



## NAUTILUS - RAPPORT

---

Dylan Bideau, Julien Turpin, Pierre Bogrand, Guillaume Vincenti

---

10 Avril 2018

# Sommaire

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Présentation du projet</b>	<b>3</b>
<b>3</b>	<b>Cahier des charges</b>	<b>4</b>
3.1	Analyse Fonctionnelle . . . . .	4
3.1.1	Structure . . . . .	4
3.1.2	Commandabilité . . . . .	4
3.1.3	Milieu d'utilisation . . . . .	4
3.1.4	Énergie . . . . .	5
3.1.5	Motorisation . . . . .	5
3.1.6	Acquisitions . . . . .	5
<b>4</b>	<b>Acquisition et Commandabilité</b>	<b>6</b>
4.1	Raspberry . . . . .	6
4.1.1	Cameras . . . . .	6
4.1.2	Capteur de Pression/Température . . . . .	9
4.1.3	Central Inertielle . . . . .	9
4.2	PC . . . . .	10
4.2.1	Interface . . . . .	10
4.2.2	Tunnel SSH . . . . .	12
4.2.3	Vidéos . . . . .	13
4.2.4	Capteurs et Moteurs . . . . .	16
4.2.5	Affichage 3D . . . . .	17
4.2.6	Cartographie . . . . .	19
<b>5</b>	<b>Conclusion</b>	<b>21</b>
	<b>Bibliographie</b>	<b>22</b>
	<b>Table des figures</b>	<b>23</b>

# **Chapitre 1**

## **Introduction**

Les fonds marins réunissent aujourd’hui de nombreux secteurs et enjeux, tant professionnels que particuliers. On y retrouve entre autre l’exploration sous-marine, la surveillance et maintenance d’installations professionnelles, ainsi que la cartographie des fonds marins. Tout ces domaines demandent le développement de solutions techniques plus rentables et pratiques qu’une intervention humaine. Notre projet propose ainsi un ROV (Remotely Operated Vehicle) polyvalent et simple d’utilisation à cet effet.

## Chapitre 2

# Présentation du projet

Un ROV est un robot sous-marin contrôlé à distance et permettant une acquisition d'informations, visuelles ou à partir de capteurs. Notre projet de ROV filoguidé, Nautilus, sera transportable et pilotable à l'aide d'un ordinateur portable. Il permettra d'observer facilement des installations ou des fonds marins à l'aide de caméras. Disposant également de fonctions avancées, le Nautilus sera en mesure de recréer le fond marin d'une zone géographique déterminée par l'utilisateur à partir d'une batterie de photographies prises lors de la phase d'exploration. Les différentes fonctionnalités du Nautilus en font ainsi un outil polyvalent, permettant exploration, maintenance et cartographie des fonds.

# Chapitre 3

## Cahier des charges

### 3.1 Analyse Fonctionnelle

#### 3.1.1 Structure

Facilement transportable et peu encombrant.

**Contraintes :**

- Poids : 2-3kg
- Dimension : 300\*200\*150mm
- Étanche de norme IP 68

#### 3.1.2 Commandabilité

Commandé à distance par une liaison filaire.

**Contraintes :**

- Câble : 15m
- Carte intégrée dans le ROV
- FPV (First Person View)
- Piloté au clavier

#### 3.1.3 Milieu d'utilisation

Adapté aux contraintes imposées par son environnement.

**Contraintes :**

- Eau non salé (moins de 1 g de sels dissous par kilogramme d'eau)
- Eau translucide (transmittance de la lumière entre 75% et 95%)
- Lieu : Piscine, lac
- Écoulement laminaire
- Courant marin inférieur à 2 noeuds
- Profondeur de 10m (résistant à 2 bars)

### **3.1.4 Énergie**

Être entièrement autonome.

**Contraintes :**

- Autonomie de 20 minutes

### **3.1.5 Motorisation**

Être mobile une fois immergé.

**Contraintes :**

- Propulsion électrique
- Déplacement horizontal (Vitesse maximale de 1m/s)
- Déplacement vertical (Vitesse maximale de 0.5m/s)
- Direction droite/gauche à 360 degrés

### **3.1.6 Acquisitions**

Acquérir et transmettre l'information.

**Contraintes :**

- Acquisition et retransmission d'un signal vidéo
- Acquisition et stockage de photographies
- Mesure de la pression
- Mesure de la position relative avec signaux GPS

## Chapitre 4

# Acquisition et Commandabilité

Dans un second temps, nous devons relier les différents éléments de notre ROV sur une carte et ensuite traiter les informations reçus pour pouvoir agir sur les moteurs vus précédemment. Nous avons choisi la Raspberry.

