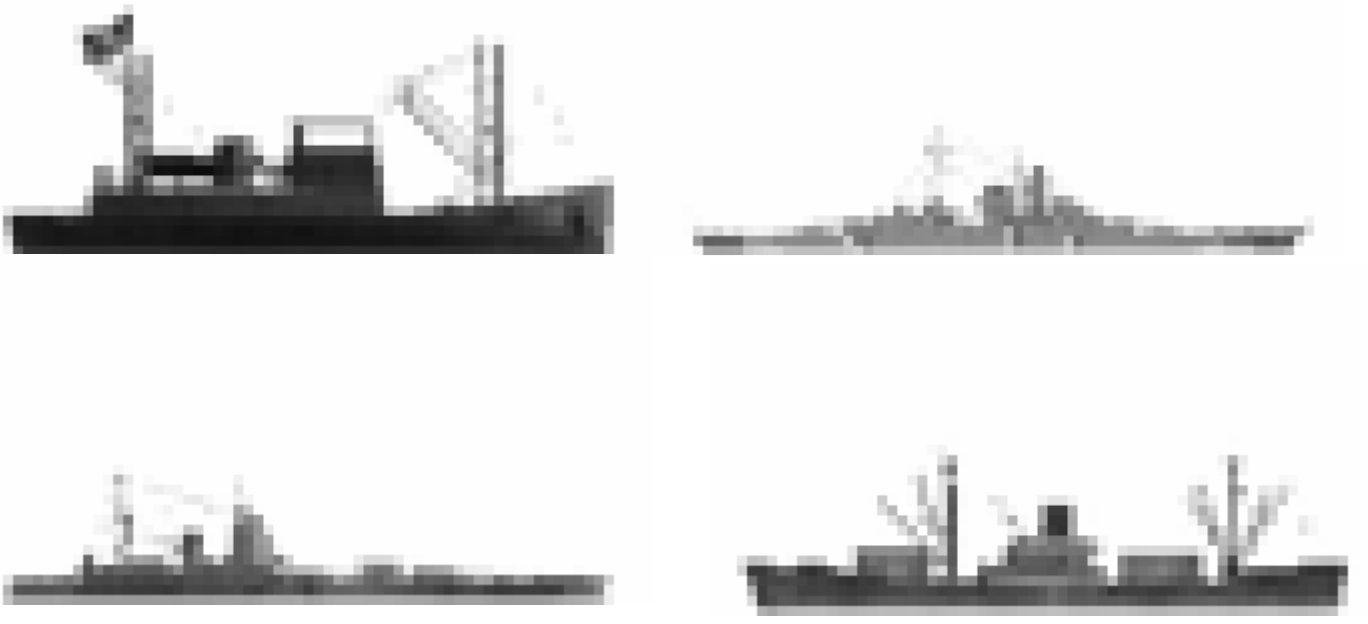
18-551, Fall 2006

Group 7, Final Report

Optical Ship Recognition: Eyes on the Horizon

Greg Jaworski <gregoryjaworski@gmail.com>
Tyler Paulk <tpaulk@andrew.cmu.edu>
Theodore Trebaol <ttrebaol@andrew.cmu.edu>



INTRODUCTION

Surfacing is the most dangerous maneuver for any submarine. In order for a submerged submarine to return to the surface, the boat must determine if there are any other vessels in the vicinity that could pose a collision threat to the submarine. Today, boats determine the locations of surface vessels by coming close to the surface and looking through the periscope. The entire safety of the crew rests upon one person's ability to visually detect other vessels from a few sweeps of the horizon with the periscope. In 2001, the US Naval Submarine, *USS GREENEVILLE* while surfacing accidentally struck and sank the Japanese fishing vessel *Ehime Maru* killing all nine members of her crew. Had this been a larger vessel, such as a tanker, the submarine could have been more seriously damaged and more lives could have been lost.

Submarines when surfaced want to maintain a far distance from all other vessels. This is to prevent a collision and to remain undetected. Use of radar could accurately identify a targets location but would give away the submarines position. Active sonar, like radar, could accurately identify a targets location but would give away a targets position. Passive sonar, the type of sonar modern submarines use, only listens to acoustic signal and does not transmit. This can provide an accurate bearing to a target but cannot provide accurate range information. Submarines can move relatively fast through the water but not fast enough to make synthetic aperture sonar effective.

We processed images from periscopes to optically locate other distant vessels, classify them. This would minimize the amount of time a submarine has to spend near the surface, where it is most vulnerable. This will also reduce the amount of human error involved in ensuring the safety of the crew. This is an image-processing problem and the signals to be processed are 512 x 512 bitmaps at a rate of 30 seconds per filter.

OUR SOLUTION

We requested databases of periscope images and video from the Office of Naval Research and Submarine Development Group 12, but they were not able to get us periscope video. Several weeks into the project they offered to give us video from a harbor security camera. We could not wait for this database and so we elected to use a computer simulated submarine environment, a video game. We used Ubisoft's *Silent Hunter III*, which uses models of ships from World War II. We detect and classify 4 different types of ships at ranges where it is difficult to determine the classification visually. We classify a German Bismarck class battleship, a British Nelson class destroyer, a Japanese merchant vessel and an American coastal vessel, as they are referred to in *Silent Hunter III.* Since the 4 classes we classify are different sizes, we consider this range different for each class. We set those ranges to be those where each ship is 60 pixels long. Following the advice of Professor Casasent and our TA Rohit Patnaik, we decided to approach this problem with MINACE[1] filters and linear correlation. We detect the horizon and the location of a possible ship without correlation in order to restrict the region in which we linearly correlate. Due to the general position of a submarine periscope with relation to a surface vessel, our problem is also restricted to a single depression angle of 0 degrees. We do however have to consider the various poses surface vessel could take relative to the periscope. In designing the filters, we use 18 training images for each class of ship at every 20 degrees of pose from 0 degrees to 340 degrees. The DSK processes 512 x 512 input images and if a ship is present in the image the DSK outputs to our graphic under interface (GUI) the relative location of the surface vessel within the input image and the ship's class. Our GUI takes this

---

•[1] *Filter Classification Algorithms for ATR Using MSTAR Data*, Rohit Patnaik and David Casasent. Dept. of Electrical and Computer Engineering, Carnegie Mellon Univeristy, SPIE Proceedings Volume 5807, May 2005.

information and re-displays the input image with a class-specific colored box around the ship. Our project is restricted to scenes of single ships on the open seas. We do not address clutter rejection.

Previous Projects

Our project is fairly unique but several projects did have related applications.

Group 1, Fall 2005 – Used a morphological process to prevent counter fitting. They did not use many of the same methods that we used, except they did use a bilinear transform to resize images.

Group 2, Fall 2005 – Recognized and classified tanks. This was the project that was most similar and useful for our project.

Group 7, Spring 2004 / Group 6, Spring 2003 – Iris scanning. This project had little in common with our project

Group 2, Spring 2003 – Face verification. They used the same type of filters that we used (MINACE filters).

Group 5, Spring 2003 – Written character recognition. This project had little in common with our project.

Group 7, Spring 2003 – Noise reduction in text images. This had little in common with our project.

Group 8, Spring 2003 / Group 18, Spring 2000 – License Plate Recognition. This project involved image processing, but it was used for a different application.

Group 2, Spring 2002 – Face Detection. This project had little in common with our project.

Group 3, Spring 2002 – Gender and Emotion classifier. This project had little in common with our project.

