An ATR architecture for algorithm development and testing

Gøril M. Breivik*, Kristin H. Løkken, Alvin Brattli, Hans C. Palm and Trym Haavardsholm

Norwegian Defence Research Establishment (FFI), P.O. Box 25, N-2027 Kjeller, Norway

ABSTRACT

A research platform with four cameras in the infrared and visible spectral domains is under development at the Norwegian Defence Research Establishment (FFI). The platform will be mounted on a high-speed jet aircraft and will primarily be used for image acquisition and for development and test of automatic target recognition (ATR) algorithms. The sensors on board produce large amounts of data, the algorithms can be computationally intensive and the data processing is complex. This puts great demands on the system architecture; it has to run in real-time and at the same time be suitable for algorithm development.

In this paper we present an architecture for ATR systems that is designed to be flexible, generic and efficient. The architecture is module based so that certain parts, e.g. specific ATR algorithms, can be exchanged without affecting the rest of the system. The modules are generic and can be used in various ATR system configurations. A software framework in C++ that handles large data flows in non-linear pipelines is used for implementation. The framework exploits several levels of parallelism and lets the hardware processing capacity be fully utilised.

The ATR system is under development and has reached a first level that can be used for segmentation algorithm development and testing. The implemented system consists of several modules, and although their content is still limited, the segmentation module includes two different segmentation algorithms that can be easily exchanged. We demonstrate the system by applying the two segmentation algorithms to infrared images from sea trial recordings.

Keywords: System architecture, algorithm development, image processing, segmentation, parallel processing

1. INTRODUCTION

Automatic Target Recognition (ATR) is the process where computer algorithms detect and classify specified types of targets in sensor data. Target tracking can also be an integrated part of such a system. At the Norwegian Defence Research Establishment (FFI) we currently develop a research pod that will be used for image acquisition and ATR algorithm testing. The pod is based on a cargo/travel pod used as a transport volume for a wide range of jet aircrafts. We have rebuilt the pod into a research platform containing cameras, navigation hardware and processing computers. The pod will be mounted on the wing of an RNoAF F-16 aircraft, as illustrated in Figure 1. This enables image acquisition and ATR algorithm tests at high speeds.

Figure 1. A research pod (illustrated in orange) will be mounted on an F-16 aircraft wing for data acquisition and ATR algorithm tests at high speeds.

*E-mail: goril-m.breivik@ffi.no; telephone: +47 63 80 70 00

Automatic Target Recognition XXIII, edited by Firooz A. Sadjadi, Abhijit Mahalanobis, Proc. of SPIE Vol. 8744, 874402 · © 2013 SPIE · CCC code: 0277-786X/13/$18 · doi: 10.1117/12.2015369

The four cameras in the pod operate in the infrared and visible spectral domains. They will be mounted on a steerable platform that can move in the yaw and pitch directions. This will be used to point the cameras towards an aim point in the scene. The aim point can be defined in several ways; programmed preflight, provided during flight by the pilot or by the target itself (using e.g. AIS), or computed continuously by the ATR system on board.

The pod will be used for both image acquisition and to test ATR algorithms in a relevant environment in terms of hardware as well as scenes. This means that the software system in the pod must fulfil certain requirements. It has to be based on an architecture that enables simple exchange of certain algorithms, and that at the same time ensures efficient processing of large amounts of camera and navigation data. For algorithm development, desktop computers and recorded data will be used. Hence, the software architecture must also be reconfigurable into simpler systems for algorithm development purposes.

The ATR architecture presented in this paper is well suited for efficient data processing and algorithm tests as well as for algorithm development. The architecture is module based so that certain parts, e.g. specific ATR algorithms, can be exchanged without affecting the rest of the system. A software framework[1,2] that makes full use of the pod's processing capacity is used for implementation. Finally, the generic form of the modules makes it possible to reconfigure the pod software to better exploit the pod hardware, and also to build simpler systems for ATR algorithm development on desktop computers.

