ATRENGINE:
AN ORIENTATION-BASED ALGORITHM
FOR AUTOMATIC TARGET RECOGNITION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Justin Ting-Jeuan Kuo

June 2014

COMMITTEE MEMBERSHIP

TITLE:	ATREngine: An Orientation-Based Algorithm for Automatic Target Recognition
AUTHOR:	Justin Ting-Jeuan Kuo
DATE SUBMITTED:	June 2014
COMMITTEE CHAIR:	Dr. John Saghri, Professor Electrical Engineering Department
COMMITTEE MEMBER:	Dr. Jane Zhang, Associate Professor Electrical Engineering Department
COMMITTEE MEMBER:	Dr. Bridget Benson, Assistant Professor Electrical Engineering Department

ABSTRACT

ATREngine: An Orientation-Based Algorithm for Automatic Target Recognition

Justin Ting-Jeuan Kuo

Automatic Target Recognition (ATR) is a subject involving the use of sensor data to develop an algorithm for identifying targets of significance. It is of particular interest in military applications such as unmanned aerial vehicles and missile tracking systems. This thesis develops an orientation-based classification approach from previous ATR algorithms for 2-D Synthetic Aperture Radar (SAR) images. Prior work in ATR includes Chessa Guilas' Hausdorff Probabilistic Feature Analysis Approach in 2005 and Daniel Cary's Optimal Rectangular Fit in 2007.

A system incorporating multiple modules performing different tasks is developed to streamline the data processing of previous algorithms. Using images from the publicly available Moving and Stationary Target Acquisition and Recognition (MSTAR) database, target orientation was determined to be the best feature for ATR. A rotationally variant algorithm taking advantage of the combination of target orientation and pixel location for classification is proposed in this thesis. Extensive classification results yielding an overall accuracy of 76.78% are presented to demonstrate algorithm functionality.

Keywords: Automatic Target Recognition, Synthetic Aperture Radar, Image Processing, Machine Learning, MSTAR

ACKNOWLEDGMENTS

I express my gratitude to Dr. John Saghri for his time and patience in advising me on this thesis. He was very passionate about the subject of Automatic Target Recognition and obtained a commitment from Raytheon to sponsor this project. He also provided invaluable insight into image processing and recognition techniques.

I thank Dr. Jane Zhang for serving on the thesis committee.

I thank Dr. Bridget Benson for serving on the thesis committee.

Special thanks to Daniel Cary for the foundation he built with his work in Automatic Target Recognition. The filtering methods of his Optimal Rectangular Fit project provided a starting point for my thesis.

Finally, I thank my parents for their support throughout my academic career. Their efforts over the years have greatly contributed to my success.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Automatic Target Recognition applications often employ Synthetic Aperture Radar for its ability to penetrate various weather and light conditions. For this reason, ATR research at Cal Poly has been conducted using SAR images from the public MSTAR database. Over the years, students have developed various software algorithms for classifying targets of military significance. The incremental functionality contributed by each project yields new possibilities for target recognition along with additional research opportunities for future students.

The purpose of this project is to develop an algorithm that matches an unknown target to an averaged image of one of eight different military vehicles created using the orientation of training images. Image conversion and filtering techniques from previous projects are utilized in the preprocessing step of this algorithm. The ability to automatically generate multiple data sets for testing algorithm reliability is a major strength of this project when compared to previous projects which present results for a single data set. Classification accuracy, processing speed and algorithm flexibility are metrics taken into consideration during the development of this project.

CHAPTER 2

Prior Work

Chessa Guilas developed the Hausdorff Probabilistic Feature Analysis algorithm in 2004 for her Master's thesis. Guilas' algorithm extracted the edge, corner and peak features of Synthetic Aperture Radar images for classification according to the Hausdorff distance metric. Assigning weights of 30% to edges, 30% to corners and 40% to peaks yielded a classification accuracy of 85%, sufficient for ATR. The main contribution of this project was code for converting SAR images from RAW to BMP format.

Figure 1 MSTAR Vehicles

Daniel Cary developed the Optimal Rectangular Fit algorithm in 2007 as a senior project. This algorithm attempted to use the length and width of a target as features for classification. The classification accuracy of the Optimal Rectangular Fit algorithm was capped at 44% due to the statistical overlap of lengths and widths of different vehicles. With the exception of SLICY length, the probability densities for length and width are clustered together and make it difficult to distinguish between the different vehicle types.

The main accomplishment of the Optimal Rectangular Fit algorithm was the implementation of the least squares line fit calculation, useful for determining target orientation. The feasibility of features suggested in Daniel Cary's report such as pixel area, fill ratio, length-to-width ratio and target orientation (angle of line of best fit) were initially tested using MATLAB. Of these features only target orientation was viable for use in ATR, as pixel area, fill ratio and length-to-width ratio showed very little correlation for the eight vehicle types. In this paper, a new ATR algorithm based on the processes of the Optimal Rectangular Fit algorithm will be discussed.

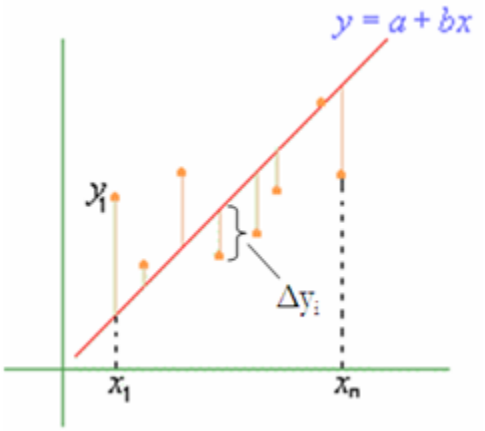


Figure 2 Linear Least Squares Calculation[1]

[1] Automatic Target Recognition via Optimal Rectangular Fit, Daniel A. Cary, 2007

CHAPTER 3

Architecture

A multitasked application to streamline and automate the process of algorithm testing was developed using C++ in Microsoft Visual Studio 2010. In prior ATR projects, each step had to be carried out manually by the user after the previous step had been completed. In the proposed automated system, new classification methods can be easily integrated into the existing algorithm as additional tasks while old methods can be similarly removed. Using Daniel Cary's Optimal Rectangular Fit code as a starting point, individual methods are linked together in a system of tasks initiated by a root task.

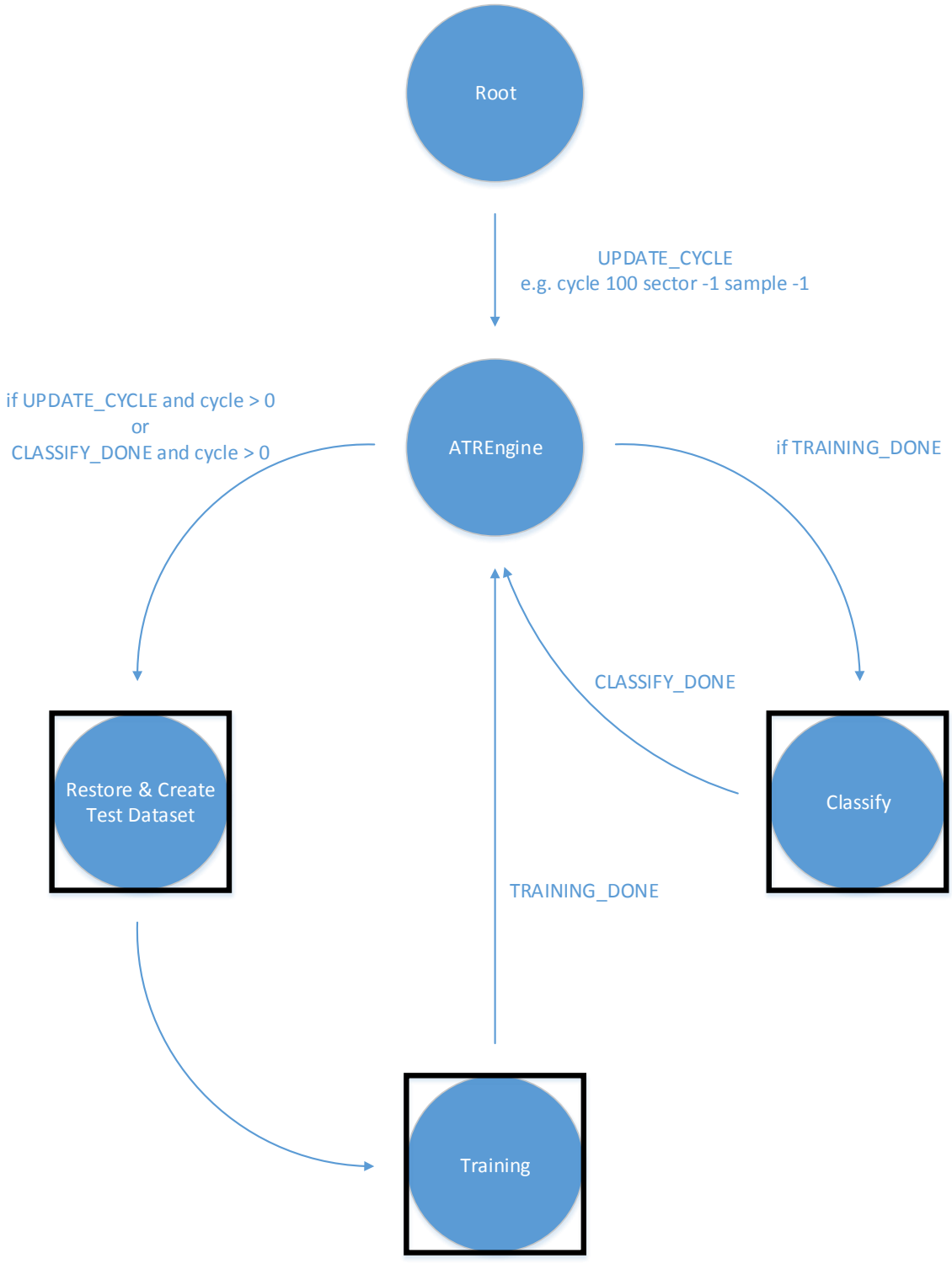


Figure 3 Automation Process

The automation process consists of a state machine maintained by ATREngine. Restore & Create Test Dataset, Training and Classify are the operations involved in the ATR algorithm. Each operation executes of a series of tasks. For example, the Classify operation includes the Classify task along with other tasks for converting and filtering images. Upon receipt of an event message triggered by the root task or from a previous operation, ATREngine will maintain or change its state and proceed with the corresponding operation. The automation process begins when the Root task receives a console command to update the cycle count and ends when the specified number of cycles has completed (See Figure 3 Automation Process).

Restore & Create Test Dataset

1. Inform Database task to clear the database.
2. Restore
3. Select unknown image set for testing (SamplePctIdx)
4. Select sector size for training (nSectSizeIdx)

Training

1. Raw2Tif(“training”)
2. HistoMedianFilter(“training”)
3. AdaptiveFilter(“training”)
4. AssignSectors
5. AverageImages
    - Update database each time an averaged image is created.
    - Inform ATREngine of TRAINING_DONE upon completion of AverageImages.

Classify

In order to simplify implementation, the following image processing functions are executed sequentially in this classification operation instead of through message passing:

1. Raw2Tif("unknown")
2. HistoMedianFilter("unknown")
3. AdaptiveFilter("unknown")
4. ClassifyTargets
    - Inform Database to clear previous classification summary.
    - Classify each filtered unknown image against trained database.
    - Inform Database to report classification summary once all unknown images complete.

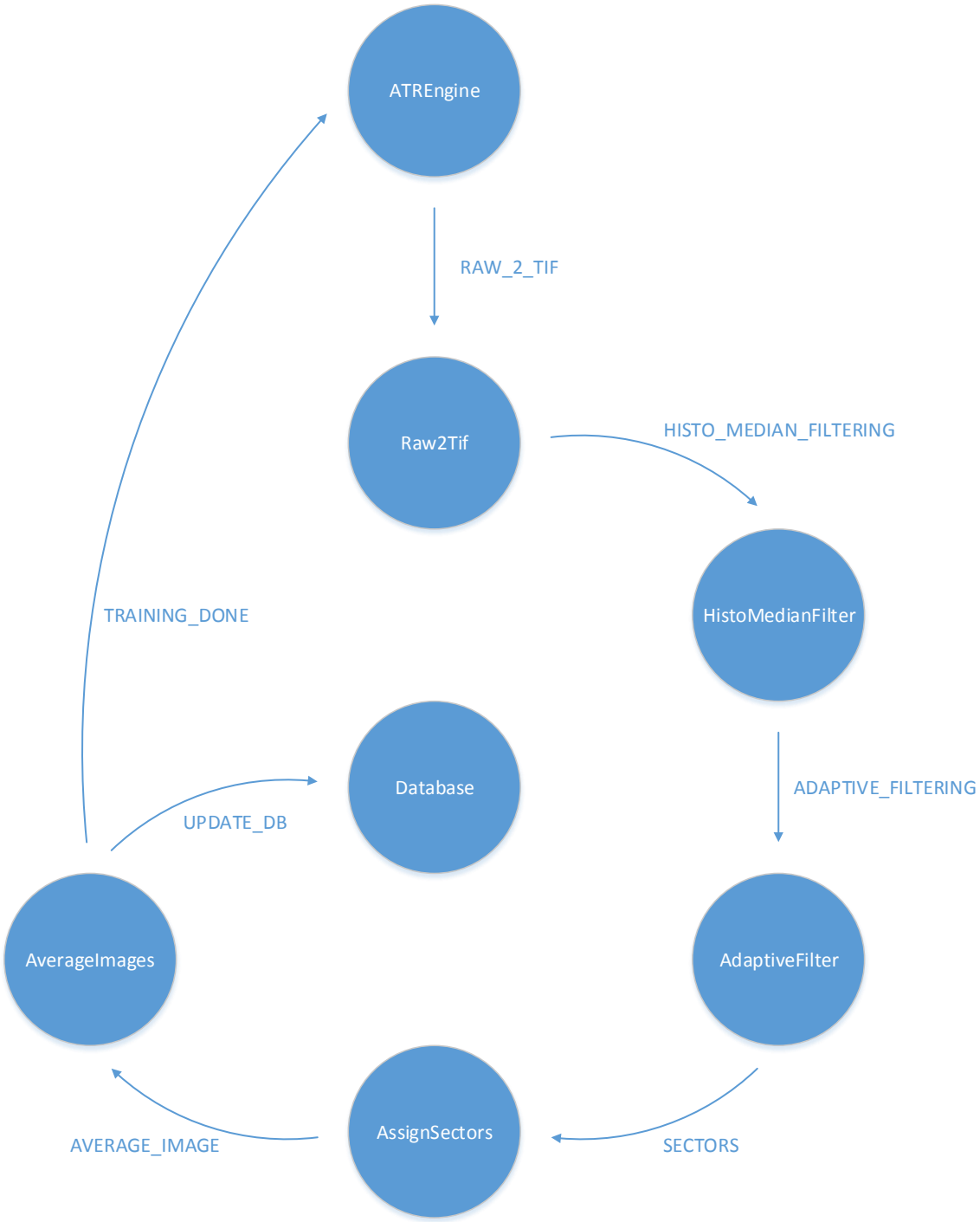


Figure 4 Training Workflow

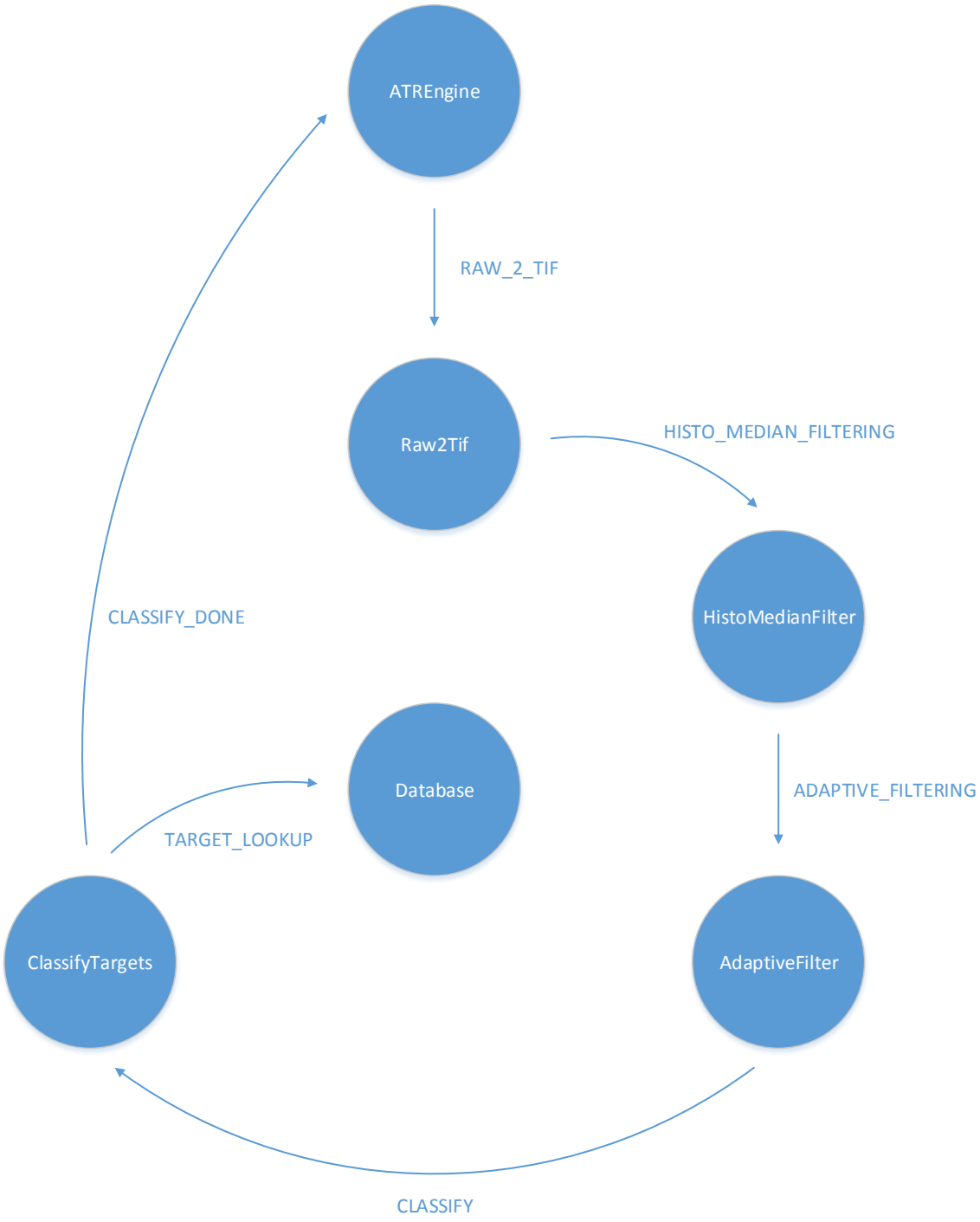


Figure 5 Classification Workflow

3.1 Tasks

The following tasks perform the high level operations that automate the system.

Root – Initialize and start all child tasks and check user inputs from command console.

Raw2Tif – Convert images from RAW to BMP format.

HistoMedianFilter – Create binary images using histogram threshold and median filter.

AdaptiveFilter – Advanced filtering to discard noise regions and segment target regions.

AssignSectors – Separate images into sets according to angle.

AverageImages – Create an averaged image from each angle set.

ClassifyTargets – Convert and filter testing images, then compare them with averaged images stored in database.

Database – Maintain database* containing averaged images and perform target matching.

RestoreFolders – Remove all intermediate files and folders and clean up database to prepare for next cycle.

ATREngine – The main task which automates the ATR system, it creates randomized image data sets from the data bank, executes training and performs classification according a user-defined number of cycles.

* The ATR database is a table maintained by the Database task to store the trained average images using orientation as indices for target classification. Figure 9 shows the database access among tasks during machine learning and target classification.

Raw2Tif, HistoMedianFilter, AdaptiveFilter, AssignSectors, AverageImages, ClassifyTargets and RestoreFolders perform message passing to coordinate transitions between tasks and interface with Root, Database and ATREngine by sending and receiving commands. convert_to_bmp.cpp, threshold_and_medfilter.cpp, adaptive_filter.cpp, sector.cpp, average.cpp, classify.cpp and restore.cpp execute the image processing algorithms. They are called by Raw2Tif, HistoMedianFilter, AdaptiveFilter, AssignSectors, AverageImages, ClassifyTargets and RestoreFolders, respectively. Since ATREngine calls multiple image processing tasks, high level and image processing operations are designed separately to promote modularity.

3.2 Console Commands

The root task starts the training process triggered by the Raw2Tif message with the "=train" command as user input in a command console. As each task completes execution, it passes a message to the next task until all processing for the algorithm has completed (See Figure 4 Training Workflow). Once training has completed, classification of unknown images is performed using the "=classify" command (See Figure 5 Classification Workflow). The "=restore" command then allows the user to reset the system to prepare for classification using a new image set. It deletes all files and folders for each vehicle type except the ones containing the original raw images. A three-input command "=cycle %d sector %d sample %d" generates randomized training and unknown sets from each image set and combines the "=train", "=classify" and "=restore" commands to thoroughly test algorithm accuracy.

CHAPTER 4

Image Processing Algorithms

This project consists of old tasks for image conversion and filtering taken from Daniel Cary's Optimal Rectangular Fit combined with new tasks implementing an orientation-based algorithm. It is important to note that new folders are created at each step of the algorithm to store the resulting images from the current task. A summary of the old tasks convert_to_bmp.cpp, threshold_and_medfilter.cpp and adaptive_filter.cpp from the Optimal Rectangular Fit algorithm is provided as a reference. The new tasks sector.cpp, average.cpp, classify.cpp, restore.cpp and ATREngine.cpp that make up the new algorithm proposed in this thesis are explained in detail.

convert_to_bmp.cpp

This task reads in each line of a text file in the place_raw_here folder as an image name and converts the image from RAW to BMP format. Each pixel of the RAW image is read into a 2-D character array, which then has the rows and columns inverted to display correctly in BMP format. A header with preset values is the first to be added to the BMP image, followed by the inverted image data and zero padding at the end of rows if the number of columns is not divisible by four. The resolution of all vehicle types is normalized to 128 x 128 so that they can be processed by the same algorithm. Images generated by this task are placed in the raw_to_bmp folder along with a text file containing all image names called filename.txt.

threshold_and_medfilter.cpp

This task performs a histogram threshold and median filter operation on each image in the raw_to_bmp folder and converts it into a binary image. Since noise constitutes the lower 95.5 – 98.5% of all pixel intensities, a threshold value is selected such that all intensities below the histogram minimum are set to 0 while the rest are set to 255. This creates a black and white image containing speckle noise, some of which is suppressed using a 3 x 3 median filter. The median filter sorts all pixels in a 3 x 3 grid from lowest to highest intensity and then replaces the center pixel with the median value.

