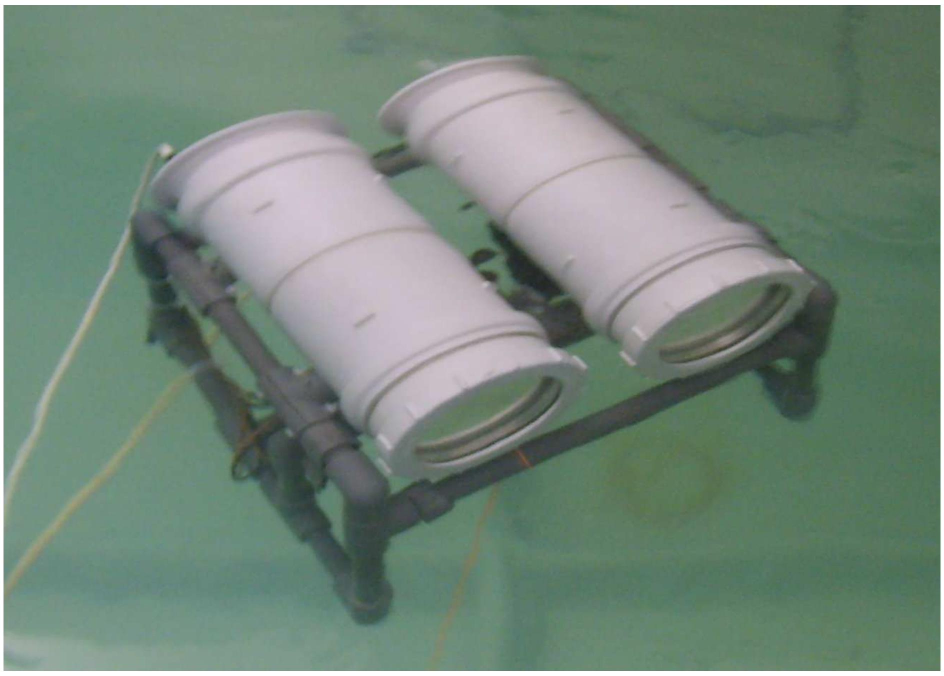
Institut Universitaire de Technologie
de Clermont-Ferrand
Département informatique
Option SI



Placement report

2006-2007

# Development of a control system for an Autonomous Underwater Vehicle: *Image capture, processing and analysis*

*Rémy Mellet*

Supervisors :
Chis Aust & Graeme Dunbar

## Acknowledgments

First of all, I would like to thank Dr. Chris Aust my supervisor for his helping, his sympathy and his advice brought during this placement.

I would like to thank Mr. Graeme Dumbar for his advice too.

My thanks are also for the three others French students: Guillaume Bresson, Mathieu Bertrand and Vincent Bonnefoy with whom I had a lot of good moments. Markedly, I thank Guillaume who worked with me during these eleven weeks.

Lastly, I would like to thank my school senior lecture at the IUT Mr François Delobel and Mrs Carol Fynn who helped me to spend this placement abroad.

# Table of contents

*Introduction* ........ - 3 -

*1 Context presentation* ........ - 4 -

**1.1 Aberdeen** ........ - 4 -

**1.2 RGU** ........ - 5 -

**1.3 School of Engineering** ........ - 6 -

*2 Subject study* ........ - 7 -

**2.1 SAUC-E competition** ........ - 7 -

**2.2 Work allocation** ........ - 8 -

**2.3 Equipment used** ........ - 9 -

*3 AUV composition* ........ - 10 -

**3.1 K8055 board** ........ - 12 -

**3.2 The webcam** ........ - 16 -

3.2.1 Video for Linux ........ - 16 -

3.2.2 Dark image ........ - 16 -

3.2.3 Field of view ........ - 17 -

*4 Image processing* ........ - 19 -

**4.1 Buoy detection** ........ - 20 -

4.1.1 HSV colour space ........ - 22 -

4.1.2 Algorithm ........ - 23 -

**4.2 Target detection** ........ - 25 -

4.2.1 LED detection ........ - 26 -

4.2.2 Circle detection ........ - 27 -

4.2.3 Comparing between circle position and LED position ........ - 30 -

**4.3 Surfacing zone detection** ........ - 31 -

4.3.1 Approach ........ - 31 -

4.3.2 Explication of the Hough technique ........ - 32 -

4.3.3 Algorithm ........ - 34 -

4.3.4 Optimization ........ - 36 -

*Technical balance sheet* ........ - 37 -

*Conclusion* ........ - 38 -

*Reference* ........ - 39 -

*Appendix* ........ *see French report*

# Introduction

For my second year of my DUT in computing[1] at the IUT of Clermont-Ferrand, I had made an eleven weeks placement at the Robert Gordon University located in Aberdeen in Scotland. During this placement, I was supervised by Dr. Chis Aust and Mr Graeme Dunbar and worked with Guillaume Bresson, another French student.

The aim of this placement was to develop an application which controls an underwater vehicle. This vehicle must be autonomous and participated at the SAUC-E competition. This competition is divided by many task and for each task, we must to detect an object and interact with it.

Once the subject studied, we concluded that this project can be divided in two parts: the GUI[2] development and the development of algorithms which permit to detect objects. I worked on the second part. Like we just use a webcam, I spent mostly time to develop image processing algorithm. The competition is divided in many tasks, so I searched for each task the algorithm matches to this task.

[1] Equivalent to second year of BSc in computing

[2] Graphical User Interface

# 1 Context presentation

### *1.1 Aberdeen*

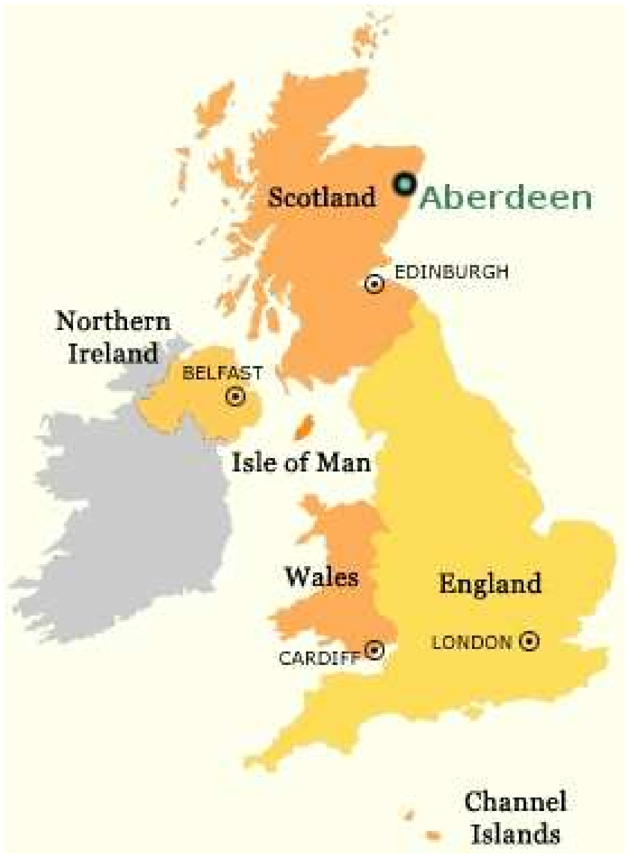


***Figure 1 : carte du Royaume Uni***

During my work placement I decided to take the opportunity that the IUT gave us and thus to carry out my training course abroad and more precisely in Aberdeen in Scotland.

With its 250 000 inhabitants, Aberdeen is after Edinburgh and Glasgow the third town of Scotland. Located in the North-East of the United Kingdom it is called “the granite city” because between the 18th and the 20th century the buildings were built with the granite stone.

Since the 1970s, the date where oil was discovered in the North Sea, it is mainly recognized to be the European capital of oil.

Aberdeen has two universities: “University of Aberdeen” and “The Robert Gordon University”. It is in this second university that I worked.

## *1.2 RGU*

The “Robert Gordon University” has the name of its founder Robert Gordon (1668-1731). Born in Aberdeen, he made his studies in Marischal College and became rich when he went to Danzig.on the Baltic.

The origins of the university go back to the years 1750 when the “Robert Gordon Hospital” was built with the aim of offering lessons for children of the middle-class of the city. This school evolved to become in 1881 the “Robert Gordon College”. It then gave lessons going from primary education to secondary as well as lessons in mechanics.

***Figure 2 : RGU’s logo***

In 1884, the Institute of Mechanics of Aberdeen was attached to the Robert Gordon College. At the end of the century, with an aim of providing more means to the pupils, “Gray's School of Science and Art” merged with the Robert Gordon College to become a few years later the Robert Gordon Technical College then in 1965 the Robert Gordon Institute of Technology. Lastly, it obtained the title of University in 1992 to become such as well-known today as the “Robert Gordon University”.

This university has a great reputation. This year RGU has no less than 13.000 students coming from 113 different countries.

## 1.3 School of Engineering

***Figure 3 : logo of the School of Engineering***

The School of Engineering is part of RGU. This department has many technical courses linked of the town’s economic context:

- Oil and Gas Engineering.
- Electronic and Electrical Engineering.
- Computer Network Management and Design.
- Mechanical Engineering.
- Engineering Design.
- Engineering for Sustainability and the environment.
- Electronic and Communication Engineering.
- Artificial Intelligence and Robotics.
- Electronic and Computer Engineering.
- Information Systems.

It’s in this department that I worked for 11 weeks.

# 2 Subject study

For the fist time, RGU decided to participate in a competition where different autonomous underwater vehicles (AUV) have to perform some tasks autonomously. Therefore, my team-mate Guillaume and I had to make a robot which would be able to perform in some tasks.

## *2.1 SAUC-E competition*

SAUC-E is the acronym for Student Autonomous Underwater Challenge – Europe. This competition started two years ago. This year eight teams from two differents countries (UK and France) took part in it.

The goal of this competition is to advance the state-of-the-art of Autonomous Underwater Vehicles by challenging many teams of students and engineers to perform autonomous missions in the underwater environment. Furthermore, it allows the fostering of ties between young engineers and the organisations involved in AUV technologies.

The AUV must perform many tasks autonomously

- ♦ Pass through a validation gate without contacting it.
- ♦ Drop a marker on a circular target situated on the bottom of the tank. The target has a light in its centre.
- ♦ Locate a mid-water target and contact it
- ♦ Surfacing within a designated surfacing zone

New tasks appear with the new edition of SAUC-E 2007:

- ♦ Produce a map of the objects which are in the tank.
- ♦ Detect a tyre and a cone located at the bottom of the tank.

