RADSAT (Reconnaissance And Demolition Super Attack Tank)

Jeff Hildebrandt, Bradley Raley, Mick Muzac, Dylan Lambe

School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450, USA

***Abstract* — This paper covers the final design of the Reconnaissance and Demolition Super Attack Tank, A.K.A RADSAT. RADSAT is a semi-autonomous tank designed to search out a specified target based on its color and shape. The autonomous movement of the tank and its turret is controlled by two separate microcontrollers. While not in autonomous mode, RADSAT can also be remotely controlled by a user via a laptop and GUI over a network. A live video stream from the tank can also be viewed on this GUI. Once a target has been determined and locked on, the user has the option of firing off a tank "round" at the target.**

***Index Terms* — Autonomous, color recognition, GUI, microcontrollers, MJPEG stream, RC-tank, sensors, servo, shape recognition, video feedback, voice control.**

I. Introduction

As unmanned vehicles are becoming more popular in today's warfare, the goals of RADSAT, though small scale, are comparable to those being applied today by government agencies. As autonomous behavior allows for the prevention of human casualty, this is an important issue for today's military commanders who want to avoid the unnecessary loss of a human life at all costs. RADSAT itself is a prototype of the technology that could be applied to real life tanks.

The goals of RADSAT are to apply this technology to an R/C tank that can search out "enemy targets" without the assistance of human input. Given a specified target color, such as red, green, or blue, RADSAT can then begin to execute a series of commands to autonomously search out a target with that color in a specified shape, such as a square or a circle. Once the target is identified, RADSAT can then begin to position itself such that it is locked on to the target and capable of hitting it when the human user gives the command to fire. With these goals in mind, the following lists all of the requirements that are fulfilled by RADSAT.

1) Color Recognition
2) Shape Recognition
3) Autonomous search algorithm
4) Wireless capability
5) Voice control

II. Hardware Specifications

The physical components that comprise the body of this project are a tank chassis with motors, a circuit board and a turret. The tank chassis and turret were purchased and assembled or stripped down as the case may be. The PCB was designed and built specifically for this project. It was designed to be able to route power from the battery to the various electrical components on board the tank. It also takes care of passing control signals received via Wifi to the necessary components such that the tank moves and is controlled as needed.

A. Tank Body

The tank body is comprised of the chassis and the turret. Because of its small size, approximately 9 x 14 inches, the tank chassis was bought as a 1:24 scale R/C replica of a M26 Pershing tank from World War II. The turret that came with the R/C tank was scrapped in favor of a specialized turret. This specialized turret was built from two servos and a servo attachment kit. It is mounted with the camera and the laser, which is, in this case, a representation of a gun. It is simpler and less messy to use a laser rather than an airsoft gun for demonstration purposes.

B. Circuit Layout

The layout of the circuit board is shown in Fig. 1. This is the final product that is used in the tank. It shows how the power allocation is achieved by use of parallel regulator sub-assemblies and relays for switching voltages as needed. Many of the wires shown are for testing and troubleshooting purposes.

At the bottom of the picture is a socket with eight slots where wires can be plugged into and taken out. These are the outputs to the motors. They were not soldered in because it was desired to be able to remove the circuit board from the tank chassis if needed. There is another socket on the upper left, it serves as power supply to the servos, servo controller, sensors, and camera.

