

INFORMATIQUE ET RÉSEAUX



CINÉMATIQUE D'UN SIMULATEUR HEXAPODE

Killian LOPES - Flavian LAXENAIRE - Yohann RAIMBAULT – Aurélien FERREIRA



Session 2021-2022

Sommaire :

Partie commune	5
Remerciements	5
Présentation du projet	6
Présentation globale	6
Hexapod	1

Cahier des charges	7
Répartition	7
Analyse	8
Ressources	8
Contraintes	10
Spécifications	11
Diagramme UML/SYML	11
Diagramme de cas d'utilisation	11
Diagramme de déploiement	12
Diagramme de classes	13
Robotique asservissement	14
Fonctionnement d'un hélicoptère	14
Comment ça fonctionne ?	14
Le collectif	15
Le cyclique	15
Le palonnier	16
Terminologie imposée	16
Partie personnelle	17
Etudiant 1 : Killian LOPES	17
Mon cahier des charges	17
Protocole de dialogue UDP	17
Une petite définition	17
Caractéristiques de l'UDP	19
Les avantages de l'UDP	19
les désavantages	19
Les DATAREF et leur utilité	20
Qu'est ce qu'un DATAREF ?	20
Comment les utiliser ?	22
implémentation en C++	22
Choix de la période d'échantillonnage	23
Interface Homme Machine	24
FlightSimMockup	26
Conclusion	27
Etudiant 2 : Flavian LAXENAIRE	28
Mon cahier des charges	28
Le protocole Modbus TCP / IP	28
Hexapod	2

Qu'est ce que le protocole Modbus TCP/IP ?	28
Comment fonctionne-t-il ?	30
Pourquoi utiliser ce protocole ?	31
Le fichier CSV	32
Qu'est ce qu'un fichier CSV ?	32
À quoi servent les fichiers CSV ?	32
Utilisation de ce fichier dans notre projet	33
QamModbus	34
Diagramme UML/SYML	34
Diagramme de classes	34
Diagramme de séquence	35
Descriptions des classes	36
FlightSimMotionControl	37
Interface Homme Machine	38
Architecture	38
Problématique	39
Résolution	39
Serveur Modbus	41
MGI	42
FlightSimMockup	45
Caractéristiques	45
Débogage	47
Problématique	47
Résolution	47
Conclusion	48
Etudiant 3 : Yohann RAIMBAULT	49
Mon cahier des charges	49
Matlab/Simulink	50
Qu'est ce que Matlab?	50
Qu'est ce que Simulink ?	50
Comment fonctionne-t-il ?	50
Programme Simulink	50
Gantt	53
Conclusion	53
Etudiant 4 : Aurélien FERREIRA	54
Cahier des charges	54
Hexapod	3

Mes différentes tâches	55
Exemple d'un simple cube	56
La partie du siège	57
La partie de la console	58
La partie des commandes de vol	59
Texturage photo	61
Conclusion	61

Partie commune

I. Remerciements

L'ensemble des étudiants membres du projet Hexapode tient à remercier le Service Départementale d'Incendie et de Secours (SDIS) ainsi que l'Union Départementale des Sapeurs Pompiers (UDSP) de nous avoir confié un tel projet.

Nous souhaitons également remercier M.Menu, M.Facchin, M.Maylaender et M.Wozniak, nos professeurs responsables, qui nous ont accompagnés et aidés tout au long de notre projet.

II. Présentation du projet

A. Présentation globale

Le SDIS 77 développe depuis quelques années (2003) une politique d'utilisation de moyens héliportés dans le cadre de ses missions péri-opérationnelles et opérationnelles. Ces missions aussi variées que la reconnaissance aérienne, la recherche de personnes, la projection de spécialistes (GRIMP, plongeurs, équipes cynophiles...) ou le transport sanitaire héliporté (TSH) impliquent l'ensemble des personnels opérationnels du SDIS 77.

Les entraînements avec les hélicoptères du SDIS 77 sont devenus difficiles, car ils sont utilisés de plus en plus pour les missions de secours. C'est pour cela que le SDIS 77 nous a demandé de réaliser un de leur projet qui est de créer un simulateur sur la base d'une cellule d'hélicoptère (Alouette III). Ce simulateur leur permettra de répondre aux exigences des formations, de la disponibilité et de la variété des scénarios des missions. Cela leur permettra, la formation à la navigation destinée aux officiers, à la sensibilisation aux règles de sécurité aux abords des aéronefs impliquant chaque sapeur-pompier. De nos jours, la simulation est un outil incontournable en matière de formation.

Le lycée La Fayette, dans le cadre des projets de BTS SN-IR, s'est chargé ces dernières années de réaliser un simulateur d'intervention à partir d'une cellule réelle d'hélicoptère Alouette III instrumentée et couplée au progiciel X-Plane.

Dans un simulateur, le pilote est isolé du monde extérieur et perçoit uniquement les informations que le simulateur lui communique. Pour donner la sensation au pilote qu'il est dans un hélicoptère réel, le simulateur doit agir sur 3 aspects :

- Donner une vue de l'extérieur qui est le résultat de la simulation de vol. Ce résultat est visualisé sur un écran permettant de représenter en temps-réel ce que l'on voit au travers des fenêtres du cockpit. Les images sont fournies par X-Plane et projetées au moyen de 3 vidéoprojecteurs.
- Le progiciel produit aussi les effets sonores. La sonorisation reproduit exactement les bruits perçus dans le cockpit (bruit de moteur, bruit de l'air...). L'équipage est en conditions quasi-réelles.
- Faire subir au pilote les effets de l'accélération et les mouvements de l'hélicoptère. Cet aspect est perçu par le système vestibulaire humain (cavité de l'oreille interne). Le simulateur (le siège) devra donc être capable de s'orienter en fonction de la configuration de l'hélicoptère sur X-Plane.

Actuellement, la vue et l'ouïe sont déjà pris en charge, il nous reste donc à réaliser la mise en mouvement de l'hélicoptère.

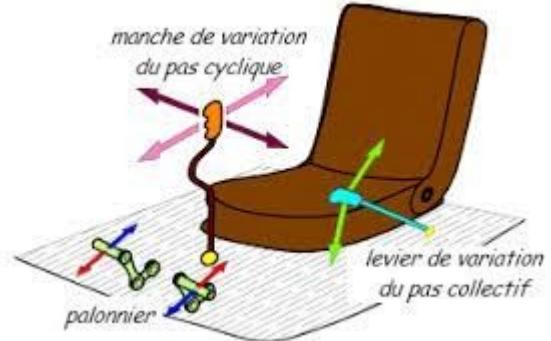
B. Cahier des charges

Le projet dans sa globalité consiste à étudier et valider une solution de mise en mouvements de la maquette, conformément aux informations d'assiette fournies en temps-réel par le progiciel X-Plane.

La cinématique des simulateurs les plus complets est basée sur un hexapode (ou plate-forme de Stewart) doté de 6 axes.

Ces simulateurs permettent de reproduire :

- les mouvements angulaires de roulis, tangage et lacet ;
- les déplacements longitudinaux, transversaux et verticaux.



(une solution moins coûteuse consiste à n'installer que 3 axes, de façon à ne reproduire que les mouvements principaux de l'avion : roulis, tangage et déplacement vertical).

Répartition

Le groupe de ce projet est constitué de quatre étudiants.

Deux étudiant créent et travaillent sur la même application, "*FlightSimMotionControl*" :

- Killian LOPES : Assurer la récupération des informations de la cinématique de l'aéronef via une liaison UDP.
- Flavian LAXENAIRE : Mettre en place un serveur Modbus/IP et implémenter le MGI (Modèle Géométrique Inverse)

Ces tâches sont partagées entre les deux étudiants :

- Faire une interface permettant de contrôler l'état de connexion avec X-Plane, ainsi que l'affichage des différentes variables du serveur Modbus.
- Fusionner les deux parties

Un étudiant travaille sur le design de l'application déjà existante, "*FlightSimMockup*" :

- Aurélien FERREIRA : Compléter le modèle 3D de la maquette afin de visualiser les éléments du poste de pilotage (siège, instruments de vol, commandes, pilote...)