These different tasks have to be performed in a maximum of 40 minutes, the AUV must not contact the edge of the tank and must not surface until the end of the competition.

Each team in this competition consists of about fifteen students, engineering or doctorate, studying electronics, mechanical engineering and computing.

## *2.2 Work allocation*

In the first week, we searched for information about AUVs, ROVs and the SAUCE competition. At the end of this week, we had identified two parts of the project:

- ♦ the GUI development
- ♦ the development of algorithms which permit the detection of objects

I worked on the second part while Guillaume worked on the first part. But before the beginning of this work, we found it very useful for each of us to understand how to retrieve an image from the webcam.

More precisely, the part I worked on consisted of:

- ♦ retrieve an image
- ♦ process it
- ♦ analyze it
- ♦ control the motors

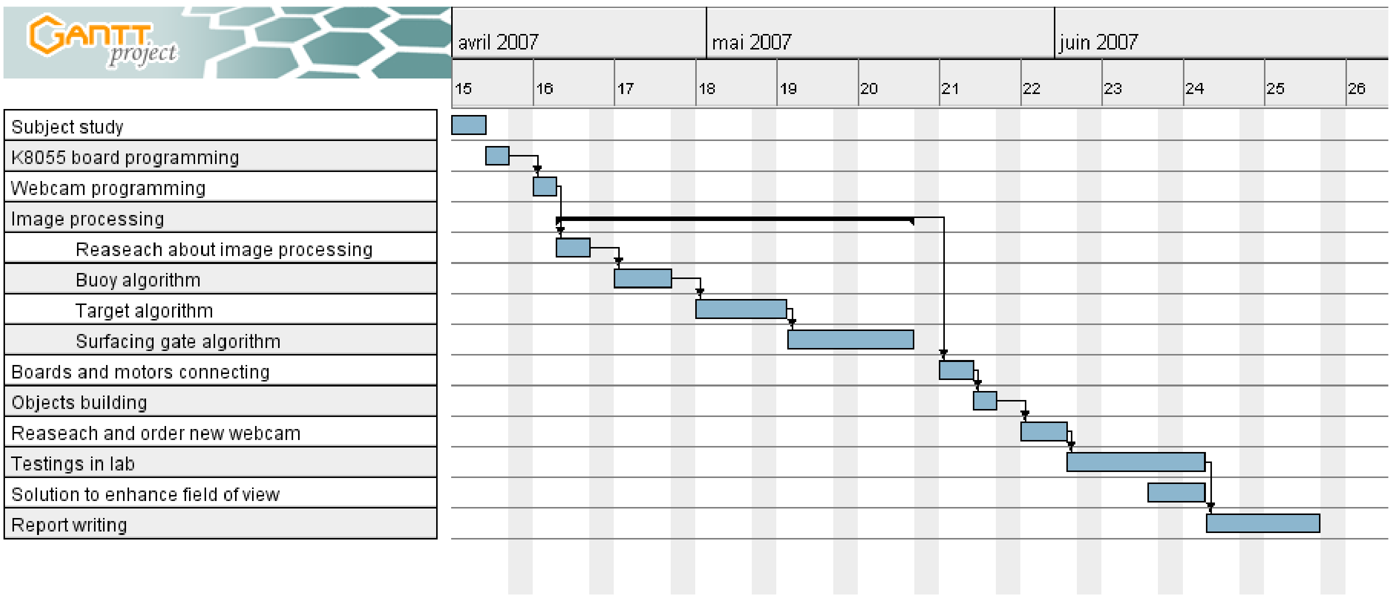


***Figure 4 : work make during these 11 weeks***

## *2.3 Equipment used*

Even if we started this project, we did not begin with nothing, in fact we didn’t have to build a robot because we used an existing ROV.

The robot is controlled by a PC which runs on Linux. To control motors of the vehicle an experimentation board is used. To explore the environment of the AUV, a webcam is used.

Regarding the computing tools, I had the choice between two languages: C and C++. To enhance readability, re-use, and to be compatible with Guillaume's programme and because I prefer use object oriented programming, I developed my application in C++ and used the G++ compiler.

# 3 AUV composition

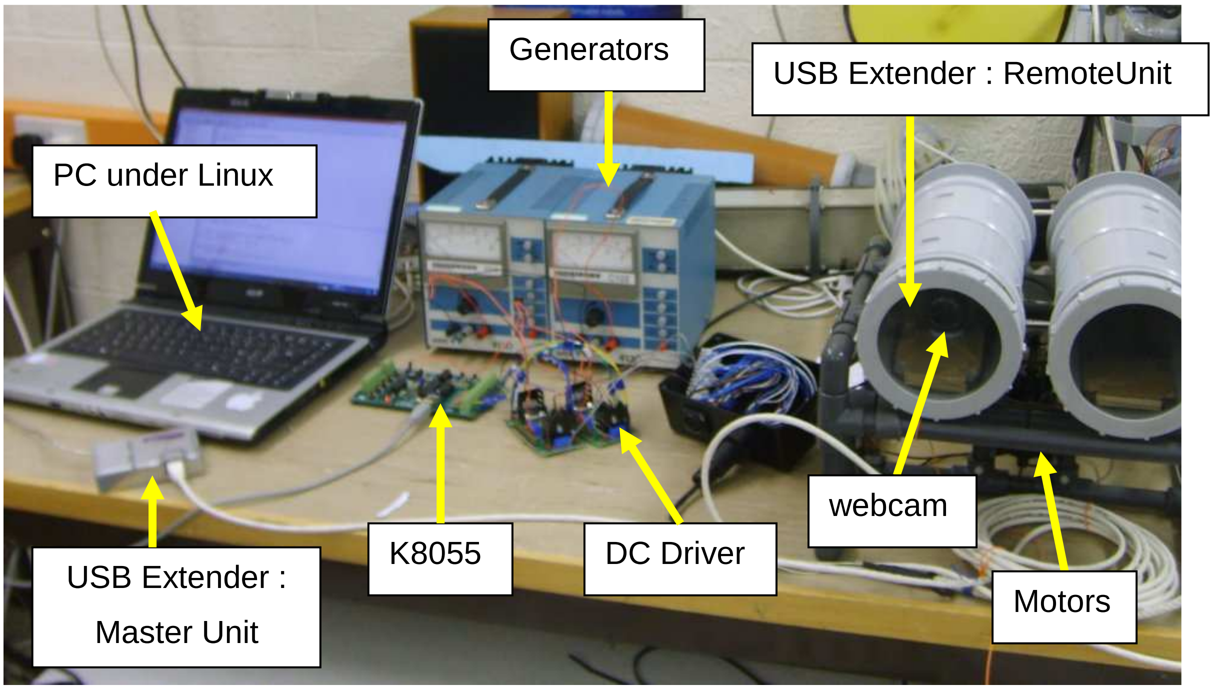


***Figure 5 : AUV's elements***

To retrieve and process data, we used a laptop which runs on Linux (Debian). The PC is connected to a webcam through the USB Extender. This USB Extender is an extender which uses a Cat-5 cable. In the first part of the placement we used a Sweex webcam but it provides a poor quality image, so we changed it for a Logitech Communicate Technology STX webcam.

To move the vehicle and also to emit signals, we used an experimental board. This card is built by Velleman and is called K8055. This card is fed through USB but this type of connection does not provide enough current to feed a motor. So we used two current generators which are connected to a DC Driver, itself connected to the motors.

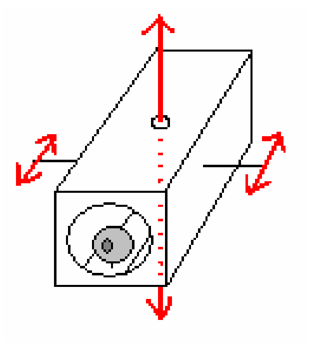


***Figure 6 : position of the motors***

Three motors permit the control of the vehicle. These motors are integrated in the ROV which was built by another group of students. The position of the motors let the AUV go forward, backward, up and down, and to turn it to the left and the right. In our case, there is just one difference between an ROV and an AUV: the ROV is controlled by remote control while an AUV is controlled by a computer because it is autonomous.

We have also taken off the remote control of the ROV to connect the ROV to the computer.

In the continuance of this project, all elements named above will have to be integrated in the vehicle. So, the PC will probably be replaced by a PC-104[3] and generators by a battery.

[3] PC-104 is a standard for embedded computer.

