Semi-Automatic Extraction of Information from Satellite Images

Ousmane Barry

Master Thesis
Stockholm, Sweden, April 2011

XR-EE-SIP 2011:004
Semi-automatic extraction of information from satellites images

Ousmane Barry

Supervisors

Pierre CARIOU Magellium
Jocelyn CHANUSSOT PHELMA
Markus FLIERL KTH

September 2010 – April 2011
ACKNOWLEDGEMENT

Firstly, I would like to thank my family which has always provided me support and motivation. Their presence and advice helped me to never deviate from my goals. Thank to my father Ibrahima BARRY, my mother Khadidiatou DIALLO who always fought for my education, my brothers and sisters: Mamadou, Diaryou, Dieynabou, Mamoudou, Alimou, Marlyatou, Saliou Dian, Bassirou, Telly and Néné Sadio. Thank you to my friend Jean Bernard for offering me accommodation at his place.

I also thank Markus FLIERL and Jocelyn CHANUSSOT for their help and advice throughout the internship.

I want to especially thank and express my gratitude to the following people for the rewarding experience and full of interest during the six (6) months within the company Magellium:

Eric PENNORS: for making me very welcome and reliance in entrusting me with this project.

Pierre CARIOU: my supervisor, for his guidance and advice throughout the internship, for his interest in my work, for his availability, for his good humor and finally for his confidence in the choices I had to do regarding the project.

Damien MAYER: for his contribution and advice in the project but also for his good humor.

Raphael KOURDIAN: for his advice in imaging, for his availability and the debates on recent events in his office.

Tarek HABIB: for his advice on technology platform and help on debugging.

And all the Magellium staff for their warm welcome, the good atmosphere and their professional cooperation.
ABSTRACT

This master thesis project deals with the semi-automatic extraction of information on satellite images. Some Geographic information systems (GIS) are dedicated to the issue of data production. The graphical user interface of these GIS is essentially passive, and only provides basic CAD tools for intelligence information mapping such as geometric and semantic capture of spatial objects and semantics improvement of geographic objects. As well as CAD software, they improve the operator productivity in certain limits that of ergonomics. Thus, by combining some generic image processing algorithms, we have implemented a component of semi-automatic extraction of features on satellite images. We gave a priority on the interaction between a user and the component. The user will be only focused on the interpretation of the images and the component will perform the repetitive task for him. The addressed features were suburban roads, hydrographic area boundaries and shorelines. This system based on powerful tools such as the Orfeo Toolbox (core of the) and Qt (for the GUI) has been tested on images from different satellites and the results are quite satisfactory. This opens perspectives to improve and optimize this system in the aim to integrate it into a GIS solution.
## Glossary

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITK</td>
<td>Insight Toolkit, C++ image library for medical imaging</td>
</tr>
<tr>
<td>OTB</td>
<td>Orfeo Toolbox, C++ image library for satellites images</td>
</tr>
<tr>
<td>GIS</td>
<td>System that captures, stores, analyzes, manages, and presents data with reference to geographic location data.</td>
</tr>
<tr>
<td>CNES</td>
<td>Centre National d'Études Spatiales → French Government Spatial Agency</td>
</tr>
<tr>
<td>CORE</td>
<td>Module which is the core of our system</td>
</tr>
<tr>
<td>DATA</td>
<td>Images and vector data</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>PP</td>
<td>PreProcessing Module</td>
</tr>
<tr>
<td>ROI</td>
<td>Region Of Interest</td>
</tr>
<tr>
<td>VEC</td>
<td>Vectorization Module</td>
</tr>
<tr>
<td>Qt</td>
<td>Pronounced « Cute », C++ library for application and user interface framework</td>
</tr>
</tbody>
</table>
## CONTENTS

1. Introduction .................................................................................................................. 1  
   1.1. Magellium ............................................................................................................... 1  
       1.1.1. Business line ................................................................................................. 1  
       1.1.2. Organization chart ......................................................................................... 2  
       1.1.3. Presentation of the reception structure or steering committee .................... 4  
   1.2. The context of the project ........................................................................................ 5  
   1.3. System overview .................................................................................................... 7  
       1.3.1. ITK/OTB: Core of our system ....................................................................... 7  
       1.3.2. Qt .................................................................................................................. 10  
       1.3.3. Future GIS Software ..................................................................................... 11  
2. Background and State-of-the-art .................................................................................. 12  
   2.1. Objectives ............................................................................................................... 12  
   2.2. Road Detection ...................................................................................................... 13  
   2.3. Hydrographic areas & Shoreline .......................................................................... 13  
   2.4. Conclusion ............................................................................................................ 14  
3. System architecture ..................................................................................................... 16  
   3.1. First level ............................................................................................................... 18  
   3.2. Second level .......................................................................................................... 19  
       3.2.1. CORE ............................................................................................................ 19  
       3.2.2. Visualization & Commands Module ............................................................. 21  
4. Our approach ................................................................................................................ 22  
   4.1. Preprocessing: Contour detection ......................................................................... 22  
       4.1.1. Anisotropic diffusion ...................................................................................... 22  
       4.1.2. Gradient calculation ...................................................................................... 23  
       4.1.3. Thresholding .................................................................................................. 25  
       4.1.4. Thinning ........................................................................................................ 25  
   4.2. Preprocessing: Road detection .............................................................................. 28  
   4.3. Vectorization ......................................................................................................... 29  
       4.3.1. Labeling and Extraction ............................................................................... 29  
       4.3.2. Edge Tracking with Dijkstra algorithm ......................................................... 30  
5. Software Architecture .................................................................................................. 32
5.1.1. Core .......................................................................................................................... 32

5.1.1.1. Preprocessing Module: Contour Extraction ......................................................... 32

5.1.1.2. Preprocessing Module: Road detection ............................................................... 33

5.1.1.3. Vectorization Module ......................................................................................... 34

5.1.2. GUI.......................................................................................................................... 35

6. Benchmark .................................................................................................................... 38

7. Roadmap & Perspectives ............................................................................................. 39

   A continuous vectorization......................................................................................... 39

   Ergonomics .................................................................................................................. 40

   Contour interpolation................................................................................................ 40

8. Conclusion ..................................................................................................................... 41

9. References ..................................................................................................................... 42

10. Appendix ..................................................................................................................... 44
Figure 1: Hierarchical, Functional and Operational Organization ........................................ 3
Figure 2: Paris R&D team project .................................................................................. 4
Figure 3: A system of vector data production ................................................................. 6
Figure 4: GIS Software ................................................................................................. 6
Figure 5: Signals & Slots Principle (Source: Qt Assistant) ............................................. 10
Figure 6: CORE - GUI Interactions ............................................................................... 16
Figure 7: System Overview ......................................................................................... 17
Figure 8: Interactions at the first level .......................................................................... 18
Figure 9: The CORE ..................................................................................................... 19
Figure 10: Implemented GUI ....................................................................................... 21
Figure 11: Detection with anisotropic filtering before .................................................. 23
Figure 12: Detection without anisotropic diffusion filtering ........................................ 23
Figure 13: Gradient image ............................................................................................ 24
Figure 14: Binary image obtained after threshold ......................................................... 25
Figure 15: Illustration of thinning operation ................................................................. 26
Figure 16: Sobel edge detector ..................................................................................... 27
Figure 17: Our edge detector method .......................................................................... 27
Figure 18: Canny edge detector ................................................................................... 27
Figure 19: Road's centerline detection ........................................................................ 28
Figure 20: Labeled image ............................................................................................ 29
Figure 21: Extracted features ....................................................................................... 29
Figure 22: Vectorization results ................................................................................... 31
Figure 23: Activity diagram of the Preprocessing for contour extraction ....................... 32
Figure 24: Diagram activity of Preprocessing for road detection .................................. 33
Figure 25: Diagram activity for vectorization ............................................................... 34
Figure 26: Path simplified with a tolerance value equal to 1 .......................................... 35
Figure 27: Path simplified with a zero tolerance value ................................................ 35
Figure 28: Class diagram for the output visualization and vectorization part ................. 37