Le dernier étudiant crée l'application "*FlightSimMotionSystem*" :

- Yohann RAIMBAULT : Faire un programme MATLAB / SIMULINK pour actionner les 6 vérins. Faire un client Modbus pour demander les longueurs de vérins.

III. Analyse

A. Ressources

Document spécifiques :

Manuel d'instruction SA316B (Alouette III) tomes 1 & 2 ;

Manuel de vol SA316B ;

Getting/Setting Data from/to X-Plane.

Ressources matérielles :

- Nous avons à notre disposition une maquette à l'échelle 1:1 du poste de pilotage de l'hélicoptère qui a été réalisé lors des premières phases de travaux par l'équipe pédagogique de la section SN-IR du lycée La Fayette.
- Tableaux d'instruments virtuels avec masques alvéolés équipés de témoins/actionneurs ;
- Automate AmB « Android Modip Butler » de gestion des E/S ;
- Poste « simulateur de vol X-Plane » équipé de son système de vidéo-projection ;
- Système de sonorisation de la cellule ;
- Vérins électriques avec modules d'interface de commande ;
- Module Arduino pour pilotage des vérins ;
- Poste Linux dédié « LLF-FlightSim » ;
- Poste Linux dédié « LLF-FlightSimMotionControl » ;
- Poste Linux dédié « LLF-FlightSimMockup » ;
- Poste de développement équipé de Matlab/Simulink version R2021b ou supérieure.

Ressources logicielles :

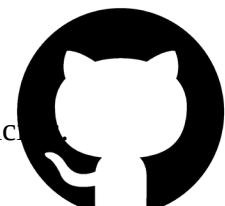
Chaque membre du projet travaillait avec un seul et unique langage : le C/C++.

Que se soit dans le serveur ou client Modbus, la liaison UDP ou bien la modélisation du siège, ce langage nous est indispensable dans ce projet.



Les membres du projet, excepté l'étudiant travaillant sur arduino, utilisent le framework QT. C'est une API orientée objet et développée en C++ qui permet l'utilisation de composants d'interface graphique (widgets).

Cette environnement de développement est utilisé pour concevoir l'architecture de son application à l'aide de signaux et slots.



GitHub est un service web d'hébergement et de gestion de développement de logiciels.

Le nom GitHub est composé du mot « git » faisant référence à un système de contrôle de version open-source et le mot « hub » faisant référence au réseau social.

Il nous a permis de sauvegarder notre code au fil du temps en créant des versions.



X-Plane (*eXperimental-Plane*) est un logiciel de simulation de vol. C'est le logiciel qui nous fournit les données de l'hélicoptère.

Nous avons à notre disposition différentes librairies :

Hexapod



- Les classes de QT : QamSockets et QamModbusMap qui sont utilisées pour le protocole Modbus.
- Les librairies de notre professeur d'informatique, M. Menu : GLam et Qam 3D. Elles sont utilisées pour la modélisation 3D d'objets.

Nous avons aussi accès aux programme déjà existants :

- Sources de l'application *FlightSim*
- Sources de l'application *FlightSimMockup*



Simulink intégré à Matlab est un environnement de schémas bloc utilisé pour concevoir des systèmes, les simuler avant de passer sur du hardware, puis les déployer sans avoir à écrire de code.

Il est utilisé dans notre projet pour piloter les six vérins de l'hexapode.

B. Contraintes

Pour la réalisation du projet, le client nous a imposé certaines contraintes.

Tout d'abord, nous avons une contrainte financière qui est sur les fonds propres de la section SN IR, nous avons aussi quelques contraintes matérielles, comme le poste de simulation X-Plane sous système d'exploitation Windows10, un poste de contrôle-commande sous Linux.

Nous avons aussi des contraintes de fiabilités/sécurité comme prévoir une position de repos (tous les vérins rentrés), et une mise en place de butées logicielles de limitation des mouvements.

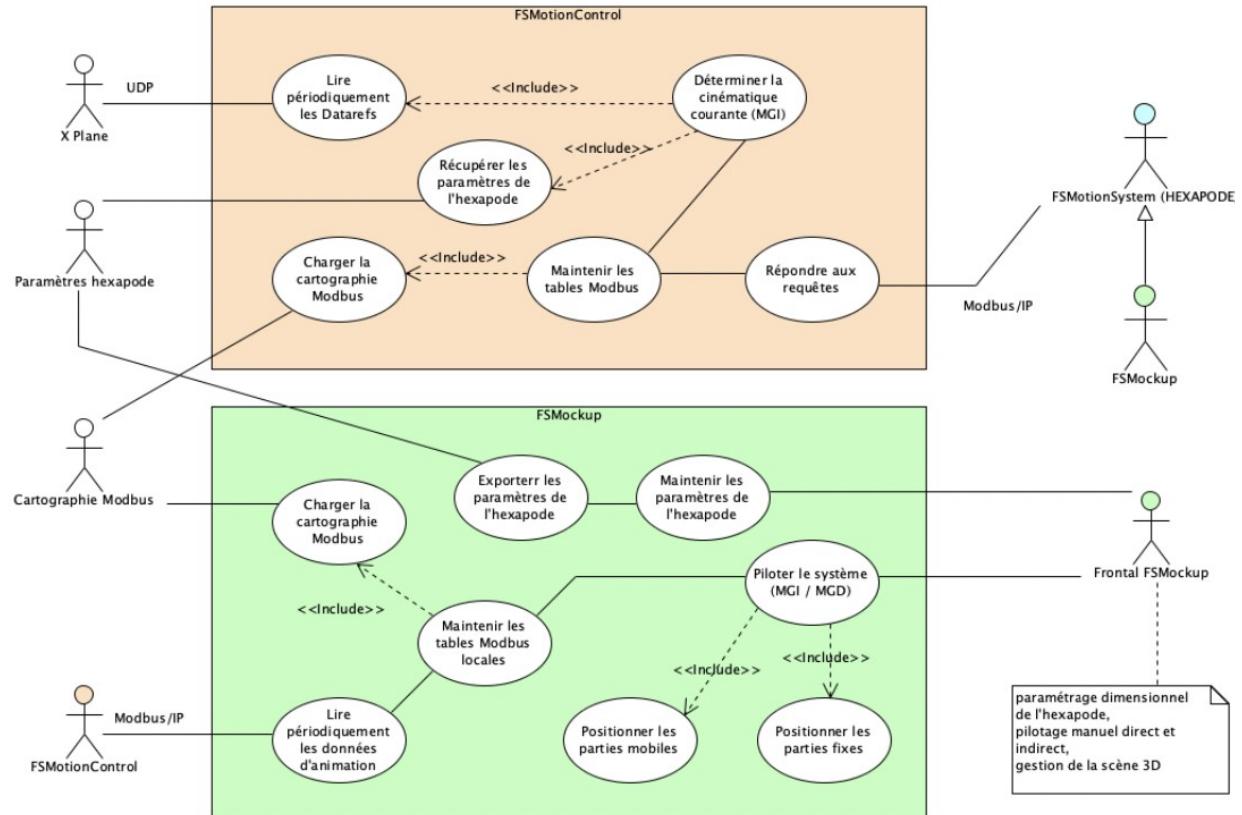
De plus, il y a une contrainte de qualité. Nous devons être prêts à effectuer des tests en situation courant mai / juin. Il faut que les mouvements de la maquette soient opérationnels et réalistes et qu'il y ait une mise à jour en temps réel.