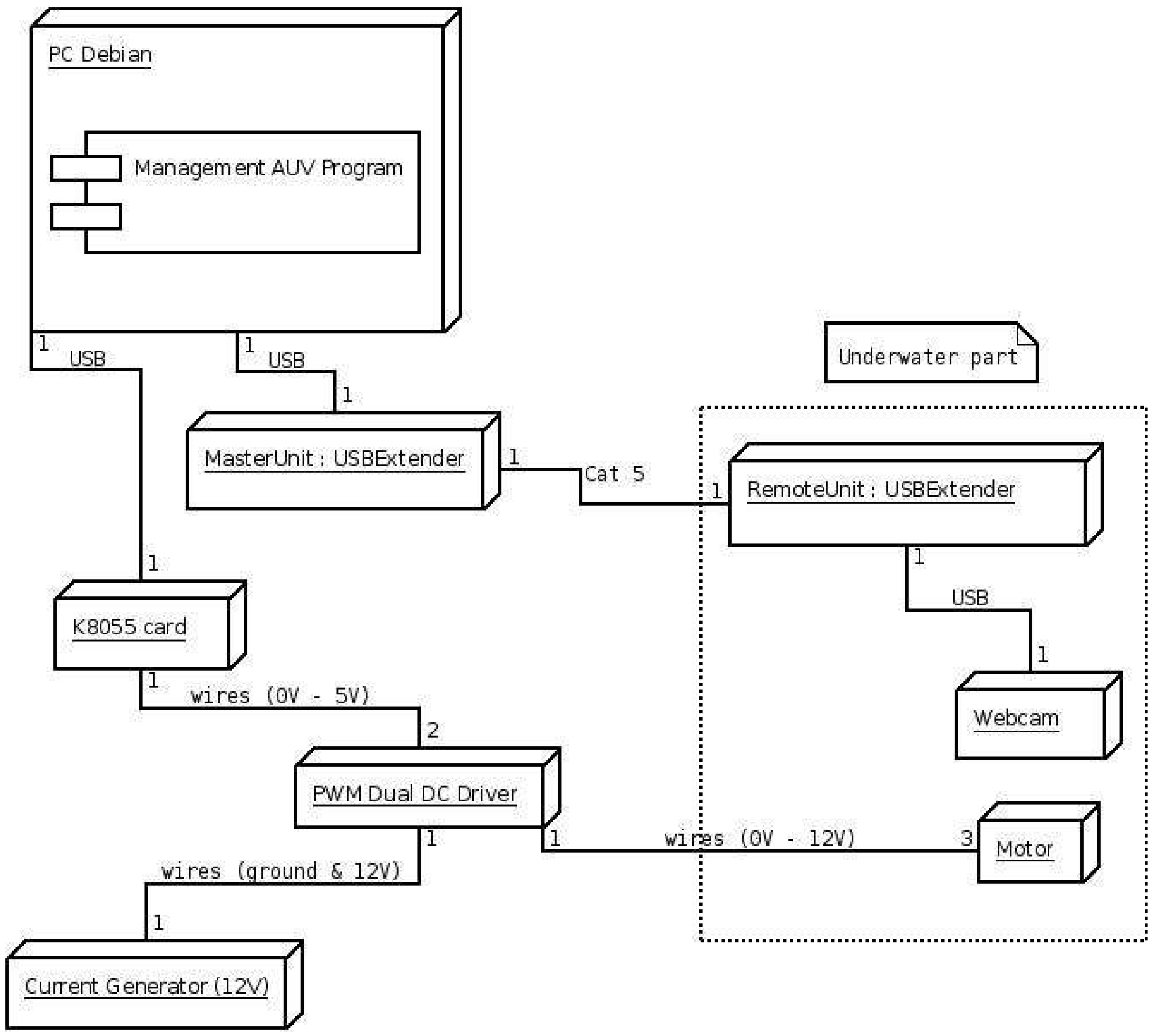


***Figure 7 : deployment diagram of the AUV***

## *3.1 K8055 board*

This board has 5 digital inputs, 8 digital outputs, 2 analogue inputs and 2 analogue outputs which can emit PWM[4] signals.

[4] Pulse Width Modulation: in our case, this signal allows to set motors speed.

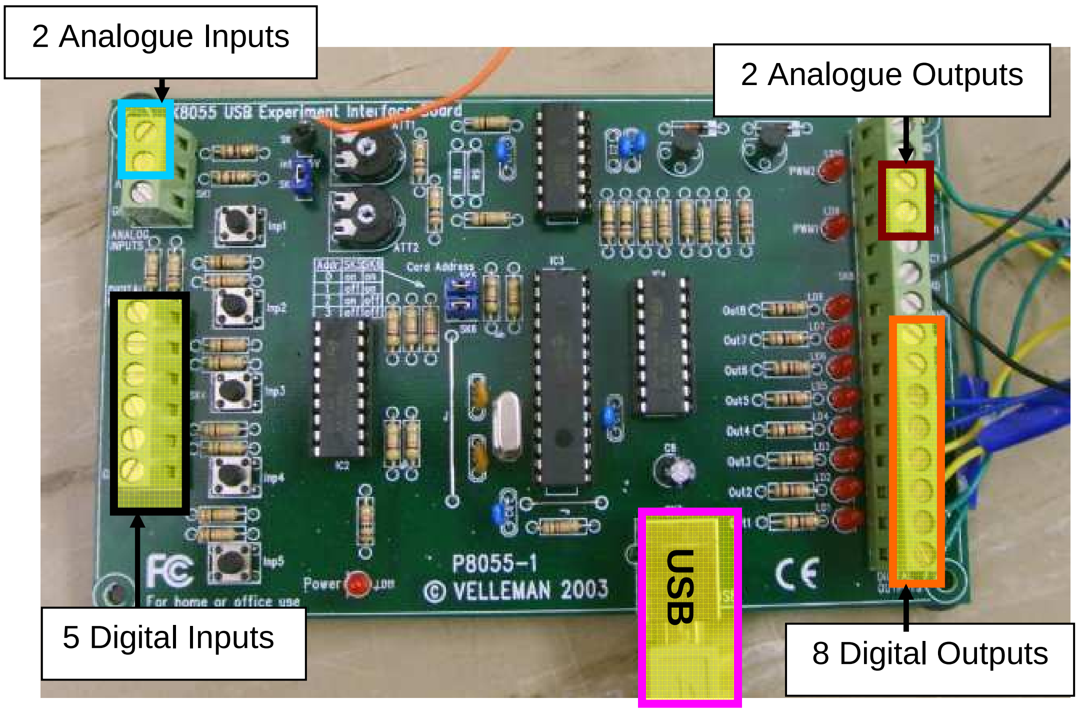
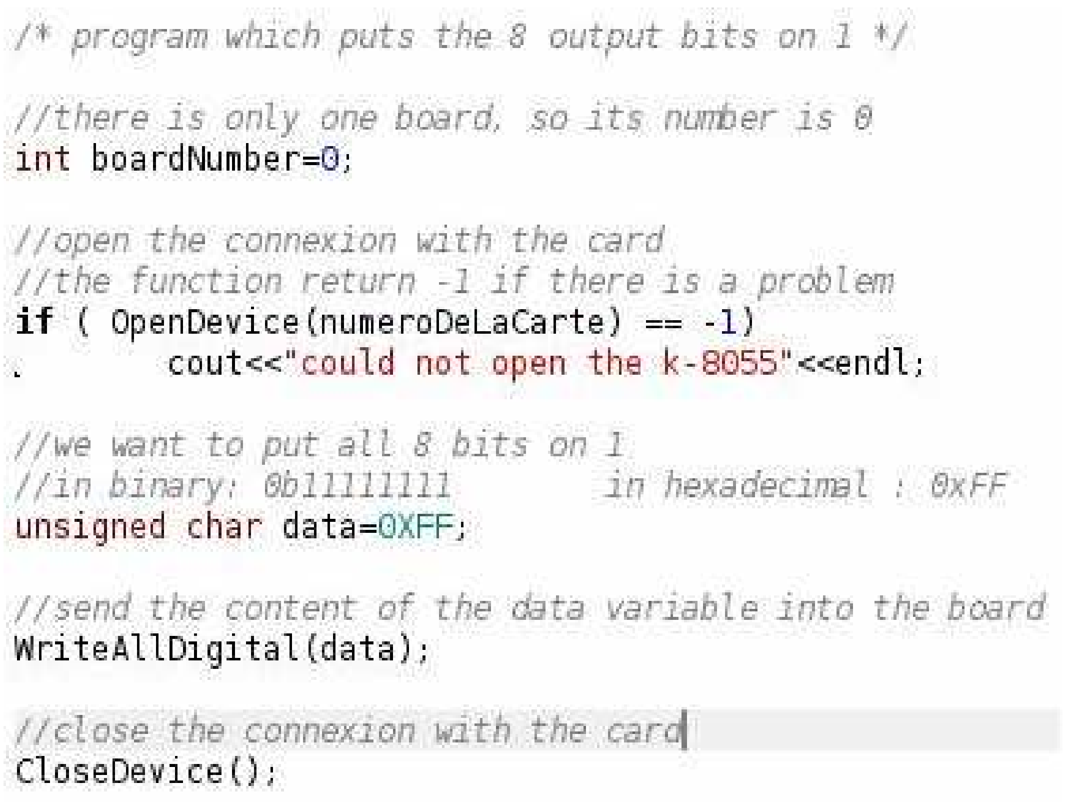


***Figure 8 : K8055 board***

With this card there are many libraries which permit the card te be used by many Windows' applications development tool like Delphi, Visual Studio or C++ Builder. But even if nothing is provided for using this board under Linux, I found on the internet a library developed by amateurs. This library permits the use of the card like the windows library, very easily.

The Figure 9 shows the facility to control the board.

Before using this library, I had to install it. I just had to download the archive from the author’s website and run the “make” command.



```
/* program which puts the 8 output bits on 1 */

//there is only one board, so its number is 0
int boardNumber=0;

//open the connexion with the card
//the function return -1 if there is a problem
if ( OpenDevice(numeroDeLaCarte) == -1)
        cout<<"could not open the k-8055"<<endl;

//we want to put all 8 bits on 1
//in binary: 0b11111111        in hexadecimal : 0xFF
unsigned char data=0XFF;

//send the content of the data variable into the board
WriteAllDigital(data);

//close the connexion with the card
CloseDevice();
```

***Figure 9 : all output bits are set to 1***

List of the functions used in the application :

| | |
|---|---|
| - OpenDevice | *opens the communication link* |
| - CloseDevice | *closes the link* |

<u>Canaux digitaux</u>

| | |
|---|---|
| - WriteAllDigital | *sets the 8 digital outputs according to the data* |
| - ReadAllDigital | *reads the status of the 8 input channels* |

<u>Canaux analogiques (PWM)</u>

| | |
|---|---|
| - OutputAllAnalog | *sets both analogue output channels according to the data* |
| - ReadAllAnalog | *reads the status of both analogue input-channels* |

Each motor is connected to two digital channels corresponding to 2 bits. Even if we just have eight of them, it's enough because we use only three motors. The first bit is used to switch the motor on or off while the second bit is used to choose the direction of motor rotation.