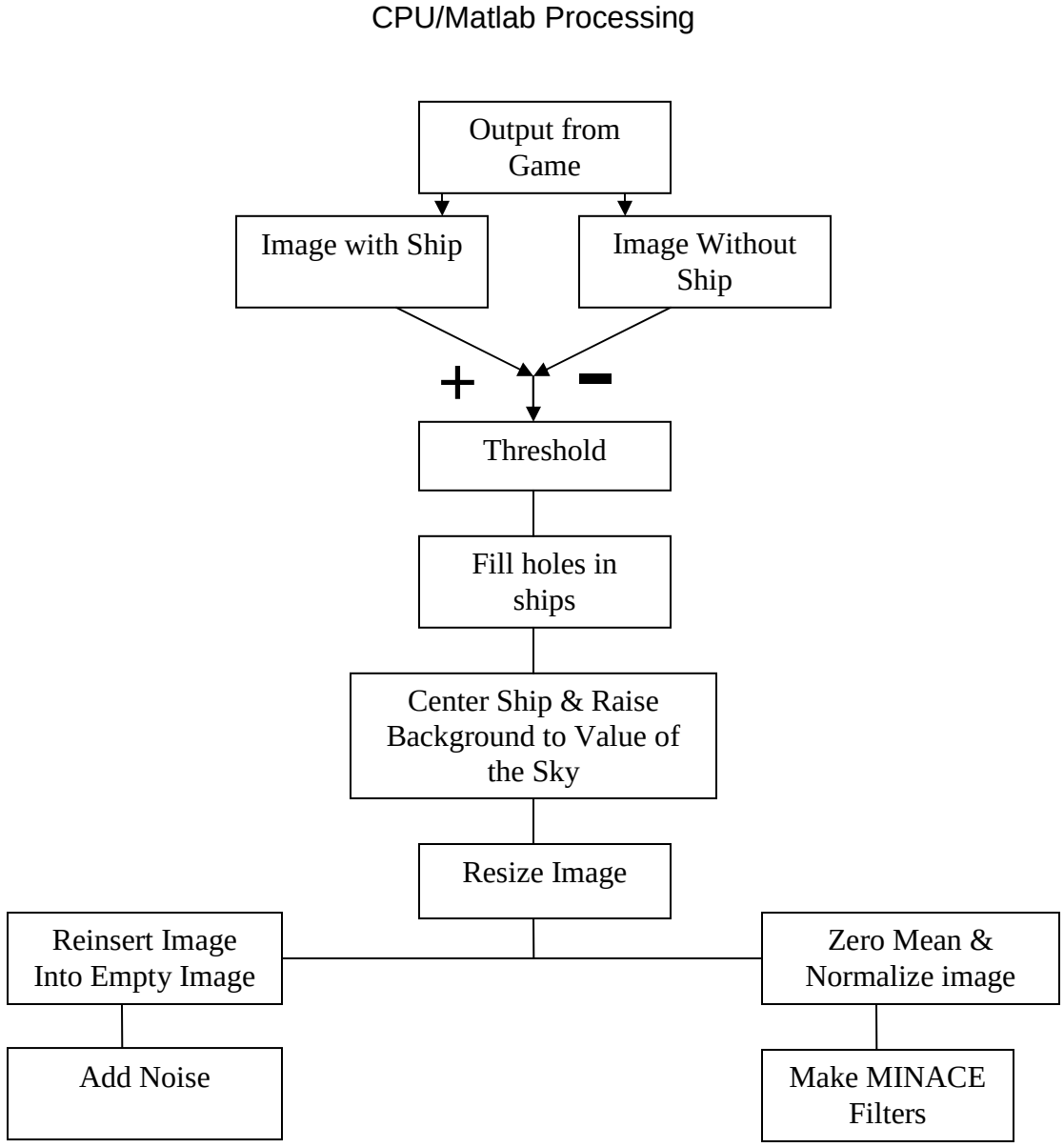
Group 6, Spring 2002 / Group 5, Spring 2001 – Finger print recognition. These projects had to deal with distortion issues that were similar to issues encountered in our project.

Group 7, Spring 2002 – Road sign recognition. This group needed to resize images but most processing was accomplished through a morphological process.

Group 2, Spring 2000 - Automatic Target Recognition in Synthetic Aperture Radar Images. This project did not use MINACE filters.

Group 15, Spring 2000 – Object Tracking. Image processing that used different algorithms than we used.

DATABASE SYNTHESIS



In order to make the project interesting, we were instructed by Professor Casasent and Rohit to process images of ships at a distance such that their broad side views (090 pose) were 60 pixels long. We could not take screen shots from the game with the ships at such ranges. This was due to the fact that the game automatically ignores rendering pixels of ships that are too far away. In order to overcome this problem we took screen shots from the game with the ships at relatively close

ranges and then extracted the ship from the background, raise the background to a constant value, resize the image, and insert the image back into an empty background on the horizon.

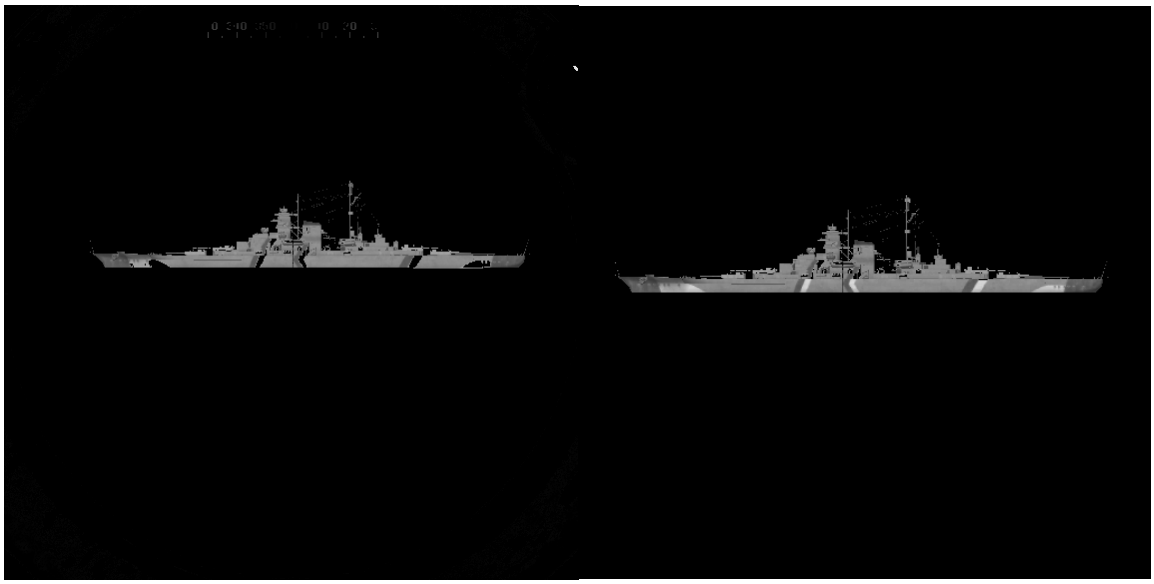
To get the game images we installed the game and used the mission editor to place the ship at the desired location and range. We then saved and exited the mission, loaded the saved mission, saved a screen shot, exited the game and saved the screen shot as a bitmap. We then reentered the mission editor and changed the ship's location again. Due to the high level of graphics and a relatively old computer it took approximately 5 minutes to acquire each of the 72 (18 X 4) images that were used to create validation images. When we went through the same process to create the testing images we found a way to speed up the process by using the mission editor to place several ships in a single mission at different bearings. This allowed us to get game output quicker but made extracting the ship from the background more difficult due to the changing location of the sun and the illumination of the background.

Output from Game

Once we had the output from the game we needed to extract the image. To accomplish this we subtracted the image from the background. We then thresholded the image to get rid of unwanted sky pixels. But a there was a trade-off between getting rid of these left over pixels and losing values on the ship. The higher the threshold was raised the more pixels that were near the value of the sky

were removed. We continued to raise the value of the sky until the background no longer touched the ship. At this point the ship had several “holes” in it. Initially we tried morphology to fill the holes but due to the location of the hole, it introduced too much distortion in the superstructure of the ship. Most of the holes were in the hull and were easy to identify visually. To fill the ship, we used an image editor to manually place white boxes over areas of the ship that had holes and erased any remaining pixels. We then used Matlab to replace each white pixel with the pixel value in the original image.

Image subtracted from empty image and threshold Image after the holes are filled

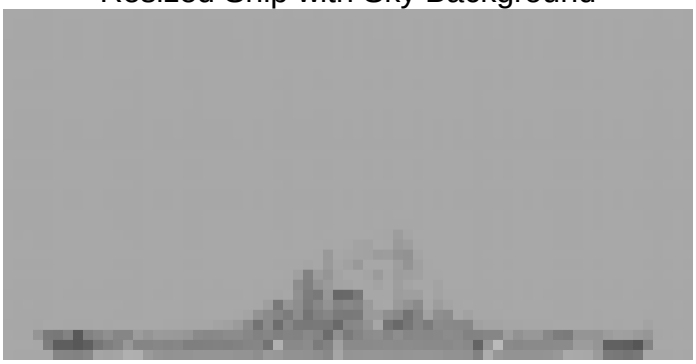


Once the image was filled we centered the image. To do this we looked at bottom row to see if there ware any nonzero values, then we looked at the next row higher until we found a row that had a nonzero value. We took the average of the first and last nonzero value and specified that to be the bottom middle of the ship. We then moved the ship so that the bottom middle was at location (255,255).