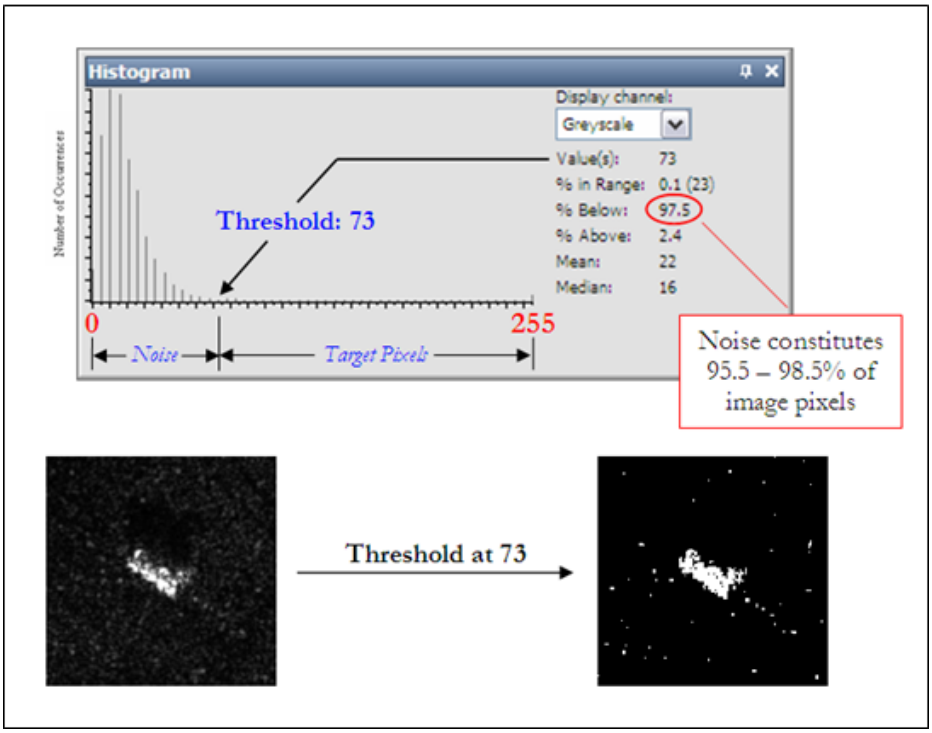


Figure 6 Histogram Threshold[2]

[2] Automatic Target Recognition via Optimal Rectangular Fit, Daniel A. Cary, 2007

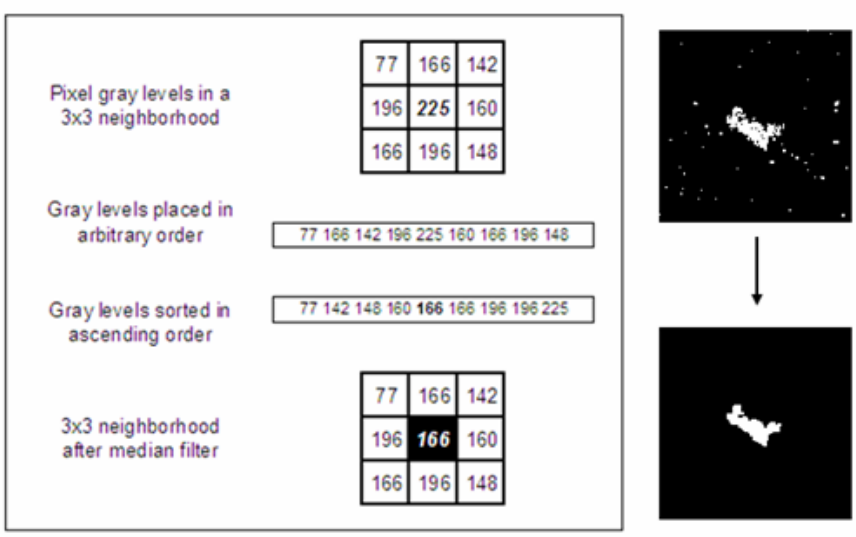


Figure 7 Median Filter[3]

adaptive_filter.cpp

This task performs advanced filtering to discard noise regions and segment target regions in each image. A least squares algorithm is used to determine a line of best fit through all possible target regions in the image, then calculates the maximum normal distance from that line to a pixel in the largest target region. Minor regions with center of mass further from the line than the maximum normal distance are classified as noise and suppressed. The pixel area of each remaining minor region is used as a maximum radius arm and compared to the pixel distance from that region to the main target region. If the pixel distance is larger than the maximum radius arm the minor region is classified as noise and suppressed. Otherwise, it is classified as a part of the target and connected to the main target region using a region growing algorithm. This process repeats until only a single target region remains in the image.

[3] Automatic Target Recognition via Optimal Rectangular Fit, Daniel A. Cary, 2007

sector.cpp

This task separates the images generated from adaptive_filter.cpp into sets according to angle. find_angle, the function that determines target orientation, is adapted from the rectangular fit code for the linear least squares calculation. However, the least squares algorithm is unable to distinguish between angles with a 180° difference. This presents a problem as the head and tail of a vehicle often look different enough that averaging them would reduce the distinguishing features of the vehicle.

The function classify_headtail was developed to distinguish between angles with a 180° difference. It calls the function find_reference to select a reference image from a set by counting the number of target pixels in each image and storing it as an element in a vector of integers. This vector is sorted and the median value is taken as the number of target pixels in the reference image. The image in the set with the number of target pixels matching this median value is thus selected as the reference image. For every image in the set, the reference image is compared to it pixel-by-pixel using an XNOR operation, then compared to a 180° rotation of the same image to determine which has more matching pixels. The image is categorized as the angle calculated by find_angle if it has more matching pixels prior to rotation or as that angle plus 180° if it does not.

average.cpp

This task generates an averaged template image for each set of images grouped by sector.cpp. The average of a set of images of a given target is obtained by adding up all the images and re-thresholding the result. Assume there are "n" binary images of a target (0 is black, 1 is white) with white representing the target. Add up all the images so that the maximum pixel value in the "sum" image will be (n x 1) = n, which corresponds to common target pixels in all n images. Re-threshold the sum image at a threshold level of T = n/2 and set pixels values greater than T to "white" and pixel values less than T to "black". This yields a good "averaged image" of an ensemble of n similar images of a given target. This technique works just as well for targets made up of multiple blobs.

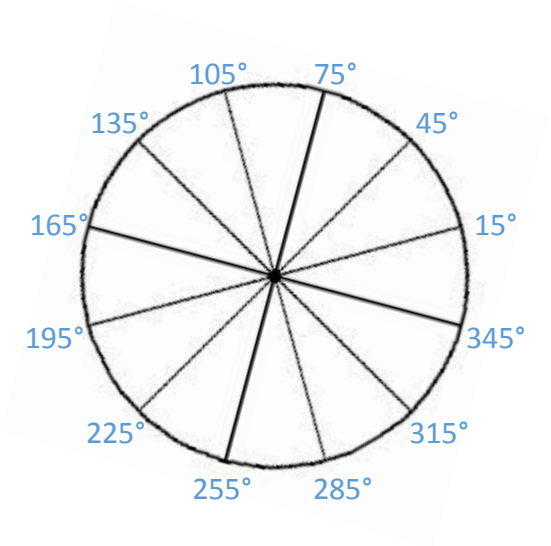


Figure 8 Sector Size of 30°

Sector size determines the number of averaged images generated. For instance, a sector size of 30° creates 12 averaged images according to the orientation of each training set (first set covers 345° to 14°, second set covers 15° to 44°, third set covers 45° to 74°,

etc.) After training is complete, a total of 96 averaged images (8 vehicle types with 12 sets each) are generated. These averaged images are stored in a database and used to perform target matching. The purpose of introducing the database is to accelerate the classification process. Figure 9 shows the interaction of related tasks accessing the database during training and classification.

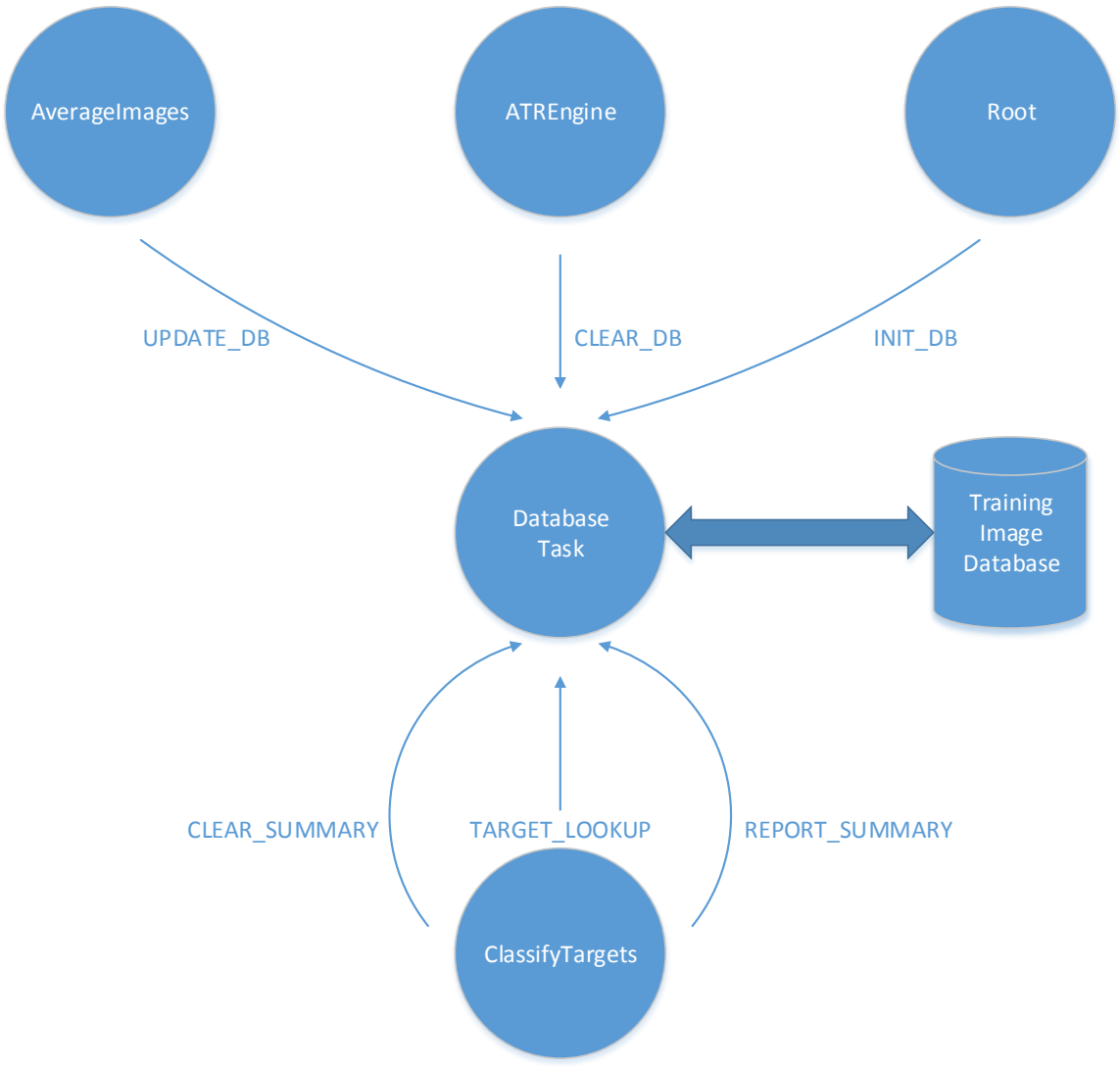


Figure 9 Database Access for Machine Learning and Target Classification

classify.cpp

This task is used to classify images from the testing sets after averaged images have been generated from the training sets. For each "unknown" image in the testing sets, two averaged images with angles 180° apart are selected from one of the eight training sets based on the angle of the unknown image. The unknown image and averaged images are each read into a 128 x 128 character array for comparison. Each pixel of the unknown image is compared to the corresponding pixel of one of the unknown images using an XNOR operation to determine the number of matching pixels between the two images. The number of matching pixels is compared with the highest total of previous comparisons and replaces it if it is greater. This pixel-by-pixel matching and comparison with the highest total is performed for the other averaged image. The process of selecting two averaged images for comparison is then repeated for the seven remaining training sets. A string variable is used to keep track of the vehicle type with the highest total of matching pixels to the unknown image. A correct classification occurs if the vehicle type of the highest matching averaged image is the same as the vehicle type of the unknown image after comparison with averaged images from all eight training sets is complete. When an unknown image is correctly classified, an integer array containing the number of correctly classified images for each testing set has the element corresponding to the vehicle type of the unknown image incremented. After images in all testing sets have been classified, the number of correctly classified images for each vehicle type is displayed in command console.

restore.cpp

This task cleans up all intermediate files and folders generated during training and classification and has the option to keep or delete the original raw images. If called by ATREngine, it deletes all training and unknown files including the original raw images in preparation of generating a new image set from the ImageSource directory. However, the original raw images are preserved by default as a way to quickly evaluate new algorithm functionality using the same dataset. A system command is called to remove all folders in each vehicle type subdirectory of the training directory except for place_raw_here, which contains the original raw data from which all training images are generated. In the unknown directory containing images reserved for testing, the same system command is used to remove all folders in each vehicle type subdirectory except for place_raw_here. The remaining files and folders are the same as before training and classification.

ATREngine.cpp

This task generates randomized image sets for each vehicle type and executes a user-defined number of training and classification cycles. The bin directory contains a subdirectory called ImageSource, the image bank that stores the entire pool of images for each vehicle type. Instead of reserving the same 100 images for testing each time, ATREngine randomly selects a user-specified percentage of ImageSource (sample space), copies them to the unknown subdirectory and then assigns all unselected images to the training subdirectory (See Figure 10 Dataset Generation). Options for sample space are 0.08, 0.17, 0.25 and 0.33, which represent 25, 50, 75 and 100 testing images divided by 300 total images. Percentages are used for sample space since each vehicle type has a different number of total images (BTR-60 has only 256 while other vehicle types have 298 or 299). The actual number of images selected for testing differs slightly from the percentage of total images since a random integer determines whether each image is selected. A user-specified sector size is used to define the averaged template images during training. Options for sector size are 10°, 20° and 30° since these provide high accuracy in the single classification routine. restore.cpp is called to reset the system after each training and classification cycle to prepare for the next cycle.

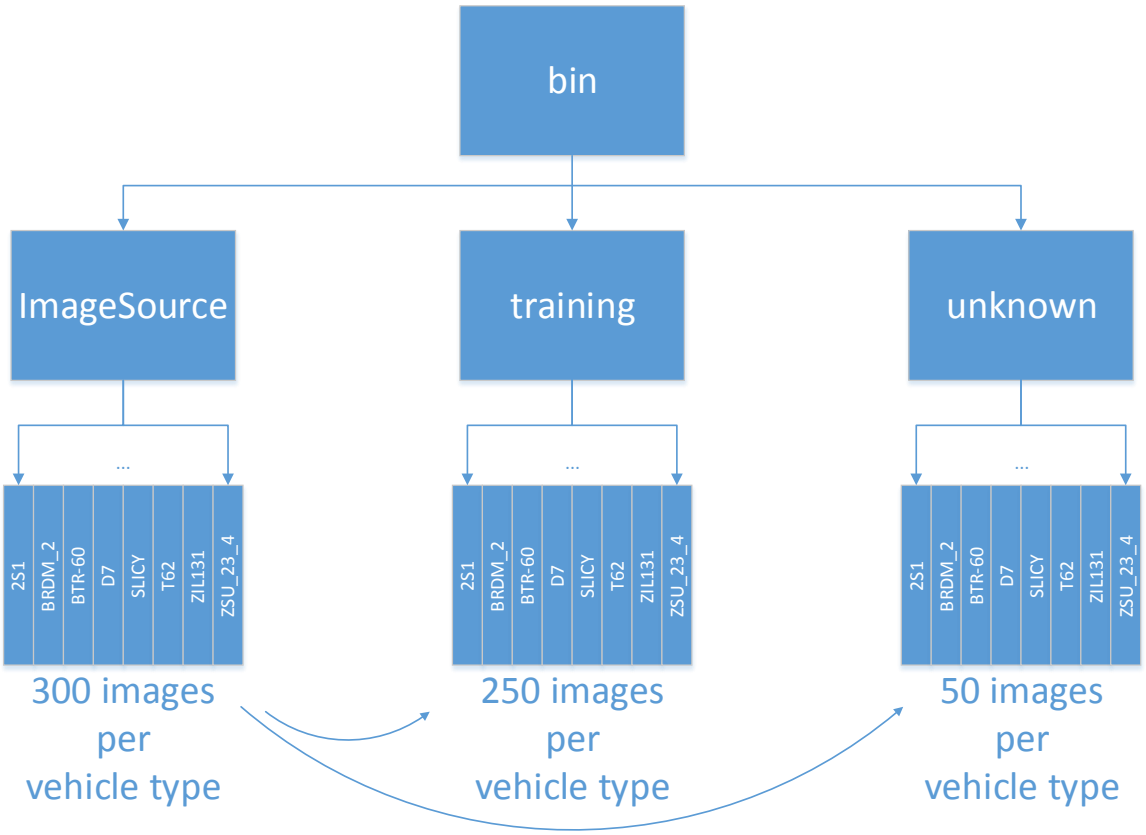


Figure 10 Dataset Generation

The three-input command "=cycle %d sector %d sample %d" that initiates this task provides a way to thoroughly test algorithm performance using various combinations of sector size and sample space. The options for sector size and sample space are stored in arrays and must be accessed using the array index when using the command. Two text files are created in the bin folder and updated during each cycle of ATREngine. The sector size and sample space options along with the classification results of each cycle are stored as a row in ATRSummary.txt. New invocations of the "=cycle" command append the classification result to the end of ATRSummary.txt upon completion. Sector size and sample space options of the last cycle are saved in ATRCfg.txt before the program exits.

CHAPTER 5

Design and Development

The first full-scale testing routine for the orientation-based algorithm used a hardcoded sector size of 30°. This routine along with all subsequent versions are run on a 64-bit Windows 7 system. A total of twelve template images are created for each vehicle type. Each unknown image is compared to all template images of each vehicle type using the XNOR operation (96 total comparisons) before being classified as one of the eight vehicle types. The initial testing run correctly classified 588 of the 800 unknown images, an accuracy of 73.5%. However, the processing time of twenty minutes was prohibitive for rapid algorithm development.

The second version uses the angle of each unknown image to determine which averaged template images to compare. This method was expected to reduce classification time to one-sixth of the first routine, as only two of the twelve template images would be used for comparison with each unknown image. One of the two template images used for comparison would be the same angle as the unknown image while the other would be that angle plus 180°. This version correctly classified 575 of the 800 unknown images, an accuracy of 71.9%. However, the processing time of six minutes represented a significant improvement in exchange for slightly less accuracy.

The third version replaced the hardcoded sector size of 30° with a variable sector size defined by a macro in Config.h. Sector sizes ranging from 1° to 60° were selected to find trends in classification accuracy. Results indicate 10° as the optimum sector size and thus it is used for subsequent algorithm testing. Sectors of 10°, 20° and 30° all yield high accuracy and are later included in the sector size option of the "=cycle" command for randomized training and classification.

Table 1 Results of Original Dataset using Standard Sector Sizes

Sector Size		10°		20°		30°	
Vehicle Type	Total Images	Classified Images	Accuracy	Classified Images	Accuracy	Classified Images	Accuracy
2S1	100	74	74.00%	65	65.00%	69	69.00%
BRDM_2	100	88	88.00%	80	80.00%	73	73.00%
BTR-60	100	64	64.00%	72	72.00%	73	73.00%
D7	100	92	92.00%	93	93.00%	97	97.00%
SLICY	100	99	99.00%	98	98.00%	94	94.00%
T62	100	67	67.00%	65	65.00%	54	54.00%
ZIL131	100	68	68.00%	53	53.00%	47	47.00%
ZSU_23_4	100	75	75.00%	69	69.00%	63	63.00%
Overall Accuracy			78.38%		74.38%		71.25%

Table 2 Results of Original Dataset using Customized Sector Sizes

Sector Size		1°		5°		60°	
Vehicle Type	Total Images	Classified Images	Accuracy	Classified Images	Accuracy	Classified Images	Accuracy
2S1	100	53	53.00%	64	64.00%	65	65.00%
BRDM_2	100	43	43.00%	88	88.00%	58	58.00%
BTR-60	100	37	37.00%	60	60.00%	49	49.00%
D7	100	61	61.00%	92	92.00%	69	69.00%
SLICY	100	68	68.00%	92	92.00%	97	97.00%
T62	100	45	45.00%	63	63.00%	46	46.00%
ZIL131	100	42	42.00%	70	70.00%	25	25.00%
ZSU_23_4	100	39	39.00%	69	69.00%	54	54.00%
Overall Accuracy			48.50%		74.75%		57.88%

The fourth version uses a database of structures to store the angle and 2-D character array information (representing pixel values) for each averaged template image generated from training. The purpose of this database is to decrease classification time by using angle as a key to rapidly access the appropriate template image arrays for comparison instead of reading the template images into arrays each time a new unknown image is to be classified. Accuracy remains unchanged from the previous version since the training and classification algorithms are the same. This database reduces classification time for the 800 unknown images to two minutes. Specifically, the preprocessing time using convert_to_bmp.cpp, threshold_and_medfilter.cpp and adaptive_filter.cpp to convert and filter the unknown images is one minute and thirty seconds, while the actual classification time of classify.cpp is thirty seconds.

The fifth version adds a task to clean up the intermediate files and folders generated during training and classification. The "=restore" command deletes all files and folders for each vehicle type except the place_raw_here folder containing the original raw images. This provides a convenient way for the user to reset the system to prepare for classification using a new training or testing set. The time to clean up all training and unknown images is ten seconds.

The final version includes the three-input command "=cycle %d sector %d sample %d" for ATREngine to generate randomized training and unknown sets from the entire pool of images for each vehicle type. A discrepancy in classification accuracy when using the first and last 100 images of the original dataset for testing motivated the development of this functionality. Sample space was implemented as a parameter in ATREngine to determine if the smaller training set of BTR-60 negatively impacted its classification accuracy. The ability to automatically generate multiple data sets is a major asset as it allows for more rigorous testing of algorithm reliability.

Table 3 ATREngine Options

Index	Sector Size	Sample Space
0	10°	0.08
1	20°	0.17
2	30°	0.25
3	N/A	0.33
-1	random	random

CHAPTER 6

Results

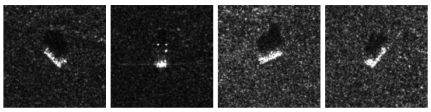
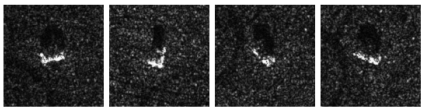
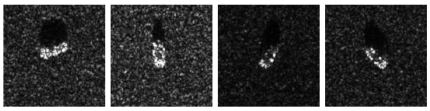
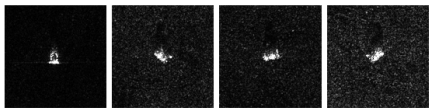
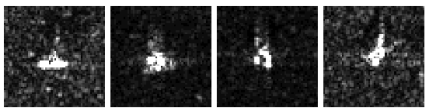
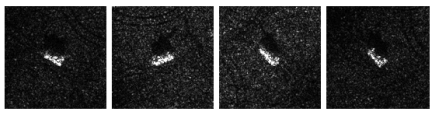
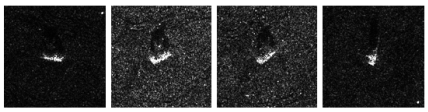
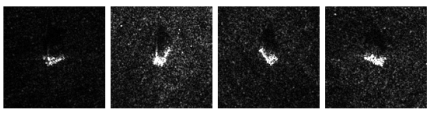
This project uses SAR images collected at a 17° depression angle for training and classification. The image set for this depression angle is selected because it provides nearly 300 images for each vehicle type. Other depression angles in the MSTAR database have an uneven distribution of images for each vehicle type and are therefore unsuitable for use in algorithm testing. The algorithm normalizes the resolution of all vehicle types to 128 x 128 when converting images from RAW to BMP format. SLICY consistently yields higher classification accuracy compared to other vehicle types due to its smaller size after normalization.

