



NAUTILUS - RAPPORT

Dylan Bideau, Julien Turpin, Pierre Bogrand, Guillaume Vincenti

10 Avril 2018

Sommaire

1	Introduction	2
2	Présentation du projet	3
3	Cahier des charges	4
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	Energie	5
3.1.5	Motorisation	5
3.1.6	Acquisitions	5
4	Motorisation et énergie	6
4.1	Moteurs brushless et ESC	6
4.2	Calibration et commande des ESC	8
4.3	Alimentation et montage	11
5	Acquisition et Commandabilité	12
5.1	Raspberry	12
5.1.1	Cameras	12
5.1.2	Capteur de Pression/Temperature	15
5.1.3	Central Inertielle	15
5.2	PC	16
5.2.1	Interface	16
5.2.2	Tunnel SSH	18
5.2.3	Videos	19
5.2.4	Capteurs et Moteurs	21
5.2.5	Affichage 3D	21
5.2.6	Cartographie	21
6	Références	22

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 emcombrant.

Contraintes :

- Poids : 2-3kg
- Dimension : 300*200*150mm
- Etanche 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
- Ecoulement laminaire
- Courant marin inférieur à 2 noeuds
- Profondeur de 10m (résistant à 2 bars)

3.1.4 Energie

Etre entièrement autonome.

Contraintes :

- Autonomie de 20 minutes

3.1.5 Motorisation

Etre 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 :

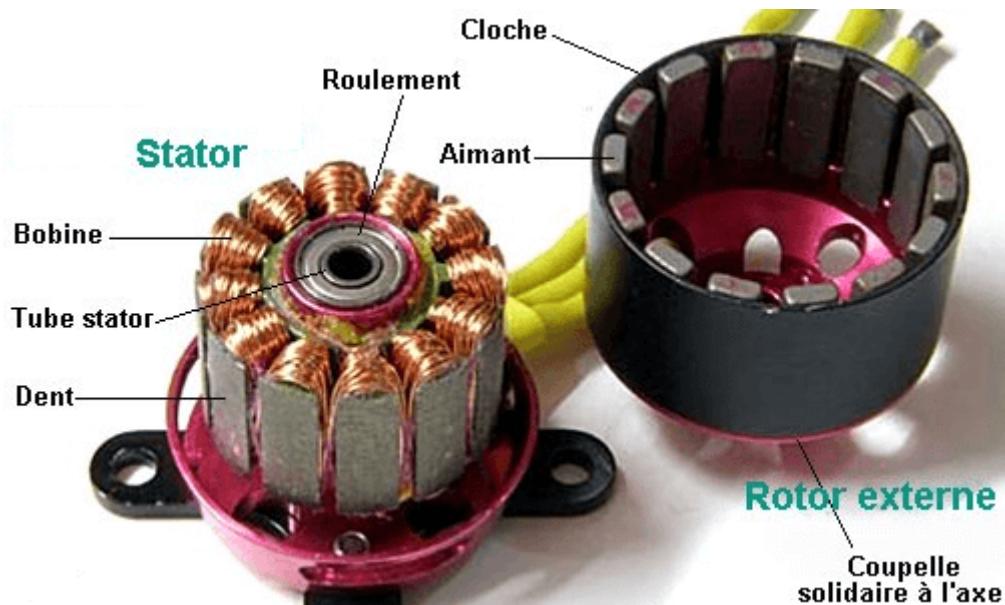
- Acquisition et retransmission d'un signal vidéo
- Acquisition et stockage de photographies
- Mesure de la pression
- Mesure de la position relative avec signaux GPS

