The Target Discrimination and Neutralization System

Daine Richard Lesniak
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
Platteville, WI 53818
lesniakd@gmail.com

Douglas J. Hickok
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
Platteville, WI 53818
hickokd@gmail.com

Kristopher Whisler
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
Platteville, WI 53818
sgt.servo@gmail.com

Michael C. Rowe, Ph.D.
Computer Science and Software Engineering Department
University of Wisconsin – Platteville
Platteville, WI 53818
rowemi@uwplatt.edu

Abstract

In 2005, we constructed an automated missile defense system to detect, target, track, and shoot paper airplanes. This project was done to demonstrate the feasibility of such systems for university multi-disciplined design and development projects, and to create a system to get students who attend the annual UW-Platteville's Engineering Expo interested in software engineering and computer science. This project exceeded our expectations and was by all measures both successful and educational, but it was not perfect.

For 2006, we have attempted to address some of the imperfections in this system as well as add new, challenging features to this system. This paper discusses our analysis of last year's project, the plans to improve deficiencies in last year's project, as well as the design and implementation of new functionality that we refer to as the Target Discrimination and Neutralization System. This year's version of the system will be available for demonstration at MICS.

Introduction

Last year we set out to attempt to design and implement a system that could autonomously detect, locate, track, target and shoot paper airplanes [1]. We were very pleased with our accomplishments, but we were also very aware of deficiencies in this system. There were also many more features that we wanted to add to the system. This paper will discuss the successes and deficiencies of last year's project, and describe this year's new system that addresses deficiencies as well as introduces significantly new features to this year's version of the system that we refer to as the Target Discrimination and Neutralization System.

2005 Project Summary

Last year's Autonomous Paper Airplane System was a proof of concept prototype that demonstrated that we could implement a system to detect, locate, track, target and shoot paper airplanes.

Description of 2005 System's Functionality

A standard webcam was used to produce a bit map that was sent to a standard personal computer for processing. This bit map was sampled, using a trip grid technique, to determine rapidly the presence and general location of a paper airplane. If a plane was detected, then all pixels in the area around the pixel of the trip grid that detected the airplane were analyzed to ascertain the front point of the airplane. After the front point was identified, the targeting software located a target point behind the front point near the center of the plane. The coordinates of this point, in image space, where then translated to a coordinate system that matched the granularity of the actuators' movement. The recalculated coordinate was then sent to the hardware controller to trigger the reorientation of the armament. Once the armament finished reorienting itself, the operator could click a button on the personal computer, which commanded the microcontroller to open a bank of solenoid valves that shot a plastic dart from the turrets blowgun.

Once the system was calibrated it achieved a better than 95 percent hit rate.

Analysis of the 2005 System

We were extremely happy with the 2005 system's amazing accuracy, better than 95 percent hit rate, and we believe that the few target misses that we did experience were caused by bent darts. We did realize that the system had some significant defects and limitations in other areas. These limitations and deficiencies included portability of the system, setup complexity, interface used between personal computer and microcontroller, and software architecture. This section will discuss each of these issues.

The system was *not very portable*. This was due primarily to the need for a large air compressor to propel the dart out of the blowgun. Initially we had not considered the need to transport this system to multiple sites as it was originally planned that we would use the system for one demonstration at our University's annual Engineering Expo. As it turned out, the 2005 system was demonstrated to over 600 people at the Expo and won first prize. Based on this success, we were requested to do additional local demonstrations and even took it to Eau Claire for MICS-2005. We would have liked to have taken this system to area high schools and other locations to help recruit new students, but it was just not portable enough. The compressor was also very noisy and disrupted near by events.

The system required a *significant setup time*. There are many factors involved with setting up the 2005 system. One setup issue was related to running power cords for the compressor (required a dedicated 20amp circuit), personal computer as well as the microcontroller. Another setup issue was sighting in the system. The web cam was independent of the turret assembly, and thus, the two coordinate systems required a trial and error calibration. This calibration issue was also responsible for our project's only casualty, when crewmember was holding a paper airplane and was shot in the finger with a plastic dart.