IV. Spécifications

A. Diagramme UML/SYSPML

Diagramme de cas d'utilisation

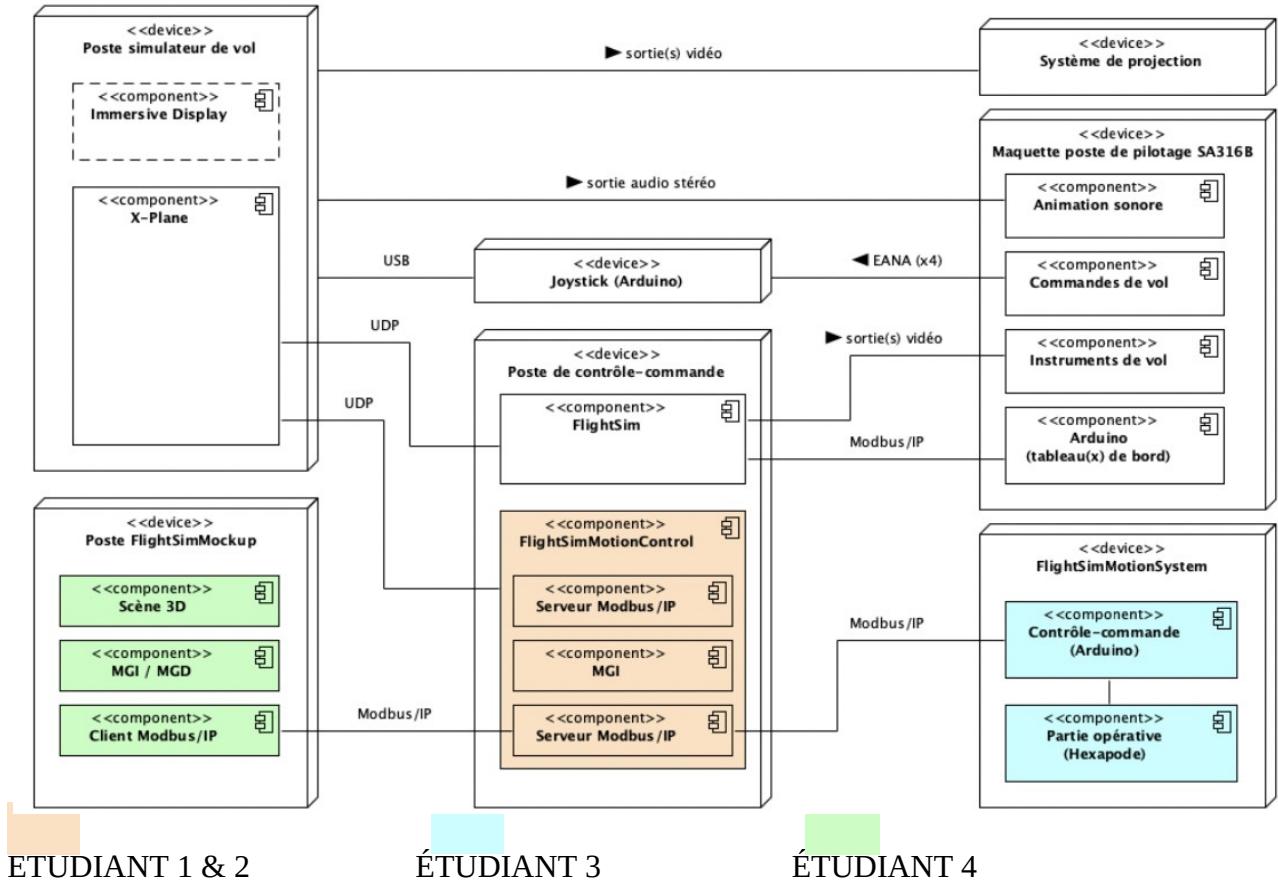
Diagramme de cas d'utilisation : PARTIE MISE EN MOUVEMENT



Le diagramme de cas d'utilisation permet d'identifier les différentes interactions entre les acteurs (personnes extérieures) et le système.

Diagramme de déploiement

Diagramme de déploiement : PARTIE MISE EN MOUVEMENT



Le système est composé de diverses machines reliées entre elles.

Nous avons la maquette du poste de pilotage avec différentes commandes de vol.

- Le « collectif », levier situé à gauche du pilote
- Le cyclique, situé entre les jambes du pilote
- Le palonnier, situé aux pieds du pilote, sous forme de pédales

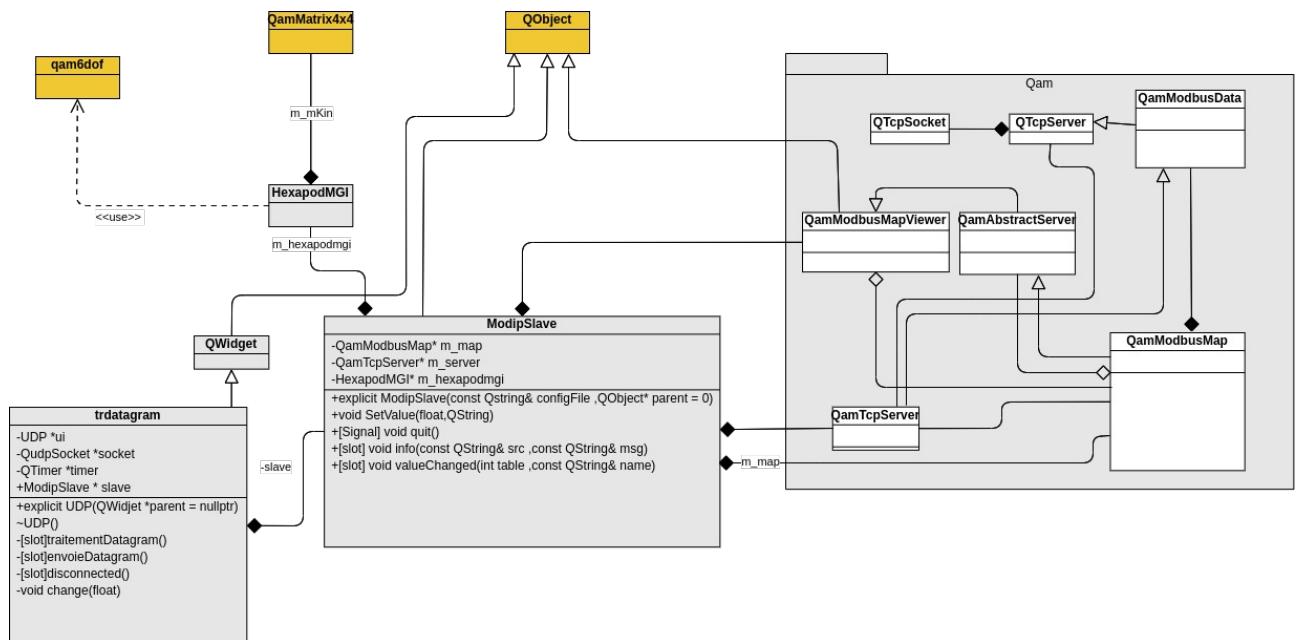
Ces commandes sont envoyées via une carte arduino au poste de contrôle comportant le logiciel

FlightSim qui les transmet au progiciel X Plane permettant la simulation de vol. Il y a un système de projection qui reçoit la sortie vidéo du poste de simulateur de vol.