Table 4 Comparison of ATREngine with Previous Algorithms[4]

	Classification Accuracy		
Vehicle Type	Hausdorff Probabilistic Feature Analysis	Optimal Rectangular Fit	ATREngine
2S1	100%	36%	74%
BRDM_2	80%	49%	88%
BTR-60	70%	32%	64%
D7	70%	44%	92%
SLICY	100%	87%	99%
T62	100%	37%	67%
ZIL131	80%	36%	68%
ZSU_23_4	80%	34%	75%
Overall	85%	44%	78%

[4] All algorithms evaluated using the same dataset (ATREngine sector size 10°)

Table 5 MSTAR Images at 17° Depression Angle[5]

Target	Pixel Resolution	Example Images
2S1	158 x 158	
BRDM_2	128 x 129	
BTR-60	128 x 128	
D7	177 x 178	
SLICY	54 x 54	
T62	172 x 173	
ZIL131	192 x 193	
ZSU_23_4	158 x 158	

[5] Automatic Target Recognition via Optimal Rectangular Fit, Daniel A. Cary, 2007

Prior to the development of ATREngine, the image set of each vehicle type was divided such that 100 images were selected for testing while the rest were used for training. Since the Nyquist criterion states that sampling rate must be at least twice the signal frequency, the smaller training set of BTR-60 may be the reason it ranks among the lowest accuracies of all vehicle types in the original dataset. A proportional assignment of 176 training images and 80 unknown images appears to validate this claim as it produced an average classification accuracy of 75% for BTR-60 (See Figure 11 Trends of 1300 Datasets). The original dataset used in this project is the same as the one in the Optimal Rectangular Fit as a baseline for comparison between the two algorithms.

Table 6 XNOR vs. AND Operation

A	B	AND	XNOR
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	1

The algorithm in sectors.cpp and classify.cpp originally used a logical AND for pixel-by-pixel comparison of two images. Initial testing revealed higher match rates for the larger vehicle types D7, T62 and ZSU_23_4. These larger vehicles had more target pixels and produced more matches than smaller vehicles when using AND for comparison with an unknown vehicle type. Larger vehicles almost always have more overlapping target pixels than smaller vehicles and thus unknown images that were actually of the smaller type would more often be incorrectly classified as the larger type.

The solution to this problem was to use the XNOR operation instead of AND to match both target and background pixels. Recalling that target pixels have a value of 1 and background pixels have a value of 0, the inclusion of background pixels in the total number of matching pixels eliminates this overlapping effect and results in a more even distribution of vehicle types during classification.

Algorithm performance is evaluated using a trial of 1300 ATREngine cycles to generate approximately 100 data points for each combination of sector size and sample space. Since the trial executed the command "=cycle 1300 sector -1 sample -1" each cycle used a random combination of sector size and sample space and thus not every combination has exactly 100 data points. The processing time of each cycle including sample creation, training and classification takes between five to eight minutes. An average time of six minutes per cycle resulted in the trial completing after five days of continuous processing on a 2.20GHz CPU (See Appendix A System Specifications). The automation process involved intensive disk I/O resulting from many image read/write operations and was not fully optimized due to time constraints. Algorithm accuracy for the 1300 cycles spanning all combinations and vehicle types is 76.78%.

Table 7 Results of 1300 Datasets

	Sample Space	Sector Size	2S1	BRDM_2	BTR-60	D7	SLICY	T62	ZIL131	ZSU_23_4
Average00	0.08	10°	68.85%	83.18%	81.29%	90.36%	98.61%	66.57%	75.91%	77.50%
Average01	0.08	20°	62.19%	87.13%	73.63%	90.86%	95.96%	67.81%	68.53%	76.46%
Average02	0.08	30°	62.67%	82.79%	67.60%	92.45%	96.47%	58.03%	63.67%	75.15%
Average10	0.17	10°	68.81%	83.53%	78.29%	90.97%	97.91%	64.84%	76.05%	76.13%
Average11	0.17	20°	61.64%	85.04%	74.79%	91.54%	95.62%	67.92%	69.19%	76.72%
Average12	0.17	30°	61.73%	82.28%	67.44%	91.84%	96.14%	58.03%	63.19%	75.58%
Average20	0.25	10°	65.87%	81.31%	77.30%	91.33%	98.00%	62.90%	75.19%	76.75%
Average21	0.25	20°	60.38%	84.78%	73.50%	90.66%	95.93%	67.81%	68.33%	75.58%
Average22	0.25	30°	61.55%	80.63%	65.86%	91.87%	95.98%	56.79%	61.02%	75.78%
Average30	0.33	10°	67.49%	81.90%	75.00%	90.79%	97.40%	63.02%	71.69%	74.72%
Average31	0.33	20°	61.79%	84.11%	71.96%	90.80%	95.16%	65.18%	66.17%	74.59%
Average32	0.33	30°	61.75%	79.48%	66.13%	91.70%	95.42%	56.59%	61.17%	73.17%

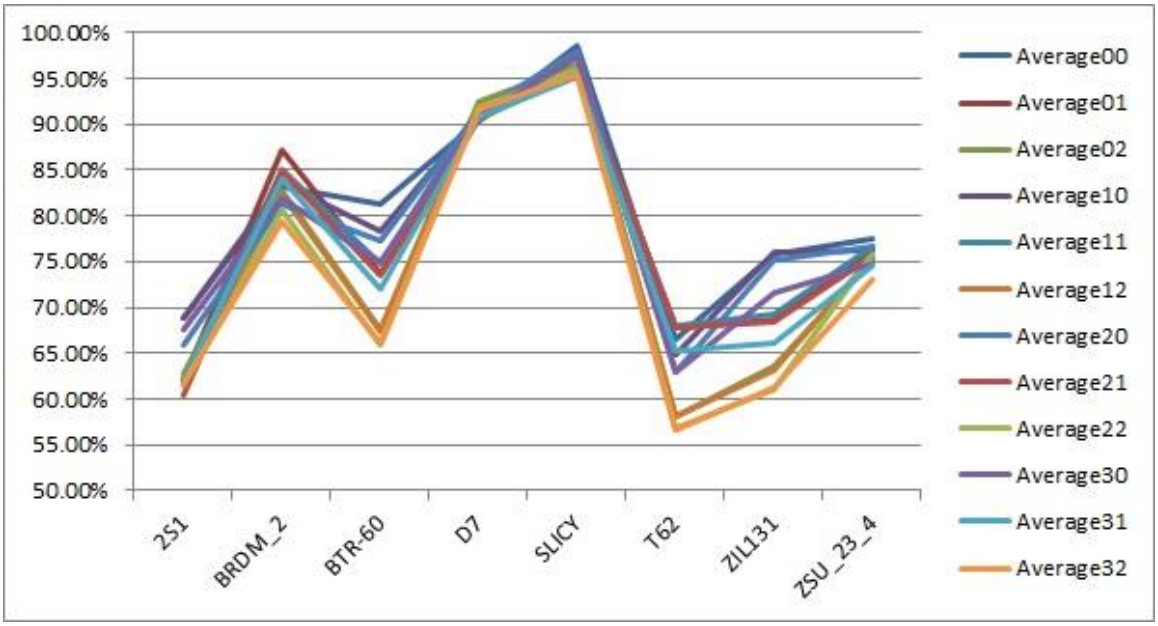


Figure 11 Trends of 1300 Datasets

CHAPTER 7

Future Work

Classification accuracy and processing speed are areas that can be improved upon by future students. Although this project offers reliable speed and accuracy, it can be enhanced through the optimization of its processes. The flexibility of the task-based automated system provides a way to rapidly prototype and test new methods. Combining previous algorithms as tasks in this system may yield new possibilities for Automatic Target Recognition as well.

Chessa Guilas' Hausdorff Probabilistic Feature Analysis Approach presents an accuracy of 85% compared to the 76.78% accuracy of this orientation-based algorithm. One possible project would be to convert Guilas' edges, corners and peaks methods into tasks and include them in this project to create a more accurate version of the automated system. Combining the edges, corners and peaks methods with the orientation-based approach could potentially increase accuracy to above 85%. In addition, testing Guilas' algorithm across multiple datasets generated by ATREngine should give a more accurate assessment of its feasibility for ATR.

Increasing processing speed is important since the intensive read/write operations of ATREngine slows down other applications running on the operating system. A more efficient design minimizing image read/write operations would be a viable extension to this project. Since the algorithm is not optimized, performance can also be improved by introducing better pipelining methods. As this thesis is entirely based in software, another idea is to implement the image processing algorithm in hardware such as a DSP or FPGA to take advantage of parallel processing. A hardware implementation of this algorithm can also provide the real-time functionality critical to military applications.

BIBLIOGRAPHY

[1] Daniel A. Cary; Automatic Target Recognition via Optimal Rectangular Fit; Senior Project, EE Department, Cal Poly, SLO, June 2007

[2] Chessa F. Guilas; Hausdorff Probabilistic Feature Analysis in SAR Image Recognition; Thesis, EE Department, Cal Poly, SLO, December 2004

[3] Andrew DeKelaita; Automatic Target Recognition via Shadow Exploitation; Senior Project, EE Department, Cal Poly, SLO, 2006

[4] "Problem with Image Rotation C++", Internet: http://www.codeproject.com/Questions/481930/problempluswithplusimageplusrotationplusc-2b-2b, October 2012 [May 14, 2013]

[5] Stimson, George W. *Introduction to Airborne Radar 2nd ed*. Raleigh: SciTech Publishing, 1998

[6] Cumming, Ian G. and Frank H. Wong. *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation*. Norwood: Artech House, 2005

[7] Ulaby, Fawwaz T., Richard K. Moore and Adrian K. Fung. *Microwave Remote Sensing: Active and Passive, Volume II: Radar Remote Sensing and Surface Scattering and Emission Theory*. Norwood: Artech House, 1982

[8] J. Zhang, W. Zhao, G. Cheng and S. Mao, "Compounding Segmentation Method for SAR Images," *8th International Conference on Signal Processing, IEEE*, Nov. 2006, pp. 1-4

[9] J. A. Ratches, C. P. Walters and R. G. Buser, "Aided and automatic target recognition based upon sensory inputs from image forming systems," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 1004–1019, Sept. 1997

[10] Y. Yang, Y. Qiu and C. Lu, "Automatic Target Classification Experiments on the MSTAR SAR Images," *Proceedings of the Sixth International Conference on SNPD/SAWN'05*, 2005, pp. 2-7

[11] J. Cui, J. Gudnason and M. Brookes, "Radar Shadow and Superresolution Features for Automatic Recognition of MSTAR targets," in *Proc. IEEE Int. Radar Conference*, May 2005, pp. 534-539

[12] Y. Sun, Z. Liu, S. Todorovic and J. Li, "Adaptive boosting for SAR automatic target recognition," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 43, no. 1, pp. 112–125, Jan. 2007

[13] P. Han, Z. Han and R. Wu, "A SVR-Based SAR Target Azimuth Fusion Estimation," in *Conference on Synthetic Aperture Radar, 2009. APSAR 2009. 2nd Asian-Pacific*, pp. 169-172, IEEE, 2009

[14] J. Pisane, R. Marée, L. Wehenkel, and J. Verly, "Robust Automatic Target Recognition using Extra-Trees," in *Proceedings of the IEEE International Radar Conference*, pp. 231, 2010

[15] J. I. Park, S. H. Park and K. T. Kim, "New Discrimination Features for SAR Automatic Target Recognition," *IEEE Geoscience and Remote Sensing Letters*, Vol. 10, No. 3, 2013, pp. 476-480

APPENDIX A

System Specifications

Windows Edition

Windows 7 Home Premium

Service Pack 1

System

Model:	Lenovo Win7 PC
Processor:	Intel Core i7-2670QM CPU @ 2.20GHz
Installed Memory (RAM):	8.00 GB
System Type:	64-bit Operating System
Resolution:	1366 x 768
Hard Disk:	654 GB
Free Space:	405 GB

Software

Microsoft Visual Studio 2010

MATLAB 2012a

APPENDIX B

Source Code

Root.cpp

```
#include <stdlib.h>
#include <iostream>

#include "Build.h"
#include "Raw2Tif.h"
#include "HistoMedianFilter.h"
#include "AdaptiveFilter.h"
#include "AssignSectors.h"
#include "AverageImages.h"
#include "Database.h"
#include "ClassifyTargets.h"
#include "RestoreFolders.h"
#include "ATREngine.h"
#include "MyMsg.h"
#include "Config.h"

// Since some of the code is shared across Windows and QNX.
// Disable this warning message.

#pragma warning(disable : 4996)
using namespace std;


//// Prototypes //////////////////////////////////////////////////

extern void dbgprint(const char* fmt, ...);

//// Passed in global setting ////////////////////////////////////

int g_nDbgEnable = 1;
int g_nSectSizeIdx = DEFAULT_SECTOR_SIZE_IDX;
int g_nSmplPctIdx = DEFAULT_SAMPLE_SIZE_IDX;
int g_nSectorInDegs = DEFAULT_SECTOR_IN_DEGS;
int       g_nSectorNum = 360 / g_nSectorInDegs;     // 360 degrees per circle.  So, we
have (360 / SectorInDegs) sectors.
```

```
float g_fSampleSpcRate = (float)DEFAULT_SAMPLE_SIZE / (float)
TOTAL_IMAGES_PER_TYPE;

//// Constants //////////////////////////////////////////////////
const int g_nSectorSize[SECTOR_TABLE_SIZE] = {10, 20, 30};
        // SECTOR_SIZE_RANDOM_PICK (-1): Random pick anyone of them
const int g_nSampleSpacePct[SAMPLE_SPACE_TBL_SIZE] = {25, 50, 75, 100};
        // SAMPLE_SPACE_RANDOM_PICK (-1): Random pick anyone of them

// Kludge:  Actual image count of each data type from the ImageSource directory.  Refer
to filename.txt for each type.
const int g_nTotalImagesPerType[NUM_OF_CAR_TYPES] = { 299, 298, 256, 299, 298,
299, 299, 299 };

const char* g_szVehicleName[NUM_OF_CAR_TYPES] =
{
        "2S1      ",
        "BRDM_2    ",
        "BTR-60    ",
        "D7       ",
        "SLICY     ",
        "T62      ",
        "ZIL131    ",
        "ZSU_23_4  "
};

void Crash(MSGID nCode)
{
        static BOOL bCalled=FALSE;
        int                     nIdx = 0;
        if (bCalled)
                return;                                 // No need to call Crash() twice.

        bCalled = TRUE;                                 // Crash() has been called.
        dbgprint("Crash():  gets called...\n");

        Raw2Tif::Stop();
        HistoMedianFilter::Stop();
        AdaptiveFilter::Stop();
        AssignSectors::Stop();
        AverageImages::Stop();
        Database::Stop();
        ClassifyTargets::Stop();
        RestoreFolders::Stop();
        ATREngine::Stop();
```

```
	Sleep(500);	// Wait for 0.5 seconds.

	// Deallocate the resources.
	Raw2Tif::Delete();
	HistoMedianFilter::Delete();
	AdaptiveFilter::Delete();
	AssignSectors::Delete();
	AverageImages::Delete();
	Database::Delete();
	ClassifyTargets::Delete();
	RestoreFolders::Delete();
	ATREngine::Delete();

	// Delete more application resources.

	printf("Crash exiting...\n");
}


BOOL DoLocalCmd(char* tzCommand)
{
	// Either quit or do system command.

	char*		pCmd = tzCommand;
	MY_MSG*	pMsg = NULL;
	char*		szCmd = NULL;

	if (!strncmp(pCmd, "=", 1))
	{
		int nBufferSize = (int)strlen(pCmd)+1;		// Add a null char at the end.
		char* szActualCMD = new char[nBufferSize];
		memset(szActualCMD, 0, nBufferSize);
		memmove(szActualCMD, tzCommand+1, nBufferSize);	// Skip "=".

		if (!strnicmp(szActualCMD, "raw2tif", strlen("raw2tif")))
		{
			char filename[80];
			sprintf(filename, szActualCMD+strlen("raw2tif")+1);
			pMsg = new MY_MSG(Q_RAW_2_TIF, filename, nBufferSize);
			delete [] szActualCMD;

			if (Raw2Tif::Send(Q_RAW_2_TIF, pMsg) == false)
			{
```

```
                                dbgprint("crash file is %s, line is %d.\n", __FILE__,
__LINE__);
                                delete pMsg;
                                Crash(Q_EXIT_FROM_ERROR);
                        }

                        return TRUE;
                }
                else if (!strnicmp(szActualCMD, "train", strlen("train")))
                {
                        pMsg = new MY_MSG(Q_RAW_2_TIF);
                        delete [] szActualCMD;

                        if (Raw2Tif::Send(Q_RAW_2_TIF, pMsg) == false)
                        {
                                dbgprint("crash file is %s, line is %d.\n", __FILE__,
__LINE__);
                                delete pMsg;
                                Crash(Q_EXIT_FROM_ERROR);
                        }

                        return TRUE;
                }
                else if (!strnicmp(szActualCMD, "classify", strlen("classify")))
                {
                        char filename[80];
                        sprintf(filename, szActualCMD+strlen("classify")+1);
                        pMsg = new MY_MSG(Q_CLASSIFY, filename, nBufferSize);
                        delete [] szActualCMD;          // Hand it over.  TaskPing will delete
this piece of buffer.

                        if (ClassifyTargets::Send(Q_CLASSIFY, pMsg) == false)
                        {
                                dbgprint("crash file is %s, line is %d.\n", __FILE__,
__LINE__);
                                delete pMsg;
                                Crash(Q_EXIT_FROM_ERROR);
                        }

                        return TRUE;
                }
                else if (!strnicmp(szActualCMD, "restore", strlen("restore")))
                {
                        pMsg = new MY_MSG(Q_RESTORE);
```

```
                        delete [] szActualCMD;          // Hand it over.  TaskPing will delete
this piece of buffer.

                        if (ATREngine::Send(Q_RESTORE, pMsg) == false)
                        {
                                dbgprint("crash file is %s, line is %d.\n", __FILE__,
__LINE__);
                                delete pMsg;
                                Crash(Q_EXIT_FROM_ERROR);
                        }

                        return TRUE;
                }
                else if (!strnicmp(szActualCMD, "sector", strlen("sector")))
                {
                        //SectorSzIdx and SmplSpcIdx are indices to actual sector size and
sample space percentage.
                        // e.g. Sector size indices 0, 1, and 2 represents sector size in 10,
20, 30 degrees.  -1 means randomizes for each cycle.
                        //                  Sample space percentage indices 0, 1, 2, and 3
represents 25, 59, 75 or 100 images out of 300 images from the image source.
                        int nSectSzIdx = g_nSectSizeIdx;
                        char SectorSize[20];            // Use as a separator.

                        // The command is something like:  SectorSize %d
                        sscanf( szActualCMD, "%s %d", SectorSize, &nSectSzIdx);

                        if ((nSectSzIdx >= 0) && (nSectSzIdx <
SECTOR_TABLE_SIZE))
                        {
                                g_nSectSizeIdx = nSectSzIdx;
                                g_nSectorInDegs = g_nSectorSize[g_nSectSizeIdx];
                                g_nSectorNum = 360 / g_nSectorInDegs;
                                dbgprint("Change Sector size index %d, sector size %d
sector num %d\n", g_nSectSizeIdx, g_nSectorInDegs, g_nSectorNum);
                        }
                        else
                        {
                                dbgprint("Ignore setting invalid sector size index %d\n",
nSectSzIdx);
                                dbgprint("Current Sector size index %d, sector size %d
sector num %d\n", g_nSectSizeIdx, g_nSectorInDegs, g_nSectorNum);
                        }
```

```
                        delete [] szActualCMD;        // Hand it over.  TaskPing will delete
this piece of buffer.
                        return TRUE;
                }
                else if (!strnicmp(szActualCMD, "cycle", strlen("cycle")))
                {
                        //SectorSzIdx and SmplSpcIdx are indices to actual sector size and
sample space percentage.
                        // e.g. Sector size indices 0, 1, and 2 represents sector size in 10,
20, 30 degrees.  -1 means randomizes for each cycle.
                        //                  Sample space percentage indices 0, 1, 2, and 3
represents 25, 59, 75 or 100 images out of 300 images from the image source.
                        int nCycle = -1, nSectSzIdx = g_nSectSizeIdx, nSmplSpcIdx =
g_nSmplPctIdx;
                        char Cycle[20], SectorSize[20], SampleSpace[20];              // Use
them as separators.

                        // The command is something like:  Cycles %d SectorSize %d
SampleSpace %d.
                        sscanf( szActualCMD, "%s %d %s %d %s %d", Cycle, &nCycle,
SectorSize, &nSectSzIdx, SampleSpace, &nSmplSpcIdx );

                        delete [] szActualCMD;        // Hand it over.  TaskPing will delete
this piece of buffer.

                        CycleInfo *pCycleInfo = new CycleInfo;
                        pCycleInfo->m_nCycle = nCycle;
                        pCycleInfo->m_nSectSizeIdx = nSectSzIdx;
                        pCycleInfo->m_nSmplPctIdx = nSmplSpcIdx;

                        pMsg = new MY_MSG(Q_UPDATE_CYCLE, pCycleInfo);

                        if (ATREngine::Send(Q_UPDATE_CYCLE, pMsg) == false)
                        {
                                dbgprint("Q_UPDATE_CYCLE failed, file is %s, line is
%d.\n", __FILE__, __LINE__);
                                delete pCycleInfo;
                                delete pMsg;
                                Crash(Q_EXIT_FROM_ERROR);
                        }

                        return TRUE;
                }
                else
                {
```

```
                        pMsg = new MY_MSG(Q_DO_TEST_CMD, szActualCMD,
nBufferSize);
                        delete [] szActualCMD;        // Hand it over.  TaskPing will delete
this piece of buffer.

                        return TRUE;
                }
        }

        if (!strcmp(pCmd, "bye"))
                return FALSE;

        if (!strcmp(pCmd, "exit"))
                return FALSE;

        if (!strcmp(pCmd, "quit"))
                return FALSE;

        if (!strcmp(pCmd, "ver"))
        {
                printf("%s\n", ATR_BLD_INFO);
                return TRUE;
        }

        if (pCmd[0])
                        system(pCmd);

        return TRUE;
}

int main(int argc, char* argv[], char* envp[])
{
        char progname[MAX_BUF_SZ];
        char cCommand[MAX_BUF_SZ];
        int       nIdx = 0;
        DWORD dwTickCnts = 0;
        DWORD dwTime = 0;
        DWORD dwResult = 0;
        FILE *hFile = NULL;
        char KeyName_1[20], KeyName_2[20], KeyName_3[20], KeyName_4[20],
Separator_1[20], Separator_2[20], Separator_3[20], Separator_4[20];                // Use
this as a separator to skip the word "SectorSize".

        strcpy(progname, "ATR");
```

```
    // Do a little sanity checking.

        switch(argc)
        {
                case 1:
                        /* Use default */
                        break;

                case 2:
                        printf("Set dbg flag %s\n", argv[1]);
                        g_nDbgEnable = argv[1][0] - '0';       // Assume passed in a char.
                        break;

                default:
                        printf("usage: %s Dbg flag.\n"
                                        "example: %s 0\n", progname, progname);
                        exit(EXIT_FAILURE);
                        break;
        }

        hFile = fopen ("./ATRCfg.txt","r");
        if (hFile != NULL)
        {
                fscanf (hFile, "%s %s %d %s %s %d %s %s %f %s %s %d", KeyName_1,
Separator_1, &g_nSmplPctIdx, KeyName_2, Separator_2, &g_nSectSizeIdx,
KeyName_3, Separator_3, &g_fSampleSpcRate, KeyName_4, Separator_4,
&g_nSectorInDegs);
                dbgprint("SampleSpaceIdx = %d SectorSizeIdx = %d SampleSpcRate =
%04.2f SectorInDegs = %d\n", g_nSmplPctIdx, g_nSectSizeIdx, g_fSampleSpcRate,
g_nSectorInDegs);
                fclose (hFile);
        }
        else
        {
                g_nSectSizeIdx = DEFAULT_SECTOR_SIZE_IDX;
                g_nSmplPctIdx = DEFAULT_SAMPLE_SIZE_IDX;
                g_nSectorInDegs = DEFAULT_SECTOR_IN_DEGS;
                dbgprint("ATRCfg.txt does not exist.  Use default SmplPctIdx %d
SectorSizeIdx %d SampleSpcRate %04.2f SectorInDegs %d\n", g_nSmplPctIdx,
g_nSectSizeIdx, g_fSampleSpcRate, g_nSectorInDegs);
        }

        g_nSectorNum = 360 / g_nSectorInDegs;

        // Seed the random-number generator with the current time so that
```

```
        // the numbers will be different every time we run.

        // Windows always return the same random number sequence in following
statement.
        //      srand( (unsigned)time( NULL ) );
        // Use Exclusive-OR with Windows GetTickCount to do the trick.

        dwTickCnts = GetTickCount();
        dwTime = (DWORD) time(NULL);
        dwResult = dwTickCnts ^ dwTime;
        srand (dwResult);

        Raw2Tif::Init();
        Raw2Tif::Priority(g_nTestPrior);
        Raw2Tif::Start();

        HistoMedianFilter::Init();
        HistoMedianFilter::Priority(g_nTestPrior);
        HistoMedianFilter::Start();

        AdaptiveFilter::Init();
        AdaptiveFilter::Priority(g_nTestPrior);
        AdaptiveFilter::Start();

        AssignSectors::Init();
        AssignSectors::Priority(g_nTestPrior);
        AssignSectors::Start();

        AverageImages::Init();
        AverageImages::Priority(g_nTestPrior);
        AverageImages::Start();

        Database::Init();
        Database::Priority(g_nTestPrior);
        Database::Start();

        ClassifyTargets::Init();
        ClassifyTargets::Priority(g_nTestPrior);
        ClassifyTargets::Start();

        RestoreFolders::Init();
        RestoreFolders::Priority(g_nTestPrior);
        RestoreFolders::Start();

        ATREngine::Init();
```

```
        ATREngine::Priority(g_nTestPrior);
        ATREngine::Start();

        while (gets(cCommand))
        {
                if (!DoLocalCmd(cCommand))
                {
                        dbgprint("ATR:  Root task exiting...");
                        break;  // Time to exit.
                }
        }

        Crash(Q_EXIT_BY_ROOT);

        hFile = fopen ("./ATRCfg.txt","w");

        if (hFile != NULL)
        {
                fprintf (hFile, "SampleSpaceIdx = %d SectorSizeIdx = %d
SampleSpcRate = %04.2f SectorInDegs = %d", g_nSmplPctIdx, g_nSectSizeIdx,
g_fSampleSpcRate, g_nSectorInDegs);
                dbgprint("Update configuration:  SampleSpaceIdx = %d SectorSizeIdx =
%d SampleSpcRate = %04.2f SectorInDegs = %d\n", g_nSmplPctIdx, g_nSectSizeIdx,
g_fSampleSpcRate, g_nSectorInDegs);
                fclose (hFile);
        }
        else
        {
                g_nSectSizeIdx = DEFAULT_SECTOR_SIZE_IDX;
                g_nSmplPctIdx = DEFAULT_SAMPLE_SIZE_IDX;
                g_nSectorInDegs = DEFAULT_SECTOR_IN_DEGS;
                g_fSampleSpcRate = (float)DEFAULT_SAMPLE_SIZE / (float)
TOTAL_IMAGES_PER_TYPE;
                dbgprint("ATRCfg.txt failed to open.  Use default SmplPctIdx %d
SectorSizeIdx %d SampleSpcRate %04.2f SectorInDegs %d\n", g_nSmplPctIdx,
g_nSectSizeIdx, g_fSampleSpcRate, g_nSectorInDegs);
        }

   return 0;
}
```