Chapitre 4

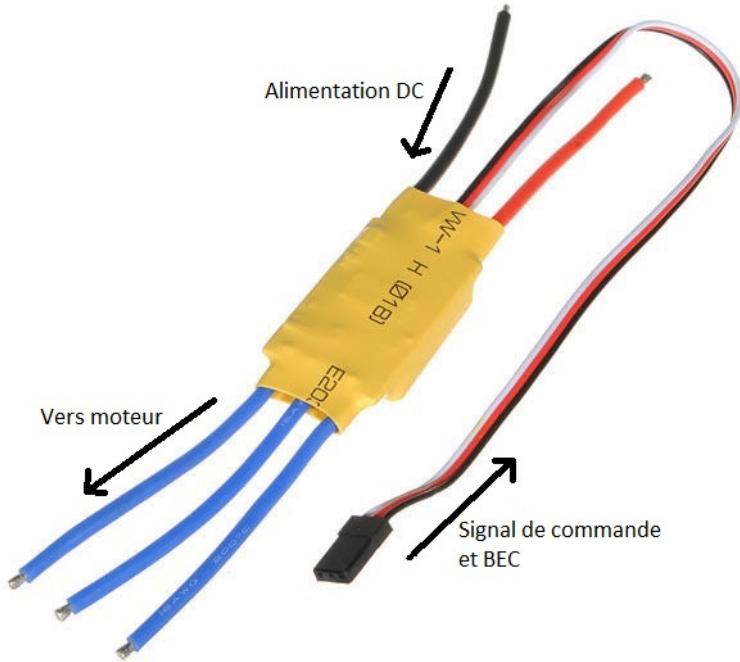
Motorisation et énergie

4.1 Moteurs brushless et ESC

Dans un premier temps, il a été question de la technologie des moteurs à utiliser. Après une étude des différentes solutions disponibles, nous avons finalement choisi des moteurs brushless (Référence 1). En effet, les moteurs brushless sont des machines synchrones auto-pilotées à aimants permanents et donc sans balais.



Le principal avantage de ces moteurs est qu'ils peuvent être utilisés immersés dans l'eau sans aucun traitement particulier au préalable. En revanche, un système électronique de commande doit assurer la commutation du courant dans les enroulements statoriques : les ESC, ou Electronic Speed Controllers. Un ESC transforme un signal d'alimentation continu, dans notre cas issu d'une batterie, en un signal triphasé envoyé ensuite au moteur brushless. Pour contrôler la vitesse de rotation du moteur, on envoie à l'ESC un signal de commande, généralement créneau, et dont le rapport cyclique définit la vitesse du moteur.

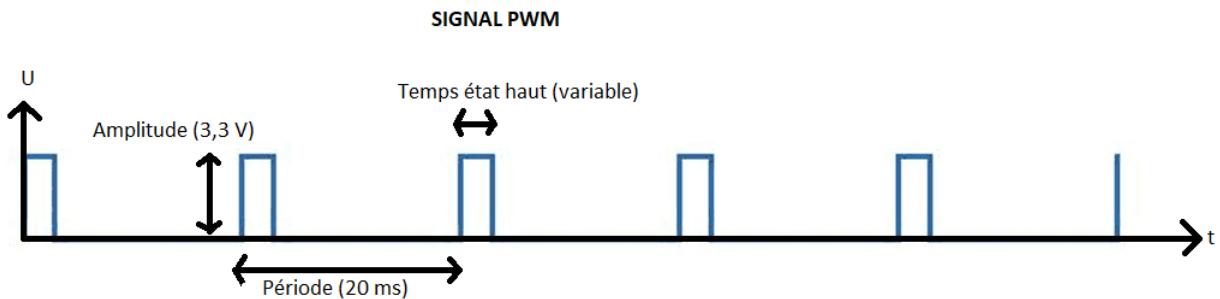


Les trois ESC utilisés dans un premier temps pour nos moteurs (Référence 2) sont également équipés d'un circuit éliminateur de batterie, ou BEC, permettant de générer un signal d'alimentation constant de 5V et 3A maximum. Ce dernier permet d'alimenter un autre composant, comme une carte Raspberry Pi dans notre cas, sans avoir à recourir à une seconde batterie.

