Copy No. _____

Defence Research and Development Canada | Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Structured Software Implementation of Computer Detection Algorithms for Sidescan Sonar Data

Final Report

Jason McInnis and George Ryan
Akoostix Inc.

Akoostix Inc.
10 Akerley Blvd, Suite 12
Dartmouth, NS B3B 1J4

Contract Project Manager: Derek Burnett, 902-404-7464

Contract Number: W7707-088118/001/HAL

Contract Scientific Authority: John Fawcett, 902-426-3100 ext 324

Defence R&D Canada – Atlantic

Contract Report

DRDC Atlantic CR 2009-046

May 2009

Canada

Structured Software Implementation of Computer Detection Algorithms for Sidescan Sonar Data

Final Report

Jason McInnis
George Ryan
Akoostix Inc.

Akoostix Inc.
10 Akerley Blvd, Suite 12
Dartmouth, NS B3B 1J4

Contract Project Manager: Derek Burnett, 902-404-7464
Contract Number: W7707-088118/001/HAL
CSA: John Fawcett, 902-426-3100 ext. 324

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2009-046
May 2009

Principal Author

Original signed by Jason McInnis

Jason McInnis

Lead Systems Engineer

Approved by

Original signed by John Fawcett

John Fawcett

Scientific Authority

Approved for release by

Original signed by Ron Kuwahara for

Calvin Hyatt

Head/Document Review Panel

Abstract

This report summarizes the work performed for the Structured Software of Computer Detection Algorithms for Sidescan Sonar project for Defence Research and Development Canada (DRDC) - Atlantic. This contract developed an initial test bed for mine-hunting Automated Target Detection (ATD) algorithms to run using batch processing on sidescan sonar data such as the Klein 5500. The first version of the software developed under this contract achieved its goal through various methods including incorporation of DRDC mine detection algorithms, through the reuse and upgrade of existing DRDC libraries, and through the development of new code. The resulting software is faster than real-time, cross-platform and easy to extend through a dynamic run-time plugin interface.

Résumé

Le présent rapport résume le travail effectué dans le cadre du projet de logiciel structuré pour les algorithmes de détection informatique de cibles à partir des données de sonar à balayage latéral de Recherche et développement pour la défense (RDDC) - Atlantique. Le contrat visait à mettre au point un premier banc d'essai pour les algorithmes de détection automatique des cibles destinés à la chasse aux mines; ce banc d'essai permet de traiter par lot les données provenant d'un sonar à balayage latéral comme le Klein 5500. La première version du logiciel mis au point dans le cadre du contrat a atteint son objectif grâce à diverses méthodes, notamment l'intégration d'algorithmes de détection des mines de RDDC, la réutilisation et la mise à niveau de bibliothèques existantes de RDDC, et le développement de nouveau code. Le logiciel obtenu est multiplateforme et permet de traiter les données plus rapidement qu'en temps réel. De plus, il est facile de l'étendre au moyen d'une interface de chargement de modules à l'exécution.

This page intentionally left blank.

Executive summary

Structured Software Implementation of Computer Detection Algorithms for Sidescan Sonar Data: Final Report

Jason McInnis; George Ryan; DRDC Atlantic CR 2009-046; Defence R&D Canada – Atlantic; May 2009.

Introduction: Over the last several years, DRDC Atlantic has been involved with the development of automated target detection/classification algorithms. Many of these algorithms have been written in MATLAB and have been used as research tools in post-processing. The goal of this present contract was to develop a testbed structure in C++ which would allow for the structured development and testing of automated detection and association methods for sidescan sonar data obtained in trials.

Results: A robust and efficient structure has been developed for the reading of sonar data, the automated detection of targets, and the fusion of detections from different algorithms and sonar passes. Standardized interfaces for automated detection and fusion algorithms were developed. The use of multi-threading for multiple algorithms was addressed.

Significance: The structure will allow for the simple addition of other detection and fusion algorithms in the future. These algorithms may be “plugins” developed by other nations.

Future plans: It is hoped that mode modules will be added to the structure to allow for the input of more sonar data types. Additional detection methods will be added and simple classification methods considered. This structure could also provide the basis for collaboration with other nations.

Sommaire

Structured Software Implementation of Computer Detection Algorithms for Sidescan Sonar Data: Final Report

Jason McInnis; George Ryan; DRDC Atlantic CR 2009-046; R & D pour la défense Canada – Atlantique; Mai 2009.

Introduction : Au cours des dernières années, RDDC Atlantique a participé à la mise au point d'algorithmes de détection ou de classification automatiques des cibles. Bon nombre de ces algorithmes ont été écrits dans MATLAB et utilisés comme outils de recherche en posttraitement. Le présent contrat visait à mettre ou point une structure de banc d'essai en C++ qui permettrait de procéder au développement et à la validation structurés des méthodes automatiques de détection et d'association des cibles à partir des données obtenues au cours des essais de sonar à balayage latéral.

Résultats : On a mis au point une structure robuste et efficiente permettant de lire les données de sonar, de détecter automatiquement les cibles, et de fusionner les résultats de détection provenant de différents algorithmes et de plusieurs balayages de sonar. On a également développé des interfaces normalisées pour les algorithmes de détection et de fusion automatiques. L'utilisation de fils multiples pour l'exécution de différents algorithmes a été rendue possible.

Portée : À l'avenir, la structure permettra d'ajouter facilement d'autres algorithmes de détection et de fusion au logiciel; il pourrait s'agir de « plugiciels » développés par d'autres pays.

Recherches futures : Nous espérons que d'autres modules seront ajoutés à la structure afin de permettre de traiter de nouveaux types de données sonar. D'autres méthodes de détection seront ajoutées et des méthodes simples de classification seront étudiées. La structure pourrait également devenir la base d'une collaboration avec d'autres pays.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1 Introduction

This is the final contractor report for the Structured Software of Computer Detection Algorithms for Sidescan Sonar project for Defence Research and Development Canada (DRDC) - Atlantic, contract number W7707-088118/001/HAL. This report summarizes the work performed during this contract and provides details not provided in either the DOyxgen software documentation or other README files provided with the software source deliverable. Where appropriate, the software documentation will be referenced to avoid repetition. This documentation will focus on how effort was spent during the contract and any design information that is required to explain how the work was executed.

The objective of this contract was to develop an initial test bed for mine-hunting, Automated Target Detection (ATD) algorithms in a batch processing manner on sidescan sonar data such as the Klein 5500. The first version of the software developed under this contract achieved its goal through various methods including input from DRDC on algorithms (including example code), through the reuse and upgrade of existing DRDC libraries, and through the development of new code. Some of the new code that was generic to the domain was contributed to the DRDC reusable code base by either being included in an existing library or becoming its own independent library. The detection algorithms of this report had been previously described in other reports and papers [1-4].

1.1 Project Overview

The contract was awarded on October 21st, 2008. A kick-off meeting was held October 22nd, 2008. The project team began to develop the functional requirements of the desired system during the kick-off meeting and during a series of subsequent requirements analysis meetings over a two week period. The project requirements were distilled into two categories that can be summarized as follows:

1. DRDC desired a cross-platform test bed for mine-hunting algorithms that work on individual sonar files in a batch processing manner. This vision was not immediately apparent to Akoostix during the development of the proposal. The project was refocused on developing dynamically loadable algorithm modules on multiple platforms without introducing too many third party libraries.

2. Algorithm development was important but not as important as the plugin architecture. Despite this prioritization it was crucial to develop correctly operating algorithms but their actual detection performance was not a factor in the development. DRDC provided the algorithm in MATLAB code but in the end it was a combination of requirements analysis and examination of the MATLAB code that was used to develop the software and not a simple porting process. This resulted in more concise and efficient C++ code.

Akoostix expressed its concern about the effect on scope that certain Standard Acquisition Clauses and Conditions (SACC) would have on the project. In particular the rigorous testing and documentation process was more fitting of production software than a research software project. DRDC subsequently removed this clause from the contract. These tests consisted of an end-to-end synthetic and real data test demonstrated to DRDC at various times throughout the project. These tests, combined with the numerous built-in unit tests developed for key software modules, provided sufficient validation that the software was working correctly with the expected quality of functionality.