We next raised the background to the value of the sky, then resized the image in matlab using bilinear interpolation. We did this to make sure that the pixels were averaged with the background

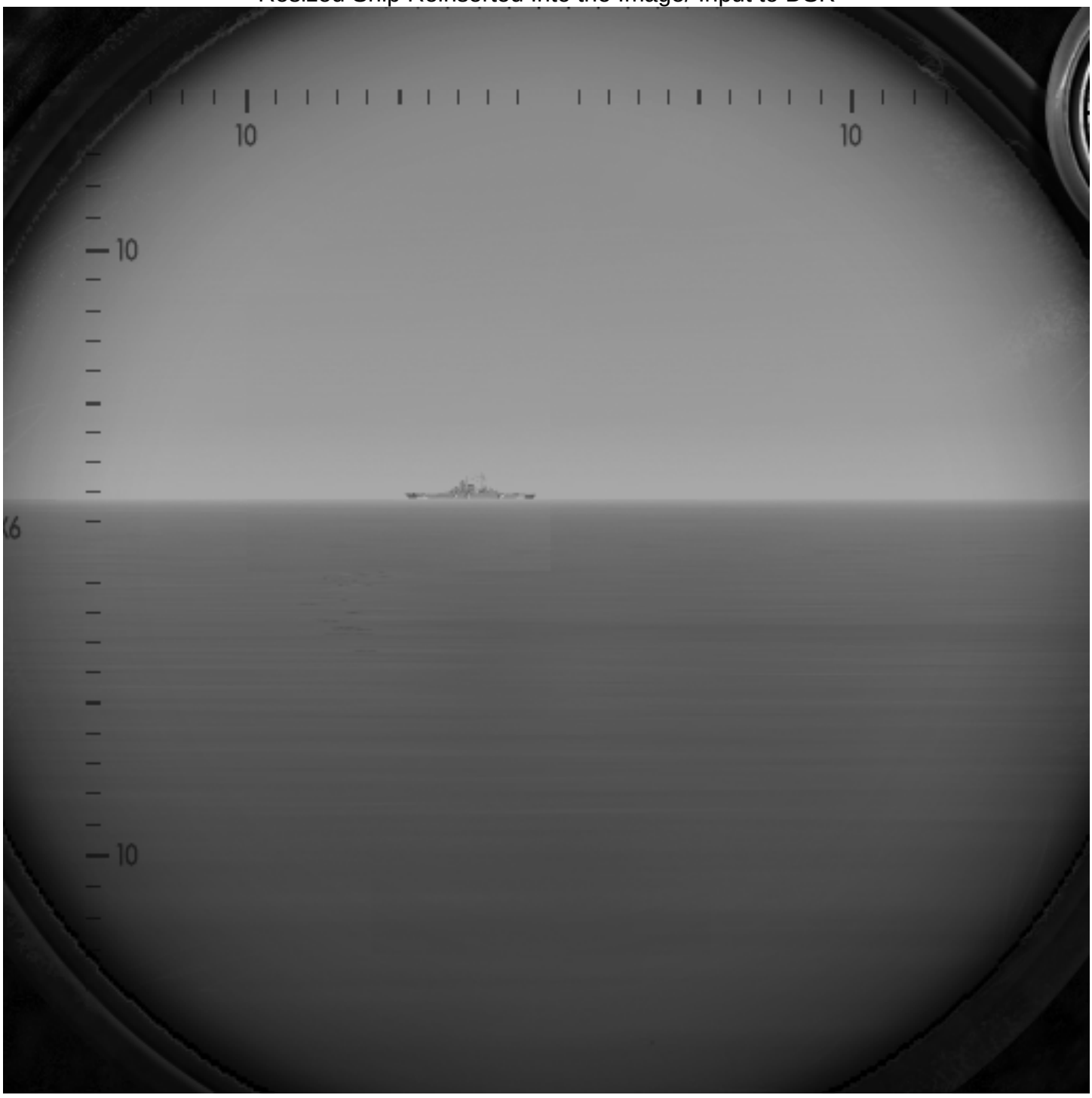
and not zero. Otherwise there would be a dark line around the ship, where values of the ship merged with values of the zero background creating low nonzero value pixels. This would add unwanted noise which would make the image unrealistic. Each of the classes were resized different amounts based on their original size, therefore the images are different sizes. We removed all but the center 73 X 73 pixels, making all the classes the same size.

Resized Ship with Sky Background



At this point making the filter diverged from making the validation and test images. In making the validation and test set we removed all pixels below the bottom of the hull for easier placement of the ship on the horizon. For the validation set all ships were inserted at the same location. For the test set we placed the ship at a random location along the horizon. To insert the ship along the horizon we started at a specified (validation) or unspecified (test) index and replaced every value in the empty image with the ship image for every pixel ship pixel that is greater than (sky +5) or less than (sky-5). After this we can add Gaussian random noise to test images if necessary, but due to time constraints we were not able to adequately test these images.

Resized Ship Reinserted Into the Image/ Input to DSK



WWW.K2PDF.COM TRIAL

ALGORITHMS: HIGH LEVEL

We approach this problem using MINACE filters and linear correlation. Since no ship is greater than 27 pixels high, and we input a 512 x 512 image, it is highly desirable to restrict the region in which we apply our filters. We consider it reasonable to restrict the region in which we correlate to a strip of the image along the horizon. This is a fair restriction in so far as it is fair to assume that ships cannot fly and appear above the horizon. Once we know where the horizon is, we define a strip that is 10 pixels wider that the size of our filter to allow a small amount of error (less than 1%) in the horizon detection. We then linearly correlate inside of the 37 x 512 strip and our correlation plane output is 11 x 512. After being unsuccessful in classifying ships using only one filter per class, we elected to use two filters per class. We divide up the poses from 0 degrees to 160 degrees for the first filter of every class and 180 degrees to 340 degrees for the second filter of every class. Once each of the 8 filters runs, only one of the filters should return a value larger than 0.85, and that filter will determine the class. The location of the peak in the 11 x 512 correlation plane determines the location of the ship in the 512 x 512 image. This location is calculated on the board and sent back to the PC along with the class. The GUI reads this information and re-displays the input image with a class-specific box around the calculated location.

ALGORITHMS: MINACE FILTERS

For each filter, there exists a training set, a validation set and a test set. The images in the training set consist of all the images that could actually be included in the filter. The validation set consists of the initial training set in addition to the images from the training sets of the other classes in the came annular range. For example, the validation set of the filter that is supposed to classify the British Nelson class destroyer from 180 degrees to 340 degrees, the validation set will consist of

images of the British ship from 180 degrees to 340 degrees and images of the other three classes of ships from 180 to 340 degrees. The validation set for this filter does not include images of the other ships between 0 and 160 degrees. The test set for each filter is the same. It is a set of images of each class of ship at the following poses: 5, 10, 25, 30, 45, 50, 65, 70, 85, 90, 105, 110, 125, 130, 145, 150, 165, 170, 185, 190, 205, 210, 225, 230, 245, 250, 265, 270, 285, 290, 305, 310, 325, 330, 345 and 350. There are 36*4 in all, since there are 4 classes of ships, and the location of the ship in the input image is different for each test image. MINACE filters are defined in *Filter Classification Algorithms for ATR Using MSTAR Data,* by Rohit Patnaik and David Casasent. The following is from their paper:

3. MINACE FILTER THEORY

This section describes the version of the original MINACE filter[7] that we use. Vectors (matrices) are denoted as lower (upper) case bold letters. All data are in the Fourier Transform (FT) domain. The 2-D FT of the filter is lexicographically ordered into a column vector $\mathbf{h}$. The 2-D FT of each training set image included in the filter is lexicographically ordered into a column vector $\mathbf{x}_i$ of the data matrix $\mathbf{X}$. The filter is required to give a specified correlation peak value for each training set image included in the filter; these values (usually one) are specified by the elements of a column vector $\mathbf{u}$. These *peak constraints* are described by

$$\mathbf{X}^{+}\mathbf{h} = \mathbf{u} = [1 \quad 1 \ldots 1]^{T}, \tag{1}$$

Lexicographically ordered means a matrix is transformed into a column vector. This is accomplished by taking a row of a matrix, transposing it, and placing the next transposed row beneath it. So a 12 x 37 matrix becomes a 444x1 column vector with every 37 elements corresponding to a row from the original matrix. Continuing in the paper:

where $(\ )^{+}$ denotes the complex conjugate (Hermitian) transpose. To improve performance, the filter $\mathbf{h}$ is also required to minimize a combination of correlation plane energy due to training images and correlation plane energy due to distorted versions of the objects to be recognized. We use zero-mean white Gaussian noise to model the expected distortion power spectrum. We choose the energy function to be minimized as

$$E = \mathbf{h}^{+}\mathbf{T}\mathbf{h}, \tag{2}$$

where $\mathbf{T}$ is a diagonal matrix whose diagonal entries are the spectral envelope of the training images and noise at each frequency. That is,

$$\mathbf{T}(k,k) = \max[\mathbf{S}(k,k), c\mathbf{N}(k,k)], \tag{3}$$

$$\mathbf{S}(k,k) = \max[\mathbf{S}_1(k,k), \mathbf{S}_2(k,k), \ldots, \mathbf{S}_{N_T}(k,k)], \tag{4}$$

where $\mathbf{S}_i$ is a diagonal matrix whose diagonal entries are the elements of the lexicographically ordered 2-D power spectrum ($|FT|^2$) of training image i, N_T is the total number of training images, $\mathbf{N}$ is the identity matrix, the maximum value in $\mathbf{S}$ is normalized to one, and c ($0 \le c \le 1$) controls the variance of the noise. The Lagrange multiplier solution that minimizes the expression in Eq. (2) subject to the constraints in Eq. (1) is

$$\mathbf{h} = \mathbf{T}^{-1}\mathbf{X}(\mathbf{X}^{H}\mathbf{T}^{-1}\mathbf{X})\mathbf{u}\,. \tag{5}$$

It should first be noted that there is a typo in equation (5) which was not realized until several weeks into the project. Rohit pointed out to the group one day when we were asking him for help with our strange results. Once this typo was fixed, our results were correct. The correct formula for **h** is:

$$\mathbf{h} = \mathbf{T}^{-1}\mathbf{X}(\mathbf{X}^{H}\,\mathbf{T}^{-1}\mathbf{X})^{-1}\,\mathbf{u}$$

Since we linearly correlate in our project, rather than find the correlation using the FFT of the input image, our final filter is the inverse FFT of the de-lexicographically ordered **h**.

ALGORITHMS: MINACE FILTER SYNTHESIS

Following the guidelines set out in Professor Casasent's and Rohit's paper and those guidelines given to us verbally, we restrict the number of images included in each filter and automate the selection of the MIANCE filter parameter *c.* We create two filters per class by dividing the pose range in half. We manually select an initial value for *c* and an initial image from a training set of 9 images to include in the filter by defining **X** as the initial image's lexicographically ordered 2-D FFT. We then calculate **h** and correlate the filter with the 9 image training set. We then reiterate the process now including the 2-D lexicographically ordered FFT of the image with the lowest correlation peak in the filter. We do this until either all the images have a correlation peak above 0.9, or we have 8 training images included in the filter. If we have 8 images in the filter and a correlation peak is below 0.9, the filter is likely to fail in the next step, validation, and be subject to re-synthesis which will eliminate the low peak. Once we have a filter, we then validate it by correlating the filter with its own training set as well as the training set of the other three classes. If a correlation peak for any of the

images in the filter's own training set is below 0.85, we increase *c* and re-synthesize the filter. If we reach a point where all the correlation peaks of the filter with its own training images are above 0.85 but there are other peaks from one of the other training sets also above 0.85 we raised the 0.85 threshold to 0.9. If this raised threshold does not work we must conclude that a 9 image training set is insufficient. We only have one class for which we must use a 0.9 validation threshold however no classes for which the 9 image training set was insufficient.

In order to speed up the process and fine-tune the selection of *c*, we first use large increments whenever the filter must be re-synthesized. This process yields a range for *c* within which we re-iterate the whole process using smaller increments. In many cases, multiple values of *c* yield a filter which detects the correct ship in validation 100% of the time. In these cases, we chose the lowest value of *c* because lower values of *c* suppressed the correlation peaks of the other three classes. Also, once an appropriate value of *c* was determined, if *c* continues to increase we have an increase rate of false detection of wrong (confuser) classes. The following figures illustrate this issue:

Figure 1: The percentage of correlation peaks above the threshold for increasing values of *c*. Circles represent peaks from the correct filter, which is the German filter in this case

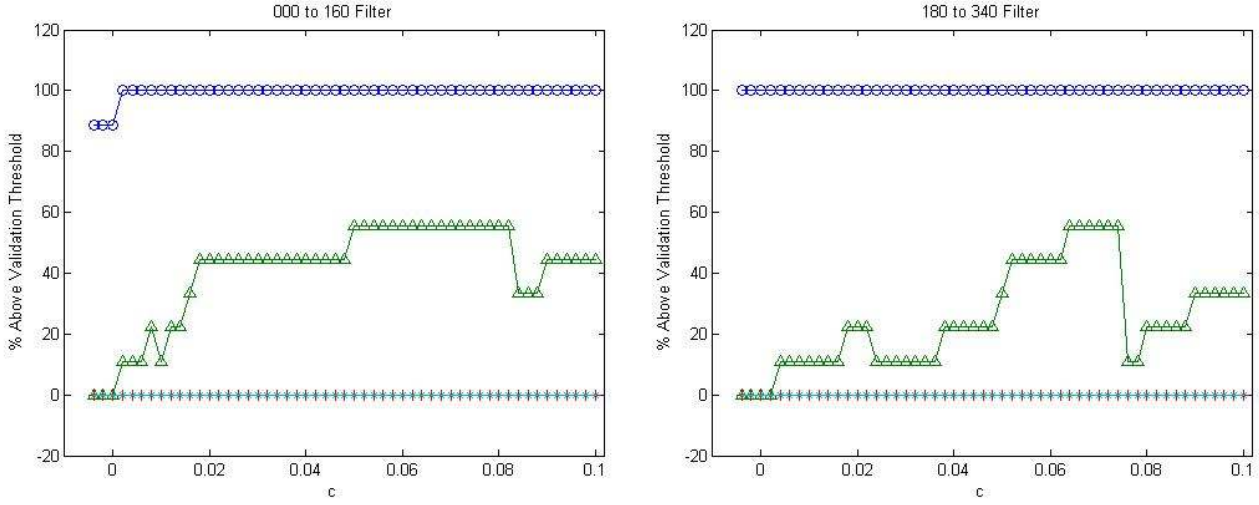


Figure 2: The percentage of correlation peaks above the threshold for increasing values of *c*. Triangles represent peaks from the correct filter, which is the Nelson filter in this case

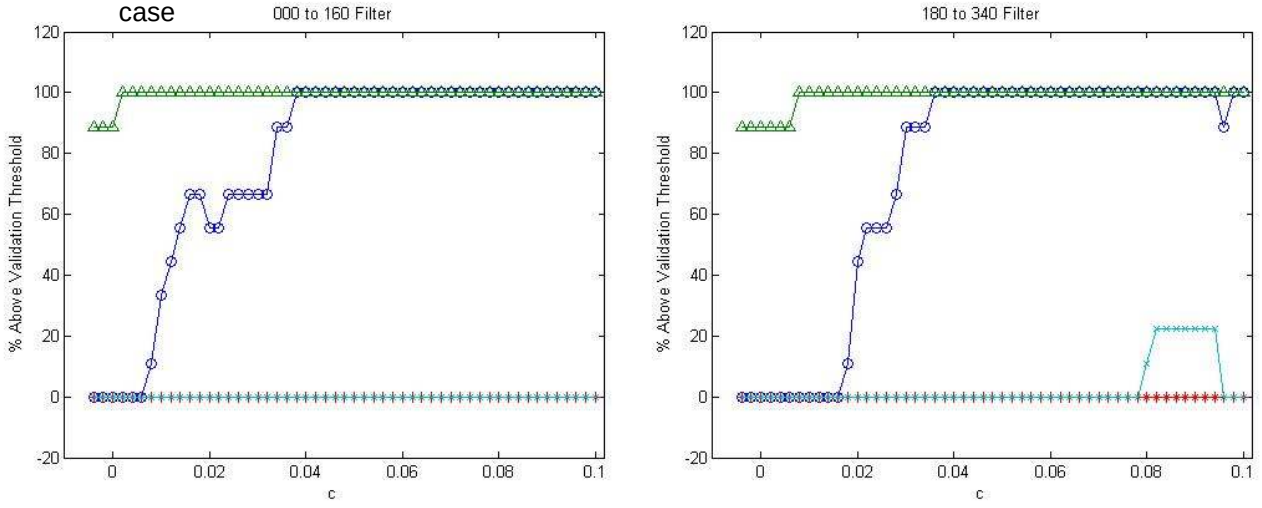


Figure 3: Percentage of correlation peaks above the threshold for increasing values of *c*. Asterisks represent peaks from the correct filter, which is the American filter in this case

