SI – P.I (Projet Interdisciplinaire)

Projet Aéroglisseur

Enseignants : **M. ENEAU**, **M. PIETRE** et **M.LESAGE**

Personnes composant le groupe (5) : CHALUMEU Pierre, COUCHOUD Thomas, DUTREUIL Raphaël, FROGER Olivier et JACQUES Johann

Dossier de **COUCHOUD Thomas**

Table des matières

I - Présentation du projet

Dans le cadre du Projet Interdisciplinaire, nous avons été amenés à réaliser un projet par nous-même. Pour cela nous avons décider de concevoir un aéroglisseur de taille modélisme qui peut se déplacer facilement dans un environnement respectant des conditions, tout en étant commandé à distance. Il devra également être capable de donner une vision de son environnement proche à l'utilisateur.

Notre projet se trouve ainsi enveloppé dans plusieurs disciplines :

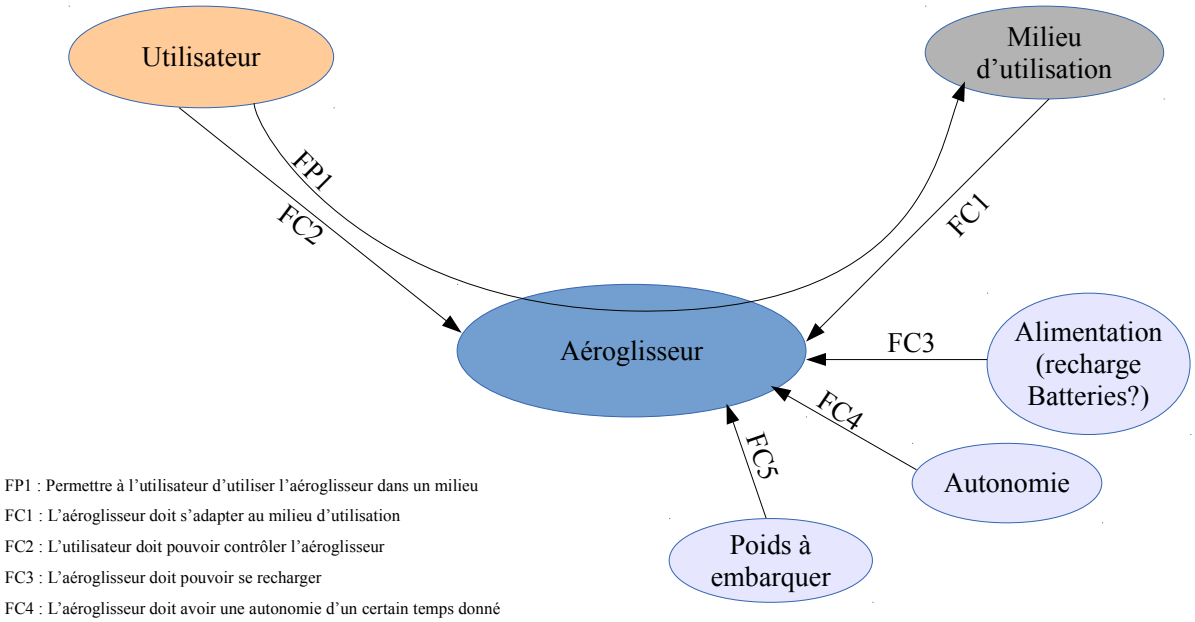
- Une partie mécanique
- Une partie électronique
- Une partie physique

II - Répartitions des tâches

Étant donné que nous sommes cinq dans notre groupe, nous avons décidés (selon nos envies et compétences) ensemble de la répartition des tâches pour en arriver à celle-ci:

- CHALUMEAU Pierre → Designer l'aéroglisseur en intégrant les solutions et en répartissant les masses
- **COUCHOUD Thomas** et JACQUES Johann → Rendre l'aéroglisseur autonome et contrôlable à distance
- DUTREUIL Raphaël → Réaliser la sustentation (élever et stabiliser l'aéroglisseur)
- FROGER Olivier → Trouver un moyen permettant de diriger et propulser l'aéroglisseur

III - Cahier des charges du projet



FP1 : Permettre à l'utilisateur d'utiliser l'aéroglisseur dans un milieu
FC1 : L'aéroglisseur doit s'adapter au milieu d'utilisation
FC2 : L'utilisateur doit pouvoir contrôler l'aéroglisseur
FC3 : L'aéroglisseur doit pouvoir se recharger
FC4 : L'aéroglisseur doit avoir une autonomie d'un certain temps donné
FC5 : L'aéroglisseur doit pouvoir embarquer une masse donné

Énonce général du besoin :

- Nécessité de pouvoir inspecter des lieux avec des sols de natures différentes
- Capacité de se déplacer et visionner son espace frontal

Contraintes imposées au projet :

- L’aéroglisseur est un drone (pas de transport de personnes)
- Limites de coût

Limites du projet :

- Utilisation en intérieur sur sol lisse
- La partie vision ne fait pas partie du projet

IV - Présentation globale des différentes parties

A - Design de l’aéroglisseur

Le design de l'aéroglisseur peut être compris comme étant le corps de l'aéroglisseur. Les autres parties qui sont indispensables au fonctionnement de l'aéroglisseur, viendront se greffer sur cette structure. La difficulté de cette partie repose sur le fait d’imaginer une silhouette aérodynamique, solide, permettant d’intégrer les autres composants. Une fois la partie disons « imaginaire » réalisée, il a fallut procéder à une étude de matériaux pour savoir lequel serait le plus approprié. Ensuite vint la conception d'un premier prototype qui nous permit d'identifier certains problèmes. La création d'un deuxième prototype, en corrigeant les ennuis rencontrés précédemment, nous a conduit à de nouveaux problèmes. Ceux-ci seront finalement corrigés en améliorant et finalisant le premier prototype.

B - Sustentation

La sustentation est le principe même de l'aéroglisseur. Sustenter consiste à surélever de quelque millimètres du sol l’aéroglisseur de manière à pouvoir annuler sa masse et ainsi pouvoir se mouvoir sans aucun frottement au niveau du sol. Pour pouvoir sustenter il nous a fallu créer une jupe. Celle-ci permet de mettre l’air sous pression avant de le faire sortir sous l’aéroglisseur. De la même manière que pour la structure on a du procéder à une étude des matériaux ainsi qu'à des rajustements lors de la réalisation des deux prototypes.