During the requirements phase it became clear that the current scope of the contract may be ambitious. There were two contributing factors that caused this concern:

1. The algorithms could not be generated using a simple porting process. There were enough changes warranted in the original MATLAB code that required investigation.

2. The plugin architecture was to support independent development of a module without the need for recompiling.

3. Concurrency and performance was to be considered to take advantage of multiple processors and to reduce batch processing times.

DRDC and Akoostix reprioritized the requirements after their impact on the design was fully explored. Akoostix re-estimated the project based on an updated design that considered the revised project vision. Akoostix reviewed the results of the re-estimation with DRDC on December 1st, 2008. The re-estimation showed that it would be only possible to implement one, possibly two, ATD algorithms and only one sonar reader. This allowed the plug in architecture to be developed as desired along with the inclusion of concurrent processing capability.

The initial version of the software was demonstrated to DRDC on December 18th, 2008. During this demonstrations the Klein Sonar Data Format (SDF) file reader, initial Match Filter algorithm and the Tagged Image File Format (TIFF) reading and writing software were reviewed including a design and code walkthrough.

During January and February of 2009 the development team focused its attention on the ATD control application code, specifically, the deadline for completing Tasks 1 through 6 of the Statement of Work (SOW). This effort included development of the control software, integration of the sonar readers and match filter algorithm as well as the initial framework for handing concurrency. This software was demonstrated to DRDC on January 26th, 2009. Several bugs were detected in the Match Filter during this demonstration and were subsequently fixed. By the end of that week, the Match Filter algorithm was detecting on synthetic data and real data with expected configuration parameters and thresholds. Testing and integration effort was moved up in the schedule to perform this test as well as to ensure that the software executed on Windows XP prior to field trials being attended by Warren Connors.

The last phase of the project included developing the mugshot capability, completing the concurrency framework and integrating the association and fusion stages. This software was demonstrated to DRDC on February 18th, 2009. During this demonstration the project team verified that real detections and their positions were being generated correctly. The Klein viewer was used to independently verify the results of the processing during this demonstration. There was some discrepancy over how mugshots were generated. This was rectified and described in more detail in Section 3.7.

2 Operator and Developer Instructions

The software is documented in several places within the source distribution. Any software documentation has been excluded from this report in order to decrease duplication of information. These sources are referenced as necessary throughout this report. They include:

- A *README* text file containing build instructions, installation instructions and version information is located at the root of the source directory.

- DOyxgen comments which are compiled into the Application Programming Interface (API) documentation set. See the README for instructions on building this documentation. The DOxygen describes the relevant class interfaces and their parameters. It also contains a design overview and a section on developing your own ATD plugin. The DOxygen documentation is located in the *doc/release/html* directory once it has been built.

- Configuration file templates are located in the *cfg/* directory of the source distribution. These configuration files are self documenting and provide all possible key-value pairs for the control software, both stages of the association algorithm and both ATD algorithms. They can be used to get started on setting up your own installation of the ATD software.

- Source code templates were developed that explain the steps required to develop a new ATD or association algorithm to be used with the control software. They are located in the *doc/* directory of the source tree.

3 Work Performed

The following section describes the work performed against each of the tasks specified in the SOW.

3.1 Task 1 – Requirements Review

Through a series of several meetings and e-mail exchanges, the requirements were defined in enough detail so that the project team could proceed with a clear picture of the required functionality of the system. Table 1 was created and published on the project portal. Included in this table is all the information required to understand how the algorithms work and the theory of operation for the control software. This information will not be repeated in Sections 3.3 thru 3.9 when describing the work performed for their respective SOW tasks.

A significant amount of effort was spent defining the requirements for this project. By the end of the requirements task the project team had a clear vision on what was required of the ATD software. Although there were several minor revisions to the requirements after the requirements review task was complete, the project team had a clear vision early in the project which helped keep development focused and efficient. Akoostix would recommend taking this approach to all ATD software development projects of this size in the future.

During the requirements task a performance test was developed and bench marked. The *mf_loading_test.cpp* found in the *src/ssmm/test* directory of the source distribution was created to mimic the processing steps and memory management issues that may occur in that of the matched filter processing. Although the results of the test do not directly map to that of the actual matched filter, it did reveal that there could be significant performance issues when running many complex detection algorithms in sequence without considering the sharing of sonar data objects or distribution of load for systems that have multiple processors. This investigation prompted the change in scope for handling concurrent execution of algorithms.

Table 1: Final Version of the Technical Requirements

1	Status	Technical Requirements
1.1	Complete	The software shall provide the ability to read in Klein 5500 format sidescan sonar data files.
1.1.1	Complete	Beams overlap in the Klein sonar data. The format provides this information and can locate redundant beam information. This information shall be ignored or removed in the data stream prior to processing.
1.1.2	Complete	Ping data from the Klein 5500 sidescan sonar data should orient the data so that the largest visible beam number is at the beginning of the data.
1.2	Deferred	The software shall provide the ability to read in eXtended Triton Format (XTF) sidescan sonar data files

1.2.1	Deferred	Only a portion of the XTF information is required. The United States Geological Survey (USGS) parsing information can be used as is.
1.3	Deferred	The software shall provide the ability to read in Marine Sonics format sidescan sonar data files.
1.4	Complete	All formats of sidescan sonar data shall be represented internally by a standard sidescan sonar data object.
1.4.1	Complete	The sidescan sonar data shall contain an index of relevant telemetry data including: • altitude in meters • cable out in meters • configuration (enumerated type) • date/time (year, month, day, hours, minutes, seconds, fractional seconds) • depth in meters • despeckle switch (on/off) • error flags (enumerated type) • towfish position (latitude/longitude in degrees) • GPS height in meters • towfish heading in degrees • magnetic variation in degrees • manual speed switch (on/off) • number of visible beams • number of samples • ping number • pitch in degrees • range in meters • resolution (along track and across track in meters) • roll in degrees • sampling frequency in Hertz • sheave x,y and z offsets in meters • ships heading in degrees • ships position (latitude/longitude in degrees) • ships speed in meters/second • towfish speed in meters/second • speed of sound in meters/second • tvg page • transmit waveform (enumerated type)
1.5	Complete	The software shall provide the ability to convert the sidescan sonar data objects into TIFF.
1.5.1	Complete	The software shall provide the ability to convert the TIFF into a sidescan sonar data object.
1.6	Complete	The software shall provide the ability to write out TIFF files to the file system.

1.6.1 Complete The software shall provide the ability to read in TIFF files to the file system.

1.7 Complete The software shall associate any related telemetry data to the TIFF data using an associated human readable file.

1.8 Complete The software shall provide a standard interface for mine detection algorithms, allowing for custom extension by using a run-time plugin mechanism.

1.8.1 Complete The standard interface for the mine detection algorithms will not require processing parameters. Each pluggable algorithm will use a configuration file to set/modify processing parameters.

1.9 Complete The standard interface for the mine detection algorithm shall use sidescan sonar data object as input.

1.10 Complete The standard interface for the mine detection algorithm shall allow for the production of one or more sets of detection target data.

1.10.1 Complete The target detection data shall consist of:

- Position (latitude/longitude in decimal degrees).
    - The position of the max value for a raw detection.
    - The geometric mean of the positions from each fused detection.
- Max value for a raw detection is user defined (i.e. for matched filter this will be the max value for the normalized data). Meaningless to a fused detection.
- Confidence Level (default value required).
    - For a raw detection P = 1.0 - exp(- 0.692 * max value / probability reference) where the probability reference is a user configurable value.
    - For a fused detection this is the P resulting from the probability fusion step noted in section 1.18.2.
- Reference to Detection Data (input telemetry and image data):
  For a raw detection this points to a file, ping and beam number
  For a fused detection this is a list of children that produced the fusion (which in turn contains the file, ping and beam number information)
- Identifier for the detector or fuser used to produce the detection.
- A Boolean flag denoting whether the detection is valid or a potential false detection.