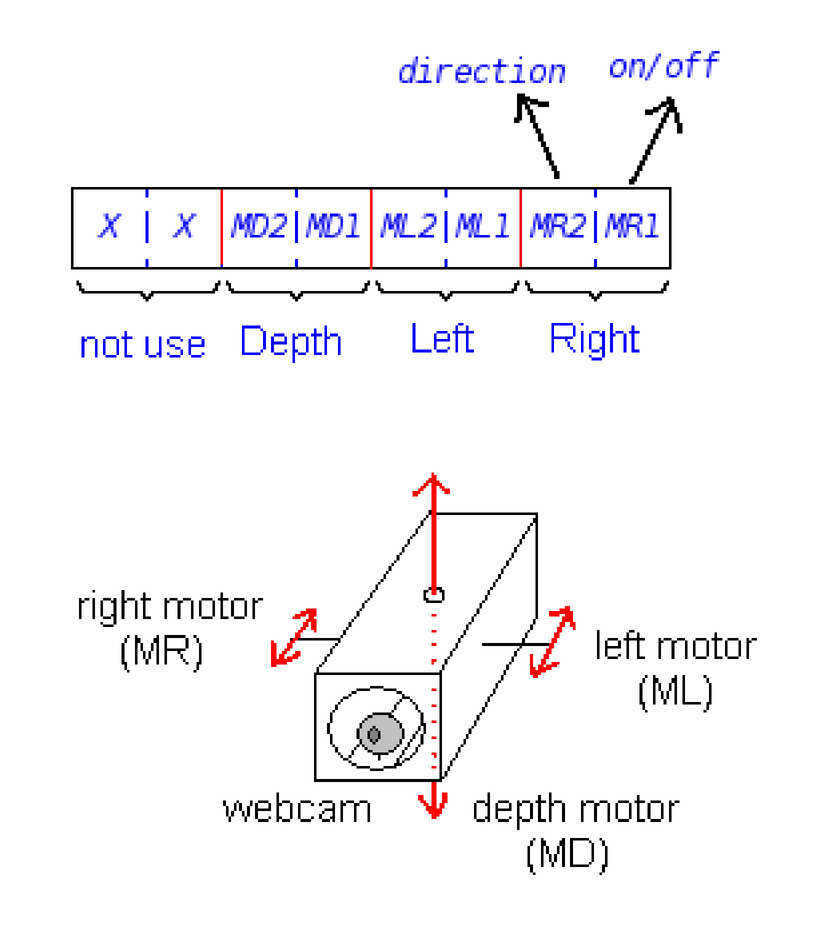


***Figure 10 : matches between bits/digital output and motors***

For example, when the robot must go up and turn to the right, the fellowing 6 bits will be sent to the card: 01 | 01 | 11

In this case, the direction of rotation of the depth and left motor is clockwise while the direction of rotation of the right motor is anticlockwise.

### *3.2 The webcam*

At the beginning, the Sweex webcam was used to provide images to the computer. This webcam was compatible with Linux and its driver used the V4L[5] API[6]

The name of this webcam's driver is « spca/gspca », it is developed by amateurs and manages at least 270 different types of webcam. So we studied this API to know how to capture an image from the webcam.

#### 3.2.1 Video for Linux

The problem of driving video peripherals is rather complex because there is a multitude of possibilities, in fact there are many different types of peripherals (camcorder, webcam, TV card) with many features (image colour, greyscale, black and white, image size). Thus to unify that, Linux developers have defined an API named Video For Linux or V4L which allows us to drive all types of video peripheral easily and in the same way.

In the appendix a program is available showing how to capture an image with V4L.

#### 3.2.2 Dark image

When we wanted to put the robot in the water for the first time the image captured by the webcam was totally dark. Indeed as the water is not as transparent as air, the image is darker in the swimming pool than outside.

To enhance the quality of the picture, we hung a torch on the AUV but nothing changed. Another person working in the same room as us owned a webcam. I borrowed it and attached it to the vehicle. From a luminosity point of view, the webcam provides a good image but the problem was that the image was rather fuzzy.

[5] Video For Linux

[6] Application Programming Interface : source code interface that a computer system or program library provides

After some research, I realized that two types of webcam matched my requirements (to have a correct image when the luminosity is poor). One type uses a CCD[7] sensor and the other type use the “RightLight” technology developed by Logitech.

The sensor is an electronic component used to convert the light received into electrical signals. It’s the basic element of digital cameras. There are two families of these sensors: CCD sensors and CMOS sensors[8]. Most webcams use a CCD sensor while digital cameras use a CMOS sensor. In general, the quality of a CCD sensor is higher especially when the image is dark. But on the other hand, this type of sensor is more expensive.

According to the Logitech website, the technology named « RightLight » uses new powerful CMOS sensors, but personally I think there is in fact an image processing built-into the camera which enhances the quality of the picture.

Finally the Logitech Communicate STX webcam was ordered. Although this uses a CMOS sensor, thanks to the “RightLight” technology, this webcam provides a very good image even in an underwater environment.

### 3.2.3 Field of view

Thereafter, a second problem appeared: the field of view of the webcam is too small (approximately 30°) but in order to detect th e surfacing zone, the algorithm need to see the whole of this object.

We tried to improve this field of view by adding concave lenses[9] which were available in RGU. But these lenses did not let us improve considerably the field of view.

[7] CCD : Charge-Coupled Device

[8] CMOS : Complementary Metal Oxide Semi-conductor

[9] convex lens : kind of lens which permit to widen an image.

Then for curiosity, I tried to use a spyhole, the field of view was multiplied by 7 but the image became too fuzzy and distorted.

As little of time remained and optics is not in my knowledge, I preferred to research a software solution to solve this problem.

# 4 Image processing

To perform the competition, the robot must detect many objects having defined features. To detect an object, there are two types of algorithm: those using colour detection and those using edge detection. The algorithm type choice depends on the object features. We prefer to work with a colour detection algorithm because it is easier to understand and code but the most of the time a colour detection algorithm is not efficient enough.

In the following pages, I will explain step by step my algorithm to detect each of the first three objects namely the buoy detection, the target detection and the surfacing zone detection.

To simplify the robot movement, all the objects are located horizontally. Indeed, the arrangement of the motors doesn't facilitate the moving of the robot on its side.

The global algorithm of the following program is:

While the AUV doesn't detect the object, it turns on its axis and when the object is detected, the AUV moves towards.

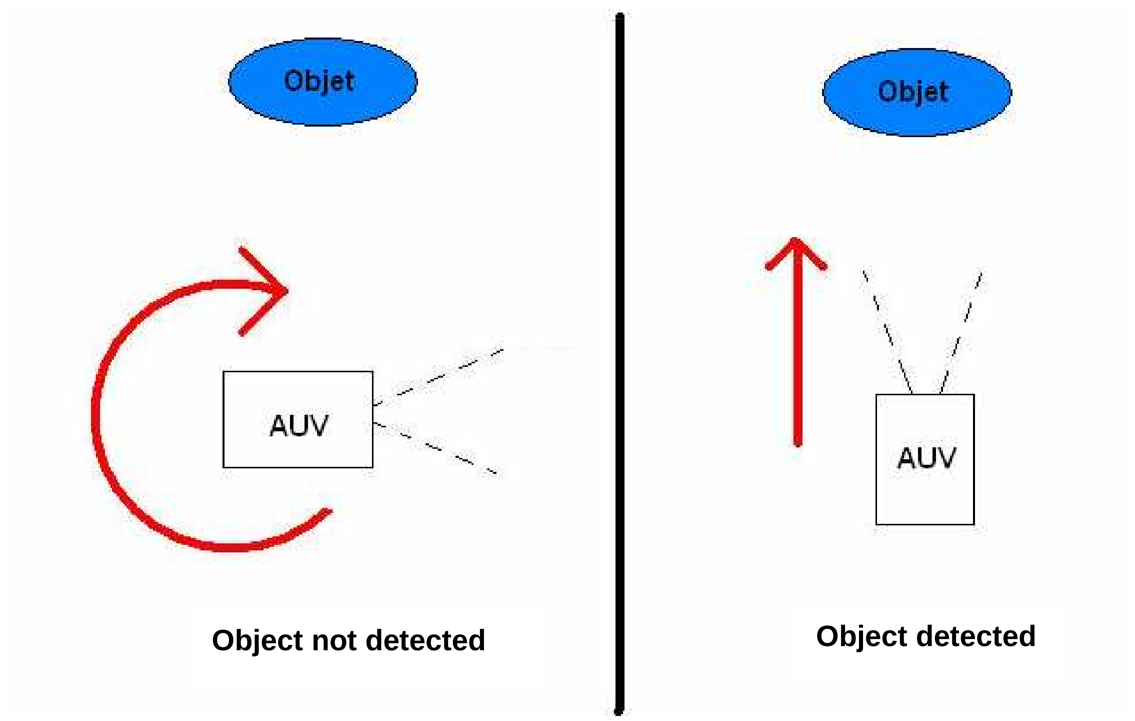


***Figure 11 : Global  algorithm***

I made my algorithms as generic as possible with the aim to reuse them in other projects and facilitate the addition of new object detection with different features without modifying the program.

## *4.1 Buoy detection*

I started with the simplest algorithm: the algorithm which detects the orange buoy. No object with the same colour is in the tank, so I can use colour detection.

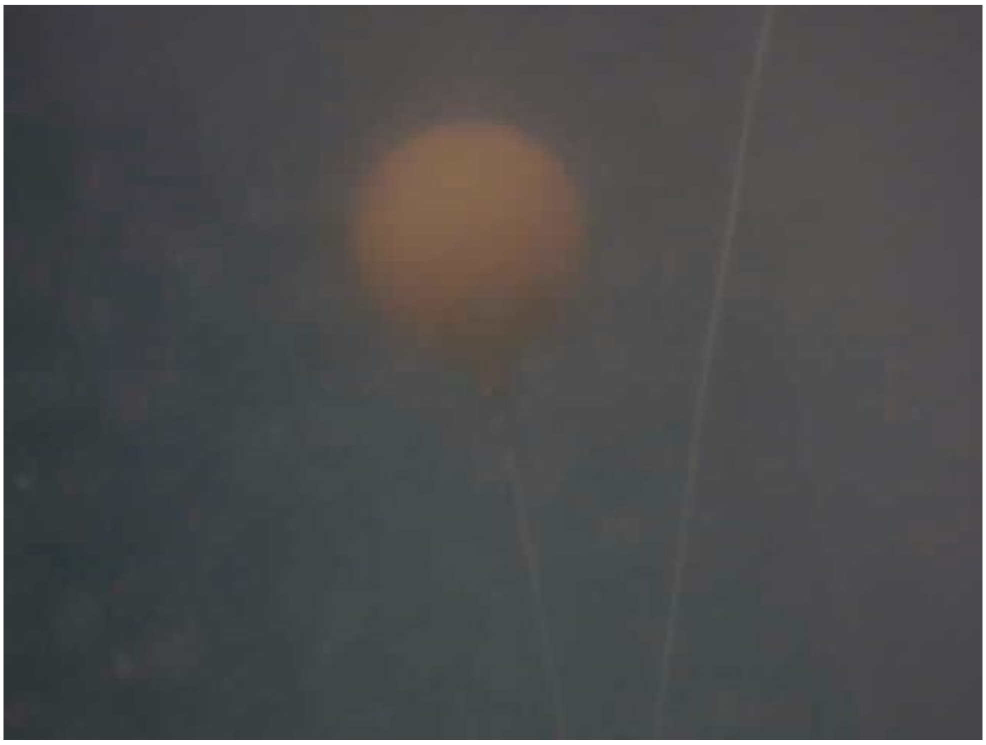


***Figure 12 : Object to detect***

Many ways exist to represent a colour. Often the RGB colour model is used. This model of colour by uses three primary colours Red, Green and Blue combined to reproduce other colours. For instance, this representation is used to print a document, to display an image on a screen or to store an image in informatics. The image captured by the webcam uses the RGB model.