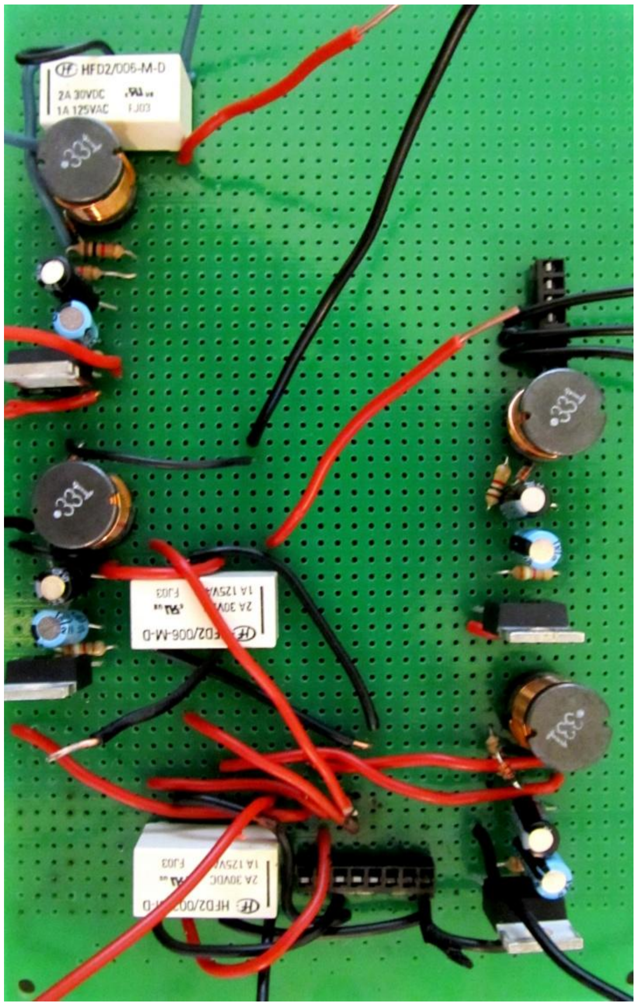


Fig. 1. Photograph of the circuit board showing components and wiring.

As seen in the picture there are five regulator sub-assemblies, easily spotted by the large round inductors. Each sub-assembly has an inductor, two capacitors, resistors, a diode and the LM2576 regulator itself. The purpose of these five regulators is to drop the voltage down from 7.8V to the required 6V or 3.5V for each output as well as turn off and on the voltage.

The white rectangular items in the picture are DPDT relays. Their purpose is, in the case of the lower two relays, is to provide output voltage to the motors that drive the tank treads. A relay is needed to enable forward as well as reverse. When the relay is not turned on the voltage at the output makes the motors move forward, however, when the relay is turned on, the output from the relay in that case switches the negative and positive terminals of the output, thus making the motors move backward.

The relay at the top of the picture is one of two outputs of the regulator at the top of the picture. The regulator supplies a constant 3.5V output to the microchip. It also must supply that 3.5V to the laser, but it has to be able to turn off and on. When the relay is off, it has no output, but when it is turned on it supplies the 3.5V to the laser pointer, which in turn causes the laser to shine.

Not shown in the above picture are the wireless receiver and micro-controller. They are also an integral part of the circuitry, but are mounted separately and attached to the circuit with connectors. There are actually two microchips, the main microchip controls the motors, laser, and sensor feedback, and the servo microchip controls the servos of the turret. The turret requires one micro-controller dedicated to it because it handles servo positioning for targeting, which is a vital part of this project, and some of the programming for it requires delays. It is also advantageous to have two microchips because with only one, there is the risk of running out of memory.

III. Autonomy and Sensors

The problem of implementing autonomous path finding without possessing a method of determining an absolute position is not trivial. Fortunately, a noteworthy advantage of the RADSAT project is that its objective is not to find a path from point A to point B with minimal cost, but to start at point A and find a potentially hidden point B. Although consistently finding the most optimal path without first being given the location of point B is generally impossible, minimizing cost is done by ensuring the same area is searched the fewest possible number of times. To adequately search a given area, RADSAT is given three pieces of information. The first is the relative distance between it and any obstructions that are within range, both to the front, the sides, and behind it. A way to identify the color that is being searched for once an instruction has been received is the second. Lastly, RADSAT must be aware of its heading.

There are generally two basic types of obstructions identified as being distinct in terms of sensing requirements. The first consists primarily of walls and objects with relatively 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. In order to achieve a reasonable level of autonomy, RADSAT has methods of detecting both type 1 and type 2 obstructions ("type 1" and "type 2" obstructions are used to refer to obstructions with large surface areas and small surface areas, respectfully), 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).