C - Direction et propulsion

Cette partie a pour but de permettre à l'aéroglisseur de se déplacer dans une plusieurs directions. Il à été nécessaire d'imaginer un système permettant de propulser l'aéroglisseur. Dans un premier temps, il fut décidé de positionner un moteur à l'arrière de la structure celui ci serait surélevé et pourrait pivoter de manière à ce que le flux d’air soit orienté. Ce système fut abandonné faute de solidité, les sollicitations au niveau du système permettant au moteur de pivoter était trop importantes. Le deuxième système, celui actuellement présent sur le projet consiste à surélever un moteur au dessus du bâti, envoyer l'air à travers des volets, ceux-ci s'occupant de diriger l'air induite par la propulsion.

D - Contrôle du système

Cette partie repose sur le fait de récupérer les commandes de l’utilisateur et les appliquer au système. Dans un premier temps il a fallut déterminer par quel biais nous allions récupérer les commandes de l'utilisateur, en l’occurrence ce fut un gamepad pour PC. Dans un second temps, il fut nécessaire de créer un programme récupérant les commandes de l’utilisateur. Dans un tiers temps nous avons du créer un système de transmission et de réception des commandes saisies par l'opérateur de manière à ce que l'aéroglisseur puisse les exécuter (ici on a un système wi-fi, émission et transmission des commandes par un ordinateur portable sur un serveur créer par une

Arduino Yùn embarquée à bord de l'aéroglisseur).

E - Autonomie du système

Cette partie repose sur demandes :

- Gérer les différents composants de l'aéroglisseur en fonctions des commandes de l'utilisateur. Ce système est géré grâce à un microcontrôleur dans notre cas une Arduino Yùn.
- Gérer l'autonomie et les dépenses énergétique de manière à répartir au mieux celles-ci pour pouvoir accroître l'autonomie tout en restant fonctionnel.
- Gérer le sous système de visualisation de l'environnement, sa portée et son autonomie ainsi que la réception des données numériques.

V - Présentation détaillée de la partie « autonomie »

La partie autonomie (réalisée par *COUCHOUD Thomas* et JACQUES Johann) a plusieurs objectifs :

- Permettre le contrôle à distance de l'aéroglisseur
  - Récupération des commandes de l'utilisateur
  - Traitement de ces informations
  - Action en conséquence sur les actionneurs
- Assurer une certaine autonomie au système
- Procurer une vision de l'environnement à l'utilisateur

A - Contrôle à distance de l'aéroglisseur

Du fait que l'aéroglisseur soit en mouvement, la contrainte de contrôle à distance nous est vite venue à l'esprit. En effet il ne nous sera pas possible de communiquer avec notre micro contrôleur par USB car la distance entre l'utilisateur et l'aéroglisseur est variable et peut devenir plus grande qu'un simple câble. Des raisons de confort et d'esthétisme nous ont poussé dans l'idée de réaliser une communication sans fil. Nous avons finalement opté pour une communication WI-FI entre un PC (utilisateur) et une carte Arduino Yùn (aéroglisseur).

1 - Récupération des commandes de l'utilisateur

Première étape pour pouvoir réaliser cette communication est de récupérer les demandes de l'utilisateur (tourner à droite, ralentir...). Pour cela nous avons décidé d'utiliser une contrôleur de jeu USB pour plusieurs raisons :

- Facilité de prise en main
- Nombreuses touches
- Connexion simple par USB

Une fois la manette connectée à l'ordinateur, il faut aller « lire »

les valeurs des différents boutons pour pouvoir connaître les intentions de l'utilisateur. C'est pourquoi j'ai réalisé un programme en JAVA qui sera chargé de cette tâche.

Celui-ci se trouve ici.

Celui-ci fonctionne grâce à une boucle qui va vérifier constamment les états des différents boutons et les comparer aux valeurs précédentes. Si un des états a changé on appelle une fonction qui va gérer ce changement.

Prenons par exemple le cas du bouton A (relié à l'activation de la sustentation).

```
for(int i = 0; i < buttonsPressed.size(); i++)
    if(buttonsPressed.get(controller.getButtonName(i)) != controller.isButtonPressed(i))
    {
        if(controller.isButtonPressed(i))
            onButtonPressed(controller.getButtonName(i));
        else
            onButtonReleased(controller.getButtonName(i));
        buttonsPressed.put(controller.getButtonName(i), controller.isButtonPressed(i));
    }
```

Ligne 1 → On commence une boucle qui va vérifier pour les boutons du type « bouton pressoir »

Ligne 2 → Pour chacun d'entre eux on vérifie si son état a été modifié depuis la dernière vérification

Ligne 4 à 7 → Si l 'état du bouton a été changé, on appelle la bonne fonction en conséquence de l'action :

- onButtonPressed si le bouton a été pressé
- onButtonReleased si le bouton a été lâché

Ligne 8 → On garde en mémoire l'état actuel des boutons pour la prochaine vérification

```
private void onButtonPressed(final String name)
{
    Main.logger.log(Level.INFO, "Button " + name + " pressed");
    if(name.equals(ACT_SUSTENT))
        Interface.changeValue("st", -9999);
}
```

Pour notre exemple nous avons appuyé sur le bouton « A » et nous retrouvons ainsi dans cette fonction.

Celle-ci va gérer l'événement « un bouton a été appuyé » (marche de façon similaire pour les autres événements). Pour cela on procède à une simple vérification du nom du bouton, si le bouton appuyé est bien le bouton A, on change la valeur de « st » (sustentation) à -9999 (qui sera interprété comme en changement d'état par le programme : si la valeur était « allumé » elle sera maintenant à « éteint »).

2 - Envoi des données à l'Arduino

Maintenant que nous avons traduit les demandes de l'utilisateur en données informatiques, il est nécessaire de les communiquer à l'aéroglisseur. Pour cela, une communication WI-FI est utilisée et gérée par la partie « Sender » du programme (Annexe).

Celle-ci marche de façon similaire à la réception des commandes de l'utilisateur : une boucle vérifie constamment si un changement dans les valeurs ont étés effectuées. Si c'est le cas, on envoi cette valeur à l'Arduino.

L'envoi se traduit par l'accès une page internet de l'Arduino :

`http://<ADRESSE_IP_DE_L'ARDUINO>/<CLEF>/<VALEUR>`

L'adresse IP de l'Arduino est donnée au programme au démarrage. Les deux autres valeurs sont déterminées en fonction du paramètre qui change.

