UHC Steyr mini electric airsoft gun. This decision is favorable because its functionality and quality are immediately observable and testable. Of course it is also a reasonable choice since it was something that we already had, and no money will be spent on it. Pictured below is the gun itself, in fully functional condition despite being approximately seven years old.

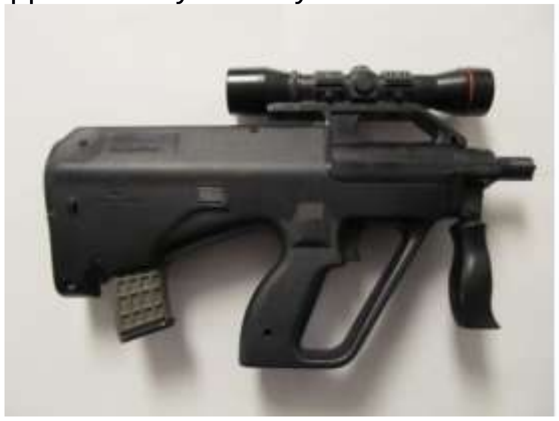


*Figure 4.4.2.1: The Steyr airsoft gun*

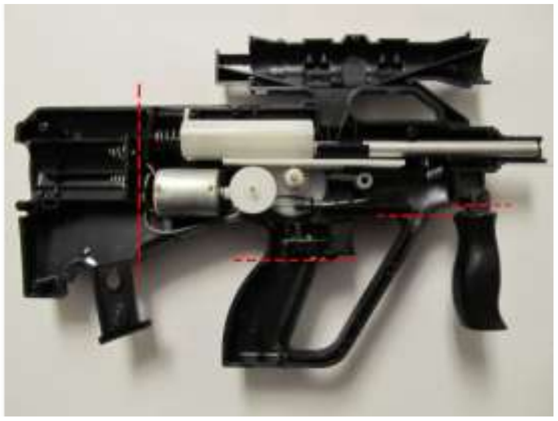


*Figure 4.4.2.2: The red dashes indicate where the gun can be cut to reduce size and weight*

It is also worth noting that there are more advantages than simply being easy to decide on and costing nothing on our budget for this project. This gun is electric powered which means it needs only a six volt potential applied across its input and it will fire; there is no need for CO2 or physical pump mechanisms. It will shoot rapid fire as long as the motor is switched on and it has ammunition in the hopper, which in this case is modeled to look like a scope. It shoots approximately two BB's per second, so the number of rounds fired can be easily controlled. It is also simple and easy to take apart, modify and put back together.

The gun will be mounted directly to the servo's pan/tilt attachment. It will be mounted with two screws through the side of the base of the gun connecting it straight to the top of the tilt servo. The camera will be mounted in a similar fashion beside the gun, on top of the tilt servo. The mass of the gun is 222g, which is used in calculations in section 4.4.3 of the maximum torque the turret assembly will exert on the servo. In the diagram below the photo on the left indicates the approximate center of mass of the gun and the distance to where the gun will be mounted. This distance of 7 centimeters is rounded up to 9 centimeters to be on the safe side and used in the aforementioned calculations.

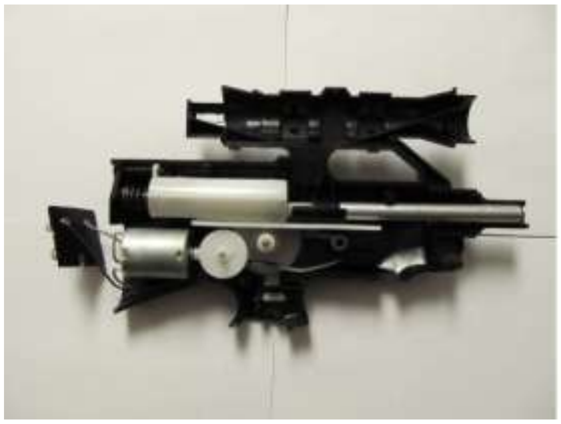


*Figure 4.4.2.3: The inside after the cuts*

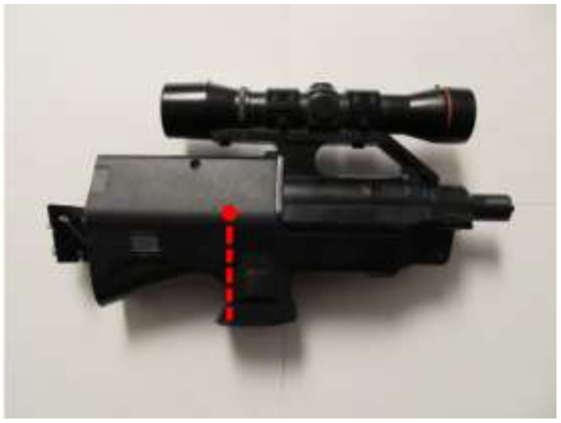


*Figure 4.4.2.4: The reassembled piece*

After reassembling the modified gun it was tested to confirm that it was still able to function as desired and it passed the test by shooting just as before. In it's present state it is small enough that it is not too bulky for the robot to carry around. Also, with its muzzle velocity of 150 FPS and hop-up system, it is powerful enough to shoot targets accurately, within at least a forty foot range. Hop-up is an effect  achieved by a small, built in rubber nub on the ceiling of the inside of the barrel. This system gives the BB backspin, alloying it to fly further before dropping. See figure below:

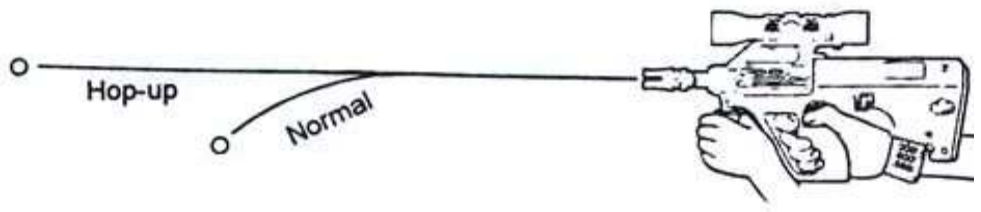


*Figure 4.4.2.5: Illustration from the gun manual showing the contrast between hop-up and normal shots. Reprinted with permission from www.airsplat.com*

This is a very important detail for our design because a concern early on was that target distance would be a problem. It was noticed that the robot will not be able to tell whether an object is small and close or large and far away, and if the object was far away, the BB's would travel toward the object when shot, but fall to the ground before reaching it. Because this gun utilizes hop-up, that problem is less of a problem. It will only come into effect at ranges of approximately sixty feet to seventy feet and beyond. If at the time of testing it is more of a problem than anticipated or it seriously hinders the effectiveness of the tank, solutions will be pursued. Possible solutions are upgrading the existing gun, equipping a more powerful gun, or aiming the turret up to arc the trajectory.

## 4.4.3 - Servo Selection Process

To mount the turret and allow it the full range of motion, two servos are needed. These servos give it two degrees of freedom. The first is the pan servo, it can point the camera and airsoft weapon at anything on the same plane as the tank. The second is the tilt servo, it can point the camera and airsoft weapon at things level with the tank or above it.

The servos can be attached by a specialized attachment device. These attachments are built to connect two standard sized servos to form one pan and tilt unit. One of the two choices below will be used.

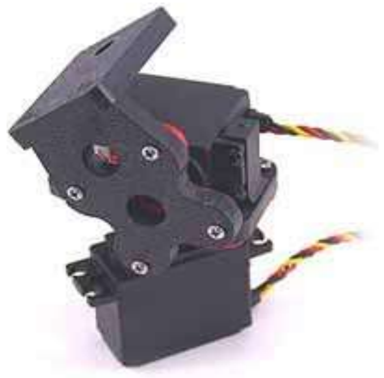
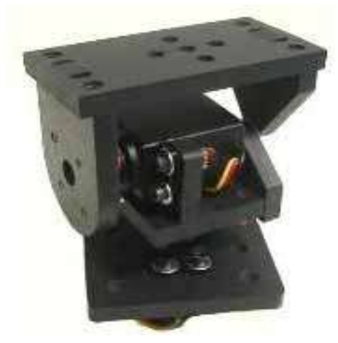



*Figure 4.4.3.1(left): Attachment from servocity*
*Figure 4.4.3.2(right): Attachment from Endurance-Robotics*
*Reprinted with permission from www.servocity.com and www.endurance-robotics.com, respectively.*

Also available are mounts that have both pan and tilt built in. These are manufactured primarily for security cameras and satellite dishes. It is worth noting that these options are available, however, they are prohibitively expensive with the majority being at least one hundred dollars and going up into the tens of thousands of dollars.

Since many standard servos have a range of 60 to 90 degrees in each direction, a special kinda of servo is needed for the pan function. It would not be nearly as useful if the turret could only move slightly to the left or right of the direction the tank is facing. Therefore, the desired servo, often referred to as a continuous rotation servo, was selected from a compiled table of available servos of this type. *See table below:*

| Pan Servo | Speed (rpm) | Torque (Kg*cm) | Cost | Type | Size (inches) | Degrees |
|---|---|---|---|---|---|---|
| EXI B1227 | 46 | 11 | $11.70 | Analog | 2.29x1.10x2.05 | 2880 |
| HS-422 | 62 | 4.1 | $10.00 | Analog | 1.56x0.8x1.64 | ∞ after mod |
| SpringRC S4303 | 70 | 4.8 | $13.00 | Analog | 1.57x0.78x1.35 | ∞ |
| HSR-1425CR | 52 | 3.1 | $17.00 | Digital | 1.59x0.77x1.44 | ∞ |
| HS-635HB | 67 | 6 | $28.00 | Analog | 1.6x0.8x1.5 | ∞ after mod |

*Figure 4.4.3.1: Initial list of eligible pan servos to choose from*

Even though the required value for degrees of rotation is 720°, a servo with infinite rotation will be used because typically servos are either below 360° or unlimited. The HSR-422 by Hitec is the chosen servo because it is produced by a reputable brand and there is support. As noted in the table, this servo will require a modification before it can rotate indefinitely, but this modification is said to be easy and instructions are available. This servo also provides sufficient speed and torque. The price of this servo is also the lowest, which makes it perfect for keeping costs down.

When searching for a good pan servo, the choice between digital and analog servos came up. Because this choice must be considered, a brief discussion on the difference between digital and analog servos is in order. Digital and analog servos are the essentially the same with a few exceptional capabilities of digital servos. First, given the same input, both an analog and digital servo will behave the same. In this way, the two are interchangeable. However, a digital servo holds two heightened abilities over its analog counterpart: it updates at 300Hz as opposed to the 30Hz of an analog servo, meaning it will respond more quickly and have more torque. Also, its direction of rotation, speed, and center can be programmed. The digital servo usually costs more and has higher starting current.

For this project the servos are required to simply rotate as commanded, a simple task not requiring a programmable servo. A digital servo  will work, but is typically more expensive and therefore if an analog servo will suffice, it will be the more appealing of the two. The deciding factor for this project is the issue of torque the servo is capable of handling. The torque on the tilt servo comes from the turret mounted on it. The following calculation estimates this torque.

Torque calculation – This is an estimation of the maximum torque the tilt servo will encounter.

The approximate values that will be used in these calculations are:

| Variable | Value |
|---|---|
| Distance | 0.09m |
| Mass | 0.5kg |
| Acceleration | 9.81 m/s² |

*Figure 4.4.3.2: Estimated values*

| Equation used | Resultant |
|---|---|
| F=ma | 4.905 N |
| T=Fd | 0.44145 N*m |
| 1N*m=10.2kgf*cm | 4.5 kgf*cm |

*Figure 4.4.3.3: Calculated results*

For the end result the desired unit is kgf*cm because this unit is given in servo specifications. The values chosen for this estimation are on the high end so that if a servo meets the requirement of 4.5 kgf*cm torque or higher, even if the estimations are slightly wrong, the servo will still be adequate. In other words, there is room for some error.

There is an almost unlimited number of servos available through online stores, and a table taking a sample of available servos was constructed by finding a sufficient amount of relevant candidates, and listing their attributes. This was the initial table:

| Tilt Servo | Speed (rpm) | Torque (Kg*cm) | Cost | Type | Size (inches) | Degrees |
|---|---|---|---|---|---|---|
| HS-635HB | 67 | 6 | $28.00 | Analog | 1.6x0.8x1.5 | 180 |
| HS-425BB | 62 | 4.1 | $13.00 | Analog | 1.6x0.8x1.6 | 180 |
| Vigor 39g | 53 | 3.2 | $3.80 | Analog | 1.6x0.8x1.7 | 180 |
| Futaba S3003 | 53 | 4.1 | $11.00 | Analog | 1.6x0.8x1.4 | 180 |
| Futaba S3010 | 62 | 6.5 | $25.00 | Analog | 1.6x0.8x1.5 | 180 |
| EXI D123F | 50 | 8.5 | $9.70 | Digital | 1.6x0.8x1.6 | 180 |

*Figure 4.4.3.4: Initial list of possible tilt servos*

After the above table was assembled it became apparent that analog servos are more common, the size requirement for standard sized servos need not be listed, and the number of degrees can be assumed to be 180°. Also, because the first servo has been chosen and it is Hitec brand, it is appealing to choose the second servo to be from the same brand. Given this new information, the table is remade for a more concise, easy to read view.

| Tilt Servo | Speed (rpm) | Torque (Kg*cm) | Cost |
|---|---|---|---|
| HS-311 | 66.7 | 3.7 | $8.00 |
| HS-322HD | 66.7 | 3.7 | $10.00 |
| HS-325HB | 66.7 | 3.7 | $13.00 |
| HS-422 | 62.5 | 4.1 | $10.00 |
| HS-425BB | 62.5 | 4.1 | $13.00 |
| HS-485HB | 55.56 | 6 | $17.00 |
| HS-625MG | 66.7 | 6.8 | $31.50 |
| HS-635HB | 66.7 | 6 | $28.00 |
| HS-645MG | 50 | 9.6 | $31.50 |
| HS-985MG | 76.9 | 12.4 | $70.00 |

*Figure 4.4.3.5: The complete list of standard-sized analog Hitec servos available from servocity.*

It can now be seen plainly that the best choice that meets the torque requirement and has the lowest price is the HS-485HB. This, along with the HS-422 and the endurance robotics attachment will cost $57.00. The entire assembly will be mounted to the tank using screws. The gun and camera will be affixed to the top of the servo attachment and thus will comprise the complete turret.

## 4.5 Video Processing

This section details the process of retrieving an MJPEG video stream with C# code from the wireless security camera's server. This is the basis for the rest of the design sections that use and edit the video stream. This section will be using the open source libraries of AForge.Video as well as follow many of the libraries, and guidelines set forth by the open source video project iSpy. The following is an overview of the functions and their specific roles in acquiring the video feed:

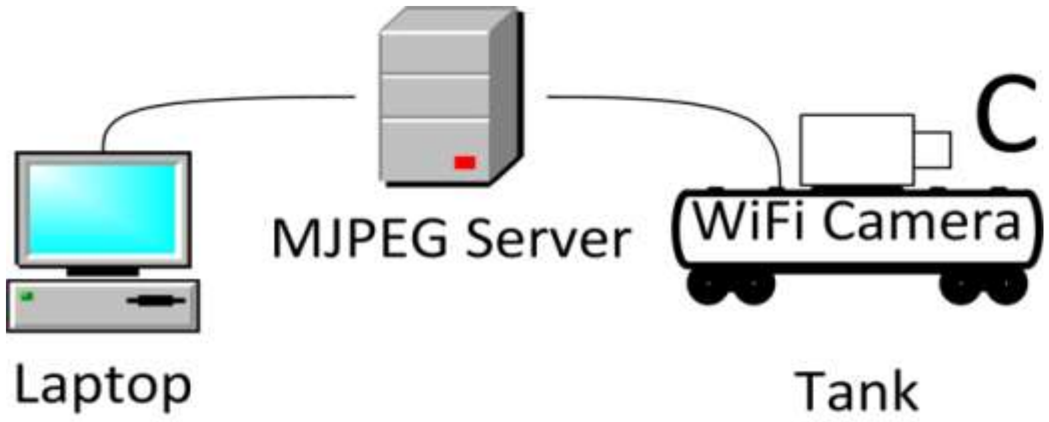


*Figure 4.5.1: Acquiring the video feed*

<u>TestURL()</u> - This function will make sure the program is properly connected to the server that the MJPEG video feed is on. This will accomplish this task by using the existing AForge.Video function, getMJPEG(). If there isn't a video running or there isn't a proper connection established, then the program will display a message such as "Connection could not be established" within the video screen display on the GUI. There is another function within the AForge.Video library, called IsRunning(), which tests whether any data is located on the server. TestURL() will use the isRunning() function to make absolutely sure there is a connection established. If there is no data on the specified server, or if the username or password is incorrect, the software will stop attempting to get data, and will not try again until the user specifies it to retry.

The server location for the security camera in use for the project is http://192.168.1.178/videostream.cgi?rate=0, the username is "admin", the password is "123456" and the SSID the camera connects to is named, "RADSAT".

<u>GetMJPEG()</u> - This function will take place after a successful run of the testURL() function. The function will assume a successful connection is already in place (since testURL() should have confirmed this) then run from there. GetMJPEG() will use the existing AForge.Video class called "MJPEGStream". This function is used exclusively for returning an MJPEG stream as a bitmap image. It has to read in the MJPEG Stream, convert that to a JPEG then convert the JPEG into a Bitmap image. It is important that the JPEG be converted to a Bitmap image, because the Bitmap image breaks the JPEG up into individual pixels, which then allows for manipulation of all the pixels.

Within the GetMJPEG() function, the boundaries for the video feed need to be specified. When specified, the function will create an array of bytes which will be equivalent to the boundary size of the MJPEG stream, which then will be passed to the DisplayMJPEG() function. Once everything is in order, and the stream is confirmed to exist, the program will continue in a while loop until the user specifies it to stop by using a button on the GUI, and the following function.

StopStream() - This function simply uses the SignalToStop() function within the MJPEGStream class. This will signal the threads to stop, and will disconnect from the server. The other purpose of this function is to break the program out of getMJPEG's while loop.

DisplayMJPEG() - This function takes in the bitmap image obtained through the getMJPEG() function. It is responsible for sending the obtained image to every other function which requires an image to edit. The functions include: all functions from section 4.9 Targeting System, as well as the functions from section 4.8 Color Recognition. Once the completed picture is ready, the function will send it to the GUI.

This function will also take advantage of C#'s existing System.Drawing libraries. The System.Drawing libraries have several methods for easily displaying images to a computer screen.

## 4.6 Autonomous Design

### 4.6.1 Supporting Sensors

RADSAT is designed to be able to operate autonomously using only three different types of sensors: the MaxSonic MB1210 ultrasonic sensor, the Sony GP2Y0A02YK infrared sensor, and the Honeywell HMC6352 Compass module. The actual configuration will consist of a total of two GP2Y0A02YK sensors mounted on the sides of RADSAT's body, one MB1210 sensor mounted on the front, and one HMC6352 mounted anywhere, away from magnetic fields. The ultrasonic sensor on the front is used to ensure that RADSAT does not collide with any potential obstructions located near its vicinity. The two infrared sensors will be mounted on each side of the tank and is used to ensure that RADSAT would be able to make informed decisions regarding the direction it should turn. This would effectively prevent it from turning and facing an obstruction. It would not be necessary to mount sensors to detect obstructions behind RADSAT because reversing implies that it has already navigated to its current position successfully.

Calibrating the GP2Y0A02YK is a very necessary step as discussed before because of its nonlinear output in relation to obstruction distance (figure 4.6.1). Calibration is done by placing the sensor at some distance from an obstruction. The distance to the obstruction must be measured at discreet steps of 1 inch.

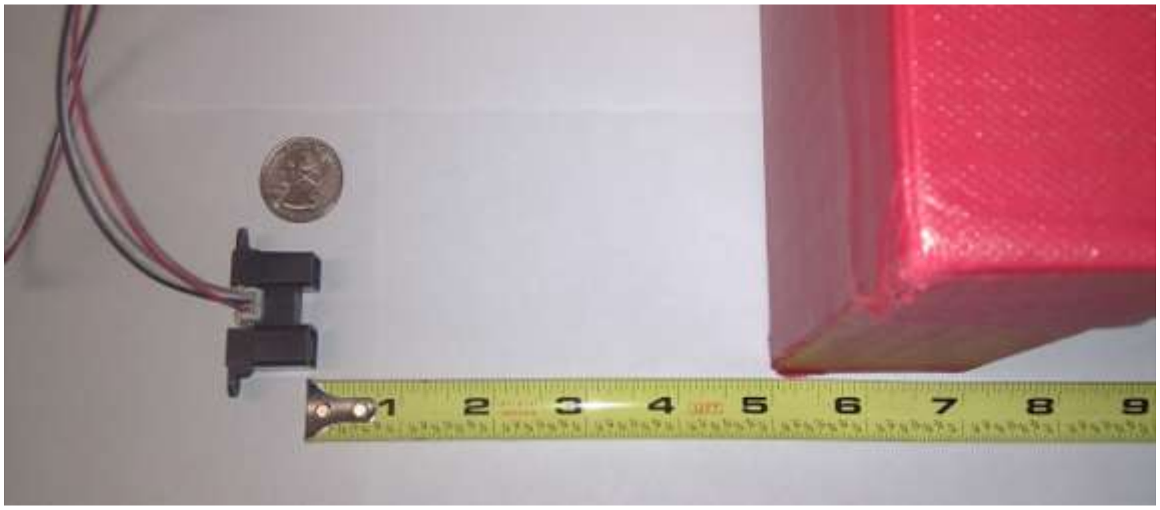


*Figure 4.6.1.1: Calibrating the distance versus output voltage of the Sony GP2Y0A02YK (measuring at an invalid distance).*

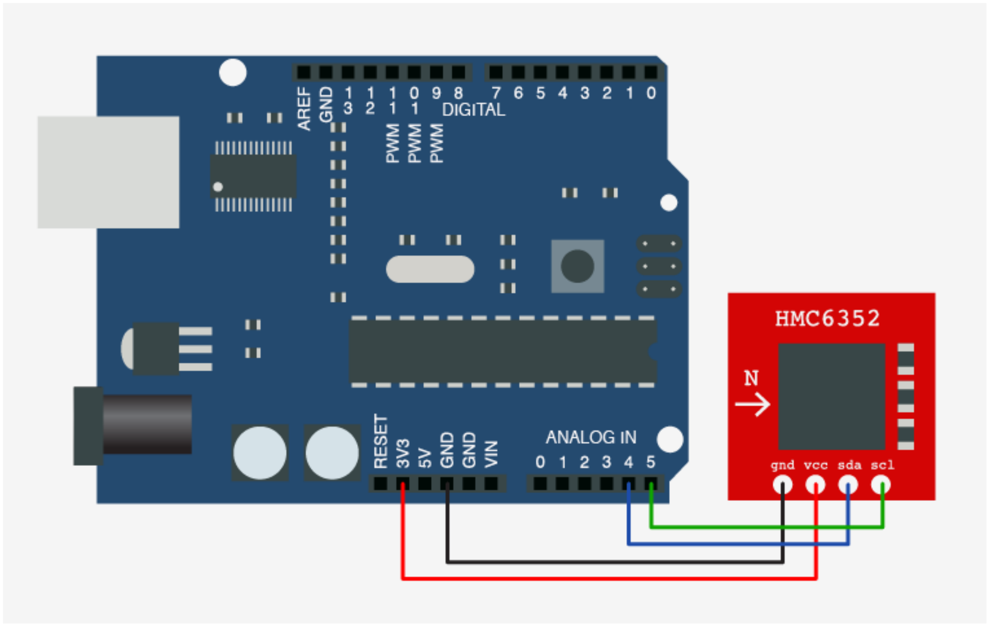


*Figure 4.6.1.2: Diagram of HMC6352 compass module connection to Arduino board, reprinted under license from bilder.org.*

Onboard sensor processing vs. processing on the control software - At this point it is necessary to discuss whether the sensor processing will be handled on board, via the micro-controller, or by the computer. This depends on a wide range of factors. These include, cost, difficulty, speed, capabilities, bandwidth, number of sensors. According to the project requirements the speed at which the robot can respond to the sensors is of high importance. Even more important is the accuracy and reliability; in a word, effectiveness. These factors are primary, and factors such as cost, ease of execution and aesthetics are secondary. A decision will be made based on what information can be gathered before the prototype is built.