A. Sensing Large Surface Areas (Type 1)

RADSAT uses three Sony GP2Y0A02YK infrared sensors. Two are used as proximity sensors to detect obstructions located on either side, and one is used to locate those that are behind it. A major disadvantage of using infrared sensors is that they are only able to detect obstructions with sufficiently large surface areas and can only detect other obstructions in very specific circumstances. Ultrasonic sensors (sometimes referred to as sonar sensors) can detect obstructions that infrared sensors cannot, however, they are slightly more complex than their infrared counterparts and generally cost more. Both work using the same general concept: both send a signal and take note of either the time it takes to receive a reflection, or the angle at which the reflection is returned.

There are generally two relevant types of proximity sensors considered by the RADSAT team; 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 microcontrollers. Digital sensors 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 output an analog voltage that relates the output voltage to the distance between itself and the obstruction. There are some types of digital sensors that are able to give distance information using multiple output lines, though these sensors are very expensive. As such, all of the proximity sensors used by RADSAT are analog sensors.

B. Sensing Small Surface Areas (Type 2)

At one point, the RADSAT team considered dropping support for sensing obstructions with small surface areas given the increase in the level and difficulty and the resources available, coupled with the scheduled project completion date.

The Sharp GP2Y0A02YK infrared sensor uses triangulation to determine the distance between it and an obstruction. This approach may seem ideal, but after considering the beam size of the infrared sensor, which at only a few millimeters in radius, makes using it to detect small surface areas increasingly difficult. There exists infrared sensors that are more capable of more broad detection; however, these sensors are prohibitively expensive. RADSAT uses one MaxBotix MB1210 as its forward facing proximity sensor. It is a self-calibrating, ultrasonic sensor 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 0 cm to a maximum of over 760 cm.

One major advantage is that the MB1210 does not have the problem of associating virtually every output voltage with more than one distance (which the Sony GP2Y0A02YK has). Instead, if an obstruction is detected at a distance below 20 cm, the sensor will output a voltage equal to what would have otherwise been the output for an obstruction detected at exactly 20 cm. In more concise terms, the sensor's output voltage remains constant at any distance less than or equal to 20 cm. This feature greatly simplifies the problem of choosing a mounting location for the sensor because although 20 cm is the minimum range, any voltage output values at a distance below 20 cm *aren't* considered invalid [1].