Dans notre exemple nous avions modifié la sustentation, dans ce cas la clef sera « st ». Si dans celui-ci nous aurions effectué l'opération « éteindre la sustentation », la page à laquelle se connectera le programme sera :

```
http://<ADRESSE_IP_DE_L'ARDUINO>/st/0
```

3 - Réception des données sur l'Arduino

La partie opérant sur l'ordinateur de l'utilisateur est maintenant terminée et la suite géré par la carte Arduino. Celle-ci va devoir « réceptionner » les connexions qui sont effectuées par le PC. Notre carte Arduino Yùn est en réalité équipée d'une puce permettant une connexion en WIFI mais surtout qui permet de créer un réseau où l'ordinateur va se connecter. Par la suite on lance un objet Server sur l'Arduino qui va nous permettre de réceptionner les requêtes.

```
void loop()
{
  YunClient client = server.accept();
  if (client)
  {
    String command = client.readString();
    command.trim();
    printMessage("Command recevied : " + command);
    String result = decrypt(command);
    printMessage("Result : " + result);
    client.print(result);
    client.stop();
  }
}
```

Pour cela, une fois que le serveur est lancé, on vérifie constamment si une requête nous est parvenue et si tell est le cas, on la décrypte.

4 - Décryptage des requêtes

Cette partie est relativement simple, il suffit d'identifier les paramètres de l'URL à laquelle le PC est connecté.

```
http://<ADRESSE_IP_DE_L'ARDUINO>/<CLEF>/<VALEUR>
```

L'objet serveur va nous indiquer que le PC s'est connecté à « <CLEF>/<VALEUR> ». Il nous reste plus qu'à récupérer ce qu'il y a avant le '/' pour la clef d'identification du paramètre et ce qui est après ce caractère pour la valeur.

5 - Action sur les actionneurs

Maintenant que nous avons récupéré la clef et la valeur de la requête du coté de l'aéroglisseur, il nous reste à les communiquer aux actionneurs. Pour cela la clef va nous permettre de choisir l'actionneur concerné (moteur de sustentation pour la clef « st », moteur de direction pour « di »...) .

Si l'on continue notre exemple d'extinction de la sustentation, nous allons nous retrouver dans ce cas :

```
else if(key == sustentation_key)
{
    if(value == 1)
    {

        motorSustentation.write(RPMToServoSpeed(rpm_sustent));
        sustentation_value = 1;
        return "Switching sustentation ON";
    }
    else
    {

        sustentation_value = value;
        motorSustentation.write(RPMToServoSpeed(0));
        return "Switching sustentation OFF";
    }
}
```

Deux possibilités s'offrent à nous. Soi on veut allumer la sustentation (la valeur est égale à 1) ou l'on ne veut pas l'allumer (la valeur est tout sauf 1).

Si l'on éteint, on dit au moteur concerné de s'arrêter sinon on lui dit de tourner à la vitesse nécessaire pour effectuer la sustentation.

6 - Expérimentations sur cette partie

Afin d'avoir un système fonctionnel, plusieurs tests on étés effectués.

Au niveau de la liaison WIFI, des tests de qualité du signal en fonction de la distance on été effectués. Par manque de temps, ce test a été fait rapidement avec des intervalles de distance élevés. Après réflexion on peu penser que le graphique que l'on aurai obtenu ressemblerai à une droite ou l'on pourrai remarquer que plus la distance entre les deux périphériques sont élevés, plus la qualité du signal diminue. De plus cette qualité dépend énormément de l'environnement (certains des matériaux absorbent plus les ondes que d'autres, problèmes de réflexion du signal, interférences...).

Avec cette distance des tests sur les réponses entre l'ordinateur et l'Arduino ont aussi étés effectués, et comme la qualité du signal, plus la distance est élevée plus le temps de réponse se dégrade en augmentant.

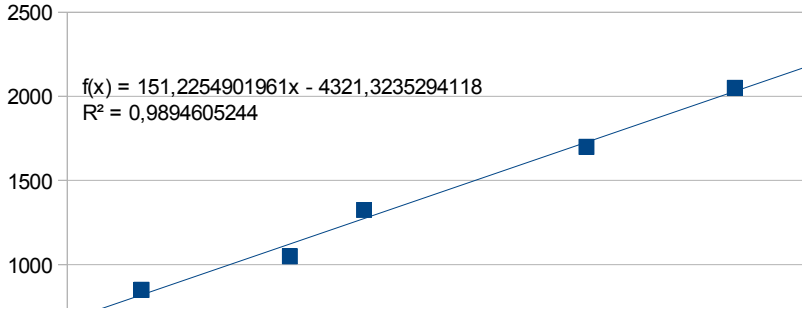
Les autre expérimentation effectué concernent les actionneurs. En effet il a fallu déterminer quelles valeurs informatiques correspondaient à quelles grandeur physique (vitesse, angle).

Ceux-ci ont étés réalisés de la manière suivante :

- Mesure de la vitesse (ou orientation) pour différentes valeurs données.
- Création d'un graphique des valeurs mesurées en fonction des valeurs envoyées
- Interprétation de ce graphique (souvent une droite)
- Création d'une fonction permettant le passage de la valeur reçue à la valeur « réelle »

Exemple avec le moteur de propulsion :

Ici cela nous permet de convertir des valeurs envoyées par le PC en tr/min (de 0 à 9500) en valeurs comprises par le moteur (de 90 à 180).




```
int RPMToServoSpeed(unsigned int rpm)
{
    if(rpm < 0)
        return 0;
    else if(rpm == 0)
        return 90;
    return (119 + (int)(rpm / 151.22));
}
```

B - Assurer une autonomie au système

Maintenant que nous avons un système capable de gérer les déplacements de l'aéroglisseur à distance. Il nous faut comme défini dans nos contraintes pouvoir le faire se déplacer pendant une certaine durée.

Étant donné que l'aéroglisseur n'est jamais situé au même endroit, il nous est nécessaire d'utiliser des batteries. Dans notre cas il y en a deux embarquées, une qui alimente l'ensemble des composants nécessaire au déplacement de l'aéroglisseur et une autre ayant une capacité inférieur s'occupant de l'alimentation de la caméra et de son émetteur.

Dans un premier temps on parlera de la batterie principale, celle qui alimente l'aéroglisseur en lui même. Pour avoir une idée de son autonomie, il nous a fallu procéder à des test et beaucoup d'analyse théorique tentant à modéliser la consommation de l'aéroglisseur.

Nous avons du commencer par vérifier que les batteries choisies étaient bien capables d'alimenter nos moteurs, ce qui est bien le cas.

Puis nous avions voulu effectuer des tests sur les deux prototype pour nous faire une idée de l'autonomie mais suite à plusieurs problèmes due à l'état d'avancement de l'aéroglisseur qui n'était pas totalement finalisé nous n'avons pas pu procéder à de réels tests autonomiques concluants.