The *interface between the microcontroller* and the personal computer was implemented through a standard serial connection. This was done, as it was the only communications port available on the scavenged microcontroller that we were using. We did not think that this was going to be a problem until we tried to find a serial port on our laptop computers. After checking all of the laptops available to the Computer Science and Software Engineering department, we discovered that serial ports were no longer standard. To solve this problem we had to use a less than portable large chassis workstation. This decreased portability of the system. In addition, Microsoft Visual Studio .Net 2003 no longer supported the serial communications. This issue was addressed by adding an old dll from Visual Studio 6 that luckily was still compatible with .Net.

Software architecture was the fourth major issue of the 2005 system. The system was implemented using primarily prototyping techniques. We have all heard in our software engineering courses that a prototype is a great way of proving concepts and reducing project risks, but that the code generated by prototyping efforts should be thrown away and yield to software that is systematically designed and implemented. In the analysis of the 2005 effort, it was determined that the production code was done for the most part, by pure prototyping. We would like to both endorse and practice the policy of throwing prototype code away. The 2005 code is seriously lacking in the dimension of maintainability, and thus, would be very hard to extend with new functionality.

Goals for 2006 Project

Based on the successes of and issues with the 2005 system we defined several goals for the 2006. This will be the last version of this system for the present student team as two of the team members will graduate this spring and one is already in a graduate program. Since it is our last version, we want to ensure that the next team of students will still have kind words to say about us after looking at the code. Let us first describe how we have addressed the issues described in the preceding section, and then we will discuss new and wonderful functionality that we have set our sites on achieving.

The issue of portability has been addressed by upgrading the weapon subsystem. Rather than using a blowgun powered by a large air compressor, we have integrated a $24.98 Kalashnikov AK-47 Airsoft Gun, which uses an electro mechanical propulsion system that runs off of 6 volts and has a muzzle velocity of 130 fps, and is auto-loading and contains a magazine of 88 plastics BBs. The gun was purchased from the local K-Mart [2] (see Figure 1). This device, off the shelf, is normally powered by four batteries, but we reengineered it to run off of the same scavenged power supply that is used by our microcontroller. This has eliminated the need for the bulky air compressor, makes loading much simpler, and opens the opportunity for multiple shoots per trial. The new system is now very portable and our department will be easily able to take this system on the road to help recruit middle and high school students into our college of engineering.

Figure 1: Kalashnikov AK-47 Airsoft Gun

Significant setup time was largely addressed by eliminating the need for the air compressor. A single extension cord and a plug strip can easily power the whole system. We have also statically mounted the web cam on the turret assembly. This greatly reduces the calibration needed to register the webcam and actuator coordinate systems. We hope that this will also minimize the risk to our crew during calibration.

Interface between the microcontroller and personal computer was addressed by purchasing a new microcontroller that supports USB communications. This interface is supported by nearly every computer that has been built in the last four years. We also expect that the USB standard will continue to be supported by hardware for several more years. In addition, Microsoft Visual Studio .Net has full support for this communications protocol.

Software Architecture was the final area earmarked for improvement this year. Last year's system was largely a prototyped effort. This will be the last version of this system for the present student team, as two of the team will graduate this spring and one entered a graduate program this past January. Since it is our last version on which our current student team will be working, we want to ensure that the next team of students will still have kind words to say about us after looking at the code. Also, we have envisioned several functional enhancements to undertake this year. Before we started to implement these new features we wanted to "do the software engineering thing" right, thus we analyzed our current and future requirements and started designing an extensible yet efficient architecture. We will discuss this software design and the envisioned enhancements in subsequent sections.

Software Architecture

The main thrust of the software architecture of the Target Discrimination and Neutralization system is to allow for the maintainability and extensibility of the system necessary for it to be a suitable legacy project. To accomplish this solid Object Oriented Analysis and Design as well as good Software Engineering principles were utilized. The system was constructed out of a number of classes that are as loosely coupled and highly cohesive as possible. High cohesion in classes means that everything in the class contributes to a clearly identified functionality or purpose, while low coupling means that classes are as independent as possible. Taken together, these two factors contribute greatly to the maintainability and extensibility of a system. Low coupling ensures that changes made to one class do not propagate causing needless changes to other classes, while high cohesion helps ensure that to change or extend functionality, only the classes that cohesively contribute to that functionality need to be changed. While there are many data and utility classes, the main classes are discussed below.