Cependant, ces ESC ne permettent la rotation du moteur que dans un seul sens. Le seul moyen de modifier le sens de rotation du moteur dans ce cas est d'échanger deux des trois signaux déphasés envoyés au moteur. La direction droite/gauche étant assurée par les deux moteurs de propulsion arrière, cette particularité n'est pas problématique : la rotation plus rapide d'un des deux moteurs arrière par rapport à l'autre permet de diriger le ROV à gauche ou à droite. En revanche, le moteur vertical devant assurer la propulsion verticale doit pouvoir tourner dans les deux sens. Un second modèle d'ESC a donc été nécessaire pour permettre au moteur de tourner dans les deux sens. Ce dernier (Référence 3) possède ainsi un mode "reverse" permettant au moteur de tourner dans les deux sens, ainsi qu'un BEC, et sera donc attribué au moteur vertical du ROV.

4.2 Calibration et commande des ESC

Traditionnellement, le signal de contrôle envoyé à l'ESC est un signal PWM de fréquence 50 Hz environ, un certain écart de cette valeur étant accepté. Dans notre cas, le signal est généré par une Raspberry Pi 3, et l'amplitude du signal est donc de 3,3 V environ.



La caractéristique la plus importante de ce signal est la largeur de chaque impulsion. C'est cette dernière, généralement entre 0.5 ms et 2.5 ms, qui commande directement l'amplitude du signal envoyé aux moteurs et donc leur vitesse de rotation. Notre carte doit donc être capable de générer un signal crête-à-crête à 50 Hz, et de rapport cyclique variable sur commande. Pour générer un tel signal, on utilise le programme ServoBlaster, disponible sur Github. En effet ce dernier a été conçu pour piloter des servomoteurs à partir des pins GPIO de la Raspberry Pi, et est donc le plus adapté pour générer un tel signal. Pour installer ServoBlaster sur la carte, on rentre les commandes suivantes sur le terminal de Raspbian :

```
1 sudo apt-get install git
2 git clone https://github.com/richardghirst/PiBits.git
3 cd PiBits/ServoBlaster/user
4 nano init-script
5 # Supprimer la valeur --idle-timeout=2000 des options par défaut
6 nano servod.c
7 # Modifier la ligne 960 pour avoir "else if (strstr(modelstr, "BCM2709") ||
8 strstr(modelstr, "BCM2835"))"
8 make
9 sudo make install
```

Une fois cette installation effectuée, on peut générer le signal à l'aide de la commande, toujours dans le terminal de Raspbian, "echo x=y > /dev/servoblaster", où x représente le GPIO que l'on utilise et y la valeur du temps d'état haut, en dizaine de us (par exemple y=100 correspond à un temps d'état haut de 1 ms). Les valeurs de y peuvent aller de 50 à 250, correspondant respectivement à un temps haut de 0.5 ms et 2.5 ms.

Les valeurs de x désignent les GPIO suivant :

Servo number	GPIO number
0	4
1	17
2	18
3	21/27
4	22
5	23
6	24
7	25

On obtient ainsi des commandes comme ci-dessous :

```
1 | echo 0=200 > /dev/servoblaster
2 | echo 0=100 > /dev/servoblaster
```

Avec chaque nouvelle valeur de y modifiant la valeur du temps haut du signal sortant.

Toutefois, avant leur utilisation, les ESC nécessitent d'être calibrés. En effet, en fonction des radiocommandes ou autre dispositifs de commande utilisés, l'intervalle des valeurs du temps d'état haut du signal de commande envoyé à l'ESC peut différer. L'ESC doit donc être calibré pour que la valeur maximale de temps d'état haut corresponde à la vitesse maximale de rotation du moteur. De même pour une valeur neutre (moteur à l'arrêt), et une valeur minimale correspondant à la vitesse de rotation inverse maximale, disponible uniquement pour l'ESC "Reverse", et donc le moteur vertical associé.

