Flexible AUV Vision Framework

Jean-François Im
Undergraduate student
SONIA AUV project
École de technologie supérieure
1100 Notre-Dame West
Montreal, Quebec Canada
jeanfrancois.im@gmail.com
http://sonia.etsmtl.ca
P : (514) 396-8800 #7622
F : (514) 396-8624

Martin Morissette
Undergraduate student
SONIA AUV project
École de technologie supérieure
1100 Notre-Dame West
Montreal, Quebec Canada
martin.morissette@gmail.com
http://sonia.etsmtl.ca
P : (514) 396-8800 #7622
F : (514) 396-8624

Félix Pageau
Undergraduate student
SONIA AUV project
École de technologie supérieure
1100 Notre-Dame West
Montreal, Quebec Canada
felix.pageau@gmail.com
http://sonia.etsmtl.ca
P : (514) 396-8800 #7622
F : (514) 396-8624

Abstract

Proper object recognition using machine vision can be a hard to achieve task especially within hostile environments. It requires time consuming calibration upon deployment and can be hard to properly test. This paper will present how a team of volunteer students from the École de technologie supérieure (ETS) in Montréal, Canada has come up with a solution for their Autonomous Underwater Vehicule (AUV) platform. The team's vision framework supports rapid filter prototyping using a What You See Is What You Get (WYSIWYG) editor, fully customizable algorithms through a flexible telemetry interface and rapid testing thanks to its MPEG4 streaming, recording and playback capabilities.

***Keywords**: machine vision, autonomous vehicle, AUV, framework, image processing*

Introduction

Machine vision is an essential technology when it comes to autonomous vehicles. Acting as a sensor, it provides vital information to intelligent systems and allows for proper navigation and interaction with the environment. A machine vision system can be considered as a stand-alone passive sensor that doesn't require the deployment of external devices and has no apparent emissions that could eventually be detected.

Unfortunately, this class of systems requires extensive configuration and calibration when applied to differing environments. Slight changes in the lighting conditions can increase the occurrence of false positives (α error) and false negatives (β error) and cause the systems to react incorrectly. When used in natural environments, educated trial and error is the typical choice for the calibration and configuration of a machine-vision system. Trial and error is a simple approach to empirically determine the proper camera and algorithm settings without requiring exhaustive prior knowledge of the given environment.

The design of vision algorithms that reliably detect objects can be extremely hard to achieve. The task requires extensive knowledge of computer vision theory; the technology used for the development (programming language and operating system) as well the natural environment in which it is to be used. It also requires time-consuming testing on various data sets to confirm the applicability of the end-result.

This paper describes how the SONIA vision framework was designed to simplify the entire process of developing, configuring and deploying machine vision within autonomous systems. We will enumerate the requirements and specification that were established before the creation of the framework, give a detailed description of the implementation and discuss future developments the team is considering in order to keep improving the framework.

Background

Established in 1999, project SONIA Autonomous Underwater Vehicle (AUV) of École de technologie supérieure is a group of volunteer undergraduate students working to develop an innovative and high-performance AUV specifically designed for the needs of an international engineering competition. Each year, SONIA AUV takes on the AUVSI and ONR's challenging

Autonomous Underwater Vehicle Competition [1] against teams from the United States, Canada, India and Japan. The competition's mission objectives are representative of tasks carried-out by civilian and military AUVs currently deployed in the world's oceans, seas, lakes and rivers. Many of these objectives require object recognition and navigation using machine vision. Examples include:

- Locating and homing-in on an underwater buoy
- Inspecting a pipeline
- Locating and marking an underwater box (Target Bin)
- Recovering an "X"shaped treasure

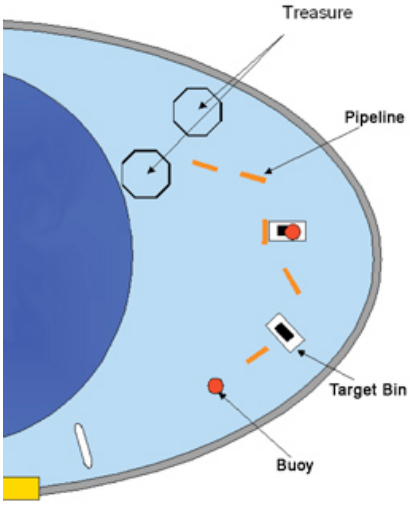

Figure 1 - Exemple mission environment layout [1]