### 4.1 Raspberry

Une partie de la programmation et des calculs est effectué sur une Raspberry PI 3 qui supportait tout les types de connections que l'on voulait, en voici la description.



FIGURE 4.1 – Raspberry

#### 4.1.1 Cameras

Nous avons 2 caméras qui permettent, l'une la direction (vision frontale) et l'autre la cartographie (vision par dessous). Dans un premier temps, détaillons leur connexion entre la Raspberry et le traitement effectué par celle-ci.

## Logitech C170

La première est une webcam Logitech C170, que nous avons démonté pour l'assemblage, relié en USB à la Raspberry.

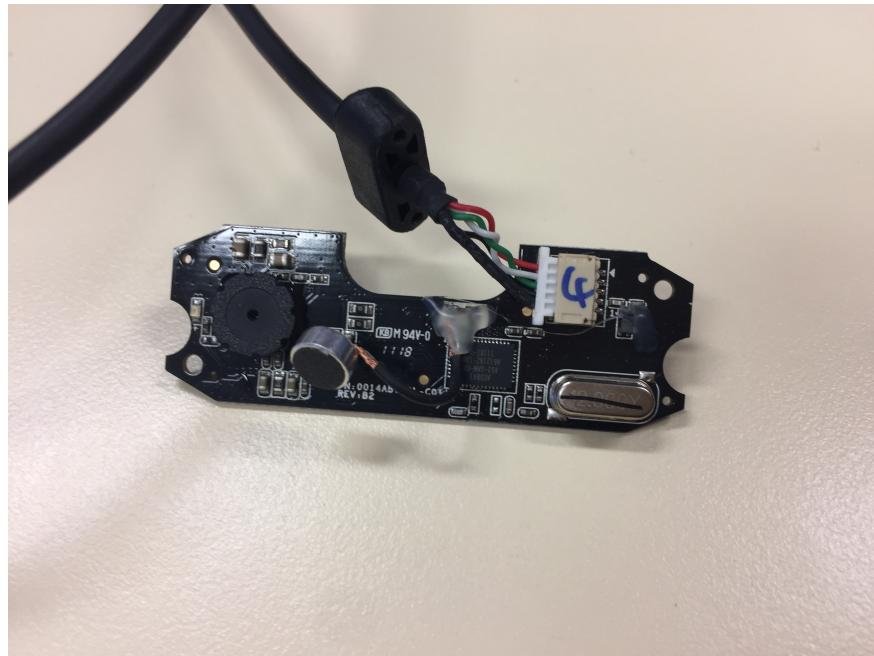


FIGURE 4.2 – Logitech C170

Nous l'avons choisi car le pilote de celle ci est déjà installé nativement sur la Raspberry. Nous utilisons motion qui permet d'envoyer le flux vidéo venant de la caméra et de le diffuser en ligne sur notre adresse local [1]. En voici le résultat sur un navigateur :

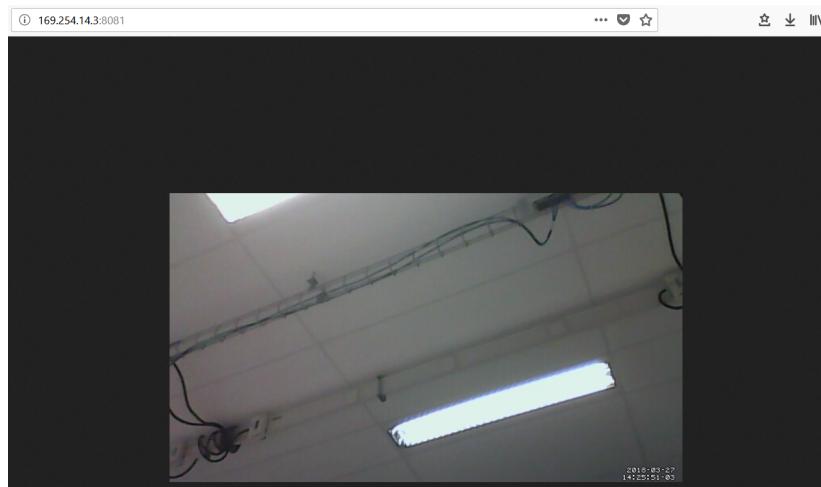


FIGURE 4.3 – Photo Caméra Frontale

Cette vidéo sera récupérée par l'interface (expliquée dans le chapitre associé) en 640\*360.

Caméra V2

La deuxième est un module caméra pour Raspberry [2] qui se raccorde directement par une nappe (un bus de type CSI-2).