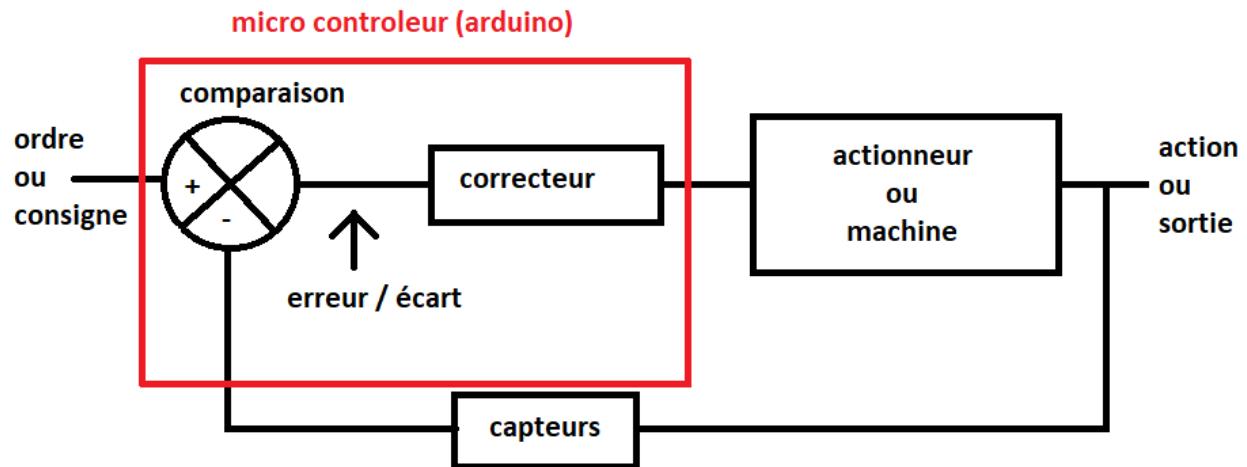
Sur le poste de contrôle, une requête UDP est envoyée à X Plane pour recevoir les données de

position et de rotation de l'hélicoptère. Ces données sont écrites localement dans le serveur Modbus TCP qui les met à disposition aux clients Modbus. De plus, une conversion MGI (Modèle Géométrique Indirecte) permet d'avoir les six longueurs des vérins en fonction des angles de l'hélicoptère. Le premier client Modbus est l'application *FlightSimMockup* qui permet de simuler la position de l'hexapode, ainsi que le siège, les instruments de vol et les commandes. Le deuxième client envoie une requête au serveur Modbus demandant les longueurs des vérins pour pouvoir actionner la maquette de l'hexapode à l'aide d'une carte arduino.

Diagramme de classes



Robotique asservissement

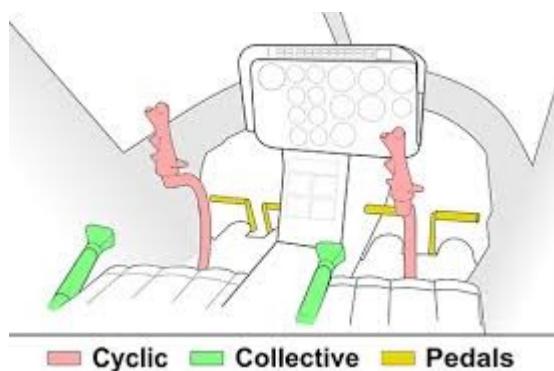


B. Fonctionnement d'un hélicoptère

Comment ça fonctionne ?

Le contrôle d'un hélicoptère en vol s'effectue à l'aide de trois commandes :

- Le collectif ;
- Le cyclique ;
- Le palonnier.



Ces trois commandes ont une incidence directe sur le déplacement des pâles de l'hélicoptère. En tournant les pâles d'un rotor d'hélicoptère, celles-ci génèrent de la portance permettant à l'hélicoptère de voler. En changeant l'angle d'incidence des pâles du rotor principal, on peut faire monter, descendre, s'incliner ou tourner l'hélicoptère.

Le collectif

Dans un hélicoptère, le pilote contrôle le collectif de la main gauche. Ce levier se manœuvre de haut en bas. Pour faire monter l'hélicoptère, le pilote doit lever le collectif, alors que s'il le baisse, l'hélicoptère baissera. En vol stationnaire, c'est le collectif qui contrôle la trajectoire verticale de l'hélicoptère. L'angle de toutes les pales dépend également de la position du collectif. C'est ce qui modifie la portance générée par le rotor. La portance est une action dynamique créée par la vitesse des pales, ce qui soulève l'appareil.



Le cyclique

Le pilote prend le cyclique avec la main droite. Bien qu'il s'apparente à un « joystick » utilisé pour les jeux vidéo, le cyclique est très sensible et par conséquent, il doit être contrôlé avec une grande précision.

C'est grâce au cyclique que le pilote peut faire avancer ou reculer l'appareil. Si le pilote avance le manche, l'hélicoptère avancera, alors que s'il ramène le cyclique vers lui, l'hélicoptère reculera.

Pour tourner à droite ou à gauche, il suffit de bouger le cyclique du côté où vous voulez aller.



Le cyclique ne contrôle pas la direction vers laquelle le devant de l'hélicoptère pointe. Toutefois, il fait pivoter l'hélicoptère vers l'avant, l'arrière, ou de côté.

Le palonnier

C'est le palonnier qui contrôle la direction vers laquelle l'hélicoptère va, par l'entremise des deux pédales qui le constituent.

En appuyant lentement sur la pédale de droite, le pilote fera pivoter le nez de l'hélicoptère vers la droite et vice versa. En fait, le palonnier contrôle le rotor de queue, en modifiant son angle d'attaque. Sans rotor de queue, un hélicoptère pivoterait sur lui-même dans le sens contraire du rotor principal.



C. Terminologie imposée

Les abréviations des noms ci-dessous doivent être utilisés comme indicatifs des noms de classes développées dans le cadre du projet :

- *FlightSim* (abréviation : FS) : nom de l'application de pilotage des instruments des tableaux de bord ;
- *FlightSimMotionControl* (abréviation : FSmc) : nom de l'application d'interface entre X-Plane et les systèmes de mise en mouvements (esclave Modbus) ;
- *FlightSimMotionSystem* (abréviation : FSms) : nom du système de mise en mouvements (client Modbus) pour la maquette de l'hélicoptère (hexapode) ;
- *FlightSimMockup* (abréviation : FSmu) : nom du système virtuel de mise en mouvements par hexapode (client Modbus).

Partie personnelle

Etudiant 1 : Killian LOPES

I. Mon cahier des charges

Mon travail consiste à développer et implémenter la liaison UDP entre l'application *FlightSimMotionControl* et la progicielle X-plane en utilisant les DATAREF mis à disposition par X-plane

Je dois donc développer un programme capable d'envoyer et recevoir de multiple DATAREF venant d'X-plane.

Cela permettra de recevoir les multiples informations de vol. Il faudra aussi les enregistrer dans le serveur Modbus ,c'est valeur seront utilisés pour la création de la cinématique de l'application *FlightSimMockup* ainsi que pour l'hexapode.

L'application *FlightSimMotionControl* devra disposer d'une IHM montrant à minima les données d'entrée ainsi que les longueurs des vérins de l'hexapode.