Below is a high-level class diagram showing the relationships of the classes discussed in this section.

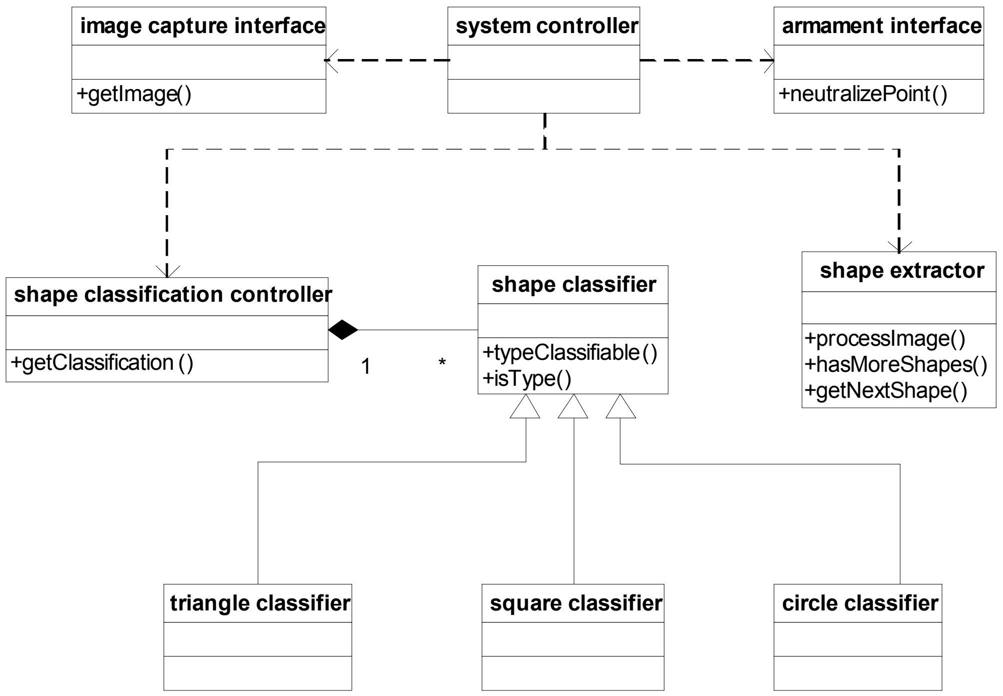

Figure 2: Class Diagram of Target Discrimination and Neutralization system

Armament interface

The armament interface class is responsible for encapsulating the external armament and armament control aspects of the system, exposing only what is needed for interfacing with the rest of the system. The armament interface exposes the ability to aim the armament and the ability to fire the armament, while hiding the details of what the armament is and how it works.

Encapsulating and hiding internal details of the armament used by the system into the armament interface allows for one extremely important benefit: the armament can be changed and affect only the armament interface class itself, leaving the other classes unchanged and blissfully unaware of the changes. This allows future groups to change the external hardware used to neutralize targets, keep up with the engineering arms races, and try exotic new concepts while leaving the rest of the system unscathed. This is also important for the maintainability and lifetime of the project, as hardware breaks down and becomes obsolete. Should the Brainstem microcontroller become unusable and replacements unavailable, the forced replacement will not cause any more rework than absolutely necessary, as the major if not all changes would be to the armament interface class.

Image capture interface

The image capture interface class is responsible for encapsulating the external image capture hardware interaction of the system. This is very similar to the responsibility of the armament interface as the idea is to hide unnecessary details and provide the rest of the system with a generic, hardware independent interface to interact with and depend on. The primary exposed functionality of the image capture interface is the ability to return a bitmap for the captured image. A bitmap was chosen due to its excellent functionality and support within the .Net[5] framework.

As with the armament interface, the primary benefit of encapsulating and hiding internal details of the image capture hardware into the image capture interface is that the stress placed on the system by changing the hardware is reduced. This allows future students to maintain and or extend the image capture capabilities and hardware with minor system changes.