The ATR system for the pod is under development and the implementation has reached a first level that can be used for segmentation algorithm test and development. Two segmentation algorithms have been tested in this system. We present a description of the current implementation, the segmentation algorithms and results from tests on infrared sea trial recordings.

This paper is structured as follows: In Section 2 the pod and its hardware components relevant for the ATR system are described. The proposed ATR software architecture for the pod is presented in Section 3. This section also gives a brief introduction to the pipeline structure and the framework used for implementation, as well as a discussion on how to best exploit the given pod hardware. In Section 4 the current ATR pipeline implementation is described and utilised to compare performance for two different segmentation algorithms on infrared imagery. Section 5 concludes this paper and gives an outlook on future work.

2. THE HARDWARE SYSTEM

In Figure 2, the pod hardware components most relevant for the ATR system are illustrated with boxes. The boxes also indicate the approximate hardware component location in the pod.

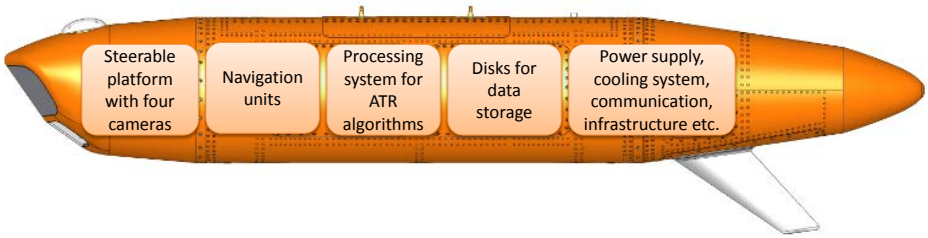

Figure 2. The pod and the hardware components most relevant for the ATR system.

The main sensors in the pod are two IRCAM 327k Equus infrared cameras. One of the cameras covers the long wave infrared (LWIR) spectral region, and the second covers the mid wave infrared (MWIR) spectral region. The cameras are developed for our needs, and comprise the possibility to add up to six different filters in a filter wheel that spins in front of the detector. Hence, when the camera acquisition rate is set to 100 Hz, up to six sub-bands within the original wavelength range can be captured at 16.7 Hz rate. The third camera is a Sensors Unlimited SU640HSX-1.7RT infrared camera in the short wave infrared (SWIR) spectral region, and the last camera is either a black/white or a colour 2360/2360C visible (VIS) light camera from Gevicam.

The navigation hardware includes one 300 Hz HG9900 inertial measurement unit (IMU) from Honeywell, and one 20 Hz OEMV-3 global navigation satellite system (GNSS), including GPS, GLONASS and DGPS, from

NovaTel. Real-time navigation is performed using NavP, which is a real-time implementation of an inertial navigation system developed at FFI.[3] All time-critical data is time stamped using a digital synchronization unit (DSU) also developed at FFI. The DSU can divide the high precision 1PPS signal from the GPS into clock resolutions up to microseconds. Time stamps are set e.g. when trig pulses are received from the cameras.

The captured raw data will be stored directly to X-25E SLC 2.5" SATA-II solid state hard-disks from Intel. The processing system that runs the ATR algorithms consists of three computer processing units (CPU) and three graphical processing units (GPU). The CPUs are of type KTQ77/FLEX from Kontron and the GPUs are of type Tesla C2070 from NVIDIA. The CPUs have three Gigabit network inputs each and the possibility to add more using add-on boards. Communication between the components in the pod system will be based on 1 Gb Ethernet. Figure 3 shows the steerable camera platform that is developed by SWESYSTEM AB. The platform can be controlled in yaw and pitch directions. Shifts in the roll direction will be handled by the software.

The LWIR and MWIR cameras deliver 640×512 pixel images at a rate up to 100 Hz. The SWIR camera delivers 640×512 pixel images at a rate up to 30 Hz and the visible light camera delivers 656×494 pixel images at a rate up to 100 Hz. Hence, up to 175 MB of image data is produced per second. The 100 Hz rate might seem over-dimensioned when considering that mechanical control requires a minimum of time between each update, and that, even when flown at high speeds, the pod will move only a few meters in 0.01 seconds. However, for the sub-band images to be delivered at 16.7 Hz rate the 100 Hz camera rate is required.