Nous nous sommes donc rabattu sur un modèle théorique d'autonomie. Dans une première phase nous avons estimé l'autonomie en cas d'utilisation maximal pour cela nous nous somme servis de cette formule :

$$I = \frac{Q}{T} \left(I\ en\ A, Q\ en\ Ah, T\ en\ h \right) \quad d'où\ T = \frac{Q}{I}$$

Lors de l'élaboration de ce premier modèle autonomique nous avons pris en compte uniquement les moteurs car le reste du système est négligeable vis à vis des moteurs. Avec notre batterie ayant une capacité de 1300mAh, et nos moteurs qui ont une consommation de 20Ah chacun, nous en avons déduis qu'en cas de consommation maximal c'est à dire que le couple des deux moteurs serait maximum nous aurions une autonomie d'environ 1,95 minute.

Nous avons vite constaté que ce modèle était opérationnel mais quasiment inutile pour se faire une idée de l'autonomie réelle. Ce qui nous poussa a créer le deuxième modèle mettant en jeu la puissance développée par le moteur de sustentation et la puissance fourni par le moteur de propulsion mesurée grâce à un vélocimètre. Malheureusement par manque de temps ce modèle n'as pas pu aboutir.

Néanmoins de par les test pratiques nous avons pu déterminer par estimation qu'avec le système final nous avions une autonomie comprise entre un quart d'heure et vingt minute.

Dans un second temps nous allons brièvement parler de la batterie qui alimente la caméra. Celle-ci faisait parti du pack avec lequel nous avons récupéré la caméra. Nous avons pu constater que l'autonomie ne dépassait que rarement les cinq minutes. Nous supposons que cette batterie a fait son temps car ce module optique appartient au lycée depuis quelques année, elle a donc été pas mal utilisé. On peut aussi constater que l'autonomie actuelle ne correspond plus à celle décrite par le constructeur.

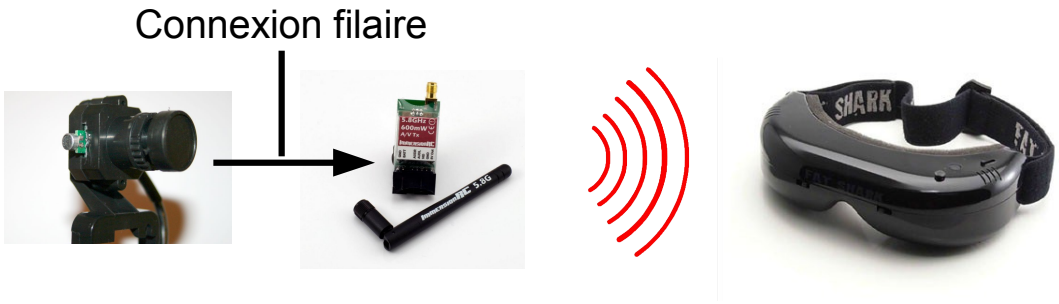
C - Procurer une vision de l'environnement à l'utilisateur

Dans la réalisation du projet, nous avions pensés à intégrer un système permettant de voir l'avant de l'aéroglisseur. Ainsi l'utilisateur n'aurait pas à se déplacer pour regarder constamment où est son engin et procure ainsi une utilisation plus simple.

Pour cela nous avons simplement implémenté un système tout fait du nom de FatShark. Celui-ci nous procure plusieurs éléments :

- Une mini caméra
- Un boîtier émetteur
- Une paire de lunettes faisant office de boîtier récepteur

Le principe est simple, la caméra enregistre ce qui est devant elle, puis transmet ses données au boîtier émetteur. Celui-ci va transmettre ses données par le biais d'ondes à une fréquence de 5,8GHz qui seront réceptionnées par les lunettes. Ces dernières affichent ensuit l'image sur des écrans.



Afin d'assurer un meilleur confort, la caméra est située sur deux servo moteurs permettant de l'orienter de façon horizontale et verticale.

Un test pour connaître la distance maximale de réception de l'image a été effectué. On a pu constater que l'image reste nette un bon moment puis se dégrade très vite pour finir avec un résultat sans image. Le procédé était simple, après s'être mis dans un couloir, une personne avait les lunettes et servait « d'indicateur de réception » et une deuxième personne avançait progressivement dans ce couloir en s'éloignant des lunettes avec l'émetteur et la caméra. Les résultats furent les suivants :

- Dégradation de l'image notable : ~40m
- Réception de l'image non effectuée : ~45m

VI- Conclusion

Pour conclure sur notre projet, nous pensons avoir réalisé une bonne partie des contraintes que nous nous étions imposées. Même si toutes les parties ne sont pas encore totalement assemblées sur la « version finale », nous pensons qu'avec un peu plus de temps le projet aurai pu être fini. A ce stade, le système de pilotage à distance est fonctionnel, avec une autonomie

raisonnable qui est capable de visionner son environnement.

Les points positifs de notre projet sont :

- Un aéroglisseur capable de visionner son environnement.
- L' innovation du système, notre aéroglisseur peut sustenter de manière stationnaire (contrairement aux aéroglisseurs commerciaux).

D'un point de vue critique nous estimons que nous aurions pu pour optimiser le temps de réalisation de différentes manières:

- Se baser sur des structure préexistante ainsi que des systèmes de communications et d'actions plus simple à gérer.
- Tenter d'optimiser le choix les matériaux (un choix d'optimisation a déjà été réalisé concernant ce point mais pas poussé à bout pour des raisons financières).

Ce projet fut très intéressant car il nous a permis un cohésion entre les membres de l'équipe, et nous apporté de nouvelles connaissances. Ce procédé est aussi enrichissant car il nous permet de concrétiser les savoir qui nous sont enseignés.