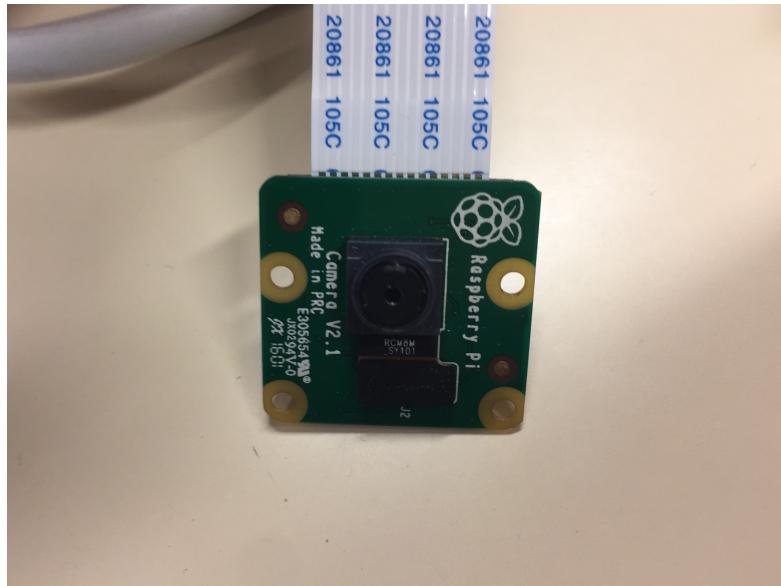


FIGURE 4.4 – Module Camera V2 Raspberry

Elle se paramètre en python avec les librairies données par le constructeur. De même que l'autre caméra, nous renvoyons un flux vidéo en ligne sur notre adresse local [3] mais cette fois-ci sur un autre port.

Le résultat sur un navigateur :

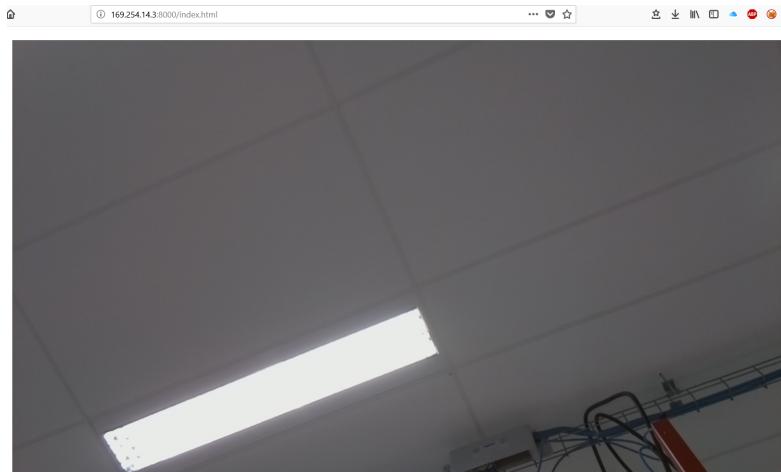


FIGURE 4.5 – Photo Caméra du dessous

La vidéo est diffusée en 1920\*1080 et affichée par l'interface.

#### **4.1.2 Capteur de Pression/Température**

#### **4.1.3 Central Inertielle**

Julien

## 4.2 PC

Maintenant que nous avons relié tous nos moteurs, caméras et capteurs à la Raspberry ainsi qu'un premier traitement des informations, nous allons voir comment nous traitons cela sur le PC.

### 4.2.1 Interface

En premier lieu parlons de l'interface, celle ci à pris différentes formes au cours du temps, ici nous présenterons que la dernière version. Toute la partie PC a été programmé en JAVA [4], l'interface utilise la bibliothèque graphique Swing qui nous permet de gérer l'affichage facilement que ce soit pour la vidéo ou pour les interactions. Lorsque l'application est lancée, nous arrivons donc dans un premier menu :