Figure 3. The camera platform located in the front of the pod is steerable in the yaw and pitch directions. Four cameras are mounted on the platform; the orange LWIR and MWIR cameras on the top, and the black SWIR and visible cameras below.

3. ARCHITECTURE

This section describes the proposed pod ATR system architecture in more detail and discusses how to best exploit the given pod hardware. However, first we explain the pipeline structure chosen for the architecture and the framework applied for system implementation.

3.1 Pipeline processing

For an automatic target recognition system, important problems include target segmentation, classification and tracking. The system receives data, streamed either in real-time from cameras or from recordings on a disk, and the images normally need to be preprocessed to be ready for analysis. The output from an ATR system is typically a list of recognised targets and their positions.

The preprocessing, segmentation, classification, tracking and target selection constitute a list of sequential tasks. Each task depends on the output of the previous task, but may perform its processing independently of all the other tasks. This programming equivalent to an assembly line is called a pipeline structure, and each task is called a pipeline stage. The pipeline structure allows concurrent processing of successive tasks, and will thereby increase throughput with the possible cost of introducing a higher latency. Figures 4 and 5 are examples of pipeline structures.

The pipeline structure also yields a modular design that can later be extended or modified. This means that one task (stage), e.g. the segmentation algorithm, can be modified or exchanged without affecting the rest of

the pipeline. It is also possible to extend the pipeline with additional stages, or build a new pipeline with the same stages applied in a different configuration.

The ATR system proposed in this paper is based on hyPipe,[1] a cross-platform software framework implemented in C++, that handles large data flows in a non-linear pipeline structure. The hyPipe framework was originally built for an airborne real-time hyperspectral target detection system.[2]

The framework lets the ATR system naturally exploit task level parallelism, as previously explained. Data level parallelism, where e.g. all pixels in an image are processed in parallel, can be exploited within each stage using e.g. CUDA. Thus the framework makes the most of both the CPU and the GPU hardware and their inherent processing capacity. It is designed to be generic and easily adaptable, using object oriented design principles, and distribution across several computers is possible using TCP/IP sockets.

3.2 The ATR architecture

The pipeline shown in Figure 4 is a schematic representation of the proposed ATR architecture for the pod. The dark blue boxes represent hardware; the long wave, mid wave, short wave and visual cameras are at the start of the pipeline, and at the end is the camera platform that will be steered towards an aim point in the scene, based on analysis of images delivered by the cameras.