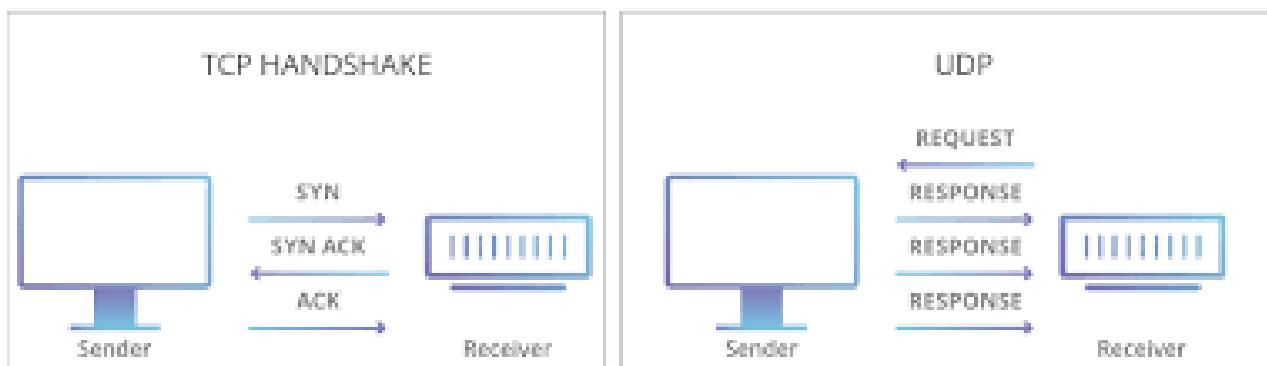
II. Protocole de dialogue UDP

A. Une petite définition

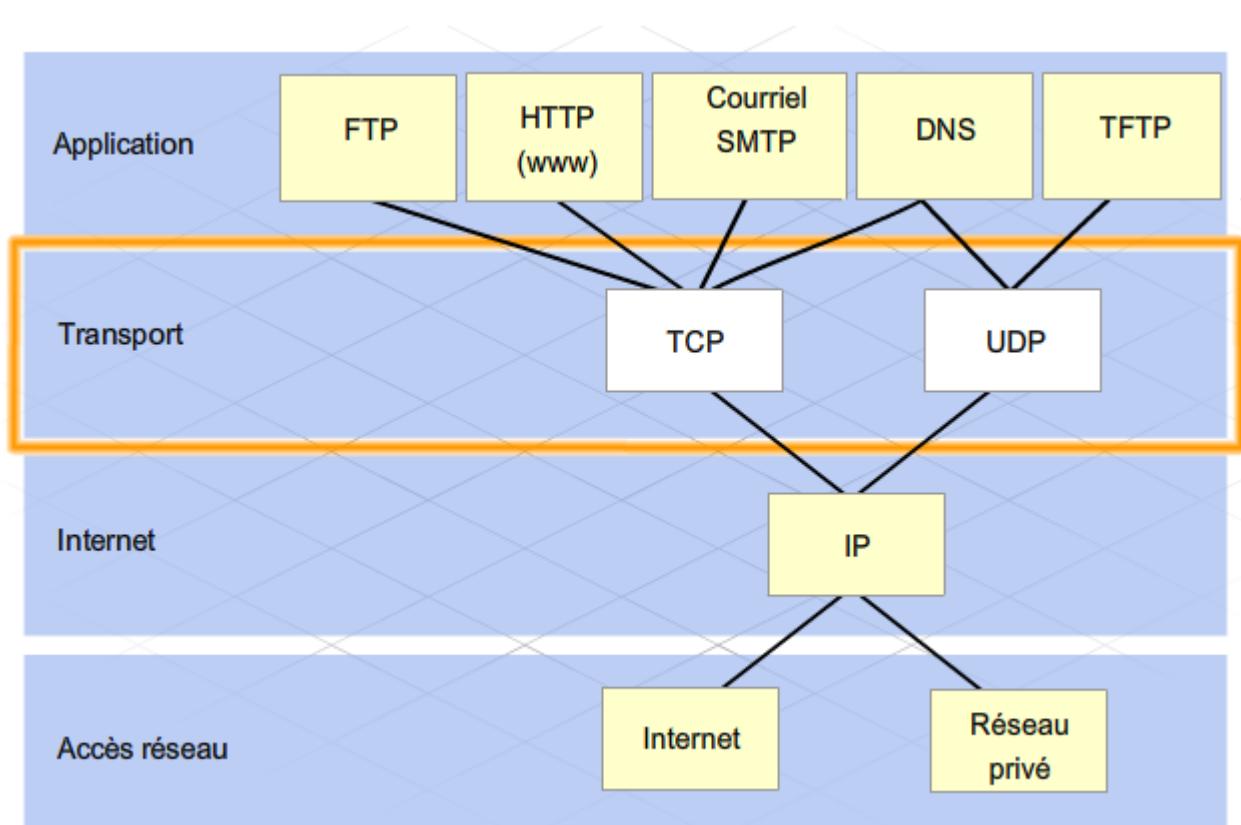
Le protocole UDP ou User Datagram Protocol est un protocole de communication qui est principalement utilisé pour établir des connexions à faible latence et à tolérance de perte entre les applications sur l'internet. Il accélère les transmissions en permettant le transfert de données avant qu'un accord ne soit fourni par la partie destinataire. L'UDP est donc utile pour les communications sensibles au temps, notamment la voix sur IP (VoIP), la consultation du domain name system (DNS) et la lecture vidéo ou audio. L'UDP est une alternative au protocole de contrôle de transmission (TCP) qui permet une connexion plus stable et sécurisée mais rend la connexion plus lente.

L'UDP fournit deux services qui ne sont pas fournis par la couche IP. Il fournit des numéros de port pour aider à distinguer les différentes demandes des utilisateurs et, en option, une capacité de contrôle pour vérifier que les données sont arrivées intactes.

TCP vs UDP Communication



Exemple de communication UDP vs TCP



Modèle TCP/IP

B. Caractéristiques de l'UDP

Le protocole UDP possède des attributs qui le rendent avantageux pour les applications qui peuvent tolérer la perte de données. Par exemple :

- Il permet aux paquets d'être déposés et reçus dans un ordre différent de celui dans lequel ils ont été transmis, ce qui le rend adapté aux applications en temps réel
- Il peut être utilisé lorsqu'un grand nombre de clients sont connectés
- Il est utilisé lorsqu'il n'est pas nécessaire de corriger les erreurs en temps réel, comme dans le cas des jeux, des conférences vocales ou vidéo et des médias en continu.

C. Les avantages de l'UDP

- L'UDP est sans connexion, cela veut dire que le transport des données via le protocole UDP se démarque par le fait qu'il a lieu sans connexion existante entre l'expéditeur et le destinataire. Les paquets concernés sont ensuite envoyés à l'adresse IP privilégiée en indiquant le port de destination, sans que l'ordinateur auquel cette adresse est attribuée n'ait à envoyer une réponse. Si des paquets doivent être renvoyés à l'expéditeur, l'en-tête UDP peut également contenir le port source.
- L'UDP permet une communication rapide et sans délai, ce protocole de transport est adapté à une transmission rapide des données, car il n'établit pas de connexion. Ceci résulte également du fait que la perte de paquets individuels impacte uniquement la qualité de la transmission.

D. les désavantages

L'inconvénient avec une connexion UDP par rapport à une connexion TCP est qu'elle n'est pas fiable:

Dans TCP, si un paquet est perdu, le protocole est capable de le gérer et le paquet est renvoyé.

Alors que la communication UDP fonctionne sans acquisition de message, c'est-à-dire que l'envoyer ne peut savoir si une information n'est pas arrivée ou si elle est corrompue, ce qui veut dire qu'elle ne permet pas d'avoir une connexion aussi sécurisée et stable que le TCP.

III. Les DATAREF et leur utilité

A. Qu'est ce qu'un DATAREF ?

Un DATAREF est une référence de donnée utiliser par le logiciel X-plane pour transmettre ou recevoir via UDP des informations utiliser par des plug-in ou tout autre programme .Il permet de définir toutes les informations de la simulation ,comme par exemple la température extérieure, l'angle d'un aileron , ou bien même la température d'huile d'un des moteurs.

theta	float	660+	yes	degrees	The pitch relative to the plane normal to the Y axis in degrees – OpenGL coordinates
phi	float	660+	yes	degrees	The roll of the aircraft in degrees – OpenGL coordinates
psi	float	660+	yes	degrees	The true heading of the aircraft in degrees from the Z axis – OpenGL coordinates

Exemple de DATAREF

Les trois DATAREF en exemple permettent de lire respectivement le tangage ,le roulis et le lacet du véhicule .