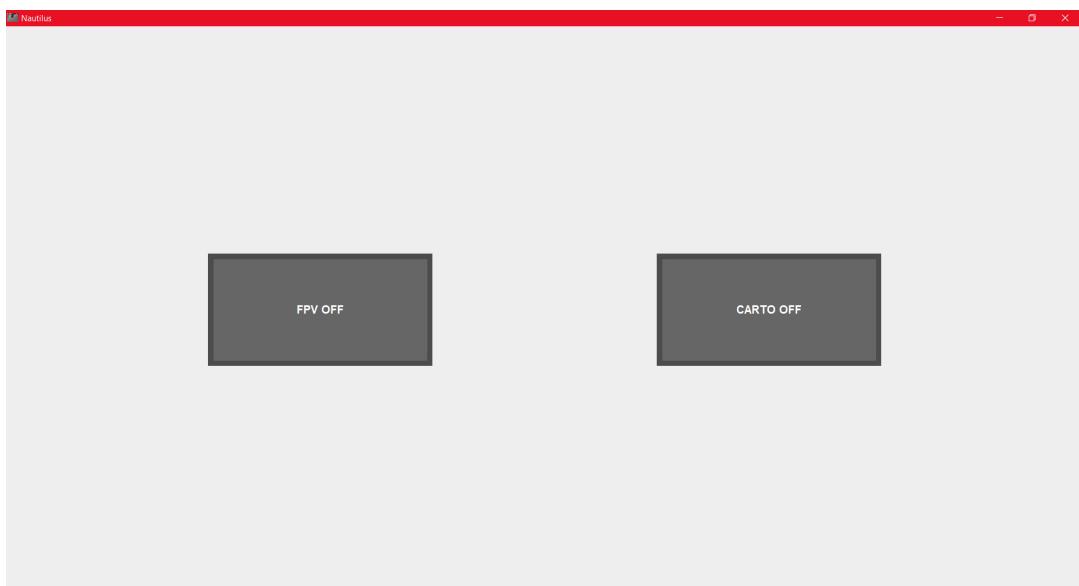


FIGURE 4.6 – Premier Menu Interface

Le bouton de Droite mène à une partie que nous développerons dans la partie 4.2.6 liée à la Cartographie.

Nous avons donc crée une fenêtre JFrame :

```
1 Interface inter = new Interface("Nautilus", 0, 0, 1920, 1080, true);  
2
```

Elle est paramétrée de façon à prendre tous l'écran, ici 1920\*1080, les objets se trouvant dans cette fenêtre sont gérés pour se placer en fonction de la taille de l'écran.

Le manager s'appelle GridBagLayout, il nécessite de paramétrier chaque objet. Ce manager crée une grille qui se construit en fonction des paramètres de chaque objet qu'il contient.

Prenons exemple du premier bouton FPV :

```
1 axPanel1.setMinimumSize(new Dimension(400,210));
2 axPanel1.setMaximumSize(new Dimension(400,210));
3 axPanel1.setPreferredSize(new Dimension(400,210));
4 c.fill = GridBagConstraints.BOTH;
5 c.anchor = GridBagConstraints.CENTER;
6 c.gridx = 0;
7 c.gridy = 0;
8 c.weighty = 0.0;
9 c.weightx = 0.0;
10 c.gridwidth = 1;
11 c.gridheight = 1;
12 c.insets = new Insets(0, 0, 0, 200);
13
```

Les 3 premières lignes correspondent à la taille du bouton, que nous avons choisi ici de garder fixe.

La ligne 4 n'est pas utile dans ce cas mais permet de correctement redimensionner l'objet lorsque la fenêtre change de taille.

La ligne 5 fixe l'objet au centre de la partie qui lui a été alloué.

La ligne 6 et 7 donne la ligne et la colonne où doit se situer l'objet.

La ligne 8 et 9 définissent des poids en x et y qui sont utilisé lors d'un redimensionnement, cela permet de donner plus de poids à un objet plutôt qu'à un autre. Nous ne l'utilisons pas d'où la valeur 0.

La ligne 10 et 11 permettent de définir combien de ligne et combien de colonne va prendre notre objet.

La ligne 12 insère une marge dans l'ordre suivant (margeSupérieure, margeGauche, margeInférieure, margeDroite).

Chaque objet de notre interface est défini de cette façon.

Ensuite il y a le bouton de Gauche qui lance le système complet. Un nouveau menu remplace le précédent :

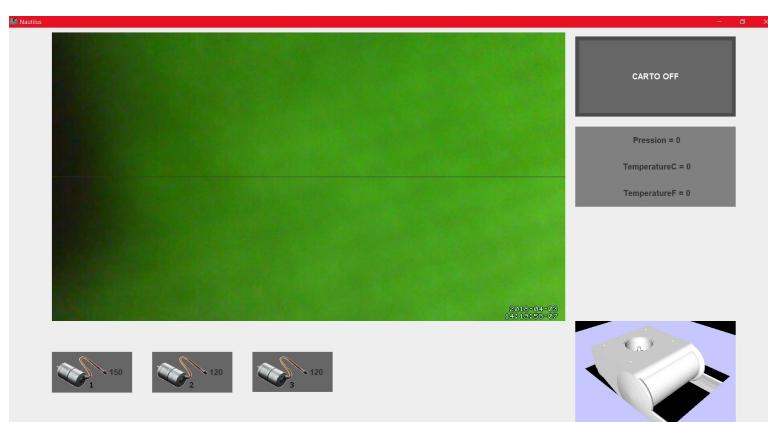


FIGURE 4.7 – Photo Deuxième menu interface

Nous avons maintenant la FPV en haut à gauche, le bouton à droite est le même précédemment, les informations des capteurs sont affichées à droite, les commandes envoyées aux moteurs sont

en bas et pour finir un affichage 3D (4.2.5) avec JAVA3D en bas à droite. Toutes les informations étant actualisées en temps réel avec la Raspberry. Nous allons voir comment.