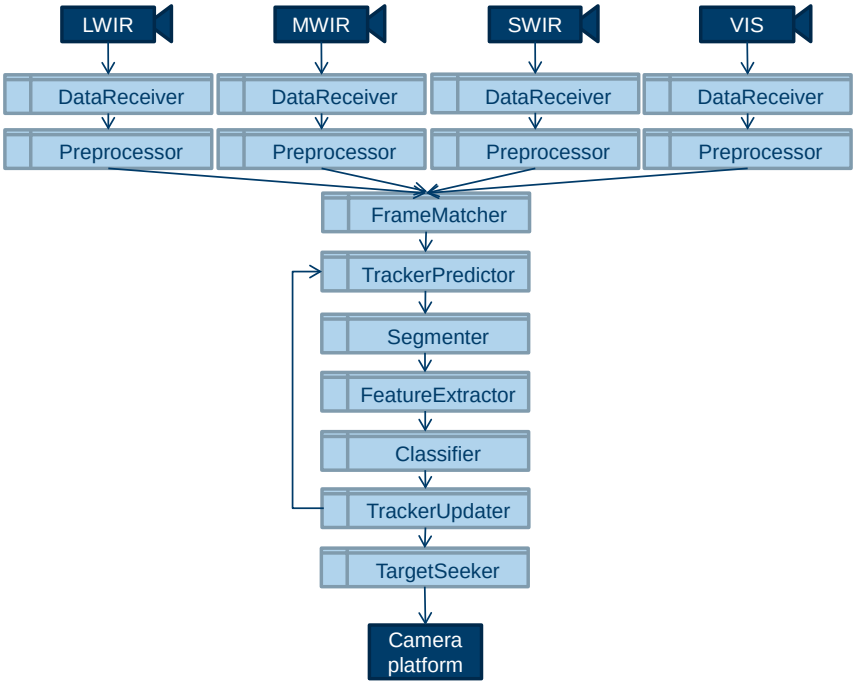

Figure 4. Schematic illustration of a pipeline where data flows from the cameras and through a list of stages. Output from the system is a position to an aim point, and this is used to control the camera platform. This pipeline represents the ATR architecture designed for the research pod.

Each of the light blue boxes represents one stage (module) in the ATR software. Each stage can process its task independently from the other stages. The data flow, i.e. the input and output, is represented with arrows between the different stages, and between the pipeline stages and the hardware components.

In the designed architecture, the DataReceiver stage receives images from the cameras and navigation data from the navigation system in the pod and combines them based on timestamps. The navigation unit is omitted in Figure 4 to simplify the figure. The Preprocessor stage prepares the camera images for further analysis and performs e.g. noise reduction and roll stabilisation. These two stages run separately and in parallel for each of the cameras.

The FrameMatcher receives streamed images from each of the cameras and makes sure the different images are represented with the same timestamp and in a common coordinate system, so that the images can be analysed with respect to each other. The TrackerPredictor uses the time elapsed since last tracker update and predicts where previously detected targets will be located with a given uncertainty for the current images. This information can be utilised in the subsequent image analysis stages to facilitate re-detection of tracked objects and new targets.

The image analysis stages involve the Segmenter, the FeatureExtractor and the Classifier. The Segmenter separates interesting areas from the image background. Characteristic features for each segment are computed by the FeatureExtractor and sent to the Classifier for target classification.

The TrackerUpdater tracks each object over time. It establishes, maintains and terminates tracks based on the previously predicted tracks and computed segments, features and classes for the current image. As indicated with an arrow in Figure 4, the TrackerPredictor and the TrackerUpdater seem mutually dependent on each other. However, the track prediction performed before image analysis starts will not wait for the updated tracks from the previous image, since this could violate the concurrent task processing inherent in the pipeline structure. Instead, the track prediction will be based on the last updated tracks available when the new image is received. The TargetSeeker will perform application specific tasks, such as target selection and aim point determination.

The interfaces between the stages are important, since they have to be predefined for the stages to be exchangeable and generic. We do not present them here, since their details will likely be subject to change during the early implementation phase. Instead, we give an idea on the type of data flowing between the stages.

The input and output for each stage are organised into data structures, and which data structures that flow between the stages vary. Typically, the structures early in the pipeline contain images from one or more cameras, their timestamps and e.g. their corresponding navigation data. After image analysis is performed, the data structures can contain binary images defining segmented areas, lists of features describing each segment or classification results such as class and confidence. The tracking will result in a data structure containing a list of tracks, their position and uncertainty in position estimates. The output from the ATR pipeline to the camera platform is an aim point in global coordinates.

3.3 Exploiting the pod hardware

The proposed ATR architecture is very flexible and exploits several levels of parallelism. The question is how to distribute the ATR pipeline stages on the given pod hardware to ensure maximum processing efficiency. In general, one can say that computationally intensive tasks should be divided into parallel stages that execute in different threads on a multi-core CPU or distributed over several CPUs, or they should exploit data level parallelism using e.g. the GPU.

The modular ATR design enables non-linear pipelines with parallel stages for concurrent processing. This is utilised at the DataReceiver and Preprocessor stages in Figure 4, where processing is kept separate and in parallel stages for the four camera image streams. Thus, images from the four cameras can be processed simultaneously, either in several threads on one CPU, or distributed over more computers. Other stages in the pipeline, e.g. the FeatureExtractor or the Classifier, can with minimum effort be split into similar parallel stages if needed. The processing need for these stages is typically dependent on the number of objects segmented in the scene. However, some overhead is introduced with splitting, and processing speed-up using the GPU will probably be preferred where possible.