Database.cpp

```
#include "JKQueue.h"
#include "JKTask.h"
#include "Database.h"
#include "ClassifyTargets.h"

#include "Config.h"
#include "MyMsg.h"

#include<fstream>
#include<iostream>
#include<sstream>

//situation-dependent constants
#define NUM_GRAY_LEVELS 256

//constants that should not be changed
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024
#define PI 3.14159265359

//global variables
static unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];

using namespace std;

extern void Crash(MSGID nCode);
extern void dbgprint(const char* fmt, ...);
extern int AverageImages_main();

JKQueue*	Database::m_pTestQueue;
HANDLE		Database::m_hTestTask;
JKTask		Database::m_TestTask;
SECTOR		Database::m_ATR_DB[MAX_SECTOR_NUM];
FILE*		Database::m_hSummaryFile = NULL;

BOOL Database::Init()
{
	memset(m_ATR_DB, 0, sizeof(m_ATR_DB));
	m_hTestTask = NULL;
	m_pTestQueue = new JKQueue(g_nTestQSize);
	_tprintf(_T("Database():  Init...  STANDBY.\n"));

	InitDB();

	if (m_pTestQueue)
```

```
                return TRUE;
        else
        {
                _tprintf(_T("Database():  unable to create queue"));
                ExitProcess(0);
                return FALSE;
        }
}


void Database::Priority(int nPriority)
{
        m_TestTask.SetPriority(nPriority);

        _tprintf(_T("Database():  Set Task Priority.\n"));
}


BOOL Database::Start()
{
        m_hTestTask = (HANDLE)m_TestTask.Start(Entry, NULL);

        if (!m_hTestTask)
                Crash(Q_TASK_FAIL_TO_START);

        return (BOOL)(m_hTestTask != INVALID_HANDLE_VALUE);
}


BOOL Database::Send(MSGID nMsgId, void* pData)
{
        if (!m_pTestQueue)
                return TRUE;  // not started yet.

        QMSG Msg(nMsgId, pData);

        return (m_pTestQueue->Enqueue(&Msg));
}


BOOL Database::Stop(void)
{
        if (m_pTestQueue)
                Send(Q_END_TASK);
        else
```

```
                return TRUE;  // Task MultiCast has not been started yet.  Nothing has to
be done.

        if (m_TestTask.Running())
        {
                DWORD dwError = m_TestTask.Wait(INFINITE);

                if (dwError == WAIT_OBJECT_0)
                        m_hTestTask = NULL;
                else
                        _tprintf(_T("Database():  Task fails to exit.\n"));
        }

        if (m_TestTask.Running())
        {
                // Dangerous, but I got no chance.
                BOOL bError = TerminateThread((HANDLE)m_hTestTask, 0);

                if (bError)
                {
                        m_hTestTask = NULL;
                        _tprintf(_T("Database():  Task is forced to exit.\n"));
                }
                else
                {
                        _tprintf(_T("Database():  Unable to terminate Database"));
                }
        }

        return TRUE;
}


BOOL Database::Dispatch(QMSG& Msg, int& nTimeOut)
{
        BOOL                bStatus = TRUE;
        MY_MSG                      *pMsg = NULL;
        CarInfo                     *pCarInfo = NULL;
        TargetInfo          *pTargetInfo = NULL;
        int                         nSectorIdx = 0, nCarType = 0;
        int                         nActualCarType, nHead, nTail;
        unsigned char*      pImage;

        switch (Msg.m_nMsgID)
        {
```

```
                case Q_DO_TEST_CMD:
                        pMsg = (MY_MSG*)Msg.m_pData;
                        if (pMsg)
                        {
                                dbgprint("Recv msg:  %s\n", pMsg->m_Data);
                        }
                        break;

                case Q_UPDATE_DB:
                        pMsg = (MY_MSG*)Msg.m_pData;

                        if (pMsg)
                        {
                                pCarInfo = pMsg->m_pCarInfo;
                                nSectorIdx = pCarInfo->m_nSectorIdx;
                                nCarType = pCarInfo->m_nActualType;

                                if
(m_ATR_DB[nSectorIdx].m_Car[nCarType].m_pAvgImg)
                                        delete
m_ATR_DB[nSectorIdx].m_Car[nCarType].m_pAvgImg;  // This one comes in as a 2-D
array.  See makeaverage.

                                m_ATR_DB[nSectorIdx].m_Car[nCarType].m_pAvgImg
= pCarInfo;
                        }
                        break;

                case Q_CLEAR_DB:
                        ClearDB();
                        break;

                case Q_INIT_DB:
                        InitDB();
                        break;

                case Q_DUMP_DB:
                        DumpDB();
                        break;

                case Q_CLEAR_SUMMARY:
                        ClearSummary();
                        break;

                case Q_TARGET_LOOKUP:
```

```
                        pMsg = (MY_MSG*)Msg.m_pData;
                        pTargetInfo = pMsg->m_pTargetInfo;
                        nActualCarType = pTargetInfo->m_nActualType;
                        nHead = pTargetInfo->m_head;
                        nTail = pTargetInfo->m_tail;
                        pImage = pTargetInfo->m_Img;
                        ClassifyTargetsLookup(nActualCarType, nHead, nTail, pImage);
                        delete pTargetInfo;
                        break;

                case Q_REPORT_SUMMARY:
                        ReportSummary();
                        break;

                case Q_END_TASK:
                        _tprintf(_T("Database():  Get end task msg.\n"));
                        bStatus = FALSE;                        // Receive shutdown event.
                        break;

                default:
                        _tprintf(_T("Database():  Get msg id %d, no act taken.\n"),
Msg.m_nMsgID);
                        break;
        }

        if (Msg.m_pData)
        {
                delete Msg.m_pData;
                Msg.m_pData = NULL;
        }

        return bStatus;
}


unsigned __stdcall Database::Entry(LPVOID pArgList)
{
        _tprintf(_T("Database():  Start.\n"));

        int             nTimeOut = INFINITE;
        QMSG Msg;

        for (;;)
        {
                Msg.m_nMsgID = Q_MSG_NONE;
```

```
                m_pTestQueue->Dequeue(&Msg, nTimeOut);

                if (Msg.m_nMsgID)
                {
                   if (!Dispatch(Msg, nTimeOut))
                                break;  // Dispatch fails.
                }
        }

        _tprintf(_T("Database():  Entry is down!\n"));

        m_TestTask.Stop();
        _endthreadex(0);
        return 0;
}


void Database::InitDB()
{
        CarInfo* pAvgImg;
        string vehiclename;
        string database_name;
        unsigned char byte;
        int r, c;
        int a;
        int vehiclenum, i;

        for (vehiclenum = 0; vehiclenum < 8; vehiclenum++)
        {
                switch(vehiclenum)
                {
                        case 0 : vehiclename = "2S1"; break;
                        case 1 : vehiclename = "BRDM_2"; break;
                        case 2 : vehiclename = "BTR-60"; break;
                        case 3 : vehiclename = "D7"; break;
                        case 4 : vehiclename = "SLICY"; break;
                        case 5 : vehiclename = "T62"; break;
                        case 6 : vehiclename = "ZIL131"; break;
                        case 7 : vehiclename = "ZSU_23_4"; break;
                        default : vehiclename = "";
                }

                for (i = 0; i < g_nSectorNum; i++)
                {
```

```
                        int filenum = g_nSectorInDegs*i;
                        database_name =
".\\training\\"+vehiclename+"\\average_bmp\\average"+static_cast<ostringstream*>(
&(ostringstream() << filenum) )->str()+".bmp";

                        ifstream imagein (database_name.c_str(), ios::in | ios::binary);

                        if (!imagein)
                                break;

                        pAvgImg = new CarInfo;       // first idx - sector degree, second idx
- car type

                //retrieve header from file
                        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                header_info[a] = imagein.get();

                //retrieve image data from file
                        for(r=NUM_ROWS-1;r>=0;r--)
                                for(c=0;c<NUM_COLS;c++)
                                {
                                        byte = imagein.get();
                                        pAvgImg->m_Img[r][c] = byte;
                                }

                        imagein.close();
                        pAvgImg->m_nActualType = vehiclenum;
                        pAvgImg->m_nSectorIdx = filenum;
                        Database::m_ATR_DB[i].m_Car[vehiclenum].m_pAvgImg =
pAvgImg;       // first idx - sector number, second idx - car type
                }
        }

        _tprintf(_T("Database():  Init...  DONE!\n"));
}


void Database::ClearDB(void)
{
        int i = 0, j = 0;

        for (i=0; i<g_nSectorNum; i++)
        {
                for (j=0; j<NUM_OF_CAR_TYPES; j++)
                {
```

```
                        if (m_ATR_DB[i].m_Car[j].m_pAvgImg)
                        {
                                delete m_ATR_DB[i].m_Car[j].m_pAvgImg;
                                m_ATR_DB[i].m_Car[j].m_pAvgImg = NULL;
                        }
                }
        }

        _tprintf(_T("Database():  ClearDB done!\n"));
}


void Database::DumpDB(void)
{
        int i = 0, j = 0;

        for (i=0; i<g_nSectorNum; i++)
        {
                for (j=0; j<NUM_OF_CAR_TYPES; j++)
                        printf("DumpDB:  Sect %d CarType %d AVgimg %p\n", i, j,
m_ATR_DB[i].m_Car[j].m_pAvgImg);
        }

        _tprintf(_T("Database():  DumpDB done!\n"));
}


void Database::Delete(void)
{
        ClearDB();

        delete m_pTestQueue;
        m_pTestQueue = NULL;
}


void Database::ClassifyTargetsLookup(int nActualCarType, int nHead, int nTail,
unsigned char* pImage)
{
        int matchedpixels = 0, highestmatch = 0;
        int nCarIdx = 0, nIdx = 0;
        int nExpCarIdx = 0;
        int nCurrSectIdx = 0;
        unsigned char* pImg = NULL;
```

```
	for (nCarIdx=0; nCarIdx < NUM_OF_CAR_TYPES; nCarIdx++)
	{
		if ((m_ATR_DB[nHead].m_Car[nCarIdx].m_pAvgImg == NULL) ||
(m_ATR_DB[nTail].m_Car[nCarIdx].m_pAvgImg == NULL))
		{
			dbgprint("Warning!  No database is set up.  Train the system.\n");
			return;
		}

		matchedpixels = 0;

		// Treat it as a one-dimension array.  Easy to perform logic operation.
		pImg = (unsigned
char*)m_ATR_DB[nHead].m_Car[nCarIdx].m_pAvgImg->m_Img;

		for (nIdx=0; nIdx<IMAGE_SZ; nIdx++)
		{
			if (!(pImage[nIdx] ^ pImg[nIdx]))	//use XNOR to include
matching background pixels
				matchedpixels++;
		}

		if (matchedpixels > highestmatch)
		{
			nCurrSectIdx = nHead;
			highestmatch = matchedpixels;
			nExpCarIdx = nCarIdx;
		}

		matchedpixels = 0;

		// Treat it as a one-dimension array.  Easy to perform logic operation.
		pImg = (unsigned
char*)m_ATR_DB[nTail].m_Car[nCarIdx].m_pAvgImg->m_Img;

		for (nIdx=0; nIdx<IMAGE_SZ; nIdx++)
		{
			if (!(pImage[nIdx] ^ pImg[nIdx]))	//use XNOR to include
matching background pixels
				matchedpixels++;
		}

		if (matchedpixels > highestmatch)
		{
			nCurrSectIdx = nTail;
```

```
                        highestmatch = matchedpixels;
                        nExpCarIdx = nCarIdx;
                }
        }

        if (nExpCarIdx == nActualCarType)
        {
                m_ATR_DB[nCurrSectIdx].m_nCorrectCnt++;

        m_ATR_DB[nCurrSectIdx].m_Car[nExpCarIdx].m_nCorrectCntPerAvgImg++;
        }

        m_ATR_DB[nCurrSectIdx].m_Car[nActualCarType].m_nTotalCntPerType++;
}


void Database::ClearSummary()
{
        int nSectIdx = 0;
        int nCarIdx = 0;

        printf("Database::ClearSummary()\n");

        for (nSectIdx=0; nSectIdx<g_nSectorNum; nSectIdx++)
        {
                m_ATR_DB[nSectIdx].m_nCorrectCnt = 0;
                m_ATR_DB[nSectIdx].m_nTotalCntPerSect = 0;

                for (nCarIdx=0; nCarIdx<NUM_OF_CAR_TYPES; nCarIdx++)
                {

        m_ATR_DB[nSectIdx].m_Car[nCarIdx].m_nCorrectCntPerAvgImg = 0;
                        m_ATR_DB[nSectIdx].m_Car[nCarIdx].m_nTotalCntPerType =
0;
                }
        }
}


void Database::ReportSummary()
{
        int nSectIdx = 0;
        int nCarIdx = 0;
        int nCorrectCnt = 0;
        int nTotalCnt = 0;
```

```
	float fMatchRatio = 0.0;
	char Buffer[80], Summary[1200];

	m_hSummaryFile = NULL;
	m_hSummaryFile = fopen ("./ATRSummary.txt","a+");		// Open for append and update.  Created if not existed.

	if (m_hSummaryFile == NULL)
		dbgprint("Warning:  Open ATRSummary.txt failed.  Will not append the new statistics.\n");

	printf("***	Database::ReportSummary	***\n");
	memset(Summary, 0, sizeof(Summary));

	printf("SmplSpc %04.2f SectSz %d\n", g_fSampleSpcRate, g_nSectorInDegs);
	sprintf(Buffer, "SmplSpc %04.2f SectSz %d ", g_fSampleSpcRate, g_nSectorInDegs);
	strcat(Summary, Buffer);

	for (nCarIdx=0; nCarIdx<NUM_OF_CAR_TYPES; nCarIdx++)
	{
		memset(Buffer, 0, sizeof(Buffer));
		nCorrectCnt = nTotalCnt = 0;

		for (nSectIdx=0; nSectIdx<g_nSectorNum; nSectIdx++)
		{
			nCorrectCnt += m_ATR_DB[nSectIdx].m_Car[nCarIdx].m_nCorrectCntPerAvgImg;
			nTotalCnt += m_ATR_DB[nSectIdx].m_Car[nCarIdx].m_nTotalCntPerType;
		}
		fMatchRatio = (float)((float)nCorrectCnt/(float)nTotalCnt);
		printf("%sMatched %d/%d Ratio %04.2f\n", g_szVehicleName[nCarIdx], nCorrectCnt, nTotalCnt, fMatchRatio);
		sprintf(Buffer, "%s%04.2f ", g_szVehicleName[nCarIdx], fMatchRatio);
		strcat(Summary, Buffer);
	}

	if (m_hSummaryFile)
	{
		fprintf(m_hSummaryFile, "%s\n", Summary);
		fclose(m_hSummaryFile);
		m_hSummaryFile = NULL;
	}
}
```