1.11 Changed to 1.11.4 The software shall allow a mugshot of TIFF data to be extracted from the sidescan sonar data given detection target data information.

1.11.1 Complete Mugshots of the TIFF data shall be configurable with a default of 50 pixels along track by 500 pixels across track.

1.11.2	Changed to 1.11.4	The mugshot may use an intensity scale that the mugshot can be viewed by an image viewer that cannot adjust image intensities.
1.11.3	Changed to 1.11.4	The mugshot may maintain enough intensity scale conversion information to be able to be converted back to its unmodified raw values within the selected quantization noise.
1.11.4	Complete	The mugshot shall be produced from the normalized data from the automated target detection algorithm. The normalized data consists of the original data after it has been divided by the mean across-track curve. The scaling is produced by sorting the data values and computing the value of the 95% percentile (call it *rm*) and then this value and anything higher becomes 255 of an 8 bit unsigned pixel value. The rest are scaled by 255/*rm*. The percentile shall be a configurable value.
1.12	Complete	The software shall provide a match filter mine detection algorithm, ported from an existing algorithm supplied by the Project Authority (PA) with the following exceptions: • Code to read from the sonar should be factored out. • A sonar oblivious data reader will abstract the data sources, providing a uniform data interface to the algorithm(s). The data reader will handle clipping the data to min-max range. Also, the images (port or starboard) will always be presented to the algorithm in a contiguous chunk, beginning at the closest range, and oldest ping. • Processing parameters should be made configurable using a configuration file noted in 1.8.1. • The convolution kernel shall be dynamic, growing in size to account for range. The current algorithm uses static kernels with three different sizes depending on where in the data it is convolving. This implementation will use a kernel specific to each range, growing by the smallest unit (a pixel) at a time. These will be pre-calculated. • Ignore the decimation step. The algorithm shall perform the following steps, note that the starboard side is processed as a separate chunk as the port side in the data: • Normalize the entire data by summing across columns, and dividing through by the average. • Compute the median of the normalized image in step 1. Call this value u. • Re-bin the data into 5 basic values: o roughly dark shadow (-1), if value is < 0.6u o shadow(-0.5), if value is < 0.3u o background (0), if value is < 2u o medium highlight (+0.5), if value is < 4u o bright highlight (+1), if value is >= 4u • Perform a 2d correlation with a "signal" kernel, which is two levels (object/shadow) and a highlight point (the center of the object region). The image will be padded before correlation to

ensure the output image S is the same size as the input image.

- Perform another 2d correlation with a "background" kernel which is a single level and a highlight point. Again pad the image but, for the background estimate the absolute value is taken before correlation. (output B)
- Create an image which is S / (1 + 4B).
- Threshold the resulting image, +1 for values over the threshold, else 0.
- Perform a "smearing" 2d correlation over the image. The kernel will be 2d consisting of a single value, and the image will be pre-padded to end up with output image of the same dimensions as the input image.
- Detect and label each region by grouping adjacent values of a group. The center point of the region in the maximum value in the non-thresholded data.
  Convert the information into a detection for each region.

1.13 Deferred — The software shall provide a statistical Z-test mine detection algorithm, ported from an existing algorithm supplied by the PA.

1.14 Complete — The software shall provide a Speckle mine detection algorithm, ported from an existing algorithm supplied by the PA. Exceptions are the same as for 1.12.
The algorithm shall perform the following steps, note that the starboard side is processed as a separate chunk as the port side in the data. Also note similarities with 1.12:

- Normalize the entire data by summing across columns, and dividing through by the average.
- Compute the local lacunarity (the ratio of variance to mean) over the image using the same size kernels as for the matched filter. Each cell in the kernel will have a value of 1/kernel_size. The edges of the image will be padded with a user-defined value (default = 1.0).
- Convolve the kernel over the normalized data (computing the local mean) and square the result [call this result A].
- Square the normalized data and convolve the kernel over the result [call this result B].
- Compute lacunarity L = B/A – 1.0
- Perform another 2d correlation with a "background" kernel which is a single level and a highlight point. Again pad the image, for the background estimate the absolute value is taken before correlation.
- Create an image which is L / (1 + 4B).
- Threshold the resulting image, +1 for values over the threshold, else 0.
  Perform a "smearing" 2d correlation over the image. The kernel will be 2d consisting of a single value, and the image will be pre-padded to end up with output image of the same dimensions as the input image.
- Detect and label each region by grouping adjacent values of a group. The center point of the region in the maximum value in the unthresholded data.

		Convert the information into a detection for each region.
1.15	Complete	The software shall provide the ability to write out detection target data to the file system.
1.15.1	Complete	The detection target data will be associated with the telemetry data of the detection region within the same human readable file.
1.15.2	Complete, Mugshots generated at all stages.	The detection algorithms shall produce just the detection information (not mugshots required at this point) into a detections directory. The detections will be read from disk, into memory, and passed to the first association/fusion stage. Raw input detection will be associated with a synthetic detection by copying the raw detection information with it to a new detection file. The synthetic detection will be flagged as either having passed all thresholds and confidence tests or not. (pass/fail flag). Mugshots of the synthetic detections will be generated and for all failed detections in the synthetic output directory for Stage 1. Second pass fusion step will use the information from the first stage and put the results in a second synthetic output directory, including mugshots.
1.16	Complete	The software shall provide a standard interface to target data association functionality, allowing for custom extension by specialization/inheritance.
1.17	Complete	The software shall provide a standard interface to target data fusion functionality, allowing for custom extension by specialization/inheritance.
1.18	Complete	The software shall provide target data association and decision fusion functionality based on a supplied algorithm(s) from the PA.
1.18.1	Complete	Data association will be performed using the following steps: • The user will define a max range for association. • A 'reference' detector will be used for range calculations. The reference detector will be picked as the detector that has the most sub-detections for the pass (in a file, or a set). • Any detections from other detectors will be associated to a reference detection if it falls within the max range (though multiple detections do not also need to be within max range of each other). Once associated, a detection is removed from further consideration. After a pass all remaining reference detections will be considered single detections (there shouldn't be any other detections within max_range of them). • If a pass is completed and unassociated detections remain from non-reference detectors, the one with the most unassociated detections will be picked as the reference detector and iterations will continue. At most N-1 passes should be required if N is the number of detectors that were used. • Two detections from the same detector cannot be associated unless they are from separate "looks" of the data (another

		pass).
1.18.2	Complete	Two fusion tests will be made and both must pass for a detection (or fused detection) to be written out. If it doesn't pass, no results are produced, but the input detections remain intact: • M of N Fusion: The user will set a threshold integer (min value = 1, default value = 1). If set to 1 then all detections will pass. The number of associated detections must exceed the threshold to pass. For an input detection into the second stage fusion the number of 'child' detections will not be a factor. • Probability fusion: The user will set a threshold probability (min_value = 0.0, default value = 0.5). If set to 0.0 then all detections will pass. The combined probability will be computed iteratively using a pair of detections (arbitrary order) P = P1 + P2 (1 - P1); Iterating over P2 = all input probabilities. If only one detection exists the output probability is the same as its probability. • By setting M of N threshold = 1 and probability threshold=0.0 then effectively only association is performed, though the output detections still needs to have its probability updated and a list of associated detections added.
1.19	Complete	The software shall provide an overall, automated target detection object that shall provide a standard method of configuring one or more detectors to run on one or more sets of sidescan sonar data.
1.19.1	Complete, plugins existing in plugin directory are initialized which a configuration file of the same name.	Selection of plugins shall be performed using a configuration file. The configuration file will consist of one or more sets of the following information. Each set represents a different instance of a plugin that will run concurrently to generate the entire detection set: • Plugin name • Plugin location • Plugin configuration file
1.20	Complete	The automated target detection object shall collect and return the target detection data from its configured algorithms.
1.21	Complete	The automated target detection object shall produce the target mugshot TIFFs from the target detection data produced after the decision fusion processing, if any.
1.21.1	Complete	If multiple looks of the data are available, each look mugshot TIFF shall be produced and associated with the telemetry and target detection data.
1.22	Complete	The automated target detection object shall allow for the addition of one or more optional association and fuse algorithms to be applied from the output of its configured algorithms.
1.23	Complete	The automated target detection object shall provide output in-memory and to a configurable output directory.