Table 1: Input / Output of the System ........................................................................... 18
Table 2: Input/Output of the CORE ............................................................................. 20
Table 3: Distance from a central pixel .......................................................................... 20
Table 4: Execution time in seconds for Preprocessing Module ....................................... 30
Table 5: EdgeTracking with images from Preprocessing (iteration=5) ......................... 38
Table 6: Dijkstra with images from EdgeTracking ......................................................... 38
Table 7: Simplification of vertices from Dijkstra ........................................................... 38
Table 8: Vectorization rate ........................................................................................... 38

Equation 1: Anisotropic diffusion equation ................................................................... 22
Equation 2: Properties of the estimate of the location of boundaries ............................ 22
Equation 3: Thinning operation .................................................................................... 25
1. Introduction

1.1. Magellium

Magellium is a company of specialists on the leading edge of technology who leverage their expert knowledge in signal and image processing, geomatics, **innovative learning technologies, robotics and intelligent systems** to offer high value-added engineering, systems design, solutions and services. Magellium serves private and public-sector customers all over Europe and operates mainly in the sectors of **Defense & Security, Space, Health** and **Environment**. Magellium has two sites, one of which is located in Saint-Agne Ramonville near Toulouse and the other in Le Pecq near Paris.

Magellium Le Pecq is originally from the buyout in March 2009 of the branch of Defense of Générale d'Infographie by Magellium. Générale d’Infographie is an information technology consulting (IT consulting), a subsidiary of Veolia Environment and Vinci Energies, founded in 1991, specialized in developing and integrating business solutions based on Geographic Information Systems (GIS).

Magellium made this acquisition to strengthen its position in the defense sector; particularly in the fields of geography and intelligence at the heart of the strategic function "knowledge and anticipation" entered in the new Defense White Book. It also allows Magellium to have a presence in the Paris area near its customers and partners and continue its development in its various fields of excellence for the benefit of the defense sector and other strategic sectors of the company.

Magellium had already collaborated with the branch of Defense of Générale d’Infographie and been able to assess their skills for cooperating with them on various projects for several years. This acquisition brings the personnel of Magellium over 135 people (80 previously).

1.1.1. Business line

Like its slogan, "Innovation for a better tomorrow", innovation is placed in the heart of strategic management Magellium through different areas of imaging:

- Geomatics
- Engineering solutions and services
- Robotics and Intelligent System
Innovative technologies for training
Signal and image procession.

She responds efficiently to the needs of major public and private actors, both in France and abroad. Include among its partners and customers:
- CNES, DGA, IGN, ESA, ONERA
- EADS, EADS ASTRIUM, THALES, THALES Alenia Space, NEXTER
- SPOT IMAGE, CLS, Infoterra
- Capgemini, Communication & Systèmes
- The University hospitals (CHU) of Toulouse, Laboratoires Pierre Fabre, FCBM
- TOTAL, VEOLIA Transports.

In signal and image processing, the main areas of Magellium’s expertise offer: algorithm analysis – architecture and standardization expertise – software component design and distribution – image analysis and processing systems development – image quality control systems – systems integration. The principal applications are:
- Earth observation
- Medical and scientific imaging
- Real-time video
- Computer vision.

1.1.2. Organization chart

Through the organization chart (see next page), we can say that the organizational structure adopted by Magellium is a functional structure for the division of labor based on the various functions performed (Development, Production, Finance and HR). Thus, under the General Management, we have four divisions organized into services and/or departments. Business strategy and decisions adopted at the highest level of summits and directions are relayed and implemented by service managers. Different services may also be asked to cooperate on cross-cutting projects. The mode of communication is both vertical (hierarchical) and horizontal (between parallel services).
Figure 1: Hierarchical, Functional and Operational Organization
1.1.3. Presentation of the reception structure or steering committee

My internship is an upstream R&D project. Thus, I was in coordination with Toulouse with the R&D Innovation Department through Tarek HABIB: R&D Engineer in signal and image processing. He followed my work and gave me some ideas about the system architecture and the technology to use.

In Paris, we have set up a small R&D team as you can see in the following diagram:

![Diagram of the R&D team project](image)

**Figure 2: Paris R&D team project**

**Eric PENNORS**: Director of Paris GIS department was the project director. He leaded the project by establishing specifications and arbitrated decisions.

**Damien MAYER**: Project Manager in vector data production played the role of the customer. He expressed preferences of the client, thus helping to set up specifications. His views were more focused on the vectorization’s environment and ergonomics of the system.

**Raphaël KOURDIAN**: Project Engineer in GIS and remote sensing acted as a consultant giving advice on algorithms. I also take advantage of its expertise in remote sensing and knowledge of the images we could have to work with.

**Pierre CARIOU**: Assistant Project Manager was the chief project. He structured the project by defining the deadlines with my agreement. He assured the communication with Eric and Damien to keep them informed. He assisted the formalization of goals. He participated actively in the system architecture. I was able to share his expertise in project management and technical skills in GIS and image processing.

Every month, a steering meeting was set up with all the team members to showcase the progress of my work and set new directions if needed. And with Pierre, we had an information point each week. This proved to be gainful and efficient in the management of the internship and can be adopted for future internships.
1.2. The context of the project

Every day, a huge amount of images is provided by Earth Observation satellites. These images cover many fields:

- **Military**: decision making, strategy and terrain analysis
- **Natural Resources**: remote detection of oil or gas seepage, and exploration of mineral deposits
- **Agriculture**: identification of water or nutrient deficiencies, wind and hail damages, or plants populations
- **Institutions**: mapping areas affected by natural disasters (hurricanes, tsunamis, earthquake, etc) in order to help and organize rescue
- **Administration**: characterization and monitoring of agricultural parcels (farms, parks oysters, etc) for the control of subsidies

In the early years of remote sensing, most researches were oriented through semi-automated and automated methods in order to extract usable information on rough images. High resolution sensors brought a new paradigm in Earth Observation history, where objects are now defined by a group of pixels. In this context, extraction of linear objects (roads networks, water body (lakes, rivers), shoreline, bridges) or surfaces (buildings, vegetation, etc) in order to assist the production of geographic data using Geographic Information System (see illustration next page), require new approaches, where the operator can actively participate. Nowadays, this process is a fully manual operation and requires a lot of time and human resources. The subject of this project is at this level. It is motivated by to the identification of an operable approach that can provide assistance to the photo interpreter in the aim to increase his productivity. For 40 years, algorithms for information extraction in remote sensing images have been prolific. This work is organized as follows; first we give an overview of the developed system. In Section 2, we present the background and the state-of-the-art in this field and their limitations. In Section 3, we describe the system architecture. In Section 4, we present the approach that we developed to solve our problem. In Section 5, we describe the software architecture and present the choices made for the implementation to achieve the main goals. We finish the work with a benchmarking and an elaboration of a roadmap for the perspectives.
Figure 3: A system of vector data production

The figure is a rough representation of a GIS. Basically an operator via a GUI connects to a database of images and vector data. It downloads the data it needs to do its task and generate new data vector, maps, etc.…

Our project is involved in the CAD of the GIS. The goal is the enrichment of CAD tools with image processing algorithms to increase the operator productivity.