ATREngine.cpp

```
#include<fstream>
#include<iostream>
#include "JKQueue.h"
#include "JKTask.h"
#include "ATREngine.h"
#include "HistoMedianFilter.h"
#include "Raw2Tif.h"
#include "Database.h"
#include "ClassifyTargets.h"

// Random header
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#include "Config.h"

#include<string>
using namespace std;

extern void Crash(MSGID nCode);
extern void dbgprint(const char* fmt, ...);
extern int RestoreFolders_main(bool bDelRaw);
extern int g_nSectSizeIdx;
extern int g_nSmplPctIdx;

JKQueue*	ATREngine::m_pTestQueue;
HANDLE		ATREngine::m_hTestTask;
JKTask		ATREngine::m_TestTask;

int ATREngine::m_nCycle = 0;
int ATREngine::m_nSectSizeIdx = DEFAULT_SECTOR_SIZE_IDX;
int ATREngine::m_nSmplPctIdx = DEFAULT_SAMPLE_SIZE_IDX;
int ATREngine::m_nNormSmplCnt = 0;
ATRState  ATREngine::m_nATRState = eATR_Idle;

inline int RangedRandInt( int range_min, int range_max )
{
	int nTickCnts =0;	// Take the lower 16 bits and make it an integer.
	int nRand = 0;
	double dblRand = 0.0;
```

```
        // In Windows, the rand() always returns the same random number sequence.
        // Use the following simple exclusive-OR with Windows GetTickCount to do the
trick.

          nTickCnts = GetTickCount() & 0x7FFF;     // Take the lower 16 bits and make it
an integer.
          nRand = rand();
          dblRand = (double)(nRand ^ nTickCnts);
          return (int)((double)(dblRand * (double)(range_max - range_min)) /
(RAND_MAX + 1) + range_min);
}

void RangedRandDemo( int range_min, int range_max, int n )
{
  // Generate random numbers in the half-closed interval
  // [range_min, range_max). In other words,
  // range_min <= random number < range_max
  int i;
  for ( i = 0; i < n; i++ )
  {
    int u = (int)((double)rand() / (RAND_MAX + 1) * (range_max - range_min)
        + range_min);
    printf( "  %6d\n", u);
  }
}

BOOL ATREngine::Init()
{
        m_hTestTask = NULL;
        m_pTestQueue = new JKQueue(g_nTestQSize);
        _tprintf(_T("ATREngine():  Init.\n"));

        m_nCycle = 0;
        m_nSectSizeIdx = DEFAULT_SECTOR_SIZE_IDX;
        m_nSmplPctIdx = DEFAULT_SAMPLE_SIZE_IDX;
        m_nNormSmplCnt = 0;

        if (m_pTestQueue)
                return TRUE;
        else
        {
                _tprintf(_T("ATREngine():  unable to create queue"));
                ExitProcess(0);
                return FALSE;
```

```
        }
}


void ATREngine::Delete(void)
{
        delete m_pTestQueue;
        m_pTestQueue = NULL;
}


void ATREngine::Priority(int nPriority)
{
        m_TestTask.SetPriority(nPriority);

        _tprintf(_T("ATREngine():  Set Task Priority.\n"));
}


BOOL ATREngine::Start()
{
        m_hTestTask = (HANDLE)m_TestTask.Start(Entry, NULL);

        if (!m_hTestTask)
                Crash(Q_TASK_FAIL_TO_START);

        return (BOOL)(m_hTestTask != INVALID_HANDLE_VALUE);
}


BOOL ATREngine::Send(MSGID nMsgId, void* pData)
{
        if (!m_pTestQueue)
                return TRUE;  // not started yet.

        QMSG Msg(nMsgId, pData);

        return (m_pTestQueue->Enqueue(&Msg));
}


BOOL ATREngine::Stop(void)
{
        if (m_pTestQueue)
                Send(Q_END_TASK);
```

```
	else
		return TRUE;  // Task MultiCast has not been started yet.  Nothing has to
be done.

	if (m_TestTask.Running())
	{
		DWORD dwError = m_TestTask.Wait(INFINITE);

		if (dwError == WAIT_OBJECT_0)
			m_hTestTask = NULL;
		else
			_tprintf(_T("ATREngine():  Task fails to exit.\n"));
	}

	if (m_TestTask.Running())
	{
		// Dangerous, but I got no chance.
		BOOL bError = TerminateThread((HANDLE)m_hTestTask, 0);

		if (bError)
		{
			m_hTestTask = NULL;
			_tprintf(_T("ATREngine():  Task is forced to exit.\n"));
		}
		else
		{
			_tprintf(_T("ATREngine():  Unable to terminate ATREngine"));
		}
	}

	return TRUE;
}


void ATREngine::Cleanup(void)
{
	bool bDelRaw = true;

	// Clear database should be done way earlier than deleting the files.
	// Too lazy to implement the synchronization here.
	if (Database::Send(Q_CLEAR_DB) == false)
	{
		dbgprint("crash file is %s, line is %d.\n", __FILE__, __LINE__);
		Crash(Q_EXIT_FROM_ERROR);
	}
```

```
	RestoreFolders_main(bDelRaw);	// Delete all train and unknown files.
}


void ATREngine::PrepareImage(int nSmplPctIdx)
{
	int nSmplSpcPctIdx = -1;
	int nDestDir = -1;
	int nActSmplCnt = 0;
	string vehiclename, name, full_name, dest_name, copy_cmd;
	int vehiclenum = 0;

	// Is this a valid setting?
	if ((nSmplPctIdx >= 0) && (nSmplPctIdx < SAMPLE_SPACE_TBL_SIZE))
	{
		nSmplSpcPctIdx = nSmplPctIdx;
		m_nSmplPctIdx = nSmplPctIdx;	// Fix the sample space rate for the following runs.
		m_nNormSmplCnt = g_nSampleSpacePct[nSmplSpcPctIdx];
	}
	else
	{
		// Randomize for other condition and overwrite passed-in nSmplPctIdx.
		m_nSmplPctIdx = SAMPLE_SPACE_RANDOM_PICK;		// Randomize the following runs.
		nSmplSpcPctIdx = RangedRandInt( 0, SAMPLE_SPACE_TBL_SIZE );
		m_nNormSmplCnt = g_nSampleSpacePct[nSmplSpcPctIdx];
	}

	g_nSmplPctIdx = nSmplSpcPctIdx;
	g_fSampleSpcRate = (float) (m_nNormSmplCnt) / (float) TOTAL_IMAGES_PER_TYPE;

	// Make directories before copying image files from ImageSource to destination directories.

	system("mkdir .\\training\\2S1\\place_raw_here");
	system("mkdir .\\training\\BRDM_2\\place_raw_here");
	system("mkdir .\\training\\BTR-60\\place_raw_here");
	system("mkdir .\\training\\D7\\place_raw_here");
	system("mkdir .\\training\\SLICY\\place_raw_here");
	system("mkdir .\\training\\T62\\place_raw_here");
	system("mkdir .\\training\\ZIL131\\place_raw_here");
	system("mkdir .\\training\\ZSU_23_4\\place_raw_here");
```

```
        system("mkdir .\\unknown\\2S1\\place_raw_here");
        system("mkdir .\\unknown\\BRDM_2\\place_raw_here");
        system("mkdir .\\unknown\\BTR-60\\place_raw_here");
        system("mkdir .\\unknown\\D7\\place_raw_here");
        system("mkdir .\\unknown\\SLICY\\place_raw_here");
        system("mkdir .\\unknown\\T62\\place_raw_here");
        system("mkdir .\\unknown\\ZIL131\\place_raw_here");
        system("mkdir .\\unknown\\ZSU_23_4\\place_raw_here");

        for (vehiclenum = 0; vehiclenum < 8; vehiclenum++)
        {
                switch(vehiclenum)
                {
                        case 0 : vehiclename = "2S1"; break;
                        case 1 : vehiclename = "BRDM_2"; break;
                        case 2 : vehiclename = "BTR-60"; break;
                        case 3 : vehiclename = "D7"; break;
                        case 4 : vehiclename = "SLICY"; break;
                        case 5 : vehiclename = "T62"; break;
                        case 6 : vehiclename = "ZIL131"; break;
                        case 7 : vehiclename = "ZSU_23_4"; break;
                        default : vehiclename = "";
                }

                nActSmplCnt = (m_nNormSmplCnt *
g_nTotalImagesPerType[vehiclenum]) / TOTAL_IMAGES_PER_TYPE;

                ifstream
ImgSrcfilenames(".\\ImageSource\\"+vehiclename+"\\place_raw_here\\filename.txt",
ios::in);
                ofstream
Trainingfilenames(".\\Training\\"+vehiclename+"\\place_raw_here\\filename.txt",
ios::out);
                ofstream
Unknownfilenames(".\\Unknown\\"+vehiclename+"\\place_raw_here\\filename.txt",
ios::out);

                string   ImgSrcDir =
".\\ImageSource\\"+vehiclename+"\\place_raw_here\\",
                                TrainingDir =
".\\Training\\"+vehiclename+"\\place_raw_here\\",
                                UnknownDir =
".\\Unknown\\"+vehiclename+"\\place_raw_here\\";

                while (!ImgSrcfilenames.eof())
```

```
		{
		ImgSrcfilenames >> name;
			full_name = ImgSrcDir + name;

			// Does this file go to Training or Unknown directory?

			if (RangedRandInt( 0, g_nTotalImagesPerType[vehiclenum]) <
nActSmplCnt)
			{
				Unknownfilenames << name;
				dest_name = UnknownDir + name;

				// This could generate a line which has only carriage return,
				// While there might be more images in the ImgSrc folder,
				// we cannot guarantee if the rest of the images all go to
train or unknown,
				// because we randomly assign images to train and
unknown folders.

				if (!ImgSrcfilenames.eof())
					Unknownfilenames << endl;
			}
			else
			{
				Trainingfilenames << name;
				dest_name = TrainingDir + name;

				// This could generate a line which has only carriage return,
				// While there might be more images in the ImgSrc folder,
				// we cannot guarantee if the rest of the images all go to
train or unknown,
				// because we randomly assign images to train and
unknown folders.

				if (!ImgSrcfilenames.eof())
					Trainingfilenames << endl;
			}

			// Copy from ImageSource to nDestDir.  The image file must exist.
			// write filename to nDestDir filename.txt. (could be Training or
Unknown.)
			copy_cmd = "copy " + full_name + " " + dest_name + "> nil";
	// To accelerate file copy, add "> nil" to depress the operating system reply.
			system(copy_cmd.c_str());		// Pass in copy_cmd buffer as
the command.
```

```
                }

                ImgSrcfilenames.close();
                Trainingfilenames.close();
                Unknownfilenames.close();
        }
}


void ATREngine::AdjustSectorSize(int nSectSizeIdx)
{
        int nSectSzIdx = SECTOR_SIZE_RANDOM_PICK;

        if ((nSectSizeIdx >= 0) && (nSectSizeIdx < SECTOR_TABLE_SIZE))
        {
                // Valid sector size index
                m_nSectSizeIdx = nSectSizeIdx;
                nSectSzIdx = nSectSizeIdx;
        }
        else
        {
                // Random pick sector size for other condition and overwrite passed-in
nSmplPctIdx.
                m_nSectSizeIdx = SECTOR_SIZE_RANDOM_PICK;
                nSectSzIdx = RangedRandInt( 0, SECTOR_TABLE_SIZE );
        }

        g_nSectSizeIdx = nSectSzIdx;
        g_nSectorInDegs = g_nSectorSize[nSectSzIdx];
        // Sector size can only be updated after the train and unknown data have been set
up.
        g_nSectorNum = 360 / g_nSectorInDegs;     // 360 degrees per circle.  So, we
have (360 / SectorInDegs) sectors.

}


BOOL ATREngine::Dispatch(QMSG& Msg, int& nTimeOut)
{
        BOOL            bStatus = TRUE;
        MY_MSG                  *pMsg = NULL;
        CycleInfo* pCycleInfo = NULL;
        int nCycle = -1;
        int nSectSizeIdx = -1;
        int nSmplPctIdx = -1;
```

```
        switch (Msg.m_nMsgID)
        {
                case Q_DO_TEST_CMD:
                        pMsg = (MY_MSG*)Msg.m_pData;
                        if (pMsg)
                        {
                                dbgprint("Recv msg:  %s\n", pMsg->m_Data);
                        }
                        break;

                case Q_RESTORE:
                        if (m_nATRState != eATR_Idle)
                        {
                                dbgprint("ATREngine is not Idle (state %d).  Ignore
Restore request.\n", m_nATRState);
                                break;
                        }

                        pMsg = (MY_MSG*)Msg.m_pData;
                        if (pMsg)
                        {
                                pCycleInfo = (CycleInfo*)&pMsg->m_pCycleInfo;
                                dbgprint("Running RestoreFolders_main\n");
                                RestoreFolders_main(pMsg->m_bDelRaw);
                        }
                        break;

                case Q_UPDATE_CYCLE:
                        pMsg = (MY_MSG*)Msg.m_pData;
                        if (pMsg)
                        {
                                pMsg = (MY_MSG*)Msg.m_pData;
                                pCycleInfo = pMsg->m_pCycleInfo;
                                nCycle = pCycleInfo->m_nCycle;
                                nSectSizeIdx = pCycleInfo->m_nSectSizeIdx;
                                nSmplPctIdx = pCycleInfo->m_nSmplPctIdx;
                                dbgprint("ATREngine:Q_UPDATE_CYCLE cycle %d
nSectSizeIdx %d nSmplPctIdx %d\n", nCycle, nSectSizeIdx, nSmplPctIdx);
                                delete pCycleInfo;

                                if (nCycle >= 0)
                                        m_nCycle = nCycle;
                                else
                                        break;          // Ignore invalid cycle.
```

```
                                if (m_nCycle >= 0)
                                {
                                        dbgprint("%d cycle(s) to go.\n", m_nCycle);

                                        if (m_nATRState == eATR_Idle)
                                        {
                                                Cleanup();
                                                PrepareImage(nSmplPctIdx);
                                                AdjustSectorSize(nSectSizeIdx);

                                                m_nATRState = eATR_Learning;
                                                pMsg = new MY_MSG(Q_RAW_2_TIF);
                                                if (Raw2Tif::Send(Q_RAW_2_TIF, pMsg)
== false)
                                                {
                                                        dbgprint("crash file is %s, line is
%d.\n", __FILE__, __LINE__);
                                                        delete pMsg;
                                                        Crash(Q_EXIT_FROM_ERROR);
                                                }
                                        }
                                        else
                                        {
                                                dbgprint("ATR status is %d.  Update cycle
count to %d but ignore the rest of the command.\n", m_nATRState, m_nCycle);
                                        }
                                }
                        }
                        break;

                case Q_TRAINING_DONE:
                        if (m_nCycle == 0)
                        {
                                dbgprint("Cycle count is %d.  Training complete.  Return
to idle per user request.\n", m_nCycle);
                                m_nATRState = eATR_Idle;
                                break;
                        }

                        dbgprint("Machine learning completed!\n");
                        m_nATRState = eATR_Matching;
                        pMsg = new MY_MSG(Q_CLASSIFY);

                        if (ClassifyTargets::Send(Q_CLASSIFY, pMsg) == false)
```

```
                        {
                                dbgprint("crash file is %s, line is %d.\n", __FILE__,
__LINE__);
                                delete pMsg;
                                Crash(Q_EXIT_FROM_ERROR);
                        }
                        break;

                case Q_CLASSIFY_DONE:
                        if (m_nCycle == 0)
                        {
                                dbgprint("Cycle count is %d.  Classification complete.
Return to idle per user request.\n", m_nCycle);
                                m_nATRState = eATR_Idle;
                                break;
                        }

                        dbgprint("Target matching done!\n");

                        if (m_nCycle > 0)
                                m_nCycle--;

                        if (m_nCycle > 0)
                        {
                                dbgprint("%d cycle(s) to go.\n", m_nCycle);

                                Cleanup();

                                // Proceed with existing setting.
                                PrepareImage(m_nSmplPctIdx);
                                AdjustSectorSize(m_nSectSizeIdx);

                                m_nATRState = eATR_Learning;
                                pMsg = new MY_MSG(Q_RAW_2_TIF);
                                if (Raw2Tif::Send(Q_RAW_2_TIF, pMsg) == false)
                                {
                                        dbgprint("crash file is %s, line is %d.\n",
__FILE__, __LINE__);
                                        delete pMsg;
                                        Crash(Q_EXIT_FROM_ERROR);
                                }
                        }
                        else
                        {
```

```
                                dbgprint("Cycle count is %d.  Operation completed\n",
m_nCycle);
                                m_nATRState = eATR_Idle;
                        }
                        break;

                case Q_END_TASK:
                        _tprintf(_T("ATREngine():  Get end task msg.\n"));
                        bStatus = FALSE;                        // Receive shutdown event.
                        break;

                default:
                        _tprintf(_T("ATREngine():  Get msg id %d, no act taken.\n"),
Msg.m_nMsgID);
                        break;
        }

        if (Msg.m_pData)
        {
                delete Msg.m_pData;
                Msg.m_pData = NULL;
        }

        return bStatus;
}


unsigned __stdcall ATREngine::Entry(LPVOID pArgList)
{
        _tprintf(_T("ATREngine():  Start.\n"));

        int             nTimeOut = g_nSystemTick;
        QMSG Msg;

        for (;;)
        {
                Msg.m_nMsgID = Q_MSG_NONE;

                m_pTestQueue->Dequeue(&Msg, nTimeOut);

                if (Msg.m_nMsgID)
                {
                   if (!Dispatch(Msg, nTimeOut))
                                break;  // Dispatch fails.
                }
```

```
        }

        _tprintf(_T("ATREngine():  Entry is down!\n"));

        m_TestTask.Stop();
        _endthreadex(0);
        return 0;
}
```

convert_to_bmp.cpp

```
#include<fstream>
#include<iostream>
#include<string>
#include<string.h>
#include<direct.h>

#include "Config.h"

//constants
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024
#define TIF_HEADER_SIZE 8

//static locals
static unsigned char image_i[NUM_ROWS][NUM_COLS];
static unsigned char image_invert[NUM_ROWS][NUM_COLS];
static unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];

//prototypes
void initialize_bmp_header();
void initialize_bmp_invert();

using namespace std;

static string vehiclename;

int Raw2Tif_main(string imagetype)
{
        for(int vehiclenum = 0; vehiclenum < 8; vehiclenum++)
        {
                switch(vehiclenum)
                {
```

```
                        case 0 : vehiclename = "2S1"; break;
                        case 1 : vehiclename = "BRDM_2"; break;
                        case 2 : vehiclename = "BTR-60"; break;
                        case 3 : vehiclename = "D7"; break;
                        case 4 : vehiclename = "SLICY"; break;
                        case 5 : vehiclename = "T62"; break;
                        case 6 : vehiclename = "ZIL131"; break;
                        case 7 : vehiclename = "ZSU_23_4"; break;
                        default : vehiclename = "";
                }

                string raw_to_bmp = ".\\"+imagetype+"\\"+vehiclename+"\\raw_to_bmp";

                //Create raw_to_bmp folder
                _mkdir(raw_to_bmp.c_str());

                //Set place_raw_here folder as input and raw_to_bmp folder as output
                ifstream
filenames(".\\"+imagetype+"\\"+vehiclename+"\\place_raw_here\\filename.txt", ios::in);
                ofstream
newnames(".\\"+imagetype+"\\"+vehiclename+"\\raw_to_bmp\\filename.txt", ios::out);

                string name,

        before=".\\"+imagetype+"\\"+vehiclename+"\\place_raw_here\\",

        after=".\\"+imagetype+"\\"+vehiclename+"\\raw_to_bmp\\",
                                full_name;

                int image_size = NUM_COLS * NUM_ROWS;
                int index;

                int zero_pad_cols = NUM_COLS % 4; //number of values in a bitmap row
must be divisible by 4

                //array indexes
                int r, c;
                int a;

                //initialize header information
                initialize_bmp_header();

                while(!filenames.eof())
                {
                        name = "";              // Empty string buffer.
```

```
                filenames >> name;

                // In case this line has only a carriage return.
                // This could happen while randomly generating unknown and train
images.
                // See ATREngine::PrepareImage for more information.

                if (name.size() == 0)
                        continue;       // Skip carriage return.

                full_name = before + name;
                ifstream imagein(full_name.c_str(), ios::in | ios::binary);

                if(imagein.fail())
                        imagein.clear();
                else
                {
                        index = name.find(".raw",0);
                        name.replace(index,4,".bmp");

                        full_name = after + name;
                        ofstream imageout(full_name.c_str(), ios::out | ios::binary);

                //retrieve image data from TIFF file
                        for(r=0;r<NUM_ROWS;r++)
                                for(c=0;c<NUM_COLS;c++)
                                        image_i[r][c] = imagein.get();

                //call bmp invert function required for bmp format output
                        initialize_bmp_invert();

                //output bmp header
        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                imageout.put(header_info[a]);
                //output bmp image data
                        for(r=0;r<NUM_ROWS;r++)
                        {
                                for(c=0;c<NUM_COLS;c++)
                                        imageout.put(image_invert[r][c]);

                        //zero padding the end of rows if # of cols not divisible by 4
                                for(a=0;a<zero_pad_cols;a++)
                                        imageout.put('0');
                        }
```

```
                        //write new bmp filename and threshold value to text file
                                newnames << name;
                                if(!filenames.eof())
                                        newnames << endl;

                                imageout.close();
                        }
                        imagein.close();
                }
                filenames.close();
                newnames.close();
        }
        printf("Raw2Tif_main() complete\n");
        return 0;
}

void initialize_bmp_header()
{
        unsigned char byte = 0x00, reserved = 0x00;
        int i;
        header_info[0] = 0x42;   //bmp identifier value 0x4d42
        header_info[1] = 0x4d;
        header_info[2] = 0xaa;   //file size, (128 x 128 + 1074) [bytes] = 0x39aa
        header_info[3] = 0x39;
        header_info[4] = 0x00;
        header_info[5] = 0x00;
        header_info[6] = 0x00;   //reserved1, always 0
        header_info[7] = 0x00;
        header_info[8] = 0x00;   //reserved2, always 0
        header_info[9] = 0x00;
        header_info[10] = 0x36;  //offset to pixel data, 1078 [bytes] = 0x0436
        header_info[11] = 0x04;
        header_info[12] = 0x00;
        header_info[13] = 0x00;
        header_info[14] = 0x28;  //size of info header, 40 [bytes] = 0x28
        header_info[15] = 0x00;
        header_info[16] = 0x00;
        header_info[17] = 0x00;
        header_info[18] = NUM_COLS;  //image width in pixels, 128 [pixels] = 0x80
        header_info[19] = 0x00;
        header_info[20] = 0x00;
        header_info[21] = 0x00;
        header_info[22] = NUM_ROWS;  //image height in pixels, 128 [pixels] = 0x80
        header_info[23] = 0x00;
        header_info[24] = 0x00;
```

```
        header_info[25] = 0x00;
        header_info[26] = 0x01;  //# of planes?, 0x01
        header_info[27] = 0x00;
        header_info[28] = 0x08;  //# of bits per pixel, 8 [bits] = 0x08
        header_info[29] = 0x00;
        header_info[30] = 0x00;  //amount of compression, 0x00
        header_info[31] = 0x00;
        header_info[32] = 0x00;
        header_info[33] = 0x00;
        header_info[34] = 0x00;  //image data in bytes. 0x4000 (no compression)
        header_info[35] = 0x40;
        header_info[36] = 0x00;
        header_info[37] = 0x00;
        header_info[38] = 0x00;  //horizontal pixels per meter, 0x00
        header_info[39] = 0x00;
        header_info[40] = 0x00;
        header_info[41] = 0x00;
        header_info[42] = 0x00;  //vertical pixels per meter, 0x00
        header_info[43] = 0x00;
        header_info[44] = 0x00;
        header_info[45] = 0x00;
        header_info[46] = 0x00;  //# of colors used, 0x00
        header_info[47] = 0x00;
        header_info[48] = 0x00;
        header_info[49] = 0x00;
        header_info[50] = 0x00;  //# of important colors, 0x00
        header_info[51] = 0x00;
        header_info[52] = 0x00;
        header_info[53] = 0x00;

   //include rgbquad array
        for(i=BMP_HEADER_SIZE;i<(BMP_PALETTE_SIZE+BMP_HEADER_SIZE)
;i+=4)
        {
                header_info[i] = byte;
                header_info[i+1] = byte;
                header_info[i+2] = byte;
                header_info[i+3] = reserved;
                byte += 0x01;
        }

        return;
}

void initialize_bmp_invert()
```

```
{
   int r, c;
   for(r=0;r<NUM_ROWS;r++)
      for(c=0;c<NUM_COLS;c++)
         image_invert[c][r] = image_i[NUM_COLS-1-c][r];
   return;
}
```