The parallel for this decision is the human brain. In humans the cerebral cortex processes deep thoughts and can handle complicated situations requiring reasoning, decisions, judgments and planning. The medulla oblongata, on the other hand, is a more basic part of the brain and handles unconscious movements, reflexes, and reactions such as making the heart beat, controlling the muscles used to sneeze, and blinking when something gets near to the eye.

In many ways, RADSAT should process the sensor input like the medulla, automatically producing one preset action for a given stimulus. This is a starting point for this discussion. After reviewing the facts, the decision will be made whether to modify this hypothesis.

The four sensors together provide the following information: The compass will provide directional orientation which will be useful for determining where the tank has been and is going to be. The two infrared sensors are useful for sensing if the side of the tank is approaching or close to a wall or obstruction. Finally, the ultrasonic sensor provides information as to what is in front of the vehicle. These give the tank spatial relation to its immediate surroundings.

For a given sensor to relay information to the computer or to the microcontroller it takes such a short time it is almost instant, and therefore, both about the same. Thus, reaction time is not an issue. The difficulty of programming the microcontroller to give a response to a sensor input is not high. If, for example, the data from the sensors was to be recorded and used to construct a virtual map for memorization of obstacle location, then it would need to be relayed to the computer. Because that is not within the scope of this project, handling sensor feedback will be done on the microcontroller. It is the best, easiest and most reasonable way. This is discussed further in 4.6.3 Microcontroller Design.

## 4.6.2 Autonomous Design – C# Control Functions

The Aruino Diamondback (the microcontroller which is being used for the RADSAT) is capable of receiving ASCII characters. With this knowledge it was decided to give each individual control function of the RC tank a different command protocol. Each command protocol is listed within table 4.6.2.1 located below.

To be able to communicate with the Arduino Diamondback with C#, a dedicated static IP needs to be assigned to the board. The default IP address the Arduino connects to is 192.168.3.177, and since only the laptop, the security camera, and the Arduino board will be connected to this network, there is no reason to change it. Now the C# code will be connected to the same address and send the necessary data to the Arduino, which will be constantly checking for new data via a while loop. The following bulleted list shows the different functions within C# that will control each individual motion of the RC tank. The function names should be self-explanatory for which motion it will control. The command protocol in table 4.6.2.1 will be used when each function sends data to the robot.

- LeftForward()
- LeftBackward()
- RightForward()
- RightBackward()
- BothForward()
- BothBackward()
- TurretRight()
- TurretLeft()
- TurretUp()
- TurretDown()
- ShootTurret()

| **Motion** | **Command Protocol** |
|---|---|
| Stand-By | 01 |
| Left Forward | 02 |
| Left Backward | 03 |
| Right Forward | 04 |
| Right Backward | 05 |
| Both Forward | 06 |
| Both Backward | 07 |
| Turret Turn Right | 08 |
| Turret Turn Left | 09 |
| Turret Go Up | 10 |
| Turret Go Down | 11 |
| Shoot the Turret | 12 |

*Table 4.6.2.1: Protocol Information*

## 4.6.3 Microcontroller Code

This section details the code which will be embedded on the RADSAT's microcontroller. Initially, this code is written for the Arduino Diamondback, however, the code will be altered at a later stage to accommodate the XBEE and PCB layout. This code explains the overview of how the Arduino Diamondback will communicate with C# code, as well as how it will communicate with the RADSAT's servos and chassis.

All Microcontroller Code will be written within the Arduino 1.0 IDE. This IDE is specifically built to cater to all Arduino boards. This allows for easy integration between the laptop and board, as the Arduino 1.0 IDE is able to quickly and efficiently load data onto the board. It also color codes .INO files, which not too many IDE's are compatible with, and makes the code much easier to read. Although Eclipse can become compatible with a plug-in, without extensive knowledge about Ardiuno, and for programming an Arduino board, going with the IDE specifically dedicated to Arduino seemed like the best bet.

The connection the microcontroller will be using will be a similar set-up as the WiFi Security camera. It will be using the same Netgear WGR614 Router, with the same SSID of "RADSAT". The IP address of the WiShield (Diamondback WiFi module) /XBEE will be 192.168.3.177, the router's IP is 192.168.1.1, and the subnet mask is the default 255.255.255.0. The Netgear WGR614 Router will not have a connection password. The reason no connection password was chosen is because the board would take at least thirty seconds to create a connection with a password protected router. However, an unprotected router can obtain a connection almost instantaneously.

The following is what the code used to connect to the router and its own IP address will look like.

```
unsigned char local_ip[] = {
  192,168,3,177};    // IP address of WiShield
unsigned char gateway_ip[] = {
  192,168,1,1};      // router or gateway IP address
unsigned char subnet_mask[] = {
  255,255,255,0};    // subnet mask for the local network
const prog_char ssid[] PROGMEM = {
  "RADSAT"};
unsigned char security_type = 0; // 0 - open; 1 - WEP; 2 - WPA; 3 - WPA2
```

The code is done in the C programming language. And the three variables listed above (local_ip, gateway_ip, subnet_mask, ssid, security_type) are used within the library "WiServer.h", which is catered to the RADSAT project's specific information. The functions within WiServer.h will know what to do with this information and connect the RADSAT accordingly.

After all the above code is entered along with the rest of the code within the server.INO program, the board will be constantly checking to see whether there's any information it needs to read that is located on the server. all code within the void loop() function needs to run endlessly on the WiShield, otherwise there will be no connection between the RADSAT and the laptop. The code will be constantly connecting to the server, and will search the file motion.txt, for any updates on movement commands. The protocol for the movement commands can be seen in Table 4.6.2.1 within the 4.6.2 Autonomous Design – C# Control Functions section.

```
void loop() {
        //The following line will grab any new commands for the robot
        GETrequest  getControl(ip, 5000, "localhost", "/motion.txt");
        //Following the GETrequest there will be several IF, SWITCH
        statements which will try to //determine which command was
        issued, and then it will start the appropriate //functions
        WiServer.server_task();}
```

Within the IF and SWITCH statements, following the "GETrequest", depending on what is written on motion.txt, the code will tell the different digital I/O pins to send a signal to the tank. All pins will use the following code, so that they will be considered ouput pins, pinMode(pinumber, OUTPUT);, where pinumber represents pins 1-14. The command to send the signal will be, digitalWrite(pinumber,HIGH);, where this time pinumber represents whichever pin was selected to interact with the RADSAT chassis.

For the I/O pins which are connected to servos, the command will be different. The servo code will be using the SoftwareServo library. Since the servos operate with pulses instead of voltages, the code will need to reflect that. The first thing that needs to be accomplished is establishing the pulse rate for the servo. Following that, certain commands are used in order attach and detach a pin from driving a servo. SoftwareServe variable, serves as a new servo within the SoftwareServo library. The servo code will be written as follows:

```
SoftwareServo servo1;

void setup() {
        servo1.attach(pinnumber); //makes a pin become a pin dedicated to
servos
        servo1.setMaximumPulse(2200);} //sets the max pulse to 2200
microseconds

void loop() {
        //various IF, SWITCH statements
        digitalWrite(pinumber, HIGH);} //sets the attached pin to "on"
```

The analog input pins (pins 4 and 5) on the Diamondback will be reserved for a compass sensor. The compass sensor will enable the RADSAT to acknowledge where it is at all times. This will be useful when it is autonomously trying to find its location. The compass will enable the RADSAT to rotate certain degrees, which will allow for easy navigation. The compass sensor will be using the Wire.h header and will return a float value telling the degrees the RADSAT is facing. The following is an example of what the code will look like:

```
void loop() {
                Wire.beginTransmission(HMC6352SlaveAddress);
        Wire.send(HMC6352ReadAddress);      // This command gets the
        data
        Wire.endTransmission();
        byte MSB = Wire.receive();       //received in binary format
        byte LSB = Wire.receive()
        float headingSum = (MSB << 8) + LSB; //converts to float
        float headingInt = headingSum / 10; }
```

There's one more part which needs to be accounted for within the microcontroller code; the analog inputs for the sensors. This is necessary, so the RADSAT will not run into any objects or walls, thus creating a smarter robot. The whole code segment within the function, void loop(), from all the above code, will be surrounded by an ELSE statement. The IF statement to go with the ELSE statement will check to see if any of the sensors have noticed anything. If they have, then their reading will override any commands that were given on the server, from the laptop. The statement will check to see whether any of the analog pins are receiving an input voltage, and if they are, it will know there is an obstacle in its way. Each sensor will take a different action within the code when it goes off. This is because the sensors are located at different locations on the RADSAT, and will require different motions accordingly. The code will use the function analogRead(pinumber) to check the sensors, and it returns an integer which ranges from 0 to 1023 (0V to 5V). When inside each IF statement, the program will issue commands to move the RADSAT using timer commands. For example, if the sensor on the right side goes off, the RADSAT might be commanded to move left-backward and right-forward for 2 seconds, then be asked to move forward until disrupted again. The following is an example of what the code will look like:

```
void loop() {
        if(analogRead(sensor1) > 500 || analogRead(sensor2) > 500 ||
        analogRead(sensor3) > 500){
                if(analogRead(sensor1) > 500){
                //commands to move the RADSAT according to sensor1
                }
                if(analogRead(sensor2) > 500){
```

```
            //commands to move the RADSAT according to sensor2
             }
            if(analogRead(sensor3) > 500){
            //commands to move the RADSAT according to sensor3
            }}
      else{
            //all previous code
          }}
```

This is the general layout of the code which the microcontroller will have loaded onto it, and the 500s represents the voltages sent back from the sensors. Of course however, the many of the commented sections will have actual working code within them. The code will also have fluid variables, which will make for a much easier interpretation of exactly what is going on within the code.

## 4.7 Power Allocation

The requirements for the power supply are that it is rechargeable or renewable, can sufficiently supply the power needs of all the electronics on board the tank, and last at least thirty minutes before being drained. In general, the simplest solution is a rechargeable battery pack such as those used in radio controlled cars, thus this option is more appealing than solar power or other less conventional sources. Solar power would also hinder indoor operation and complicate power distribution as well as being a weaker source.

The battery needs to supply power to the PCB, the motors that move the tank, the servos that aim the turret, the gun, the camera, and the sensors. The tank motors, servos, and gun each require 6 volts. The PCB needs 3.4 volts, the camera needs 5 volts and the sensors each need 5 volts. This means that the battery has to have a high capacity and be able to hold up to the high amount of current that will be drawn from it. It also means that regulators will be necessary to ensure voltages and currents through the various components stay within recommended operating range.

Another option is multiple power supplies. At most, two should be used. One would be to power the servos, motors and gun; and the other to power the PCB, sensors and camera. There are two advantages that this presents. First, if one battery dies everything will not shut down but some systems will remain functional  and take measures to avoid problems. Second, having two separate systems means each system will be less complicated. The reason having two batteries is not as appealing is that it is simply easier for the used to only have one battery to charge.

## 4.7.1 Power Routing

To regulate the voltage to each component and avoid inefficiency, a switched voltage regulator is needed. For this project the LM576 was chosen. The main reason for choosing it is that there is some familiarity with it due to the EEL 4309 lab. It is the right switched voltage regulator for this project because its output voltage can be adjusted, it requires simple wiring, and it can be switched on and off by supplying a voltage to the ON/OFF pin. When the voltage is less than 1.6 volts it is ON and when the voltage is greater than 3.3 volts it is OFF. These signals will be sent from the outputs of the microcontroller. It will be necessary to have this function apply to the motors to control when the tank starts, stops and turns. It will also be necessary to control when the gun fires. The figure below shows the wiring diagram for the switched voltage regulators with the power supply on the right and the load on the left.

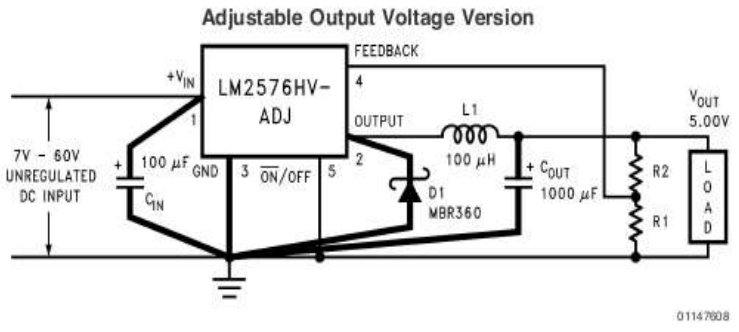


*Figure 4.7.1.1: The wiring diagram for the LM2576.*

The output is grounded and ON by default and can be turned OFF by applying a voltage greater than 3.3 volts to pin five. This will be done by the microcontroller sending a 5 volt output to that pin. The resistors R1 and R2 can be selected to produce the desired output voltage. The equation R2 = R1(Vout/Vref – 1) can be used with Vref = 1.23V and R1 = 1kΩ to find R2 for each desired Vout as seen in the table below:

| R1 | R2 | Vout |
|---|---|---|
| 1000 Ω | 1764 Ω | 3.4 V |
| 1000 Ω | 3065 Ω | 5 V |
| 1000 Ω | 3878 Ω | 6 V |

*Figure 4.7.1.2: The resistor values for each voltage*

One LM2576 will be used for each motor and the gun and turned off and on as needed. Another one will be used for the camera, another for the PCB input and one more for the servos and sensors. Those components that share a voltage and are always on will be wired in parallel and share a regulator. To run the motors backwards a double pole double throw relay was used. When it is given a signal from the microcontroller it switches the positive and negative inputs to the motor. An outline of the wiring diagram as a whole is shown below, after it is the diagram of the relay:

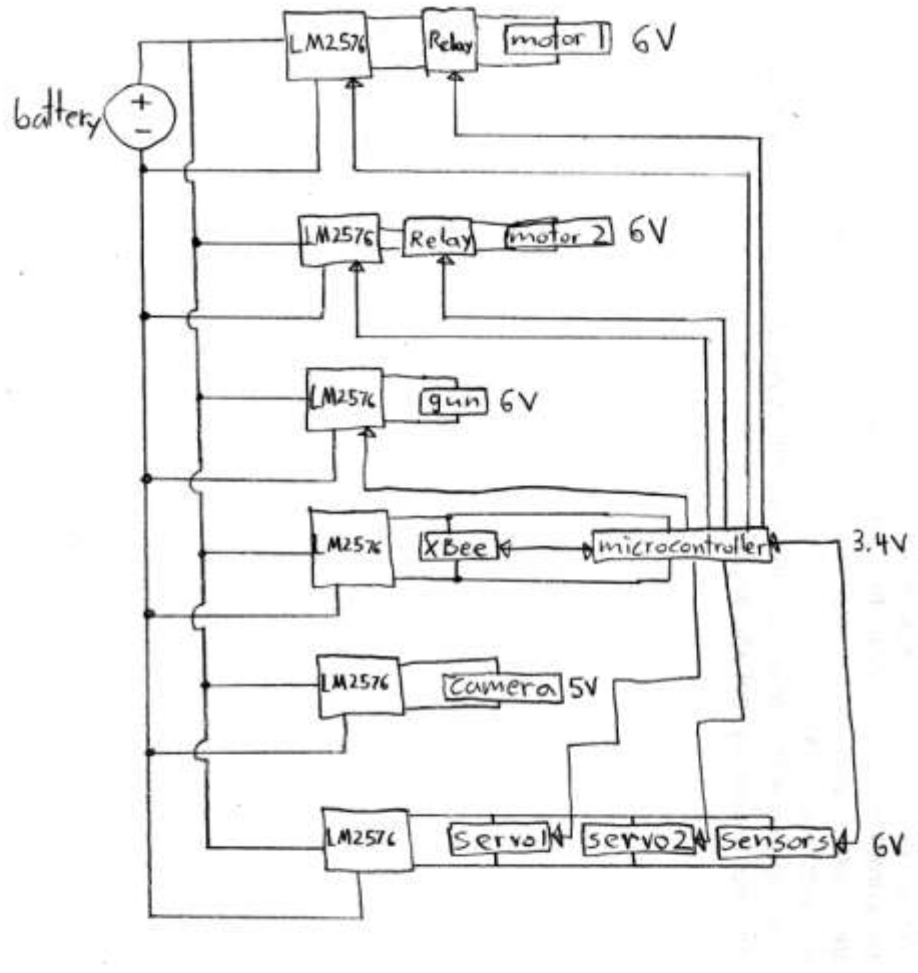


*Figure 4.7.1.3: Wiring diagram for all components*

In the next figure one of the relays from the above figure is shown in detail. The line labeled +V is the voltage for the motor. The line labeled signal will be the line from the microcontroller. It will have no voltage across it when the tank's motors need to move it forward and a 5 volt voltage across it for backwards movement. The diagram shows a dotted line for the 8 pin relay with the pin locations as seen from above. When the two switches in the middle flip to the inner nodes, the polarity of the voltage across the motor is reversed.

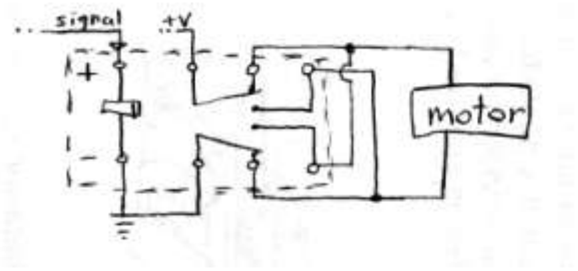


*Figure 4.7.1.4: Wiring diagram for relay*

## 4.7.2 Power Source

As alluded to above, the power source will not be a solar panel or anything special; it will be a battery. According to the specifications for this project the battery needs to be at least 6 volts, have a 2,000 mAh capacity, and supply power for 30 minutes. There are many choices for the type of battery, but because a rechargeable battery is desired, the most interesting options are LiPO, Ni-Cd, and Ni-MH.

Lithium-poly R/C car batteries seem to have the best reputation for serious applications. Some of the reasons are that they have high capacities, are lightweight, and have a high discharge rate. However, after further investigation it was found that they are more delicate when it comes to charging and discharging. They can be damaged by over-discharging, over-charging and being dropped or taking a blow. The main problems arising from these kinds of damage are shortened battery life and fire.

Nickel-Cadmium rechargeable batteries are more resilient and will last for more charge cycles than LiPO batteries. They can also be fully discharged without the adverse effect of acquiring a memory. Ni-Cds have a relatively high energy density and discharge rate compared to lead-acid batteries. These batteries are the oldest technology of the three discussed here. New types of batteries have been developed that are less expensive.

Nickel-metal hydride batteries are notable for their high capacity, ease of use and low price. These batteries require a smart charger, but this is a good thing as it will prevent over-charging. These batteries are common and available in a wide range of voltages and capacities, from numerous sources. Because they are easy to find, easy to use and inexpensive these batteries will be used for this project.

The battery that the tank comes with is a 7.2 volts 1700 mAh Ni-Cd, but it is lower capacity, and according to the tank's spec page it runs for 20 minutes. A power source must be bought rather than using the battery that comes with the tank, but that battery can be used as a backup. Two new 7.2 volt 3800 mAh NiMH batteries and a charger will be purchased. Despite costing approximately $60.00 this will provide redundancy so that any crisis that may arise from a battery being lost, damaged or discharged may be averted.

## 4.8 Color Recognition

One of RADSAT's distinguishing features is the ability to recognize different colors which appear in its surroundings. This feature will be used to discover the targets that the RADSAT will shoot at. Using several functions which manipulate the MJPEG stream, the RADSAT will be able to distinguish objects by color. The following are the functions within the Color class:

ColorRecognition() – This function takes in a Bitmap image, as well as a string which represents the target color and looks at all the pixels within the image in a systematic way. This function will use multiple threads which will check different sections of the image at the same time. The first thread will be checking every pixel in the upper-right quadrant, the second thread will be checking every pixel in the upper-left quadrant, the third thread will be checking every pixel in the bottom left quadrant, and the fourth thread will be checking every pixel in the bottom right quadrant. With all of these threads working together, the program should work far more efficiently, and since the ColorRecognition() function will surely take the most processing power and would involuntarily be laggy, this technique will greatly reduce lag-time.

While the function is cycling through every pixel, it will use a fairly simple equation to check whether the pixel is the appropriate color. First of all, there will be three IF statements, all checking whether the user specified the input to be red, blue, or green. As an example, if the user selected green, there will be an equation which will check the color as follows. D = sqrt((R - 0)^2 + (G - 255)^2 + (B - 0)^2), where R represents red within the RGB color scheme, G represents green, and B represents blue. The result D will be compared against a variable. That variable determines how far from pure green the software will allow to still be considered green. The value which will be used will most likely be 70, and the variable will be called "threshold". The camera which will be used for the RADSAT doesn't have perfect color imaging. Therefore, a fairly large threshold value will be used. Having a threshold of 70 will allow a broader range of colors which will be viewed as the specified color. For example, if a pixel has RGB values of R=5, G=250, and B=5, the D value will be 8.66 which easily clears the threshold. Another example, where R=21, G=234, and B=21, then the D value will be 36.37 which also clears the threshold. However, an RGB value of R=41, G=214, and B=41 will break the threshold with a D value of 71.01.