Figure 4: GIS Software
1.3. System overview

Our system consists of a part called CORE which encapsulate image processing algorithms and a GUI to control the CORE. This implies the use libraries: some for the implementation of image processing algorithms and some for the design and implementation of the GUI.

The following sections describe the tools used for our system. First, we will present the ITK, then the OTB with its relation with ITK, after we will present the framework Qt used for the visualization and the commands and finally we will discuss about the existing GIS software and their limitations for our needs.

1.3.1. ITK/OTB: Core of our system

In 1999 the US National Library of Medicine of the National Institutes of Health awarded six three-year contracts to develop an open-source registration and segmentation toolkit, that eventually came to be known as the Insight Toolkit (ITK) and formed the basis of the Insight Software Consortium. ITK’s NIH/NLM Project Manager was Dr. Terry Yoo, who coordinated the six prime contractors composing the Insight consortium. These consortium members included three commercial partners—GE Corporate R&D, Kitware, Inc., and MathSoft (the company name is now Insightful)—and three academic partners—University of North Carolina (UNC), University of Tennessee (UT) (Ross Whitaker subsequently moved to University of Utah), and University of Pennsylvania (UPenn). In addition, several subcontractors rounded out the consortium including Peter Raitu at Brigham & Women’s Hospital, Celina Imielinska and Pat Molholt at Columbia University, Jim Gee at UPenn’s Grasp Lab, and George Stetten at the University of Pittsburgh. In 2002 the first official public release of ITK was made available. ITK is developed through extreme programming methodologies; ITK employs leading-edge algorithms for registering and segmenting multidimensional data.

The ORFEO Accompaniment Program was set up, to prepare, accompany and promote the use and the exploitation of the images derived from the sensors of Pleiades (PHR) and Cosmo-Skymed (CSK) systems. The program was initiated by CNES mid-2003. In this context, CNES decided to develop the ORFEO Toolbox (OTB), a set of algorithms encapsulated in a software library. The goals of the OTB is to capitalize a methological savoir

1 Source : from the OTB Software Guide
faire in order to adopt an incremental development approach aiming to efficiently exploit the results obtained in the frame of methodological R&D studies. All the developments are based on FLOSS (Free/Libre Open Source Software) or existing CNES developments. OTB is implemented in C++ and is mainly based on ITK:

- ITK is used as the core element of OTB
- OTB classes inherit from ITK classes
- The software development procedure of OTB is strongly inspired from ITK’s (Extreme Programming, test-based coding, Generic Programming, etc.).

In addition to ITK, OTB depends on two others libraries:

- GDAL: which ensures the support of remote sensing imagery formats
- Fltk: used for the visualization functionalities.

OTB is a multi-platform software through the use of CMake, a cross-platform, open source build system. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of our choice (e.g.: GCC under Linux, Visual Studio under Windows, etc…). CMake is quite sophisticated and supports complex environments requiring system configuration, compiler feature testing, and code generation. The OTB consists of several subsystems and concepts that we are going to present and describe those we used for our system:

- **Generic Programming**: is a method of organizing libraries consisting of generic—or reusable—software components. The idea is to make software that is capable of “plugging together” in an efficient, adaptable manner. C++ templating is a programming technique allowing users to write software in terms of one or more unknown types $T$. To create executable code, the user of the software must specify all types $T$ (known as *template instantiation*) and successfully process the code with the compiler. The $T$ may be a native type such as `float` or `int`, or $T$ may be a user-defined type (e.g., `class`). At compile-time, the compiler makes sure that the templated types are compatible with the instantiated code and that the types are supported by the necessary methods and operators. The advantage of this approach is that an almost unlimited variety of data types are supported simply by defining the appropriate template types (see example at the next page).
**Example:** use of genericity to create any list of objects.

```cpp
template <typename T> /* Inspired from Wikipedia */
class MyListOf
{
    /* Implementation of methods for the class */
    MyListOf <Car> mylist_of_cars;
    MyListOf <Animal> mylist_of_animals;
}
```

- **Data Representation and Access.** For this purpose, two principal classes are used: `otb::Image` and `itk::Mesh`. The `otb::Image` class follows the spirit of Generic Programming, where types are separated from the algorithmic behavior of the class. OTB supports images with any pixel type and any spatial dimension. The `itk::Mesh` is used to represent geometry in the form of a set of points in n-dimensional space and provides the methods necessary to manipulate sets of point.

  **Example:** code needed to instantiate, declare and create the image class

  ```cpp
  #include "otbImage.h"
  typedef int PixelType; /* image with int pixel date */
  int Dimension = 2;
  typedef otb::Image <PixelType, Dimension> ImageType;
  ImageType::Pointer image = ImageType::New(); /* Create and assign the result to a pointer */
  ```

- **Data Processing Pipeline:** data objects (e.g., images and meshes) are used to represent data, *process objects* are classes that operate on data objects and may produce new data objects. Process objects are classed as *sources, filter objects, or mappers*. Sources (such as readers) produce data, filter objects take in data and process it to produce new data, and mappers accept data for output either to a file or some other system. Sometimes the term *filter* is used broadly to refer to all three types. The data processing pipeline ties together data objects (e.g., images and meshes) and process objects. The pipeline supports an automatic updating mechanism that causes a filter to execute if and only if its input or its internal state changes. Further, the data pipeline supports *streaming*, the ability to automatically break data into smaller pieces, process the pieces one by one, and reassemble the processed data into a final result.

  The OTB also includes smart pointers and memory management, error handling and exceptions, event handling, multi-threading, etc…
1.3.2. Qt

Once we had chosen the API for our image processing algorithms, we needed to control it through a GUI. The task was to choose a library which is cross-platform, intuitive, easy to learn and manipulate. There are many libraries that met these criteria: .NET, GTK+, wxWidgets, FLTK, etc… However our choice fell on Qt which is more a library. The choice was also motivated by the fact that it was a technology used in Magellium Toulouse. Indeed, Qt provides a powerful principle which is unique to Qt and constitutes one of its strengths: **signal** and **slot**. A signal is a message sent by a widget when an event occurs and a slot is the function that is called when an event occurred. The principle is illustrated in the following figure from Qt Assistant. Qt is cross-platform application and UI framework with:

- Intuitive C++ class library
- Integrated development tools with cross-platform IDE (**Qt Creator**)
- Create windows for applications with the mouse (**Qt Designer**)
- Translate programs in several languages (French, English, etc…) (**Qt Linguist**)
- A full documentation with all functionalities, classes and functions in Qt (**Qt Assistant**)

![Diagram](image.png)

**Figure 5: Signals & Slots Principle (Source: Qt Assistant)**
1.3.3. Future GIS Software

At the end of my internship, one question arose: How shall we integrate this project?

- Design and develop a GIS proper to Magellium or
- Integrate our system into existing GIS and benefit from their vectorization and editing environment. We will also benefit of their servers database of images and vector data.

The second option seems to be the best especially it is possible to integrate Qt applications in ArcGIS using ArcGIS Engine. For this purpose, it is worth to invest in a team gathering the following resources:

- A project manager to organize, guide and direct the team.
- An engineer in architecture that will bring solutions on how to integrate the developed system in the chosen GIS Software.
- Someone who will think of the ergonomics of the system and valorize it next to the customers.
- An engineer in image processing with skills in informatics. His goals will be to find new algorithms in image processing that offer a good interaction with the operator.
- And finally an engineer who will optimize the implementations and processing for a fast, efficient and performing system.
2. Background and State-of-the-art

Here, we will define the project objectives and present the state-of-the-art in relation to the objectives.