La calibration ne peut pas être effectuée à l'aide des commandes issues de ServoBlaster, l'ESC nécessitant une variation douce du temps d'état haut pour cela. C'est le cas d'un joystick de télécommande, mais pas du programme ne permettant que des variations abruptes de la valeur du temps d'état haut. La calibration des ESC a donc été effectuée à l'aide d'une radiocommande disponible à l'ENSEA (Référence 4). Pour celà, on place le joystick de la manette au point neutre, on branche l'ESC à l'émetteur de la radiocommande et on allume dans l'ordre la radiocommande, puis l'ESC. Lorsque ce dernier s'allume, il associe ainsi automatiquement le signal en train d'être reçu au point neutre, donc moteur immobile. On effectue la même opération avec le joystick au maximum pour calibrer la valeur maximale de temps d'état haut et l'associer à la vitesse maximale de rotation du moteur. Une fois cette calibration effectuée, l'ESC en marche ne fonctionnera qu'après avoir reçu le signal correspondant à l'état neutre pendant un certain temps. Celà permet d'éviter entre autres un démarrage intempestif des moteurs.

Une fois la calibration effectuée, on observe à l'oscilloscope le signal de commande envoyé par la manette pour déterminer le temps d'état haut correspondant au point neutre, à la vitesse maximale, et à la vitesse maximale en rotation inverse pour le moteur vertical.

On obtient les résultats suivants :

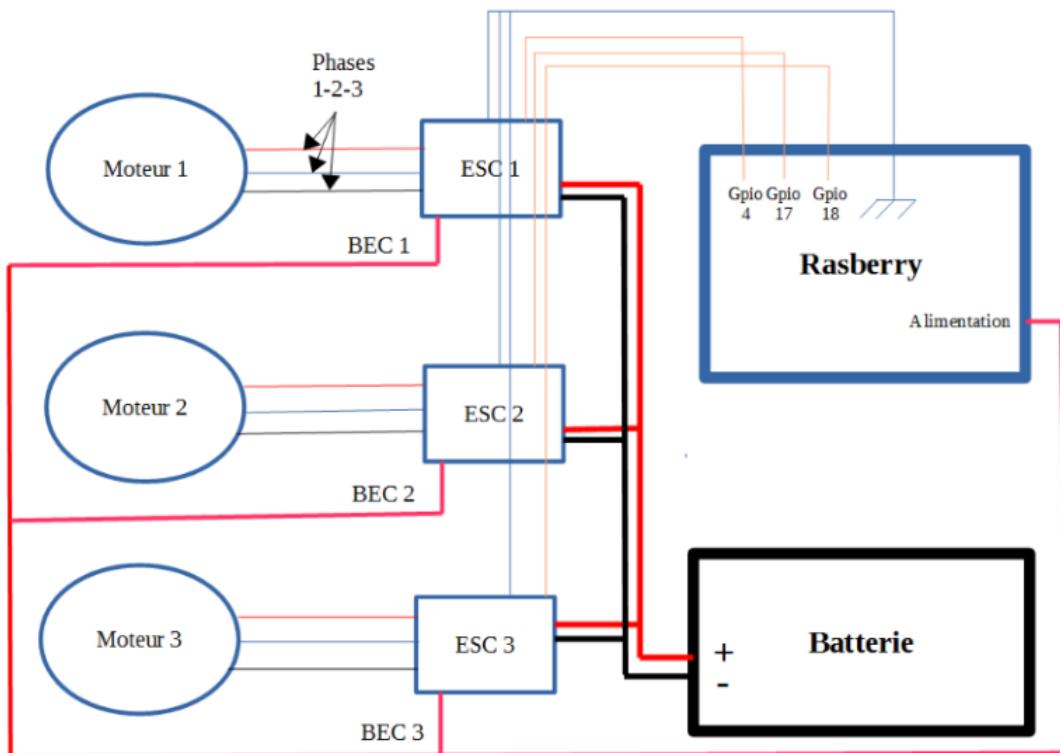
Type d'ESC	Temps état haut	Point neutre (valeur de y)	Vitesse maximale (valeur de y)	Vitesse maximale inverse (valeur de y)
ESC 20A (propulsion)	1,2 ms (120)		1,8 ms (180)	Pas de mode Reverse
ESC 30A (moteur vertical)	1,5 ms (150)		1,9 ms (190)	1,4 ms (140)

Après calibration et relevé de ces valeurs, on peut bien commander les 3 moteurs à l'aide des commandes de ServoBlaster.