The ColorRecognition() function will keep track of all pixels that meet the color specified. It will do this by keeping an array which lists all of the qualified pixel's coordinates. This array will later be used in the ColorArea() function which will be explained later in this section.  Also if a certain pixel does not meet the color specification then that pixel will be sent to the RobotVision() function, also to be explained later in this section, as long as the checkbox for Robot Vision is checked.

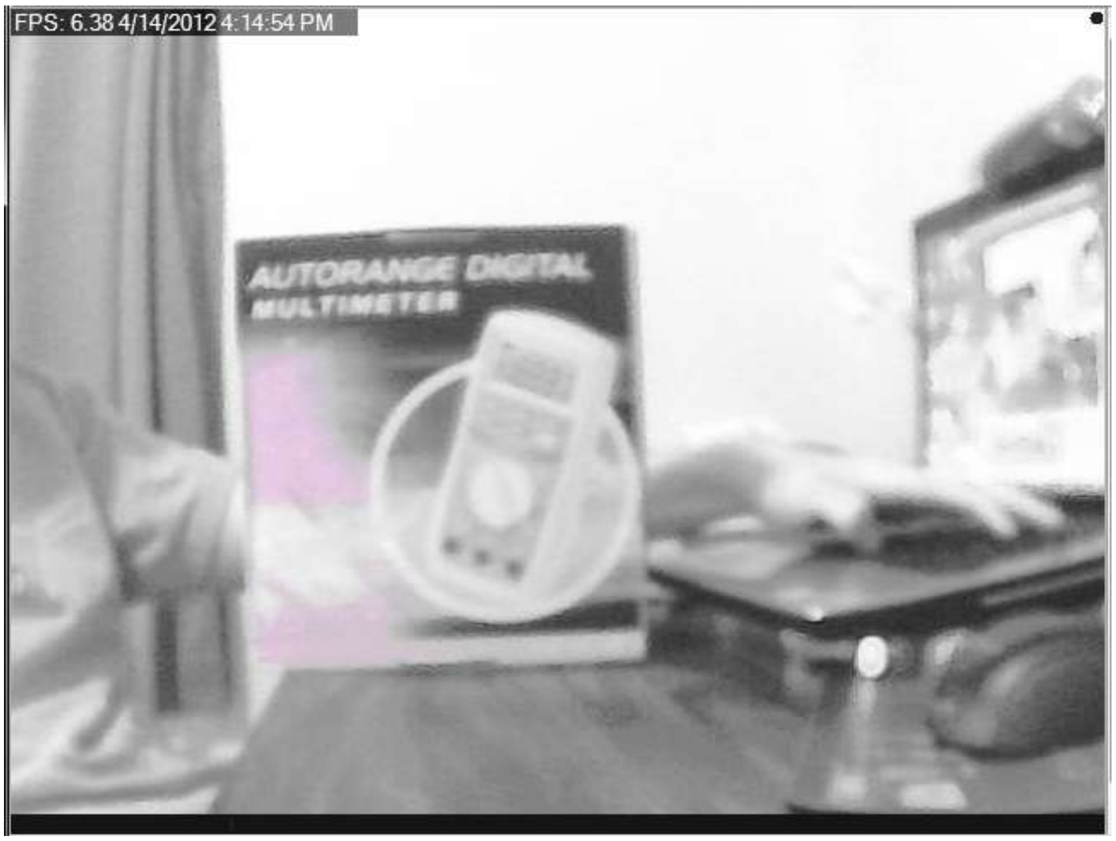


*Figure 4.8.1: Robot Vision*

<u>RobotVisionMode()</u> - Figure 4.8.1 depicts an example view of the GUI when the “Robot Vision” checkbox is filled, and the color “Red” has been input for the RADSAT to target.  This feature allows the user to see what the RADSAT is viewing, giving them a good idea of what targets are available.  The RobotVision() function will be called from the ColorRecognition() function, and will turn any non-specified-color pixel gray.  The equation to convert a color pixel into a gray pixel is actually very simple.  The equation is N = (R + G + B) / 3, which again is based on the RGB color scale.  The value obtained “N” will be the new value for R, G, and B. Since ColorRecognition() will send only pixels which do not meet the specified color requirements every pixel which enters this function will be safe to convert.  This function will directly alter the Bitmap image which ColorRecognition() is checking.

<u>ColorArea()</u> – This function will act as the link between the Color class and the Target class.  It finds the center of any clump of pixels which are the same color. This will first find the border for the area, reconfigure the area so it becomes a rectangle, then return the exact center of the rectangled area.  This will be easy

to accomplish, as the height and width of the rectangle will be recorded and the center will simply be located at H/2, W/2.

## 4.9 Targeting System

The targeting system for the RADSAT will be implemented as so the RADSAT will shoot in the dead center of the target. This will be done after extensive testing to see the trajectory of the air-soft rifle. Once the trajectory of the air-soft rifle has been discovered, this class will serve to aim, and reposition the RADSAT in order to achieve a perfect shot.

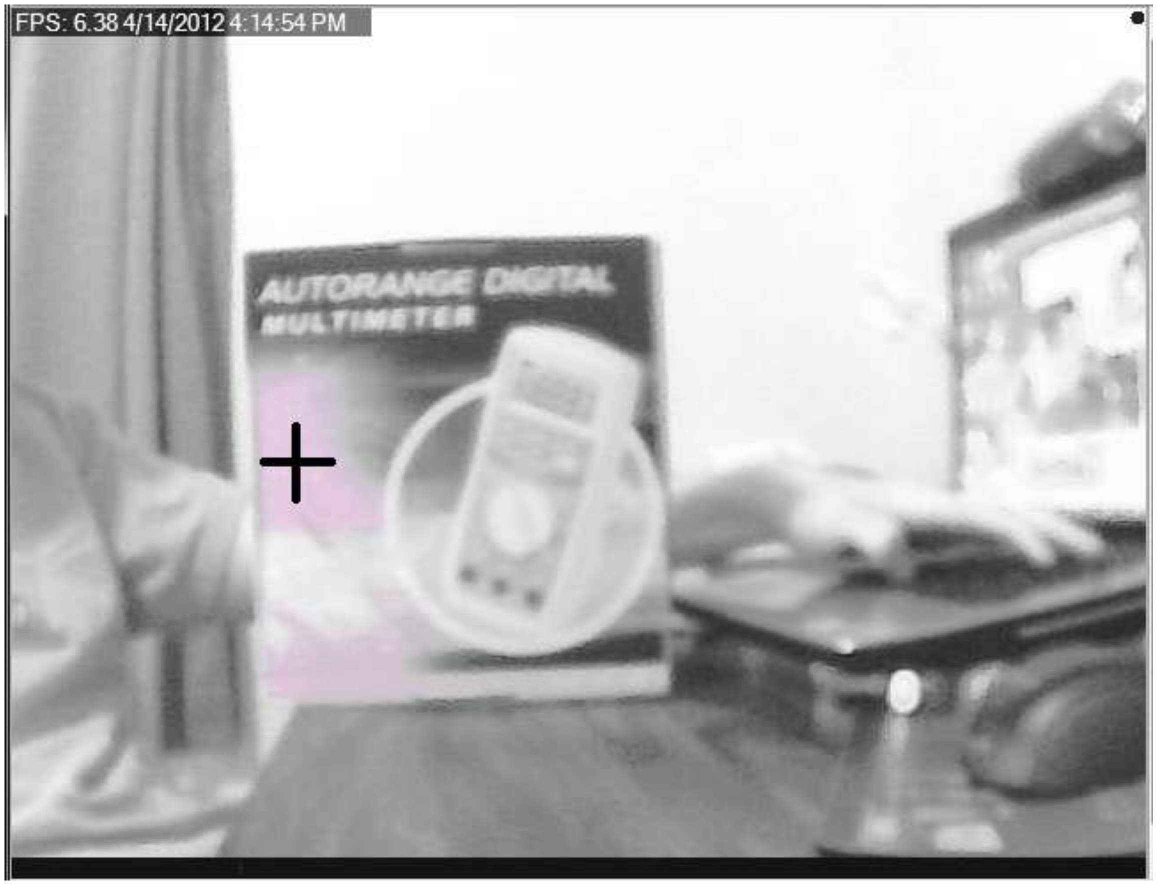


*Figure 4.9.1: Crosshair*

CreateCrosshairs() – This function takes in the center of a target’s area obtained from ColorArea(), from 4.8 Color Recognition. What it does is change a number of pixel’s color, shaped as a crosshair, around the center of the target’s area and colors them black, which displays to the user, a crosshair in the center of the possible target. This will allow the user to know exactly what the program is doing. The crosshairs will always update, and will be within the center of the target, no matter how much the RADSAT might move.

LockTarget() – The program will sit idly by until a command to either lock the target or continue searching has been issued. Once the target has been commanded to lock, either by voice command, or by a GUI command, this function will tell the program to progress and the current target (the one with a crosshair) will be set to be the followed target.

FollowTarget() – This function will reposition the RADSAT so the current target's crosshair appears in the desired location, so the air-soft rifle can target it. This function will try to move the center pixel to the desired location. What it will do is draw two invisible lines, one through the vertical axis of the desired location, and one through the horizontal axis of the desired location. It will then adjust the RADSAT using the commands from 4.6.2 Autonomous Design - C# Control Functions, so the center pixel meets with the vertical axis of the desired location. It will achieve this with horizontal motion from the turret, or from rotating the base of the RADSAT. Following that, the function will adjust so that the center pixel meets with the horizontal axis of the desired location, while vertical axis still stays at the desired location. Once these steps have been achieved, the RADSAT will be in prime position for shooting the target.

Fire() – This function is pretty self explanatory. It will be the "executioner" so to speak for the target. Since the FollowTarget() function should have the air-soft gun in the correct position so that it can be hit, all this function has to do is initiate the attack. It will use the ShootTurret() function from 4.6.2 Autonomous Design – C# Control Functions, for the initiation, and of course it will not shoot until a command has been given from either voice command or from the GUI.

## 4.10 Vocal Command Design

As specified in the requirements and specifications RADSAT is able to accept a set of at least 10 spoken words and phrases in order to successfully carry out its mission. These words are outlined in table 4.11.

| | |
|---|---|
| Stop | RADSAT is given a stop command. All movement and tracking must stop. |
| Find | RADSAT is given a go command. Movement and tracking is restarted. |
| Red | Finds a red target. |
| Green | Finds a green target. |
| Blue | Finds a blue target. |
| Yellow | Finds a yellow target. |
| Black | Finds a black target. |
| Forward | Moves RADSAT forward, only if there are no obstructions present. |

| | |
|---|---|
| Fire | Fires at a target, once it has been acquired. |
| Left | Rotates RADSAT 90 degrees counterclockwise. |
| Right | Rotates RADSAT 90 degrees clockwise. |

*Table 4.11: Minimum list of words and phrases RADSAT will be able to recognize.*

RADSAT's voice module is designed to be written in the C program language using the Port Audio library. The library helps with sampling and device interfacing, but contains no code for any voice or word recognition functionality. To implement such functionality, as discussed previously in this document, there is a need to create a dictionary and to fill it with prerecorded verified commands that incoming streams of sound will be compared with. These functions are listed below. The most important aspect of

AppendDictionary(string s) - This function's primary goal is to create the dictionary as described above. It accepts a single string parameter, s, and will create a file whose name is the value of s. This file's format will be a series of floating point values.

GetStream() - This function will open Port Audio and record an incoming stream. It will sample and store the incoming audio stream in a struct whose values include and array containing its length and an a sample of the floating point values returned by PortAudio.

GetCommand(struct s) - The GetCommand function will accept the struct produced by GetStream(). It will then compare the values in the struct's array to the dictionary it produced using AppendDictionary(s). It also outputs the command to stdout so that its value can be read by its master process.

## 4.11 Wireless Communication Design

During the research stage it was decided upon by the team that an Arduino Diamondback prototyping board with an ATmega328P microcontroller would be used in the initial design of the RADSAT, and a self-developed PCB will be implemented in the final design. The Diamondback has the capability of establishing a connection with a laptop through a WiFi Shield implemented on the board. The WiFi capabilities of the Diamondback communicating with any WiFi enabled laptop was the main reason this prototyping board was chosen, as well as the additional design opportunities that it will allow the team when it comes to designing RADSAT. Using this prototyping board and WiFi Shield, it will be necessary to look at how the PCB will be able to wirelessly connect to the RADSAT in the final design.

Set-up for the Diamondback is rather simple. First, the “arduino.exe” program will be installed and a server will be set up with the IP address, default gateway, and subnet mask of our WiFi Shield. Once this is done, it will be downloaded to the microcontroller via a USB connection. After this has been accomplished, the connection of the microcontroller and the laptop can be tested to ensure that a connection has been properly established. This set-up process can be done on any computer that would be used to control RADSAT's movement and firing capabilities.

Once the connection is established, software developed by the team will be used to send packets of information from the laptop to the microcontroller on the Diamondback through our GUI. These packets will follow TCP/IP protocol for delivery. Free software is provided for the Arduino board with functions that communicate with the WiFi Shield. These functions can be research and utilized by the team in order to write the program that will ultimately control all of RADSAT's functions.

Commands that will be sent over the WiFi connection include the following:

- Tank: Forward, backward, left, right – The RADSAT will move in the specified direction
- Turret: Up, down, left, right – The turret and camera will pan in the specified direction
- Search – the RADSAT will autonomously search out a target
- Fire – the RADSAT will fire at the specified target
- Connect – Establishes a connection
- Exit – breaks the connection

These commands will each have their own unique packet sequence that will be recognized by the microcontroller. Immediately upon receiving the command, the RADSAT will carry out the specified function.

In the final design of RADSAT, the wireless commands that the laptop will send to the R/C tank will be handled by a server created by the laptop for this purpose. The server that is to be setup will be programmed in the C# language at a specified IP address. Commands that the user decides to issue to RADSAT will be taken by C# program and written to a text file. This text file will be located on the server.

The IP address of the server that is created will also be downloaded to the Wifi Shield on the PCB. In this fashion, the WiFi Shield will be able to connect to the server and read the text file that the C# program wrote. The WiFi Shield will then issue a command to the tank based on what was read in the text file. At this point, the WiFi Shield will be in a "standby" mode where it is reading any new commands. Once RADSAT has fully carried out the command that was requested of it, it will go back into “listening” mode to read in a new command. If there are no commands to be completed at the time, the R/C tank will remain in an idle state. The process of how this communication will take place is shown in F*igure 4.11.1* below.

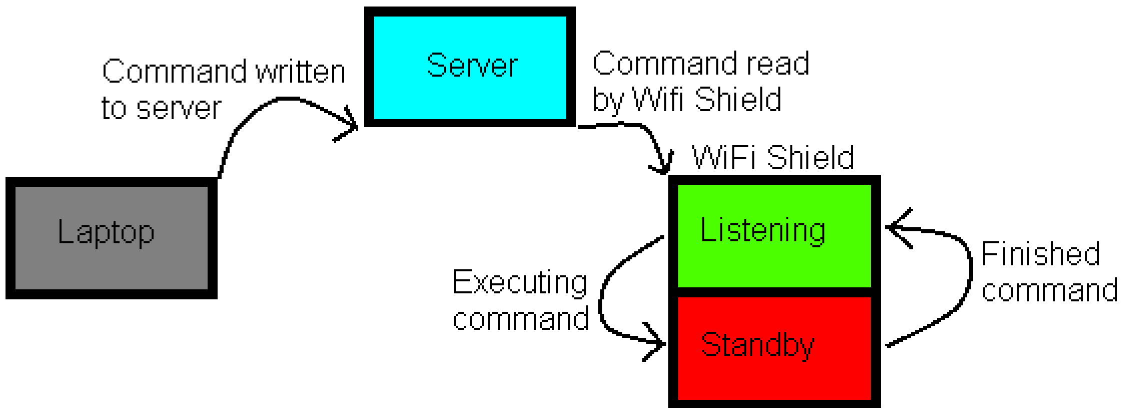


*Figure 4.11.1: Wireless communication*

The commands issued to RADSAT through the server will be bytes of information that will need to be decoded by the microcontroller. The commands will be read by the WiFi Shield in the same order that they are written to the text file on the server by the laptop. This will ensure that the commands are dispatched in the order that they were given.

The video stream that is to be sent from RADSAT’s camera back to the laptop well be done by the camera itself over a separate server. The reason the WiFi Shield is not used for this purpose is due to the fact that it would not be powerful enough to handle the video processing. For further explanation of how communication between the camera and laptop will take place, please refer to the sections on video processing.

## 4.12 Control Using the Microcontroller

In order to achieve full control over the tank using the laptop connection that will be established, the main circuit board that is used to control all of the circuitry on the R/C tank will be hardwired to the ATmega328P microcontroller through the PCB that was designed specifically for RADSAT. With this connection in place, the commands that are sent by the laptop will be received by the WiFi shield, forwarded on to the microcontroller, and dispatched as commands to the tank through the circuit board. The instruction flow that is to be followed by RADSAT for each command can be seen in figure 4.12.1below.

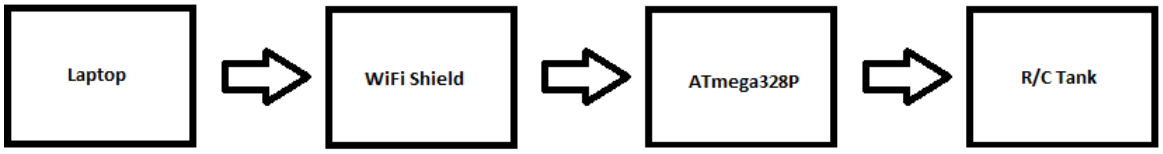


*Figure 4.12.1: Instruction flow*

The packets that will be sent over the physical layer to be decoded by the microcontroller will be in binary. Each packet will be sent as a byte, with a separate binary value being used to control different features of the tank. Once the instruction is decoded, the microcontroller will output the command to the tank through the specified digital I/O pin. Table 4.12.1 below lists the commands and the binary message that will be associated with them. As the tank will only respond to a limited number of commands, most of the bits transmitted will be redundant and ignored by the microcontroller. These redundant bits, which can be either 0 or 1, are represented by an "x" in the table.

| **Command** | **Binary Transmission** |
|---|---|
| Tank: Forward | 0000 xxxx |
| Tank: Backward | 0001 xxxx |
| Tank: Left | 0010 xxxx |
| Tank: Right | 0011 xxxx |
| Turret: Up | 0100 xxxx |
| Turret: Down | 0101 xxxx |
| Turret: Left | 0110 xxxx |
| Turret: Right | 0111 xxxx |
| Fire | 1000 xxxx |
| Search | 1001 xxxx |
| Connect | 1010 xxxx |
| Exit | 1011 xxxx |

*Table 4.12.1: Binary Commands*

The commands "Connect" and "Exit" will be two commands that are not sent on to the R/C tank through the microcontroller. Instead, they will be processed by the WiFi Shield to setup and break down the connection between the laptop and microcontroller. The "Search" command will also be different from the other commands in the fact that it will send multiple signals to the R/C tank from the microcontroller based on the search algorithm.

Control of the R/C tank will also come from signals sent by the sensors that are attached to the analog input pins on the PCB. When the tank is being controlled by a human user, these pins will send signals to override a command that would result in a crash. While RADSAT is in "Search" mode, these signals will be used by the search algorithm to avoid obstacles.

As there will be many hard connections made between the PCB and the R/C tank's circuit board, they will need to be set up relatively close to one another in the final design. In this way, there will not be random wires stretching far across the tank. The wires that are used will be left slightly longer than is required to ensure that all of the connections can be made. Once this is confirmed, the wires will be shortened to their required lengths and zip ties will be used to bunch them up so that they are not all loose and scattered recklessly about.

## 4.13 Microcontroller Layout

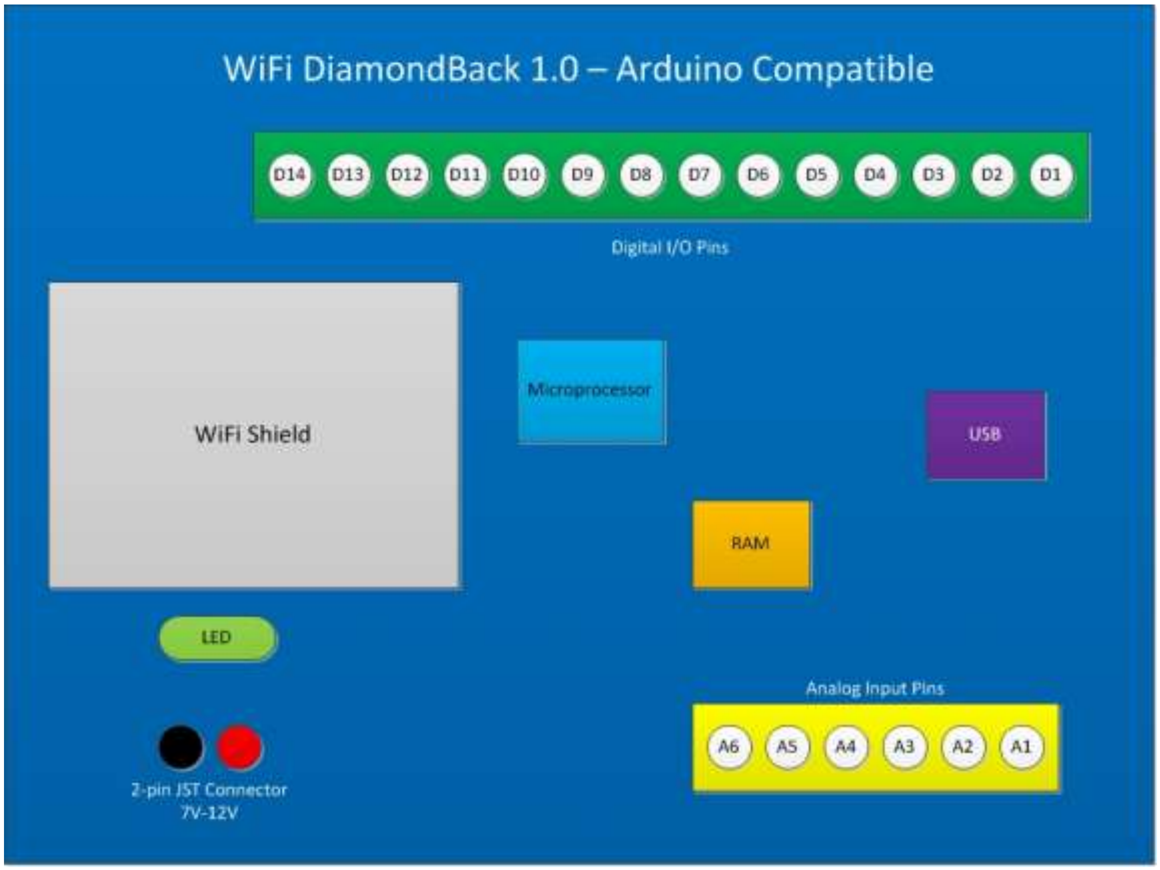


*Figure 4.13.1: Microcontroller Layout*

<u>2-pin JST Connector</u> - This is the connector that will power the board when power is not being supplied by the USB. The connector is able to power the board from the range of 7 volts to 12 volts. The board will be powered by a parsed part of the single battery source which will power every component on the robot. Because this is only a 2-pin connector it will be very easy to implement power onto the board, as we will only need to connect the two leads of the JST connector to the power-bread-board.