In the context of a team of volunteer students, knowledge about complex subjects like machine vision needs to be passed-down from one generation of members to the next as senior students graduate and freshmen join the team. Tools and frameworks need to be developed to alleviate the steepness of the learning curve for core technologies to allow freshmen to quickly adapt to the system and become productive. The SONIA vision framework was designed to limit such learning curves.

Requirements and specifications

The SONIA AUV team follows the unified process for the development of all its software. Table 1 lists the requirements that were elaborated during the inception phase of the vision framework.

Table 1 - SONIA vision framework requirements

ID	Requirement description
RS1	Execute algorithms for object recognition
RS2	Enable/disable algorithms on-the-fly
RS3	Create, modify, delete vision algorithms visually
RS4	Configure algorithms remotely
RS5	Monitor camera output remotely
RS6	Monitor algorithms output remotely
RS7	Record camera footage to a video file
RS8	Playback recorded videos

Given that the success of our vehicle's mission relies greatly on the vision framework, here are some of the quality requirements [2] that were considered important:

- High performance, meaning that the system has to process images fast enough for the vehicle to navigate in real time using the vision data as input.
- High availability, meaning that the system must be fully functional for at least the maximum time of a deployed mission, which in our case is 3-5 hours.
- The ability to interoperate with SONIA's current software modules.

Design and implementation

The SONIA vision framework is written in C++ for the Linux operating system. It was designed to natively support OpenCV [3] data structures and algorithms. The following section delves into the design and implementation of the framework.

SONIA's software architecture

To properly understand some of the decisions taken during the design of the vision framework, a brief overview of SONIA's current software architecture [4] is required. The main module

responsible for data-gathering and mission execution is called Autonomous Underwater Vehicle version 4 (AUV4). It is entirely written in Java and communicates directly with the electronics bus while providing information to a telemetric interface. **Figure 2** shows a block diagram of the original architecture before the advent of the vision framework. AUV4 was responsible for the machine vision processing, which did not answer any of the previously defined requirements and quality attributes. Furthermore, it added an undesired complexity factor to AUV4.

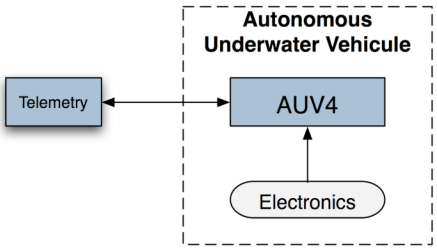

Figure 2 - Software architecture block diagram – before vision

Client-Server architecture

To simplify the integration of the vision framework within SONIA's existing software modules, the Client-Server architecture was followed. Clients, in our case AUV4 and a telemetric interface (vision client), can establish a connection with the vision framework that provides them with the required data services. **Figure 3** shows how the vision framework was integrated within SONIA's global architecture.

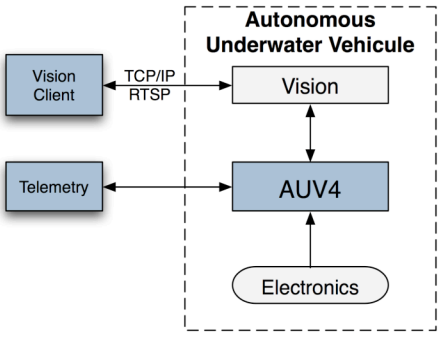

Figure 3 - Software architecture block diagram - with vision

The Client-Server architecture offers many advantages other than being simple to integrate. Since all data and transactions happen over a socket connection, the framework can be executed from virtually anywhere accessible over a TCP link, which means it can be deployed to its own dedicated computer for optimal performances. Acting as a server, the vision framework is capable of providing services to many external systems (clients). This allows for more than one client to be connected to the same source requesting mission critical information such as the location of objects. Such a situation is illustrated in Figure 3 where AUV4 and the Vision Client communicate with the vision framework.