To detect an object with colour detection we can think that we just have to compare the three channels of the picture (Red-Green-Blue) to know if it matches to the orange of the buoy. The problem is than we cannot indicate precisely the buoy’s colour with one value for each channel because even if the colour of the buoy is uniform, the buoy’s colour we receive through the webcam or our eyes is not uniform. This colour depends on the luminosity and others factors. Furthermore, we cannot indicate the buoy’s colour with a set value because it’s not precise enough.

So we need to use another colour space unconcerned by the luminosity variation to represent the image.

.

### 4.1.1 HSV colour space

In everyday life, to express a colour, we don't say "this colour is composed of 30% red, 15% green and 80% blue" but we say "it's light blue". The new colour space I used describes the colour exactly like that. This colour space is named HSV standing for Hue Saturation Value. This way to represent colour is more natural, more logical.

The **hue** describes the shade of colour. Hue is expressed by an angle (between 0° and 360°).

The **saturation** describes how pure the hue is with respect to a white reference. For example, a colour that is all red and no white is fully saturated. If we add some white to the red, the result becomes more pastel, and the colour shifts from red to pink. Saturation is a percentage that ranges from 0 to 100.

The **value** describes the brightness of the colour. If the colour reflects a lot of light, we would say that it is bright. Value is a percentage that ranges from 0 to 100.

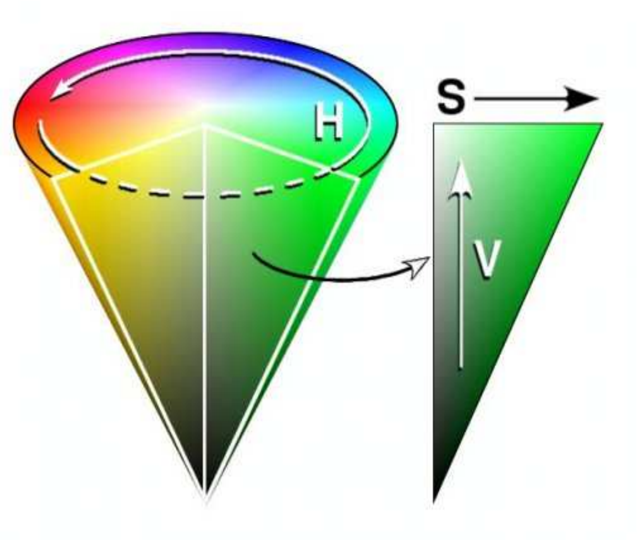


***Figure 13 : HSV colour space***

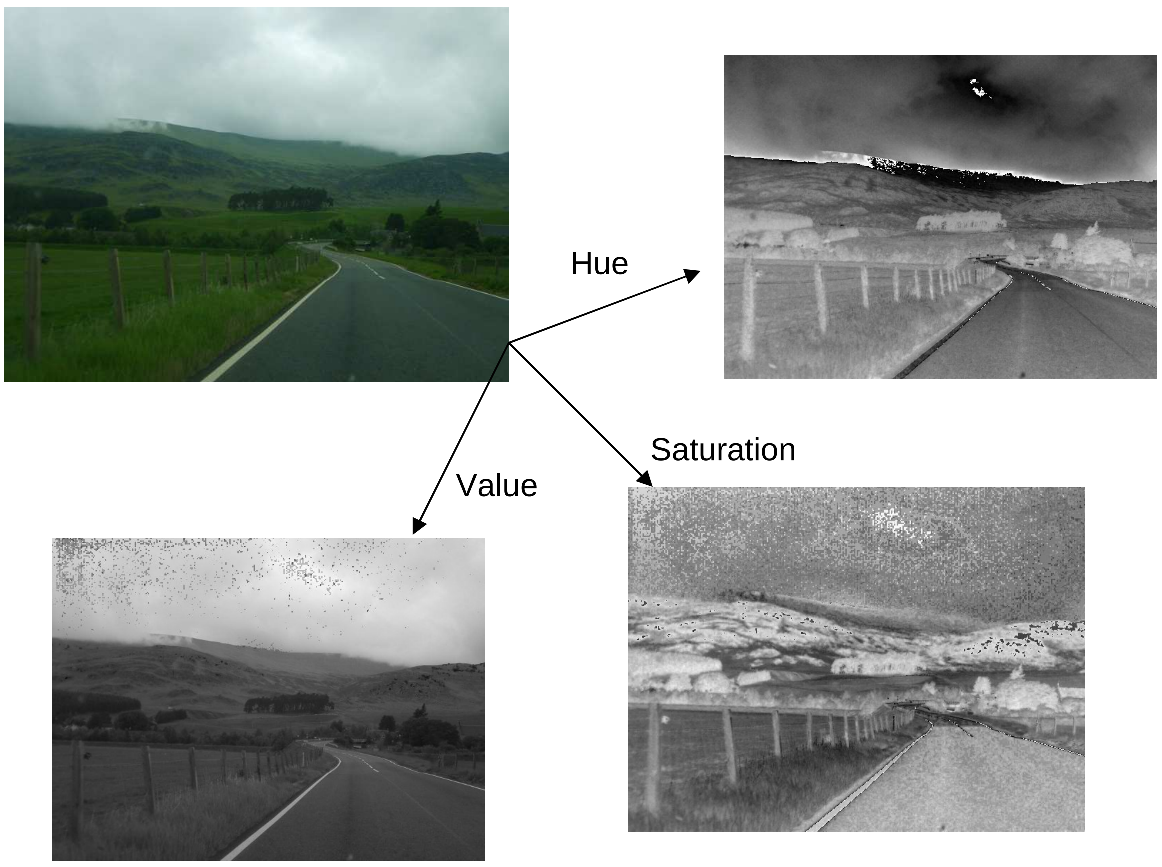


***Figure 14 : channels of an image in HSV colour space***

### 4.1.2 Algorithm

Firstly, to detect the object, I convert the RGB image to an HSV image. I “cut” my image twice into two parts (see below) and count for each part how many pixels with the required hue are there. To have an approximation of the buoy’s diameter - with the aim of knowing if the AUV has reached the buoy – I get back the line of the image which contains the largest number of pixels having the hue being sought.

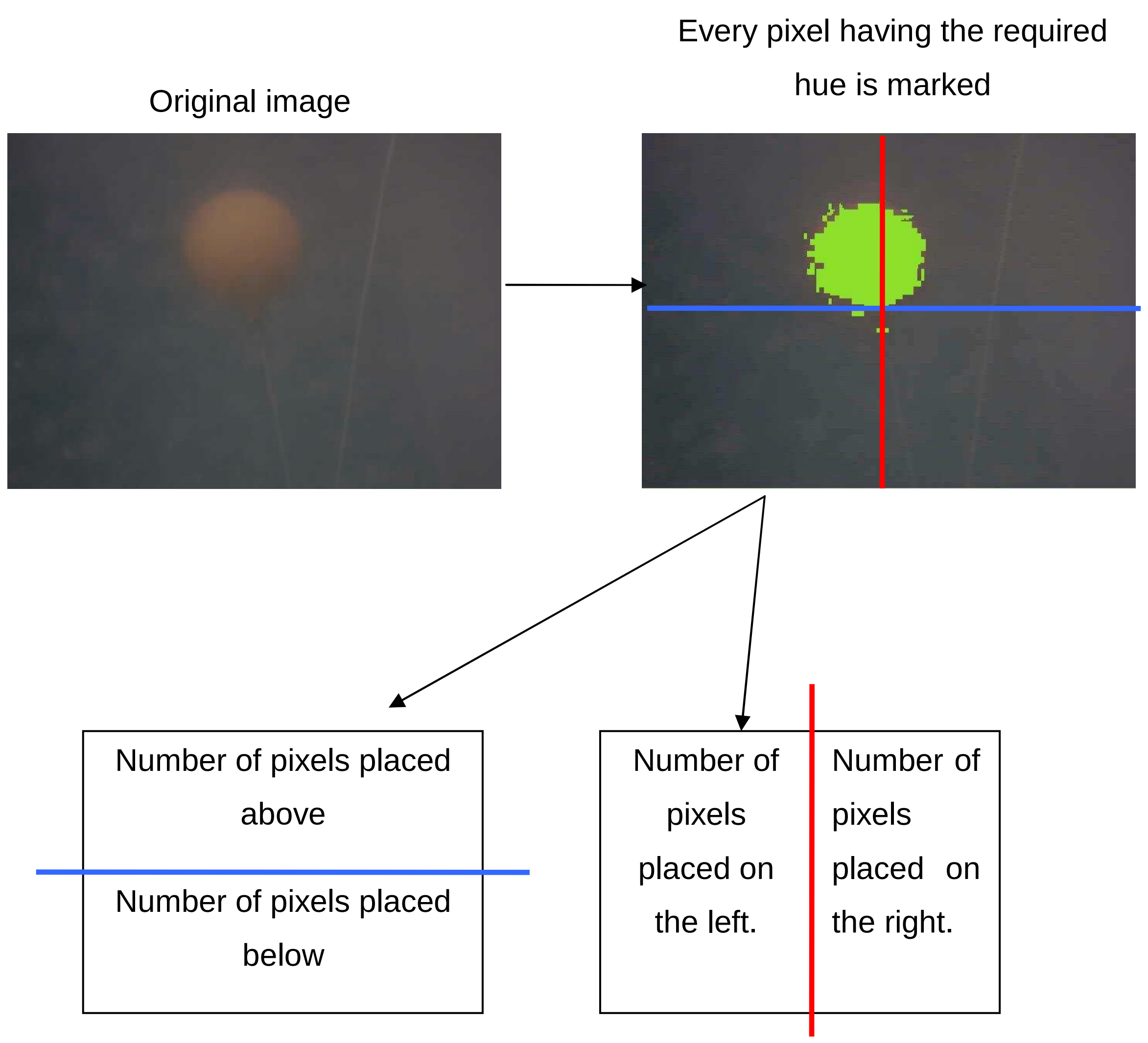

Original image
Every pixel having the required hue is marked
Number of pixels placed above
Number of pixels placed below
Number of pixels placed on the left.
Number of pixels placed on the right.