VII – Annexes

GamepadHandler.java

```
package fr.vaucanson;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import org.lwjgl.LWJGLException;
import org.lwjgl.input.Controller;
import org.lwjgl.input.Controllers;

public class GamepadHandler extends Thread
{
	private String ACT_SUSTENT, VALUE_ORIENTATION, VALUE_ORIENTATION_POV =
"POVX", VALUE_SPEED, VALUE_SPEED_POV = "POVY", VALUE_CAM_OR_VERT,
VALUE_CAM_OR_HOR;
	private final int CONF_ABSOLUTE = 0, CONF_RELATIVE = 1;
	private final int conf = CONF_RELATIVE;
	private Controller controller;
	private Map<String, Boolean> buttonsPressed;
	private Map<String, Float> axisStatus, povStatus;

	/**
	 * Constructor
	 */
	public GamepadHandler()
	{
		if(System.getProperty("os.name").toLowerCase().contains("win"))
		{
			ACT_SUSTENT = "Bouton 0";
			VALUE_CAM_OR_VERT = "Rotation Y";
			VALUE_CAM_OR_HOR = "Rotation X";
			VALUE_SPEED = "Axe Y";
```

```
                VALUE_ORIENTATION = "Axe X";
            }
            else
            {
                ACT_SUSTENT = "A";
                VALUE_CAM_OR_VERT = "ry";
                VALUE_CAM_OR_HOR = "rx";
                VALUE_SPEED = "y";
                VALUE_ORIENTATION = "x";
            }
        }

        /**
         * Used to initalize the controller
         *
         * @return Nothing
         * @throws Exception
         */
        private void init() throws Exception
        {
            Main.logger.log(Level.INFO, "Setting Gamepad Controller...");
            try
            {
                Controllers.create();
            }
            catch(LWJGLException e)
            {
                e.printStackTrace();
            }
            Controllers.poll();
            for(int i = 0; i < Controllers.getControllerCount(); i++)
            {
                controller = Controllers.getController(i);
                if(controller.getName().contains("Gamepad F310") ||
controller.getName().contains("Generic X-Box pad"))
                    break;
            }
            if(controller.equals(null))
            {
                Main.logger.log(Level.SEVERE, "No gamepad found!");
                throw new Exception();
            }
            Main.logger.log(Level.FINE, "Controller: " + controller.getName() + "\tAxis: " +
controller.getAxisCount() + "\tButtons: " + controller.getButtonCount());
            Main.logger.log(Level.FINE, "Waiting for joysticks to be set to the middle...");
            while(controller.getAxisValue(0) != 0 || controller.getAxisValue(1) != 0 ||
controller.getAxisValue(2) != 0 || controller.getAxisValue(3) != 0)
                controller.poll();
            buttonsPressed = new HashMap<String, Boolean>();
            for(int i = 0; i < controller.getButtonCount(); i++)
                buttonsPressed.put(controller.getButtonName(i),
controller.isButtonPressed(i));
            axisStatus = new HashMap<String, Float>();
            for(int i = 0; i < controller.getAxisCount(); i++)
                axisStatus.put(controller.getAxisName(i), controller.getAxisValue(i));
            povStatus = new HashMap<String, Float>();
            povStatus.put("POVX", controller.getPovX());
            povStatus.put("POVY", controller.getPovX());
        }

        /**
```

```
	 * Checking the inputs of the controller
	 */
	@Override
	public void run()
	{
		try
		{
			init();
			Thread.sleep(450);
		}
		catch(Exception e)
		{
			return;
		}
		while(!Thread.interrupted())
		{
			try
			{
				Thread.sleep(50);
			}
			catch(InterruptedException e)
			{
				Main.logger.log(Level.WARNING, "Error when sleeping for the
controller", e);
			}
			controller.poll();
			for(int i = 0; i < buttonsPressed.size(); i++)
				if(buttonsPressed.get(controller.getButtonName(i)) !=
controller.isButtonPressed(i))
				{
					if(controller.isButtonPressed(i))
						onButtonPressed(controller.getButtonName(i));
					else
						onButtonReleased(controller.getButtonName(i));
					buttonsPressed.put(controller.getButtonName(i),
controller.isButtonPressed(i));
				}
			for(int i = 0; i < axisStatus.size(); i++)
				if(axisStatus.get(controller.getAxisName(i)) !=
controller.getAxisValue(i))
				{
					onAxisValueChange(controller.getAxisName(i),
controller.getAxisValue(i), true);
					axisStatus.put(controller.getAxisName(i),
controller.getAxisValue(i));
				}
				else
					onAxisValueChange(controller.getAxisName(i),
controller.getAxisValue(i), false);
			if(controller.getPovX() != povStatus.get("POVX"))
			{
				onPovValueChange("POVX", controller.getPovX());
				povStatus.put("POVX", controller.getPovX());
			}
			if(controller.getPovX() != povStatus.get("POVX"))
			{
				onPovValueChange("POVY", controller.getPovY());
				povStatus.put("POVY", controller.getPovX());
			}
		}
	}
```

```
/**
 * Called when a button is pressed
 *
 * @param name The button's name
 * @return Nothing
 */
private void onButtonPressed(final String name)
{
	Main.logger.log(Level.INFO, "Button " + name + " pressed");
	if(name.equals(ACT_SUSTENT))
		Interface.changeValue("st", -9999);
}

/**
 * Called when a button is released
 *
 * @param name The button's name
 * @return Nothing
 */
private void onButtonReleased(final String name)
{
	Main.logger.log(Level.INFO, "Button " + name + " released");
}

/**
 * Called when an axis value changed
 *
 * @param name The axis' name
 * @param value The new value
 * @return Nothing
 */
@SuppressWarnings("all")
private void onAxisValueChange(final String name, final float value, boolean newValue)
{
	if(newValue)
		Main.logger.log(Level.INFO, "Axis " + name + " modified to " + value);
	if(conf == CONF_ABSOLUTE)
	{
		if(name.equals(VALUE_ORIENTATION))
			Interface.setValue("or", value);
		else if(name.equals(VALUE_CAM_OR_VERT))
			Interface.setValue("cv", value);
		else if(name.equals(VALUE_CAM_OR_HOR))
			Interface.setValue("ch", value);
		else if(name.equals(VALUE_SPEED))
			Interface.setValue("vi", -value);
	}
	else if(conf == CONF_RELATIVE)
	{
		if(name.equals(VALUE_SPEED))
			Interface.changeValue("vi", (int) (-250 * value));
		else if(name.equals(VALUE_ORIENTATION))
			Interface.changeValue("or", (int) (5 * value));
		else if(name.equals(VALUE_CAM_OR_VERT))
			Interface.changeValue("cv", (int) (-5 * value));
		else if(name.equals(VALUE_CAM_OR_HOR))
			Interface.changeValue("ch", (int) (5 * value));
	}
}
```

```
	/**
	 * Called when a POV axis value changed
	 *
	 * @param name The POV axis name
	 * @param value The new value
	 * @return Nothing
	 */
	private void onPovValueChange(final String name, final float value)
	{
		Main.logger.log(Level.INFO, "POV " + name + " modified to " + value);
		if(name.equals(VALUE_ORIENTATION_POV))
		{
			Interface.changeValue("or", 10 * (int) value);
		}
		else if(name.equals(VALUE_SPEED_POV))
		{
			Interface.changeValue("vi", 100 * (int) value);
		}
	}
}
```