Pipes and Filters

Machine vision algorithms are designed following the Pipes and Filters architectural pattern [5]. This provides a structural approach in the creation of elements capable of detecting objects in a real-world environment. As denoted in **Figure 4**, filters are atomic algorithms that process the images to provide a desired output. Combined together, these filters can detect objects and return positioning information to clients (i.e. AUV4 navigation module).

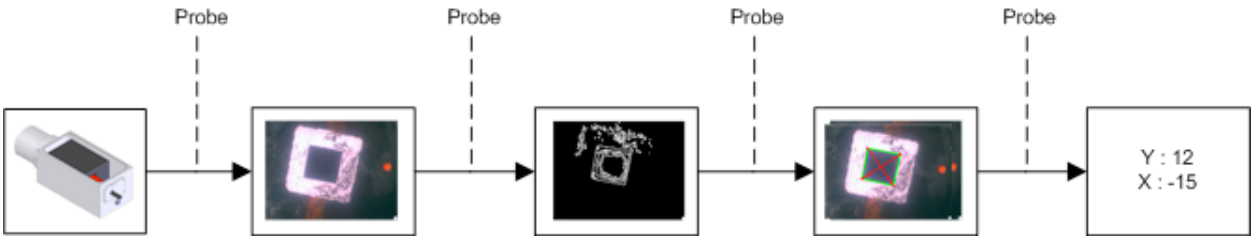

Figure 4 - Pipes and Filters architecture: this configuration is used for the detection of a target bin

Many processing algorithms were designed as filters and can therefore be reused by developers to create complete machine vision algorithms for the detection of complex objects by simply joining filters together. This architecture, combined with the client user interface allows for rapid

prototyping of such algorithms. The following list outlines some of the algorithms [6] readily available with the framework.

- RGB2YUV
- YUV2RGB
- RGB2HSL
- HSL2RGB
- ThresholdRGB
- ThresholdHSL
- AdaptiveThreshold
- RedEnhancer
- GaussianBlur
- HoughLine
- Sobel
- Canny
- SquareDetector
- Histogram
- Piexelizer

The ordering of filters can be changed at run-time, which simplifies the testing of machine vision algorithms on various data sets.

The image processing algorithms classes are based on the Strategy design pattern [7]. This means that all algorithms are encapsulated and are interchangeable. For the developers of an algorithm, all that is required is to inherit from the common algorithm class and implement a single method. This makes the creation of new algorithms extremely easy.

```
17 bool Canny::execute(IplImage *in, IplImage *out){
18
19     if(gray == NULL){
20         gray  = cvCreateImage(cvGetSize(in), IPL_DEPTH_8U, 1);
21         edge  = cvCreateImage(cvGetSize(in), IPL_DEPTH_8U, 1);
22     }
23
24     cvCvtColor(in, gray, CV_BGR2GRAY);
25     int threshold1 = ((IntProperty*)getProperty("threshold1"))->getValue();
26     int threshold2 = ((IntProperty*)getProperty("threshold2"))->getValue();
27     int aperture_size = ((IntProperty*)getProperty("aperture_size"))->getValue();
28     cvCanny(gray,edge, threshold1, threshold2, aperture_size);
29     cvCvtColor(edge, in, CV_GRAY2BGR);
30
31     return false; // Les donnes ont ete copier dans out!
32 }
```

Figure 5 - Canny algorithm code snippet

Recording and playback

Machine vision can be difficult to test properly when used in underwater vehicles, especially since the communication with the vehicle is often impossible. Video recording and playback is therefore an essential requirement to the success of any mission. At any point in time, a video recording can be started of either the raw camera output or the output of any given filters in a machine vision algorithm. The XviD MPEG-4 codec is used as the encoding mechanism for

stream recording and playback [8]. It was chosen for its open source availability and excellent encoding capabilities.

The system was designed to allow many input sources such as Firewire cameras, MPEG-4 videos and PNG files. The input source is set at startup and the system loads the proper resources. Since all input sources implement the same interface, the system sees no difference, whether the raw data is coming from one source or another.