#### 4.2.2 Tunnel SSH

Nous avons choisi d'échanger les données avec la Raspberry par SSH, pour cela on utilise la bibliothèque Jcraft, plus précisément jsch. Au lancement de l'application, il y a 5 tunnels qui se créent (1 pour chaque caméras, 1 pour chaque capteurs et 1 pour les moteurs), ils sont créés au tout début pour ensuite permettre de transmettre directement les données sans refaire la procédure de connexion, le système gagne en rapidité.

Détaillons maintenant la procédure qui permet de créer ses tunnels. Comme nous sommes en JAVA, une seule classe permet de créer un tunnel et cette classe est ensuite instanciée autant de fois que l'on veut.

Voici le code (Disponible dans /SSH/Exec.java) de création d'un tunnel :

```

1 try {
2     JSch jsch=new JSch();
3     this.session=jsch.getSession("pi", "169.254.14.03", 22);
4     UserInfo ui=new MyUserInfo();
5     this.session.setUserInfo(ui);
6     this.session.connect();
7     this.channel = this.session.openChannel("exec");
8     ((ChannelExec)this.channel).setCommand(this.Commande);
9     this.channel.setInputStream(null);
10    ((ChannelExec)this.channel).setErrStream(System.err);
11    this.in=this.channel.getInputStream();
12    this.channel.connect();
13    byte[] tmp=new byte[1024];
14    while(true){
15        while(this.in.available()>0){
16            int i=this.in.read(tmp, 0, 1024);
17            if(i<0)break;
18            System.out.print(new String(tmp, 0, i));
19            this.retour=new String(tmp, 0, i);
20        }
21        if(channel.isClosed()){
22            if(this.in.available()>0) continue;
23            System.out.println("exit-status: "+channel.getExitStatus());
24            break;
25        }
26    }
27 }
28 catch(Exception e){
29     System.out.println(e);
30 }
31

```

De la ligne 1 à 5, le tunnel est créé et les informations tel que le nom d'utilisateur, le mot de passe, l'adresse IP et le port sont ajouté aux couches correspondante du tunnel, puis ligne 6 la session est lancée.

A la ligne 7, nous ouvrons un canal d'exécution de commande et ligne 8 nous donnons à ce canal la commande que nous voulons envoyée.

Les lignes 9,10 et 11 définissent où vont les données entrées, sorties et sorties erreur. Dans notre

cas l'entrée ne nous intéresse pas car cela passe par une commande, la sortie normal sera stockée et la sortie erreur renvoyée sur l'afficheur d'erreur du système (la console). La ligne 12 lance la connexion du canal, c'est ici que la commande est envoyée.

De la ligne 13 à 20 nous permet de récupérer les valeurs retournées par la Raspberry et de les stocker pour ensuite pouvoir les traiter.

Puis les dernières lignes, permettent de gérer le cas où le canal est coupé et nous renvoyer d'où vient l'erreur (par exceptions).

Ces tunnels nous permettent d'envoyer des commandes à la Raspberry et de pouvoir récupérer le retour de cette commande. C'est par ce moyen que nous récupérons presque tous. L'exception étant pour la vidéo IP. Abordons ce sujet.

#### 4.2.3 Vidéos

Nous avons donc 2 flux vidéos à récupérer depuis une adresse IP. Tout d'abord nous devons introduire quelque chose que nous utilisons partout dans notre code : le multi-Thread. Cela permet de lancer plusieurs actions en même temps, c'est ce qu'il se passe avec la récupération des flux vidéos. Chaque image des flux vidéos est récupérée, traitée puis affichée en temps réel sans interrompre le reste du programme, tout comme l'envoie de chaque commande.

Pour les flux vidéos, c'est une connexion http qui est effectuée, pour cela la méthode connect est appelée :

```
1 public void connect()
2 {
3     try
4     {
5         URL u = new URL(useMJPEGStream?mjpgURL:jpgURL);
6         huc = (HttpURLConnection) u.openConnection();
7         InputStream is = huc.getInputStream();
8         connected = true;
9         BufferedInputStream bis = new BufferedInputStream(is);
10        dis= new DataInputStream(bis);
11        if (!initCompleted) initDisplay();
12    }
13    catch(IOException e)
14    { //Relance la connexion si pas de connexion en attendant 60 sec
15        try
16        {
17            huc.disconnect();
18            Thread.sleep(60);
19        }
20        catch(InterruptedException ie)
21        {
22            System.out.println(ie);
23        }
24    }
25    catch(Exception e){;}
26}
27
```

De la ligne 5 à 10, la procédure de connexion http est effectuée.