The Segmenter stage is expected to be computationally intensive, due to image processing operations such as filtering, morphological operations etc. In principle, the Segmenter could be separated into parallel stages for each image stream, as explained above. However, we have chosen to keep the Segmenter in one stage for the possibility to process images from more cameras simultaneously and with respect to each other. Thus, rapid processing is ensured at this stage utilising the GPU. Image processing algorithms are often well suited for parallel processing using either CUDA to write algorithms, or exploiting existing functions from libraries that utilise the GPU.

The interfaces will be carefully considered when distributing the pipeline stages on the hardware. One IRCAM camera image stream will need the full capacity of a Gigabit Ethernet line when running at maximum rate. The

CPU boards have at least three network inputs each, and the hyPipe framework allows the pipeline stages to communicate over TCP/IP. However, if the images from all the cameras are put into one data structure, this will be too large to be delivered at full rate over one network line.

One way to avoid handling large data streams on the Ethernet is to distribute the stages so that those that communicate the most data run on the same computer. When running on the same CPU, the framework ensures that communication between the stages is performed through safe use of pointers instead of moving the data itself. This is less I/O intensive and thus more effective. Which stages that utilise the GPUs must also be considered. One GPU is mounted on each of the CPU boards, and all the stages that utilise the GPU should not be run on the same CPU but rather be distributed according to their computational needs.

These considerations are taken to ensure that all the pipeline stages perform as efficient as possible. Whether each stage is loaded with equally intensive tasks, is rather difficult to predict before a specific set of algorithms is applied. Nevertheless, equal processing loads for each stage is important for the pipeline structure to run efficiently and to avoid that one stage ends up waiting for the previous one to finish its task. Thus, once all stages are implemented, the relative processing load for each stage will be analysed and improved where possible, and the complete distribution of stages on the available pod hardware will be decided.

4. IMPLEMENTATION AND TEST

This section presents an example of a pipeline implementation based on the ATR system presented in Section 3. The implementation can be used for segmentation algorithm development and testing. We give a brief introduction to the two segmentation algorithms included at the Segmenter stage, and present the results from utilising the implemented system to segment targets in an infrared image recording.

4.1 Implemented system

The ATR system for the pod is under development. We are currently implementing the system in C++, utilising the hyPipe framework described in Section 3.1 and the OpenCV library. The implementation has reached a first level and can be used for segmentation algorithm development and test purposes. Figure 5 illustrates the test system and the modules included.

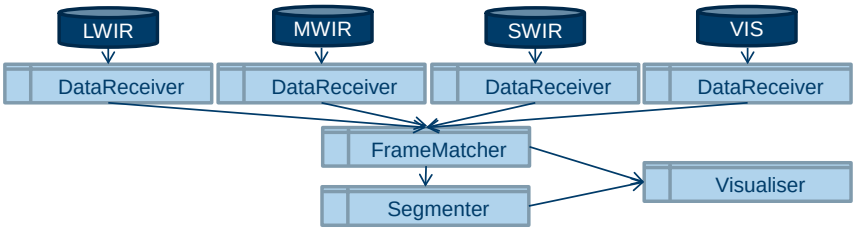

Figure 5. Schematic representation of the implemented pipeline for segmentation algorithm development and testing.

The currently implemented DataReceivers differ from the ones that will be implemented for the pod, since their task in the current implementation is to read image recordings from file, not to receive image streams from cameras. The future DataReceivers have to offer both types of functionality; file reading for replays and streamed image reception from cameras for pod runs. The FrameMatcher is very simple; it forwards the image streams, but still lacks the image synchronisation functionality.