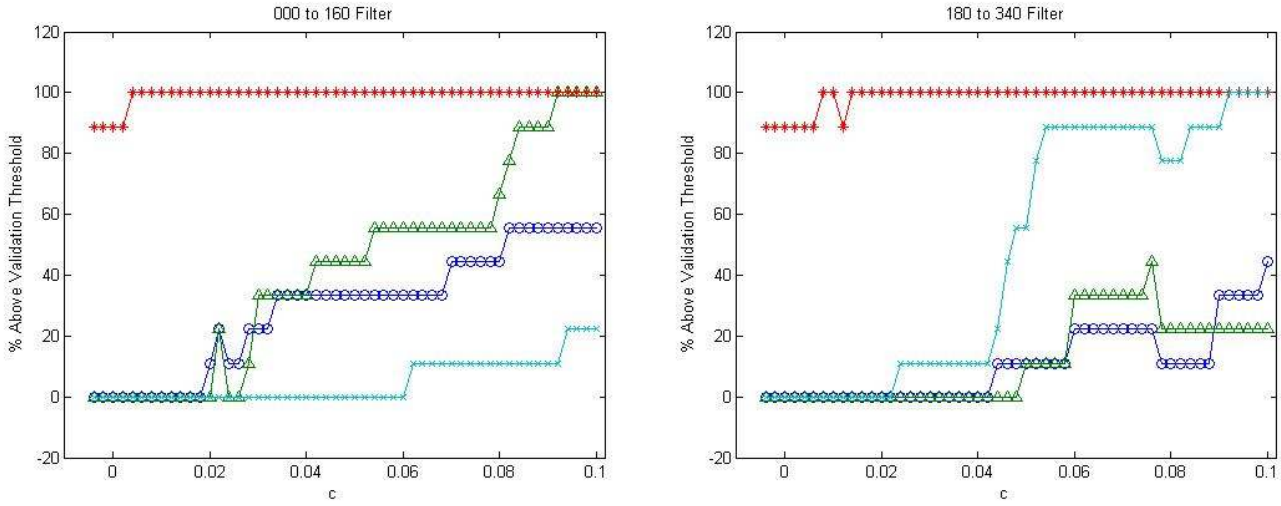
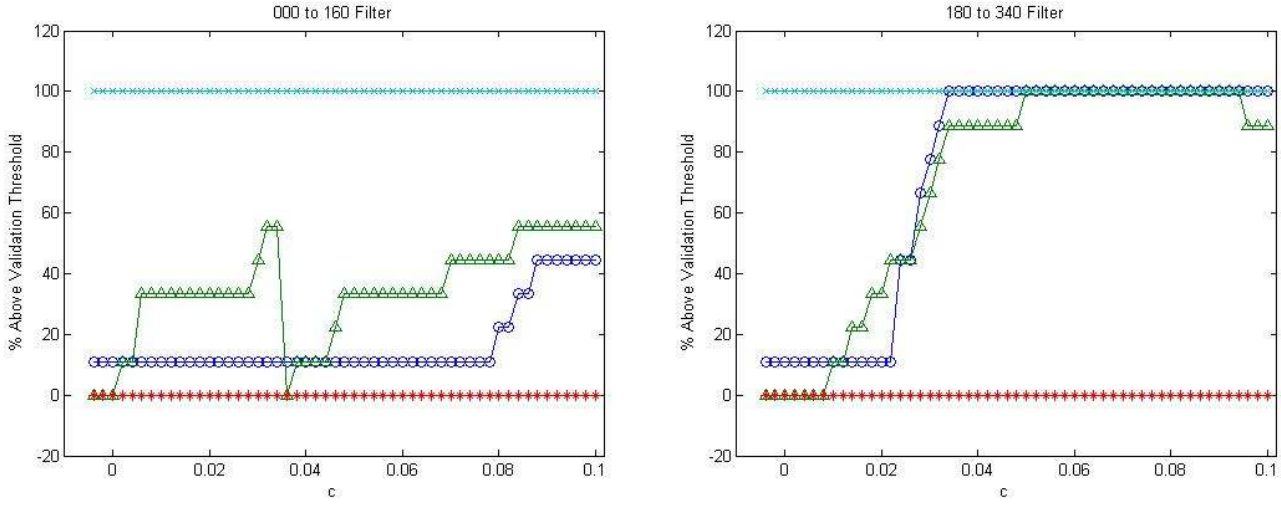


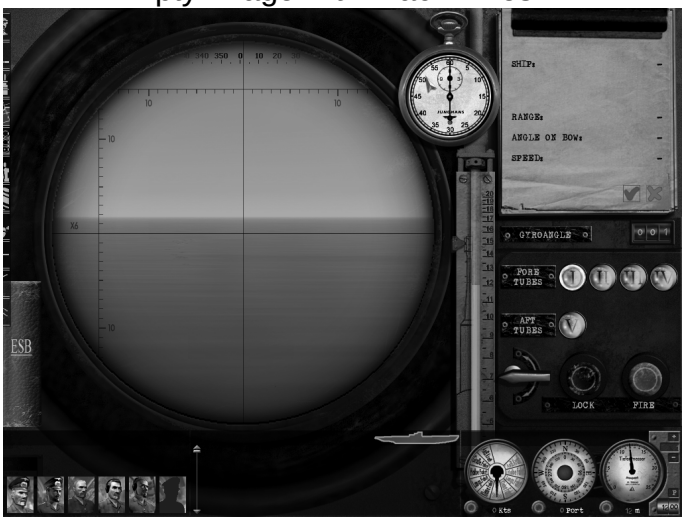
Figure 4: Percentage of correlation peaks above the threshold for increasing values of *c*. X's represent peaks from the correct filter, which is the Merchant filter in this case



ALGORITHMS: HORIZON DETECTION

There are two horizontal lines and two vertical lines that run horizontally and vertically through the game images representing the cross-hairs of the periscope. Before we detect the horizon we replace the horizontal black lines with the pixels that are right above it and the vertical black likes with the pixels to the left of it. We do this because the horizon detector locates the horizon based on changes in pixel values. Due to the fact that there is a greater difference between the value of the sky and 0 than there is between the value of the sky and the value of the ocean, the horizon detector will locate the black lines instead of the horizon.

Empty Image with Black Lines



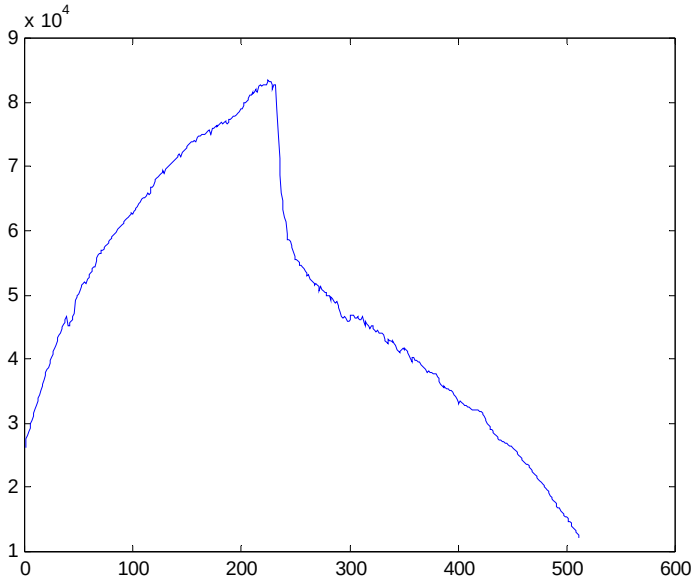
Empty Image with Black lines Removed

Detecting the Horizon

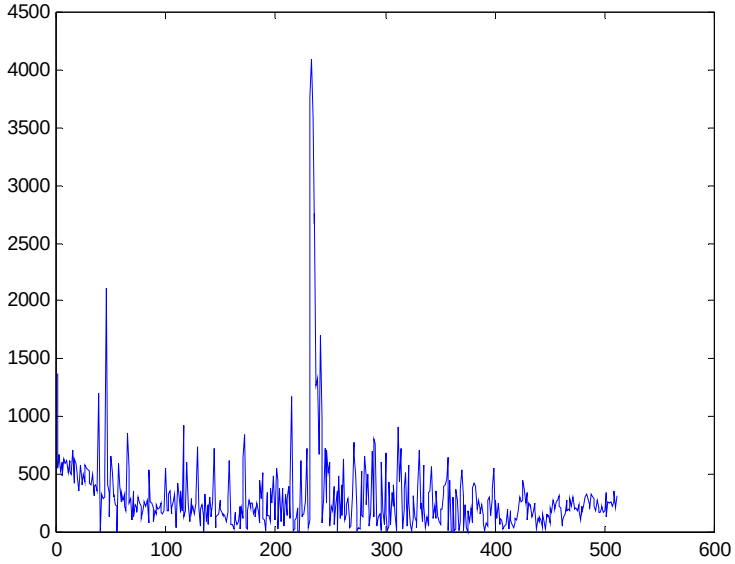
To detect the horizon we sum the values of each pixel in each row of the image (512 X 512). This gives us a 512 element long column vector. The plot for these values is displayed below. Because the ocean and the sky are different colors and the transition is relatively sharp we can