Shape extractor

The shape extractor class acts as a "shape tokenizer" by breaking down an image and extracting and returning individual images that are of interest. It is given a bitmap of the entire field of view to process, extracts individual shapes, and then allows access to the extracted shapes. The shape extractor is for extracting multiple "non background" shapes from a background, allowing for the system to engage multiple targets at once. The shape extractor can be viewed as the first step of processing the bitmap returned by the image capture interface, and acts as the *detection* of potential targets.

The shape extractor class has many architectural benefits. It makes a lot of sense to have a class dedicated to the detection and extraction of shapes, as it not only has its own cohesive and clearly defined purpose, but also helps to decouple the image capture and the classification of targets. This is achieved with the help of the shape data class, which is what is actually returned by the shape classifier. The shape data class holds all information on an extracted shape.

A benefit of the cohesiveness of the shape classifier is that improvements to the system's detection and extraction capabilities can be accomplished by simply changing the shape extractor class. Currently, the shape extractor extracts white images from a black background; making more robust and flexible detection and extraction a clear extension point for the system. Should future students decide on allowing the system to detect and extract shapes from a heterogeneous background, they can focus their creative powers and expertise on the task at hand while treating the rest of the system as a black box so long as they respect the defined interface of the shape extractor class.

Shape classification controller and shape classifier

Classification of a shape is accomplished through the use of a shape classification controller class, which in turn uses a number of shape classification classes. The shape classification controller class is responsible for actually determining what the classification for a shape is, but it accomplishes this by polling a collection of shape classifier classes. The shape classifier is simply a general class which defines the specific interface all shape classifier classes adhere to, and the specifics of classification is the responsibility of the child classes that inherit the interface.

Each child class of the shape classifier parent class is responsible for determining if the shape it has been asked to classify is or is not of the type it can recognize. This one to one relationship between shape classifications and children of shape classifier creates a pseudo plug-in architecture, making additional classification capabilities easy to add.

There are two distinct ways in which the pseudo plug-in architecture helps the extensibility of shape classification, by ensuring that algorithmic improvements in classifying a shape already classifiable does not affect the classification of other shapes needlessly, and by allowing classifiers to be added for new shapes with ease. A monolithic classifier that utilized the same classification paradigm for all categories would prove it a nuisance should future teams decide to try new classification techniques on a subset of shapes. By utilizing a pseudo plug-in architecture, we can support a diverse set of classification concepts and tie them together through the shape classification controller. The pseudo plug-in architecture also allows adding new categories to the set of shape types that can be classified. Rather than rebuilding the monolithic classifier, retraining it, or attempting to include the rules for the new classification, the pseudo plug-in architecture supports simply creating a new child of shape classifier and adding it to the shape classification controller.

The current system is capable of classifying circles, squares, and equilateral triangles provided they are close to a standard orientation. Future students can extend this project by increasing the robustness of classification by eliminating the orientation constraints, or by increasing the set of classifiable shapes by adding children of shape classifier for more than circles, squares, and equilateral triangles.

System controller

The system controller interfaces with the main classes such as the armament interface, shape extractor, image capture interface, and shape classification controller to coordinate the entire system and provide the central functionality. The system controller class is the main access point of the system, and the part that interfaces with the user interface.

The system controller decouples the main classes from each other, and ensures that changing one of the main classes would affect the others as little as possible. Should the basic process or procedural flow of the system be changed drastically, it is the system controller that would be altered. The system controller also isolates the rest of the system from the UI, allowing the UI to be altered and experimented with freely.

System aspects

The overall system of the Target Discrimination and Neutralization project retained the use of a standard USB webcam for image capture and the use of a Brainstem hardware controller for armament control, but traded in the blowgun rigged to an air compressor that was used for the Autonomous Missile Defense system for an Air Soft gun. The use of an Air Soft gun has many advantages, including decreased setup time, increased portability, and support for future expansions. The Air Soft gun also creates a more entertaining demonstration.

By using an Air Soft gun as the primary armament, the Target Discrimination and Neutralization system gains a higher degree of portability and decreased setup time. This is because the Air Soft gun can be powered and triggered by the same hardware used to control its orientation. An air compressor was a large burden to move, set up, and could cause issues with respect to whether rooms could accommodate a demonstration. By having an armament that can remain permanently integrated, inputs and all, with the system hardware unit, we have reduced the system footprint dramatically.