Digital I/O Pins - These are the pins that will control all of the different components on the robot, hopefully in a perfect harmony. Each pin on the diagram is labeled with D (for digital) and a number representing its placement on the board. They are responsible for sending a signal to each separate moving part on the robot, similar to how the tank would move via radio signals.

| Pin Location | Description |
|---|---|
| D1 | will be connected to the part of the RC tanks circuit which controls the forward motion for the left tread |
| D2 | will be connected to the part of the circuit which controls the backward motion for the left tread |
| D3 | the forward motion for the right tread |
| D4 | the backward motion for the right tread |
| D5 | will be attached to the servo which will control the rotation for the turret, will also send out varying pulses to change the direction of the servo |
| D6 | will be reserved for a servo in case we might need another one |
| D7 | will be used to control the firing mechanism for the air-soft gun. |
| D8 – D14 | for now will be extra I/O ports, there may be some use for them during the building process |

*Table 4.13.2: Digital I/O Pins*

Analog Input Pins – A1 – A4, these are the pins that are responsible for receiving the signals from the sensors, which are used for collision detection. When the sensors detect, or run into an object they will send back a voltage. When the voltage is sent back, signals will be sent to the digital I/O pins to autocorrect the path of the robot. The signals coming from the sensors will override any directions that are coming from the laptop. A5-A6 will be responsible for the directional compass which will send the degree that the RADSAT is facing relative to North, South, East, and West.

| Pin Location | Description |
|---|---|
| A1 | will be used for the infrared sensor on the right-side of the robot |
| A2 | will be used for the infrared sensor on the left of the robot |
| A3 | will be empty |
| A4 | will be used for the sonar sensor on the front of the robot |
| A5 – A6 | will be used for the compass sensor |

*Table 4.13.3: Analog Input Pins*

Rest of the Components - **USB** – the USB port on this board is the means by which the board is programmed. Also, the board is capable of switching power sources automatically between the USB and 2-pin JST connector. **Microprocessor** – uses a Atmel ATMega328P processor which is plenty fast for the simple operations that the board will be carrying out, but wasn't fast enough to transmit a video feed. _**RAM** – has 32KB of flash RAM, 2KB SRAM, and 1KB EEPROM. **LED** – this LED has a function slightly greater than most LEDs you find on a microcontroller. This LED will light up whenever WiFi connectivity is achieved. **WiFi Shield** – the component which enables our robot to communicate with the laptop and has connectivity speeds between 1MB and 2MB.

## 5.0 Design Summary

RADSAT (Reconnaissance and Demolition Super Attack Tank), as its name implies, is a tank whose main mission is to locate a possibly obstructed target, take proper aim at it, and fire upon command. On its surface, this task sounds simple enough for newly enlisted military recruits to do, but behind the scenes, the realization of its objective implies that a set of many subproblems must also be solved. Overall, the design of RADSAT solves all of these challenges by combining knowledge from years of formal and self learning, and months of research and design. The following block diagram (Figure 5.0.1) summarizes RADSAT's basic functionality and design.

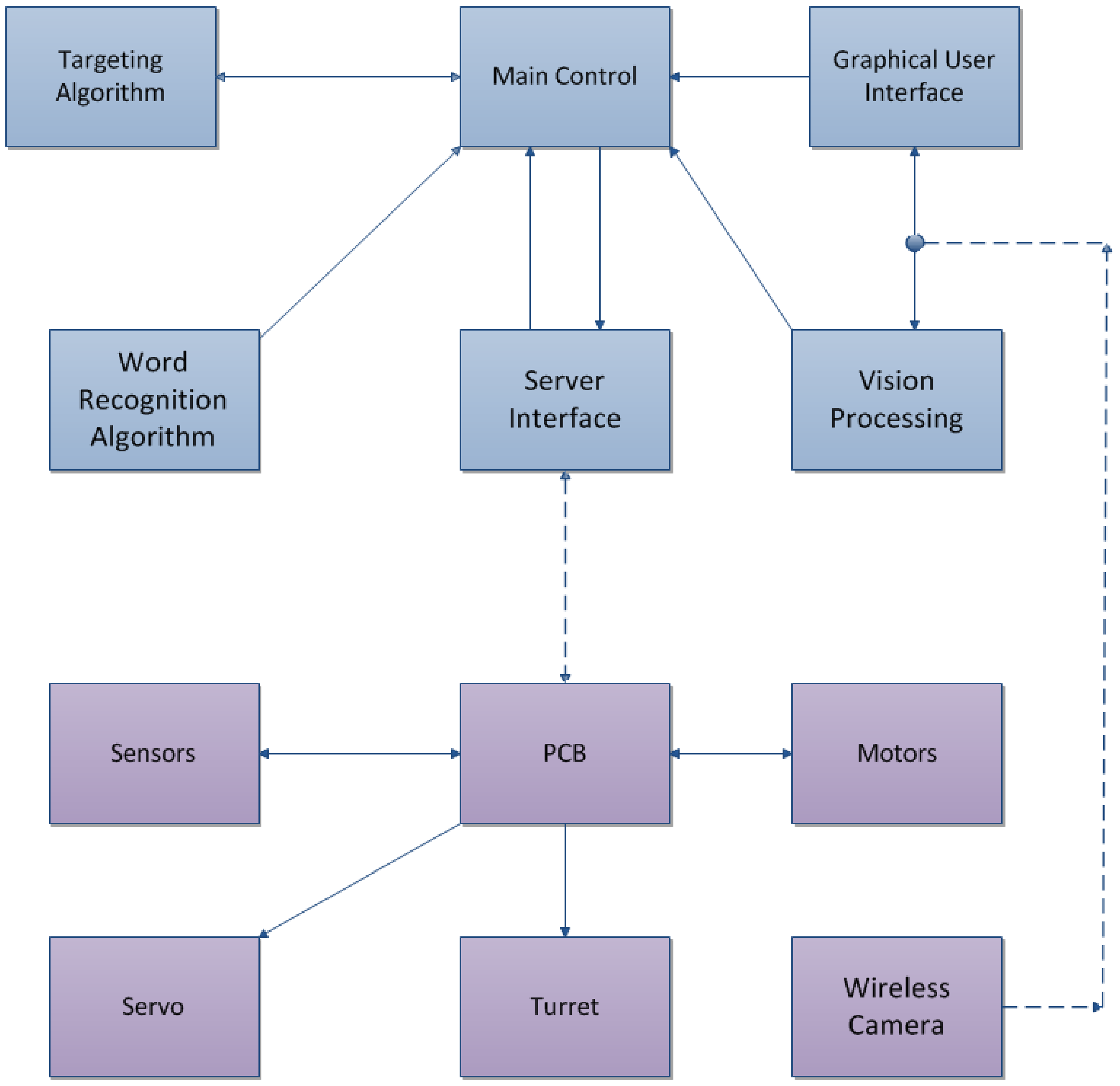


*Figure 5.0.1: Full System Block Diagram.*

Blocks color coded purple are blocks that are physically mounted on RADSAT's body. Blocks color coded blue are software blocks present on a computer.

## 5.0.1 Software Design Summary

The word recognition algorithm, written in C, uses the Port Audio library to interface with a computer's microphone to accept audio input. That input is available to the algorithm as an array of floating point values. The algorithm's dictionary is prerecorded and contains information on all required commands. Each command's values are compared to the values of the incoming stream.

The GUI is written in C# and is the most direct way to control RADSAT. Commands can be both directional and non-directional (e.g. up and fire, respectfully). It also presents a live video stream from the wifi camera mounted on RADSAT.

The video processing functions are written in C#. They use the AForge.Video libraries. It will take in a MJPEG stream from a server, and read each individual pixel, checking for a specific color and saving that color to an array. After the array is created, all pixels not in the array will be turned to grayscale.

The microcontroller code is written in C. It makes use of several existing libraries, including a library for servo control, server connectivity, as well as wire functions. This code will move RADSAT according to the will of the user, or the will of the sensors and programming.

## 5.0.2 Hardware Summary

The following section summarizes the options that were discussed and decisions that were made by the group for the final design of RADSAT. As the previous pages serve as a more detailed look into each aspect of RADSAT, the following pages will only be a brief discussion of what was decided upon.

Tank Body – Knowing that the tank body we would be using would be pre-manufactured and purchased by the group, the pros and cons of buying an off-the-shelf R/C tank or a hobbyist’s tank that could be expanded upon were compared. In the end, after comparing prices, functionality, sizes, consumer reports, available space, and mobility, it was decided to use a pre-manufactured R/C tank. Specifically, the SnowLeopard M26 was chosen to be used in the final design of RADSAT. This tank will cost the team about $95, but will be well worth it for the high quality reviews that it has received from other consumers and the fact that it should be easy to use when it comes to mounting equipment to its body. The body of the SnowLeopard M26 is again shown in figure 5.0.2.1 found below.

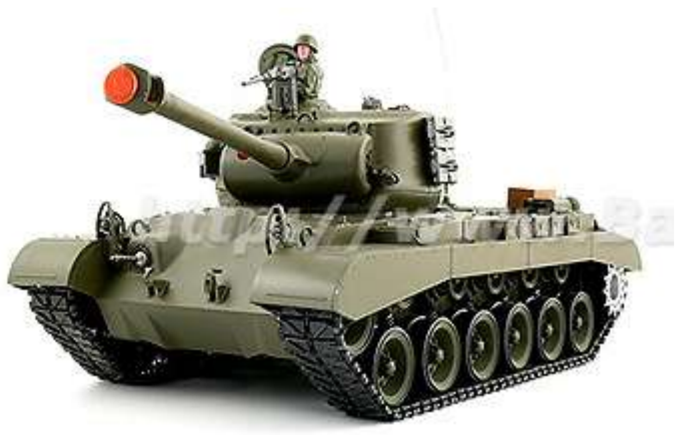


*Figure 5.0.2.1: The Snow Leopard M26*
*With permission from www.bananahobby.com*

Turret – With the body in place, the group then looked at options of how the turret might be implemented to allow full range in motion while it tracked the specified target. A 360 degree motion turret was considered, but it was decided that this option would not be ideal due to the twisting of the wires and the design issue this would present. Wireless connectivity between the turret and base was considered but dismissed due to various reasonings Limiting the turret's rotation to a 360 degree pan was also thrown out, as the idea of the camera having to make a full rotation if the target passed a certain trajectory point relative to the tank was unappealing. In the end, a turret with 720 degrees of rotation was chosen to be implemented in the final design of RADSAT. This will allow the turret to follow the target fully around the tank without causing issues with the wiring associated with the turret.

Gun – The gun that would be mounted on the servos and used by RADSAT to fire at a specified target was then decided upon. Though the option of using the cannon on the SnowLeopard M26 was a possibility, it was decided that this would not be as accurate or reliable as an airsoft gun that could be mounted to the base of the tank. Having made the decision to use an airsoft gun, the team then had the option of buying a new one or using a 7-year old one that was already owned by one of the team members. In the end, a UHC Steyr mini electric airsoft gun which was already owned by one of the team members was decided to be efficient enough to use in the final design and avoid having to include an extra cost for RADSAT by buying a new gun. This airsoft gun will be modified in order to reduce its size and weight and will be mounted directly on top of the servo in the design of the turret. The gun is again shown in Figure 5.0.2.2 below.

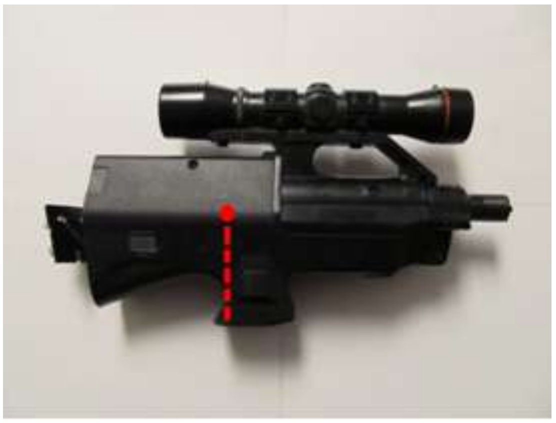


*Figure 5.0.2.2: The gun*

Servos – In order to perform the targeting of the airsoft gun by RADSAT, two servos were required to act as the base of the turret. A pan servo would be used to offer the turret full range of motion around the body of the tank, while a tilt servo would be used to target the turret at objects that are above the horizontal plane of the tank. And two-in-one pan and tilt servo could have been considered, but was thrown out to the high price that would have been associated with it. After comparing several pan servos, the HS-422 by Hitec was chosen to be implemented in the final design of RADSAT. The tilt servo took a little more deliberation with the added torque calculations that needed to be considered. In the end, the HS-485HB, also by Hitec, was chosen. These two servos would serve as the base of the turret and would screw directly into the body of the tank, while the airsoft gun and camera could both be mounted on top of the tilt servo, completing the turret.

Sensors - A requisite for effective autonomous behavior is the ability to accept some kind of sensory input on which to base decisions. RADSAT is designed to use two Sony GP2Y0A02YK infrared sensors, one MaxBotix MB1210 ultrasonic sensor, and one Honeywell HMC6352 compass module. To mitigate the problem with the infrared GP2Y0A02YK's narrow beam width, these sensors will only be used to sense for obstructions located to the sides of RADSAT. For forward obstruction detection, the MB1210 will be used because of its relatively wide beam pattern, compact size, and its ability to detect small obstructions. The HMC6352 is required to keep track of the direction RADSAT is facing at any given time.

Microcontroller – The microcontroller that is onboard the Arduino Diamondback prototyping board that the group will be using for testing and the final design is the Atmel ATMega328P. Assuming that all goes well using the Diamondback for testing, this microcontroller will again be used in the final design of RADSAT on the PCB. The microcontroller will then be mounted on the body of the R/C tank and hooked up to all of the other hardware components. Digital I/O pins on the microcontroller will be hooked up to the SnowLeopard M26's circuit board to control the movement of the treads, the servos for rotation of the turret, and the airsoft gun for firing. The Analog input pins will be hooked up to the various sensors on the side of RADSAT. This microcontroller will be the brain of RADSAT, and will be the unit that carries out all of the specified functions according to the software that is develop for RADSAT.

Wifi Shield – Also included on the Diamondback is a WiFi Shield that has 802.11b wireless connectivity capabilities. During prototyping, this WiFi Shield will be used to communicate signals from the laptop computer to the microcontroller. These signals will be used for the commands that RADSAT will carry out according to the software. The laptop and WiFi Shield will communicate using a server that is to be setup by the laptop. Assuming all goes well during prototyping, the WiFi shield will again be implemented on the PCB in the final design.

Laptop – The human commands that RADSAT will respond to will be delivered by a laptop computer. As stated above, communication will take place between the laptop and the WiFi Shield on the PCB. Given that all of the programs that will be used to implement RADSAT can be downloaded on any computer, the group will be able to use any laptop in the final design of RADSAT.

PCB – The PCB was designed to incorporate all of the necessary components that were used on the Arduino Diamondback prototyping board while excluding all of the unnecessary components. The microcontroller that will be used, as listed above, will be the Atmel Mega 328P. There will also be a WiFi Shield on the PCB for wireless connectivity. The PCB will include fourteen I/O digital pins and six analog input pins. A RAM with has 32KB of flash RAM, 2KB SRAM, and 1KB EEPROM will be used on the board. There will also be an LED that will light up whenever there is a successful connection for the WiFi Shield. Finally, there will be a 2-pin JST connector to power the PCB and a USB connection to download the programs to the board.

## 5.1 Video Processing Class Diagram

The following class table below in Figure *5.1.1* can be used to summarize the design that will be implemented by the team in regards to how RADSAT searches for and locks on to a target. As can be seen in the diagram, the software that is written for each class will be implemented by the other classes to accomplish the functionality set out for RADSAT.

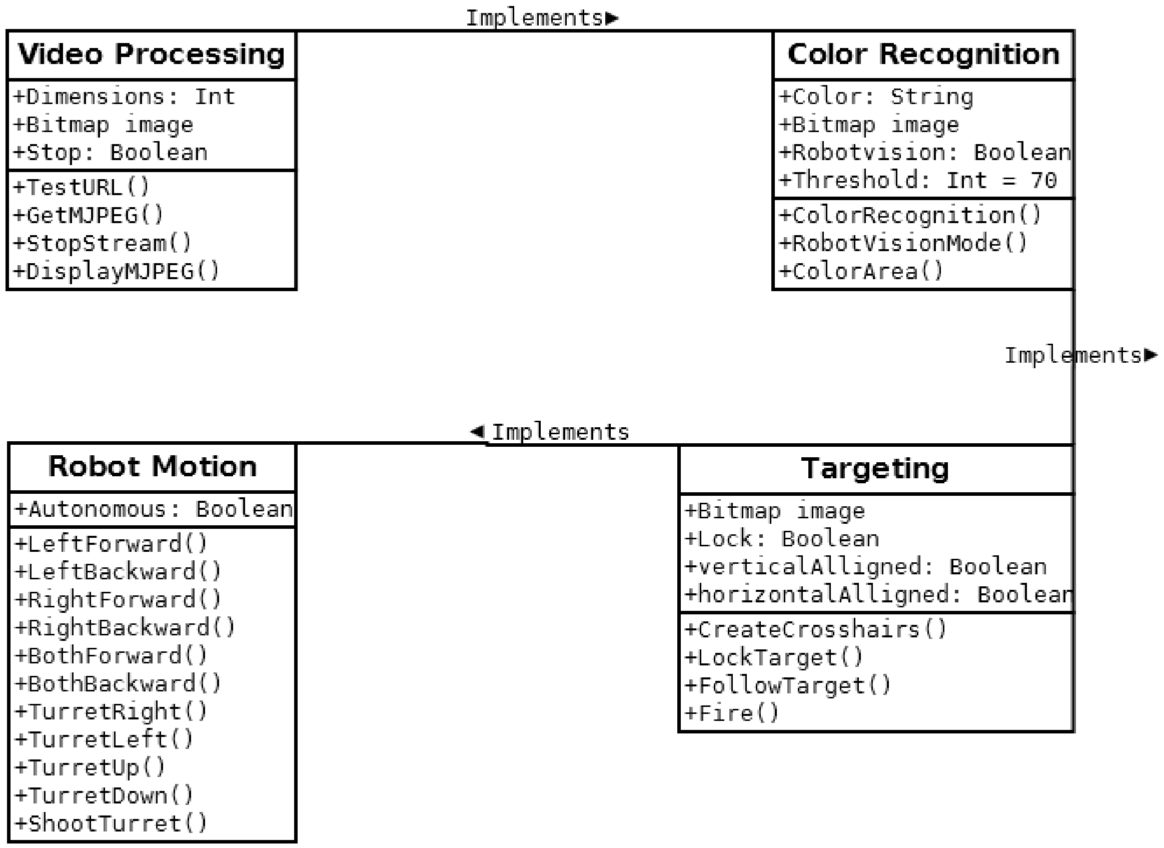


*Figure 5.1.1: Class table*

In order to accomplish the task of searching out a target, the video processing class utilizes the color recognition class in order to search out a specified color as the target. The color recognition class utilizes the targeting class for this goal of locking on to the target. The targeting class utilizes the robot class as it moves through the steps of its search algorithm to find a target. Finally, the robot motion class controls all of the motions of RADSAT through the movement commands for both the tank and turret, as well as the firing command for the airsoft gun. These sections are all expanded upon in the summary below.

<ins>Video Processing</ins> – The video processing class deals with the handling of the MJPEG video stream from the webcam's server using AForce Video's and iSpy's open source libraries. The TestURL() function ensures that a connection is established between the program and the server. Once a connection is established and it is confirmed that there is data on the server, the function GetMJPEG() retrieves the MJPEG video stream images, converts them into JPEG's, and then into Bitmaps so the pixels can be manipulated individually. In this function, the dimensions attribute is also specified as the boundaries of the MJPEG stream. Upon pressing the "Exit" button the GUI, the StopStream() function is used to break from the GetMJPEG() loop and breaks the connection with the server. The DisplayMJPEG() is the final function that will call upon the other classes for video editing as needed. When the image is deemed ready, it is

this function that will ultimately display the image captured by the webcam on the computer.

Color Recognition – The color recognition class is what determines what objects to target on by using functions to manipulate the pixels in the MJPEG stream in order to recognize colors. The ColorRecognition() function utilizes threading as it searches the bitmap image for the specified color by separating the image into four separate quadrants and searching each quadrant at the same time. The function looks at every pixel individually and uses the equation D = sqrt((R - X)^2 + (G - X)^2 + (B - X)^2) to process the value of each pixel. In the equation, R represents red, G represents green, and B represents blue. The X value will be set to 255 within the parentheses of the color it is searching for and 0 inside the other two sets of parentheses. This D value will be compared to a threshold value of 70 to see if it is the color that RADSAT is searching for. Finally, this function will keep an array of the coordinates of each individual pixel that is found to be the color RADSAT is searching for. When active by the Robotvision boolean, the RobotVisionMode() function will take in all of the pixels that are not the color RADSAT is searching for and turn them gray using the equation N = (R + G + B) / 3, using N to replace the value for R, G, and B. This produces an image to display that will show the user what the robot is seeing in its search for the target. Finally, the ColorArea() function will determine the center of a rectangular area of pixels of the specified color and send this information to the targeting class so that RADSAT will be able to determine where to aim the turret so that it hits the target.

Targeting – The targeting class is used to aim the turret to the center of the specified target. The CreateCrosshairs() function creates black crosshairs located in the middle of the ColorArea() determined by the color recognition class and displays these crosshairs on the image for the user. The LockTarget() function is used to lock RADSAT on to the specified target, aiming the turret towards the center of the target no matter how much it or RADSAT moves. The FollowingTarget() function is used to move RADSAT in order to center the target in front of RADSAT. This is done using movement functions from the Robot Motion class. RADSAT first positions itself vertically in front of the target by moving the R/C tank's body or by rotating the camera. RADSAT then rotates the turret up or down until the turret is horizontally aligned with the center of the target. Finally, the Fire() function will of course fire off the airsoft gun once it is locked on to a specified target and a human user issues the command to fire through the GUI or voice command.