2.1. Objectives

The roadmap can be divided into three (3) parts

- **Identification of automation** in order to assist and help the photo-interpreter. In one hand, this study focused on methods of image processing and the other hand on GIS solutions. The latter aimed to study “on-the-shelf” software dealing with our problems and see if one of these technologies is appropriate for integration in our prototype.

- **Design** of a HCI for the production of geographical data in a semi-automatic way.

- **Realization** of the HCI to validate the performance and functionalities. The system must be able to increase the **productivity** of photo-interpreter. Thus, a good supply from the human-computer interaction is expected.

The state-of-the-art allowed us to identify existing extraction methods for satellite images and the types of features extracted. We focused it on the following features: road networks, hydrographic areas and shoreline. The main goal is to implement a system that will be able to detect and extract these features in interaction with an operator. Here are the top-level system requirements:

- **A semi-automatic scheme**: that means a good interactivity with the operator and the system.

- **Good accuracy** of boundaries of extracted features: we have to match as close as possible to the geometry and shape of the objects.

- **Speed**: to increase the productivity of the photo-interpreter in his task of data scratching and made all the operations turning in background transparent for him.

- **Ergonomics**: the system has to offer an interface easy to understand and manipulate by even a non-expert in photo-interpretation.

- **Images heterogeneity**: a system that will work with a large range of satellites images which is very challenging.

The next sections will present the state-of-the-art for the desired features to extract.
2.2. Road Detection

Geman and Jedynak, 1996 [1] proposed a method of tracking roads tested on SPOT images given a starting point and starting direction. The algorithm combines entropy filtering and decision tree rule. Steger, 1996 [2] presented a method to extract linear structures and their widths from digital images and it is applied to the roads. His method combines edge detection and Hessian matrix to determine the roads direction. Rianto et al., 1998 [3] developed an algorithm that is independent of the size of the image combining several techniques such as Canny Edge Filtering [4], edge parallel detection, Hough transform and optimal graph searching. Katartzis et al. 2001 [5] described a model-based method for automatic extraction of linear features from airborne images using mathematical morphology and Markov Random Fields (MRF). Their work is a combination and extension from two approaches [6], [7] for road detection in Synthetic Aperture Radar (SAR) satellites images. In [8], Udomhunsakul presented a semi-automatic way to detect roads in low resolution imagery by means of the “à trous algorithm”, Principal Component Analysis (PCA) and a graph searching. Miriam Amo et al [9] presented a semi-automatic scheme for road extraction where the operator guides the algorithm by giving seed points based on the radiometry and curvature of the roads. Their algorithm used different techniques: region growing, region segmentation, deformable models, B-splines. Some others methods combine mathematical morphology, MRF with Watershed transform and graph searching [10]. Active contours (snakes) are also very used for edges and roads in detection and tracking [11], [12]. In [13], [14], an object approach combined with probability theory is used to detect roads, intersections and buildings. Objects features such as descriptors, signatures, etc… are extracted from a set of training images and fed to a classifier for learning. The most popular classifiers for the learning are: Support Vector Machine (SVM), neural networks, boosting, Hidden Markov Models (HMM), Expectation Maximization, Bayes classifier.

2.3. Hydrographic areas & Shoreline

As in road detection, a significant amount of research has been conducted in the aim to extract hydrographic area (water body). In [15], Liu et al. proposed a method based on classification using a coupled network with adaptation. Their algorithm was tested on digital orthophoto quarter-quadrangle (DOQQ) and the water bodies were extracted with accurate boundaries. In [16], a neural networks classification is used to extract rivers from multispectral images. Aijun Chen [17] used a fuzzy classification and a clustering analysis to
Zhang et al. [18] proposed an entropy-based method for water body extraction from multi-source satellite images under the two following assumptions that the water body is a smooth area, and the water body expands upon a large area. Kumar et al. present in [19] an introduction and a state-of-art in the field of water bodies’ extraction from satellite images.

Concerning shorelines, the results from water body’s extraction can be used for its extraction by means of mathematical morphology operations. In [20], Jishuang et al. proposes a threshold-based morphological approach for extraction of coastal line feature in remote sensing images.

2.4. Conclusion

The major issue of the algorithms and methods developed consists in their incapacity to handle the whole kind of satellite images. Most of them have been developed for a specific sensor, or resolution... In addition, the detection and extraction of objects or structures can differ from one image to another depending on whether the image is affected by:

- Illumination variations
- Occlusions
- Scale variations
- Deformations of objects
- Problems induced by the background of the image.

Some methods have a generic approach using a preprocessing step to segment features and a post-processing step to handle the issues listed above. The best solutions are the ones including the human in the operations for supervision in the sense of guiding the system and providing the good parameters to a semi-automatic system combining several methods for a good accuracy, speed and robustness in the extraction of features. For our system, we decided to make it semi-automatic and to combine several generic image processing algorithms inspired from the state-of-the-art to achieve our goals.

In our study, we included two existing GIS software: ITT Envi and Erdas Imagine. For both of them, it is possible to develop applications though their development toolkit. For the first one, we would have needed to have an interface in C language to link IDL and C++ programs. Thus, learning the language IDL was necessary. Envi offers technologies path called bridges to link IDL with another programming languages:
- **SPAWN**: to access programs (executable) external to IDL and can be very slow when a large amount of data is exchanged.

- **CALL_EXTERNAL**: load and call programs contained in shared object libraries. IDL and the called program share the same memory space. **CALL_EXTERNAL** is easier to use than the other system routines and is often the best way to communicate with other languages. However, coding errors can easily corrupt the IDL program. You must have a good understanding of system programming, compiler and linker because IDL does not initialize any necessary C++ runtime code.

**Erdas** also offers a large panel of tools for the analysis and processing of geospatial images. **Erdas Imagine** developed for the information extraction in satellite images, is one of them. It integrates some products like **EasyTrace** for assisted vectorization or **DeltaCue** which is a set of algorithms and tools to assist the photo interpreter for the analysis and detection of change. Contrary to Envi, we can direct connect C++ programs in **Erdas**. They developed a **C++ toolkit** for this purpose in the aim to increase the performance and the speed. Thus, we are able to customize **Erdas Imagine** tools and integrate new algorithms. The downside is that this is not the technology **used (or selected)** by Magellium, but this can evolve in the future.

In our study, we also included **INGRID**, software developed by Magellium to allow evaluation, test and manipulation of image processing chains. But for now, the API written just in C (an interface will be needed to link it with C++) does not allow us to solve our problem because it will not be possible to have the interactions that we want between the operator and the system. Here, we place blocks (image algorithms), connect them, enter the parameters and execute the process. So, the control in the process is reduced and the user has to be a bit aware on image processing to set the parameters.
3. System architecture

The proper functioning of a system, its robustness, its performance and its modularity go through a good design of the architecture. Thus, we decided to put in place a design phase of the architecture of our system. The goals were to:

- identify the functionalities of the system to meet the needs of the client (an operator in our case) and to ensure the project monitoring,
- identify modules supporting these features,
- formalize the interactions between modules.