4.3 Alimentation et montage

On utilise pour l'alimentation des ESC et des moteurs une batterie NiMh, de capacité 3000 mAh et délivrant une tension de 7,2 V (Référence 5). L'alimentation de la Raspberry est elle assurée par les BEC des ESC : ces derniers, mis en parallèle, délivrent une alimentation constante d'environ 5V et 3A, suffisante à alimenter la carte. On obtient donc le schéma fonctionnel suivant :

Architecture de la motorisation du ROV



Lorsque les 3 moteurs sont à leur vitesse maximale, avec la carte fonctionnelle, le courant prélevé sur la batterie est de 2A maximum. L'autonomie théorique dans cette configuration est donc d'environ 90 minutes. Toutefois cette dernière risque de diminuer avec l'utilisation et donc l'alimentation des différents capteurs, ainsi que la résistance de l'eau en condition.

Chapitre 5

Acquisition et Commandabilité

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

5.1 Raspberry

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

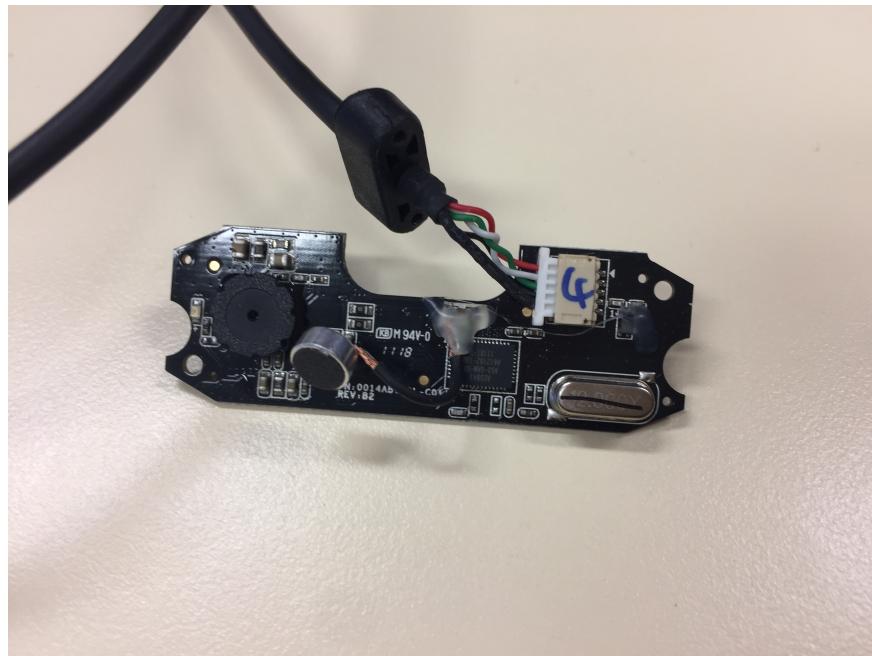


5.1.1 Cameras

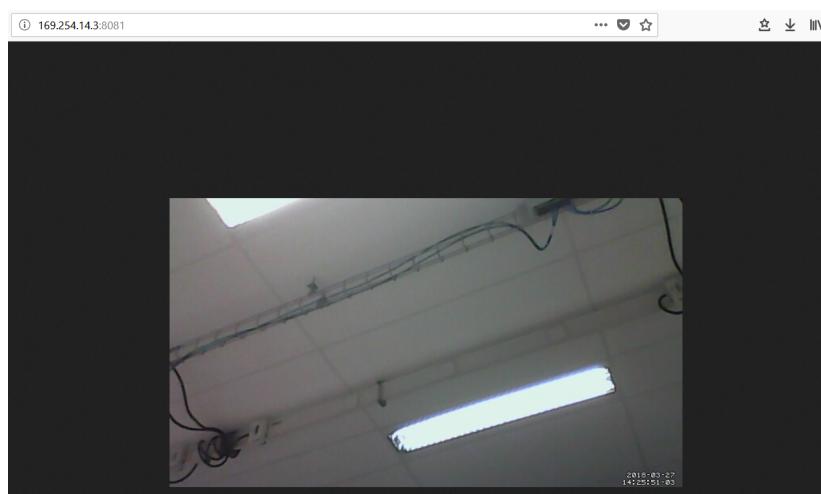
Nous avons 2 caméras qui permettent, l'une la direction (vision frontale) et l'autre la cartographie (vision par dessous). Dans un premier temps, détaillons leur connexion entre la Raspberry et le traitement effectué par celle-ci.