Mais pour pouvoir accéder à ces DATAREF il faut tout d'abord envoyer une requête telle que RREF, DREF ou bien encore RPOS. Ces 3 requêtes sont les plus utilisées parmi les 22 requêtes disponibles :

- Les RREF permettent de demander au progiciel X-plane d'envoyer à un intervalle défini dans la trame UDP de la requête un DATAREF ,ce qui permet de lire une information de vol .
- Les DREF permettent d' interagir avec X-plane en modifiant une information de vol tels que le cap à suivre ou l'altitude souhaitée pour le pilote automatique.
- La requête RPOS fonctionne comme la requête RREF car elle demande a X-plane d'envoyer à un intervalle défini, mais permet de recevoir de multiple information en une seule trame telle que le roulis , le tangage et le lacet(roll ,pitch ,yaw).

0	1	2	3	4	5	6	7	8	9	10	11	12	13 ... 412
'R'	'R'	'E'	'F'	0	freq (int)		id (int)		DataRef (ASCII string)			padding (0)	

Datagramme requête RREF

0	1	2	3	4	5	6	7	8	9	10	11	12
'R'	'R'	'E'	'F'	0	id (int)		value (float)		padding (0)			

réponse de la requête RREF

0	1	2	3	4	5	6	7	8	9	10	11	12	9 ... 508
'D'	'R'	'E'	'F'	0	value (float)		DataRef (ASCII string)		padding (0)				

Datagramme requête DREF

0	1	2	3	4	5	6	7	8
'R'	'P'	'O'	'S'	0	freq (int)			

Now, each frame, X-Plane will send the minimum data needed to drive an external visual or moving map to the same IP address and port number you sent this message from! Easy!

This will simply be the 5 chars "RPOS" (plus a NUL at the end) followed by a struct as follows: (machine-native floats and doubles, 4-byte struct alignment)

```
struct position_struct
{
    double longitude,latitude ;           // double is needed for smooth data here. this
                                         // is good for simple 2-d maps
    float elevation_MSL,elevation_AGL ;   // we have AGL (Above Ground Level) so you can correct the
                                         // height above ground for your visuals to match
                                         // X-Plane, if desired (MSL = Mean See Level)
    float pitch,heading,roll ;            // this is for making your own visuals
    float Vx,Vy,Vz ;                   // Vxyz are the speeds in the east, up, and
                                         // south directions, in meters per second. you
                                         // can use these to predict longitude,
                                         // latitude, and elevation smoothly.
    float Pdeg,Qdeg,Rdeg ;             // P Q R are the roll, pitch, and yaw rates, in
                                         // aircraft axis, in degrees per second. you
                                         // can use these to predict pitch, heading, and
                                         // roll smoothly.
}
```

Datagram size = 65 byte

0	1	2	3	4	5 .. 12	13 .. 20	21 .. 24	25 .. 28
'R'	'P'	'O'	'S'	0	longitude	latitude	elevation_MSL	elevation_AGL

29 .. 32	33 .. 36	37 .. 40	41 .. 44	45 .. 48	49 .. 52	53 .. 56	57 .. 60	61 .. 64
pitch	heading	roll	Vx	Vy	Vz	Pdeg	Qdeg	Rdeg

Datagramme d'envoie et de réception de la requête RPOS

B. Comment les utiliser ?

Pour notre projet nous avons à notre disposition 2 façons de récupérer les informations qui nous sont nécessaires pour la cinématique de l'hexapode. La première consiste à utiliser 3 DATAREF différents, ce qui implique d'envoyer 3 requêtes RREF. La seconde consiste à utiliser une seule requête RPOS, elle permet de recevoir les mêmes informations que les requêtes RREF tout en évitant d'envoyer 3 différentes requêtes RREF, ce qui permet de simplifier le code. C'est donc pour cela que nous allons utiliser uniquement la requête RPOS .

C. implémentation en C++

```
socket->readDatagram(datagram.data(),datagram.size(),&sender,&port); //lecture datagramme recu  
grâce au DATAREF nous pouvons récupérer via cette ligne de code le contenu du datagram de réponse qui pourra être traité grâce à code suivant:
```

```
float* h = (float*)(&datagram[37]);  
QString hs = QString::number(*h);
```

ces deux ligne correspond au code d'extraction d'une des 3 valeur de cinématique, il permet d'extraire les informations souhaitée de la trame datagram et de le convertire en QString pour pouvoir l'envoyer au serveur ModBus ,

```
slave->SetValue( QString::number(round(hs.toFloat()*100)) ,QString("Rz"));
```

ensuit nous utiliserons les 3 variable crée au dessus pour les transmettre au Modèle de Géométrie Indirect (MGI) pour transformer des degré de rotation en longueur de vérin qui seront ensuite transmises au serveur ModBus pour être récupéré par l'application *FlightSimMotionSystem* pour animer d'hexapode.

```
QamMatrix6x1 LengthVerin = slave->MGI(rs.toFloat(),ps.toFloat(),hs.toFloat());
```

IV. Choix de la période d'échantillonnage

La période d'échantillonnage est la fréquence à laquelle les valeurs seront récupérées et enregistrer sur la cartographie Modbus, idéalement la période d'échantillonnage dont permet de transcrire une valeur réelle tout en limitant la charge réseau, ce qu'il veut dire qu'il faut utiliser la période d'échantillonnage la plus faible en tout en ayant une bonne qualité de valeur.

La période d'échantillonnage a été réglée à 50 fois par seconde, cette valeur a été choisie pour permettre la meilleure fluidité dans l'application *FlightSimMockup*, dans le futur une période de rafraîchissement qui assure une parfaite fluidité sur la maquette de l'hexapode sans pour autant surcharger le réseau sera choisie. Idéalement la fréquence d'échantillonnage devrait être la plus faible possible tout en garantissant une animation fluide.

comme vue dans la gestion des DATAREF nous utilisons le DATAREF RPOS qui permet avec une simple commande de demander l'envoi de toutes les informations nécessaires à la cinématique de l'hexapode, un envoi de 50 Hz correspond à une intervalle d'envoyer toutes les 20 ms, ce qui correspond à l'intervalle minimum de *FlightSimMockup*.



```
QByteArray RPOS = "RPOS050" ; //datagrams pour position , envoie a 50Hz  
socket->writeDatagram(RPOS.data(),RPOS.size(),QHostAddress("192.168.0.103"),49000); //envoie demande de datagramme RPOS a Xplane
```

V. Interface Homme Machine

Une Interface Homme Machine ou IHM est un tableau de bord qui permet à un utilisateur de communiquer avec une machine, un programme informatique ou un système comme par exemple un écran de téléphone ,même si le terme IHM est très généralement utilisé dans le milieu industriel.

Une IHM permet d'afficher des données en temps réel et peut aussi permettre d'envoyer des informations au programme.



Interface Homme Machine de l'application FlightSimMotionControl

dans notre cas nous avons opter pour une IHM simple qui nous affiche les informations de la cinématique de l'hélicoptère ainsi que la longueur des vérins envoyée à l'hexapode, l'IHM possède aussi un état de connexion et un bouton qui permettra de quitter l'application *FlightSimMotionControl* , les "TextLabel" contiendront les informations de vol grâce à la ligne de code suivante

```
ui->LL1->setText(QString::number(LengthVerin(0)));
```

Même si tous les prérequis ont été faits , j'ai souhaité ajouter une information supplémentaire au programme , l'état de connexion entre Xplane et *FlightSimMotionControl*.