Après la ligne 12, on gère les exceptions liées à la connexion.

A la ligne 10, les données récupérées sont stockées dans une variable globale à Caméra. Les informations vont être ensuite traité par initDisplay qui est appelée en ligne 11. Observons cette méthode :

```
1 public void initDisplay()
2 {
3     if (useMJPGStream) readMJPGStream();
4     else
5     {
6         readJPG();
7         disconnect();
8     }
9     imageSize = new Dimension((image.getWidth(this)*2), image.getHeight(this)
10 *2);
11     setPreferredSize(imageSize);
12     parent.validate();
13     initCompleted = true;
14 }
```

Le premier If/Else permet de distinguer le cas où le flux serait une vidéo ou juste une photo. Dans notre cas c'est un flux vidéo composé d'image JPG.

Le reste de la méthode permet de préparer l'affichage des images.

Nous sommes toujours dans la procédure de connexion, la lecture de la première image récupérée est faite. Pour cela la méthode readMJPGStream qui appelle readJPG permet de décoder l'image et de la stocker dans la variable image. Voici les 2 méthodes :

```
1 public void readMJPGStream()
2 {
3     readLine(4, dis); //enlève les 3 premières lignes
4     readJPG();
5     readLine(1, dis); //enlève les 2 dernières lignes
6 }
7
8 public void readJPG()
9 {
10     try {
11         JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(dis);
12         image = decoder.decodeAsBufferedImage();
13     } catch (Exception e) {
14         disconnect();
15     }
16 }
17 }
```

La procédure de décodage d'une image JPEG est faite par une bibliothèque externe.

Après cette procédure de connexion, les 2 méthodes ci dessus sont appelées en boucle et l'affichage mis à jour :

```
1 public void readStream()
2 {
3     try {
4         if (useMJPEGStream) {
5             while(true) {
6                 readMJPEGStream();
7                 parent.repaint();
8             }
9         }
10    } catch (Exception e) {}
11 }
12
```

L'affichage étant gérée par la bibliothèque Swing, c'est la méthode paint() qui affiche l'image et de plus trace une ligne qui en fonction des angles d'Euler renvoyés par la centrale inertie :

```
1 public void paint(Graphics g)
2 {
3     if (image != null)
4         image=scale(image, 2);
5         g.drawImage(image, 0, 0, this);
6         Graphics2D g2 = (Graphics2D) g;
7         double alpha = Interface.rotZ* Math.PI/180f;
8         int a = image.getHeight();
9         int b = image.getWidth();
10        double S =(b/2)*(Math.sin(alpha))*(Math.cos((Math.PI/2)-alpha));
11        double T =(b/2)*(Math.sin(alpha))*(Math.sin((Math.PI/2)-alpha));
12        g2.draw(new Line2D.Double(S,(a/2)-T, b-S,(a/2)+T));
13    }
14
```

L'image est redimensionnée pour le bien de l'affichage et la ligne est tracé avec 2 points calculés par rapport à une rotation avec une origine au centre de l'image.

Nos 2 flux vidéos sont donc récupérés et peuvent donc être appellés par un JPanel n'importe où.

#### 4.2.4 Capteurs et Moteurs

##### Capteurs

Nous avons déjà précédemment préparé la Raspberry pour qu'elle renvoie les informations que l'on veut en fonction d'une commande. Il nous suffit maintenant d'envoyer en temps réel et en boucle infini les commandes (capteur de pression et centrale inertie) grâce au système de multi-thread.

Expliquons la méthode pour le capteur de pression. Tout d'abord il faut créer le thread lié à la classe qui sera exécuté en boucle, puis on lance le thread avec la méthode, ce qui donne :

```
1 Pression ex4 = new Pression();
2 Thread a4 = new Thread(ex4);
3 a4.start();
4
```

La classe Pression doit être héritière de Runnable pour permettre le lancement en thread.

Observons dans la classe Pression, la méthode start :

```
1 Interface.exPression.Commander("python MS5803_14BA.py");
2 String[] a = Interface.exPression.retour.split("/");
3 this.Pression=a[0];
4 this.TemperatureC=a[1];
5 this.TemperatureF=a[2];
6 Interface.pre=Pression;
7 Interface.tempC=TemperatureC;
8 Interface.tempF=TemperatureF;
9 Interface.pression.setText(" Pression = "+Interface.pre);
10 Interface.pression.repaint();
11 Interface.temperatureC.setText(" TemperatureC = "+Interface.tempC);
12 Interface.temperatureC.repaint();
13 Interface.temperatureF.setText(" TemperatureF = "+Interface.tempF);
14 Interface.temperatureF.repaint();
15
```

Nous commençons en ligne 1 par envoyer la commande grâce au tunnel précédemment créé pour le capteur de Pression puis nous récupérons les valeurs sous forme de chaînes de caractères. Enfin nous mettons à jour les valeurs globales et l'affichage.