The system is divided into several modules and interactions between them are described. The architecture will take into account the modularity to allow us to facilitate:

- adaptation to new needs and requirements as part of an evolution of the system,
- changes, substitution and integration of new modules as part of a future development,
- testing and maintenance.

Thus, the system must be able to easily integrate new modules to address new problems of extraction and meet new requirements from the customer.

The system is divided into two (2) modules: a module of heart (CORE) consisting of two sub-modules: the “Preprocessing Module (PP)” and “Vectorization Module (VEC)”, and a module for “Visualization & Commands” will be designated by the acronym GUI. CORE is intended to encapsulate plugins, which are delivered with the system as a C++ API. As we can see from Figure 4, there will be an exchange of data between the CORE through its two (2) sub-modules and the GUI. Therefore, the establishment of an interface between two modules is required. This communication interface can be seen as a process of converting data formats between the two modules. The heart of our system is based on the OTB, which offers a range of important image processing algorithms implemented and the possibility to easily integrate other according to our problem. It is important to emphasize that the CORE is delivered with white plugins to be able to work 100% in manual mode from a terminal or by running a script.

Figure 6: CORE - GUI Interactions
Figure 7: System Overview

The GUI provides commands to the Preprocessing and Vectorization module as:
- Mask or Coordinates
- List [values, intervals]
- Inputs of algorithms

The Vectorization module provides the vector data to the GUI to be visualized and helps the operator to judge about the quality.

The preprocessing outputs N-tuple images which are provided to the visualization and vectorization modules.
3.1. First level

Figure 8: Interactions at the first level

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>CORE</th>
<th>GUI</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF-1-01: Images</td>
<td>I/O</td>
<td>I/O</td>
<td>O</td>
</tr>
<tr>
<td>IF-1-02: Vector data</td>
<td>I/O</td>
<td>I</td>
<td>I/O</td>
</tr>
<tr>
<td>IF-1-03: Parameters</td>
<td>I</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Input / Output of the System

**IF-1-01: Images**

The images come from DATA and are in GeoTIFF format. Panchromatic or multispectral images can be processed. They feed the GUI for viewing and the CORE for algorithmic processing. With OTB, it is possible to read a list of images in series and concatenate them into a vector image. But initially, we will only process a single image as input of the CORE. Currently DATA is a file system, later it will be images and vector data servers.

**IF-1-02: Vector data**

Vector data from data are used as input for the CORE and the GUI. Data vectors will be produced by the CORE. Production data vectors will be in a semi-automatic supervised by an
operator though the GUI. Here, we also use the tools of the OTB. Indeed, it has a "reader" of DXF files, and a "reader / writer" of Shapefile.

**IF-1-03: Parameters**

The input parameters and commands will be done by an operator from the GUI. They will take the form of sliders for varying values and automatically see the result of processing algorithms through the visual part of the GUI. These values will be C++ float and will be passed directly to the CORE as input to the algorithms. Thus, the operator can play with these sliders to get a result it deems satisfactory without worrying about algorithms running in the background. The operator can define work areas (see next section) on an image with the mouse. Initially, we restrict ourselves to rectangular areas.

**IF-1-04: Refer to the GUI**

**Commands**

This diagram shows how the operator interacts with the system through its various modules and interfaces. The operator loads an image from DATA in the GUI, then he will choose to perform the desired processing and enter the appropriate parameters via the GUI, then the commands and parameters are retrieved and forwarded to the CORE, after that the corresponding algorithm’s executes and returns the processing results to the GUI that displays for the operator. Thus, it is a loop with multiple iterations.

### 3.2. Second level

#### 3.2.1. CORE

![Diagram of the CORE](image)

**Figure 9: The CORE**
**Tableau 2: Input/Output of the CORE**

<table>
<thead>
<tr>
<th>CORE</th>
<th>PP</th>
<th>VEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF-2C-01: Images</td>
<td>I/O</td>
<td>I</td>
</tr>
<tr>
<td>IF-2C-02: Mask</td>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>

CORE consists of modules PP and VEC. The first takes as input an image in GeoTIFF format. It is controlled by the parameters that the operator will select, and outputs **N-tuple images** (IF-2C-01) for the vectorization module. This **N-tuple images** is a set of images to be stored in a directory. Remember that it is the operator who decides to move to the vectorization stage when it deems satisfactory the preprocessing work. The masks are a region of interest (work area) to be processed by the CORE. Because these areas of work will be defined by rectangles, they are fully described by four (4) coordinates: $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$ stored in C++ float. This region is defined by the operator. VEC modules whose inputs are the outputs of the sub-module PP and also controlled by the parameters but also by vector data from the manual input of the operator (in case of adjustment of vertex entered by hand, etc ....). It outputs a vector file by putting together all the vectors generated by work area of the input image.
3.2.2. Visualization & Commands Module

This figure is the implemented GUI which will be described in the rectangle in white. This latter is reserved for the output image after processing. All the vectorization tasks are also performed in this area.

---

**Figure 10: Implemented GUI**
4. Our approach

In this part, we will describe in details the retained approach to achieve our objectives. We have two (2) approaches for the Preprocessing regarding the features we want to extract. The first one is designed to extract contours that bound hydrographic areas (lake, rivers, etc...) and shoreline and the second one, to extract a centerline of a road. The Vectorization is common to the two Preprocessing approaches. All the figures shown in this section are from Matlab. We implemented and tested all the algorithms in this environment before integrating them in our final system.

4.1. Preprocessing: Contour detection

4.1.1. Anisotropic diffusion

In our approach, we are interested in the contours that bound our surface features. Then before the edge detection, it will be very helpful to enhance the contours of the images. For this purpose, we decided to use anisotropic diffusion. This latter is a technique aiming at reducing image noise without removing significant parts of the image content, typically edges, lines or other details that are important for the interpretation of the image. It’s an adaptive smoothing filtering (noise removal) having the property of feature preservation [21]. For our system, we will use the algorithm developed by Perona and Malik [22]. It has the following properties which make it suitable for our problem: rotation invariance, avoidance of blurring artifacts (dissipativity), and accuracy. The following equation is the anisotropic diffusion equation:

\[ I_t = \text{div}(c(x, y, t)\nabla I) = c(x, y, t)\Delta I + \nabla c \cdot \nabla I \quad (1) \]

Equation 1: Anisotropic diffusion equation

where \( I \) is the input image, \( c(x, y, t) \) the conduction coefficient, \( \text{div}, \nabla, \Delta \) are respectively the divergence, gradient and Laplacian operators. The goal is to smooth within regions than across edges to maintain region boundaries sharp. So, an appropriate estimate of the location of boundaries at a scale \( t \) is needed. The estimate is defined as a vector-valued function on the image with the following properties:

1) \( E(x, y, t) = 0 \) in the interior of each region.

2) \( E(x, y, t) = Ke(x, y, t) \) at each edge point.

Equation 2: Properties of the estimate of the location of boundaries

where \( e \) is a unit vector normal to the edge at the point, and \( K \) is the local contrast of the edge.
The availability of an estimate allows choosing the conduction coefficient to be a function \( c = g(|E|) \) of the magnitude of \( E \), where \( g(.) \) is a nonnegative monotonically decreasing function with \( g(0) = 1 \). Thus, the goal is achieved because the diffusion process will mainly take place in the interior of regions and the edge boundaries where the magnitude of \( E \) is large will not be affected. They show that the simplest estimate of the edge positions, the gradient of the processed image at the scale \( t \), i.e., \( E(x,y,t) = \nabla I(x,y,t) \), gives excellent results. The literature has show that \( c(x,y,t) = e^{-\left(\frac{(|\nabla I(x,y,t)|^2)}{2\kappa^2}\right)} \) is quite effective\(^2\).