Robot Motion – The final class in the class diagram above, the robot motion class controls the movements of RADSAT while it is set to autonomously search out the target. The LeftForward() function moves the left cannon tread forward. The LeftBackward() function moves the left cannon tread backward. The RightForward() function moves the right cannon tread forward. The RightBackward() function moves the right cannon tread backward. The

BothForward() function moves both of the cannon treads forward.  The BothBackward() function moves both of the cannon treads backward.  The TurretRight() function rotates the turret and camera to the right.  The TurretLeft() function rotates the turret and camera to the left.  The TurretUp() function rotates the turret and camera up.  The TurretDown() function rotates the turret and camera down.  Finall, the ShootTurret() function fires off a round from the airsoft gun at the locked on target.  This function, again, only is called when a human user decides to fire.

# 6.0 PCB Design

## 6.0.1 PCB Overview

The PCB has the responsibility of processing wireless transmitted information and turning it into signals that the can be read by the various components that need to be controlled. It has a wireless XBee transceiver to send and receive information from the control software on the laptop. The wireless chip will interface with the microprocessor, an Atmel ATMega328P which will in turn handle all control signals on board the tank. Overall, the board must have two outputs for servo signals, two outputs for motor signals, four input lines for sensor signals, and one output for gun activation. It must have power input as well.

The circuit will also have an LED for the wireless signal to indicate that data or power is or is not being transmitted. This is to help with setup and debugging of the device. It is designed by looking at the schematic for the ATMega328P, the schematic for the Xbee and the schematic of the Arduino board. It will be modeled after the Arduino, removing parts that are unneeded for this project, and designing a concise version.

Eagle will be the software used because it has good support and good tutorials. There are libraries available and places to order from that accept .brd format. The decision still remains as to whether the board will be ordered with the parts populating it, or separate such that the parts must be soldered. Since there is not an available source of expertise in board soldering, having it assembled is more attractive. However, The price is much higher, in the range of two hundred dollars.

Because this project has no outside funding, keeping a low budget is a high priority. Therefore, the PCB will be ordered, and the parts that go on it will be ordered, and proper soldering tools will be ordered, and the parts will be hand soldered by a team member. Soldering is an important skill for an engineer to have so it will save money and build experience. Despite the fact that it is likely to be more difficult than expected and take a long time and perhaps even be painful or impossible, it is the first choice plan.

The PCB will be ordered from http://www.dorkbotpdx.org. It is a website for robot enthusiasts and similar types. According to the order form from this website a PCB will cost $5 per square inch and take about two weeks to process. This website is less professional-looking and more hobbyist-looking than most, but that is probably why is it also cheap and more personal. Due to the long fabrication process and unfamiliarity with PCB assembly, this will be the first design prototyped and worked on in the latter half of senior design.

## 6.0.2 PBC Circuit Layout

The main components on the PCB are the power-routing systems, the control systems, the microcontroller and the wireless chip. The power source is located off the PCB and plugs in at the upper left in the schematic on the next page. Each of the six IC's are voltage regulators wired in parallel. They provide the regulated voltages to the microcontroller, the wireless module and to headers where components from off the board will be plugged in. The microcontrollers signals go to the voltage regulators of the motors and gun, turning them off and on. Those are the top three components on the PCB. The motors also have signals to a relay that will invert the voltage across them. Six more signal lines go to the servos and sensors which are at the bottom of the diagram, in parallel with each other. There are output plugs here as well. Lastly, the Xbee wireless module in the upper right is interfaced with the microcontroller and is wired in parallel with it to receive its power.

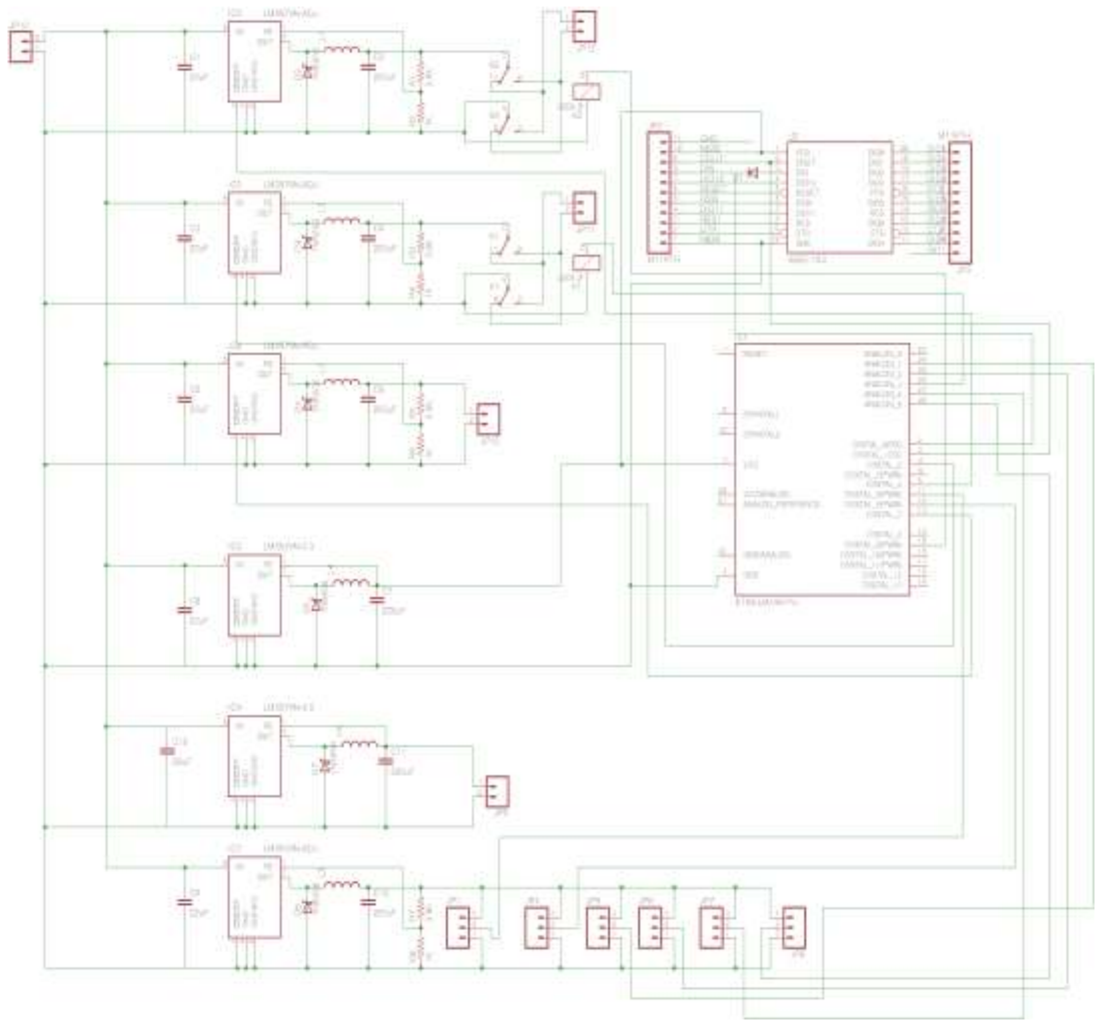


*Figure 6.2.2.1: The schematic of the PCB*

# 7.0 Project Prototype Testing

## 7.1 Hardware Test Environment

The hardware testing process should be conducted indoors in a controlled and well lit environment, preferably in an empty on campus class room or office. The room should be as quiet as possible and free from outside noise that could cause interference with any intended verbal commands. The size of the room is generally irrelevant, down to reasonable minimum and up to a reasonable maximum, but the shape should be relatively rectangular. An average room the size of HEC 118 would be ideal, although rooms slightly smaller will take less time to search. Obstructions may be placed anywhere in the room and should collectively be limited to occupying less than 10-15% of the floor's total surface area. The target should be reasonably sized (square, 12x12in or above) and of the correct color.

## 7.2 Hardware Specific testing

Each of the parameters outlined in the requirements list of section 2.3.2 will be tested for desired functionality. To track if a part works and how well it works the following table will be filled in. At the time of testing a check mark in the first column will indicate that a part passes the test. If it passes, a number in the second column will indicate the extent to which it passes.

| | Does it pass? | How well? |
|---|---|---|
| Shot distance | | |
| Fire rate | | |
| Drive speed | | |
| All-terrain op. | | |
| Battery life | | |
| Battery voltage | | |
| Chassis size | | |
| Turret movement | | |

*Figure 7.2.1: Part qualification table*

To test the gun's shot distance and fire rate a 6 volt voltage will be applied directly to it and it will be fired with the barrel level and approximately two feet from the ground. The distance will be recorded, then it will be angled up approximately 15 degrees, fired and the new distance recorded. This will help calibrate the aiming algorithm if it is found to help the tank's accuracy.

For testing the drive speed and all-terrain capabilities, the tank will be kept stock. It will be left unmodified out of the box and using the radio control it comes with, it will be driven on a variety of surfaces. After these tests, it will be taken apart and modified. After it is fully outfitted with the custom features required for this project, and if there is reason to believe its performance has hanged, it will be subjected to the same tests again. To meet the requirement of all-terrain travel, eight different types of ground have been chosen to test. The following table will document the environments in which the tank passes this simple mobility test.

| Surface | Ability to traverse |
|---|---|
| wood floor | |
| carpet | |
| concrete | |
| short grass | |
| long grass | |
| sand | |
| small rocks | |
| large rocks | |

*Figure 7.1.2: All-terrain verification*

Battery life will be measured during prototype testing, after RADSAT is assembled and can run. It will simply be timed with a stopwatch. Similarly, to test the battery voltage it will be as simple as touching a multimeter to the leads. Also a simple test, the chassis size will be measured with a ruler to make sure it fits the requirements. The real test for the chassis size will be making sure the PCB, battery and other electrical components fit inside it. Also, it is important that it is large enough that the turret does not make it top-heavy and prone to tipping over. This is not an anticipated problem, but it will be tested to be sure it is indeed not a problem.

The final batch of test procedures will apply to the turret. After the turret assembly is built the servos will be tested to make sure they can handle the weight. The results will be apparent in the form of stripped gears, burnt motors, or non-movement in the case of failure. In the case of success, properly moving turret parts will be observed. The pan servo will also be tested for 720 degree rotation. These tests will be done by applying 5 volt square impulses of 600 to 2400 µs to the tilt servo and impulses of 1500 to 1900 µs to the pan servo. For the test to be considered passed, the servos should be observed moving freely and smoothly within their ranges.

## 7.3 Software Test Environment

Although many of RADSAT's components are cross platform compatible, all tests should be conducted using Windows 7 on a device with either an integrated microphone or capable of using an external microphone. The device must also have the latest Microsoft .NET runtime libraries installed to ensure proper functionality.

## 7.4 Software Specific Testing

### 7.4.1 Introduction

Testing to verify the proper functionality with RADSAT recognizing vocal commands will happen in two phases. The first phase consists of performing unit testing on every function. The second is to test against a set of test cases. After completion of the entire set of set cases, and if every test is successful, the vocal command module would be considered fully functional. Table 7.4.1 summarizes the full set of test cases.

| | |
|---|---|
| Test Case #1 | Must be able to translate basic colors in an environment with minimal noise. |
| Test Case #2 | Must be able to translate basic commands in an environment with minimal noise. |
| Test Case #3 | Must be able to distinguish between the full range of all vocal commands |

*Table 7.4.1: Vocal command test cases.*

The test cases are used to determine whether or not the most basic of functionalities expected from the vocal command algorithm exists. As specified in the requirements section, RADSAT must be able to accept spoken commands from its controller (the user). Depending on the sensitivity and other various factors, the user should be relatively close (within normal operating distance) to a microphone and a computer. The tester is then required to speak into the microphone in a clear voice with proper pronunciation. The test should first read the colors supported and check to see if the module gives the correct value in its stdout. The tester then can test commands that are not related to color, namely left, right, stop, and find. After successful completion of the above cases, the tester should then test all commands, in random order and check the output. Upon successful completion of the three test cases, the testing of the algorithm can be considered successful.

### 7.4.2 Vocal Command Testing

Testing to verify the proper functionality with RADSAT recognizing vocal commands will happen in two phases. The first phase consists of performing unit testing on every function. The second is to test against a set of test cases. After completion of the entire set of set cases, and if every test is successful, the vocal command module would be considered fully functional. Table x summarizes the full set of test cases.

| | |
|---|---|
| Test Case #1 | Must be able to translate basic colors in an environment with minimal noise. |
| Test Case #2 | Must be able to translate basic commands in an environment with minimal noise. |

*Table 7.4.1: Vocal command test cases.*

The first and second test cases are used to determine whether or not the most basic of functionalities expected from the vocal command algorithm exists. As specified in the requirements section, RADSAT must be able to accept spoken commands from its controller (the user). Depending on the sensitivity and other various settings, the user should be relatively close (within comfortable using distance) to a microphone and a computer. The user is then

## 7.5 Autonomous Design – C# Control Functions Testing

In order to test this section the RADSAT must be connected to a server. The C# code doesn't have to control the RADSAT for initial testing; it just has to be able to post the commands on the motion.txt file on the same server the RADSAT is connected to, so that the RADSAT's microcontroller can read it when it's ready to. While testing the C# control functions, it will be assumed that the microcontroller will be able to decipher everything as long as the motion.txt file is updated correctly.

So, the C# code will need to be able to see, and update the motion.txt file on the server. Once an update command is given, physically opening and viewing the motion.txt file to see whether it has updated will occur. Later, once it is able to do this, the microcontroller will be asked to print the command that it is supposedly reading. If the command that was printed coincides with the C# code, then this part will have been programmed correctly. This part will be repeated for every function within the C# Control Functions section.

Once it is understood that the microcontroller is actually receiving the commands correctly, it is time to make sure the movement is also moving correctly. Every function will be tested to make sure its described movement is what actually occurs with the RADSAT. An example of this is: if the command for "stand-by" is the current command on the motion.txt file, then the RADSAT should not be moving. Another example is: if the current command is "move forward", the RADSAT should be moving forward. Similar testing will occur for every movement. The RADSAT should also continuously move until a new command is given, and should really only stop when the command is to be in "stand-by" mode.

Once all this testing is completed, one of the most crucial parts of the RADSAT project will be completed. Without this section of testing being complete, the RADSAT would just be a motionless lump without any real purpose. Therefore, this testing is very important to make the RADSAT a fluid, intelligent, robot.

## 7.6 Color Recognition Testing

To test the color recognition section of code, first, each function must be tested separately. Most of the functions will be tested using a simple GUI which is capable of displaying a JPEG as well as a Bitmap image. Using this simple GUI will make the testing process more fluid, because the project's main GUI will be created only after all of the functions are already working and it is known exactly what it will be needed to do. The following are the steps which will be required to test each function:

<u>ColorRecognition()</u> – In order for this function to be working properly all four threads working simultaneously must be coordinated and produce usable data. This function will be receiving a Bitmap image as well as a string indicating which color will be noticed. The test Bitmap image which will be used for testing will contain all different shades of red, blue, and green which the RADSAT should be able to recognize. All the shades of the colors will be repeated in each quadrant of the picture; doing this will allow a test for each separate thread, to make sure they are all working in cooperation; this is depicted in Figure 7.6.1. Each pixel which passes the color test will remain its designated color. However, each pixel which does not pass the color test will be passed to the "test only" function ReColorTest(). The ReColorTest() function will turn the non-color-passed pixel white, which by the RGB color scale will look like R=255, G=255, and B=255. Once the new Bitmap image is completed, the image will be displayed on the simple GUI. If all the pixels not in the color range are colored white, and all the pixels within the color range are unchanged, the test will be considered a success.

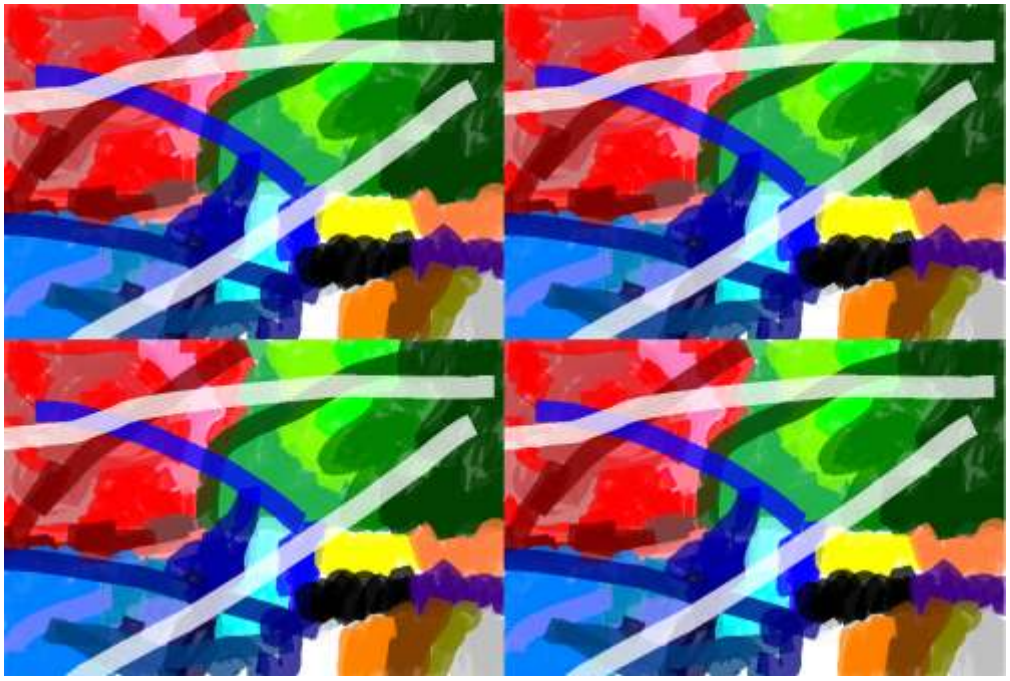


*Figure 7.6.1 Test Image*

After the test image is successful, the next test will consist of actual screen shots from the MJPEG stream. This test will be very close to what the actual final product will be testing. The final code will essentially just be taking in screenshot images, but just at a must faster rate. So, if the screenshot works, then it will be known that the function will work for the final product.

Also the ColorRecognition() function is supposed to create an array which keeps track of the pixels that are the appropriate color. Testing for this will be similar to testing for the color range described earlier. Only this time, a function to create a Bitmap image will be used. The function, called BitmapCreateTest() will take an array and color all the pixels specified in the array black (R=0, G=0, B=0) on a Bitmap image with a white background. If the shape of the black blob, which represents the specified color, is identical to the shape of the image described in the above paragraph, then the test will be deemed successful.

RobotVisionMode() – This function will not be very hard to test. For this function, first, a checkbox will have to be added to the Simple GUI, which will be the indicator for whether or not robot vision mode should be active or not. This function will just take in pixel coordinates, and if the checkbox is checked, then the image should return the pixel coordinates of the image as gray, otherwise, if the checkbox is not checked, it should not affect the image at all.

ColorArea() – This function will be easiest to test after the ColorRecognition() function is working perfectly. If the ColorRecognition() function isn't working perfectly, then an array of pixels containing an area of an image, would have to be recreated by hand. Recreating the array by hand would simply take too much time, and only give one example to test with. The ColorArea() function, once working properly, is supposed to create an invisible rectangle of four points in an array. However, in the testing phase the ColorArea() function will actually physically draw the rectangle, connecting all the points. This will allow an easier means to view what the function is doing. By seeing the actual rectangle being drawn, the area that the function is seeing, can be seen much easier by the programmer. To test whether the function is properly able to detect the center pixel, the function will color the pixel, as well as some surrounding pixels, white. The programmer will then be able to see if the white dot is where it's supposed to be. If the white dot appears in the center of the color area, then the function will be in working order.

## 7.7 Microcontroller Code Testing

The microcontroller code testing will ensure that the RADSAT is able to move according the will of the program. This is some of the most important testing for the entire RADSAT project. Almost everything within the project will be affected by being able to move the RADSAT according to a certain set of instructions. Once this testing is completed, the RADSAT will be able to sense obstructions, move accordingly, and be able to move according to the directions coming from the laptop. To test the microcontroller code, the C# code does not have to be in working order. Any orders potentially sent by the C# code can simply be recreated within the microcontroller code.

The first test will make sure that the RADSAT can act according to a C# command, even if the C# code is not in working order. Before the microcontroller is even connected to the server, the testing will begin by creating a simulated string. Since the commands will be issued in string form, the program will be able to respond according to a simulated string, even within its own code. For example, when given the string “02” the program should be in “left-forward” mode and should act accordingly. For testing purposes, a 5V LED will be hooked up to the appropriate output pin, and should light-up when that command is given. Simultaneously, the microcontroller will output a message on the IDE's terminal, stating where it is. For example, it might say, “Within boundary for string ‘02’.” when the “02” command is given. This will be repeated for every different command that the RADSAT has. The program should always enter the appropriate SWITCH statement, and if it doesn't appropriate measures will be taken.

Once the IF and SWITCH statements are properly tested, the program will be tested in a more “real life” scenario. The RADSAT will be tested to make sure it can read the string from a simulated command from C# code, which means reading from the motion.txt file. After the server is set up, the RADSAT will attempt to access the motion.txt file that is located on it. There will be a simple print command that will print whatever the RADSAT is reading from the motion.txt file. Once it is verified that the RADSAT is reading the correct information, the correct information will be the compared in the IF and SWITCH statements. Then the motion.txt file will be open on the laptop, and the programmer will continue to change the command then resave the motion.txt file, and see if the RADSAT comes up with the appropriate response (i.e. the correct LED lighting up, and the correct print statement being printed). Once this testing is completed, the RADSAT will be ready to receive commands from the laptop, and give commands to the RC tank.

Following knowing the RADSAT can give and receive commands properly, it's time to fine tune some of the outputs it gives. Besides testing the voltages required for the RC tank move properly without being damaged (testing for the hardware rather than the software), the proper pulse rate to control the servos needs to be discovered. Of course testing will begin using a very slow pulse rate, then moving the pulse faster, and faster until the desired speed is achieved. The pulse will be altered by altering the "maxspeed" value within the code.

Now, the sensor code will have to be checked to ensure all the microcontroller code is working properly. Since all code is separated by an IF statement, the sensor code should override all other code when a sensor voltage is received. First, the analog inputs will be attached to a DC volt generator, doing this will test to see if the code is receiving voltages correctly. For example, if some voltage is sent into analog pin one, then the pin should have some voltage when the analogread() function is implemented. Once it is confirmed that the microcontroller code is reading a voltage from a pin, the actual sensors will be attached.