1.24	Complete	The software shall provide the ability to read in detection target data to internal memory.
1.25	Complete	The automated target detection object shall run in three modes: • Detection and fusion • Detection only • Fusion only
1.25.1	Complete	Depending on which mode it is in, the detection data shall be read from the target detection directory, or in-memory.
1.26	Complete	Multiple automated target detection algorithms shall run concurrently in order to maximize performance based on the number of available processors.

3.2 Task 2 – Automated Target Detector Software Design

The design of the Automated Target Detector software was broken into four major sections of effort:

1. The run-time plugin architecture for the detection algorithms (requirements 1.8, 1.8.1, 1.9, 1.10, 1.10.1, and 1.22).
2. The main Automated Target Detector application (requirements 1.19, 1.19.1, 1.23, 1.25, and 1.25.1).
3. The Automated Target Detector's concurrent detection algorithm memory model (requirement 1.26).
4. The compile-time (static) association/fusion plugin architecture (requirements 1.16 and 1.17).

The run-time plugin architecture for the detection algorithms was designed early in the project to reduce the risk involved in creating a workable cross-platform run-time plugin system. A small prototype was implemented in order to verify that the design of the plugin loader and the memory management of the plugin objects were operable across the Windows and Fedora Core Linux platforms.

The dynamic plugin manager was developed to work in a four-step process: it loads a detection algorithm plugin, it then loads any accompanying configuration file for the plugin, it creates a concrete instance of the detection algorithm from the plugin, and then it hands the configuration to the algorithm for the algorithm to use. This is represented in the diagram (Figure 1) below.

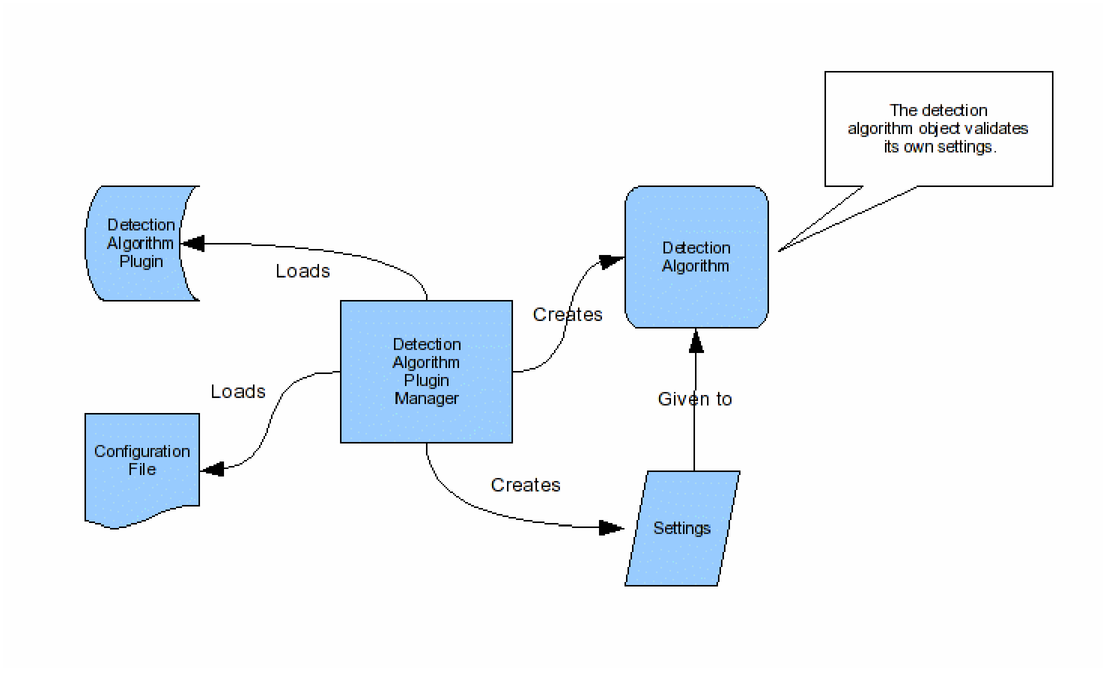
The detection algorithm object validates its own settings.
Detection Algorithm Plugin
Loads
Detection Algorithm
Creates
Detection Algorithm Plugin Manager
Loads
Given to
Configuration File
Creates
Settings

Figure 1: The detection algorithm plugin architecture

The design of the main ATD application involved managing the operations parsing command-line arguments, reading in configuration files, storing persistent parameters, using the sonar data reader(s) to load in sonar data files, and caching the sonar data in memory for use by the detection algorithms. The sonar data cache is also responsible for tracking in-use chunks of data and disposing of them when all of the detection algorithms have completed using them. It is also the responsibility of the ATD application to prompt the plugin manager to load in and create the detection algorithms from the plugin files.

The design of the sonar data cache (summarized in Figure 2) within the ATD application required special attention in order to avoid data anomalies and deadlocking due to having detection algorithms running concurrently on the same chunks of data. To keep things simple, a coarse, high-level system of locking the cache was designed. As data is provided to threaded instances of detection algorithms, the cache protects the data from subsequent algorithm call-backs stating they are prepared to receive new data. No data is accidentally removed before all algorithms are finished with it. The sequence diagram below demonstrates the results of the design explicitly.

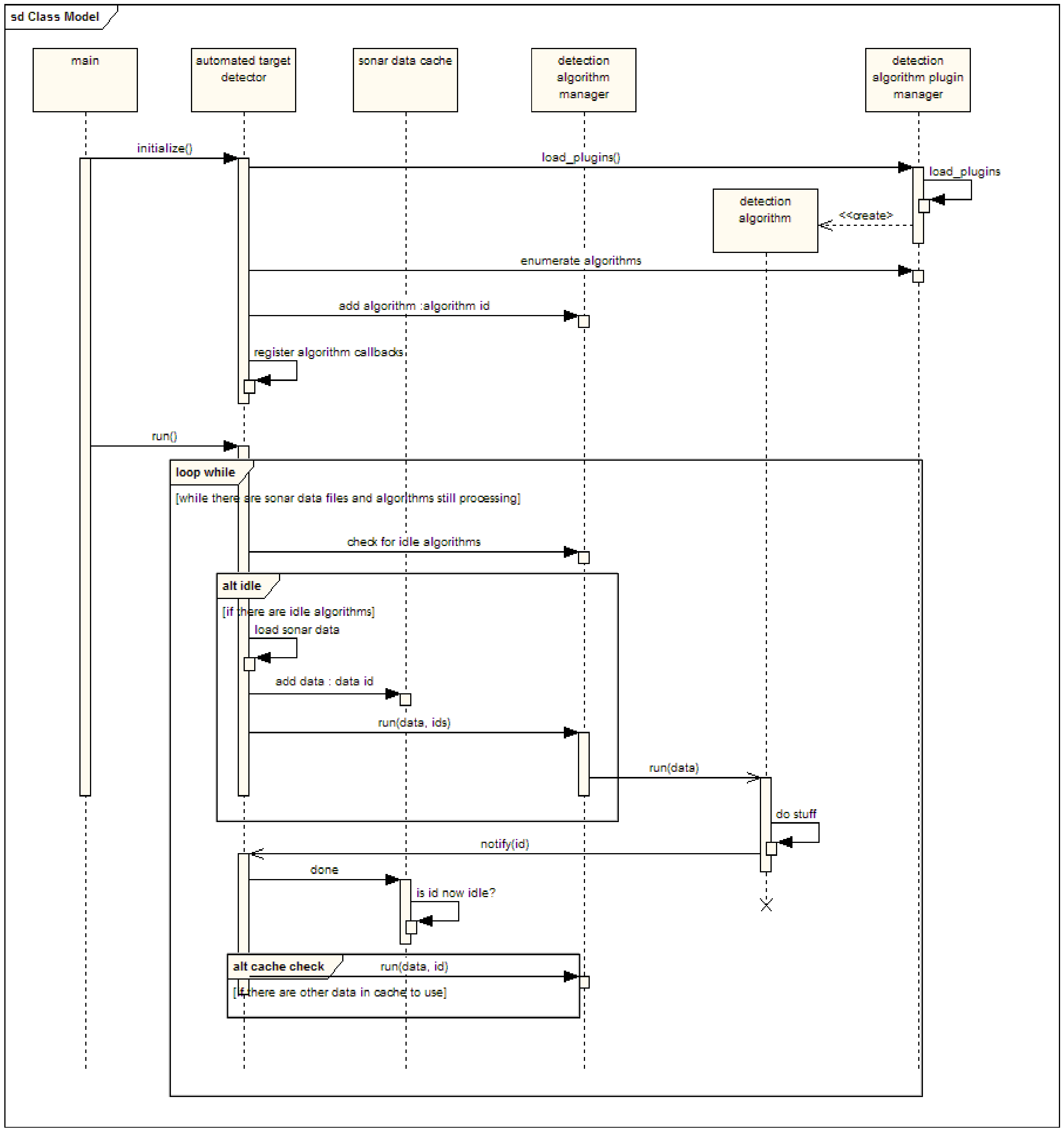
sd Class Model
main
automated target detector
sonar data cache
detection algorithm manager
detection algorithm plugin manager
initialize()
load_plugins()
load_plugins
detection algorithm
<<create>>
enumerate algorithms
add algorithm :algorithm id
register algorithm callbacks
run()
loop while
[while there are sonar data files and algorithms still processing]
check for idle algorithms
alt idle
[if there are idle algorithms]
load sonar data
add data : data id
run(data, ids)
run(data)
do stuff
notify(id)
done
is id now idle?
alt cache check
run(data, id)
[if there are other data in cache to use]