The next two following figures show the power of smoothing of the anisotropic diffusion. For Figure 10, we use the water body extraction based on entropy from [18] directly on the input image. For the Figure 9, the input image is smoothed with the anisotropic diffusion.

![Superposition of the input image and the binary image after minor region suppression and holes filling](image1)

![Superposition of the input image and the binary image after minor region suppression and holes filling](image2)

Figure 12: Detection without anisotropic diffusion filtering  
Figure 11: Detection with anisotropic filtering before

Even though the detection is not perfect in both cases, it appears clearly that the detection with a filtering step provides better results. The large water body area is detected with a good shape’s match with respect to the geometry. We will show the edge enhancement in the next method using the implemented gradient calculation.

### 4.1.2. Gradient calculation

The gradient calculation consists of convolving the input image with a kernel or gradient operator. The aim is to detect the abrupt changes in intensity in the image and can be used to extract edges. In a 2D-image, there is a gradient operator for each direction (x and y) and the gradient image is obtained by computing the gradient magnitude.

---

\(^2\) Source of the formula: OTB Software Guide
Let’s consider the two 3x3 gradient operators and I(x,y), the input image:

\[
H_x = \begin{pmatrix}
    h_{x11} & h_{x12} & h_{x13} \\
    h_{x21} & h_{x22} & h_{x23} \\
    h_{x31} & h_{x32} & h_{x33}
\end{pmatrix}
\quad \text{and} \quad
H_y = \begin{pmatrix}
    h_{y11} & h_{y12} & h_{y13} \\
    h_{y21} & h_{y22} & h_{y23} \\
    h_{y31} & h_{y32} & h_{y33}
\end{pmatrix}
\]

We have:

1) \( \text{Grad}_x(x,y) = I(x,y) \ast H_x \)
2) \( \text{Grad}_y = I(x,y) \ast H_y \)
3) \( \text{Grad}(x,y)_{I(x,y)} = \sqrt{\text{Grad}_x^2 + \text{Grad}_y^2} \)

There are different gradient operators and the most used are: Prewitt, Robert and Sobel gradient operators. During the last fifteen (15) years, some alternatives operators have been developed by Scharr et al [24]. These filters are shown to approximate rotation invariance better than the previous filters. Thus, for our system we used the following gradient operators developed by Scharr:

\[
H_x = \frac{1}{32} \begin{pmatrix}
    -3 & 0 & 3 \\
    -10 & 0 & 10 \\
    -3 & 0 & 3
\end{pmatrix}
\quad \text{and} \quad
H_y = \frac{1}{32} \begin{pmatrix}
    3 & 10 & 3 \\
    0 & 0 & 0 \\
    -3 & -10 & -3
\end{pmatrix}
\]

At the end of the section 4.1.4, we will compare our edge detection method using these gradient operators with Sobel edge detector and Canny edge detector [4].


4.1.3. Thresholding

From the *Figure 11*, we see clearly that the gradient image is composed of light pixels represented desired edges on a dark background. We want to extract the light pixels from the background and the simplest way to do so if to select a **threshold** value \( T \) that separates the two types of pixels. Thus, any pixel value \( I(x, y) \geq T \) is set to 1 and any pixel value \( I(x, y) < T \) is set to 0, to produce a binary image in the aim to show the operator where the edges to extract are located. This method is called global thresholding.

![Binary image with threshold T = 41](image)

*Figure 14: Binary image obtained after threshold*

We can see that the desired edges are detected with some artifacts. However, the operator cannot work on this image; we need to perform a last operation which is **thinning**.

4.1.4. Thinning

The *Figure 12* shows edges with more than 2 pixels of thickness. For accuracy and good extraction, the edges should be one pixel thickness. That’s in this point that the thinning operation intervenes. This latter is a morphological operation and implies the use of structuring element. For thinning a set of structuring elements \( \{B\} = \{B^1, B^2, B^3, ..., B^n\} \) [25] is used. First the binary image is thinned by one pass with \( B^1 \), and then the result is thinned with \( B^2 \), and so on, until it is thinned one pass with \( B^n \). The process is repeated until no further changes occur.

\[
\text{thin}(A, B) = \text{thin}(\ldots (\text{thin}(A, B^1), B^2)) \ldots , B^n)
\]

*Equation 3: Thinning operation*
The following structuring elements are commonly used: (origin)

\[
B^1 = \begin{array}{cccc}
0 & 0 & 0 & X \\
1 & 1 & 1 & 1 \\
\end{array} \quad B^2 = \begin{array}{cccc}
1 & 1 & 0 & X \\
1 & 1 & 1 & 1 \\
\end{array} \quad B^3 = \begin{array}{cccc}
1 & 1 & 0 & X \\
1 & 1 & 1 & 1 \\
\end{array} \quad B^4 = \begin{array}{cccc}
1 & 1 & 0 & X \\
X & 0 & 0 & 0 \\
\end{array} \\
\]

Algorithm:

- The origin of each structuring element is translated to all foreground points (pixels with value 1) in the binary image,
- And then the structuring element is compared with the underlying binary image pixels.
  - If the foreground and background pixels in the structuring element exactly match foreground and background pixels in the binary image, then the pixel underneath the origin of the structuring element is set to 1.
  - If it doesn’t match, then that pixel is set to 0. The ‘X’ means don’t care pixels.

Figure 15: Illustration of thinning operation

It appears clear that the binary image after thinning is the best one for our problem and will be used for the visualization to help the operator choose the good points for the vectorization.
The three figures are presented to compare our method (anisotropic filtering → gradient calculation → global thresholding → thinning) with Sobel and Canny edge method implemented in Matlab. The Sobel method shows some gaps in the desired contour in the binary image, these gaps don’t exist in ours and Canny method. In the Canny method, the edges are well detected but with lot of artifacts that can slow down our algorithms during the vectorization process. Thus, our method allows us to well detect desired edges while suppressing the major of artifacts.
4.2. Preprocessing: Road detection

Roads are linear features delimited with parallel lines or curves. Our goal is to extract the centerline of the roads and vectorize it. For this purpose, we used the preprocessing step in [26] to detect the centerline of the road. Let’s denote \( I \) the input image with intensities values in the interval \([0, 255]\). First, the input image is normalized to produce \( v = I/255 \), so that \( v_p \in [0,1], p \in \Omega \). Considering that a road has a contrast in both sides, they proposed to estimate this property using the following filter:

\[
\forall p \in \Omega, \tilde{n}_p = \max_{q \in O_1(p)} \left[ \min(v_p - v_{p+q}; v_p - v_{p-q}) \right],
\]

with \( O_1(p) = \{ q = (q_i, q_j) \in \Omega: \max(|p_i - q_i|, |p_j - q_j|) < l \} \)

The parameter \( l \) corresponds to the maximum width of roads to be detected. So for \( l = 2 \), roads assuming that roads are bright lines on a dark background, they consider the following contrast term:

\[
n_p = \max\{\tilde{n}_p, 0\} \text{ and } \{n_p\} = \text{contrast image}.
\]

At this step, we decided to use an Otsu Thresholding [23] on \( n_p \) instead of continuing the preprocessing algorithm in the paper. We obtained quite satisfying results:

![Figure 19: Road’s centerline detection](image-url)
4.3. Vectorization

The Preprocessing allows us to locate and detect the desired features. The next step is to extract these features and vectorize them with polylines.