threshold_and_medfilter.cpp

```
#include<fstream>
#include<iostream>
#include<string>
#include<string.h>
#include<direct.h>

#include "Config.h"

//situation-dependent constants
#define THRESH_MIN 0.955
#define THRESH_MAX 0.985            //threshold set above this percent of the pixels:
recommend >0.96
#define MED_FILTER_ITERATIONS 1      //number of binary image median filter
iterations: recommend 0, 1, or 2

//constants that should not be changed
#define NUM_GRAY_LEVELS 256
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024

//globals
static unsigned char image_i[NUM_ROWS][NUM_COLS];
static unsigned char image_o[NUM_ROWS][NUM_COLS];
static unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];

//prototypes
unsigned char median_filter(int r, int c);

using namespace std;

static string vehiclename;

int HistoMedianFilter_main(string imagetype)
```

```
{
	for(int vehiclenum = 0; vehiclenum < 8; vehiclenum++)
	{
		switch(vehiclenum)
		{
			case 0 : vehiclename = "2S1"; break;
			case 1 : vehiclename = "BRDM_2"; break;
			case 2 : vehiclename = "BTR-60"; break;
			case 3 : vehiclename = "D7"; break;
			case 4 : vehiclename = "SLICY"; break;
			case 5 : vehiclename = "T62"; break;
			case 6 : vehiclename = "ZIL131"; break;
			case 7 : vehiclename = "ZSU_23_4"; break;
			default : vehiclename = "";
		}

		string threshold_bmp =
".\\"+imagetype+"\\"+vehiclename+"\\threshold_bmp";

		//Create threshold_bmp folder
		_mkdir(threshold_bmp.c_str());

		//Set raw_to_bmp folder as input and threshold_bmp folder as output
		ifstream
filenames(".\\"+imagetype+"\\"+vehiclename+"\\raw_to_bmp\\filename.txt", ios::in);
		ofstream
newnames(".\\"+imagetype+"\\"+vehiclename+"\\threshold_bmp\\filename.txt", ios::out);

		string name,
				before =
".\\"+imagetype+"\\"+vehiclename+"\\raw_to_bmp\\",
				after =
".\\"+imagetype+"\\"+vehiclename+"\\threshold_bmp\\",
				full_name;

		//histogram related variables
		int histogram[NUM_GRAY_LEVELS];
		int number_of_pixels = NUM_ROWS * NUM_COLS;
		int pixel_count, total_pixels, last_value;
		int threshold;
		bool reached_thresh;

		int zero_pad_cols = NUM_COLS % 4; //number of values in a bitmap row
must be divisible by 4
```

```
		//for loop indexes
		int r, c;
		int a;

		while(!filenames.eof())
		{
			name = "";		// Empty string buffer.

		//handle image file io
		filenames >> name;

			// In case this line has only a carriage return.
			// This could happen while randomly generating unknown and train
images.
			// See ATREngine::PrepareImage for more information.

			if (name.size() == 0)
				continue;	// Skip carriage return.

			full_name = before + name;
			ifstream imagein (full_name.c_str(), ios::in | ios::binary);

			full_name = after + name;
			ofstream imageout (full_name.c_str(), ios::out | ios::binary);

		//initialize histogram
			for(a=0;a<NUM_GRAY_LEVELS;a++)
				histogram[a] = 0;

		//retrieve header from file
			for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
				header_info[a] = imagein.get();
		//retrieve image data from file
			for(r=0;r<NUM_ROWS;r++)
				for(c=0;c<NUM_COLS;c++)
				{
					image_i[r][c] = imagein.get();
				//set histogram values
					histogram[image_i[r][c]]++;
				}

		//reset histogram[0] in the event of zero-padding so threshold calculation
is not corrupted
			if(histogram[0]>(NUM_ROWS*NUM_COLS/2))
				histogram[0] = 0;
```

```
                //calculate total number of pixels for the purpose of thresholding (may not equal NUM_ROWS*NUM_COLS)
                        total_pixels = 0;
                        for(a=0;a<NUM_GRAY_LEVELS;a++)
                                total_pixels += histogram[a];


                //find binary threshold from histogram
                        last_value = 0;
                        pixel_count = 0;
                        reached_thresh = false;
                        for(a=0;a<NUM_GRAY_LEVELS;a++)
                        {
                                pixel_count += histogram[a];
                                if(histogram[a]>0)
                                {
                                        if(1.0*pixel_count/total_pixels >= THRESH_MIN)
                                        {
                                                if(1.0*pixel_count/total_pixels >= THRESH_MAX && reached_thresh==false)
                                                {
                                                        threshold = last_value;
                                                        reached_thresh = true;
                                                }
                                                else if(histogram[a]>=histogram[last_value] && reached_thresh==false)
                                                {
                                                        if(1.0*(pixel_count-histogram[a])/total_pixels >= THRESH_MIN)
                                                        {
                                                                threshold = last_value;
                                                                reached_thresh = true;
                                                        }
                                                }
                                                else if(a==NUM_GRAY_LEVELS-1 && reached_thresh==false)
                                                {
                                                        threshold = a;
                                                        reached_thresh = true;
                                                }
                                        }
                                        last_value = a;
                                }
                        }
```

```
                        //convert output image to binary
                        for(r=0;r<NUM_ROWS;r++)
                                for(c=0;c<NUM_COLS;c++)
                                {
                                        if(r>=NUM_ROWS/4 && r<3*NUM_ROWS/4
&& c>=NUM_COLS/4 && c<3*NUM_COLS/4)
                                        {
                                                if(image_i[r][c]>=threshold)

        image_o[r][c]=NUM_GRAY_LEVELS-1;
                                                else
                                                        image_o[r][c]=0;
                                        }
                                        else
                                                image_o[r][c]=0;
                                }

                //median filter the binary image
                        for(a=0;a<MED_FILTER_ITERATIONS;a++)
                        {
                        //reset image_o to image_i
                                for(r=0;r<NUM_ROWS;r++)
                                        for(c=0;c<NUM_COLS;c++)
                                                image_i[r][c] = image_o[r][c];
                        //calculate binary median filter
                                for(r=0;r<NUM_ROWS;r++)
                                        for(c=0;c<NUM_COLS;c++)
                                                image_o[r][c] = median_filter(r,c);
                        }

                //write bmp header to new file
                        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                imageout.put(header_info[a]);
                //write image data to file
                        for(r=0;r<NUM_ROWS;r++)
                        {
                                for(c=0;c<NUM_COLS;c++)
                                        imageout.put(image_o[r][c]);

                        //zero padding the end of rows if # of cols not divisible by 4
                                for(a=0;a<zero_pad_cols;a++)
                                        imageout.put('0');
                        }
```

```
                        imagein.close();
                        imageout.close();

                        newnames << name;
                        if(!filenames.eof())
                                newnames << endl;
                }
                filenames.close();
                newnames.close();
        }
        printf("HistoMedianFilter_main() complete\n");
        return 0;
}

unsigned char median_filter(int r, int c)
{
   int a, b;
   int sort[9], sort_index = 0;
   int median_value;
   int hold;

   //populate sort array
   for(a=r-1;a<=r+1;a++)
      for(b=c-1;b<=c+1;b++)
      {
         if(a<0 || b<0 || a>=NUM_ROWS || b>=NUM_COLS)
            sort[sort_index++] = 0;
         else
            sort[sort_index++] = image_i[a][b];
      }

   //bubble sort
   for(a=0;a<8;a++)
      for(b=0;b<(8-a);b++)
         if(sort[b]<sort[b+1])
         {
            hold = sort[b];
            sort[b] = sort[b+1];
            sort[b+1] = hold;
         }

   median_value = sort[4];
   return (unsigned char)median_value;
}
```

adaptive_filter.cpp

```
/*
Notes:
    1.  This program requires binary images for proper operation
    2.  Each call of suppress_outliers() requires a prior count_bodies() call for population initialization
*/
#include<fstream>
#include<iostream>
#include<iomanip>
#include<math.h>
#include<string>
#include<string.h>
#include<direct.h>

#include "Config.h"

//situation-dependent constants
#define NUM_GRAY_LEVELS 256

//constants that should not be changed
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024
#define PI 3.14159265359

//global variables
unsigned char image_i[NUM_ROWS][NUM_COLS];
unsigned char image_o[NUM_ROWS][NUM_COLS];
short contiguous_piece[NUM_ROWS][NUM_COLS];
unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];
double slope, intercept;
static bool CvsR;

int r_center, c_center;

extern int rnd(double val);

//prototypes
int count_bodies();
void kill_group(int group_num);
void suppress_outliers(int body_count);
void ls_filter(int body_count);
void calculate_mass_center(int which_body, int population);
void connect_group(int from_which, int to_which, int to_rc, int to_cc);
```

```
void compute_ls_parameters(int N);

using namespace std;

static string vehiclename;

int AdaptiveFilter_main(string imagetype)
{
	int vehiclenum = 0;
	int zero_pad_cols = NUM_COLS % 4; //number of values in a bitmap row must be divisible by 4
	unsigned char byte;
	int num_bodies;	//stores the number of separated pieces in binary image
	int r, c;
	int a;

	for (vehiclenum = 0; vehiclenum < 8; vehiclenum++)
	{
		switch(vehiclenum)
		{
			case 0 : vehiclename = "2S1"; break;
			case 1 : vehiclename = "BRDM_2"; break;
			case 2 : vehiclename = "BTR-60"; break;
			case 3 : vehiclename = "D7"; break;
			case 4 : vehiclename = "SLICY"; break;
			case 5 : vehiclename = "T62"; break;
			case 6 : vehiclename = "ZIL131"; break;
			case 7 : vehiclename = "ZSU_23_4"; break;
			default : vehiclename = "";
		}

		string adaptive_filter_bmp = ".\\"+imagetype+"\\"+vehiclename+"\\adaptive_filter_bmp";

		//Create adaptive_filter_bmp folder
		_mkdir(adaptive_filter_bmp.c_str());

		//Set threshold_bmp folder as input and adaptive_filter_bmp folder as output
		ifstream filenames(".\\"+imagetype+"\\"+vehiclename+"\\threshold_bmp\\filename.txt", ios::in);
		ofstream newnames(".\\"+imagetype+"\\"+vehiclename+"\\adaptive_filter_bmp\\filename.txt", ios::out);
```

```
                string name,
                                before =
".\\"+imagetype+"\\"+vehiclename+"\\threshold_bmp\\",
                                after =
".\\"+imagetype+"\\"+vehiclename+"\\adaptive_filter_bmp\\",
                                full_name;

                while(!filenames.eof())
                {
                        name = "";              // Empty string buffer.

                //handle image file io
                filenames >> name;

                        // In case this line has only a carriage return.
                        // This could happen while randomly generating unknown and train
images.
                        // See ATREngine::PrepareImage for more information.

                        if (name.size() == 0)
                                continue;       // Skip carriage return.

                        full_name = before + name;
                        ifstream imagein (full_name.c_str(), ios::in | ios::binary);

                        full_name = after + name;
                        ofstream imageout (full_name.c_str(), ios::out | ios::binary);

                //retrieve header from file
                        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                header_info[a] = imagein.get();
                //retrieve image data from file
                        for(r=NUM_ROWS-1;r>=0;r--)
                                for(c=0;c<NUM_COLS;c++)
                                {
                                        byte = imagein.get();
                                        image_i[r][c] = byte;
                                }
                //count the number of target "pieces" in the image
                        num_bodies = count_bodies();
                        if(num_bodies > 1)
                        {
                                ls_filter(num_bodies);
                                num_bodies = count_bodies();
                                while(num_bodies>1)
```

```
                                {
                                        suppress_outliers(num_bodies);
                                        num_bodies = count_bodies();
                                }
                        }

                        if(num_bodies>1)
                                cout<<"DING! DING! DING!"<<endl;

                        for(r=0;r<NUM_ROWS;r++)
                                for(c=0;c<NUM_COLS;c++)
                                        image_o[r][c] = image_i[r][c];


                //write bmp header to new file
                        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                imageout.put(header_info[a]);
                //write image data to file
                        for(r=NUM_ROWS-1;r>=0;r--)
                        {
                                for(c=0;c<NUM_COLS;c++)
                                        imageout.put(image_o[r][c]);

                        //zero padding the end of rows if # of cols not divisible by 4
                                for(a=0;a<zero_pad_cols;a++)
                                        imageout.put('0');
                        }

                        imagein.close();
                        imageout.close();

                        newnames << name;
                        if(!filenames.eof())
                                newnames << endl;
                }
                filenames.close();
                newnames.close();
        }
        printf("AdaptiveFilter_main() complete\n");
        return 0;
}

int count_bodies()
{
        int r, c, r_current, c_current, hits;
```

```
	int total_hits = 0;
	int ff = NUM_GRAY_LEVELS - 1;
	int (*will_check)[NUM_COLS] = new int[NUM_ROWS][NUM_COLS]; //>=1 if
needs to be checked, 0 otherwise
	int (*return_to_pixel)[2] = new int[NUM_ROWS*NUM_COLS][2];
	int return_to_pixel_length, return_to_pixel_current_index;
	int moved_from;
	int piece_num = 0;
	bool is_counting = false;

	short Cprev = 5, Cnext = 3, Rprev = 2, Rnext = 7; //L = left, R = right, U = up, D
= down

//initialize will_check: all points in center 50% of image set to 1, others set to 0
//  also initialize contiguous_piece array which will count the number of dislocated bodies
	for(r=0;r<NUM_ROWS;r++)
		for(c=0;c<NUM_COLS;c++)
		{
			contiguous_piece[r][c] = 0;
	if(r<NUM_ROWS/4 || r>3*NUM_ROWS/4 || c<NUM_COLS/4 ||
c>3*NUM_COLS/4)
				will_check[r][c] = 0;
			else
			   will_check[r][c] = 1;
		}
//set will_check array for each pixel based on neighboring pixels
	for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
		for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
		{
			if((int)image_i[r][c] > 0)
			{
		  if(image_i[r-1][c] == ff)
					will_check[r][c] *= Rprev;
				if(image_i[r][c-1] == ff)
					will_check[r][c] *= Cprev;
				if(image_i[r+1][c] == ff)
					will_check[r][c] *= Rnext;
				if(image_i[r][c+1] == ff)
					will_check[r][c] *= Cnext;
			}
			else
			   will_check[r][c] = 0;
		}
	for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
		for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
```

```
                {
                        if(image_i[r][c] == ff && will_check[r][c] > 0)
                        {
                                piece_num++;    return_to_pixel_length = 0;
        //each test must initialize these variables
           hits = 0;          moved_from = 1;
                                c_current = c;  r_current = r;
                                return_to_pixel_current_index = 0;
                                is_counting = true;

                                do
                                {
                   will_check[r_current][c_current] /= moved_from;
                contiguous_piece[r_current][c_current] = piece_num;
                                //see if a white pixel is in the previous row and has not been
checked already
                if(will_check[r_current][c_current]%Rprev==0 &&
will_check[r_current][c_current]>0)
                {
                                                will_check[r_current][c_current] /= Rprev;
                                        //if the previous row contains the only adjacent pixel that
has not yet been counted,
                //  count the current pixel now
                                                if(will_check[r_current][c_current]==1)
                                                {
                                                        total_hits++;
                                                        will_check[r_current][c_current] =
0;
                                                }
                                        //otherwise, keep the current pixel as a reference to
return later
                                                else
                                                {

        return_to_pixel[return_to_pixel_length][0] = r_current;

        return_to_pixel[return_to_pixel_length][1] = c_current;
                                                        return_to_pixel_length++;
                                                }
                                        //the next iteration will move to the pixel in the previous
row
                                                r_current = r_current-1;
                                                moved_from = Rnext;
                                        }
```

```
		else if(will_check[r_current][c_current]%Cnext==0 &&
will_check[r_current][c_current]>0)
		{
							will_check[r_current][c_current] /= Cnext;
					//if the previous row contains the only adjacent pixel that
has not yet been counted,
		//  count the current pixel now
							if(will_check[r_current][c_current]==1)
							{
								total_hits++;
								will_check[r_current][c_current] =
0;
							}
					//otherwise, keep the current pixel as a reference to
return later
							else
							{

	return_to_pixel[return_to_pixel_length][0] = r_current;

	return_to_pixel[return_to_pixel_length][1] = c_current;
								return_to_pixel_length++;
							}
					//the next iteration will move to the pixel in the next
column
							c_current = c_current+1;
							moved_from = Cprev;
				}
		else if(will_check[r_current][c_current]%Rnext==0 &&
will_check[r_current][c_current]>0)
		{
							will_check[r_current][c_current] /= Rnext;
					//if the next row contains the only adjacent pixel that has
not yet been counted,
		//  count the current pixel now
							if(will_check[r_current][c_current]==1)
							{
								total_hits++;
								will_check[r_current][c_current] =
0;
							}
					//otherwise, keep the current pixel as a reference to
return later
							else
							{
```

```
		return_to_pixel[return_to_pixel_length][0] = r_current;

		return_to_pixel[return_to_pixel_length][1] = c_current;
                                                        return_to_pixel_length++;
                                                }
                                        //the next iteration will move to the pixel in the next row
                                                r_current = r_current+1;
                                                moved_from = Rprev;
                                        }
                        else if(will_check[r_current][c_current]%Cprev==0 &&
will_check[r_current][c_current]>0)
                        {
                                                will_check[r_current][c_current] /= Cprev;
                                        //if the previous column contains the only adjacent pixel
that has not yet been counted,
                        //  count the current pixel now
                                                if(will_check[r_current][c_current]==1)
                                                {
                                                        total_hits++;
                                                        will_check[r_current][c_current] =
0;
                                                }
                                        //otherwise, keep the current pixel as a reference to
return later
                                                else
                                                {

		return_to_pixel[return_to_pixel_length][0] = r_current;

		return_to_pixel[return_to_pixel_length][1] = c_current;
                                                        return_to_pixel_length++;
                                                }
                                        //the next iteration will move to the pixel in the previous
column
                                                c_current = c_current-1;
                                                moved_from = Cnext;
                                        }
                                        else if(return_to_pixel_current_index <
return_to_pixel_length)
                                        {
                                                if(will_check[r_current][c_current]==1)
                                                {
                                                        hits++;
```

```
                                                    will_check[r_current][c_current] =
0;
                                                    }
                                                    r_current =
return_to_pixel[return_to_pixel_current_index][0];
                                                    c_current =
return_to_pixel[return_to_pixel_current_index][1];
                                                    return_to_pixel_current_index++;
                                                    moved_from = 1;
                                                }
                                                else if(will_check[r_current][c_current]==1)
                                                {
                    hits++;
                    will_check[r_current][c_current] = 0;
                    is_counting = false;
                }
                else
                                                        is_counting = false;
                                            }while(is_counting == true);
            }
        }
    delete [] return_to_pixel;
    delete [] will_check;
    return piece_num;
}

//this command will annihilate any pixels assigned to group "group_num"
void kill_group(int group_num)
{
    int r, c;
    for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
        for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
            if(contiguous_piece[r][c]==group_num)
                image_i[r][c] = 0;
    return;
}

//this command will connect pixels assigned to group "from_which" to pixels
//  assigned to group "to_which"; to_rc, to_cc are the (r,c) center of mass
//  coordinates for group "to_which"
void connect_group(int from_which, int to_which, int to_rc, int to_cc)
{
    int delta_c, delta_r;
    int r, c;
    int r1, c1;
```

```
int Rnxt = 0, Cnxt = 0;  //will be used to give direct the cursor
double total_r_moved, total_c_moved;
int currently_at;
double slope;
int rc = to_rc, cc = to_cc;
int (*temporary)[NUM_COLS] = new int[NUM_ROWS][NUM_COLS];
for(r=0;r<NUM_ROWS;r++)
   for(c=0;c<NUM_COLS;c++)
      temporary[r][c] = image_i[r][c];

for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
   for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
   {
      if(from_which == contiguous_piece[r][c])
      {
         currently_at = contiguous_piece[r][c];
         total_r_moved=0; total_c_moved=0;
      //starting r1, c1 at r, c
         r1 = r;         c1 = c;
      //set directions for r1 and c1 to move
         delta_r = rc - r;
         if(delta_r>0)
            Rnxt = 1;
         else
            Rnxt = -1;
         delta_c = cc - c;
         if(delta_c>0)
            Cnxt = 1;
         else
            Cnxt = -1;
      //commence connection
         if(abs(delta_c)>abs(delta_r))
         {
            slope = fabs((double)delta_r)/fabs((double)delta_c);
            while(currently_at!=to_which)
            {
               if(total_r_moved>0 && total_c_moved>0)
                  if(total_r_moved/total_c_moved < slope)
                  {
                     r1 = r1 + Rnxt;
                     total_r_moved++;
                  }
                  else
                  {
                     c1 = c1 + Cnxt;
```

```
                    total_c_moved++;
                }
            else if(total_r_moved == 0)
            {
                r1 = r1 + Rnxt;
                total_r_moved++;
            }
            else
            {
                c1 = c1 + Cnxt;
                total_c_moved++;
            }
            currently_at = contiguous_piece[r1][c1];
            if(currently_at == 0)
            {
                temporary[r1][c1] = NUM_GRAY_LEVELS - 1;
            }
        }
    }
    else
    {
        slope = fabs((double)delta_c)/fabs((double)delta_r);
        while(currently_at!=to_which)
        {
            if(total_r_moved>0 && total_c_moved>0)
                if(total_c_moved/total_r_moved < slope)
                {
                    c1 = c1 + Cnxt;
                    total_c_moved++;
                }
                else
                {
                    r1 = r1 + Rnxt;
                    total_r_moved++;
                }
            else if(total_c_moved == 0)
            {
                c1 = c1 + Cnxt;
                total_c_moved++;
            }
            else
            {
                r1 = r1 + Rnxt;
                total_r_moved++;
            }
```

```
                currently_at = contiguous_piece[r1][c1];
                if(currently_at == 0)
                {
                   temporary[r1][c1] = NUM_GRAY_LEVELS - 1;
                }
             }
          }
       }
    }
  for(r=0;r<NUM_ROWS;r++)
     for(c=0;c<NUM_COLS;c++)
        image_i[r][c] = temporary[r][c];

  delete [] temporary;
  return;
}
void suppress_outliers(int body_count)
{
  int *population = new int[body_count];        //stores # of pixels in each body
  int (*mass_center)[2] = new int[body_count][2];//stores (r,c) coordinates for each
center of mass
  int (*sort)[2] = new int[body_count][2];      //sorted populations and corresponding
class#
  int main_body, body_to_test;                  //if present, main body
contains>=80% of image white pixels
  int num_bodies = body_count;
  bool does_pass;                  //true if no suppression necessary
  int total_white_pixels = 0;          //total # of image white pixels
  int body, hold;
  int rstart, cstart;               //will hold center of mass coordinates
  int radius;                      //test radius (square) for suppression
  int a, b, r, c;
  int r_cm_main, c_cm_main, r_cm_test, c_cm_test;
  double dx, min_dist_to_main;

//initialize population array to zero
  for(a=0;a<body_count;a++)
     population[a]=0;
//count number of members for each distinct body of pixels in the image
  for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
     for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
     {
        body = contiguous_piece[r][c];
        if(body>0)
        {
```

```
            population[body-1]++;
            total_white_pixels++;
         }
      }
//bubble sort the populations
   for(a=0;a<body_count;a++)
   {
      sort[a][0] = a;
      sort[a][1] = population[a];
   }
   for(a=0;a<body_count;a++)
      for(b=0;b<(body_count-1-a);b++)
         if(sort[b][1]<sort[b+1][1])
         {
            body = sort[b][1];          hold = sort[b][0];
            sort[b][1] = sort[b+1][1];   sort[b][0] = sort[b+1][0];
            sort[b+1][1] = body;         sort[b+1][0] = hold;
         }
   main_body = sort[0][0];

//calculate the center of mass for each distinct body
   for(a=0;a<body_count;a++)
   {
      calculate_mass_center(a+1, population[a]);
      mass_center[a][0] = r_center;
      mass_center[a][1] = c_center;
   }

//find nearest to main body
   r_cm_main = mass_center[main_body][0];
   c_cm_main = mass_center[main_body][1];
   min_dist_to_main = NUM_ROWS * NUM_COLS;
   for(a=0;a<num_bodies;a++)
   {
      if(a!=main_body)
      {
         r_cm_test = mass_center[a][0];
         c_cm_test = mass_center[a][1];

         dx = sqrt( pow(r_cm_main-r_cm_test,2.0) + pow(c_cm_main-c_cm_test,2.0) );

         if(dx<min_dist_to_main)
         {
            min_dist_to_main = dx;
            body_to_test = a;
```

```
          }
       }
    }
    if(true)
    {
       //for(a=0;a<body_count;a++)
       a = body_to_test;
         if(a!=main_body)
         {
            rstart = mass_center[a][0];
            cstart = mass_center[a][1];
            does_pass = false;
            radius = population[a];//minor bodies must reach the main body within a square
"radius" equal to the population of the body
            for(r=rstart-radius;r<=rstart+radius;r++)
               for(c=cstart-radius;c<=cstart+radius;c++)
               {
                  body = contiguous_piece[r][c];
                  if(body-1==main_body)
                     does_pass = true;
               }
            if(does_pass==true)

connect_group(a+1,main_body+1,mass_center[main_body][0],mass_center[main_body][
1]);
            else
               kill_group(a+1);
         }
    }

    delete [] sort;
    delete [] mass_center;
    delete [] population;
    return;
}
void calculate_mass_center(int which_body, int population)
{
          double r_c, c_c;
          int r, c;
          int Er = 0, Ec = 0;
          for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
                    for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
                              if (contiguous_piece[r][c]==which_body)
                              {
                                        Ec += c;          Er += r;
```

```
                        }
        c_c = (Ec*1.0) / population;
        r_c = (Er*1.0) / population;
        c_center = rnd(c_c);
        r_center = rnd(r_c);
        return;
}
void ls_filter(int body_count)
{
   int *population = new int[body_count];        //stores # of pixels in each body
   int (*mass_centers)[2] = new int[body_count][2];//stores (r,c) coordinates for each
center of mass
   int (*sort)[2] = new int[body_count][2];      //sorted populations and corresponding
class#
   int main_body;
   int total_white_pixels = 0;                  //total # of image white pixels
   int body, hold;
   int a, b, r, c;
   int total_r_moved, total_c_moved;
   int r_cm, c_cm;
   double r_line, c_line;
   double max_dist_to_line = 0;
   double dr, dc, dx, normal_dist;                                   //dr = row distance
from ls line
                                    //dc = col distance from ls line
                                    //dx = hypoteneuse of the corresponding triangle
                                    //normal_dist = perpendicular distance from ls line

//initialize population array to zero
   for(a=0;a<body_count;a++)
      population[a]=0;
//count number of members for each distinct body of pixels in the image
   for(r=NUM_ROWS/4;r<=3*NUM_ROWS/4;r++)
      for(c=NUM_COLS/4;c<=3*NUM_COLS/4;c++)
      {
         body = contiguous_piece[r][c];
         if(body>0)
         {
            population[body-1]++;
            total_white_pixels++;
         }
      }
   for(a=0;a<body_count;a++)
      if(population[a] >= total_white_pixels*90/100)
         main_body = a;
```

```
//bubble sort the populations
   for(a=0;a<body_count;a++)
   {
      sort[a][0] = a;
      sort[a][1] = population[a];
   }
   for(a=0;a<body_count;a++)
      for(b=0;b<(body_count-1-a);b++)
         if(sort[b][1]<sort[b+1][1])
         {
            body = sort[b][1];          hold = sort[b][0];
            sort[b][1] = sort[b+1][1];   sort[b][0] = sort[b+1][0];
            sort[b+1][1] = body;         sort[b+1][0] = hold;
         }
   main_body = sort[0][0];
//initialize the center of mass array
   for(a=0;a<body_count;a++)
   {
      calculate_mass_center(sort[a][0]+1,sort[a][1]);
      mass_centers[sort[a][0]][0] = r_center;
      mass_centers[sort[a][0]][1] = c_center;
   }

//set least squares parameters for filtration purposes
   compute_ls_parameters(total_white_pixels);

//using the main white body in the image, find its maximum distance from the least
squares line
   for(r=0;r<NUM_ROWS;r++)
      for(c=0;c<NUM_COLS;c++)
         if(contiguous_piece[r][c]==main_body+1)
         {
            if(CvsR)
            {
               c_line = slope * r + intercept;
               r_line = (c - intercept) / slope;
               dr = r_line - r;
               dc = c_line - c;
               dx = sqrt(pow(dr,2)+pow(dc,2));
               if(dx==0)
                  normal_dist = 0;
               else
                  normal_dist = fabs(dr)*fabs(dc)/dx;
            }
```

```
            else
            {
               r_line = slope * c + intercept;
               c_line = (r - intercept) / slope;
               total_r_moved = 0; total_c_moved = 0;
               dr = r_line - r;
               dc = c_line - c;
               dx = sqrt(pow(dr,2)+pow(dc,2));
               if(dx==0)
                  normal_dist = 0;
               else
                  normal_dist = fabs(dr)*fabs(dc)/dx;
            }
            if(normal_dist>max_dist_to_line)
               max_dist_to_line = normal_dist;
         }

//run the least squares filter
   for(a=1;a<body_count;a++)
   {
   //retrieve center of mass coordinates for current body under test
      r_cm = mass_centers[sort[a][0]][0];  c_cm = mass_centers[sort[a][0]][1];
      if(CvsR)//using C vs R coordinate system
      {
      //calculate number of column and rows from the center of mass to the line
(horizontally and vertically)
         c_line = slope * r_cm + intercept;
         r_line = (c_cm - intercept) / slope;
         dr = r_line - r_cm;
         dc = c_line - c_cm;
      //calculate the perpendicular distance from the center of mass to the line
         dx = sqrt(pow(dr,2)+pow(dc,2));
         if(dx==0)
            normal_dist = 0;
         else
            normal_dist = fabs(dr)*fabs(dc)/dx;
      }
      else//using RvsC coordinate system
      {
      //calculate number of column and rows from the center of mass to the line
(horizontally and vertically)
         r_line = slope * c_cm + intercept;
         c_line = (r_cm - intercept) / slope;
         dr = r_line - r_cm;
         dc = c_line - c_cm;
```

```
      //calculate the perpendicular distance from the center of mass to the line
         dx = sqrt(pow(dr,2)+pow(dc,2));
         if(dx==0)
            normal_dist = 0;
         else
            normal_dist = fabs(dr)*fabs(dc)/dx;
      }
      if(normal_dist >= max_dist_to_line)//perpendicular distance from center of mass to
least squares line is unacceptable
{
         kill_group(sort[a][0]+1);
}
   }

   delete [] sort;
   delete [] mass_centers;
   delete [] population;
   return;
}
void compute_ls_parameters(int N)
{
//least squares parameters, "E" stands for "sum"
   double ls_Er2, ls_Er;
   double ls_Ec2, ls_Ec;
   double ls_total_pixels = N;
   double ls_Ecr;
   double ls_det_r, ls_det_c;
   double ls_theta_rad, ls_theta_deg, ls_slope, ls_intercept;
   bool draw_CvsR;

   int r, c;
   int pixel_val;

//compute least squares
   ls_Er2=0; ls_Ec2=0; ls_Er=0; ls_Ec=0; ls_Ecr=0;
   for(r=0;r<NUM_ROWS;r++)
      for(c=0;c<NUM_COLS;c++)
      {
         pixel_val = image_i[r][c];
         if(pixel_val==NUM_GRAY_LEVELS-1)
         {
            ls_Er+=r; ls_Er2+=r*r;
            ls_Ec+=c; ls_Ec2+=c*c;
            ls_Ecr+=c*r;
         }
```

```
        }

    ls_det_r = (ls_Er2*ls_total_pixels - ls_Er*ls_Er);
    ls_det_c = (ls_Ec2*ls_total_pixels - ls_Ec*ls_Ec);
//choose the axis orientation that minimizes error in angle calculation
    if(ls_det_r>=ls_det_c)
    {
        ls_slope = 1.0*(ls_total_pixels*ls_Ecr - ls_Ec*ls_Er) / ls_det_r;
        ls_intercept = 1.0*(ls_Er2*ls_Ec - ls_Er*ls_Ecr) / ls_det_r;
        ls_theta_rad = atan(ls_slope);
        if(ls_theta_rad<0)
            ls_theta_rad += PI;
        ls_theta_deg = ls_theta_rad * 180/PI;
        draw_CvsR = true;
    }
    else
    {
        ls_slope = 1.0*(ls_total_pixels*ls_Ecr - ls_Ec*ls_Er) / ls_det_c;
        ls_intercept = 1.0*(ls_Ec2*ls_Er - ls_Ec*ls_Ecr) / ls_det_c;
        ls_theta_rad = PI/2 - atan(ls_slope);
        ls_theta_deg = ls_theta_rad * 180/PI;
        draw_CvsR = false;
    }
    slope = ls_slope;
    intercept = ls_intercept;
    CvsR = draw_CvsR;

    return;
}
```