Logitech C170

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



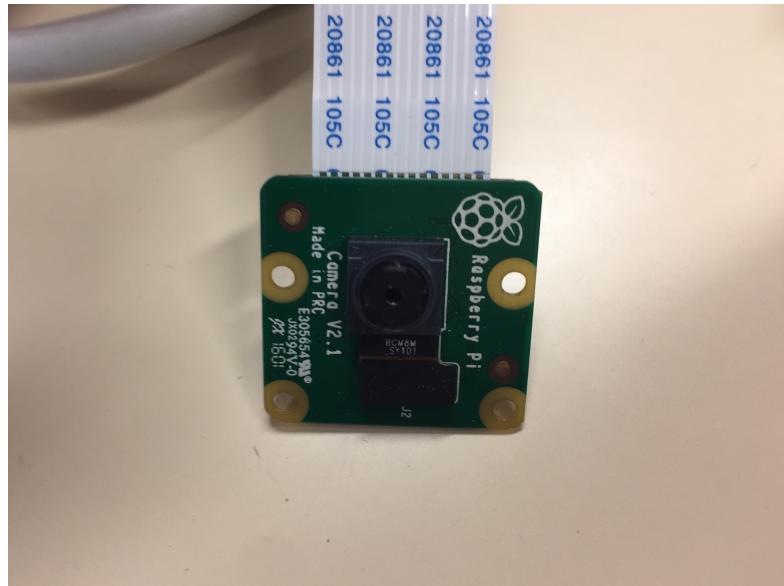
Nous l'avons choisi car le pilote de celle ci est déjà installé nativement sur la Raspberry. Nous utilisons motion qui permet d'envoyer le flux vidéo venant de la caméra et de la diffuser en ligne sur notre adresse local (Référence 6). En voici le résultat sur un navigateur :



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

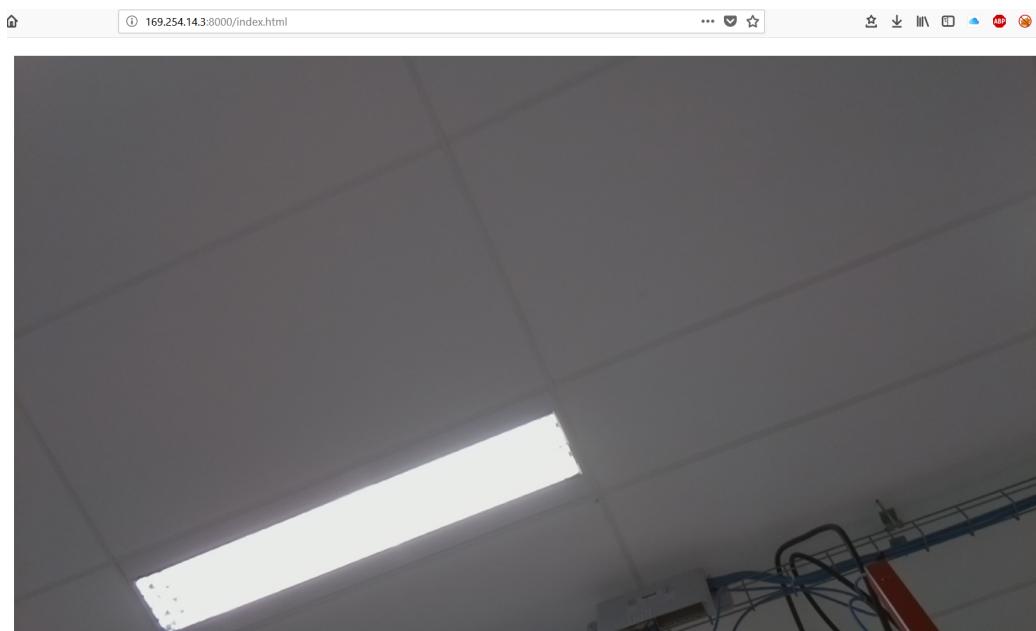
Caméra V2

La deuxième est un module caméra pour raspberry (Référence 7) qui se raccorde directement par une nappe (un bus de type CSI-2).