Toutefois, l'UDP n'est pas connecté : le transport de données via le protocole UDP se distingue par le fait qu'il se déroule sans une connexion existante entre l'expéditeur et le récepteur. Ce protocole de transport permet de transmettre rapidement les données. Cela est aussi dû au fait que la perte de chaque paquet n'affecte que la qualité de la transmission.

Comme le destinataire ne donne aucune réponse, il est théoriquement impossible de savoir si le lien est établi.

```
connect(socket, SIGNAL(readyRead()), this, SLOT(traitementDatagram()), Qt::QueuedConnection);
```

Pour résoudre ce problème nous avons mis en place moi et flavian un connecteur situé dans le constructeur du programme *FlightSimMotionControl* ce qui permet de détecter quand X-Plane envoie les données de l'hélicoptère.

Ce signal appelle la méthode ‘traitementDatagram()’ qui gère ces données.

Moi et flavian avons donc mis un timer pour vérifier si le signal n'est pas émis depuis trop longtemps et donc que la liaison serait déconnectée, pour cela nous allons utiliser la classe QTimer de qt.

Ensuite, nous initialiser une variable timer réglée à 100 ms.

```
timer = new QTimer();
timer->start(100);
```

ensuite nous avons un connecteur de la variable ‘timer’, et un signal ‘timeout()’ qui appelle la méthode ‘disconnected()’.

```
QTimer::connect(timer, SIGNAL(timeout()), this, SLOT(disconnected()));
```

La méthode ‘disconnected()’, permet de modifier le Label d'état de connexion de l'IHM.

Tant que la liaison est établie nous appelons la méthode ‘traitementDatagram()’, nous recommençons le timer à 100 ms et changeons le label dans l'IHM pour afficher l'état connecté.

```
ui->labpitch->setText(ps); //affichage du tangage sur l'application
```

Le code ci-dessus permet d'afficher dans le label ‘labpitch’ l'IHM la valeur de la variable ‘ps’ qui correspond au degré de tangage du simulateur.

VI. FlightSimMockup

L'application *FlightSimMockup* a été développée par l'un de nos professeurs, ce programme permet d'avoir la représentation 3D de l'hexapode. Malheureusement nous avons dû modifier légèrement la méthode de lecture du fichier CSV de cette application pour que *FlightSimMockup* pour lire les valeurs envoyées par *FlightSimMotionControl*. mais cela n'était pas facile ,car l'application a été développée dans une version antérieure à notre version de Qt , vous avons donc dû la mettre à jour en changeant quelle méthode de Qt obsolète , puis nous avons pu faire les changements nécessaires à la nouvelle méthode de communication.



image de la maquette de FSMockup

cette application permet aussi de sauvegarder des paramètres de l'hexapode tel que la taille du plateau supérieur et inférieur ainsi que la longueur des vérins, directement sur le fichier CSV

Hexapod Template

low radius (mm):	500	
low gap (mm):	86	
high radius (mm):	300	
high gap (mm):	35	
min len (mm):	560	
max len (mm):	860	
stroke (mm):	300	
ref. altitude (mm):	437	
max Rx/Ry/Rz (deg.):	15	
max Tx/Ty (mm):	40	
max speed (mm/s):	30	
CSV file:	<input type="button" value="Load"/>	<input type="button" value="Save"/>

VII. Conclusion

Ce projet de grande envergure m'a permis de tester les limites de mes connaissances, avec la communication UDP et le progiciel X-Plane tout en apprenant de nouvelles choses, comme par exemple les méthodes permettant de modifier la cartographie Modbus. Ma tâche n'était pas la plus complexe du projet, mais elle était extrêmement importante, car sans cette partie du projet aucun de mes camarades ne pouvait faire de test grandeur nature et donc valider leur partie

Etudiant 2 : Flavian LAXENAIRE

I. Mon cahier des charges

Mon travail consiste à développer une application autonome nommée *FlightSimMotionControl* dont le rôle sera de maintenir les données d'animation de l'hexapode sous la forme d'une cartographie Modbus.

Je dois mettre en place un serveur Modbus TCP / IP. Pour ce faire, je dois passer par une période d'apprentissage de manière à me familiariser avec le fonctionnement des outils QamModbusMap fournis par mon professeur d'informatique : M. Menu.

Il me faut aussi réfléchir à propos de la cartographie Modbus.

Ce serveur permettra de mettre à disposition les données de l'hélicoptère aux clients modbus. Il me faudra implémenter le MGI (Modèle Géométrique Inverse) en prenant en compte les paramètres de l'hexapode.

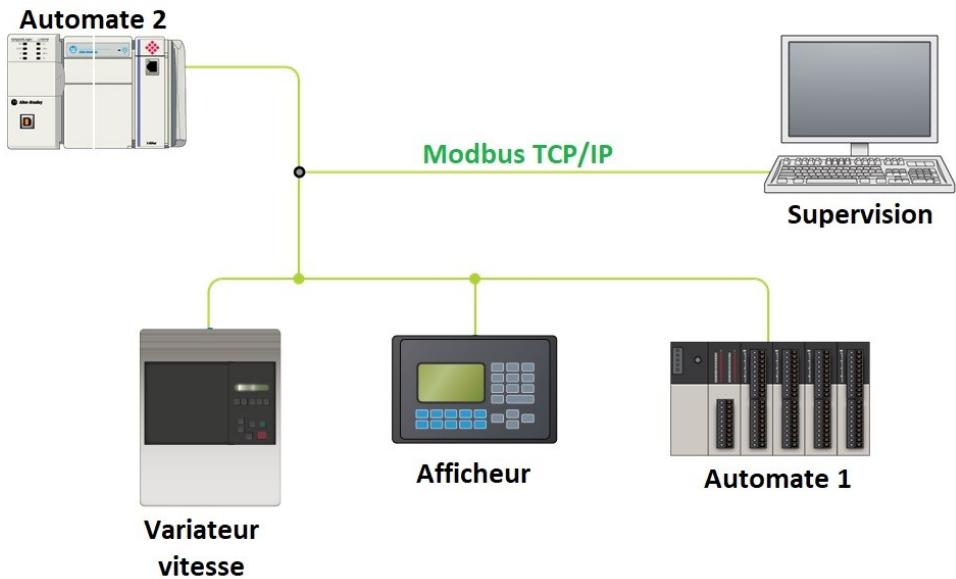
L'application *FlightSimMotionControl* devra disposer d'une IHM montrant à minima les données d'entrée.

II. Le protocole Modbus TCP / IP

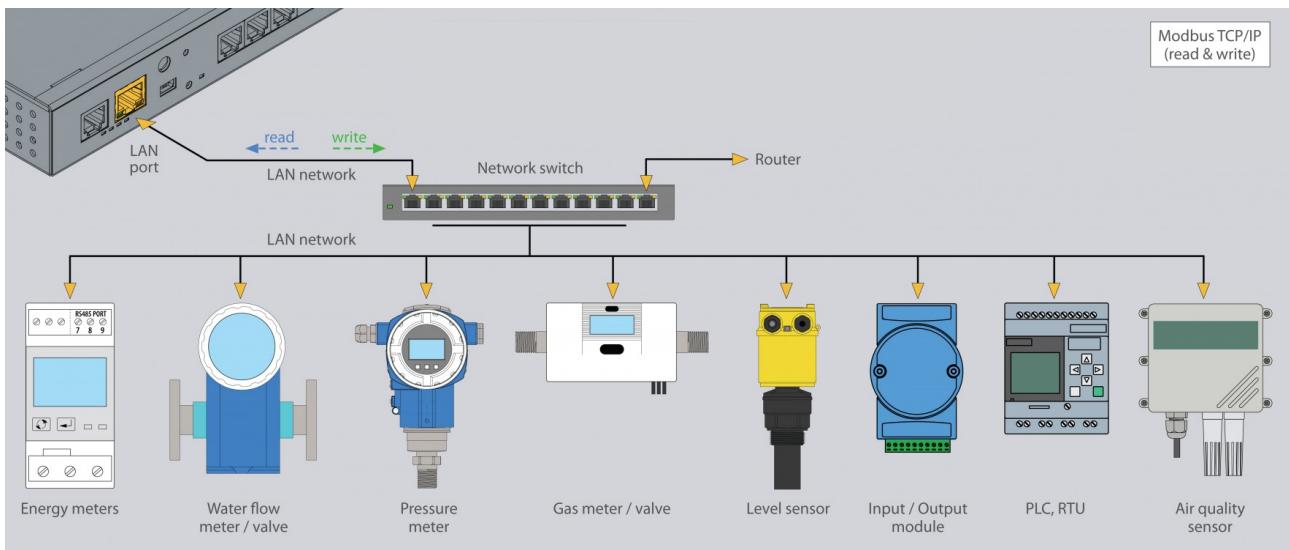
A. Qu'est ce que le protocole Modbus TCP/IP ?