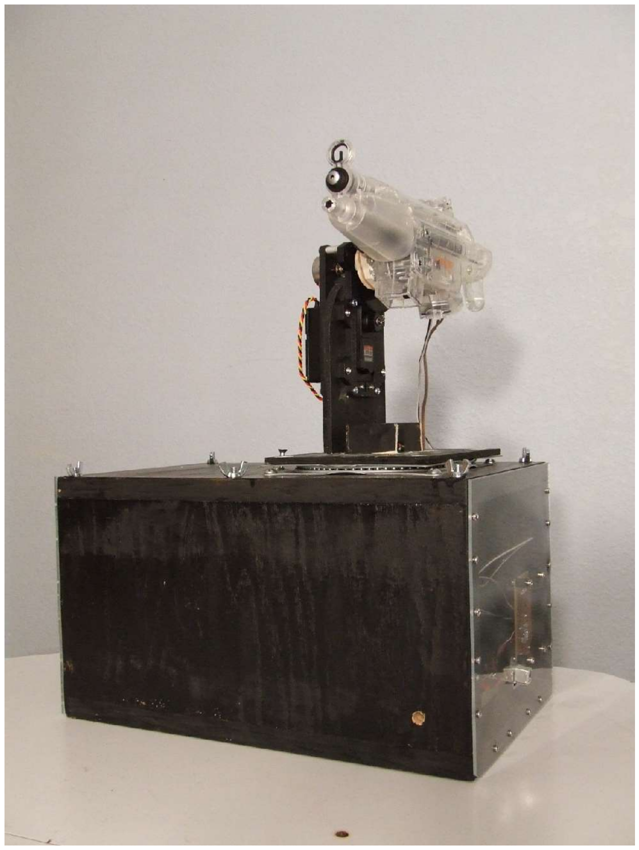

Figure 3: The entire turret assembly. The black box contains a salvaged PC power supply that powers the Airsoft gun on top, the X and Y actuators, and the BrainStem microcontroller. The BrainStem is also inside the black box.

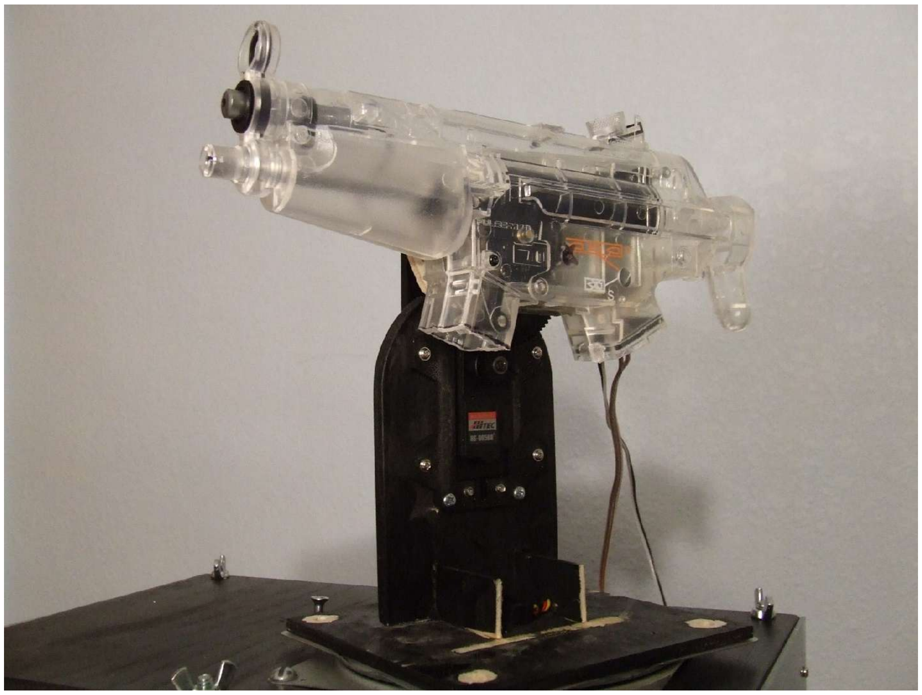
Figure 4: The AirSoft gun modified to run off the PC power supply and the trigger controlled from the BrainStem.