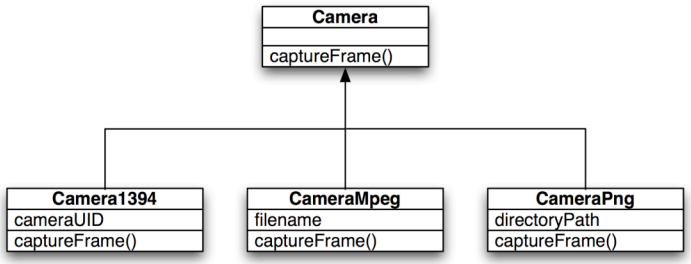

Figure 6 - Camera input design

Streaming

A vital requirement to the testing and configuration of algorithms is the capability of receiving live video from the vehicle when communication is possible. Live feedback is essential to rapidly validate the functionality of an algorithm. Video transmission is achieved by streaming the data over the network through the Real Time Streaming Protocol (RTSP). Implementation was done using the Live555 streaming media library [9]. The C++ library was simply integrated into our system as a stand-alone module.

Depending on the available bandwidth, the system can provide up to 3 live streams of 320x240 pixels at 5 frames per second (FPS). The latency on each stream is of about 500ms.

Vision client

Remote configuration, calibration, validation and monitoring are done centrally through a GUI-based client. This client connects to the vision framework and communicates using a proprietary text-based protocol. This allows the operator to intuitively:

- Enable/disable algorithms
- Configure/calibrate algorithms and cameras
- Monitor camera and algorithm output
- Create and modify machine vision algorithms visually

For each camera and algorithm that the user chooses to open, a panel displays a live stream of the selected source as well as a configuration panel to properly set the parameters for the given entity. As the user moves the sliders and sets values for configuration, he can monitor the changes in real-time.

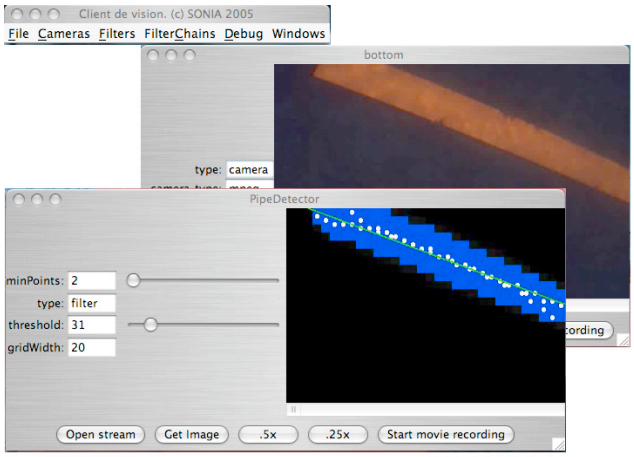

Figure 7 - Vision Client configuration and monitoring panel

A separate visual editor was developed to allow users to easily create and modify machine vision filter chains. Users can drag and drop new filters to a new or existing Pipes and Filters chain, using a What You See Is What You Get (WYSIWYG) user interface. This means that the content shown during editing is the same as the final result. Users don't have to tamper with any other external files. The created algorithms are automatically written to an XML file and loaded by the framework at runtime for self-enclosed execution within the machine.