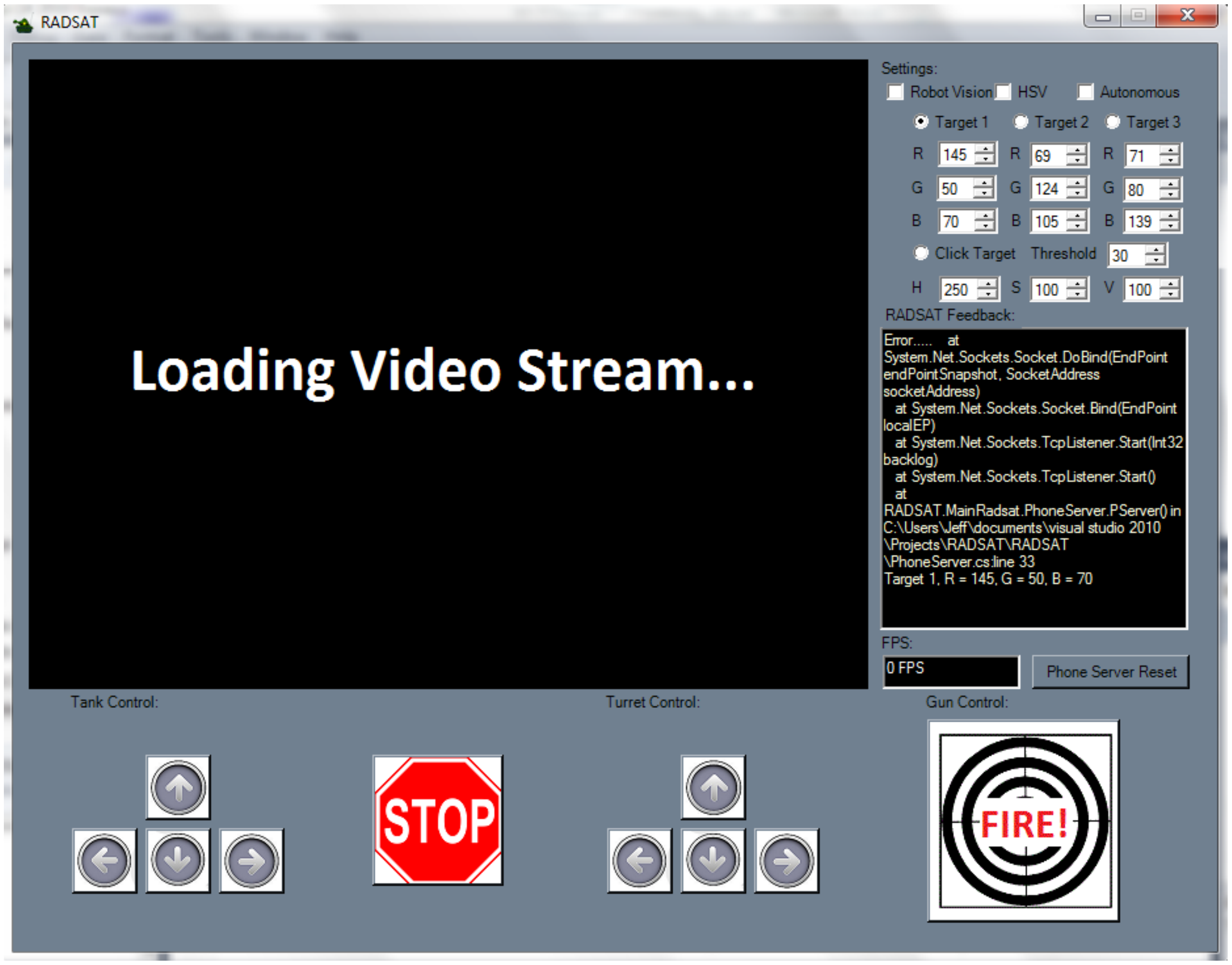
C. Heading and Searching

Deciding on whether or not to include a heading sensor is directly related to and could be directly answered by first determining the kind of searching algorithm RADSAT uses to find its target. To ensure reduce the cost and redundancy of the search, RADSAT searches an area with full knowledge of its absolute facing direction by using Honeywell's HMC6352 Compass module. The module has a heading resolution of 0.5 degrees [2], implying that it can detect a total of 730 possible headings. Considering that even 50 different headings could lead to a reasonably cost effective search, using this sensor is justified. The biggest drawback is that this sensor (and most other inexpensive magnetometers) uses the Earth's magnetic field as its basis for determining the direction north. If it enters another magnet's magnetic field, then the HMC6352 will measure north from that magnet's north pole, giving a useless measurement.

The most intuitive way to search is to start anywhere in a given area and travel in one direction until an obstruction is blocking its path. To avoid it, a complete 180 degree rotation is done in such a way that RADSAT moves parallel to the obstruction's surface before resuming its search. The search is then continued until either another obstruction is found, or the target is found.

IV. Vocal Command Recognition

Android is a Linux based operating system developed by Google and is largely focused on being compatible



[✓ Target 1]		Target 2		Target 3	
R	145	R	69	R	71
G	50	G	124	G	80
B	70	B	105	B	139
Threshold	30				
H	250	S	100	V	100

```
Error.....    at
System.Net.Sockets.Socket.DoBind(EndPoint
endPointSnapshot, SocketAddress
socketAddress)
   at System.Net.Sockets.Socket.Bind(EndPoint
localEP)
   at System.Net.Sockets.TcpListener.Start(Int32
backlog)
   at System.Net.Sockets.TcpListener.Start()
   at
RADSAT.MainRadsat.PhoneServer.PServer() in
C:\Users\Jeff\documents\visual studio 2010
\Projects\RADSAT\RADSAT
\PhoneServer.cs:line 33
Target 1, R = 145, G = 50, B = 70
```

```
0 FPS
```

Fig. 2. GUI

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 a developer with the ability to recognize incoming sound and to translate that sound into a data string. The programmer has to request the proper permissions in the project's manifest file and start an activity with an intent to start recording. From there Android's built in speech recognition will handle listening to the user and translating what the user said into an array of String objects that can be directly used by the programmer.