Figure 2: The concurrency model for the data cache

The design of the association/fusion components was very straight-forward. Since they are compile-time plugins, a standardized base class was designed which detailed the data interaction between the ATD and the algorithm. Also, the ATD was designed to take a user-supplied association/fusion algorithm for either the first or the second stage of association.

3.3 Task 3 – Development of Sonar File Reader

The Sonar File Reader task was originally supposed to implement three different sonar file readers. This task was reprioritized with the Klein format file reader being implemented first and the remaining two being deferred until it was determined if there was enough budget remaining to complete them. In the end, it was not possible to develop the eXtended Triton Format (XTF) and Marinesonics sonar file reader. This task developed software modules that addressed the following requirements:

1. Read in the Klein sonar data (1.1, 1.1.1, 1.1.2).
2. Transfer the data into a sonar data object (1.4, 1.4.1).
3. Convert the sonar data object to/from a TIFF object (1.5, 1.5.1).
4. Read/write the TIFF object to file system (1.6, 1.6.1).
5. Associating the telemetry data with the image data (1.7).

The reading portion of the Klein sonar data was modified from code provided by DRDC with a few differences in order to make the resulting sonar data object more processing friendly.

- The sonar data reader was implemented to divide the port and starboard side into two separate objects. The sonar data object provides an enumerated type to let the developer know what orientation the data object is if required.
- The file reader starts a new sonar data object if the resolution mode of the sonar changes. The sonar data object is guaranteed to be of the same resolution to reduce complexity in the automated target detection algorithms.
- The data is modelled in memory as a 2 dimensional array with the left hand side always being closest to the sonar and the bottom being the most recent data.
- Sonar data object is stored as floating point values and is ready for processing. The magnitude of the 16-bit value is casted to a floating point value.

The TIFF object was implemented as a C++ template buffer object. It provides the minimum required interface for correctly streaming the sonar data object to disk as a TIFF including:

- Compression type = pack bits
- Photometric interpretation = minimum is black
- Planar value = contiguous
- Resolution Units = no units

- Orientation = top left for port and top right for starboard
- Bits per pixel = 8
- Maximum value = 255 for 8-bit viewable or 65535 for 16-bit original sonar data value
- Minimum value = 0

It also provides an interface for encoding image strips from a pair of Standard Template Library (STL) compliant iterators (which, for the mine hunting modules is provided by the sonar data object). This class and its corresponding reader and writer were developed as part of an independent *graphics* library and made a dependency of the ATD software.

The telemetry information was captured in two different structures of the sonar data class. The first structure, *sonar_data::telemetry* contains all sonar independent data including:

- Altitude in meters
- Cable out in meters
- Date/time
- Depth in meters
- Towfish heading longitude, latitude in degrees
- Towfish speed in meters per second
- GPS height in meters
- Magnetic variation in degrees
- Ping number
- Visible beams count
- Pitch and Roll in degrees
- Sheave x,y and z offset in meters
- Ships heading, longitude and latitude in degrees
- Ships speed in meters per second
- Measured speed of sound in meters per second

The second structure, *sonar_data::configuration* consists of Klein specific flags read from the sonar. It is a separate structure because it is for informational purposes only and is used to carry this information into the telemetry file. There is no interface in the sonar data object to access this information to prevent Klein specific processing from being developed. There may be a better way to insulate this data in the future. The configuration structure consists of:

- Configuration settings flag
- Error flags
- TVG setting

- Transient Waveform
- Despeckle flag
- Manual speed switch flag
- Sampling frequency in Hertz

The telemetry file writer produces an American Standard Code for Information Interchange (ASCII) file with the information from the telemetry and configuration structures of the sonar data object along with some state held by the sonar data object itself including:

- Data orientation including port, starboard or composite.
- Number of rows in the image.
- Number of samples per row in the image.
- The minimum and maximum range of the sonar data. This may have come directly from the sonar or have been modified as a result of creating a sub-sonar object from an original sonar data object.
- The resolution along track and across track.
- The file name/data source that produces the sonar data object.

Note that the state information held by the sonar data object never changes while a list of telemetry and configuration structures will be held for each corresponding ping in the sonar data.

This information is produced by the telemetry file writer software in the following format (each item is on its own line in the file):

Table 2: Telemetry File Format

Item	Value	Description
1	1_0	The first number is the major version number of the telemetry file format and second number is the minor version number. This scheme is used for the detection files as well, with the intent that a minor version number change would be backward compatible with previous versions of the reader software while a major version would not.
2	orientation;number of rows;samples per row;min range (m);max range (m);across track res (m);along track res (m);data source;telemetry source;	The header explaining each column of the next line which consists of the sonar state. This information does not change during the sonar data objects data coverage. Note that the *(m)* means that the corresponding value

		in Item 3 is in meters.
3	*string("starboard", "port" or "composite"); unsigned int; unsigned int; float; float; float; float; string; string;*	The actual values of the sonar data object state in order noted by the header in Item 2.
4	ping number;date/time;altitude (m);roll (deg);pitch (deg);depth (+m);speed of sound (m/s);visible beams;magnetic variation (+/- deg);towfish heading (deg);towfish latitude (deg);towfish longitude (deg);towfish speed (m/s);ships heading (deg);ships latitude (deg);ships longitude (deg);ships speed (m/s);cable out (m);gps height (m);sheive x offset (m);sheive y offset (m);sheive z offset (m);configuration settings;error flags;tvg settings;transient waveform;despeckle ;manual speed switch;sampling frequency (Hz);	The header explaining each column of the subsequent lines until end of file. These lines will contain the telemetry data and the configuration item for each ping in the sonar data object. Note that (m) = meters, (deg) = floating point degrees, (m/s) = meters per second and (Hz) = Hertz.
5 to N Pings + 5	*unsigned int; YYYY-MON-DD HH:MM:SS.FFF; float; float; float; float; unsigned int; float; float; float; float; float; float; float; float; float; float; float; float; float; float; unsigned int; unsigned int; unsigned int; unsigned int; Boolean; Boolean; float*	The actual values of the telemetry and configuration structures a line per ping.