***Figure 15 : image cutting***

After several tests, I realized that there were impurities and noises which affect the calculation. To reduce noise, a pixel is considered by the algorithm if it has the required hue and if two of its pixel neighbours have that hue too.

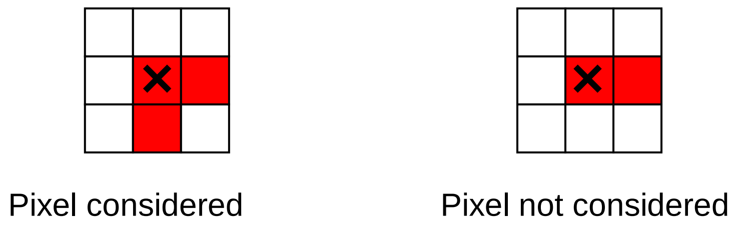


***Figure 16 : pixel considered***

Even if this algorithm is very basic, it works very well because there isn’t another object with the same colour in the tank. In the contrary case we could check that the group of pixels found matches to a circle or the most pixels found are located in a circle.

Once the image is processed, a move order is sent to the vehicle. If the object is not detected, the AUV turn on itself, alternatively if the object is detected different orders are send: If more pixels are detected in the right part than the left part, the AUV turn to the right, and vice versa. If the number of pixels on the right and left parts are approximately the same, the AUV goes forward.
The way to analyse the pixels situated in the lower and upper part is identical, except if the two parts are approximately the same. In this case the robot doesn’t move vertically as it must be at the same depth as the object being sought.

### *4.2 Target detection*

The second object to detect is the circular target. In the competition this object is situated at the bottom of the tank and the vehicle must drop a marker on it.
This object is characterized by a flashing light at its centre.

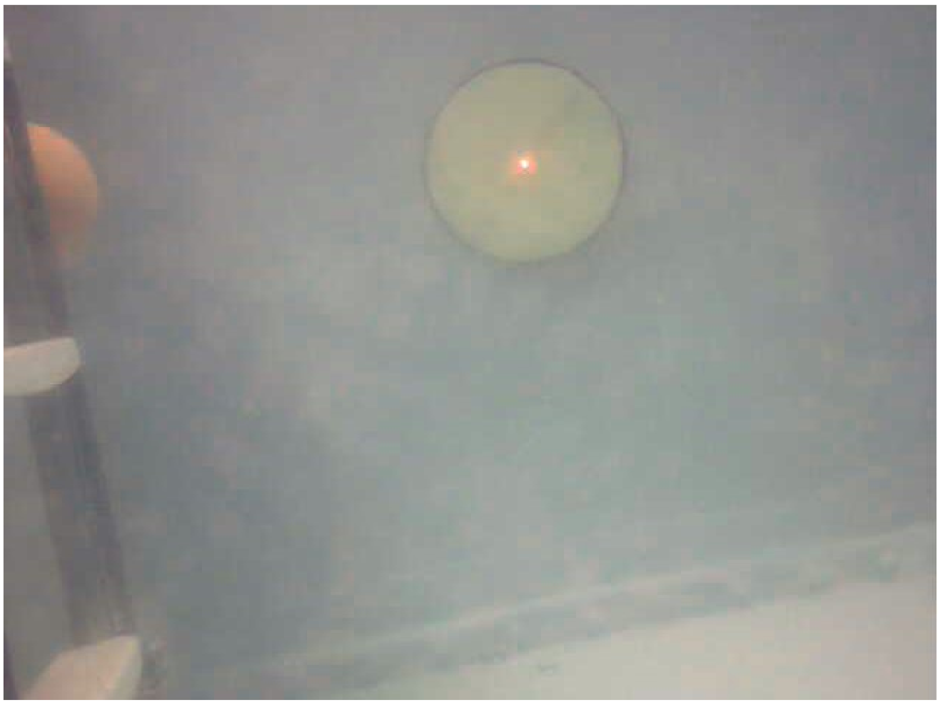


***Figure 17 : the target***

To detect this object properly and efficiently, two algorithms are put together, the first detects the lightest point corresponding to the LED and the second finds the centre of the circle.

### 4.2.1 LED detection

This algorithm was easy to implement because it's sufficient to search for the brightest point of the image. So, we convert the image to grayscale and retrieve the point having the highest value in grayscale. This point is the lightest point of the picture because an image in greyscale matches to the luminosity.
Roughly to find the greyscale of a RGB pixel, we do the average of the three channels (RGB). If we want to be more precise, the following formula can be used (I use it):

**Luminosity** = **Red * 0.299** + **Green * 0.587** + **Blue * 0.114**

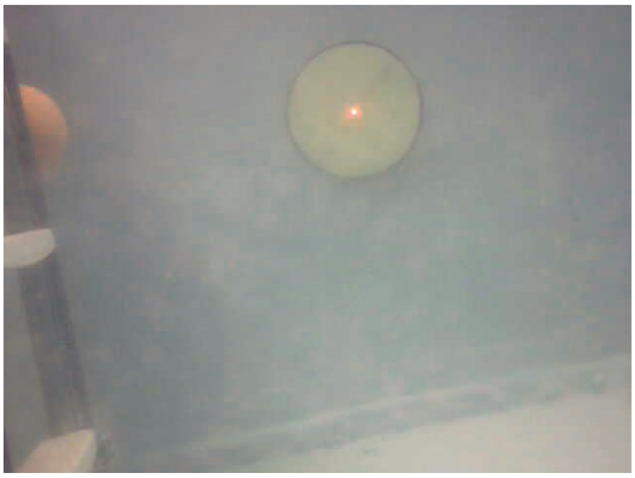


***Figure 18 : color image***

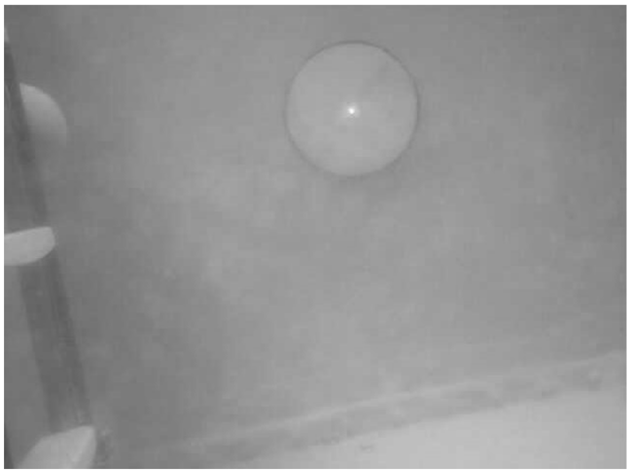


***Figure 19 : grayscale image***

This algorithm is not always sufficient because the most luminous point of the picture is not always the LED, for two reasons: the first didn't concern us but the tank can have other lights and the second reason is that the LED blinks.

### 4.2.2 Circle detection

To enhance the reliability, I joined this algorithm with another which detects the centre of a circle.

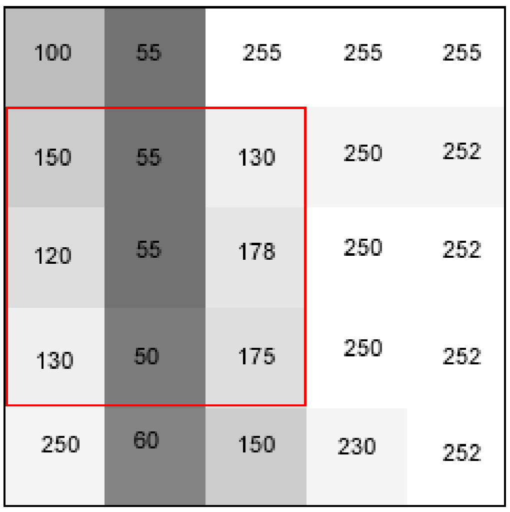
This algorithm contains two steps:

- ♦ The first step is to detect the edges of the image
- ♦ The second, to find the tangent of the edges which match to a circle with the aim of determining its centre.

Many solutions exist for determining the edge of an image; most of them apply a convolution kernel (a filter) to the image. The convolution provides a way of multiplying together two arrays of numbers to produce a third array. This is used in image processing to implement operators whose output pixel values are simple combinations of neighbour pixel values.

The convolution kernel is useful for applying blurry, edge detection, noise reduction etc to an image.

Image in grayscale



| | | | | |
|---|---|---|---|---|
| 100 | 55 | 255 | 255 | 255 |
| 150 | 55 | 130 | 250 | 252 |
| 120 | 55 | 178 | 250 | 252 |
| 130 | 50 | 175 | 250 | 252 |
| 250 | 60 | 150 | 230 | 252 |

Convolution kernel

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | -1 | 0 |

Application of the convolution kernel on the pixel situated at the centre of the red square. Its new value will be:

0*150 + 0*55 + 0*130 + 0*120 + **1*55** + 0*178 + 0*130 + **(-1)*50** + 175 = **5**

***Figure 20 : convolution kernel***

The first edge detection algorithm found was the Laplace operator but it was not efficient enough. Below is its convolution kernel:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

I found another more effective algorithm: the Sobel operator.

The Sobel Edge Detector uses two convolution kernels, one to detect changes in vertical contrast and another to detect horizontal contrast.

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

x

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

y

***Figure 21 : the two Sobel's convolution kernel***

These two convolution kernels allow us to know for each pixel:

- ♦ The contrast variation : if the contrast is larger than the threshold, it's pixel matches to an edge
- ♦ The direction of the contrast: if the pixel is placed on a circle, it allows a tangent of this circle at this point to be had.

Then, to find the circle, for each point being placed on an edge, thanks to the tangent known by the convolution kernels, we draw a line which is perpendicular to this tangent. If the pixel is placed on the circle, this line goes through the centre of the circle.