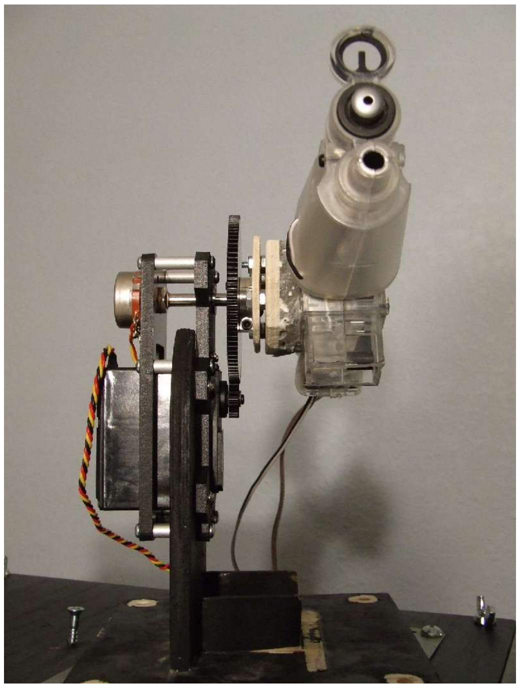
Figure 5: The AirSoft gun attached to a 5:1 reduction gearbox. The gearbox is connected to the Y actuator, which controls the elevation of the gun. The Y actuator is commanded by the BrainStem. At the bottom of the picture is the X actuator and gearbox that controls the left-right movement.

The old armament of a blowgun and air compressor was unfortunately limited to a single shot, and therefore limited to applications with but one target. By utilizing a multiple shot armament, we are able to engage multiple targets at once, and expand the possibilities for future groups. Utilizing a multiple shot armament also makes for an extremely entertaining demonstration. Not only is the Target Discrimination and Neutralization a “high impact” demonstration, but it can “impact” multiple times between setup, allowing for less down time and more demonstration.

The last major change to the hardware side of this project was the migration from a serial (RS-232) interface to USB. For all purposes the older serial connection worked great for the previous version of this system and would work equally well for this version. However, with there being an emphasis on portability and maintainability this change needed to be made. Looking at the portability issue, taking the old system around to various events for demonstrations required that a desktop computer and monitor also be brought with due to the lack of RS-232 sockets on modern laptop computers. Therefore, with the change over to USB, the amount of hardware needed at any demonstration site is limited to the armament hardware assembly, a laptop computer, and a small number of cords.

2006 Iteration Implementation

The 2006 iteration leveraged quite a bit of the implementation from the Autonomous Missile Defense System, taking the code and placing it within a better architecture with more encapsulation and hiding of internal details. Specific functionality that especially benefited from reuse was that of image capture and armament control. As with the Autonomous Missile Defense system, the Target Discrimination and Neutralization system utilized Microsoft Visual Studio C# [7] and made extensive use of built in classes such as the Bitmap class[6]. Following is a functional discussion of the 2006 implementation of the Shape Extractor and the shape classifiers, which are new to the Target Discrimination and Classification System.

Below is an example image we captured and utilized for testing shape extraction and classification. The extraction and classification of the shapes in this image was 100% successful despite the obvious fuzziness of edges and fluctuations in the background.

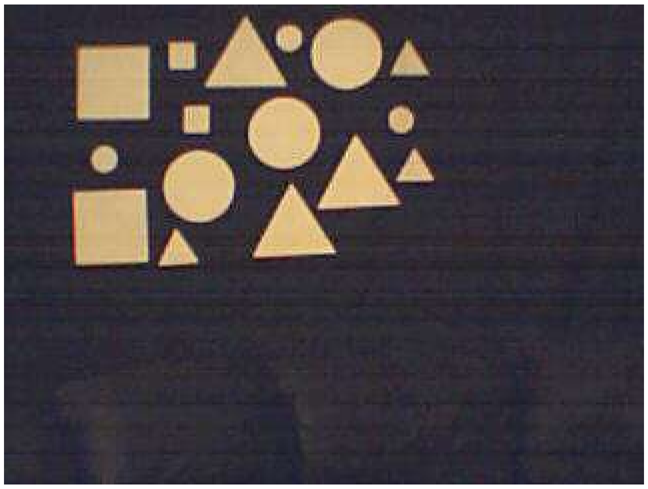
Figure 5: A bitmap image that contains shapes to be extracted by the Shape Extractor.