4.3.1. Labeling and Extraction

First, we label the binary thinned image from the Preprocessing to be able to extract all 8-connected components (representing desired edges) in the binary thinned image. The labeling is a process hidden to the operator. The following figure shows the results of a labeling on the image used for illustration:

Every 8-connected object is represented with a different color (or value). Thus, it becomes easy to extract the desired object. The operator just has to click on a point on the desired object and all the object will be extracted for the next step.

Let’s extract, the object pointed out by the arrow. All we need is a point on this object, then we retrieve the value of his label and finally we pick up all the pixels with this corresponding value (see Figure 19).

As we can see only the desired object is extracted. However, it presents some artifacts. The next step will be to vectorize this feature while suppressing the artifacts.
4.3.2. Edge Tracking with Dijkstra algorithm

Once the desired feature is extracted, we have to vectorize it. For this purpose, we used the Dijkstra algorithm. It is used in graph theory to solve the problem of finding the shortest path between two nodes in a graph. In our case, we considered the previous image as our graph and the nodes are the pixels of the extracted object. Thus, we have an undirected connected graph. The following table shows the distance (arc length) between a central pixel and its eight (8) neighbors (8-connexity). A path on this graph is by the usual definition of graph theory, a subset of directed graph nodes connected by arcs consecutively. Reported to our image plane, a path is a set of pixels forming a connected curve joining a starting point to an end point.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>( \sqrt{2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \sqrt{2} )</td>
</tr>
<tr>
<td>( \sqrt{2} )</td>
<td>0</td>
<td>1</td>
<td>( \sqrt{2} )</td>
</tr>
</tbody>
</table>

Table 3: Distance from a central pixel

**Algorithm**:  

We have a **starting point** (starting node) called X and an **end point** (end node) called Y.

1. We assign to every node (pixels representing the object) a distance value: set it to zero for our starting node and to infinity for all other nodes.
2. We mark all nodes as unvisited and set starting node as current.
3. For current node, we consider all its unvisited neighbors and calculate their tentative distance. For example, if current node (A) has distance of 6, and an edge (arc) connecting it with another node (B) is 2; the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance, we overwrite the distance.
4. When we are done considering all neighbors of the current node, we mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. If all nodes have been visited, the algorithm has finished. Otherwise, we set the unvisited node with the smallest distance (from the initial node, considering all nodes in graph) as the next "current node" and continue from step 3.

---

The next figures show the results of the extraction followed by the vectorization:

Figure 22: Vectorization results

In green, we have the features extracted with the edge tracking using Dijkstra algorithm. We also have the polylines which are the set of vertices connected by black lines. The first path is composed of the coordinates of pixels extracted by the algorithm of Dijkstra. This path is simplified because it has a lot of vertices which do not provide any information; we will come later on this simplification. The results are quite satisfactory. However, the algorithm has some drawbacks. Given that it chooses the shortest path between the start and end points, it happens that sometimes it cuts through the field and does not stick well to the real contours. This can be solved by tuning the parameters of the algorithms like the threshold values or the number of iterations for the anisotropic diffusion.
5. Software Architecture

ITK is the core of OTB, thus all OTB classes inherit ITK classes. We apply the same principle for our system having the possibilities to inherit classes of both systems.

5.1.1. Core

5.1.1.1. Preprocessing Module: Contour Extraction

The Preprocessing module is a composite filter that takes as input an image and outputs also an image. It inherits `itk::ImageToImageFilter` which is the base class for filters that take an image as input and produce an image as output. There is a link of association between the three main filters used to perform operations. I’m also using other filters by association to rescale, normalize the image, etc... We used the generic programming to implement the Preprocessing module. It is a class composed of three (3) files: a header file `.h`, a template file `.txx` where all methods are implemented and a non-templated file `.cxx` to generate an executable. In the first implementation, the Preprocessing filter provided four (4) images as outputs: the anisotropic, gradient, threshold and thinned images. At first, these images were displayed to the user and were used for Vectorization. However, it appeared that only the thinned image was necessary and the filter was changed accordingly.
5.1.1.2. Preprocessing Module: Road detection

Figure 24: Diagram activity of Preprocessing for road detection

This sub-module of the Preprocessing is simply implemented as an executable (.cxx file) even though it uses multiple filters. This is due to several reasons:

- It is the latest algorithm implemented and we were short on time.
- We wanted to test at the same time if our system was modular that is to say that we were able to add new algorithms with ease.
- Finally, there was no reason to create a class for this algorithm even if it should be the best approach in the context of genericity.

It turned out that our system is modular. Indeed, the algorithm was first tested and validated in Matlab. Then it took us just a morning work to implement it in C++ and integrate it into our system.
The Vectorization module consists of two (2) classes: **EdgeTracking** and **Dijkstra classes**, and an executable **SimplifyPath**. The classes are implemented using genericity offered by OTB. The **Dijkstra class** inherits **itk::ImageToImageFilter** which is probably not the best approach. It should inherit **otb::ImageToPathListFilter** which is the base class used to implement filters that have an image as input and a paths list as output. Thus, the **SimplifyPath** can be directly included in the **Dijkstra** class. Currently, it takes as input the **.txt file containing the extracted vertices**, simplify it and return the path in a **.txt file** too.

As mentioned above, the first path obtained from the **Dijkstra** algorithm need to be simplified because it contains lot of vertices. For this purpose we used the filter **otb::SimplifyPathListFilter**. It reduces the number of vertices in each path, according to a **tolerance criterion** provided by the operator. It aims at removing aligned vertices while keeping sharp angular points. In order to ensure the unicity of its output, each path is considered first from begin to end, and then from begin to the first vertex before the end. At each step, the consistency of the path is checked: the equation of the line passing by the first and last vertices is computed. Then, for each vertex between them, the euclidean

---

4 Source: Doxygen documentation of OTB
distance to this line is computed. If for one vertex, this distance is upper than the tolerance threshold, the path is considered to be inconsistent and no vertices can be removed. If the path is considered consistent (which will occur at least with a 2 vertices path), only the beginning and ending vertices are kept and new search iteration begin at its end.

The following two (2) figures show the effect of this filter:

![Figure 26: Path simplified with a tolerance value equal to 1](image1)
![Figure 27: Path simplified with a zero tolerance value](image2)

### 5.1.2. GUI

The **GUI** is one of the important parts of the system. It allows us to display images, to control **CORE** algorithms by providing them the good parameters set by the operator and display their results for this latter. The main window is designed by **Qt Designer**. It inherits **QMainWindow** which provides a framework to build an application's user interface. **QMainWindow** has its own layout to which we can add menu bar, toolbars, dock widgets and status bar.

The **GUI** manages all the orchestration and synchronization of all operations. Event filters are installed on various widgets (buttons, view area, sliders, spin boxes, etc…). Each time an event (click for example) occurs on these widgets, a **signal** is emitted by the widget, and it intercepts the signal and delegates it to the appropriate function (**slot**) in charge of processing the signal. The communication with the **CORE** is made by calling executables and exchanging files. It is clear that this is not the optimal way of doing things. In fact, the OTB generates shared object libraries after its building and **Qt** provides classes that allow loading dynamic libraries at runtime. In our case, we first wanted to implement a "basic" functional
system and optimize it later. The implemented GUI is the one shown and explained in Figure 8.