Sender.java

```
package fr.vaucanson;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.InetAddress;
import java.net.URL;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.logging.Level;

public class Sender extends Thread
{
  public static String IP_PING;
  private static String IP_URL;
  private static HashMap<String, Integer> requestsSended;
  private static final String[] keys = {"or", "st", "vi", "cv", "ch"};
  private static ArrayList<Long> times;

  /**
   * Used to initialize the connection with the Arduino Yun
   *
   * @return True if the connection is set
   * @throws Exception If there were an error with the initialization
   */
  private static boolean init() throws Exception
  {
		Main.logger.log(Level.INFO, "Initializing sender...");
		final InetAddress inet = InetAddress.getByName(IP_PING);
		if(inet.isReachable(5000))
			return true;
		Main.logger.log(Level.WARNING, "Proxy is needed, configuring it");
		System.setProperty("http.proxyHost", "proxy");
```

```
		System.setProperty("http.proxyPort", "8080");
		if(inet.isReachable(5000))
			return true;
		throw new Exception("Can not connect to the Arduino");
	}

	/**
	 * Used to send a GET command to the Arduino
	 *
	 * @param params The GET params to send
	 * @return The response by the server
	 * @throws Exception If there were an error with the connection
	 */
	synchronized private String sendGet(String key, String value) throws Exception
	{
		Date startDate = new Date();
		URL url = new URL(IP_URL + "arduino/" + key + "/" + value);
		HttpURLConnection connection = (HttpURLConnection) url.openConnection();
		connection.setRequestMethod("GET");
		connection.setRequestProperty("User-Agent", "Mozilla/5.0");
		int responseCode = connection.getResponseCode();
		Main.logger.log(Level.INFO, "\nSending 'GET' request to URL : " + url);
		Main.logger.log(Level.INFO, "Response Code : " + responseCode);
		BufferedReader inputSream = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
		String inputLine;
		StringBuffer response = new StringBuffer();
		while((inputLine = inputSream.readLine()) != null)
			response.append(inputLine);
		inputSream.close();
		times.add((new Date().getTime() - startDate.getTime()));
		long average = 0;
		for(long l : times)
			average += l;
		Main.logger.log(Level.INFO, "Done in " + times.get(times.size() - 1) + " ms\tAverage: "
+ (average / times.size()));
		return response.toString();
	}

	/**
	 * Constructor
	 *
	 * @param IP1 The IP for the ping test
	 * @param IP2 The URL for the Arduino internet control page
	 * @throws Exception If there were an error with the connection
	 */
	public Sender(String IP1, String IP2) throws Exception
	{
		Sender.IP_PING = IP1;
		Sender.IP_URL = IP2;
		Main.logger.log(Level.INFO, "Creating sender...");
		setName("Sender");
		times = new ArrayList<Long>();
		requestsSended = new HashMap<String, Integer>();
		requestsSended.put("or", 50);
		requestsSended.put("vi", 0);
		requestsSended.put("st", 0);
		requestsSended.put("ch", 50);
		requestsSended.put("cv", 50);
		init();
		Main.logger.log(Level.INFO, "Sender initialized on " + IP_PING + " sending requests
```

```
        to " + IP_URL + "!");
          }

          /**
           * Check every 50ms if there is an information to send and send it
           */
          @Override
          public void run()
          {
                    while(!Thread.interrupted())
                    {
                              for(final String key : keys)
                                        if(Interface.getRequests().containsKey(key) &&
        (Interface.getRequests().get(key) != requestsSended.get(key)))
                                        {
                                                  int value = 0;
                                                  try
                                                  {
                                                            value = Interface.getRequests().get(key);
                                                            sendGet(key, String.valueOf(value));
                                                  }
                                                  catch(Exception e)
                                                  {
                                                            e.printStackTrace();
                                                            continue;
                                                  }
                                                  requestsSended.put(key, value);
                                        }
                    }
          }
        }
```

Main.java

```
package fr.vaucanson;

import java.awt.Frame;
import java.io.File;
import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main
{
          public static Sender sender;
          public static Logger logger;
          public static GamepadHandler gamepad;
          private final static String IP = "192.168.240.1";

          public static void main(String[] args) throws SecurityException, IOException
          {
                    final FileHandler fileTxt = new FileHandler(new File(".", "log.txt").getAbsolutePath(),
true);
                    fileTxt.setFormatter(new LogFormatter());
                    fileTxt.setEncoding("ISO-8859-1");
                    logger = Logger.getLogger("Aeroglisseur");
                    logger.setLevel(Level.FINER);
                    logger.addHandler(fileTxt);
                    logger.log(Level.FINEST, "\n\n---------- Starting program ----------\n");
                    new Interface();
```

```
                try
                {
                        sender = new Sender(IP, "http://" + IP + "/");
                        sender.start();
                }
                catch(Exception e)
                {
                        logger.log(Level.SEVERE, "Can't reach " + e.getMessage() + "\tClosing
program!");
                        for(Frame f : Interface.getFrames())
                                f.dispose();
                }
                gamepad = new GamepadHandler();
                gamepad.run();
        }
}
```