One of the core requirements of the RADSAT project is that a video feed must be displayed for the user. While it would make logical sense for an Android device to be able to display an incoming video feed that would allow a user to see what RADSAT is "seeing", there is a major roadblock hindering this ideal. Streaming video (or even a series of images) from a wireless feed while communicating with another wireless source is currently too demanding for most Android devices. When that is combined with having to also process vocal commands, it became clear that an Android app did not have 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 the Android SDK and its built in classes to handle general word recognition. However, in the end, the team decided against using Android as the main solution because of its limited computing capacity and because it would be difficult to integrate all of the features necessary to adequately control RADSAT. Android is still incorporated into the final design as a subcomponent of the entire RADSAT system.

V. Microcontroller Communication

To enable wireless communication between the base computer (which acts as a client) and RADSAT, the MICROCHIP MRF24WB0MA wireless module is used. With the exception of the ability to stream a live camera feed back to the base computer, RADSAT never actually sends information to any other device. RADSAT generally acts as a server which is always listening for requests from the base computer. This arrangement ensures that commands will not be ignored if the base computer acted as the server, and RADSAT's requests were received too sporadically.

The wireless module communicates directly with an Atmel ATmega 328P microcontroller. The 328P is an 8-bit processor with a 32 kilobytes of flash memory and a maximum of 23 I/O pins [3]. This MCU is responsible for acting as a server, handling incoming requests, handling all sensor and path finding calculations, and controlling the general movement of RADSAT's treads. In addition, it forwards all servo related instructions to a second 328P, which directly handles the movement of both of RADSAT's servos.

VI. Algorithms and Classes

A. GUI

The GUI class includes the code that deals with all of the commands the user must have access to in order to fully control RADSAT as well as the video display from the security camera mounted on top of the tank. As it is coded in C#, this class not only includes the code that executes the commands but also a design window that shows the application in its entirety that the user will have access to while controlling the tank. The GUI can be seen in Fig. 2.

Upon opening the application, the GUI first loads the live video stream from the security camera as it appears unaltered. The user then has the option of setting a target using 1 of 3 radio buttons and their corresponding RGB values, or targeting a specific color's RGB value by clicking a specific pixel on the video stream. At this point, the user has the option of switching the video display to "Robot Vision", which will only display the targeted color recognized by the ColorRecognition class, while all other pixels in the image will be changed to black as to isolate the targeted color. An example of how "Robot Vision" looks can be seen in Fig. 3 and Fig. 4 below. The user also has the option of further filtering the targeted color based on its HSV values. The threshold values that the ColorRecognition class uses to determine whether or not each pixel is the targeted color can also be changed within the GUI. The final checkbox option that the user has in the GUI is to set RADSAT to autonomous mode, which will disable all of the command buttons and will allow RADSAT to autonomously execute its mission to search for the targeted color.

The final code and design included in the GUI class are the buttons used to control the movement of RADSAT as well as the turret mounted on top of the tank. Each of these entities can be rotated to the left or right.