identify the horizon by locating the largest difference between the sum of any two consecutive rows. We find this by taking the absolute value of the difference between every two consecutive sums of rows. The largest value is the vertical location of the horizon. The second peak you see around index 50 is due to the black hash marks at the top of the images. These are significantly lower than the horizon.

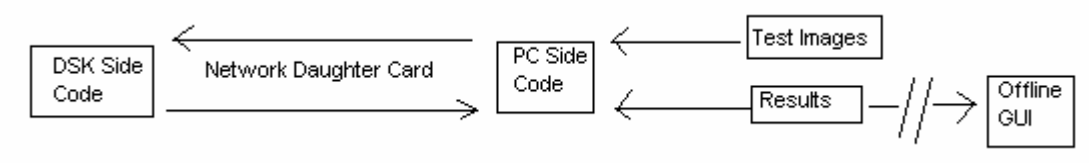
Sum of Each Row in Image

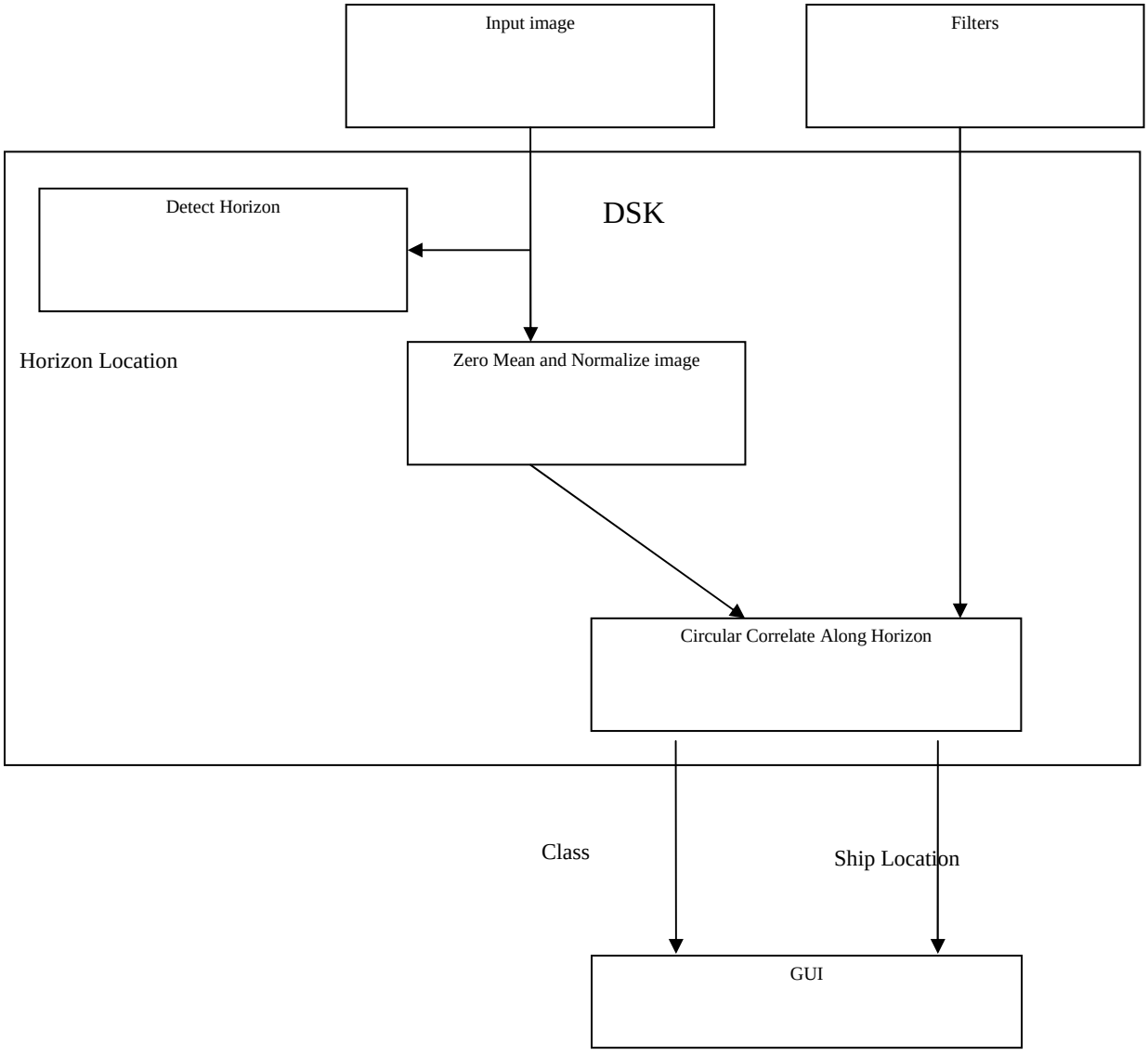


Absolute Value of Difference Between Each Sum



IMPLEMENTATION





Memory, Paging, and Speed

Images are transferred from the PC to the DSK via a network daughter card. The PC acts as the host (also the slave) and the DSK acts as the client (also the master). Each image transferred to the DSK is 512 by 512 single precision floating point, 1 MB in size. 15 total images and 8 filters (73 by 27 single precision floating point, 61.6 KB each) are stored in the external memory of the DSK at any one point in time and take approximately 7 seconds to transfer. When the DSK finishes processing an image it sends the result back to the PC. The result consists of the ship type, box location and dimensions, and values of the correlation peaks. These results are then written to a file by the PC. These files are then optionally read into the GUI, which is separate from the PC side of the processing and acts in an offline manor displaying results rather than participating in the result calculation. When the DSK has finished processing all 15 images it checks with the PC whether more images are to be processed. If they are then the PC sends the next set, up to 15, and the process repeats until all images have been processed.

The internal memory of the DSP is allotted to the following: 61.6 KB are used to store the “working filter”. A “working image” of 512 by 37 single precision floating point, 18.5 KB is stored in internal memory. Between the “working filter”, “working image”, two lookup tables, various other small variables, and a relatively large text section, 245.3 KB of the 250 KB of internal memory are utilized. Cache is obviously not used which is acceptable because memory transfers are done via EDMA. All computationally intensive operations are performed with data paged into the internal memory of the DSK.

Horizon Detection

Horizon detection is performed by successively paging into the “working image” chunks of rows from the current image being processed. Each row is summed and the magnitude of the difference between the sums of the values in 2 adjacent rows is computed. A running max is kept. The biggest magnitude of difference represents the horizon.

Linear Correlation

Based on the calculated row of the horizon, one “working image” is paged into internal memory consisting of 37 rows (10 rows more than a filter) beginning 5 rows below the horizon. One “working filter” is paged into internal memory via EDMA. For the “working filter” and “working image” a lookup table for the zero mean and the energy of any given correlation offset is computed. These lookup tables are each 11 x 512 and each entry corresponds to a frame in which we correlate. The general equation for correlation is $\sum h(n-i)x(n)$. After we zero-mean and energy normalize, the x(n) term becomes $(x(n) - \mu_i)/\text{sqrt}(\partial)$ where μ_i is the mean of the i’th frame and ∂ is the max energy of the i’th frame. Now the correlation term simplifies to:

$$(1/\text{sqrt}(\partial))*(\sum h(n-i)x(n) - \sum h(n-i)\,\mu_i)$$

$$= (1/\text{sqrt}(\partial))*(\sum h(n-i)x(n) - \mu_i \sum h(n-i))$$

Those lookup tables significantly decrease the amount of time we spend processing an image. Before this method was implemented the brute force linear correlation algorithm took 240 times as long to process. Using this methods approximately 1 image (8 filters) can be processed every second. The brute force algorithm had a yield of approximately one filter every 30 seconds, processing one image approximately every 4 minuets. Further optimization was attempted but was not successful. The idea was to store state on the left/bottom-most (oldest) row/column sum, subtracting the old row/column sum from the previous blocks summation then adding in the row/column of the next shift. Doing this was believed to have

the potential to decrease the computation time by approximately 1/10, however it was seen through experience/profiling that the increased logical complexity actually slowed down the algorithm. Increased logical complexity led to poor branch prediction and few parallel operations. Hence, the dual-lookup table approach was deemed the most effect.