The *sonar_convert_test.cpp* test file located in the src\ssmm\test directory was developed to test all the requirements implemented under this task. The test performs the following tasks:

1. Searches a directory called *src\ssmm\test\data* for any files with the extension *.sdf* or *.5kd* and attempts to read them using the SDF file reader. The SDF file reader generates the appropriate sonar data objects.

2. The sonar data object is converted to two TIFF objects. One is a 16-bit TIFF that preserves the actual values of the data and the other is an 8 bit that is bit shifted in order to make it viewable.

3. Both objects are written to the *src\ssmm\test\data* directory along with there corresponding telemetry files.

4. The 16-bit TIFF and corresponding telemetry file is written back in memory from disk and converted to the original sonar data object.

3.4 Task 4 – Development of Automated Target Control Software

The development of the Automated Target Detector control software (requirements 1.16, 1.17, 1.19, 1.19.1, 1.23, 1.25, 1.25.1, 1.26, 1.8, 1.8.1, 1.9, 1.10, 1.10.1, and 1.22) involved a straight-forward realization of the design from Task 2 (Section 3.2).

The main program's argument parsing functionality was developed, as there are a few options that make sense to be able to control from the command line rather than from a configuration file (e.g., the ATD's own configuration file location, the run mode, etc.). To support passing the application's command flags, an existing DRDC command-line parsing library was integrated, extended, and tested. The extension of the library was limited to supporting the '--help' flag for printing out the application's flag usage table and various bug fixes arising from testing.

The application's configuration phase was also developed during this task. The first phase of the application reads in its configuration file and parses, then validates the parameters within which are required for it to perform correctly. In order to support this task, a user-friendly key-value configuration file format and a robust configuration file reader were implemented. These standardized configuration values are then made available to the ATD software for it to validate and store.

Also developed during this task was the application's initialization phase. The second phase of the application loads in all of the available detection algorithm plugins and configures them. To support this phase, a simple cross-platform plugin architecture was developed that allows the user to custom-implement detection algorithms and have them automatically loaded into the application. Each algorithm is based on a standardized interface which interacts with the main ATD program in straightforward ways. The ATD will load in each algorithm plugin configuration file and make it available to the algorithm plugin for it to validate and verify.

The application's run phase was next in line for development. Once the application is successfully configured and initialized, it enters the main run mode where it performs the actual work. To handle this feature, the sonar file reader(s) from Task 3 are used to read the sonar data into an internal cache for the detection algorithms to use. This internal cache keeps track of which algorithm has finished working on which piece of data, and it eliminates the data when it is no longer required. The cache has a high water mark for the amount of data (in files) that it keeps, and when it is necessary it will go out and fetch more data for use by the algorithms until all of the available data has been processed.

After all of the sonar data has been processed, the application moves on to running the two compile-time pluggable association/fusion algorithms. This work is simple because the association/fusion algorithms require little interaction with the main ATD application.

The user messaging and error reporting was developed to provide useable feedback to the operator. In order to support robust error handling and reporting, and in order to handle user-friendly messaging, it was necessary to implement a generalized mechanism inside the ATD software to pass around error states and messages. The detection algorithms and the association/fusion algorithms have the ability to report messages back to the ATD, and the ATD will report these to the user via the standard output device. They can also report warnings and errors in a similar fashion; errors will typically cause the application to halt.

An addition, midway through the project was the multi-threaded support for detection algorithms. For the efficient use of all of the available computing power on multi-core and multi-processor machines, each detection algorithm works on chunks of data in its own thread. In order to support this, the main data cache and the error/message reporting systems within the ATD application needed to be made thread-safe. This was accomplished by implementing a simple cross-platform concurrency library and by using it to spawn new threads and perform coarse-level locking protection of the relevant internals of the ATD application.

Most of the control software was tested by manually stepping through the logic of each requirement as well as demonstrating the functionality to DRDC. A built-in test was developed to validate the plugin loading and configuring logic called *basic_plugin.cpp*. This test can be found in the *src/ssmm/test/plugin* directory.

3.5 Task 5 – Development of Standardized Interfaces for ATD Algorithms

The development of the standardized interfaces for the ATD algorithms (requirements 1.19, 1.19.1, 1.23, 1.25, and 1.25.1) involved a straight-forward realization of the design from Task 2 (see Section 3.2). The bulk of the work this task was done in the design phase, and the only work required for this task was implementing the standard base class for all detection algorithms and providing implementations for common functionality.

The file format for the detection file was designed and implemented during this task and is described in Table 3. It is similar to the format for the telemetry file described in Table 2.

Table 3: Detection File Format

Item	Value	Description
1	1_0	The first number is the major version number of the detection file format and second number is the minor version number. This scheme is used for the detection files as well, with the intent that a minor version number change would be backward compatible with previous versions of the reader software while a major version would not.
2	*string*	The second string in the file is a configuration hash that is used by ATD algorithms to determine if the same ATD with the same unique algorithm identifier

		and the same configuration was already used on the input data. This is useful when processing the detections progressively and avoids duplicate detections.
3	relationship; latitude; longitude; maximum value; confidence; image filename; telemetry filename; ping number; algorithm; valid; mugshot filename;	The header explaining each column of the remaining lines in the file.
4 + N detections with M child associations	*string("+","-","—", etc); float; float; float; ratio; string; string; unsigned int; string; Boolea n; string;*	The actual values of the sonar data object state in order noted by the header in Item 2. Note that the relationship column provides information on the hierarchy of association relationships. During the ATD detection stage, only raw detections are generated so only the "+" value is used since there are no child relationships. During the first stage association, raw detections are associated with a parent synthetic association each denoted with the "+" while the children (i.e the raw detections) are denoted with the "-". This method continues as more stages of association are implemented.

Note that control software's modal operation required information to be moved to each directory configured for each stage of processing. Therefore, the associations in the detection file for the first and second stage association processing contain all detection information. Should the detection input directory change, filename information in the detection file does not become invalid. This also applies to the mugshots as described in Section 3.7.

3.6 Task 6 – Implementation of Three ATD Algorithms

The three ATD algorithms were prioritized early on in the project in order to reduce the risk of completing the plugin architect which was the focus of the contract. Match Filter was identified as the most important algorithm and was developed first with an understanding that the Speckle and Z-test algorithms would be developed if there was time at the end of the contract. The following requirements were completed under this task:

1. Development of the detection object (1.10.1)
2. Development of the Match Filter algorithm (1.12)

3. Development of the Speckle algorithm (1.14)

A series of requirements meetings were first held between DRDC and Akoostix to determine the structure of the algorithms. A number of common components were identified, such as:

- Two dimensional correlation algorithm, with a uniform kernel (single-value).
    - To implement this efficiently, a one dimensional single value correlation was implemented.
    - Also a rolling average object which averages N raster lines of an image.
- A two dimensional correlation algorithm, vertically uniform, but with width dependant kernels.
    - To implement this variation efficiently, a one dimensional correlation (with width dependent kernels) was implemented.
        - A multiply/accumulate C++ functor was implemented and used in this correlation algorithm.
- A region finding algorithm to identify regions of 8-connected non-zero pixels in an image.
- An algorithm to pad the area surrounding images with a specific value.

Each of these building blocks were developed independently using a bottom-up approach. For each block, development consisted of a short design phase, implementation and documentation. A unit test was then written and checked for memory corruption using the memory profiler Valgrind.

Using the above building blocks, the Match Filter algorithm was implemented. Other necessary (algorithm specific) constructs were implemented and documented within the plugin. For example, the Match Filter required functors to perform:

- Thresholding
- Calculating the average towfish altitude
- Re-bin based on median breakpoints
- Sum of absolute values

The plugin algorithm design relies heavily on the Standard Template Library (STL) for containers and algorithms to perform vector operations. In particular, *std::for_each()* and *std::transform()* are used frequently to perform type safe operations.

The *math* and *geo* libraries were implemented for DRDC as part of the *star++* contract. These libraries were used to calculate latitude/longitude positions of detected targets. The co-ordinates of the towfish are retrieved from the telemetry data, and along with range, bearing, and sidescan orientation (port/starboard), the target co-ordinates are calculated.