The Segmenter stage is more set and could be plugged directly into a complete ATR system in the pod. It contains two different segmentation algorithms, and which to use is defined at system start up. The Segmenter interface takes as input an image and the output is a segmented and labelled image. The Visualiser stage is not a part of the pod architecture in Figure 4. However, it is an important module to facilitate debugging during algorithm development and testing. It takes as input the original images as well as the segmentation results, and visualises the infrared image with the segments overlaid. The system is initialised using a configuration file that defines which recordings, algorithms and parameters to be used for the current run.

The implemented pipeline stages have rather simple contents. However, the pipeline connections and communication is equal to the complete ATR system. Thus, the current system is an implemented example of parts of the pipeline in Figure 4. It also is an example of a system that can be used for segmentation algorithm development and testing. To illustrate this further, we will compare the two different algorithms included at the Segmenter stage by applying them to the same data set, utilising the implemented system in Figure 5.

4.2 Segmentation algorithms

The first segmentation algorithm separates the foreground from the background simply using a hard-coded threshold. Such a simple algorithm is very sensitive to intensity changes in the imagery and not suited for any applications. It is included in this paper only to illustrate the implemented test system's ability to switch between and test different algorithms.

The second segmentation algorithm was presented by Felzenszwalb and Huttenlocker in 2004,[4] and we call it the FH algorithm. The algorithm is a graph-based clustering method that adaptively adjusts its segmentation criteria based on the degree of variability in neighbouring regions in the image. An important characteristic of this method is thus its ability to preserve detail in low-variability image regions while ignoring detail in high variability regions. As input the algorithm needs a (preprocessed) floating point raw image and three numerical parameters, and its output is a segmented and labelled image. The numerical input parameters are σ, which controls the amount of noise-reducing Gaussian blur applied to the input image before segmentation; k, which effectively is a scale of observation (larger k causes a preference for larger components); and M, which is the minimum amount of pixels allowed in a segment. The FH algorithm runs in $O(n \log n)$ time.

Before applying the FH algorithm, the raw image is subject to some preprocessing. This includes a 3×3 median filter to remove bad pixels, and a normalisation, which linearly maps the input pixels to the range $[0, 255]$. After segmentation, the segmented image is subjected to some post processing. The post processing first removes segments bigger than a fraction f of the total image (typically large structures in the sky and in the water). Then it applies a morphological opening operation with a 3×3 kernel on each of the segments, which removes thin whiskers protruding from them, as well as removing thin bridges of connection between segment blobs. The last post processing operation joins neighbouring segments, on the assumption that segments that are 8-connected through one or more pixels actually belong to the same segment. This last operation also relabels the segmented image.

4.3 Test on sea trial recordings

We have processed long wave infrared data using the implemented system with the two different segmentation algorithms. The data is recorded with a Cedip Emerald LWIR camera and shows the cargo ship Nina passing outside the Norwegian coastline at Lista in June 2012.

Output from the system, provided by the Visualiser, is a plot showing the original LWIR image overlaid with cyan-coloured segments found by the segmentation algorithm. The plot is updated for every image in the recorded video. Figure 6 shows the output for each of the algorithms (the simple threshold to the left and the FH algorithm to the right) when applied to one of the images in the LWIR recording. For the simple algorithm, the hard-coded threshold was 0.84 after the pixels had been mapped to the range $[0, 1]$. The FH algorithm settings were $\sigma = 0.8$, $k = 500$, and $M = 20$ for preprocessing and $f = 0.2$ for the post processing.

In Figure 6, the segmented ship can be seen as the largest cyan segment in both images. The other segments are mainly due to waves in the sea. Some segments can be found in the sky, and these stem from image artefacts introduced by defects inherent in the camera. In this particular image, the simple threshold algorithm seems to give a cleaner image with fewer false segments, e.g. due to sea waves, than the FH algorithm. However, one must keep in mind that the hard-coded threshold is adapted to this dataset, and that the FH algorithm will perform better in general.