Shape Extractor

Our current implementation of the shape extractor utilizes a trip grid approach for detection, and a modified flood fill algorithm[3] for shape extraction. The trip grid approach involves checking a grid of selected pixels spaced such that the computational burden is decreased significantly while the grid is dense enough that any shape will share at least one body pixel with the grid. Once a pixel belonging to a shape is located, the scanning of the trip grid is paused while the modified flood fill algorithm extracts the shape.

The flood fill algorithm is a method for altering images by filling in contiguous regions of color with a new fill color. The type of flood fill algorithm we utilize is the four way recursive flood fill. It can be summed up in the following pseudo code:

```
Flood Fill( pixel ):
    If( current pixel is to be filled ):
        Color pixel new fill color
        Flood Fill( pixel above )
        Flood Fill( pixel below )
        Flood Fill( pixel to right)
        Flood Fill( pixel to left )
```

We utilized this algorithm for extraction of shapes by having the shape extractor keep two copies of the bitmap from which it is to be extracted, one as a reference and one for modification/scanning. On the bitmap to scan/modify we flood fill with black to remove the shape from further classification, while keeping track of the pixels that were filled. During the flood fill process we also keep track of the upper most, lower most, left most, and right most pixel location. After the flood fill is complete we utilize the information on pixel locations, minimum and maximum X and Y locations, and the reference bitmap to create a new bitmap with the extracted shape bounded in a box. This extracted shape bitmap along with other relevant information is then encapsulated within a shape object, which can later be returned by the shape extractor.

Classifiers

The specific child classes of the shape classifier utilized in this iteration are the circle classifier, the square classifier, and the triangle classifier. All three shapes are currently being classified in one dimensional feature space, which is a fancy way of saying we only look at one variable. While this is not as robust as other methods, it is an example of "good enough" software [4]. Provided orientation is standard, our classifiers work effectively, and the ability to classify circles squares and triangles is an excellent addition to the system. More robust and more varied classifications are candidate extensions for the future, but the current implementation is still and effective demonstration.

The feature utilized for classifying shapes is the ratio of background pixels to total pixels in the bitmap with the shape inscribed in it, so essentially a measure of how much of the circumscribed square the shape does not occupy. The ranges or feature space occupied by the shapes is as follows: 0%-10% = square, 15% - 30% = circle, and 35% - 55% = triangle.

Summary

This paper discussed two subjects, the successes and limitations of the 2005 project and how the limitations were addressed and successes enhanced in the Target Discrimination and Neutralization system. Aspects of the Target Discrimination and Neutralization system discussed were the upgraded software architecture, the upgraded system architecture, and the implementation of new features. These subjects are tied to the goals of the system as well as good object oriented analysis and design and software engineering principals.

Acknowledgements

The team would like to acknowledge our department's chair, Professor Mohan Gill for procuring funding for our project and also Ms Mary Jo Stutenberg, our department secretary for helping order and ensure delivery of the hardware components.

References

[1] Lesniak, D. R., Hickok, D., Whisler, K., and Rowe, M. C., "An Autonomous Paper Airplane Defense System", *38th Annual Midwest Instruction and Computing Symposium*, April 2005, University of Wisconsin, Eau Claire.

[2] K-Mart Kalashnikov AK-47 Airsoft Gun, description can be seen at http://www.kmart.com/catalog/product.jsp?productId=137621&N=0&Nty=1&Ntk=All&Ntx=mode%20matchallpartial&Ntt=airSoft%20gun.

[3] http://www.codeproject.com/cs/media/floodfillincsharp.asp

[4] http://en.wikipedia.org/wiki/Principle_of_good_enough

[5] http://msdn.microsoft.com/netframework/downloads/framework1_1/

[6] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdrawingbitmapclasstopic.asp

[7] http://msdn.microsoft.com/vcsharp/default.aspx