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 (Référence 8) mais cette fois-ci sur un autre port.

Le résultat sur un navigateur :



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

5.1.2 Capteur de Pression/Temperature

5.1.3 Central Inertielle

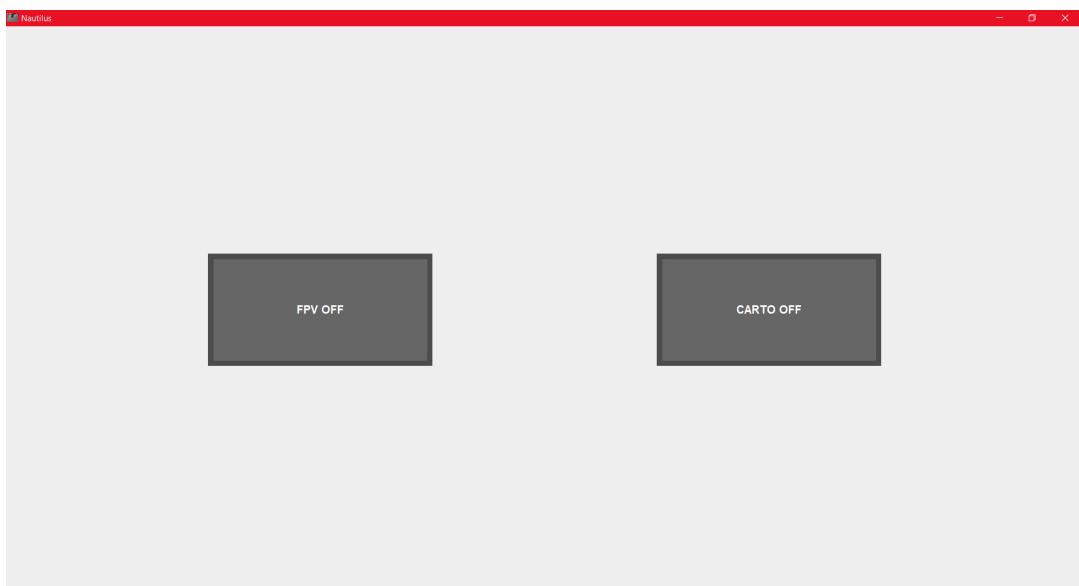
Julien

5.2 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.

5.2.1 Interface

En premier lieu parlons de l'interface, celle ci à pris différentes formes au cours du temps, ici nous presenterons que la dernière version. Toute la partie PC a été programmé en JAVA (Disponible ici : Référence 9), l'interface utilise la bibliothèque graphique Swing qui nous permet de gerer l'affichage facilement que ce soit pour la video ou pour les interactions. Lorsque l'application est lancé, nous arrivons donc dans un premier menu :



Le bouton de Droite mène à une partie que nous développerons dans la partie 5.2.6 liée à la Cartographie.

Nous avons donc créé 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 paramétriser chaque objet. Ce manager crée une grille qui se construit en fonction des paramètres de chaque objet qu'elle 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 choisis ici de garder fixe.