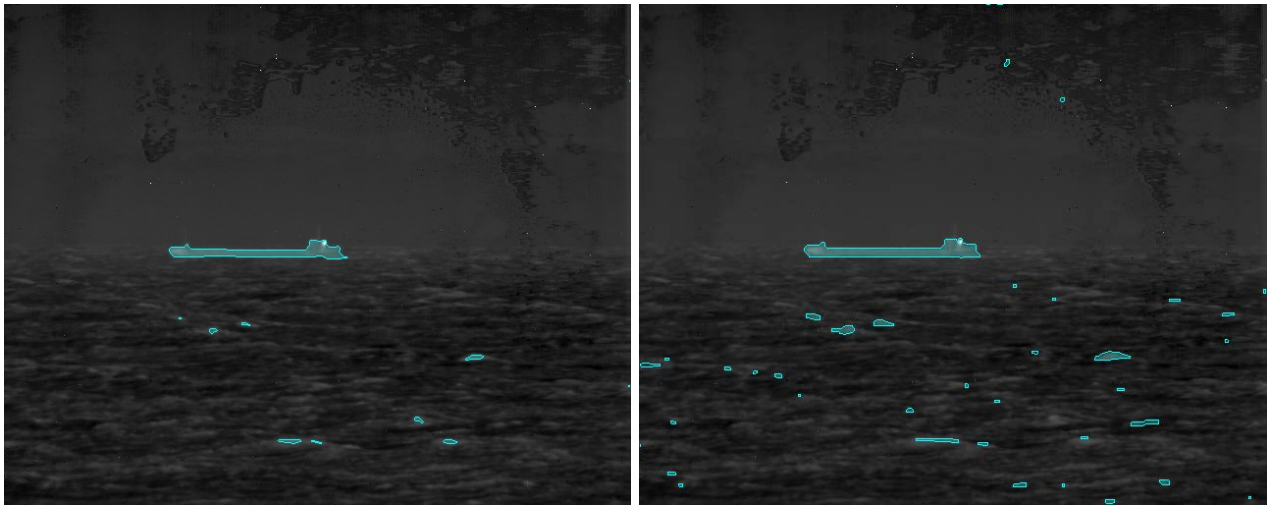

Figure 6. The original long wave infrared image with the segmented image overlaid in cyan. The segmented images are obtained using a simple threshold (left) and an algorithm by Felzenszwalb and Huttenlocker[4] (right).

5. CONCLUSIONS AND OUTLOOK

We have presented an ATR architecture for a research pod that is currently being developed at FFI. The architecture is well suited for efficient ATR processing and algorithm tests as well as for algorithm development, due to its pipeline structure where tasks are separated into generic stages and processed concurrently. The stages can be modified, exchanged and reused in other configurations. Pipeline processing has been demonstrated through the currently implemented system, where different segmentation algorithms have been applied to recorded data.

The implemented system forms the basis for the research pod ATR system. Further work includes implementing a complete ATR system that runs in real-time. Once the ATR system is finished, it will be used for image acquisition and algorithm test during flight. In the near future, interfaces for each stage will be defined in detail. Thus, development systems for specific stages can be implemented and utilised for continuous algorithm development on desktop computers. The developed algorithms can be used to update the ATR system in the pod. This enables test of new algorithms in a relevant environment, and also increases the lifetime for the pod and its ATR system.

REFERENCES

[1] Haavardsholm, T. V., Arisholm, G., Kavara, A., and Skauli, T., "Architecture of the real-time target detection processing in an airborne hyperspectral demonstrator system," in [*2nd Workshop on WHISPERS*], 1–4 (2010).

[2] Skauli, T., Haavardsholm, T. V., Kåsen, I., Arisholm, G., Kavara, A., Opsahl, T. O., and Skaugen, A., "An airborne real-time hyperspectral target detection system," in [*Proc. SPIE*], **7695** (2010).

[3] Jalving, B., Gade, K., Hagen, O. K., and Vestgard, K., "A toolbox of aiding techniques for the HUGIN AUV integrated inertial navigation system," in [*Proc. OCEANS*], **2**, 1146–1153 (2003).

[4] Felzenszwalb, P. F. and Huttenlocker, D. P., "Efficient graph-based image segmentation," *International Journal of Computer Vision* **59**(2), 167–181 (2004).