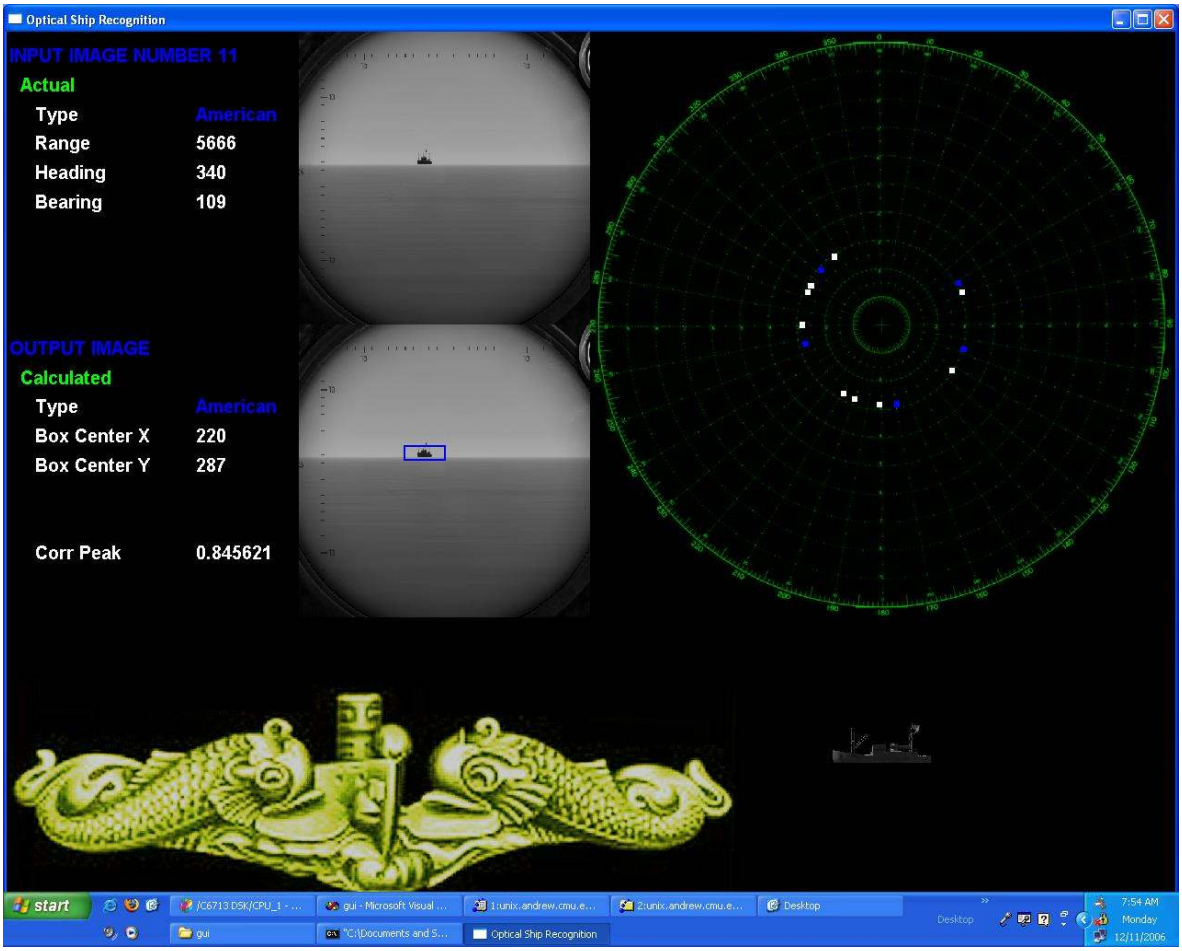
	Brute-Force	Look-up Tables	Look-up Tables with State
Time Per Image (8 filters)	4 min 22 sec	1 second	1.5 second
Time Per Filter	33 sec	Approx 1/8 of a second	Approx 3/16 of a second

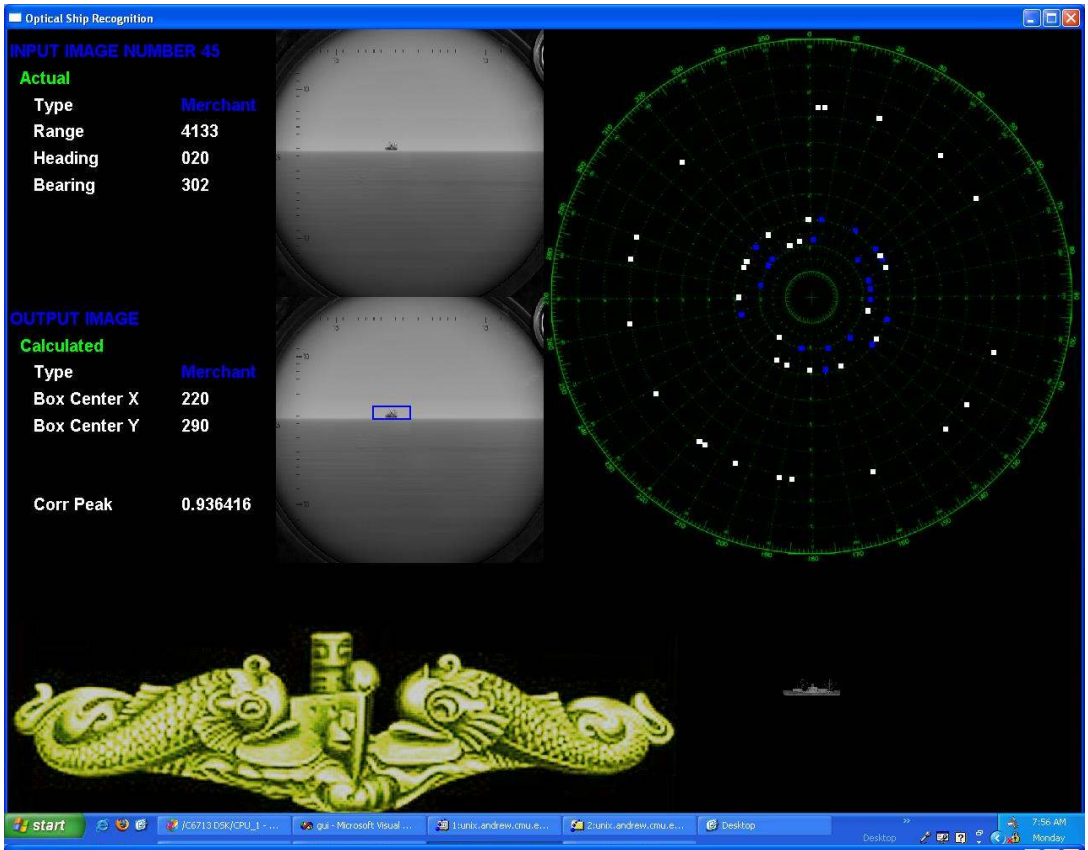
Please note that profiling was done (and it took forever to complete), however the timing differences were actually noticeable and listing these times was deemed more practical than processor cycles.