Interface.java

```
package fr.vaucanson;

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.logging.Level;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class Interface extends JFrame
{
        private static final long serialVersionUID = 7231194594050358481L;
        public JFrame mainFrame;
        private static GridBagLayout mainLayout;
        private static JPanel contentPanel, camPanel, controlPanel;
        private static Hashtable<String, Object> frameObjects;
        private static HashMap<String, Integer> requests;
        private static int frameWidth, frameHeight;

        /**
         * Constructor
```

```
	 */
	public Interface()
	{
		frameWidth = 800;
		frameHeight = 200;
		requests = new HashMap<String, Integer>();
		frameObjects = new Hashtable<String, Object>();
		Hashtable<Integer, JLabel> labelTableOrientation = new Hashtable<Integer,
JLabel>();
		labelTableOrientation.put(0, new JLabel("Gauche"));
		labelTableOrientation.put(50, new JLabel("Devant"));
		labelTableOrientation.put(100, new JLabel("Droite"));
		Hashtable<Integer, JLabel> labelTableCamVert = new Hashtable<Integer, JLabel>();
		labelTableCamVert.put(0, new JLabel("Bas"));
		labelTableCamVert.put(70, new JLabel("Devant"));
		labelTableCamVert.put(100, new JLabel("Haut"));
		Hashtable<Integer, JLabel> labelTableSustentation = new Hashtable<Integer,
JLabel>();
		labelTableSustentation.put(0, new JLabel("OFF"));
		labelTableSustentation.put(1, new JLabel("ON"));
		Hashtable<Integer, JLabel> labelTableSpeed = new Hashtable<Integer, JLabel>();
		labelTableSpeed.put(0, new JLabel("0"));
		labelTableSpeed.put(4612, new JLabel("4612"));
		labelTableSpeed.put(9225, new JLabel("9225"));
		/*************************************************************************************************
**/
		contentPanel = new JPanel();
		contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.X_AXIS));
		camPanel = new JPanel();
		camPanel.setLayout(new BoxLayout(camPanel, BoxLayout.Y_AXIS));
		controlPanel = new JPanel();
		controlPanel.setLayout(new BoxLayout(controlPanel, BoxLayout.Y_AXIS));
		mainLayout = new GridBagLayout();
		Border loweredetched =
BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
		TitledBorder titleCam = BorderFactory.createTitledBorder(loweredetched,
"Cam\351ra");
		titleCam.setTitleJustification(TitledBorder.CENTER);
		camPanel.setBorder(titleCam);
		TitledBorder titleControl = BorderFactory.createTitledBorder(loweredetched,
"Contr\364les");
		titleControl.setTitleJustification(TitledBorder.CENTER);
		controlPanel.setBorder(titleControl);
		addSlider(0, 1, 1, 0, "st", labelTableSustentation, controlPanel);
		addSlider(0, 9225, 250, 0, "vi", labelTableSpeed, controlPanel);
		addSlider(0, 100, 10, 50, "or", labelTableOrientation, controlPanel);
		addSlider(0, 100, 10, 70, "cv", labelTableCamVert, camPanel);
		addSlider(0, 100, 10, 50, "ch", labelTableOrientation, camPanel);
		/*************************************************************************************************
**/
		mainFrame = new JFrame("Controles de l'aeroglisseur");
		mainFrame.setMinimumSize(new Dimension(850, frameHeight));
		mainFrame.addComponentListener(new ComponentListener()
		{
			@Override
			public void componentHidden(ComponentEvent arg0)
			{}

			@Override
			public void componentMoved(ComponentEvent arg0)
			{}
```

```
		@Override
		public void componentResized(ComponentEvent arg0)
		{
			if(arg0.getComponent() instanceof JFrame)
			{
				frameWidth = ((JFrame) arg0.getComponent()).getWidth();
				frameHeight = ((JFrame) arg0.getComponent()).getHeight();
			}
			else if(arg0.getComponent() instanceof JPanel)
			{
				frameWidth = ((JPanel) arg0.getComponent()).getWidth();
				frameHeight = ((JPanel) arg0.getComponent()).getHeight();
			}
			resize();
		}

		@Override
		public void componentShown(ComponentEvent arg0)
		{
			if(arg0.getComponent() instanceof JFrame)
			{
				frameWidth = ((JFrame) arg0.getComponent()).getWidth();
				frameHeight = ((JFrame) arg0.getComponent()).getHeight();
			}
			else if(arg0.getComponent() instanceof JPanel)
			{
				frameWidth = ((JPanel) arg0.getComponent()).getWidth();
				frameHeight = ((JPanel) arg0.getComponent()).getHeight();
			}
			resize();
		}
});
mainFrame.setLayout(mainLayout);
mainFrame.setPreferredSize(new Dimension(frameWidth, 450));
mainFrame.setResizable(true);
mainFrame.setAlwaysOnTop(true);
mainFrame.setVisible(true);
// mainFrame.setResizable(false);
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
mainFrame.setLocationRelativeTo(null);
/************ CONTROLS *************************/
GridBagConstraints constraints = new GridBagConstraints();
constraints.anchor = GridBagConstraints.PAGE_START;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.gridy = 0;
constraints.gridx = 0;
constraints.weightx = 0.5;
contentPanel.add(controlPanel, constraints);
constraints.gridx = 1;
constraints.insets = new Insets(0, 0, 10, 0);
contentPanel.add(camPanel, constraints);
/********** BUILDING FRMAE **********************/
constraints = new GridBagConstraints();
constraints.anchor = GridBagConstraints.NORTH;
constraints.fill = GridBagConstraints.BOTH;
constraints.gridy = 0;
constraints.gridx = 0;
constraints.weightx = 1;
constraints.weighty = 1;
constraints.ipady = frameHeight;
```

```
                constraints.ipadx = frameWidth / 2;
                mainFrame.add(contentPanel, constraints);
                /************************************************/
                mainFrame.pack();
        }

        private void resize()
        {
                GridBagConstraints constraints = new GridBagConstraints();
                constraints.anchor = GridBagConstraints.NORTH;
                constraints.fill = GridBagConstraints.BOTH;
                constraints.gridy = 0;
                constraints.gridx = 0;
                constraints.weightx = 1;
                constraints.weighty = 1;
                constraints.ipady = frameHeight;
                constraints.ipadx = frameWidth / 2;
                mainLayout.setConstraints(contentPanel, constraints);
                mainFrame.revalidate();
                mainFrame.repaint();
        }

        /**
         * Used to add a slider with his label
         *
         * @param min The minimum value of the slider
         * @param max The maximum value of the slider
         * @param majT The major tick of the slider
         * @param bas The default value of the slider
         * @param key The key for the slider/label
         * @param labels The table label for the slider
         * @param cont The container to put the slider with his label
         * @return Nothing
         */
        private void addSlider(int min, int max, int majT, int base, final String key, Hashtable<Integer,
JLabel> labels, Container cont)
        {
                JPanel tempPanel = new JPanel(new GridBagLayout());
                final JTextField tempLabel = new JTextField();
                JSlider tempSlider = new JSlider();
                tempPanel.setPreferredSize(new Dimension(570, 66));
                tempPanel.setFocusable(false);
                tempSlider.setValue(min);
                tempSlider.setMaximum(max);
                tempSlider.setToolTipText("S" + key);
                tempSlider.setBounds(0, 0, 484, 66);
                tempSlider.setMajorTickSpacing(majT);
                tempSlider.setValue(base);
                tempSlider.setPaintTicks(true);
                if(labels != null)
                {
                        tempSlider.setLabelTable(labels);
                        tempSlider.setPaintLabels(true);
                }
                tempSlider.setFocusable(false);
                tempSlider.addChangeListener(new ChangeListener()
                {
                        @Override
                        public void stateChanged(final ChangeEvent arg0)
                        {
                                changeValue(key, 0);
```

```
                }
        });
        tempLabel.setEditable(false);
        tempLabel.setOpaque(false);
        tempLabel.setBorder(null);
        tempLabel.setMinimumSize(new Dimension(10, 10));
        tempLabel.setPreferredSize(new Dimension(10, 10));
        tempLabel.setMaximumSize(new Dimension(10, 10));
        tempLabel.setBackground(Color.GRAY);
        tempLabel.setText(String.valueOf(tempSlider.getValue()));
        tempLabel.setHorizontalAlignment(JLabel.CENTER);
        tempLabel.setFocusable(false);
        tempLabel.setToolTipText("T" + key);
        tempLabel.addPropertyChangeListener("text", new PropertyChangeListener()
        {
                @Override
                public void propertyChange(PropertyChangeEvent arg0)
                {}
        });
        GridBagConstraints constraints = new GridBagConstraints();
        constraints.anchor = GridBagConstraints.CENTER;
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.gridy = 0;
        constraints.gridx = 0;
        constraints.weightx = 0.85;
        tempPanel.add(tempSlider, constraints);
        constraints.gridx = 1;
        constraints.weightx = 0.15;
        tempPanel.add(tempLabel, constraints);
        cont.add(tempPanel);
        frameObjects.put("T" + key, tempLabel);
        frameObjects.put("S" + key, tempSlider);
}

/**
 * Used to set a value to an ID
 *
 * @param key The value ID
 * @param value The value to set
 * @return Nothing
 */
public static void setValue(String key, float value)
{
        JSlider slider = ((JSlider) frameObjects.get("S" + key));
        JTextField lab = ((JTextField) frameObjects.get("T" + key));
        int nValue;
        if(key.equals("cv"))
        {
                if(value >= 0)
                        nValue = (int) (70 + (value * 30f));
                else
                        nValue = (int) (70 + (value * 70f));
        }
        else if(key.equals("vi"))
                nValue = (int) (value * slider.getMaximum());
        else
                nValue = (int) ((slider.getMaximum() / 2) + (value * (slider.getMaximum() / 2)));
        slider.setValue(nValue);
        addToSend(key, slider.getValue());
        lab.setText(String.valueOf(slider.getValue()));
        frameObjects.put("S" + key, slider);
```

```
                frameObjects.put("T" + key, lab);
        }

        /**
         * Used to change the value to an ID
         *
         * @param key The value ID
         * @param value The value to add to the curent value
         * @return Nothing
         */
        public static void changeValue(String key, int value)
        {
                JSlider slider = ((JSlider) frameObjects.get("S" + key));
                JTextField lab = ((JTextField) frameObjects.get("T" + key));
                slider.setValue(value != -9999 ? slider.getValue() + value : (slider.getValue() == 0 ? 1 :
0));
                addToSend(key, slider.getValue());
                lab.setText(String.valueOf(slider.getValue()));
                frameObjects.put("S" + key, slider);
                frameObjects.put("T" + key, lab);
        }

        /**
         * Used to add to the stack a value to send
         *
         * @param key The value ID
         * @param value The value
         * @return Nothing
         */
        synchronized public static void addToSend(final String key, final int value)
        {
                Main.logger.log(Level.FINEST, "Changing " + key + " to " + value);
                requests.put(key, value);
        }

        /**
         * @return the requests
         */
        public static HashMap<String, Integer> getRequests()
        {
                return requests;
        }
}
```