Since the sensors work by sending back a voltage when there's an obstruction in their path, there will be tests that will print out the voltage to the Arduino terminal whenever it receives a voltage from the sensor. Once that part is working correctly, and the voltages are what they are supposed to be, then the actual avoidance movement functions will be tested. First the timers will be tested, since this is what the movement will be based off of, also the output pins at this point will have already been proven to work. The timers will control a while loop that will continuously send movement commands to the tank, until the timer runs out. So, to test this statement a message will be printed at the beginning of the while loop stating "Begin", then once the while loop is exited, another statement will be printed as "End". The time in-between the "Begin" and "End" statements will be timed externally with a stopwatch, to ensure correct time.

Once this section's testing is completed, the basic movements of the RADSAT, as well as most communication will be functional. This will allow for the progression of the more complex sections of the RADSAT to be implemented, such as all C# functions.

## 7.8 Targeting System Testing

Once the targeting system testing is completed, the RADSAT should be able to successfully target and shoot at objects with a good degree of accuracy. The various functions within this class, will mostly be able to be tested independently from the rest of the classes, and only FollowTarget() will be reliant on other parts of the RADSAT to be working. Most functions within this section will be tested using a Simple GUI which will display Bitmap images. The following explains how testing will go within each function:

CreateCrosshairs() – This function will be fairly easy to test. Since most of the work will be done in another function, and all CreateCrosshairs() does is read in a single location of a pixel. The CreateCrosshairs() function draws a crosshair based on the location of the pixel it receives. Therefore, in order to test if it is working correctly, all that has to be done is pass it a pixel location, and see if that location has a crosshair drawn around it. The image it is working with, will be colored completely white, as there’s no real need for any complex or realistic image.

LockTarget() – There is not much to test for the LockTarget() function. This is because; the main testing for this function will take place in the voice recognition portion of testing. The voice recognition section will be responsible for receiving the command to lock, and all this function does is activate other functions which will carry out the locking. However, a mock button on a GUI will serve to test the LockTarget() function, to make sure the target can at least be activated via the GUI.

FollowTarget() – This function will be much more difficult to test. First of all, all the functions from 4.6.2 Autonomous Design – C# Control Functions, must be working, as well as everything from 4.6.3 Microcontroller Code. These other sections need to be working, because the majority of what FollowTarget() will be doing is using the C# Control Functions to adjust the RADSAT into an appropriate firing position. The Video Processing section will have to also be working properly, otherwise, a fake MJPEG stream would have to be created that would be used solely for testing, which would take a considerable amount of time.

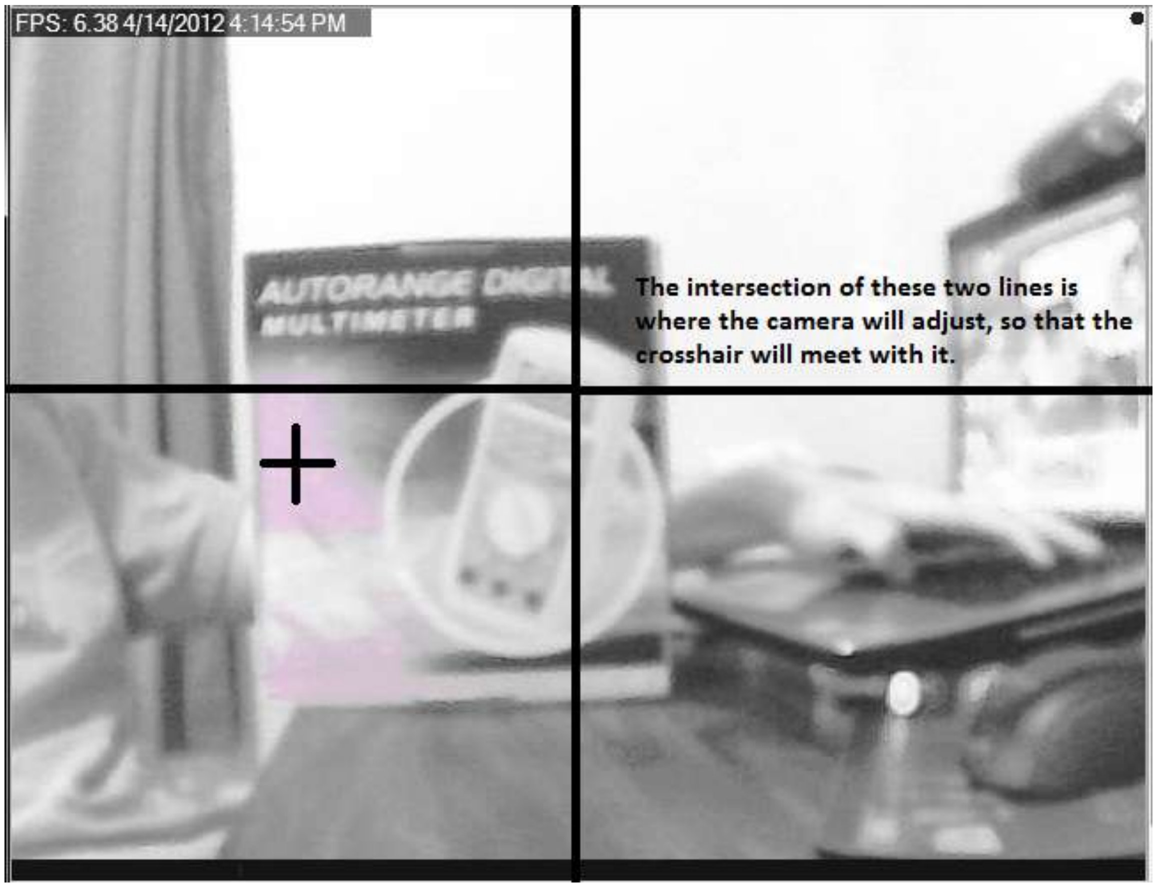

FPS: 6.38 4/14/2012 4:14:54 PM
AUTORANGE DIGITAL MULTIMETER
The intersection of these two lines is where the camera will adjust, so that the crosshair will meet with it.

*Figure 7.8.1: Crosshair and Line Intersection*

The first thing that would need to be accomplished for testing the FollowTarget() function, would be to calculate the trajectory of the bullet fired by the air soft rifle relative to the MJPEG stream. This will allow the function to know exactly how to reposition the crosshairs in order to obtain the optimal shot. Through trial and error, eventually the exact position on the MJPEG stream will be obtained, and then further testing can progress. This position will be the focal point of where the invisible horizontal and vertical lines will go, because at their intersection is the optimal place to shoot the target. However, for testing purposes, the invisible lines will be colored to black, making the invisible rectangle visible, thus making it easier to see what the function is doing. Once that is in place, making sure the program repositions the target to the center of the lines will be tested. The program will first have to stop on the vertical axis. If it doesn't stop, or stops too far left or right, then some readjusting will have to take place. Following the vertical axis will be the horizontal axis, where a very similar testing process will take place.

Fire() – This function should be in working order as long as the voice recognition and ShootTurret() functions are working. All this function's job is to receive a signal to fire, received from either voice recognition or from the a GUI button, which it then passes to the ShootTurret() function. The main purpose of this function is to allow any readers of the code to easily know what is going on.

Once this section is up and running, the majority of the tank will be fully operational. This section relies on many of the more difficult sections to be in working order, even before this section can begin its testing phase.

## 7.9 Video Processing Testing

This testing ensures the video feed from the WiFi security camera is properly connected to the laptop, as well as the C# code. This is among some of the first testing that will take place. Most of the robot's functions are reliant on this section of code being in working order. Therefore this is some of the most important testing for further development for the RADSAT.

The very first part of this testing, will be to test to make sure the WiFi security camera is properly connected to the server it is supposed to be connected to. This will be done by just following the instructions that came with the security camera, to connect to its designated server. This requires using the Firefox internet browser, in combination with the software that came with the WansView WiFi security camera. Within the WansView server, is the location of the actual MJPEG video stream. Once this location is discovered, it's time to test to make sure this location is the correct one. This can be accomplished by using a separate program which is designed to read in and display MJPEG streams. The program which will be used for this is "iSpy". Then if the MJPEG stream properly appears within the iSpy program, then it will be determined that the URL obtained from the WansView server is the correct one.

The next part of testing involves connecting to the MJPEG's URL using C# code. For this part, the C# code will connect to a camera source which has already been proven to work. There are several security cams which are free for use to the public, and one of those will be chosen for testing. Once this security camera stream can be properly viewed with C#, then since it will be known that the code has been implemented correctly, and so it'll be time to move on to the RADSAT's video stream. Once again, the new stream will be known to be working once this stream can be viewed with the C# code. Another way to tell if it is working is if it can at least return some bytes that will be tested using the function TestURL().

GetMJPEG() – to test this function, a GUI needs to be created that can display a JPEG as well as a Bitmap image. The JPEG and Bitmap image will be loaded side-by-side to test to see whether the conversion was successful. First, the GUI will be tested with stationary images to make sure it is working. Following that, it will be tested using the actual MJPEG stream. If the Bitmap side is clearly shown, then this function will be deemed successful.

StopStream() – this function is very easy to test. Basically when this function is called, its job is to stop the connection between the C# code and the MJPEG stream. If no bytes are being received after this function is called, then it will be deemed successful.

## 7.10 GUI Testing

Testing for the GUI will be the simple process of trying to execute each command on the screen and seeing if the tank responds according to its description. Testing will be done in this fashion:

- Connect – First, the connect button will be used to see if a connection between the tank and laptop is properly formed.
- Tank: Forward, backward, left, right arrows – Each of the tank's movement arrows will then be tested to be sure that the tank moves in the proper direction based on the button pushed.
- Turret: Up, down, left, right arrows – Each of the turret's movement arrows will then be tested to be sure that the turret and camera pan in the proper direction based on the button pushed.
- Fire – When this button is pressed, it will be confirmed that the tank does in fact fire at its current target.
- Search – Once this button is pressed, the tank will be examined to ensure that it properly seeks out a target. This will be one of the harder aspects to test, as a problem could occur in either the GUI or the actual auto-search algorithm.
- Exit – Finally, this button will be pushed to ensure that the connection between the tank and laptop is dropped.

The last aspect of the GUI that will need to be examined is the video feedback. This will be tested by examining the picture produced on the laptop and comparing it to what the tank should be looking at. The tank will also be moved around using the directional arrows at this point to ensure that the video feedback is in real time.

Server errors may exist outside of the GUI's code that might cause an error to occur during the testing of the GUI. First, it is possible that the code written to send each command to the R/C tank might send an improper data sequence. Also, there may exist an issue with the connection between the laptop and WiFi Shield on the PCB. If commands are not being proper read and/or written to the server, then the GUI would not be at fault. Finally, there could exist an issue in the actual wiring of RADSAT where a wire may not be completely intact, or a misconnection might have been made between the microcontroller and the R/C tank's circuitry board. All of these issues will need to be looked at in the event that RADSAT failed to perform the specified command issued by the user through the GUI.

## 7.11 Wireless Communication Testing

As discussed earlier, an Arduino Diamondback prototyping board will be used in the initial design of RADSAT, while a PCB developed by the team will be used in the final design. Once the PCB that was designed for RADSAT is properly connected and the software that was developed by the team is properly installed, the PCB will undergo testing to ensure that the wireless chip embedded into the PCB properly connects to the laptop that will control RADSAT. This testing will commence as follows.

The first indication of whether or not a connection was properly established will be a little LED light on the PCB that turns on when the connection is made. This LED should turn on as soon as the team presses the “Connect” button on the GUI of the RADSAT. If this LED were to light up, it would be a promising indication for the team that a connection was in fact established before any testing even begins.

Once it is verified that the LED indicator light has actually lit up when the “Connection” button is pressed, the team will then begin to test out other commands from the GUI. First, all of the movement arrows will be pressed to ensure that the RADSAT moves in the correct direction. If this were the case, the team would continue with testing by pressing the “Search” button and monitoring the RADSAT to ensure that it properly searches out a target with accordance to its search algorithm. Once a target has been locked onto by the RADSAT, the team will test the “Fire” button to ensure the robot fires off a shot at the target.

Once all of the commands have been thoroughly tested, the team will test out the “Exit” button to ensure that it properly breaks the connection between RADSAT and the laptop. Once this button has been pushed, the connection light on the PCB should turn off. It will be verified that the connection has actually been broken by again pressing the movement arrows, “Search” button, and “Fire” button. At this point, it is expected that the RADSAT will not react at all to any of the commands.

With all of the tests that will be done to test the wireless connection, it will need to be ensured that the software programs written by the team are able to carry out their specified functions. If a command button on the GUI was to be pressed but R/C tank fails to response, it will need to be looked into whether this is a problem with the WiFi connection or with the code that was used to issue the command. Furthermore, it will need to be ensured that all connections made by wires from the microcontroller to the R/C tank are intact so that the signal can actually be transmitted.

Furthermore, as the commands will be written to a text file located on a server by the laptop and read off of the text file on the server by the Wifi Shield on the PCB, special attention will need to be paid towards which system has a failure.  If a command was not to be received by the R/C tank during testing, it may be because the laptop failed to write the command to the text file on the server. Another problem could have occurred when the WiFi Shield attempted to read the command off of the text file on the server.  It is also possible that a combination of these two errors may arise.  Whatever the case may be, all aspects of the wireless connection will need to be looked at while the connection is being tested.

The wireless connection between RADSAT and the laptop should be maintained at all times until it is broken by the user.  During this connection, the main loop of the C# code that is used to control the WiFi Shield will continuously run while it listens for a command to be issued by the user and written to the server.  If it is discovered that the connection is not always established during the testing phase, it will need to be investigated in order to see what can be done to maintain the connection.

# 8.0 Administrative Content

## 8.1 Milestone Discussion

In order to finish this project in a timely fashion during the two semesters allowed for Senior Design I and Senior Design II, the team discussed the milestones they hoped to accomplish for the project, and when they hoped to accomplish them. These milestones included an initial design and goals for the RADSAT, the completion of all necessary research, a final design of the RADSAT, the completion of all required paperwork and documentation, the acquisition of all necessary parts, the development of a prototype, the testing phase, and the final product. A discussion of the goals and reasoning for their deadlines can be found below.

**February 29, 2012 –** The milestone that the team hoped to accomplish by the end of February was to have an initial outline of all the goals and objectives of the robot that was to be designed. Having just come together as a team with only the slightest of ideas of what shape the robot would actually take, this first step was an important part of what would become the final outcome. Possible goals and objectives were discussed, and after a long deliberation were decided upon unanimously by the team. A name for the robot was developed, Reconnaissance and Demolition Super Attack Tank (RADSAT), and an initial design of what the robot would look like was created. With this milestone complete, by the end of February the project began to take on what would be the basis for the rest of the semester.

**Milestone met? –** Yes

**March 31, 2012 –** As the semester progressed, it was the goal of the team to make sure that all necessary research for the goals and objectives of the RADSAT was completed. Knowing what the team hoped to accomplish meant nothing if they were not able to thoroughly understand all of the aspects and work that would go into the RADSAT. Many topics that required research included, but is not limited to, tank design, turret design, video processing, autonomous design, power allocation, color recognition, targeting, sensors, voice control, WiFi control, microcontrollers, etc. With all of this research, the team knew that it would have to spend countless hours searching for related topics online and how these technologies have been implemented before. Upon completion of this research, the team had a much broader idea of what they were actually doing and how the RADSAT was going to be developed.

**Milestone met? –** Yes

**March 31, 2012 –** With all of the research that had to be completed, the team knew that the design of the RADSAT would ultimately have to be changed. With all of the knowledge that the team learned, the goals and objectives of the RADSAT would be reexamined to ensure that the final product would be what the team envisioned. Every aspect of the RADSAT was relooked at, and a final design was decided upon unanimously by the team. This design was what the team would continue to research so that it could be developed as a final product by the team.

**Milestone met? –** Yes

**April 23, 2012 –** With the research completed and the final design agreed upon, it was now up to the team to ensure that all of the paperwork and documentation for the RADSAT was properly completed. To be turned in on this date was a 120+ page document that outlined all of the research and design that the team had accomplished, as well as instructions on how the RADSAT would be developed and tested during Senior Design II. Each member of the team was expected to contribute at least 30 pages to the document that would be turned in at the end of the semester. As a requirement for the completion of Senior Design I, this milestone was taken very seriously by the team, and each member began producing pages for the document as soon as possible.

**Milestone met? –** No

**May 14, 2012 –** With the completion of Senior Design I, it was the team's hopes that they would have all necessary parts for the development of the RADSAT acquired as soon as Senior Design II began. These parts, which would be determined during the course of research in Senior Design I, would be a necessary aspect for the rest of the semester. It was decided that acquiring the parts earlier would be better, and the team could begin to focus on the development of the RADSAT during the rest of the semester.

**Milestone met? –** Yes

**May 31, 2012 –** With all of the parts acquired, the team decided upon developing an initial prototype of the RADSAT as soon as possible. This would be done within the first two weeks of acquiring all of the necessary parts. Once and initial prototype was developed, the team would be able to carry on the rest of the semester with testing and developing what would be the final product for the committee board to review.

**Milestone met? –** Yes

**June 30, 2012 –** With a prototype developed, it was understood by the team that a large window would be required for the testing phase. During this stage, all goals and objectives that was hoped to be accomplished in the development of the RADSAT would be tested. Each command and function that the RADSAT was to carry out would be examined by the team. Ideally, the prototype would be able to carry out most, if not all, of its specified functions. The team was aware, however, that errors would most likely arise during the RADSAT's development, and some of the technologies might need to be reexamined. All of these errors would be inspected and recorded thoroughly by the team so that they could be corrected.

**Milestone met? –** Yes

**July 14, 2012 –** After the testing phase has been completed, it will be required for the team to reconstruct the RADSAT as a final product will all of the hardware and software issues debugged. This goal was set for mid July so that the team might have an ample amount of time to finish the project before the final few weeks of classes. As this final product will be what the team is graded on for Senior Design II, it is the most important milestone for the team to meet in a timely fashion.

**Milestone met? –** Yes

**July 28, 2012 –** After completing the final product, the team will need to have completed all of the final paperwork required for Senior Design II. Furthermore, the team will need to develop a presentation of the project so that they can present to the review board that will examine their project. The two weeks allotted for this documentation and paperwork should be ample time for the team to meet this milestone.

**Milestone met? –** Yes

**Date TBD –** As the final milestone for the team, the RADSAT will need to be presented to a review board. This review board will examine the RADSAT to see that the team was in fact successful in accomplishing all of the goals that were outlined for the RADSAT. This milestone is dependent on the successful completion of every other milestone by the team, and will be what determines the final grade of the project. Being successful during this presentation will mean that the team passes Senior Design II and is capable of graduating. Failure, however, will require the team to retake Senior Design II during the next semester, stalling graduation.

**Milestone met? –** Yes

With all of the milestones and objectives discussed and agreed upon by the team, it is now each individual's responsibility to ensure that these milestones are in fact met. The completion of each and every goal is necessary for the successful completion of both Senior Design I and Senior Design II. Failure to meet these goals may result in a delay of the project, which might ultimately result in the failure of one or both of these classes. To ensure that each team member is on track to complete their necessary work for the RADSAT, they will be held accountable by all other team members. In this way, the team will work together to make sure that every goal is met and the RADSAT successfully progresses as scheduled.

*Table 8.1.1* below summarizes all of the milestones that have been discussed above, as well as when they are to be completed and whether or not they were in fact completed on time:

| **Date** | **Milestone** | **Met?** |
|---|---|---|
| 2/29/2012 | Initial design completed | Yes |
| 3/31/2012 | Completion of all necessary research | Yes |
| 4/23/2012 | Completion of all necessary paperwork and documentation | No |
| 5/14/2012 | Acquisition of all necessary parts | Yes |
| 5/31/2012 | Prototype developed | Yes |
| 6/30/2012 | All necessary testing and debugging completed | Yes |
| 7/14/2012 | Final product developed | Yes |
| 7/28/2012 | All necessary paperwork, documentation, and presentation completed | Yes |
| TBD | Presentation before review board completed | Yes |

*Table 8.1.1: Milestones*

## 8.2 Budget  and Finance Discussion

### 8.2.1 Bill of Materials

The bill of materials lists the part name, part number (if available), the supplier, the quantity necessary, the unit price and the total price. The total price may not include shipping and handling (Figure 8.2.1).

| Part Name | Part # | Supplier | Quantity | Unit Price | Total Price |
|---|---|---|---|---|---|
| **Hitec Servo Motor 1** | HS-422 | Servocity.com | 1 | 10.00 | 10.00 |
| **Hitec Servo Motor 2** | HS-485HB | Servocity.com | 1 | 17.00 | 17.00 |
| **Arduino DiamondBack Testboard** | 610074725596 | Cutedigi.com | 1 | 73.00 | 73.00 |
| **Netgear WGR614v9** | | Already Owned | 1 | 0.00 | 0.00 |
| **WansView WiFi Security Camera** | B003LNZ1L6 | Amazon.com | 1 | 56.10 | 56.10 |
| **Maxbotix XL-Maxsonar EZ1** | MB1210 | Sparkfun Electronics | 1 | 49.95 | 49.95 |
| **Sharp IR Distance Sensor** | GP2Y0A02YK | Trossen Robotics | 3 | 15.00 | 45.00 |
| **Sharp IR Sensor to Servo Cable** | GP2D12 | Trossen Robotics | 3 | 1.95 | 5.85 |
| **Airsoft gun** | | Already Owned | 1 | 0.00 | 0.00 |
| **Snow Leopard R/C tank** | | Yourrcstore.com | 1 | 75.44 | 91.19 |
| **Pan and tilt attachment** | PT-2 | Endurance-robotics.com | 1 | 30.00 | 30.00 |
| **7.2V 3800mAh NiMH batteries+charger** | battery-superstore.com | 1 | 58.00 | 66.00 | |
| **PCB fabrication** | | dorkbotpdx.org | 3 | ~14.00 | 42.00 |
| **Switched Voltage Regulator** | LM2574 | Ti.com | 6 | 0.00 | 0.00 |
| **Resistors, capacitors, inductors** | | lab | -- | 0.00 | 0.00 |
| **DPDT 6V 2A PCB Relay** | HFD2-06 | futurlec.com | 2.00 | 0.80 | 1.60 |
| **WiFi Bee** | | | 1.00 | | 89.49 |
| **Arduino Uno Board** | | | 1.00 | | 19.95 |
| **Compass Sensor** | | Sparkfun Electronics | 1.00 | | 53.59 |
| **Breadboard + relays** | | | -- | | 21.30 |

|  |  |
|---|---|
| **Miscellaneous** | 35.00 |
| **Total Price** | 707.02 |

*Figure 8.2.1: Bill of Materials.*

### 8.2.2 Financing Issues

Without a major source of outside financing, deliberating on the best possible method on both keeping costs down and paying for RADSAT became a rather long and contentious discussion. Initially, the RADSAT team discussed the possibility of outlining a general budget, dividing that budget equally among all group members, and simply pooling the money into one large mass of liquidity. The most blatant disadvantage using a pool of funds is that it requires a very high level of trust and responsibility among group members. It also required another level of bureaucracy in the form of either a joint bank account, or if a single group member were entrusted with the funds, some kind of a decision process and a way to deliver balance statements would have to be established.