sector.cpp

```
//Note: All subroutines assume angles have units of (degrees)
#include<fstream>
#include<iostream>
#include<math.h>
#include<iomanip>
#include<string>
#include<string.h>

#include<vector>
#include<stdlib.h>
#include<cmath>
```

```
#include<sstream>
#include<direct.h>

#include<algorithm>

#include "Config.h"

using namespace std;

//constants that user may alter

//constants that should not be changed
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024
#define NUM_GRAY_LEVELS 256
#define PI 3.14159265359

#undef __USE_OLD_LOGIC__

//global variables
static unsigned char image_i[NUM_ROWS][NUM_COLS];
static unsigned char image_o[NUM_ROWS][NUM_COLS];
static unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];

unsigned char image_i_orig[NUM_ROWS][NUM_COLS];
unsigned char image_o_orig[NUM_ROWS][NUM_COLS];

static int r_center, c_center;

extern int rnd(double val);

//prototypes
void remove_newline(string textfile);
        //simple subroutine that removes empty line at end of text files

void rotate_image(string name, string src, string dest, int theta);
        //simple subroutine that rotates an image by theta degrees
void classify_headtail(string src, string dest1, string dest2, string dest1name, string
dest2name);
        //simple subroutine that divides images into two orientations (e.g. 0 degrees and
180 degrees)

string find_reference(string src);
        //simple subroutine that finds image with median number of target pixels from list
of names in a text file
```

```
double find_angle(string full_name);

static string vehiclename;

int AssignSectors_main()
{
	for(int vehiclenum = 0; vehiclenum < 8; vehiclenum++)
	{
		switch(vehiclenum)
		{
			case 0 : vehiclename = "2S1"; break;
			case 1 : vehiclename = "BRDM_2"; break;
			case 2 : vehiclename = "BTR-60"; break;
			case 3 : vehiclename = "D7"; break;
			case 4 : vehiclename = "SLICY"; break;
			case 5 : vehiclename = "T62"; break;
			case 6 : vehiclename = "ZIL131"; break;
			case 7 : vehiclename = "ZSU_23_4"; break;
			default : vehiclename = "";
		}

		//Set adaptive_filter_bmp folder as input and assign_sectors_bmp folder as
output
		ifstream
filenames(".\\training\\"+vehiclename+"\\adaptive_filter_bmp\\filename.txt", ios::in);

		string filesector[MAX_SECTOR_NUM/2],
folder[MAX_SECTOR_NUM];
		ofstream namesector[MAX_SECTOR_NUM/2];

		for (int i = 0; i < g_nSectorNum/2; i++)
		{
			filesector[i] =
".\\training\\"+vehiclename+"\\adaptive_filter_bmp\\"+"filename"+static_cast<ostringstre
am*>( &(ostringstream() << i) )->str()+".txt";
			namesector[i].open(filesector[i], ios::out);
		}

		for (int i = 0; i < g_nSectorNum/2; i++)
		{
			namesector[i].precision(3);
			namesector[i]<<setfill(' ');
		}
```

```
		string name_bef,
			name_aft,
			dir2 = ".\\training\\"+vehiclename+"\\adaptive_filter_bmp\\",
			dir4 = ".\\training\\"+vehiclename+"\\average_bmp\\",
			full_name;

		//Create average_bmp folder
		_mkdir(dir4.c_str());

		double theta_perpendicular, maximum, minimum;

		while(!filenames.eof())
		{
			filenames >> name_bef;
			name_aft = name_bef;

			full_name = dir2 + name_bef;

			//find angle of current image to be assigned to a sector
			theta_perpendicular = find_angle(full_name);

			for (int i = 0; i < g_nSectorNum/2; i++)
			{
				if (i == 0)
				{
					maximum = g_nSectorInDegs/2;
					minimum = 180-(g_nSectorInDegs/2);
					if (theta_perpendicular < maximum ||
theta_perpendicular >= minimum)
					{
						namesector[i]<<name_aft;
						if(!filenames.eof())
							namesector[i]<<endl;
					}
				}
				else
				{
					minimum = maximum;
					maximum = minimum+g_nSectorInDegs;
					if (theta_perpendicular < maximum &&
theta_perpendicular >= minimum)
					{
						namesector[i]<<name_aft;
						if(!filenames.eof())
							namesector[i]<<endl;
```

```
                                        }
                                }
                        }
                }

                filenames.close();

                for (int i = 0; i < g_nSectorNum/2; i++)
                {
                        namesector[i].close();
                        remove_newline(filesector[i]);
                }

                for (int i = 0; i < g_nSectorNum; i++)
                {
                        int foldernum = g_nSectorInDegs*i;
                        folder[i] = dir4+static_cast<ostringstream*>( &(ostringstream() << foldernum) )->str()+"/";
                        _mkdir(folder[i].c_str());
                }

                for (int i = 0; i < g_nSectorNum/2; i++)
                {
                        int foldernum_head = g_nSectorInDegs*i, foldernum_tail = (g_nSectorInDegs*i)+180;

        classify_headtail(filesector[i],folder[i],folder[i+(g_nSectorNum/2)],"filename"+static_cast<ostringstream*>( &(ostringstream() << foldernum_head) )->str()+".txt","filename"+static_cast<ostringstream*>( &(ostringstream() << foldernum_tail) )->str()+".txt");
                }
        }

        printf("AssignSectors_main() complete\n");

        return 0;
}

void remove_newline(string textfile)
{
        ifstream fin(textfile);
        vector<string> vs;
        string s;
        while(getline(fin,s))
                vs.push_back(s);
```

```
        fin.close();

        ofstream fout(textfile);
        for(vector<string>::iterator it = vs.begin(); it != vs.end(); ++it)
        {
                if(it != vs.begin())
                        fout << '\n';
                fout << *it;
        }
        fout.close();
}

void rotate_image(string name, string src, string dest, int theta)
{
   string full_name;

        int zero_pad_cols = NUM_COLS % 4; //number of values in a bitmap row must
be divisible by 4

   unsigned char byte;

   //for loop indexes
        int r, c;
   int a;

//handle image file io
        full_name = src + name;
        ifstream imagein (full_name.c_str(), ios::in | ios::binary);

        full_name = dest + name;//Angle + "_" + name;
        ofstream imageout (full_name.c_str(), ios::out | ios::binary);

//retrieve header from file
   for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
      header_info[a] = imagein.get();
//retrieve image data from file
   for(r=NUM_ROWS-1;r>=0;r--)
                for(c=0;c<NUM_COLS;c++)
                {
        byte = imagein.get();
        image_i[r][c] = byte;
      }

        double rval, cval;
        int Er = 0, Ec = 0, n = 0;
```

```
        for(r=NUM_ROWS/4;r<3*NUM_ROWS/4;r++)
                for(c=NUM_COLS/4;c<3*NUM_COLS/4;c++)
                        if(image_i[r][c]==NUM_GRAY_LEVELS-1)
                        {
                                Ec += c;
                                Er += r;
                                n++;
                        }

        cval = (Ec*1.0) / n;
        rval = (Er*1.0) / n;
        c_center = rnd(cval);
        r_center = rnd(rval);

        int width = NUM_COLS; // using 'constant' rather than literals within the code
        int height = NUM_ROWS;
        int background = 0; // this is the background color - use a suitable value here

        float rads = (float)(theta*3.1415926/180.0); // fixed constant PI
        float cs = cos(-rads); // precalculate these values
        float ss = sin(-rads);
        float xcenter = (float)(c_center);   // use float here!
        float ycenter = (float)(r_center);
        for (int r = 0; r < height; ++r) {
          for (int c = 0; c < width; ++c) {
                // now find the pixel of the original image that is rotated to (r,c)
                // rotation formula assumes that origin = top-left and y points down
                int rorig = (int)(ycenter + ((float)(r)-ycenter)*cs - ((float)(c)-xcenter)*ss);
                int corig = (int)(xcenter + ((float)(r)-ycenter)*ss + ((float)(c)-
xcenter)*cs);
                // now get the pixel value if you can
                int pixel = background; // in case there is no original pixel
                if (rorig >= 0 && rorig < height && corig >= 0 && corig < width) {
                        pixel = image_i[rorig][corig];
                }
                image_o[r][c] = pixel;
          }
        }


   //write bmp header to new file
      for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
         imageout.put(header_info[a]);
   //write image data to file
      for(r=NUM_ROWS-1;r>=0;r--)
```

```
	{
				for(c=0;c<NUM_COLS;c++)
					imageout.put(image_o[r][c]);

		//zero padding the end of rows if # of cols not divisible by 4
		   for(a=0;a<zero_pad_cols;a++)
		      imageout.put('0');
		}

			imagein.close();
			imageout.close();
}

void classify_headtail(string src, string dest1, string dest2, string dest1name, string
dest2name)
{
		ifstream infile(src, ios::in);
		ofstream outfile1(dest1+dest1name, ios::out);
		ofstream outfile2(dest2+dest2name, ios::out);

		outfile1.precision(3);
		outfile1<<setfill(' ');
		outfile2.precision(3);
		outfile2<<setfill(' ');

		string temp_files0 = ".\\training\\"+vehiclename+"\\temp_files0";
		string temp_files180 = ".\\training\\"+vehiclename+"\\temp_files180";

		_mkdir(temp_files0.c_str());
		_mkdir(temp_files180.c_str());

   string name_bef,
		dir = ".\\training\\"+vehiclename+"\\adaptive_filter_bmp\\",
			dir2 = ".\\training\\"+vehiclename+"\\temp_files0/",
			dir3 = ".\\training\\"+vehiclename+"\\temp_files180/",
		full_name;

		int zero_pad_cols = NUM_COLS % 4; //number of values in a bitmap row must
be divisible by 4

		unsigned char inbyte;

   //indexes
		int r, c;
		int a;
```

```
        // double theta, theta_perpendicular, theta_orig;
        double theta_perpendicular, theta_orig;

        ////////////////////REFERENCE IMAGE////////////////////
        name_bef = find_reference(src);
        full_name = dir + name_bef;
        ifstream imagein_ref (full_name.c_str(), ios::in | ios::binary);

//retrieve header from input file
        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                header_info[a] = imagein_ref.get();
//retrieve data from input file, place in array and calculate best-fit line
        for(r=NUM_ROWS-1;r>=0;r--)
                for(c=0;c<NUM_COLS;c++)
                {
                        inbyte = imagein_ref.get();
                        image_i[r][c] = inbyte;
                        image_i_orig[r][c] = inbyte;
                }
                imagein_ref.close();
                //find angle
                theta_orig = find_angle(full_name);
        ////////////////////REFERENCE IMAGE////////////////////

        while(!infile.eof())
        {
                infile >> name_bef;

                full_name = dir + name_bef;
                ifstream imagein (full_name.c_str(), ios::in | ios::binary);

                //retrieve header from input file
    for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
        header_info[a] = imagein.get();
                //retrieve data from input file, place in array and calculate best-fit line
                for(r=NUM_ROWS-1;r>=0;r--)
                        for(c=0;c<NUM_COLS;c++)
                        {
           inbyte = imagein.get();
           image_i[r][c] = inbyte;
        }
                imagein.close();
                //find angle
                theta_perpendicular = find_angle(full_name);
```

```
                int matchedpixels1=0, matchedpixels2=0;

                rotate_image(name_bef,dir,dir2,(int)(theta_orig-theta_perpendicular));
        //align current image with reference image

                full_name = dir2 + name_bef;
                ifstream imagein2 (full_name.c_str(), ios::in | ios::binary);

                //retrieve header from input file
                for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                        header_info[a] = imagein2.get();
                //retrieve data from input file, place in array and calculate best-fit line
                for(r=NUM_ROWS-1;r>=0;r--)
                        for(c=0;c<NUM_COLS;c++)
                        {
                                inbyte = imagein2.get();
                                image_i[r][c] = inbyte;
                        }
                imagein2.close();

                rotate_image(name_bef,dir2,dir3,180);        //rotate aligned image 180
degrees

                for(r=0;r<NUM_ROWS;r++)
                        for(c=0;c<NUM_COLS;c++)
                        {
                                if (!(image_i_orig[r][c] ^ image_i[r][c]))      //use XNOR to
compare
                                        matchedpixels1++;
                                if (!(image_i_orig[r][c] ^ image_o[r][c]))
                                        matchedpixels2++;
                        }

                if (matchedpixels1 >= matchedpixels2)
                {
                        full_name = dest1 + name_bef;
                        outfile1<<name_bef;
                        if(!infile.eof())
                                outfile1<<endl;
                }

                else
                {
                        full_name = dest2 + name_bef;
```

```
                        outfile2<<name_bef;
                        if(!infile.eof())
                                outfile2<<endl;
                }

                ofstream imageout (full_name.c_str(), ios::out | ios::binary);

        //write bmp header to new file
                for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                        imageout.put(header_info[a]);
        //write image data to file
                for(r=NUM_ROWS-1;r>=0;r--)
                {
                        for(c=0;c<NUM_COLS;c++)
                                imageout.put(image_i[r][c]);

                //zero padding the end of rows if # of cols not divisible by 4
                        for(a=0;a<zero_pad_cols;a++)
                                imageout.put('0');
                }

                imageout.close();
        }

        infile.close();
        outfile1.close();
        outfile2.close();

        remove_newline(dest1+dest1name);
        remove_newline(dest2+dest2name);
}

string find_reference(string src)
{
        int a, r, c;        //index variables

        int median_index, median_num;
        vector<int> myvector;//number of target pixels of image files to be sorted

        ifstream infile(src, ios::in), infile2(src, ios::in);
        string refname;
        string name_bef, full_name, dir =
".\\training\\"+vehiclename+"\\adaptive_filter_bmp\\";

        while(!infile.eof())
```

```
        {
                int targetpixels = 0;

                unsigned char inbyte;

                infile >> name_bef;

                full_name = dir + name_bef;
                ifstream imagein (full_name.c_str(), ios::in | ios::binary);

  //retrieve header from input file
     for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
        header_info[a] = imagein.get();
  //retrieve data from input file and place in array
                for(r=NUM_ROWS-1;r>=0;r--)
                        for(c=0;c<NUM_COLS;c++)
                        {
           inbyte = imagein.get();
           image_i[r][c] = inbyte;
                                if (inbyte == 250)
                                {
                                        targetpixels++;
                                }
        }
                imagein.close();
                myvector.push_back(targetpixels);
        }
        infile.close();

        sort(myvector.begin(), myvector.end());

        median_index = (myvector.size()/2)-1;

        for (int i = 0; i < median_index; i++)
        {
                myvector.pop_back();
        }

        median_num = myvector.back();

        while(!infile2.eof())
        {
                int targetpixels = 0;

                unsigned char inbyte;
```

```
                infile2 >> name_bef;

                full_name = dir + name_bef;
                ifstream imagein (full_name.c_str(), ios::in | ios::binary);

   //retrieve header from input file
      for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
         header_info[a] = imagein.get();
   //retrieve data from input file and place in array
                for(r=NUM_ROWS-1;r>=0;r--)
                        for(c=0;c<NUM_COLS;c++)
                        {
            inbyte = imagein.get();
            image_i[r][c] = inbyte;
                                if (inbyte == 250)
                                {
                                        targetpixels++;
                                }
         }
                imagein.close();
                if (targetpixels == median_num)
                        refname = name_bef;
         }
         infile2.close();

         return refname;
}

double find_angle(string full_name)
{
         double ls_slope, theta_perpendicular, ls_theta;
         bool CvsR;

         //indexes
   int r, c, a;
   int pixel_val;

         ifstream imagein (full_name.c_str(), ios::in | ios::binary);

   //retrieve header from input file
      for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
      {
         header_info[a] = imagein.get();
      }
```

```
   //retrieve data from input file, place in array and calculate best-fit line
                  for(r=NUM_ROWS-1;r>=0;r--)
                        for(c=0;c<NUM_COLS;c++)
                        {
                  image_i[r][c] = imagein.get();
         }
                  imagein.close();


//least squares parameters, "E" stands for "sum"
   double Er2, Er;
   double Ec2, Ec;
   double total_pixels;
   double Ecr;
   double det_r, det_c;
   // double theta_rad, theta_deg, slope, intercept;
   double theta_rad, theta_deg, slope;
   bool draw_CvsR;

   //compute least squares
   total_pixels=0; Er2=0; Ec2=0; Er=0; Ec=0; Ecr=0;

   for(r=0;r<NUM_ROWS;r++)
      for(c=0;c<NUM_COLS;c++)
      {
         pixel_val = image_i[r][c];
         if(pixel_val==NUM_GRAY_LEVELS-1)
         {
            Er+=r; Er2+=r*r;
            Ec+=c; Ec2+=c*c;
            Ecr+=c*r;
            total_pixels++;
         }
      }

   det_r = (Er2*total_pixels - Er*Er);
   det_c = (Ec2*total_pixels - Ec*Ec);

   if(det_r>=det_c)
   {
      slope = 1.0*(total_pixels*Ecr - Ec*Er) / det_r;
      theta_rad = atan(slope);
      if(theta_rad<0)
         theta_rad += PI;
      theta_deg = theta_rad * 180/PI;
```

```
        draw_CvsR = true;
    }
    else
    {
        slope = 1.0*(total_pixels*Ecr - Ec*Er) / det_c;
        theta_rad = PI/2 - atan(slope);
        theta_deg = theta_rad * 180/PI;
        draw_CvsR = false;
    }
    ls_theta = theta_deg;
    CvsR = draw_CvsR;
    ls_slope = slope;

    if(ls_theta<90)
        theta_perpendicular = ls_theta+90;
    else
        theta_perpendicular = ls_theta-90;

            return theta_perpendicular;
}
```