The *match_filter_synthetic_test.cpp* file contains a test for the match filter algorithm by executing it with synthetic data designed to test a standard case and a few corner cases. It is located in the *src\ssmm\test* directory. The test is a built-in unit test and was developed to test the match filter algorithm in isolation of the plugin architecture. Another test program, *match_filter_real_test.cpp* was written to help test the algorithm in isolation on real data files.

The Speckle algorithm is quite similar to the Match Filter. For this reason it was implemented very quickly re-using many of the generic components outlined above.

3.7 Task 7 – Image Mugshot Extraction

The image mugshot extraction functionality was integrated after the Match Filter algorithm was completed. The effort for this task was allocated to complete the following functional requirements:

- Provide the interface for extracting data from the sidescan sonar data (1.11, 1.11.1, 1.11.2, 1.11.3,1.11.4)
- Integrate the mugshot functionality at the required points in the software (1.21, 1.21.1)

A STL compliant random access iterator was developed to handle the responsibility of taking a contiguous buffer of memory and interpreting it as a two dimensional, row major matrix.

The clip iterator was designed to take the indices of a sub-region of the target matrix and treat it as a contiguous buffer even though it is not contiguous in memory. This improves processing efficiency and abstracts the complex math required to stride through memory to extract the required buffer cells. The iterator is generic enough to be used on any STL compliant target container that provides a forward iterator (which most do).

This clip iterator software was moved to the reusable DRDC library called *stlplus*. This library is designed to provide STL-like or STL compliant extensions for DRDC software.

Near the end of the project, the method of generating the mugshots from the original sonar data was changed to be extracted from the normalized data after the mean column curve was applied. This method includes finding the N^{th} percentile value and quantizing values of the normalized data between 0 and this value into 8 bits. The percentile value was made configurable to the user in the Match Filter and Speckle configuration files.

Unit tests were developed to test the clip iterator at its integration with the sonar data class. *sonar_data_test.cpp* tests the boundary setting and the continuity of the resulting data. As well, both *match_filter_synthetic_test.cpp*, *match_filter_real_test.cpp* and *speckle_filter_real.cpp* tests output the mugshots in isolation from the ATD control software from real data. These mugshots were visually verified.

3.8 Task 8 – Implementation of a Simple Association/Fusion Algorithm

The association/fusion task was allocated to address the requirements for the development and integration of two staged association/fusion process. The algorithm was defined and documented during the requirements process and subsequent follow-up meetings. This implemented the following requirements:

1. Defining the interface for association algorithms including a based class for extension by specialization (1.16 and 1.17).

2. The data association algorithm was supplied by DRDC and then revised to meet the two stage requirements to handle multiple passes of the sonar (1.18, 1.18.1, 1.18.2).

The same approach was taken when developing the association/fusion algorithm as when developing the ATD algorithms with the exception that the association algorithm is not a run-time plugin. Care was taken to configure and execute the main process for the association algorithm with the same interfaces as the ATD algorithm to facilitate easier migration of the association algorithms to a run-time plugin mechanism. For now, additional association algorithms can be included in the control software but will require a recompile.

The basic approach with the association algorithm is to internally represent relationships between raw detections, first stage synthetic detections, and second stage synthetic detections with a STL-like tree structure. Detections associated at either stage are written to a flat file in a tree model notation and subsequently read into memory by parsing the relationship when reading the detection files back in for the next stage of fusion (see Section 3.5). During processing, the tree model is modified by creating a synthetic node and re-parenting the child detections associated with the creation of the synthetic detection.

The basic process for both of the stages consists of:

1. Finding the reference detections from the list of unprocessed detections. For the first stage this is the greatest number of detections produced from the same instance of the detector for one sonar data object (which in turn was created from the same input file). For the second stage processing this is arbitrarily the first detection from the entire set of detections produced from the first stage.

2. Calculate the distance from the reference detections to all other detections. Again, for the first stage these detections are only from the same input file as the reference detections. For the second stage this is all of the remaining detections.

3. Select the closest detections within the association radius. When any detection is within the configurable radius of two reference detections, the shortest distance wins. Any reference detection without associations becomes a synthetic detection.

4. Record the associations by creating a synthetic detection from all associated detections and re-parent them in the detection tree as described above. Copy any mugshots from the input directory for detections recorded in the detection file. For the first stage processing this means the all mugshots are copied since both valid and invalid detections are being recorded. For the second stage, only detections that are part of a valid synthetic detection are copied.

5. Repeat the process until there are no detections left.

This software depends on the DRDC *geo* library to calculate the mean locations for synthetic detections.

Unit tests were developed to test the association/fusion algorithm for use with both stages of association with in the control software. The test is located in the *assocation_test.cpp* file in src\ssmm\test directory. This test creates a matrix of ASCII characters where each character represents detections with a detector id denoted by the character. The “center” of each cell is a configurable distance apart. The association algorithm is configured to generate synthetic detections based on this configurable distance and then again for the output of the first step.

3.9 Task 9 – Software Validation

The software validation task was re-planned at the beginning of the project. The testing plan was changed to a less formal demonstration and evaluation approach with the focus being on validating the run-time plugin architecture and algorithm output throughout the execution of the project. Although a specific test was not assigned to each functional requirement in Table 1, the software has been thoroughly tested using the following approach:

- Each requirement was reviewed during Akoostix engineering review process.
- The software was demonstrated to DRDC three times during the execution of the project.
- Unit tests were developed and have been described in each of the previous sections. The unit tests provide confirmation that key functional requirements have been met.
- A synthetic test was developed to verify the Match Filter (see Section 3.6) with known positions and maximum values. These tests were added as part of the development environment and can be used as built-in regression tests during future modification of the software.
- Real data was also used with known targets and positions with the entire software. The results of the detection algorithms were compared to the positions of the known targets using the Klein sonar viewing software.
- Valgrind was used to determine that there are not any memory management issues. The software has been provided twice to DRDC for free-play testing.

A status of each functional requirement can be found in Table 1.

4 Configuration Management

The ATD software has been branched to a 1.0-maint branch in the Subversion repository. This branch will be used to provide bug fixes to the recent version of the software while the trunk of the software will remain open and valid for future contract work. The 1.0-maint branch has additionally been tagged for read only purposes in subversion at 1.0.0. This tag represents the software as delivered for this contract and can be used to retrieve this state at any time.

A checkout of this tag has been provided on a software compact disk / digital video disk (CD/DVD) accompanying this document. Images of the repository are provided to Jim Theriault every calendar quarter. An image of the repository can be provided at any time if deemed necessary. While under contract by DRDC, Akoostix hosts the current DRDC code base repository in order to provide a central point for versioning the reusable portions of the codebase.

For more information on versioning refer to the README file provided in the root directory of the source code for the Sidescan Sonar Modules (SSMM).

5 Recommendations for Future Work

1. Add eXtended Triton Format (XTF) support

To add support for the XTF, the software will require a sonar data file reader specifically designed for that format. The XTF sonar data file reader would convert the XTF data to the standard sonar data objects that are used by the Automated Target Detector.

2. Add Marine Sonics support

To add support for the Marine Sonics sonar (Remus AUV), the software will require a sonar data file reader specifically designed for that format. The Marine Sonics sonar data file reader would convert the data to the standard sonar data objects that are used by the Automated Target Detector.

3. Add Z-Test detection algorithm

Adding support for another detection algorithm involves the development and testing of the actual formulas separately from the Automated Target Detector application, and then wrapping them in the standard detection algorithm plugin code.

4. Increased multi-threading support

The concurrency support currently used in the Automated Target Detector is a naive implementation. Further performance increases can be gained by adding additional logic to handle cases such as:

- Running multiple, simultaneous instances of the same detection algorithm on different parts of a sonar data file to distribute the processing.
- Dedicating a limited number of processors/cores for processing, allowing for continued response of the computer for other applications.
- Improved control over the type and number of algorithms running on specific processors, to allow for bunching low-overhead algorithms together, preventing operating system swapping of processes from one processor to another during breaks in data delivery.