The team's solution to the above problems was to allocate the responsibility of buying all necessary parts for each subsection to the group member currently working on that subsection. After the final bill of materials was calculated, the total cost was divided by 4, and each group member either contributed more or was reimbursed for overpaying.

## 9.0 Post Implementation

## 9.1 High Level Software Design

RADSAT uses both high level and low level code in conjunction. This section details all of the high level code which was used. The programming language was C# and covered all server communication as well as color and shape recognition. The design of the high level code changed a great deal from the planning stage to the implementation stage. This section details how the final code changed from the planned code, and many problems, as well as the solution to these problems. The class diagram can be viewed in Figure 9.1.1.

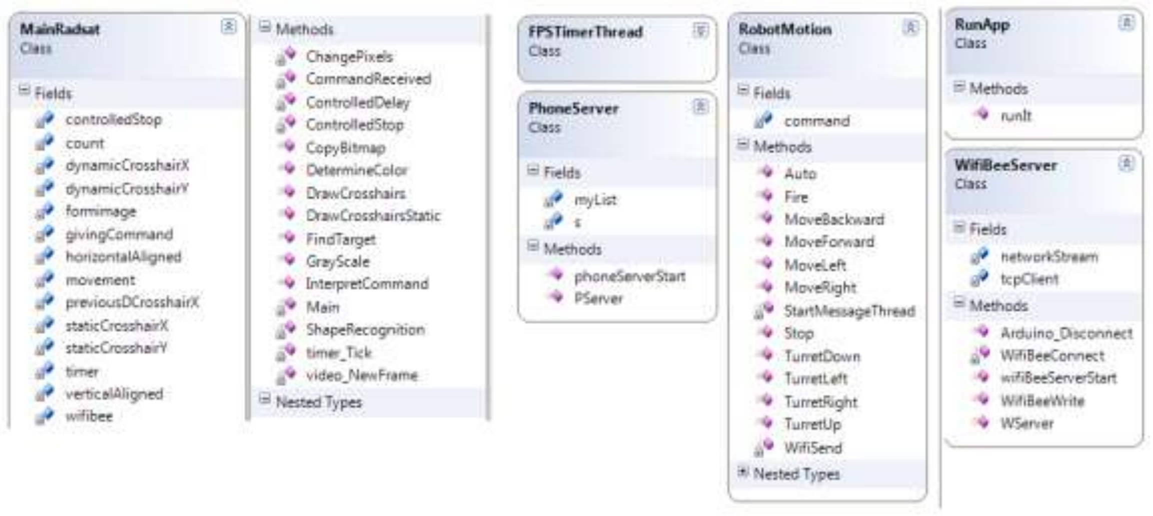


*Figure 9.1.1 Class Diagram*

### 9.1.1 Color Recognition

Color recognition works by checking every single pixel in a bitmap image and testing if the selected pixel fits within the boundaries for the desired pixel. The function “ChangePixels” does exactly this. However, it has to accomplish this task in a non-intuitive way in order to maintain acceptable speeds. For instance, using C#’s “SetPixel” and “GetPixel” functions will result in extremely slow, unacceptable speeds.

In order to obtain acceptable speeds the program copies the pixels from the bitmap and saves all the RGB values into a byte array. The program is able to very quickly cycle through the array because of the architecture an array innately has. The format of the array is for pixel one: array[0] = Red, array[1] = Green, and array[2] = Blue. This format repeats itself for every single pixel within the bitmap. The values for red, green, and blue is a number from 0 to 255.

Each pixel would be inserted into a formula to check whether its values were close to the desired color, and whether or not it can be considered for the target. The exact formula is as follows:

```
D = Math.Sqrt((R-red)*(R-red) + (G-green)*(G-green) + (B-blue)*(B-blue));
```

D represents the threshold for how close the pixel is to the desired color. Generally the most appropriate threshold is around 30. This number filters out almost all of the junk-colors that the camera might be seeing that is unrelated to the target. Also, the number 30 is large enough to allow for slight lighting differences which can occur on the target. Through all of the testing it would identify something which was not the target only 1 out of 30 times. Each pixel which matches remains its color, and every pixel that does not match is colored black. The case where the background is colored black can be seen on the right side of Figure 9.1.2.

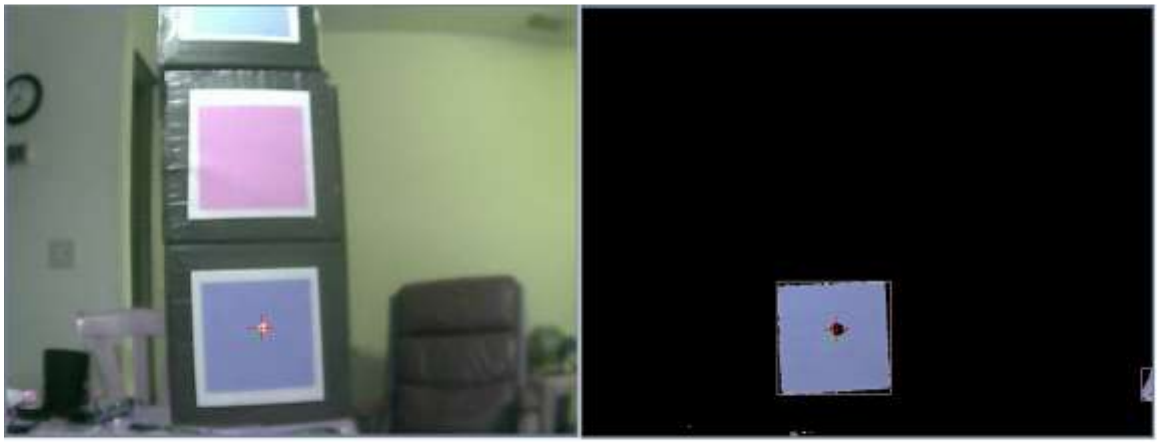


*Figure 9.1.2 Regular Vision (left) vs Robot Vision (right)*

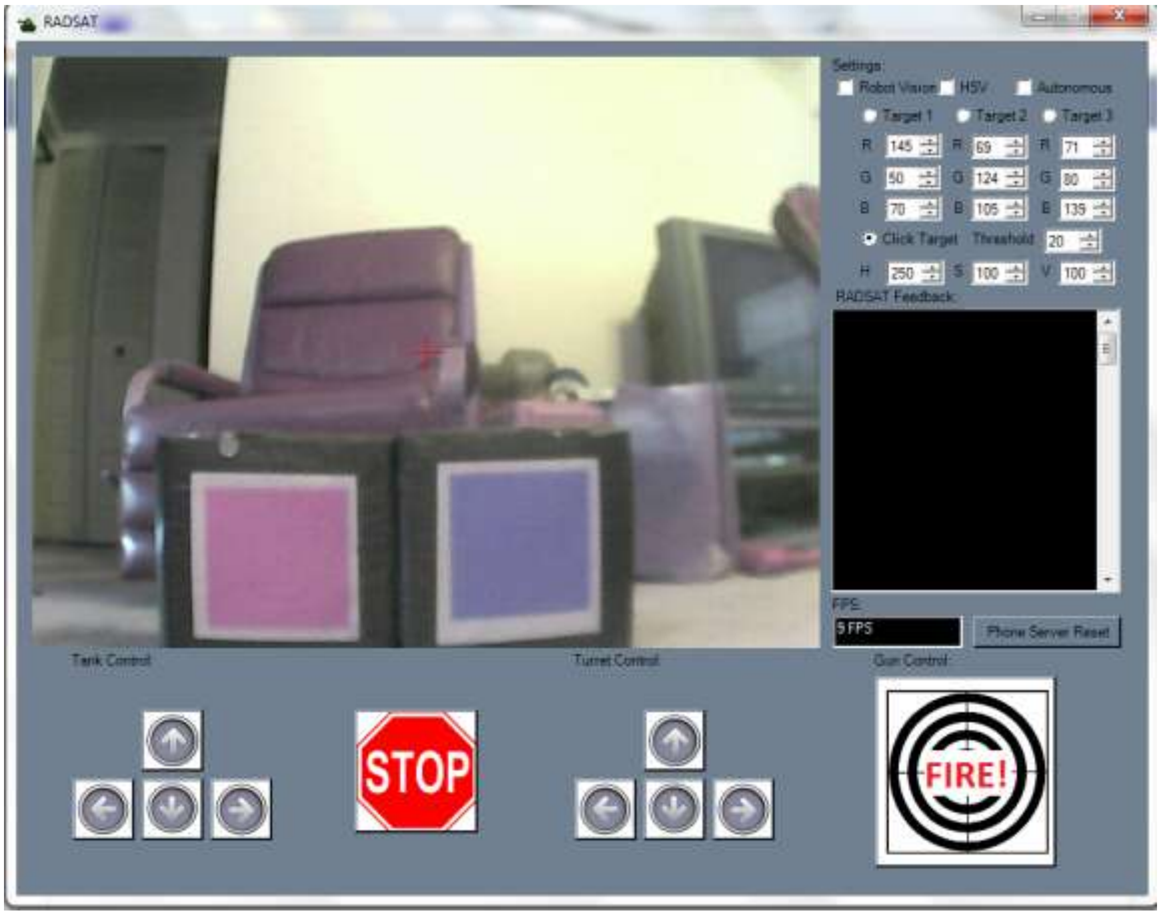
In some situations, where the lighting is particularly bad, the above formula is not good enough. So, there's an option which can be turned on and off within the GUI which will enable the program to also check the pixels HSV values to the desired color's HSV values. A situation where it might be imperative to use this option is when the lighting is very bad in the location of the target. The HSV values will still recognize the color as being the same even with poor lighting, as it determines matches differently from RGB. However, this option greatly slows the program. The program will go from 12-16 FPS down to 4-6 FPS, which will impair the targeting program greatly. If the program is running in a very fast environment where there is no noticeable difference between the speeds when HSV is checked, then it is recommended to use the HSV values.

## 9.1.2 FPS

The FPS class does exactly what would be expected of it. It simply logs the FPS the video stream is currently running at. The process to accomplish this is very simple. Every time a new frame event is created a counter increases by one. Then a timer which goes off every second reads in the value of the counter then prints that value to the FPS section of the GUI. After printing the FPS value, the counter is then set to zero. This process is repeated perpetually.

## 9.1.3 GUI

The GUI is the gateway which allows the user to view, and control everything about RADSAT. It can display errors, switch targets, control the turret and tank, display the MJPEG stream, and several other things. It was configured to be as easy to read and understand as possible. The GUI can be seen in Figure 9.1.3.



| | Target 1 | Target 2 | Target 3 |
|---|---|---|---|
| R | 145 | 69 | 71 |
| G | 50 | 124 | 80 |
| B | 70 | 105 | 135 |

Click Target   Threshold 20

H 250   S 100   V 100

FPS: 9 FPS

*Figure 9.1.3. GUI*

### 9.1.4 Main

“MainRadsat” is the class which encompasses all other classes, except the GUI. The reasoning for this is because almost every other function or class within the program edit the bitmap image of the video stream, and add to the same GUI object. Having one class with shared values makes it easy for the various threads running within the program to access the GUI and bitmap. This class is also responsible to activating said threads. This is a very important process which allows the program to continue running, even if one section of it fails. The different threads include: GUI, Phone Server, WiFi Bee Client, FPS, and the frame event handler.

### 9.1.5 Phone Server

The Phone Server class creates a server which the android phone communicates with. The server is TCP based and will only receive data from the phone, and never send data back. The server will continue to wait, until it receives a connection from the phone. Once a connection is made, the program will enter a continuous loop, and try to receive commands. If the connection between the Phone Server and phone is somehow interrupted, the only way to reconnect to the phone is to hit the “Phone Server Reset” button located on the GUI. Once this button is pressed the program will once again wait until a connection is made with the phone.

### 9.1.6 Robot Motion

Robot Motion is responsible for sending all the commands to either the tank or the camera. Both the tank and camera communicate with the user via a TCP connection established within WiFi Bee Client. Targeting, as well as many functions from the GUI, uses this class to move the camera or the tank. The commands as well as their protocols can be seen in Figure 9.1.4. The function is the function name in the program, the protocol is what is sent to the tank or camera, and the command is what is printed to the GUI as being sent.

| Function | Protocol | Command | Description |
|---|---|---|---|
| MoveForward() | 7 | up | Tank moves forward |
| MoveBackward() | 8 | down | Tank moves backward |
| MoveLeft() | 9 | left | Tank moves left |
| MoveRight() | A | right | Tank moves right |
| TurretUp() | 0 | tup | Turret moves up |
| TurretDown() | 1 | tdown | Turret moves down |
| TurretLeft() | 2 | tleft | Turret moves left |
| TurretRight() | 3 | tright | Turret moves right |
| Fire() | 4 | fire | Turns on the laser pointer |
| Stop() | 5 | stop | Stops all activity from the tank and camera |
| Auto() | 6 | auto | Activates autonomous mode for the tank and camera |

*Figure 9.1.4 Robot Motion commands*

## 9.1.7 Shape Recognition

Shape Recognition helps to narrow down the possible target from the several different clusters of pixels which are comprised of matching pixels. The filtering process begins with the AForge.net blob library. The AForge.net blob library needs a bitmap image which has images contrasting on a black background. This image is obtained from the “ChangePixels” class.

The library works by isolating different cluster of pixels which are unbroken by the black background. Within Robot Vision, the user can clearly see the different regions obtained through shape recognition. Looking at Figure 9.1.2 it is clear the regions because they are seen as pink rectangles surrounding different objects. After those rectangles are obtained, the next step is to filter out any rectangles that are not the target. There are three different conditions that the rectangle must meet in order for the program to determine that it is in fact the target.

1. The rectangle’s length and width must be within 10 pixels of each other. Because the target is a square, this step will eliminate any target which is a different shape.
2. The percentage of matching pixels must be over 65%. This ensures the rectangle is comprised almost entirely of matching pixels. The value 65% is enough to allow for light differences, but will also disallow most non-targets.
3. The length of the rectangle must be over 50 pixels. This will eliminate any tiny shapes which might meet the other conditions solely because of their tiny size.

## 9.1.8 Targeting

The targeting class takes care of moving the camera servo's into alignment with the target. The targeting process begins with drawing crosshairs onto the screen. There are two crosshairs, the static crosshair, and the dynamic crosshair. The static crosshair represents the place the laser pointer shines when fired. It can be seen as the red crosshair in Figure 9.1.2. The dynamic crosshair represents the center of the target. It can be seen as the green crosshair in Figure 9.1.2. The goal is to align the static and dynamic crosshairs.

Targeting is activated whenever autonomous mode is selected in the GUI. The camera servos pan back and forth and try to find a match with the selected targets. Once the target is found the dynamic cross hair appears and the program calculates the horizontal and vertical distance between the dynamic and static crosshair. If the horizontal distance is above 15 pixels then the program will move the turret in the correct direction. The time between the move and stop command is a formula based which takes the number of pixels between the two, multiplies it by two, then a stop command is sent after (difference of pixels)*2 milliseconds. Once the horizontal is aligned, the vertical goes through the same process to become aligned. Once both are aligned the message "Target Aligned" will be printed in the Robot Feedback section of the GUI. The user now has to option to either send a "Fire" command or to search for a different target.

## 9.1.9 Video Processing

This section uses the AForge.net MJPEGStream library in order to retrieve the MJPEG stream from the WiFi security camera. The MJPEGStream library finds the stream at the location "http://192.168.1.178/videostream.cgi?user=RADSATeye&pwd=radical" and parses the MJPEG stream to individual JPEGs which is then converted to a Bitmap. Once the Bitmap is obtained the program is able to call all the functions which edit and operate according to, the bitmap image.

## 9.1.10 WiFi Bee Client

The WiFi Bee Client is what establishes the connection between the computer and the tank/camera. It connects to the TCP server which is created on the WiFi bee located on the tank/camera. The TCP connection allows for minimal lag time for communication between the components. If the server is disconnected or interrupted, the connection will become automatically reestablished as soon as available. This class sends the commands according to Figure 9.1.4.

## 9.2 Sensor and Microcontroller Overview

### 9.2.1 Dual Controller Architecture

RADSAT uses two ATmega 328P microcontroller units. One 328P is embedded on the WiFi Bee wireless module (which also includes a Microchip IEEE 802.11 Wi-Fi transceiver module). Through testing and trial and error, it was found that supporting wireless functionality while interfacing with multiple motors, servos, and sensors was too strenuous of a task for a single MCU to handle, so another 328P was added to the design of RADSAT to handle the extra load. RADSAT originally used the WiFi Bee to listen for HTTP GET requests from the main server to handle requests and commands. Although it was much simpler to implement, it turned out to be much more inefficient, slower, and more demanding than using TCP sockets, and it was decided that the switch to sockets was warranted. The improved approach allows for relatively low latency of less than 50ms versus potentially hundreds of milliseconds using HTTP requests. As shown in figure 9.2.1, the WiFi Bee's job is to receive incoming requests and forward them to the "main" MCU's digital I/O pins (pins 16-19).

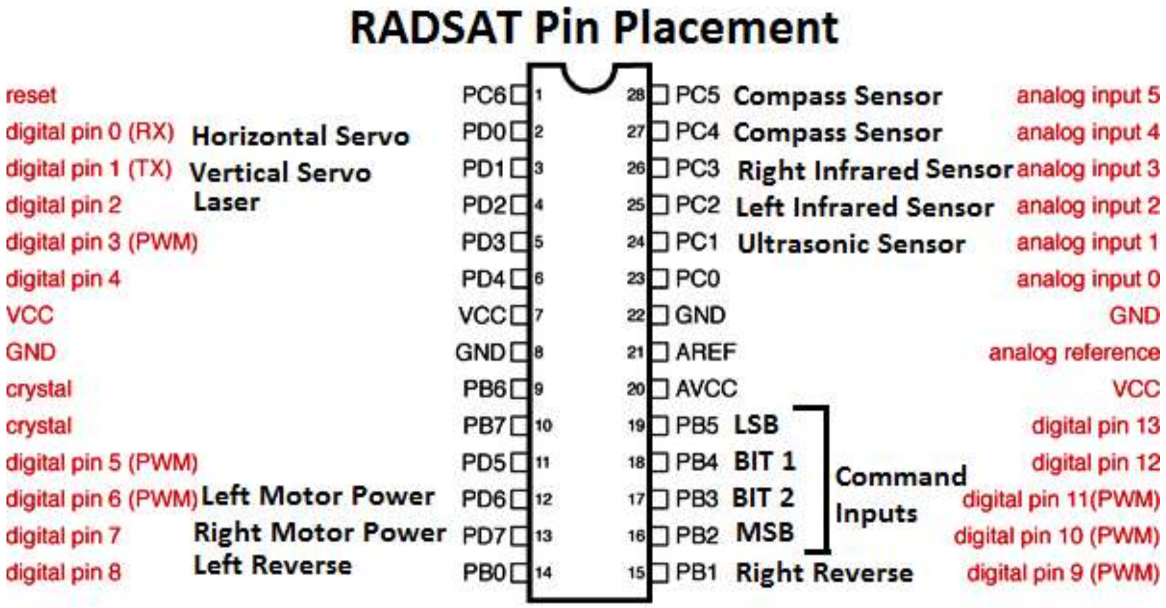


*Figure 9.2.1: Arduino to ATmega 328P pin mapping diagram. Also displayed are RADSAT's use for each pin.*

The main MCU is responsible for all of RADSAT's movement and anything not related to its communications. When a commanded is forwarded to it (all valid commands are listed in table 9.1.4), it is generally supposed to break out of any loop and handle the command. The first four commands manually control the motion of the servos and increments or decrements them in their respective directions, continuously until a stop command is received. For safety reasons, RADSAT is restricted from being able to autonomously fire and is only able to fire when the proper command is received. The stop command stops any movement RADSAT is engaged in, including servo rotation. RADSAT is able to run in both autonomous mode and in manual mode. In order to enter autonomous mode, the autonomous mode command is sent (hex 7 or binary 0111). In autonomous mode, RADSAT is able to make its own decisions regarding directions to turn while scanning the area for its target. The final four commands deal with the manual movement of RADSAT.

### 9.2.2 Sensor Overview

As previously stated in this document, RADSAT uses a total of three different proximity sensors. The forward sensor is the MB1210 ultrasonic sensor while the side sensors are GP2Y0A02YK infrared sensors. The ultrasonic sensor is mounted on the front of the tank and is able to successfully inform RADSAT of whether or not an obstruction that would generally be undetectable to infrared sensors is present. RADSAT is designed to move in a single direction until an obstruction is found in front of it. At that point, a comparison is done between the value returned by the left infrared sensor and the right and RADSAT turns such that it approaches the side with the furthest obstruction. With the help of an HMC6352 compass module, RADSAT will attempt to rotate 90 degree angles unless an obstruction is still detected by the ultrasonic sensor, at which point RADSAT will continue to rotate. Once the rotation is complete, RADSAT will attempt to move forward for approximately three seconds, and move in a direction that is opposite its original direction.

## 9.3 Circuit Design Changes

Overall the basic design of the circuit in the actual implementation did not change much from the original design covered earlier in this paper. The only real changes were the addition of a microcontroller, the change from the xbee WiFi to the WiFi bee, the removal of some unnecessary parts and a few transistors used as switches. Also, in the final demo of the project, the camera and tank had to be separated due to certain circumstances.