The next step consists in finding the intersection point of the largest number of lines. This point must be the centre of the circle.

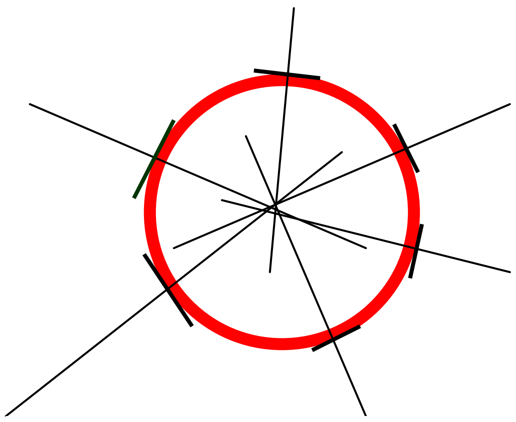


***Figure 22 : method to find the circle's center from tangentes***

### 4.2.3 Comparing between circle position and LED position

If two points match to the circle centre and the LED position are for a few pixels the same point, the circle is detected and the order to reach this object is sent to the vehicle. On the other hand, If the two points didn’t correspond, the AUV turns on its axis.

## 4.3 *Surfacing zone detection*

This detection algorithm of the surfacing zone uses the Hough[10] technique. It was the hardest algorithm to implement because it required knowledge in mathematics I didn't have and much reflection.

I could use a library which implements the Hough technique directly but I thought it important to understand how this technique works to optimize it for our own case. Furthermore, I was very interested to know how this technique works.

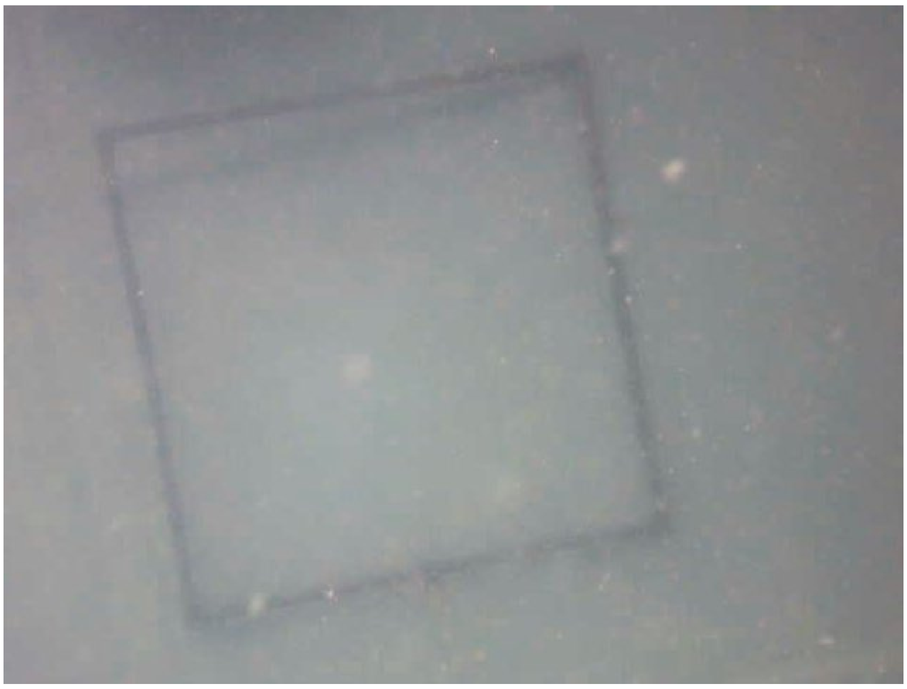


***Figure 23 : object to detect***

### 4.3.1 Approach

The object which must be detected matches to a rectangle when we see it from the front, but the AUV must recognize it even if it is not directly in front. In this case, this object doesn't match to a rectangle but rather to a trapezoid.

So we can't admit that this object consists by four segments with two of them parallel.

[10] Pattern recognition technique invented  by Paul Hough in 1962

My approach was to detect the four lines constituting the trapezoid and find the point of intersection of these lines and finally to find the “centre” of the trapezoid and order the AUV to go toward this point.

### 4.3.2 Explication of the Hough technique

There is no technique which can detect segments directly, but there is one which can detect straight lines efficiently. This technique is named “the Hough transform”. A new more complicated technique called “generalized Hough transform“ allows many different shapes to be identified but we will only use the first of them.

The rule of this technique is that it does a space modification. In this space, called “polar space” or “Hough space”, a straight line can be described by a couple of values (theta ; rho)[11]. Theta represents the angle between the x-axis and a perpendicular to the straight line, and rho represents the minimum distance between the straight line and the origin.

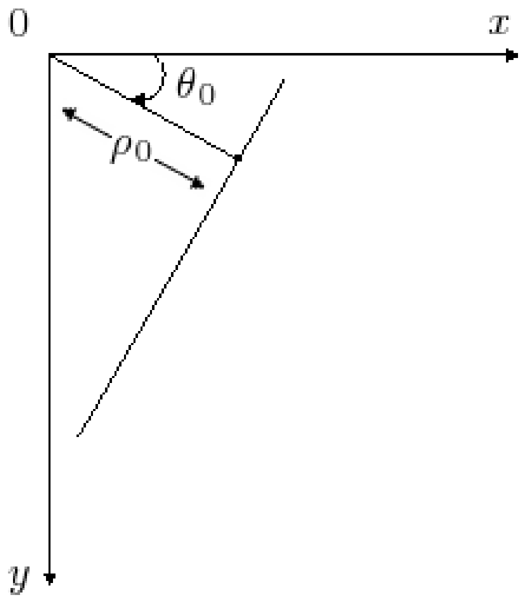


***Figure 24 : a couple (Rho ; Theta) describes a straight line***

[11] in mathematics : ( θ ; ρ)

The Hough technique works in exactly the wrong way: we don't have straight lines, but only points. We need to find the straight lines.

The nicety of this technique is that for each point that matches to a edge, we calculate the couple (Rho; Theta) of an infinite number of lines which go through this point and draw in the Hough space (Polar space) the couple found.

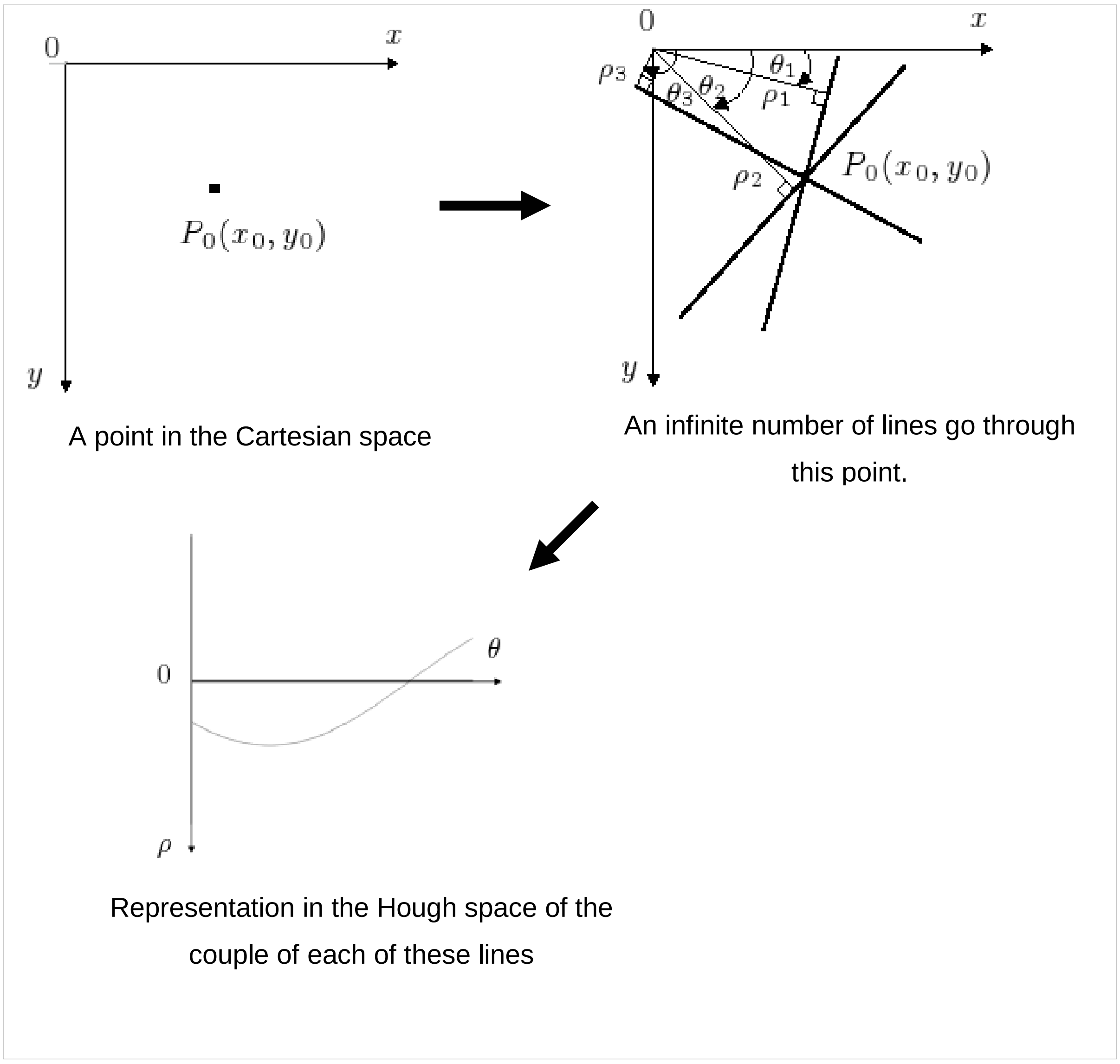

0
x
y
$P_0(x_0, y_0)$
A point in the Cartesian space
0
x
y
$\rho_3$
$\theta_3$
$\theta_2$
$\theta_1$
$\rho_1$
$\rho_2$
$P_0(x_0, y_0)$
An infinite number of lines go through this point.
0
$\theta$
$\rho$
Representation in the Hough space of the couple of each of these lines

***Figure 25 : transformation of one point into the Hough space***

So, one point in Cartesian space is described by a sinusoidal curve in the Hough space.
Now, instead of transforming only one point, make the same operation with three aligned points. Thus, the transformation of these three points draws space three sinusoidal curves in the Hough, but these curves are very particular because they have one common point. This point of Hough space, couple (Rho; Theta), describes the straight line which goes through the three points.