##### Moteurs

Pour les moteurs, on envoie juste une commande à la Raspberry :

```
1 Interface.exMoteur.setCommande("echo 0="+m1+" > /dev/servoblaster & echo 1="+m2+
> /dev/servoblaster & echo 2="+m3+" > /dev/servoblaster");
2 Thread a1 = new Thread(Interface.exMoteur);
3 a1.start();
4 while( a1.isAlive() ) {}
5 Interface.m1=this.m1;
6 Interface.m2=this.m2;
7 Interface.m3=this.m3;
8 Interface.moteur1.setText(""+Interface.m1+"      ");
9 Interface.moteur1.repaint();
```

```

10 Interface.moteur2.setText(" "+Interface.m2+" ");
11 Interface.moteur2.repaint();
12 Interface.moteur3.setText(" "+Interface.m3+" ");
13 Interface.moteur3.repaint();
14

```

---

On oublie pas de mettre à jour au passage l'affichage.

Ce code est donc exécuté à chaque fois que l'on appuie sur une des touches de directions.

Nous avons défini les touches suivantes :

Monter/Descente : Z/S

Gauche/Droite : Flèche Gauche/Droite

Monter/Descente : Z/S

Arrête d'urgence : Barre Espace

#### 4.2.5 Affichage 3D

Nous utilisons JAVA3D pour afficher le modèle qui est extrait de Solidworks.

Tout d'abord nous devons créer un canvas 3D qui contiendra notre objet 3D et nous l'insérons au milieu d'un JPanel :

```

1 Canvas3D canvas3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
2 this.add(canvas3D, BorderLayout.CENTER);
3

```

---

Puis nous créons un simple univers qui contient notre canvas3D.

Nous devons maintenant positionner notre point d'observation pour avoir une vue correcte :

```

1 OrbitBehavior orbit = new OrbitBehavior(canvas3D, OrbitBehavior.REVERSE_ROTATE);
2 orbit.setRotXFactor(0); //or any other value
3 orbit.setRotYFactor(0);
4 orbit.setTransXFactor(0);
5 orbit.setTransYFactor(0);
6 orbit.setSchedulingBounds(new BoundingSphere());
7 simpleU.getViewingPlatform().setViewPlatformBehavior(orbit);
8 ViewingPlatform vp = simpleU.getViewingPlatform();
9 TransformGroup steerTG = vp.getViewPlatformTransform();
10 Transform3D t3d = new Transform3D();
11 steerTG.getTransform(t3d);
12 t3d.lookAt(new Point3d(-1.2, 1.2, 1.2), new Point3d(0, 0, 0), new Vector3d(0, 1, 0));
13 t3d.invert();
14 steerTG.setTransform(t3d);
15

```

---

De la ligne 1 à 6, nous définissons un mouvement orbital de la caméra puis pour le moment nous fixons les rotations et les translations (on pourra les débloquer si on le veut)

Les lignes 9 à 14 permettent de définir la position initiale de notre point d'observation.

Nous devons ensuite créer ce qu'on appelle une scène, c'est dans cette scène que tous nos objets 3d se trouveront :

```

1 BranchGroup scene = createSceneGraph(simpleU);

```

```

2 scene.compile();
3 simpleU.addBranchGraph(scene);
4

```

---

Les 2 lignes suivant compile la scène et rattache notre scène à l'univers.

Maintenant nous devons crée cette scène 3d qui contiendra notre objet mais aussi la lumière, notre eau fictive et les mouvements possible.

Tous ce passe dans la méthode createSceneGraph.

```

1 public BranchGroup createSceneGraph(SimpleUniverse simpleU)
2 {
3     BranchGroup parent = new BranchGroup();
4     BoundingSphere bounds = new BoundingSphere(new Point3d(), 100);
5     Light ambientLight = new AmbientLight(new Color3f(Color.white));
6     ambientLight.setInfluencingBounds(bounds);
7     parent.addChild(ambientLight);
8
9     Light directionalLight = new DirectionalLight(
10         new Color3f(Color.white),
11         new Vector3f(1, -1, -1));
12     directionalLight.setInfluencingBounds(bounds);
13     parent.addChild(directionalLight);
14

```

---

La ligne 3 crée un objet parent qui contiendra tous nos objets.

De la ligne à 4 à 7, on crée une lumière ambiante de couleur blanche

Puis de la ligne 9 à la ligne 13, une lumière directionnel pour augmenter la luminosité dans la même direction que notre point d'observation.