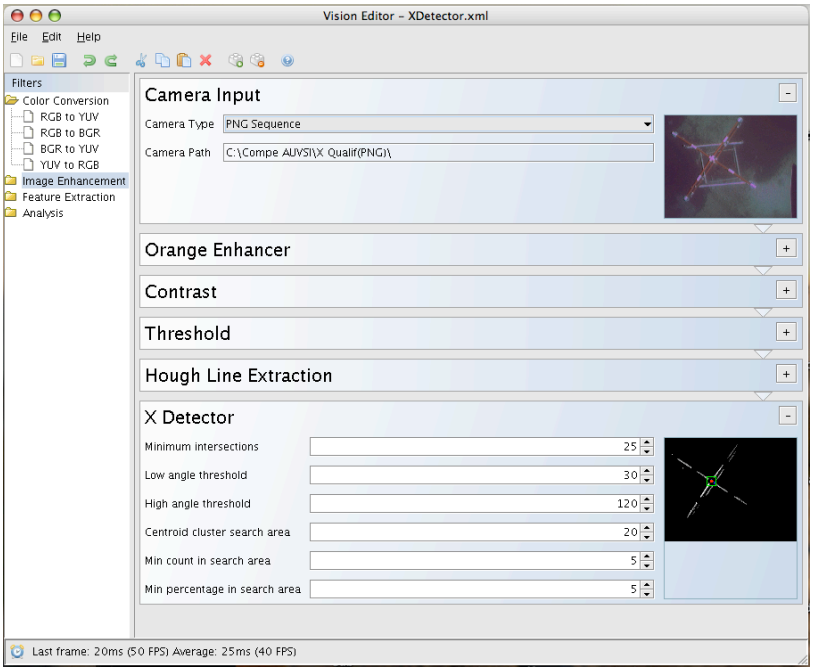

Figure 8 - Visual algorithm editor

Java technology was used for the development of the client. It was chosen for its simplicity and multi-platform capabilities. Since team members run on various platforms (Windows, Mac, Linux) it was an important design decision.

Future work

The vision framework being at a mature stage, developers are now concentrating their efforts on improving some aspects and adding new features.

We are constantly trying to improve the system's performances. Shorter response times lead to smoother navigation and therefore increase the success rate of mission accomplishment.

As the competition's mission become more complex and new objects need to be detected, more image-processing algorithms are being developed. This gives users the ability to create new and better detection algorithm using the visual editor.

The team is also evaluating the possibility of open-sourcing the framework. This would allow external communities to benefit from the framework's advantages and features and would allow the SONIA team to benefit from outside contributions.

Conclusion

Since the original implementation in 2005, the SONIA vision framework has given us a strong edge on all vision tasks in operation. It has limited the learning curve associated with machine vision by eliminating all the housekeeping tasks not related to the actual image processing. Developers no longer need to worry about image acquisition, remote configuration and monitoring. Vision development has therefore greatly improved and has become one of our main strengths. Furthermore, the vision client allows for simple and intuitive configuration of the machine vision system.

We believe that all of these features, together, provide the team with an ideal vision framework for AUVs and any other type of autonomous vehicles.

Acknowledgements

The SONIA AUV team would like to thank all of our sponsors for providing us with the tools and funding needed to build the AUV and its software. We acknowledge the amazing support we received from a large number of staff employees and students at École de technologie supérieure and from every team member of SONIA AUV. Thanks a lot to Pierre Bourque and his winter 2007 LOG410 class for the elaboration of some of the requirements, Sébastien Martin for the implementation and design of the Vision Client. Lastly, we would like to thank Tennessee Carmel-Veilleux for reviewing and editing this paper.

Contact information

SONIA AUV project
1100, Notre-Dame West, Office A-1320
Montréal, Quebec, CANADA
H3C 1K3,
http://sonia.etsmtl.ca

References

[1] Association for Unmanned Vehicle Systems International (AUVSI), & Office of Naval Research (ONR). (2007). *Final Mission Statement and Rules*. Retrieved from http://www.auvsi.org/competitions/water.cfm

[2] Bass, L., Clements, P., Kazman, R. (2006). *Software Architecture in Practice*, Addison-Wesley, 528 p.

[3] Intel Corporation, Open Computer Vision Library, http://www.intel.com/technology/computing/opencv/

[4] Carmel-Veilleux, T., Morissette, M., (2007), *SONIA 2007: Exploring the Depths with Ease*, SONIA AUV team, Journal Paper, Monteal, Canada. École de technologie supérieure. 10 p.

[5] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., (1996). *Pattern-Oriented Software Architecture*, Wiley, 467 p.

[6] Davies, E. R., (2005). *Machine Vision: Theory, Algorithms, Practicalities*, Morgan Kaufmann, 934 p.

[7] Gamma, E., Helm, R., Johnson, R., Vlissides, J., (1995). *Desing Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 393 p.

[8] XviD, http://www.xvid.org

[9] Live Network Inc., LIVE555 Streaming Media, http://www.live555.com/liveMedia/