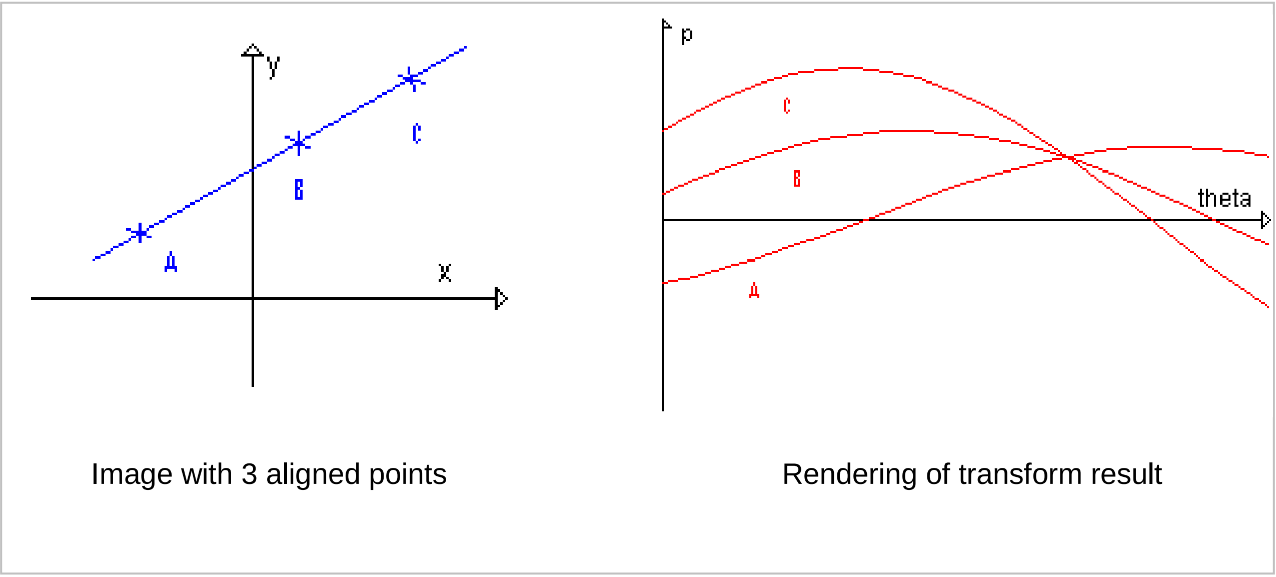


***Figure 26 : rendering of 3 aligned points in Hough space***

### 4.3.3 Algorithm

Before applying this process, the image is already converted into greyscale order to extract edges with the Sobel algorithm. Thus the Hough transform is applied to each pixel which is located on an edge.

This process creates a similar image than the right schema of Figure 26. The first four points of the Hough space, which are the point of intersection of the largest number of sinusoids and respect the rules, are extracted. The rules are then if two points own about the same Theta (the corresponding two straight lines in Cartesian space are parallel) and the same Rho (the two straight line are at the same distance from the origin), the algorithm considers that these two points corresponding to the same straight line.

The next step consists in calculating the points of intersection of these lines, if the angle between them is between 45° and 135° (PI/ 4 and 3*PI/4).

If we succeed in finding the 4 points of intersection, then the zone is found and an approximate of its centre is calculated by making the average of the coordinate of these four points.

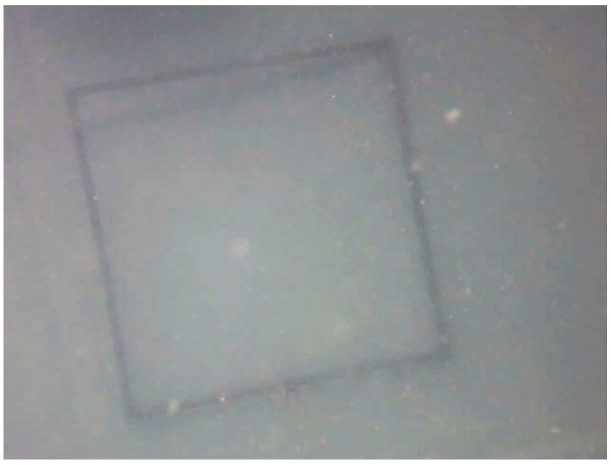


***Figure 27 : input image***

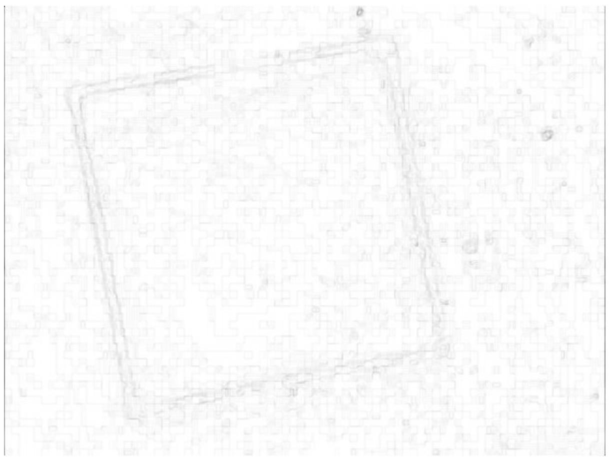


***Figure 28 : image after edge detection***

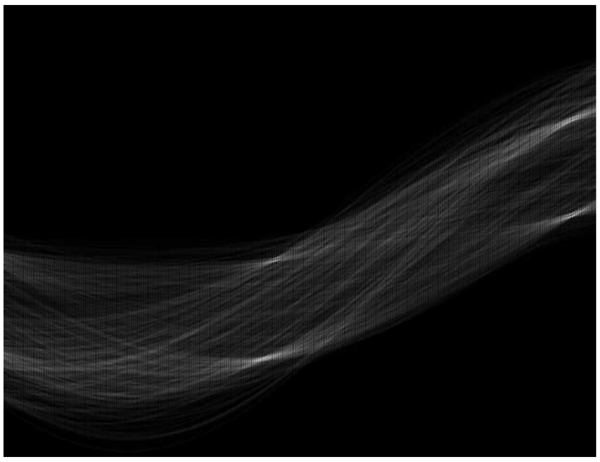


***Figure 29 : rending of the image in the Hough space***

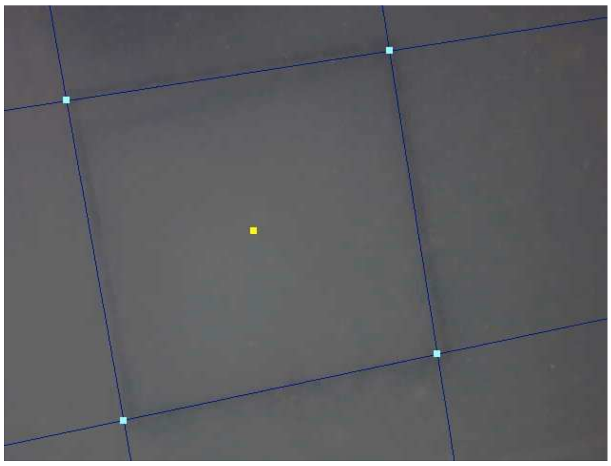


***Figure 30 : output image***

### 4.3.4 Optimization

This process costs a lot of time for the processor because for each pixel situated on an edge, it calculates the cosine function about 150 times (variable number according to the precision). For instance to produce Figure 29, it took 4 seconds and executed the cosine function 1500 times for each edge pixel.

If the process takes too much time, the robot can't reach the object, because during the computation, the robot continues to move.

To reduce this time, we can decrease the precision or if we want to be as precise as possible, another technique suggested by my supervisor can be used: when the robot is initialized, we calculate a large number of different cosines and memorize them in an array. Thereafter rather than calculate the cosine, we search for its value in the array.

## Technical balance sheet

The first two tasks are completely functional; indeed the robot detects well and moves well towards the objects. But we should not forget that even if the task which consists in detecting the target is okay, some software and material modifications remain to be carried out. Indeed, to simplify things, rather than place the circular target anywhere in the swimming pool, we put it at the height of the robot. It will be necessary to re-examine the position of the motors in order that the robot can move sideways and set up the rotation of the Webcam so that the robot can also see the bottom of the swimming pool in front of it.

Concerning the third task, there the zone is placed vertically at the height of the robot rather than on the surface of water but it is not the only problem because, when the robot approaches closer to the zone, it does not detect it any more owing to the fact that it does not see it any more in its totality. There are two solutions to this problem: software which consists in re-examining the algorithm so that the vehicle can detect the zone even if it does not see it entirely and a second material solution which consists in changing the Webcam in order to have a better field of vision or to set up an arrangement of lens. The second solution seems more suitable to me because I tried to find a software solution to this problem but I gave up by seeing that it required so much arithmetic and too much time for the processor. Moreover according to what I saw on the Internet, it is not inevitably complicated to set up a lens arrangement which will improve the field of vision.

This project is not yet finished because there are a few tasks to be set up. It is not going to create too many problems because these tasks should re-use the algorithms which I programmed, but another part which seems difficult to me will be the map generation of the objects situated in the tank.

## Conclusion

First of all, one of the most important things to me was to make my placement abroad. This placement made me improve my English, made me more responsible and mature and let me discover a new culture,
After these weeks in Scotland I am very sure that English is essential to work as an engineer or to live abroad as I did.

The second important aspect was the discovery of a new computing domain: image processing. I really liked the programming of algorithms allowing the robot to be autonomous. I think that, with the development of computing power, this sector has a great future and will probably be used one day in everyday life.

Finally, this placement let me involved in team work and more organised in the division of the tasks with Guillaume.

## Reference

[1] Sauc-e’s website: http://dstl.gov.uk/news_events/competitions/sauce/index.php

[2] Website of the k8055 library for Linux: http://libk8055.sourceforge.net

[3] Website of the webcam driver: http://mxhaard.free.fr

[4] Thomas Braunl, Mobile Robot Design and Applications with Embedded Systems, (2006)

[5] James L. Crowley, Vision par Ordinateur, (2006)

[6] S.El mejdani, R. Egli, F.Dubeau, Champ de hauteurs de la transformée de Hough standard, (2005)