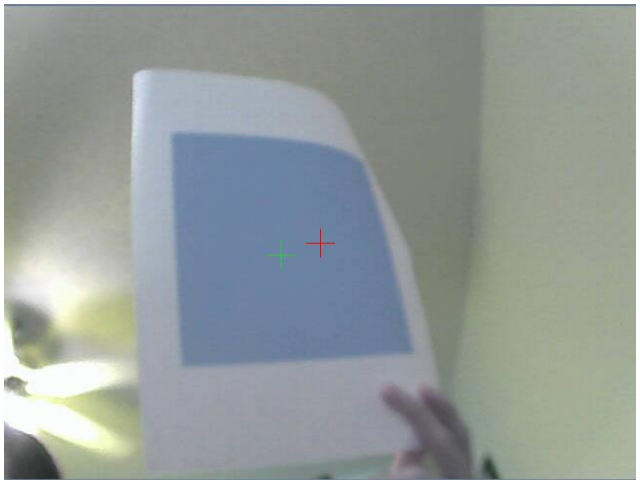


Fig. 3. Screen cap depicting the bitmap not affected by robot vision.

Furthermore, RADSAT can be controlled to move forward or backward and the turret can be controlled to rotate up or down. These commands are executed for as long as their respective buttons are held down by the user. A Stop button is included to stop any command that RADSAT is currently executing, as well as a Fire button that will fire the turret once the user has discovered the specified target. While the Autonomous checkbox is activated, all of the command buttons are disabled except for the Stop and Fire buttons.

B. Color Recognition

The ColorRecognition class is one of the two classes used to determine and isolate possible targets that RADSAT is searching for. Based on the RGB values of the target specified in the GUI, the ColorRecognition class is able to determine whether or not each pixel in the live stream matches that color within a specified threshold. In order to determine whether the color is within the specified threshold, the following equation is used:

$$D = \sqrt{(R - 0)^2 + (G - 255)^2 + (B - 0)^2}$$

Once the value D is determined for each pixel, the ColorRecognition class compares it to the threshold value. If D is greater than the threshold (i.e. it is not a match for the color), the pixel is changed to black. This is done by setting the RGB values to 0. If the D is less than the threshold (i.e. it is a match) then the pixel is left unaltered. By doing this to each pixel, the program is able to determine and isolate groups of pixels that match the targeted color in its efforts to search out and locate a target.

While the main focus of the ColorRecognition class is to determine the color based on its RGB values, the user also has the option of further filtering the colors based on their HSV values. This option is provided for the user in the case that the lighting is poor and the ColorRecognition class is having a hard time determining which pixels of the live stream from the security camera truly match the specified color. Because the ColorRecognition class' main goal is to determine the color based on RGB, however, this option is initialized to the off position in the GUI design.

C. Targeting

The Targeting class is the class that will control the autonomous motions of RADSAT once a specified target is successfully located. Once this event occurs, the Targeting class will make efforts to line up the static crosshairs of the air soft gun with the dynamic crosshairs drawn in the middle of the discovered target. This is done by determining the distance in pixels separating these two crosshairs, both horizontally and vertically. With these distances determined, the Targeting class will first continuously send commands to RADSAT to rotate to either the left or right until the crosshairs are vertically aligned within a reasonable threshold. At this point, the Targeting class will begin to continuously send commands to the turret mounted on top of the tank to rotate upwards or downwards until the crosshairs are horizontally aligned within a reasonable threshold. Once both of these goals are accomplished, the crosshairs will be aligned and RADSAT will inform the user that a target has been located and properly targeted. It will then be the user's discretion whether or not the Fire command will be executed.

If the Targeting class used a static equation to determine the movement of the tank and turret, then the instance might come up where the tank gets stuck moving back and forth between left and right, or the turret might get stuck moving up and down. For instance, if the tank only needed to be moved 30 pixels to the left but the equation was set to move it 60 pixels, then it would be stuck forever moving back and forth from left to right. In order to avoid the possibility of this situation occurring, the equation used to determine how much the tank should be moved will be dynamic based on the number of pixels the crosshairs are from the desired location. In this way, the Targeting class is easily able to quickly line up the crosshairs both vertically and horizontally.

D. Video Processing

This process allows the user to view the same image RADSAT is viewing. This is made possible by a Wifi enabled security camera which communicates with a router. The same router also communicates with the computer, RADSAT, and phone, which enables easy communication for all the components of the project. The Wifi security camera places an MJPEG stream on a local server and the program copies the images from this stream.