5. Add run-time plugin support for the association/fusion algorithms

Extending the association/fusion plugin architecture from a static (compile-time) to a dynamic (run-time) implementation would allow for:

- Fast and simple swapping of differing algorithms for comparison.
- A flexible, user-defined number of association stages vice the fixed two that currently exist.
- Multi-threading of association/fusion algorithm streams, should it be desired to perform different association/fusion experiments in a single data processing run.

6. Database support

The current implementation of the Automated Target Detector uses file names and the file system to organize and cross-reference detection data from one or more data processing runs. This method of data management works fine for simple applications, but it has scaling issues as the complexity of data inter-reference and location increases. For example:

- It becomes difficult to use file names alone to distinguish data origin and uniqueness. Since file name lengths are dependent on the file system configuration, portability is an issue. Also, it becomes increasingly difficult to read file names and uniquely distinguish them. As far as implementation goes, much effort is expended to hand-write parsers to extract meta-data information from the file names.
- The uniqueness of data with respect to data source and its configuration is impossible to be expressed in a file name. Right now, the ATD uses a pseudo hash within the detection file format to attempt to distinguish data sources and their configuration characteristics. This is a sub-optimal method of handling these issues, because it is error-prone and work-intensive to ensure that the algorithms are always correctly generating updated hash strings when the number and type of parameters they use are modified.
- Data mining is difficult. In order to do advanced queries on the data sitting in files on disk, it becomes necessary to parse the header of every single file on disk and create a pseudo in-memory database. It also means having to write custom code for particular queries on the data.

Using a very light-weight, file-based database (such as SQlite), it would be possible to organize the resulting data from the detection algorithms and association/fusion algorithms into a proper set of tables that can be manipulated easily in a variety of ways. These embedded databases are small and can be compiled directly into an application. This would provide the following:

- Simple Structured Query Language (SQL) queries can be used to produce an infinite number of data mining techniques without having to implement any custom code.
- Data is stored in a cross-platform binary format, substantially reducing the size of the data footprint on disk.
- The characteristics of the data, meta-data, and source information can be added to or modified without any dependence on file names on disk. This would allow for greater flexibility in changing data and eliminate the possibility of versioning problems trying to parse old file names.
- Support tools to parse in data from existing data files, Comma-Separated Values (CSV) files, etc. can be written in any language that supports the database tool (for example, C/C++, Python, Visual Basic).
- Support tools to convert the data to text-based or CSV files are extremely easy to make, and can be done in a wide variety of languages.
- With proper design, it is possible to extend the tool to work with an enterprise-level database (such as Oracle or MySQL).

7. Improve usability of detection object containers for detection algorithms

The standardized interface for the detection algorithm plugins is well-designed, but there are a few remaining tasks that can improve the intuitiveness and the ease with which a new detection algorithm can be developed. Right now, the detection algorithms deal with the detection data by means of a tree-like structure which represents the hierarchy of synthetic detections and their associated sub-detections. Navigating this hierarchy is not difficult, but it could be improved by providing an additional abstraction that simplified the view into the detection tree and eliminated the need to manage the detection container manually.

8. Image normalization designed specifically for viewing tiff files and mugshots

The current TIFF quantization scheme was originally ported from DRDC software. This scheme assumed that a bit shift of a 16 bit value by 2 bits would provide sufficient brightness to the image to allow viewing. This scaling factor was based on example data that may have different gain characteristics then data captured by the same Klein sonar that has been calibrated or modified. Therefore a more data generic method could be applied to the TIFF generation process that uses normalization and quantization techniques that enhance the image for viewing (which may not necessarily be the same as for detection).

The mugshots are created differently than the viewable TIFFs converted from the sonar data. The mugshots are created from the normalized data in both the Match Filter algorithm and the Speckle algorithm. For the speckle algorithm, this required an increase in computational load since the normalized data required sorting to find the 95 percentile value. In the future a more standard way of producing mugshots could be generated that normalizes just the desired mugshot region with a known complexity and would minimize the load on the algorithms.

9. Streaming model versus Batch Processing model

The current software works on data on a file-by-file basis in batch processing mode. This model of processing works fine for post-analysis processing but may not scale well to small lightweight or embedded systems with limited processing power and limited memory. A streaming model could be employed with some redesign of the system. A streaming model would have the added benefits of not having to deal with edge effects during detection processing.

10. Create an application packaging and installation program

The current process for generating a release is done manually in an error prone and time consuming way. There is some benefit for developing a script in the build environment to:

- Locate all the required dependencies.
- Package the software with these dependencies.
- Setting up the configuration scripts with default target installation directories.
- Setting up the required environment variables.

The installation program would have to support both Windows and Linux platforms.

References

[1] G.J. Dobeck, J.C. Hyland and L. Smedley, “Automated detection/classification of seamines in sonar imagery”, Proc. SPIE-Int. Soc. Optics, vol. 3079, pp.90-110, 1997.

[2] R.T. Kessel, “Texture-based discrimination of man-made and natural objects in sidescan sonar imagery”, Proceedings of the SPIE, Vol.5096, Aerosense 2003, Orlando, Florida, pp.21--5, April 2003.

[3] J. Fawcett, A. Crawford, D. Hopkin, V. Myers and B. Zerr, “Computer-aided detection of targets from the CITADEL trial Klein sonar data”, DRDC Atlantic TM 2006-115, November 2006.

[4] Y. Petillot, S. Reed and V. Myers, “Mission planning and evaluation for minehunting AUVs with sidescan sonar: Mixing real and simulated data”, Report SR-447, NATO Undersea Research Centre, La Spezia, Italy, December 2005.

List of symbols/abbreviations/acronyms/initialisms

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATD	Automated Target Detection
CD	Compact Disk
CSV	Comma-Separated Values
DRDC	Defence Research & Development Canada
DVD	Digital Video Disk
PA	Project Authority
R&D	Research & Development
SACC	Standard Acquisition Clauses and Conditions
SDF	Standard Data Format
SOW	Statement of Work
SQL	Structured Query Language
SSMM	Sidescan Sonar Modules
STL	Standard Template Library
TIFF	Tagged Image File Format
USGS	United States Geological Survey
XTF	eXtended Triton Format

Distribution list

Document No.: DRDC Atlantic CR 2009-046

LIST PART 1: Internal Distribution by Centre

1	John Fawcett
1	Warren Connors
1	Vincent Myers
5	DRDC Atlantic Library
8	TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1 Library and Archives Canada
Attention: Military Archivist, Government Records Branch

1 DRDKIM

1 Jason McInnis
Akoostix Inc.
10 Akerley Blvd, Suite 12
Dartmouth, NS B3B-1J4

1 LCdr. J. Greenlaw
DMRS 3-2, NDHQ
101 Colonel By Dr.
Ottawa, ON K1A 0K2

4 TOTAL LIST PART 2

12 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Akoostix Inc. 10 Akerley Blvd, Suite 12 Dartmouth, NS B3B 1J4	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Structured Software Implementation of Computer Detection Algorithms for Sidescan Sonar Data: Final Report

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

McInnis, J.; Ryan, G.

5. DATE OF PUBLICATION (Month and year of publication of document.) May 2009	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) 44	6b. NO. OF REFS (Total cited in document.) 4

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Contract Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Atlantic
9 Grove Street
P.O. Box 1012
Dartmouth, Nova Scotia B2Y 3Z7

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) W7707-088118/001/HAL
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.) DRDC Atlantic CR 2009-046

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This report summarizes the work performed for the Structured Software of Computer Detection Algorithms for Sidescan Sonar project for Defence Research and Development Canada (DRDC) - Atlantic. This contract developed an initial test bed for mine-hunting Automated Target Detection (ATD) algorithms to run using batch processing on sidescan sonar data such as the Klein 5500. The first version of the software developed under this contract achieved its goal through various methods including incorporation of DRDC mine detection algorithms, through the reuse and upgrade of existing DRDC libraries, and through the development of new code. The resulting software is faster than real-time, cross-platform and easy to extend through a dynamic run-time plugin interface.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

sidescan, sonar, detection

This page intentionally left blank.