Moteur.ino

```
#include <Servo.h>
#include <TinkerKit.h>
#include <FileIO.h>
#include <Bridge.h>
#include <Bridge.h>
#include <YunServer.h>
#include <YunClient.h>

const int rpm_sustent = 100;
const String speed_key = "vi";
const String orientation_key = "or";
const String sustentation_key = "st";
const String horizontal_cam_key = "ch";
const String vertical_cam_key = "cv";
```

```
const String values_file = "variables.txt";

YunServer server;
Servo motorSustentation;
Servo motorDirection;
Servo servoMotor;
Servo servoVertCam;
Servo servoHorCam;
TKPotentiometer potentiometer(I0);
int speed_value;
int orientation_value;
int sustentation_value;
int vetical_cam_value;
int horizontal_cam_value;

void printMessage(String message)
{
   Serial.println(String(millis()/1000) + " -> " + message);
}

int oriToDegrees(unsigned int ori)
{
  return (15 + (int)(ori/70.0));
}

int RPMToServoSpeed(unsigned int rpm)
{
   if(rpm < 0)
      return 0;
   else if(rpm == 0)
      return 90;
   return (119 + (int)(rpm / 151.22));
}

String writeToMotors(String key, int value)
{
   if(value < 0 || value > 10000 ||
   (key == orientation_key && value == orientation_value) ||
   (key == speed_key && value == speed_value) ||
   (key == sustentation_key && value == sustentation_value) ||
   (key == horizontal_cam_key && value == horizontal_cam_value) ||
   (key == vertical_cam_key && value == vetical_cam_value))
      return "Error value not in range";
   printMessage("Receiving key " + key + " with value " + value);
   if(key == speed_key)
   {

      motorDirection.write(RPMToServoSpeed(value));
      speed_value = value;
      return "Setting speed to " + String(value) + " which is " + String(RPMToServoSpeed(value));
   }
   else if(key == orientation_key)
   {

      orientation_value = value;
      servoMotor.write(oriToDegrees(value));
      return "Setting orientation to " + String(value);
   }
   else if(key == sustentation_key)
   {
      if(value == 1)
```

```
      {

          motorSustentation.write(RPMToServoSpeed(rpm_sustent));
          sustentation_value = 1;
          return "Switching sustentation ON";
      }
      else
      {

          sustentation_value = value;
          motorSustentation.write(RPMToServoSpeed(0));
          return "Switching sustentation OFF";
      }
    }
    else if(key == horizontal_cam_key)
   {

       horizontal_cam_value = value;
       servoHorCam.write(oriToDegrees(value));
      return "Setting orientation CamHorizontal to " + String(value);
   }
    else if(key == vertical_cam_key)
   {

       vetical_cam_value = value;
       servoVertCam.write(map(100 - value, 0, 100, 20, 70));
       return "Setting orientation CamVertival to " + String(value);
   }
   return "Error, key not recognized! (" + key + ")";
}

String decrypt(String input)
{
       if (input.indexOf('/') < 0)
          return "Error in path";
       return writeToMotors(input.substring(0, input.indexOf('/')), input.substring(input.indexOf('/') +
1).toInt());
}

void initAero()
{
   printMessage("Copying default values to txt file...");
   speed_value = -1;
   sustentation_value = 0;
   orientation_value = 50;
   vetical_cam_value = 50;
   horizontal_cam_value = 70;
   writeToMotors(speed_key, speed_value);
   writeToMotors(sustentation_key, sustentation_value);
   writeToMotors(orientation_key, orientation_value);
   writeToMotors(vertical_cam_key, vetical_cam_value);
   writeToMotors(horizontal_cam_key, horizontal_cam_value);
}

void setup()
{
   Bridge.begin();
   Serial.begin(9600);
   printMessage("Starting arduino!");
   pinMode(13, OUTPUT);
   digitalWrite(13, LOW);
```

```
    digitalWrite(13, HIGH);
    server.listenOnLocalhost();
    server.begin();
    motorDirection.attach(10); //Moteur 3 (Blanc) -> O0
    motorSustentation.attach(11); //Moteur 4 (Marron) -> O1
    servoMotor.attach(9); //O2
    servoVertCam.attach(6); //O3
    servoHorCam.attach(5); //O4
    initAero();
    printMessage("Done!");
}

void loop()
{
  YunClient client = server.accept();
  if (client)
  {
    String command = client.readString();
    command.trim();
    printMessage("Command recevied : " + command);
    String result = decrypt(command);
    printMessage("Result : " + result);
    client.print(result);
    client.stop();
  }
}
```