To copy the stream the AForge.net MJPEGStream library [4] was used to retrieve the MJPEG stream from the WiFi security camera. This library takes the JPEG image obtained from the stream, and converts it to bitmap. When it does this, the library creates a new "Frame Event" which the program handles.

Every time a new frame event occurs the program has to do accomplish several tasks. It must run the bitmap through the ChangePixels, ShapeRecognition, and DrawCrosshairs functions. It also must display the finished bitmap on the GUI.

This section also collaborates with the FPS section of the GUI. Every time a new frame event occurs, a counter is increased. Then a timer which goes off every second takes the counter and prints it to the screen, and subsequently returns the value of the counter to zero.

E. Shape Recognition

Shape recognition uses the edited version of the bitmap obtained from *Color Recognition*. An example of this image is shown in Fig. 4. Once the bitmap is obtained, the program then utilizes the AForge.net Blob library [5].

This library recognizes clusters of pixels based upon changes in color, and creates an object based around each individual cluster, meaning there can potentially be clusters inside of clusters. If you look at Fig. 4 you'll notice a black background and several pink rectangles. The pink rectangles represent a "blob" type object. Notice how every separate cluster of pixels has its own rectangle, this is because of the color contrast between the pixels and the black background. Because there are so many possible target choices from this library it is important to differentiate the desired target from the useless targets.

In order to accomplish this two conditions must be met for the blob. 1) At least 75% of the blob must contain matching pixels, meaning that at least 25% of the blob can contain black pixels. This ensures that the blob is a flat object (lighting is the same throughout the blob) with intentions on being a target, and 75% is enough leeway to cope with any lighting issues which may arise on the actual target. 2) The blob must have a length and width within ten pixels of each other. The desired targets are square, and this ensures the only shape that will be recognized will be a square, or something very similar.

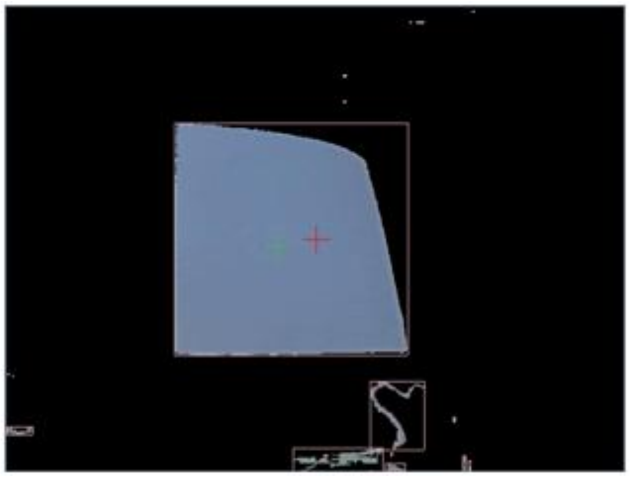


Fig. 4. Bitmap image after Color Recognition, and Targeting were applied.

C. Phone to Computer Communication

The Android-based phone and computer communicate through a TCP-type connection [6]. In order to accomplish this, the computer needs to create a TCP server which the phone can communicate with. So, a server is created in C# which communicates on a socket with the phone. The IP address of the server is "192.168.1.2", and the port for the socket is 1000. When the server is initiated, it will patiently wait for a connection to be established with the phone, in a thread separate from the rest of the program.

Once a connection is established the server will enter a perpetual loop constantly waiting for commands. The commands are received as a byte array, and then it is converted to a string. The program then takes the string and uses the data to call upon the correct function.

However, if the connection between phone and computer is interrupted, or the connection is dropped, then the program will still wait for a command even through there is no longer a connection with the phone. To adapt to this situation, a button on the GUI can be pressed to attempt to reestablish connection. The button is labeled “Phone Server Reset”, and pushing this button will reset the server to the “Waiting for Phone Connection” stage.