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

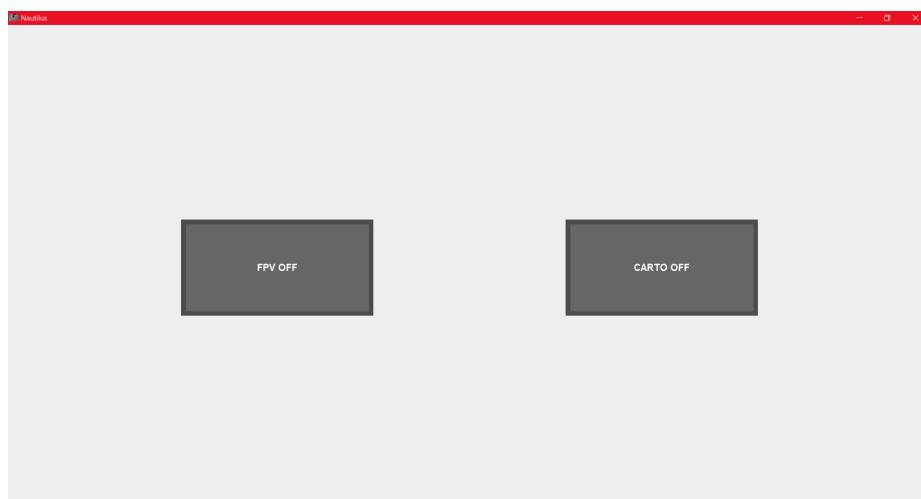
La ligne 8 et 9 définissent des poids en x et y qui sont utilisé lors d'un redimensionnement, cela permet de donner plus de poids à un objet plutot qu'à un autre. Nous ne l'utilisons pas d'où la valeur 0.

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

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

Chaque objet de notre interface est defini de cette façon.

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



Nous avons maintenant la FPV en haut à gauche, le bouton à droite est le même que celui précédemment, les informations des capteurs sont affichées dans le cadre à droite, les commandes envoyées aux moteurs sont en bas et pour finir un affichage 3D (5.2.5) avec JAVA3D du ROV en bas à droite. Toutes les informations étant actualisées en temps réel avec la Raspberry. Nous allons voir comment.

5.2.2 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 connection, 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 instancié 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 lancé.

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

Les lignes 9,10 et 11 definissent où vont les données entrées, sorties et sorties erreur. Dans notre cas l'entrée ne nous interesse pas car cela passe par une commande, la sortie normal sera stockée et la sortie erreur renvoyé sur l'afficheur d'erreur du systeme (la console). La ligne 12 lance la connection 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 stockées pour ensuite pouvoir les traiter.

Puis les dernières lignes, permettent de gerer 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 video IP. Abordons ce sujet.

5.2.3 Videos

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 récupéré puis traité puis affiché en temps réel sans interrompre le reste du programme, tout comme l'envoie de chaque commande.

Pour les flux vidéos, c'est une connection http qui est effectué, 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 connection si pas de connection 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 connection http est effectué.

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

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 appelé en ligne 11.

Observons cette méthode :

```
1 public void initDisplay()
2 {
3     if (useMJPEGStream) readMJPEGStream();
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 vidéos ou juste photo. Dans notre cas c'est un flux vidéos composé d'image JPG.

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

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

```
1 public void readMJPEGStream()
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 connection, les 2 méthodes ci dessus sont appelé 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 geré 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 renvoyé par la centrale intertie :

```
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 calculé par rapport à une rotation avec une origine au centre de l'image.

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

5.2.4 Capteurs et Moteurs

5.2.5 Affichage 3D

5.2.6 Cartographie

Chapitre 6

Références

Motorisation et énergie :

- **Référence 1** : Moteur d'avion électrique brushless ROXXY 315079 chez Conrad (x3)
- **Référence 2** : ESC Suppo Multirotor 20A M20A chez RobotShop (x3)
- **Référence 3** : ESC HobbyKing 30A avec Reverse (x1)
- **Référence 4** : Radiocommande Graupner MX-20 (disponible à l'ENSEA)
- **Référence 5** : Batterie d'accumulateurs (NiMh) 7.2 V 3000 mAh Conrad (x1)

Acquisition et Commandabilité :

- **Référence 6** : Adresse Camera Frontal
- **Référence 7** : Module Camera V2 chez Mouser Electronics
- **Référence 8** : Adresse Camera du Dessous
- **Référence 9** : Code JAVA de l'interface