average.cpp

```
#include<fstream>
#include<iostream>
#include<iomanip>
#include<math.h>
#include<string>
#include<string.h>
#include<sstream>

#include "Config.h"
#include "MyMsg.h"
#include "Database.h"

//situation-dependent constants
#define NUM_GRAY_LEVELS 256

//constants that should not be changed
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024
#define PI 3.14159265359
```

```
//global variables
static unsigned char image_i[NUM_ROWS][NUM_COLS];
static unsigned char image_o[NUM_ROWS][NUM_COLS];
static unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];

static int r_center, c_center;

using namespace std;

//prototypes
extern void calculate_mass_center();
extern int rnd(double val);
extern double find_angle(string full_name);
extern void Crash(MSGID nCode);
extern void dbgprint(const char* fmt, ...);

void makeaverage(string filename, string imagename, int vehicletype, int sectorindex);

static string vehiclename;

int AverageImages_main()
{
	int vehiclenum = 0, i = 0, filenum = 0, nSectIdx = 0;
	string file[MAX_SECTOR_NUM];

	for (vehiclenum = 0; vehiclenum < 8; vehiclenum++)
	{
		switch(vehiclenum)
		{
			case 0 : vehiclename = "2S1"; break;
			case 1 : vehiclename = "BRDM_2"; break;
			case 2 : vehiclename = "BTR-60"; break;
			case 3 : vehiclename = "D7"; break;
			case 4 : vehiclename = "SLICY"; break;
			case 5 : vehiclename = "T62"; break;
			case 6 : vehiclename = "ZIL131"; break;
			case 7 : vehiclename = "ZSU_23_4"; break;
			default : vehiclename = "";
		}

		for (i = 0; i < g_nSectorNum; i++)
		{
			filenum = g_nSectorInDegs*i;
			file[i] =
".\\training\\"+vehiclename+"\\average_bmp\\"+static_cast<ostringstream*>(
```

```
&(ostringstream() << filenum) )->str()+"\\filename"+static_cast<ostringstream*>(
&(ostringstream() << filenum) )->str()+".txt";
                }

                for (nSectIdx = 0; nSectIdx < g_nSectorNum; nSectIdx++)
                {
                        filenum = g_nSectorInDegs * nSectIdx;
                        makeaverage(file[nSectIdx],
"average"+static_cast<ostringstream*>( &(ostringstream() << filenum) )->str()+".bmp",
vehiclenum, nSectIdx);
                }
        }
        printf("AverageImages_main() complete\n");
        printf("ATR learning DONE!\n");
        return 0;
}

void makeaverage(string filename, string imagename, int vehicletype, int sectorindex)
{
   string name,
        before = ".\\training\\"+vehiclename+"\\adaptive_filter_bmp\\",
              after = ".\\training\\"+vehiclename+"\\average_bmp\\",
        full_name;

        ifstream namesangle(filename, ios::in);

        int zero_pad_cols = NUM_COLS % 4; //number of values in a bitmap row must
be divisible by 4
   unsigned char byte;
   //for loop indexes
        int r, c;
   int a;
        double numfiles = 0;
        double threshold;
        MY_MSG* pMsg = NULL;

        while(!namesangle.eof())
        {
                numfiles += 1;
        //handle image file io
        namesangle >> name;

                if (name == "")
                        numfiles = 0;
```

```
                if (numfiles == 0)
                        full_name =
".\\training\\"+vehiclename+"\\average_bmp\\empty.bmp";
                else
                        full_name = before + name;

                ifstream imagein (full_name.c_str(), ios::in | ios::binary);

                //retrieve header from file
                        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                header_info[a] = imagein.get();

        //retrieve image data from file
      for(r=NUM_ROWS-1;r>=0;r--)
                        for(c=0;c<NUM_COLS;c++)
                        {
           byte = imagein.get();
           image_i[r][c] = byte;
        }

      for(r=0;r<NUM_ROWS;r++)
                        for(c=0;c<NUM_COLS;c++)
                        {
                                if (numfiles == 1)
                                {
                                        if (image_i[r][c] == 255)
                                                image_o[r][c] = 1;
                                        else
                                                image_o[r][c] = 0;
                                }
                                else
                                {
                                        if (image_i[r][c] == 255)
                                                image_o[r][c] += 1;
                                }
                        }

                imagein.close();
        }

        threshold = numfiles/2;

        CarInfo *pCarInfo = new CarInfo;
        pCarInfo->m_nActualType = vehicletype;
        pCarInfo->m_nSectorIdx = sectorindex;
```

```
        for(r=0;r<NUM_ROWS;r++)
                for(c=0;c<NUM_COLS;c++)
                {
                        if (numfiles == 0)
                                image_o[r][c] = 0;
                        else if (image_o[r][c] >= threshold)
                                image_o[r][c] = 255;
                        else
                                image_o[r][c] = 0;

                        pCarInfo->m_Img[r][c] = image_o[r][c];
                }

        pMsg = new MY_MSG(Q_UPDATE_DB, pCarInfo);

        if (Database::Send(Q_UPDATE_DB, pMsg) == false)
        {
                dbgprint("Q_UPDATE_DB failed, file is %s, line is %d.\n", __FILE__,
__LINE__);
                delete pCarInfo;
                delete pMsg;
                Crash(Q_EXIT_FROM_ERROR);
        }

        full_name = after + imagename;
        ofstream imageout (full_name.c_str(), ios::out | ios::binary);

//write bmp header to new file
   for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
      imageout.put(header_info[a]);
//write image data to file
   for(r=NUM_ROWS-1;r>=0;r--)
   {
                for(c=0;c<NUM_COLS;c++)
                        imageout.put(image_o[r][c]);

   //zero padding the end of rows if # of cols not divisible by 4
      for(a=0;a<zero_pad_cols;a++)
         imageout.put('0');
   }

        imageout.close();
        namesangle.close();
}
```

classify.cpp

```
#include<string>
#include<cstring>
#include<iostream>
#include<fstream>
#include<iomanip>
#include<cmath>
#include<math.h>
#include<sstream>

#include "Config.h"
#include "Database.h"

//situation-dependent constants
#define NUM_COLS 128
#define NUM_ROWS 128
#define NUM_GRAY_LEVELS 256

//constants that should not be changed
#define BMP_HEADER_SIZE 54
#define BMP_PALETTE_SIZE 1024
#define PI 3.14159265359

#define USE_DB       1

//global variables
static unsigned char image_i[NUM_ROWS][NUM_COLS];
static unsigned char image_o[NUM_ROWS][NUM_COLS];
static unsigned char header_info[BMP_HEADER_SIZE+BMP_PALETTE_SIZE];

using namespace std;

//static int vehiclenum = 1;
static string vehiclename;
static string targetname;
static string anglename, anglename0, anglename1;
static string vehicleclass;
static int correct[8];    //array containing number of correctly classified test images for
each vehicle type

extern void dbgprint(const char* fmt, ...);
extern void Crash(MSGID nCode);
```

```
extern double find_angle(string full_name);

void ClearSummary()
{
	Database::Send(Q_CLEAR_SUMMARY);
}

void LookupDatabase4Unknown(int nActualCarType, int nHead, int nTail, unsigned
char* pImage)
{
	TargetInfo* pTarget = new TargetInfo;
	pTarget->m_head = nHead;
	pTarget->m_tail = nTail;
	pTarget->m_nActualType = nActualCarType;
	memcpy(pTarget->m_Img, pImage, IMAGE_SZ);

	MY_MSG* pMsg = new MY_MSG(Q_TARGET_LOOKUP, pTarget);

	if (Database::Send(Q_TARGET_LOOKUP, pMsg) == false)
	{
		dbgprint("Fail to submit Q_TARGET_LOOKUP to Database, file is %s,
line is %d.\n", __FILE__, __LINE__);
		delete pTarget;
		delete pMsg;
		Crash(Q_EXIT_FROM_ERROR);
	}
}

void ReportSummary()
{
	Database::Send(Q_REPORT_SUMMARY);
}


int ClassifyTargets_main()
{
	int vehiclenum = 0;
	unsigned char byte;

	//for loop indexes
	int r, c;
	int a;
	int highestmatch = 0;
	int matchedpixels = 0;
	double test_angle, maximum, minimum;
```

```
        int i=0, j=0;
        int     filenum0 = 0, filenum1 = 0;   // These are actual degree in sectors.
        int targetnum = 0;
        int anglenum = 0;

#ifdef USE_DB
        ClearSummary();
#endif

        for (vehiclenum = 0; vehiclenum < 8; vehiclenum++)
        {
                switch(vehiclenum)
                {
                        case 0 : vehiclename = "2S1"; break;
                        case 1 : vehiclename = "BRDM_2"; break;
                        case 2 : vehiclename = "BTR-60"; break;
                        case 3 : vehiclename = "D7"; break;
                        case 4 : vehiclename = "SLICY"; break;
                        case 5 : vehiclename = "T62"; break;
                        case 6 : vehiclename = "ZIL131"; break;
                        case 7 : vehiclename = "ZSU_23_4"; break;
                        default : vehiclename = "";
                }

                //Set place_raw_here folder as input and raw_to_bmp folder as output
                ifstream
filenames(".\\unknown\\"+vehiclename+"\\adaptive_filter_bmp\\filename.txt", ios::in);

                string name,

        before=".\\unknown\\"+vehiclename+"\\adaptive_filter_bmp\\",
                                full_name;

                while(!filenames.eof())
                {
                        name = "";              // Empty string buffer.

                        highestmatch = 0;

                        //handle image file io

                filenames >> name;

                        // In case this line has only a carriage return.
```

```
                        // This could happen while randomly generating unknown and train
images.
                        // See ATREngine::PrepareImage for more information.

                        if (name.size() == 0)
                                continue;       // Skip carriage return.

                        full_name = before + name;

                        //calculate angle of unknown image
                        test_angle = find_angle(full_name);

                        for (i = 0; i < g_nSectorNum/2; i++)
                        {
                                filenum0 = g_nSectorInDegs*i;
                                filenum1 = (g_nSectorInDegs*i)+180;

                                if (i == 0)
                                {
                                        maximum = g_nSectorInDegs/2;
                                        minimum = 180-(g_nSectorInDegs/2);
                                        if (test_angle < maximum || test_angle >=
minimum)
                                        {
                                                anglename0 =
"average"+static_cast<ostringstream*>( &(ostringstream() << filenum0) )->str()+".bmp";
                                                anglename1 =
"average"+static_cast<ostringstream*>( &(ostringstream() << filenum1) )->str()+".bmp";
                                                j = i + g_nSectorNum/2;
                                                break;
                                        }
                                }
                                else
                                {
                                        minimum = maximum;
                                        maximum = minimum+g_nSectorInDegs;
                                        if (test_angle < maximum && test_angle >=
minimum)
                                        {
                                                anglename0 =
"average"+static_cast<ostringstream*>( &(ostringstream() << filenum0) )->str()+".bmp";
                                                anglename1 =
"average"+static_cast<ostringstream*>( &(ostringstream() << filenum1) )->str()+".bmp";
                                                j = i + g_nSectorNum/2;
                                                break;
```

```
                                        }
                                }
                        }

                        //load the unknown image
                        ifstream imagein (full_name.c_str(), ios::in | ios::binary);

                //retrieve header from file
                        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                header_info[a] = imagein.get();

                //retrieve image data from file
                        for(r=NUM_ROWS-1;r>=0;r--)
                                for(c=0;c<NUM_COLS;c++)
                                {
                                        byte = imagein.get();
                                        image_i[r][c] = byte;
                                }

                        imagein.close();

#ifdef USE_DB
                        // i and j represent vehicle orientation and are passed in as the
sector indices.
                        // e.g. If the vehicle orientation are 30 degrees or 210 degrees and
the sector size is 30 degrees; i.e. there are 12 sectors,
                        // i and j will be 1 and 13.
                        LookupDatabase4Unknown(vehiclenum, i, j, (unsigned
char*)image_i);
#else
                        for(targetnum = 0; targetnum < 8; targetnum++)
                        {
                                switch(targetnum)
                                {
                                        case 0 : targetname = "2S1"; break;
                                        case 1 : targetname = "BRDM_2"; break;
                                        case 2 : targetname = "BTR-60"; break;
                                        case 3 : targetname = "D7"; break;
                                        case 4 : targetname = "SLICY"; break;
                                        case 5 : targetname = "T62"; break;
                                        case 6 : targetname = "ZIL131"; break;
                                        case 7 : targetname = "ZSU_23_4"; break;
                                        default : targetname = "";
                                }
```

```
                                for(anglenum = 0; anglenum < 2; anglenum++)
                                {
                                        switch(anglenum)
                                        {
                                                case 0 : anglename = anglename0; break;
                                                case 1 : anglename = anglename1; break;
                                                default : anglename = "";
                                        }

                                        matchedpixels = 0;

                                        //load averaged image
                                        string averagename =
".\\training\\"+targetname+"\\average_bmp\\"+anglename;
                                        ifstream imageout (averagename.c_str(), ios::in |
ios::binary);

                                //retrieve header from file

        for(a=0;a<BMP_HEADER_SIZE+BMP_PALETTE_SIZE;a++)
                                                header_info[a] = imageout.get();

                                //retrieve image data from file
                                        for(r=NUM_ROWS-1;r>=0;r--)
                                                for(c=0;c<NUM_COLS;c++)
                                                {
                                                        byte = imageout.get();
                                                        image_o[r][c] = byte;
                                                }

                                        imageout.close();

                                        //compare individual image with averaged image
                                        for(r=0;r<NUM_ROWS;r++)
                                                for(c=0;c<NUM_COLS;c++)
                                                {
                                                        if (!(image_i[r][c] ^ image_o[r][c]))
        //use XNOR to include matching background pixels
                                                                matchedpixels++;
                                                }

                                        if (matchedpixels > highestmatch)
                                        {
                                                highestmatch = matchedpixels;
                                                vehicleclass = targetname;
```

```
                                        }

                                }
                        }
                        if (vehicleclass == vehiclename)
                                correct[vehiclenum]++;
#endif
                }

                filenames.close();
        }

#ifdef USE_DB
        ReportSummary();
#else
        cout << "Classification Results: " << endl;
        for(i = 0; i < 8; i++)
        {
                switch(i)
                {
                        case 0 : cout << "2S1\t\t"; break;
                        case 1 : cout << "BRDM_2\t\t"; break;
                        case 2 : cout << "BTR-60\t\t"; break;
                        case 3 : cout << "D7\t\t"; break;
                        case 4 : cout << "SLICY\t\t"; break;
                        case 5 : cout << "T62\t\t"; break;
                        case 6 : cout << "ZIL131\t\t"; break;
                        case 7 : cout << "ZSU_23_4\t"; break;
                        default : cout << "\t\t";
                }

                cout << correct[i] << endl;
        }
#endif
        printf("ClassifyTargets_main() complete\n");
        printf("Pattern matching DONE!\n");
        return 0;
}
```

restore.cpp

```
#include<string>
#include<cstring>
```

```
#include<iostream>
#include<fstream>
#include<iomanip>
#include<cmath>
#include<math.h>
#include<sstream>

#include <direct.h>
#include <stdlib.h>
#include <stdio.h>

#include "Config.h"

using namespace std;

int RestoreFolders_main(bool bDelRaw)
{
	//Clear training folders
	if (bDelRaw)
	{
		system("rd /s /q .\\training\\2S1\\place_raw_here");
		system("rd /s /q .\\training\\BRDM_2\\place_raw_here");
		system("rd /s /q .\\training\\BTR-60\\place_raw_here");
		system("rd /s /q .\\training\\D7\\place_raw_here");
		system("rd /s /q .\\training\\SLICY\\place_raw_here");
		system("rd /s /q .\\training\\T62\\place_raw_here");
		system("rd /s /q .\\training\\ZIL131\\place_raw_here");
		system("rd /s /q .\\training\\ZSU_23_4\\place_raw_here");
		system("rd /s /q .\\unknown\\2S1\\place_raw_here");
		system("rd /s /q .\\unknown\\BRDM_2\\place_raw_here");
		system("rd /s /q .\\unknown\\BTR-60\\place_raw_here");
		system("rd /s /q .\\unknown\\D7\\place_raw_here");
		system("rd /s /q .\\unknown\\SLICY\\place_raw_here");
		system("rd /s /q .\\unknown\\T62\\place_raw_here");
		system("rd /s /q .\\unknown\\ZIL131\\place_raw_here");
		system("rd /s /q .\\unknown\\ZSU_23_4\\place_raw_here");
	}

	system("rd /s /q .\\training\\2S1\\adaptive_filter_bmp");
	system("rd /s /q .\\training\\2S1\\average_bmp");
	system("rd /s /q .\\training\\2S1\\raw_to_bmp");
	system("rd /s /q .\\training\\2S1\\temp_files0");
	system("rd /s /q .\\training\\2S1\\temp_files180");
	system("rd /s /q .\\training\\2S1\\threshold_bmp");
```

```
system("rd /s /q .\\training\\BRDM_2\\adaptive_filter_bmp");
system("rd /s /q .\\training\\BRDM_2\\average_bmp");
system("rd /s /q .\\training\\BRDM_2\\raw_to_bmp");
system("rd /s /q .\\training\\BRDM_2\\temp_files0");
system("rd /s /q .\\training\\BRDM_2\\temp_files180");
system("rd /s /q .\\training\\BRDM_2\\threshold_bmp");

system("rd /s /q .\\training\\BTR-60\\adaptive_filter_bmp");
system("rd /s /q .\\training\\BTR-60\\average_bmp");
system("rd /s /q .\\training\\BTR-60\\raw_to_bmp");
system("rd /s /q .\\training\\BTR-60\\temp_files0");
system("rd /s /q .\\training\\BTR-60\\temp_files180");
system("rd /s /q .\\training\\BTR-60\\threshold_bmp");

system("rd /s /q .\\training\\D7\\adaptive_filter_bmp");
system("rd /s /q .\\training\\D7\\average_bmp");
system("rd /s /q .\\training\\D7\\raw_to_bmp");
system("rd /s /q .\\training\\D7\\temp_files0");
system("rd /s /q .\\training\\D7\\temp_files180");
system("rd /s /q .\\training\\D7\\threshold_bmp");

system("rd /s /q .\\training\\SLICY\\adaptive_filter_bmp");
system("rd /s /q .\\training\\SLICY\\average_bmp");
system("rd /s /q .\\training\\SLICY\\raw_to_bmp");
system("rd /s /q .\\training\\SLICY\\temp_files0");
system("rd /s /q .\\training\\SLICY\\temp_files180");
system("rd /s /q .\\training\\SLICY\\threshold_bmp");

system("rd /s /q .\\training\\T62\\adaptive_filter_bmp");
system("rd /s /q .\\training\\T62\\average_bmp");
system("rd /s /q .\\training\\T62\\raw_to_bmp");
system("rd /s /q .\\training\\T62\\temp_files0");
system("rd /s /q .\\training\\T62\\temp_files180");
system("rd /s /q .\\training\\T62\\threshold_bmp");

system("rd /s /q .\\training\\ZIL131\\adaptive_filter_bmp");
system("rd /s /q .\\training\\ZIL131\\average_bmp");
system("rd /s /q .\\training\\ZIL131\\raw_to_bmp");
system("rd /s /q .\\training\\ZIL131\\temp_files0");
system("rd /s /q .\\training\\ZIL131\\temp_files180");
system("rd /s /q .\\training\\ZIL131\\threshold_bmp");

system("rd /s /q .\\training\\ZSU_23_4\\adaptive_filter_bmp");
system("rd /s /q .\\training\\ZSU_23_4\\average_bmp");
system("rd /s /q .\\training\\ZSU_23_4\\raw_to_bmp");
```

```
	system("rd /s /q .\\training\\ZSU_23_4\\temp_files0");
	system("rd /s /q .\\training\\ZSU_23_4\\temp_files180");
	system("rd /s /q .\\training\\ZSU_23_4\\threshold_bmp");

	//Clear testing folders
	system("rd /s /q .\\unknown\\2S1\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\2S1\\raw_to_bmp");
	system("rd /s /q .\\unknown\\2S1\\threshold_bmp");

	system("rd /s /q .\\unknown\\BRDM_2\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\BRDM_2\\raw_to_bmp");
	system("rd /s /q .\\unknown\\BRDM_2\\threshold_bmp");

	system("rd /s /q .\\unknown\\BTR-60\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\BTR-60\\raw_to_bmp");
	system("rd /s /q .\\unknown\\BTR-60\\threshold_bmp");

	system("rd /s /q .\\unknown\\D7\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\D7\\raw_to_bmp");
	system("rd /s /q .\\unknown\\D7\\threshold_bmp");

	system("rd /s /q .\\unknown\\SLICY\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\SLICY\\raw_to_bmp");
	system("rd /s /q .\\unknown\\SLICY\\threshold_bmp");

	system("rd /s /q .\\unknown\\T62\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\T62\\raw_to_bmp");
	system("rd /s /q .\\unknown\\T62\\threshold_bmp");

	system("rd /s /q .\\unknown\\ZIL131\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\ZIL131\\raw_to_bmp");
	system("rd /s /q .\\unknown\\ZIL131\\threshold_bmp");

	system("rd /s /q .\\unknown\\ZSU_23_4\\adaptive_filter_bmp");
	system("rd /s /q .\\unknown\\ZSU_23_4\\raw_to_bmp");
	system("rd /s /q .\\unknown\\ZSU_23_4\\threshold_bmp");

	printf("RestoreFolders_main() complete\n");

	return 0;
}
```