F. Computer to RADSAT Communication

The class RobotMotion, as well as an http server program running on the RADSAT’s microcontroller is how the two communicate with each other. The server located on the RADSAT’s microcontroller is able to read in HTTP commands, and the class “RobotMotion” is able to send those commands.

RobotMotion sends commands by using the C# command “Webrequest.Create(“http://192.168.1.100/” + command)” where “http://192.168.1.100/” is the location of RADSAT’s server. So, if the C# program sends the command “http://192.168.1.100/go” RADSAT will receive “/go”. When it receives the command RADSAT will then take the appropriate action and execute the command accordingly.

This system works well in most situations. However RADSAT can only process one command about every .1-.2 seconds which can raise a problem in certain situations. In order to cope with this weakness, several timers are set in several possible scenarios. For instance, when RADSAT is in manual mode, the control buttons will issue the command as long as the button is held down, and issue a stop command as soon as the button lifted up. So when the button is lifted a timer goes off for .5 seconds until the stop command is issued. In the event that a movement button is quickly pressed and two commands are trying to be sent almost simultaneously, because of this system, RADSAT will be able to receive and interpret all the commands sent to it.

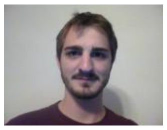
A very similar situation occurs when in autonomous mode, and the program discovered the target and is trying to move the tank and turret in order to align itself with said target. In this scenario, the program will try to send a command to realign itself every new frame. So, to cope with this situation, different timers are implemented depending on how far away the static target is from the dynamic target a different timer will be administered. For instance, if the targets are 100 pixels apart a timer for 100 milliseconds might be implemented. This will also prevent RADSAT from never lining up with the target because of constant over-movement of the target had static timers been used.

VII. Conclusion

RADSAT is a system comprised of a multitude of different components and subsystems utilizing technologies ranging from the newest mobile devices to RC tanks fitted with sensors and embedded microcontrollers. The goal of the RADSAT project is to model the methods and communications necessary to successfully complete search and destroy operations in potentially dangerous environments.

Biography

Jeff Hildebrandt will be graduating with a bachelor's degree in Computer Engineering. He enjoys cooking as well as programming. And hopes to one day land a job where he can do both.



Dylan Lambe is a 4th year Computer Engineer at the University of Central Florida. His interests include basketball, snowboarding, and computer games. After graduation, Dylan plans to move out to Utah to live with his family and to get a job as a software developer.

Mick Muzac will be graduating from the University of Central Florida with a bachelor's in Computer Engineering. He is currently employed by the ADL Co-Lab in Orlando Florida and works on the technical team as a Software Engineer. After graduation, he plans to continue working with ADL.

Bradley Raley is graduating with a bachelor's degree in electrical engineering from UCF in Summer 2012. He spent almost six years at UCF and in addition to the learning obtained from engineering professors, he trained himself in many other arts and physical disciplines. He can do a back flip so if you ever see him, ask him to do it, he totally will.

Acknowledgement

The Internet, Dr. Samuel Richie

References

[1] "MB1210." MB1210. N.p., n.d. Web. 25 July 2012. <http://www.maxbotix.com/products/MB1210.htm>.

[2] "SEN-07915." Compass Module. N.p., n.d. Web. 20 July 2012. <https://www.sparkfun.com/products/7915>.

[3] "ATmega328P." Atmel. N.p., n.d. Web. 20 July 2012. <http://www.atmel.com/devices/atmega328p.aspx>.

[4] Kirillov, Andrew. "Blob.cs." . AForge.net, Mar 9, 2010. Web. 24 Jul 2012. <http://code.google.com/p/aforge/source/browse/trunk/Sources/Imaging/Blob.cs?r=1208>.

[5] . "MJPEGStream Class." AForge.net. N.p., n.d. Web. 24 Jul 2012. <http://www.aforgenet.com/ >.

[6] "A very basic TCP server written in C#." . N.p., 27, Feb 2006. Web. 24 Jul 2012. <http://www.codeproject.com/Articles/13232/A-very-basic-TCP-server-written-in-C>.