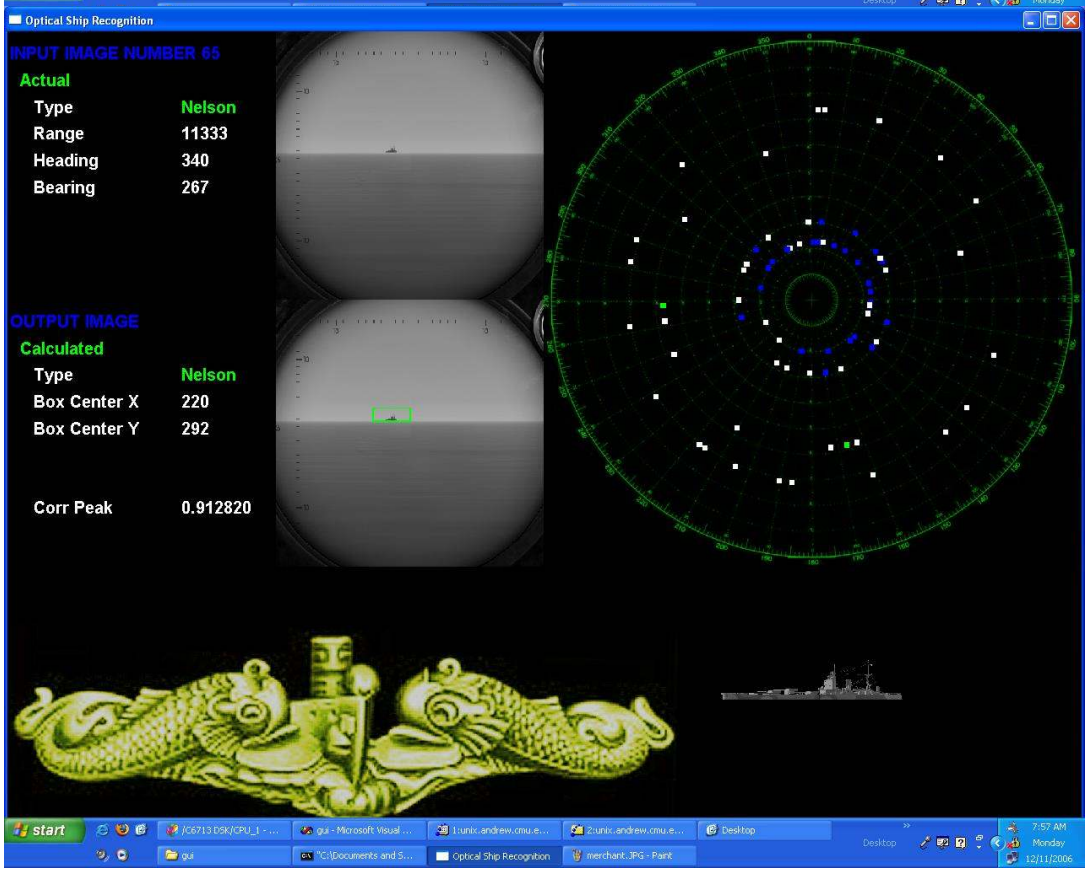
	No Optimization	O1 Optimization	O3 Optimization
Look-up Tables per Image	40 Seconds	15 Seconds	1 Second

GUI

The GUI is programmed using openGL32. In hindsight openGL was overly complex for the nature of the GUI's role in this project. The GUI runs offline. It displays results that have been pre-calculated and written to output files. The reason for the GUI being offline rather than real-time is the lack of ability for the DSP board to trigger a socket-ready-for-reading in a PC side select statement (non-blocking transfers would be required for the GUI to properly render). The GUI is not involved in any aspect of the image processing or filtering, only in displaying the results. GUI screenshots are displayed below.







INITIAL RESULTS

We first implemented 4 classification filters each of which had a training set of 18 images. We applied the same filter synthesis algorithm described earlier with the only difference here being that the maximum allowable number of filters was 16, not 8. In validation, the American and Merchant class filter's performed very well with a validation threshold of 0.84. The German and Nelson filters however did not do well. The German filter performed quite poorly in validation, it yielded peaks that were inconsistent and always below threshold. The Nelson filter yielded peaks that were mostly below threshold however they were all in a consistent range around 0.7. Unfortunately, 0.7 cannot qualify for detection so under the advice of Professor Casasent and Rohit, we re-approached the problem using two filters per class. The following table summarizes the initial results:

Table 1: Initial results using a 4-filter approach on the validation set

Filter	Correct Detection Above Threshold	Wrong Detection Above Threshold	Below Threshold
German	1	0	17
Nelson	3	0	15
American	15	0	3
Merchant	14	0	4

FINAL RESULTS

We ran our 8 filters from the 2 filter per class method and obtained the following results:

Table 2: Final results using an 8-filter approach on the test set

Filter	Correct Detection Above Threshold	Wrong Detection Above Threshold	Below Threshold
German	9	0	47
Nelson	0	0	36
American	4	0	32
Merchant	0	0	36

The test set consisted of ships at poses the filter had never seen during the design process. We did not know how good of results to expect when we ran this set. We ran the 8 filters on the same validation set as before and obtained no better nor worse results. We can only conclude that the two warship classes, the German and the Nelson simply have too low of a signal to noise ratio to be detected when their broadside is only 60 pixels long. This initially sounds un-reasonable however ships are special cases in and of themselves. The warships have a much higher length to width and length to height ratio than do the Merchant and American class ships. The Merchant and American ships are in a word, fatter than the warships. In every image under our 60-pixels at broadside requirement, the Merchant and American simply had more pixels of ship present in the image. We believe we have found a threshold range for detection in this project. If we have learned anything or demonstrated anything with this project, we can certainly detect vessels 60 pixels at broadside; however we cannot classify certain vessels that do not have large enough length to height ratios. We have established a boundary for classification.

Additional tests were run with varying image position, image backgrounds, and noise. Very dark backgrounds and noise caused false detections in the majority of cases. Very light backgrounds caused the horizon detector to fail which in turn lead to false detections. In many instances the correct filter yielded the highest value and if the max was taken the ship would have been correctly classified, however the value was below the threshold established and hence was considered to have been not detected..

THINGS WE WANTED TO DO, BUT COULDN'T, THAT FUTURE PROJECTS COULD DO

Initially, we wanted to design multiple layers of filters to accommodate multiple ranges of ships. Simply due to time constraints, we had to restrict ourselves to a single range for each class.

We also wanted to design 18 filters for each class that we would use to determine the ship's pose. These ended up being more difficult to design than anticipated. We attempted to design the so-called pose filters with a T matrix that included all 18 images and a T matrix that only had the single image however both methods were unreliable. Pose determination could be a very important part of a future project since the pose effectively determines the ships course.

Since the periscope perspective rarely includes images of clutter, future projects may wish to do more research before deciding what type of filter to use. MINACE filters may or may not have been the best choice for this project since they are specifically designed to handle clutter rejection.

While we managed to optimize our correlation algorithm such that processing of a single image with a single filter tool 1/8 of a second however this algorithm could still be made even quicker, perhaps even qualify as real-time if the correlation was obtained using FFT's.

References

List of Web References

Starter code for the GUI was used to create a single window - GUI was built up from there. Credit for starter code goes to Ronny André Reierstad, www.morrowland.com, apron@morrowland.com.

WHO DID WHAT

- Greg
  - Made the GUI
  - Wrote all of the C code - all of it
  - Handled all DSK matters
  - Ran test sets and recorded results
  - Discovered that select-read is not triggered by the DSP board
  - Re-coded the changing project again, and again, and again…
  - Optimized/profiled/timed code

- Ted
  - Researched MINACE filter theory (twice)
  - Exported 198 images from game
  - Wrote lots of Matlab
  - Code to remove background from images
  - Code to automate the number of images included in the filters
  - Code to automate the choice of c, the MINACE filter parameter
  - Code to check the correlation results
  - Code to design the 4-filter approach
  - Code to design the 8-filter approach that split the poses in half
  - Code to design the 8-filter approach that split the narrow poses from the wide poses
  - Had nightmares about code

- Tyler
  - Exported 18 images from game
  - Filled holes in validation and training images
  - Wrote and coded the horizon detection algorithm in Matlab
  - Code that generated the validation set
  - Code that generated 2880 test images, with varying noise, location on horizon, size, and class
  - Presentation

Work Schedule

- Week Of 22 OCT
  - Project radically changes
  - Figure out new way to get database
  - Compile more training images
- Week of 29 OCT
  - Compile more training mages
  - Design and test method of horizon detection
  - Create non automated MINACE filter
- Week Of 5 NOV
  - Validate system on training images
  - Fill holes and resize ships
- Week of 12 NOV
  - Code
  - Validate system on training images
- Week Of 19 NOV
  - Implement system output Algs and GUI
  - Validate MIANCE filters in matlab
- Week Of 26 NOV
  - MIANCE filters in matlab
  - Making the filters work on the DSK
  - Create testing database with noise
  - Finishing GUI
  - Prepare for Demos
- Week Of 3 DEC
  - Demo for professor and TAs
  - Address professors comments
  - Get GUI to function properly
  - Use 2 MINACE filters per class
  - Complete final paper

WWW.K2PDF.COM TRIAL