The Figure 26 shows the class diagram for the output view. It is linked to the main window by association. The View, Node and Edge classes are the ones implemented for an example in Qt. We adapted and modified it for our problem. We also added two others classes to manage the vectorization. Indeed, the vectorization is done on image areas (ROI). Thus, we must be able to assemble all the vertices with their connections to each other at the end to cover the entire image. The NodeBuffer class manages the vectorization in ROI and the class NodeStack contains all the vertices of the entire image. The full mechanism to extract contour in an image is illustrated in appendix.
Figure 28: Class diagram for the output visualization and vectorization part
6. Benchmark

Let’s remind that the purpose of this project is to improve the operator’s productivity by providing him an ergonomic, robust and efficient system. In this context, we decided to test our system on its rate (number of vertices / second) of vectorization. We made the following assumption: an operator places 2 vertices / sec. These tests give an overview of the processing speed of our system. The tests were conducted on a single image partitioned in three sub-images of size: 256x256, 512x512 and 948x1024. Each algorithm is run N = 10 times on the same sub-image with the same parameters and at each execution the execution time is estimated. This is followed by an averaging to get the execution time. The results are presented in the following tables:

Table 4: Execution time in seconds for Preprocessing Module

<table>
<thead>
<tr>
<th>Image Size Iterations</th>
<th>256x256</th>
<th>512x512</th>
<th>948x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.528</td>
<td>1.973</td>
<td>12.223</td>
</tr>
<tr>
<td>5</td>
<td>0.583</td>
<td>2.654</td>
<td>15.312</td>
</tr>
<tr>
<td>10</td>
<td>0.748</td>
<td>2.921</td>
<td>18.113</td>
</tr>
</tbody>
</table>

The threshold value is fixed to 30 and the tolerance value for the path simplification is set to 1 for the whole tests.

We see clearly that the vertices extraction with Dijkstra algorithm consumes about 91% of the total time processing. So, its implementation should be reviewed and optimized to reduce its execution time.

For images of size 256x256 and 512x512, our system outperforms the operator. Thus, in this case, the productivity of this latter is increased. An optimization of the implementation of the Dijkstra algorithm should allow us to increase the vectorization for images with bigger size.

However, tests should be run on a large amount of images data in an operational context to obtain consistent conclusions.

Table 5: EdgeTracking with images from Preprocessing (iteration=5)

<table>
<thead>
<tr>
<th>Image Size</th>
<th>256x256</th>
<th>512x512</th>
<th>948x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration (s)</td>
<td>0.058</td>
<td>0.104</td>
<td>0.232</td>
</tr>
</tbody>
</table>

Table 6: Dijkstra with images from EdgeTracking

<table>
<thead>
<tr>
<th>Image Size</th>
<th>256x256</th>
<th>512x512</th>
<th>948x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration (s)</td>
<td>6.422</td>
<td>61.134</td>
<td>176.922</td>
</tr>
<tr>
<td>Num. of Vertices</td>
<td>606</td>
<td>1169</td>
<td>1370</td>
</tr>
</tbody>
</table>

Table 7: Simplification of vertices from Dijkstra

<table>
<thead>
<tr>
<th>Image Size</th>
<th>256x256</th>
<th>512x512</th>
<th>948x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration (s)</td>
<td>0.014</td>
<td>0.037</td>
<td>0.046</td>
</tr>
<tr>
<td>Num. of Vertices</td>
<td>69</td>
<td>136</td>
<td>169</td>
</tr>
</tbody>
</table>

Table 8: Vectorization rate

<table>
<thead>
<tr>
<th>Image Size</th>
<th>256x256</th>
<th>512x512</th>
<th>948x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration (s)</td>
<td>7.078</td>
<td>63.934</td>
<td>192.514</td>
</tr>
<tr>
<td>Num. of Vertices</td>
<td>69</td>
<td>136</td>
<td>169</td>
</tr>
<tr>
<td>Rate (vertices/sec)</td>
<td>9.748</td>
<td>2.127</td>
<td>0.878</td>
</tr>
</tbody>
</table>
7. Roadmap & Perspectives

We have prepared a roadmap for identifying future system enhancements to improve the vectorization, user interaction (ergonomics) and speed of processing.

A continuous vectorization

The goal here is to vectorize in a continuous way each time we click on the blue outline.

The principle is as follows:

- At the beginning of the vectorization:
  - We place a starting point and ending point
  - A first vectorization is done between these two points

- Then:
  - We designate a new end point by just clicking on the blue curve.
  - And the ending point of the previous step becomes the starting point.
This improvement will allow us:

- To gain time because the Dijkstra algorithm will run faster because the search field will be reduced.
- Better control of vectorization because the operator can automatically correct by adjusting the vertices.
- Better ergonomics and use by a deletion of button clicks to add points replaced by mouse clicks and keyboard shortcuts.

**Ergonomics**

A snapping setting up will allow:

- Connection to the nearest pixel of the contour by a simple flight over mouse.
- Highlighting the contour of which the pixel belongs
- Contour drawing with vertices by just moving the mouse over the contour.

The benefits will be a reactive and dynamic system, more intuitive for the user and better interaction.

**Contour interpolation**

It often happens a single contour is cut into pieces due to: occlusions, obstacles on the contour (trees which cut a road), radiometric characteristics (contours with a low gradient)

To do this we can consider a simple interpolation by a straight line between the extremities start and end edges. And finally, we can improve the vector data edition by adding features such as add and remove vertices.
8. Conclusion

The production of vector data from satellite images is a tedious process that requires the involvement and attention of a photo-interpreter responsible for giving a meaning to the images. Nowadays existing tools are passive and improve the productivity of the photo-interpreter by just providing an ergonomic environment to capture vector data.

Starting from scratch, I have put in place through the state-of-the-art a bibliography addressing the extraction of road networks, hydrographic areas and shoreline on satellites and aerial images. Tools for implementation of algorithms for image processing; GUI modeling and design have been put in place to solve our problem. These powerful tools allowed us to realize a semi-automatic features extraction system.

This project allowed materializing an idea long matured in Magellium. Thus, at the end of the internship, all objectives were achieved and new perspectives are to consider as future software for assisted vectorization developed by Magellium and integrated into a GIS.

On a personal level, this internship allowed me to carry on a R&D project from beginning to end. It was an opportunity for me to apply my theoretical knowledge acquired during my training including image processing and object-oriented programming in C++. It also allowed me to gain new knowledge such as the establishment of system and software architecture. Finally, the project was performed in a friendly environment encouraging personal development.
9. References


[17] Aijun Chen, «River Extraction Based on Knowledge and Fuzzy Classification», 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery


43
10. Appendix

Contour Extraction

1. Choose the image: File ➔ Open or Ctrl+O

2. Choose the ROI (rectangle in blue) in the input image area and see it show in the output view area (here I zoomed).
3. Set the parameters of the Preprocessing module and click on button Run.

4. The results are superimposed on the ROI image. Now, we can start the vectorization process by clicking on the Vectorization tab.

5. We zoom to place a seed point (in green) in the red contour and then click on button EdgeTracking to extract the contour.
6. In blue, we see the extract contour on which we can perform vectorization between a **start point** and **an end point** on this curve. We place these points by **clicking on buttons** VectStart then VectStop and run the algorithm by clicking on **button Vectorization** after setting the **tolerance value**.

7. The algorithms runs and the extracted vertices are superimposed to the ROI image to be able to judge about the quality of the vectorization and readjust vertices.
8. Zoom and adjust vertices if necessary.

9. If satisfied, click on **button Accept**, the processed contour is show in green both in the input and output images

10. Repeat the process from the step 2 to vectorize the whole image.