During the first semester research and development phase of the project it was decided that a tank with a turret-mounted airsoft gun would be built. It would be a single stand-alone unit with capabilities of moving and locating any targeted item. It would look like the following picture:

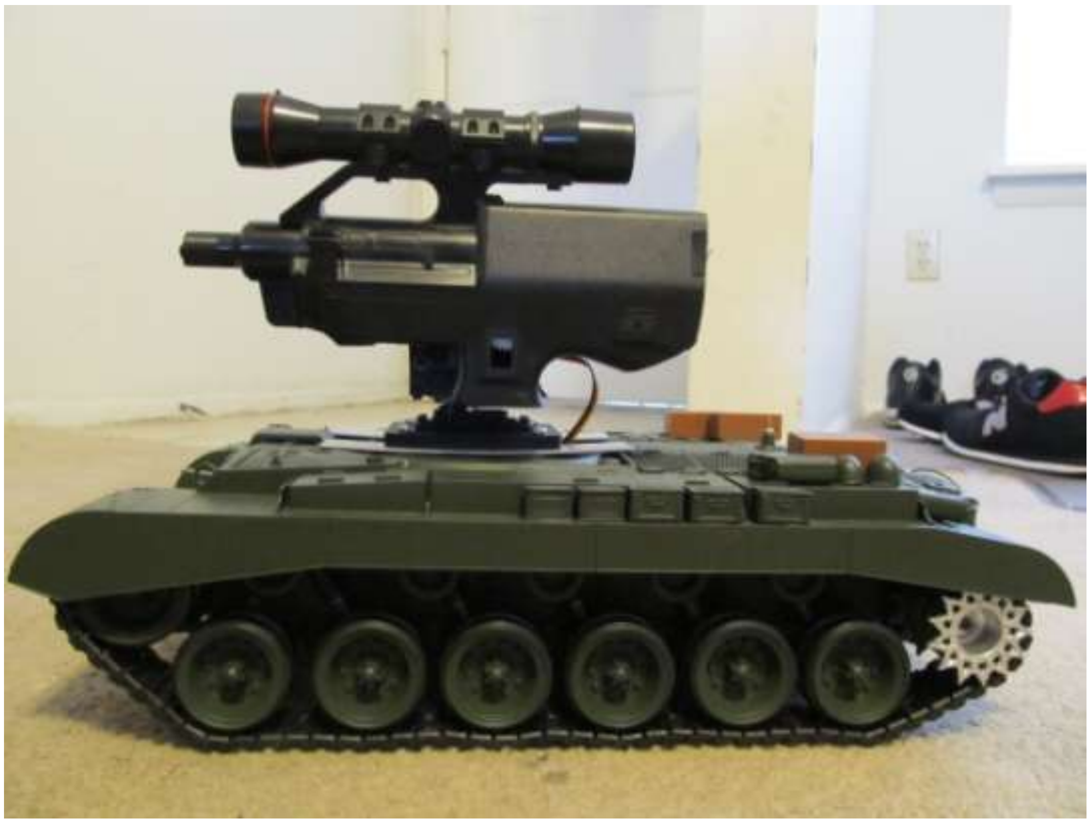


*Figure 9.3.1: Early model of a RADSAT implementation (without camera)*

Early on, this idea was changed with the removal of the airsoft gun. The gun was scrapped in favor of a laser pointer as its symbolic analog. The initial reason for the change was that upon mounting the airsoft gun it was found that mounting it distorted the shape of the body of the gun and caused the gears inside it not to mesh correctly. Finding another method to mount it would have solved this problem, but there was another reason to change it: a laser pointer would suffice to show that the target could be hit, whereas firing airsoft BB's during a presentation was not an option in the setting of the RADSAT presentation.

The next change came much later in the construction of the final product. It was the decision that the tank should be independent from the turret and camera. Thus there would be two distinct assemblies as shown in the following concept sketches:

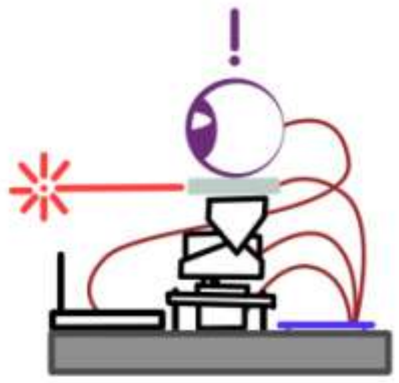


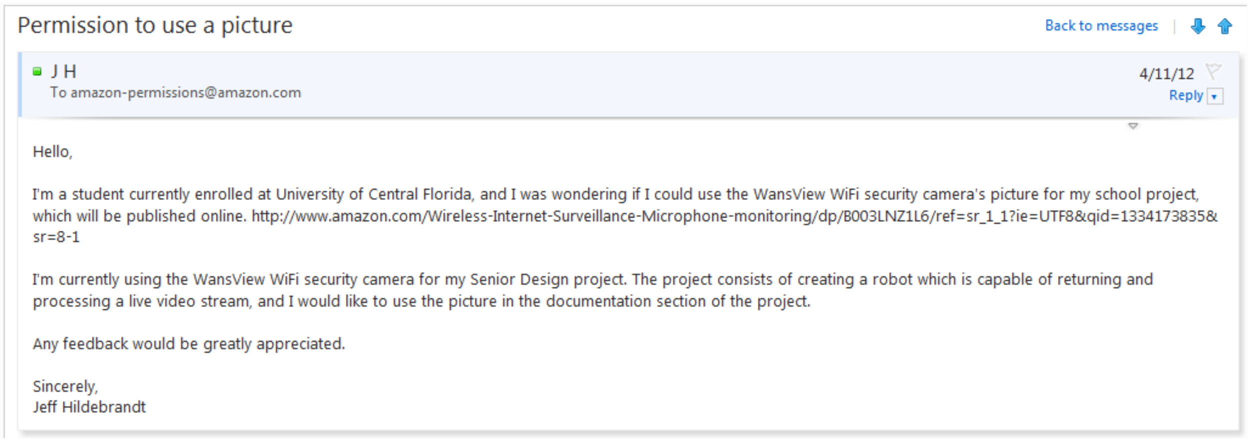
*Figure 9.3.2: Tank with target* *Figure 9.3.3 Turret with camera and laser*

The primary reason for this change was that the tank chassis did not have enough room inside for the wires connecting to the servo assembly and sensors. It was problematic because putting all the wires into a compact space led to more probability of a sensor wire or servo wire being unplugged. Another reason for the change was that the power to the camera needs to be reliable and with the turret and tank parts together it occasionally had problems. The final reason is that as the tank moved it vibrated and caused the servo to move unpredictably, causing instability.

All together there are two atmega328p microcontrollers controlling the circuit. One of them is located on the WiFi bee, and the other was mounted on the board. A few components needed to be added on the board to allow the bare microcontroller to work. These include a 16 MHz clock attached to pins 9 and 10 of the microcontroller as well as two 22 pF capacitors, one going from pin 9 to ground and the other going from pin 10 to ground, and a 10k resistor going from the reset pin to VCC+. The idea here was to keep a constant voltage on the pin, and avoid fluctuations which caused the chip to reset unnecessarily.

The parts that were removed from the original design were removed either because other changes to the circuit warranted their removal, or the function they were initially intended for was no longer relevant due to changes in the design. The DPDT relays with the purpose of driving the motors backwards were on the final circuit, but left unused because within the bounds of the autonomous navigation system there was never a need for using reverse. The tank could make small enough U-turns to be able to change direction when needed, and the supersonic sensor mounted on the front prevented the tank from running into a wall or impassable object. The other circuit components that became obsolete were the power supplies to the servos, the control signal outputs to the servos, and the transistor that sent power to the laser to turn the laser on.

# Appendix: Permissions and Licenses



Permission to use a picture

Back to messages

J H
To amazon-permissions@amazon.com

4/11/12
Reply

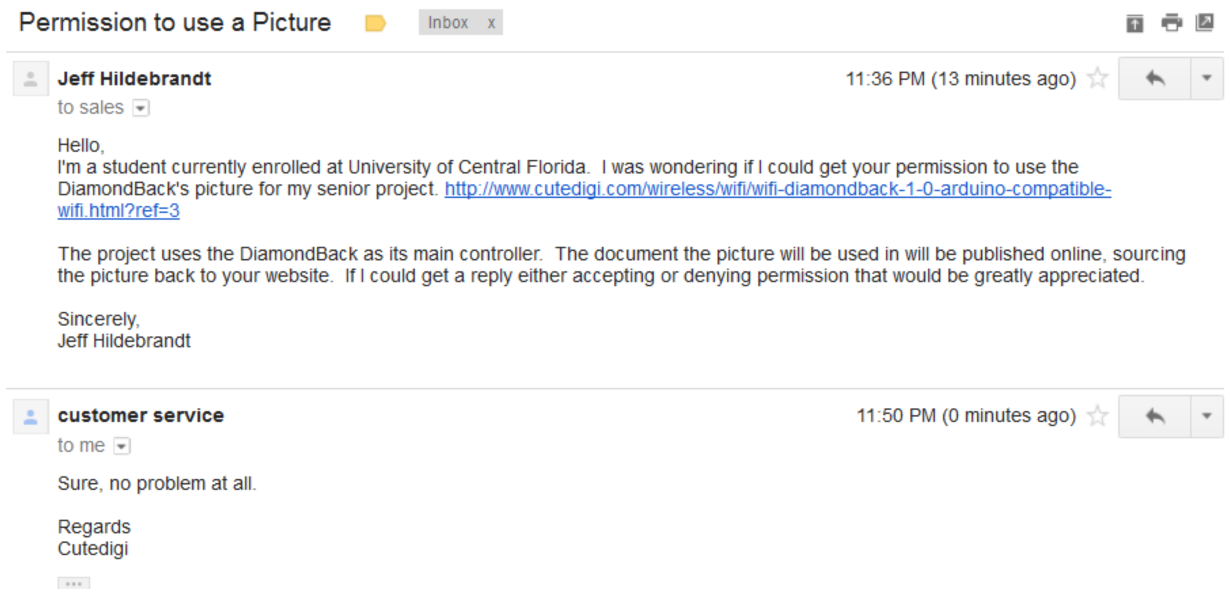
Hello,

I'm a student currently enrolled at University of Central Florida, and I was wondering if I could use the WansView WiFi security camera's picture for my school project, which will be published online. http://www.amazon.com/Wireless-Internet-Surveillance-Microphone-monitoring/dp/B003LNZ1L6/ref=sr_1_1?ie=UTF8&qid=1334173835&sr=8-1

I'm currently using the WansView WiFi security camera for my Senior Design project. The project consists of creating a robot which is capable of returning and processing a live video stream, and I would like to use the picture in the documentation section of the project.

Any feedback would be greatly appreciated.

Sincerely,
Jeff Hildebrandt



Permission to use a Picture

Inbox x

**Jeff Hildebrandt**
to sales

11:36 PM (13 minutes ago)

Hello,
I'm a student currently enrolled at University of Central Florida. I was wondering if I could get your permission to use the DiamondBack's picture for my senior project. http://www.cutedigi.com/wireless/wifi/wifi-diamondback-1-0-arduino-compatible-wifi.html?ref=3

The project uses the DiamondBack as its main controller. The document the picture will be used in will be published online, sourcing the picture back to your website. If I could get a reply either accepting or denying permission that would be greatly appreciated.

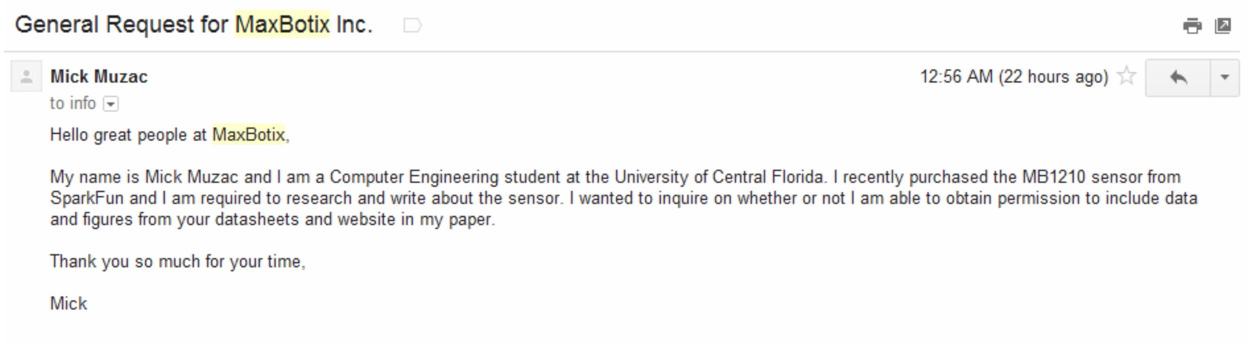
Sincerely,
Jeff Hildebrandt

**customer service**
to me

11:50 PM (0 minutes ago)

Sure, no problem at all.

Regards
Cutedigi



General Request for MaxBotix Inc.

**Mick Muzac**
to info

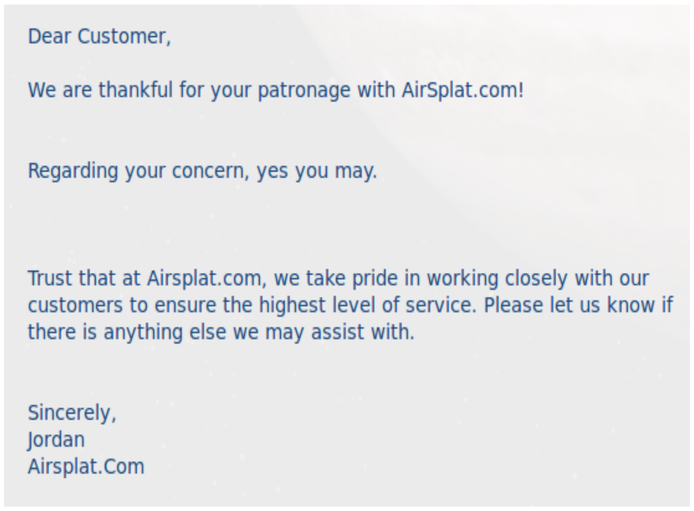
12:56 AM (22 hours ago)

Hello great people at MaxBotix,

My name is Mick Muzac and I am a Computer Engineering student at the University of Central Florida. I recently purchased the MB1210 sensor from SparkFun and I am required to research and write about the sensor. I wanted to inquire on whether or not I am able to obtain permission to include data and figures from your datasheets and website in my paper.

Thank you so much for your time,

Mick



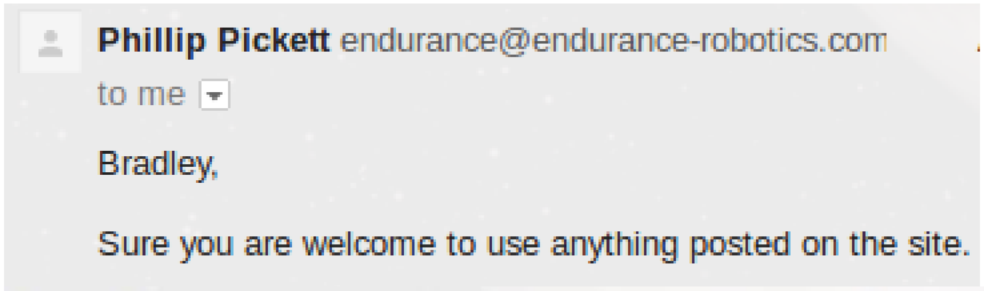
Dear Customer,

We are thankful for your patronage with AirSplat.com!

Regarding your concern, yes you may.

Trust that at Airsplat.com, we take pride in working closely with our customers to ensure the highest level of service. Please let us know if there is anything else we may assist with.

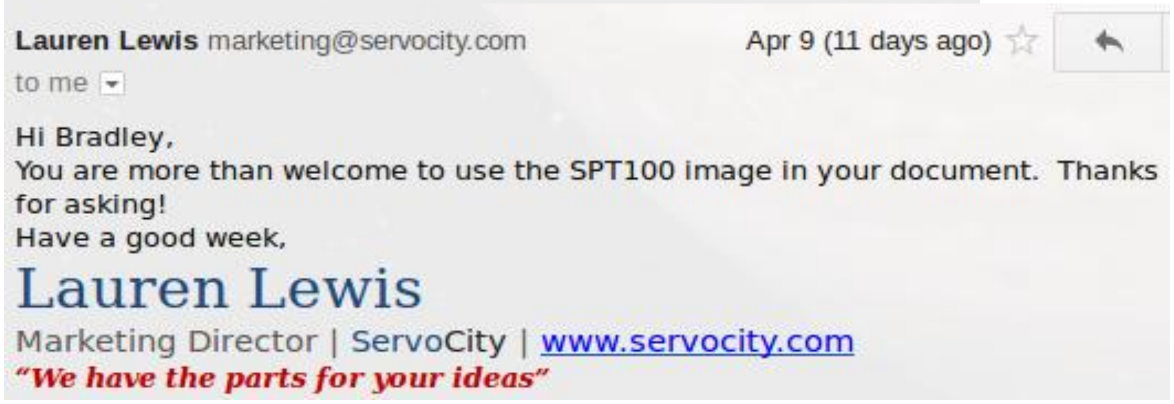
Sincerely,
Jordan
Airsplat.Com



**Phillip Pickett** endurance@endurance-robotics.com
to me

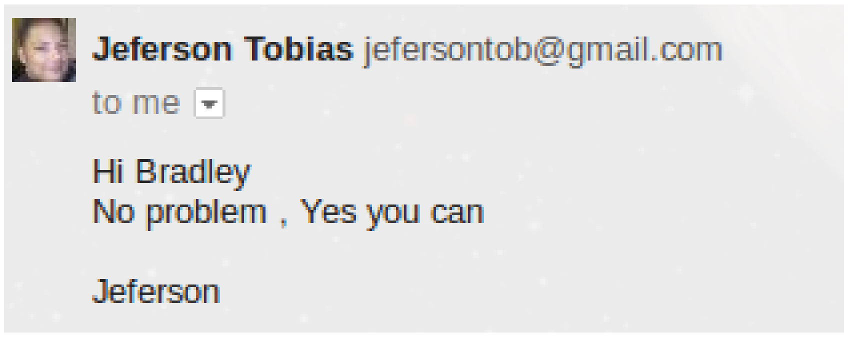
Bradley,

Sure you are welcome to use anything posted on the site.



**Lauren Lewis** marketing@servocity.com Apr 9 (11 days ago)
to me

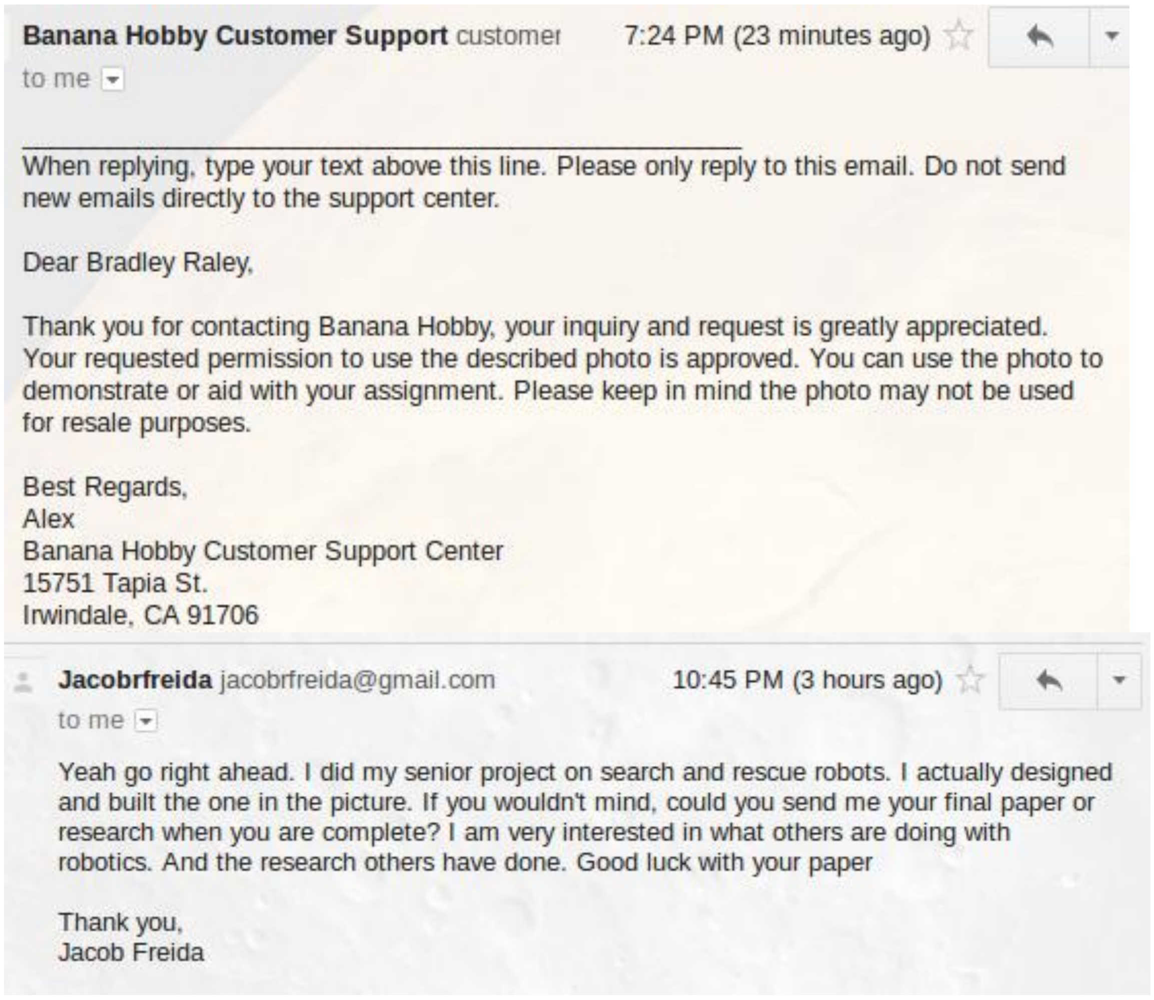
Hi Bradley,
You are more than welcome to use the SPT100 image in your document. Thanks for asking!
Have a good week,
Lauren Lewis
Marketing Director | ServoCity | www.servocity.com
*"We have the parts for your ideas"*



**Jeferson Tobias** jefersontob@gmail.com
to me

Hi Bradley
No problem , Yes you can

Jeferson



**Banana Hobby Customer Support** customer 7:24 PM (23 minutes ago)
to me

______________________________________________
When replying, type your text above this line. Please only reply to this email. Do not send new emails directly to the support center.

Dear Bradley Raley,

Thank you for contacting Banana Hobby, your inquiry and request is greatly appreciated. Your requested permission to use the described photo is approved. You can use the photo to demonstrate or aid with your assignment. Please keep in mind the photo may not be used for resale purposes.

Best Regards,
Alex
Banana Hobby Customer Support Center
15751 Tapia St.
Irwindale, CA 91706

**Jacobrfreida** jacobrfreida@gmail.com 10:45 PM (3 hours ago)
to me

Yeah go right ahead. I did my senior project on search and rescue robots. I actually designed and built the one in the picture. If you wouldn't mind, could you send me your final paper or research when you are complete? I am very interested in what others are doing with robotics. And the research others have done. Good luck with your paper

Thank you,
Jacob Freida

## License

Many of the images in this document are licensed by and are clearly captioned with CC BY-NC-SA 3.0. The license, in its entirety is printed below.