Nous ne détaillerons pas ici le code de la création de l'eau, le principe étant qu'on a crée 4 plans positionnés correctement autour du ROV et on lui applique une texture bleu (que l'on peut changer facilement). Et on oublie pas de l'ajouter à l'objet parent.

Maintenant autorisons les mouvements de la souris pour notre objet :

```

1 TransformGroup mouseTransform = new TransformGroup();
2 mouseTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
3 mouseTransform.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
4 mouseTransform.addChild(loadWavefrontObject());
5

```

---

Et en ligne 4 nous ajoutons notre ROV. La méthode loadWavefrontObject() permet de récupérer un objet 3D à partir d'un fichier .obj.

Pour la fin de cette méthode, les commentaires décrivent les opérations effectuées

```

1 // Création de l'homothétie (homothetie)
2 Transform3D homothetie = new Transform3D();
3 homothetie.setScale(Interface.scale);
4
5 // Création de la transformation (translation)
6 Transform3D translation = new Transform3D();
7 translation.setTranslation(new Vector3f(0f, 0f, 0f));

```

---

```

8 translation.mul(homothetie);
9
10 // Création de la rotation X(rotation)
11 Transform3D rotationX = new Transform3D();
12 rotationX.rotX(Interface.rotX * Math.PI/180f);
13 rotationX.mul(translation);
14
15 // Création de la rotation Y(rotation)
16 Transform3D rotationY = new Transform3D();
17 rotationY.rotY(Interface.rotY * Math.PI/180f);
18 rotationY.mul(rotationX);
19
20 // Création de la rotation Z(rotation)
21 Transform3D rotationZ = new Transform3D();
22 rotationZ.rotZ(Interface.rotZ * Math.PI/180f);
23 rotationZ.mul(rotationY);
24
25 this.rotationGroup = new TransformGroup(rotationZ);
26
27 // Autorisation de modifier les transformations
28 rotationGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
29 rotationGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
30
31 // Ajout au graphe de la scène
32 rotationGroup.addChild(mouseTransform);
33 parent.addChild(rotationGroup);
34
35 return parent;
36

```

Notre scène renvoie donc tous les objets qu'on a crée précédemment.

Maintenant nous voulons que notre ROV 3D soit capable de se mouvoir en fonction des angles d'Euler renvoyés et stockés dans notre interface.

Pour cela on a crée une méthode mouvement :

```

1 public void mouvement(double rotX, double rotY, double rotZ, Vector3d trans,
2                      double scale)

```

Elle prend donc en argument, les 3 rotations, un vecteur des translations (sur tous les axes) et un paramètre multiplicateur d'échelle.

Cette méthode modifie la matrice 4\*4 contenant la matrice de rotation, translation et de mise à l'échelle.

Puis effectue les transformations de cette matrice.

#### 4.2.6 Cartographie

Cette partie n'a pas été beaucoup développé mais un premier système a été crée mais pas intégré dans la dernière version de l'interface.

Grâce à l'appui sur une touche, une photo de la camera du dessous est enregistrée dans un dossier. Voici le code qui permet d'enregistrer notre photo en jpg :

```

1  BufferedImage img = new BufferedImage(axPanel.getWidth() , axPanel.getHeight() ,
2  	BufferedImage.TYPE_INT_RGB) ;
3  axPanel.print(img.getGraphics()) ;
4  try {
5      ImageIO.write(img, "jpg", new File(nomPhoto+".jpg")) ;
6  }
7  catch (IOException e5) {
8      e5.printStackTrace() ;
9 }
```

Le système n'est pas fini mais dans l'absolu chaque photo sera enregistrée avec un nom différent avec les données de localisation dans les méta-données de la photo.

Ensuite il y a un autre bouton (non affiché sur la dernière interface) qui ouvre une autre fenêtre avec toutes nos photos positionnées, en fonction des coordonnées GPS, sur une carte.

On peut cliquer sur chaque photo pour les agrandir dans une nouvelle fenêtre.

Tous les codes se trouvent dans le sous-dossier /Carto.

## Chapitre 5

### Conclusion

# Bibliographie

- [1] Nautilus. Adresse video caméra frontale. 169.254.14.03:8081, 2018.
- [2] Mouse Electronics. Module camera v2 raspberry. Mouse Electronics, 2018.
- [3] Nautilus. Adresse video caméra du dessous. 169.254.14.03:8000, 2018.
- [4] GitHub. Code java de l'interface. Github, 2018.

# Table des figures

4.1	Raspberry . . . . .	6
4.2	Logitech C170 . . . . .	7
4.3	Photo Caméra Frontale . . . . .	7
4.4	Module Camera V2 Raspberry . . . . .	8
4.5	Photo Caméra du dessous . . . . .	8
4.6	Premier Menu Interface . . . . .	10
4.7	Photo Deuxième menu interface . . . . .	11