Modbus est un protocole de communication non-propriétaire, créé en 1979 par Modicon, utilisé pour des réseaux d'automates programmables, relevant du niveau applicatif, c'est-à-dire du niveau 7 du Modèle OSI.

Tout comme le Modbus RTU ou ASCII, le protocole Modbus TCP / IP est une variante de la famille Modbus destinée à la supervision et au contrôle des équipements d'automatisation. Plus précisément, il couvre l'utilisation de la messagerie Modbus dans un environnement «Intranet» ou «Internet» utilisant le protocole TCP / IP. L'équipement peut être un automate programmable, une IHM, un variateur de vitesse, un compteur, etc.



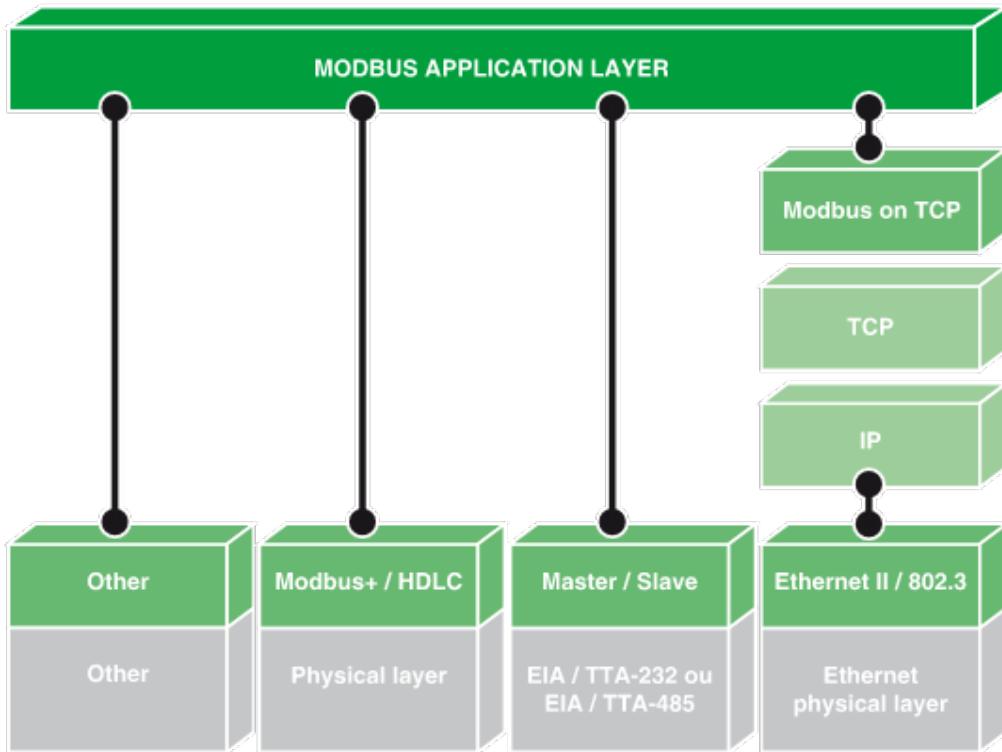
Exemple d'architecture n°1



Exemple d'architecture n°2

Dans le cas d'un réseau Modbus TCP/IP, à la place d'un maître Modbus, on aura un client et à la place d'un esclave, on aura un serveur.

Modbus est un protocole de communication du niveau "application". Il est indépendant de la couche physique (bus).

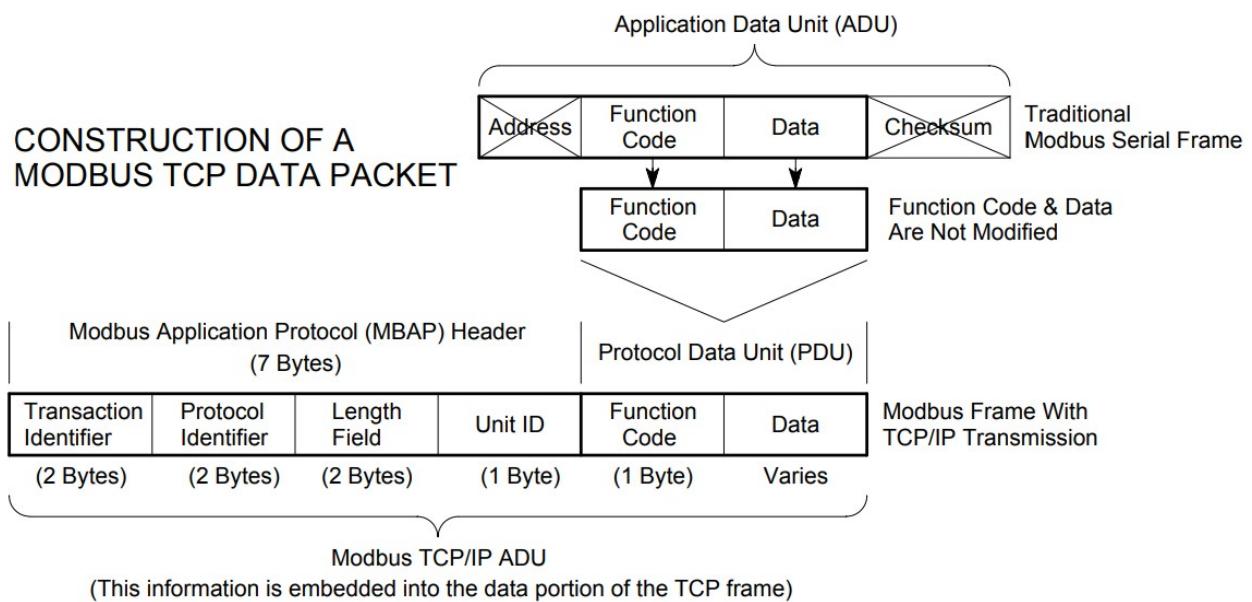


Le Modbus TCP/IP est l'un des protocoles Ethernet industriel les plus utilisés. C'est un protocole flexible et très facile à mettre en œuvre.

B. Comment fonctionne-t-il ?

Modbus TCP/IP utilise le protocole TCP/IP et Ethernet pour transporter les données de la structure de message Modbus entre les appareils compatibles. Autrement dit, le protocole Modbus TCP/IP combine un réseau physique (Ethernet), avec une norme de réseau (TCP/IP) et une méthode standard de représentation des données (Modbus comme protocole d'application). Le port logiciel 502 est destiné à ce protocole.

En pratique, Modbus TCP intègre une trame de données Modbus standard dans une trame TCP, sans la somme de contrôle Modbus, comme illustré dans le diagramme ci-dessous :



Le client, par l'intermédiaire d'une trame requête, va demander des informations au serveur et le serveur va envoyer à son tour une trame de réponse pour lui donner les informations demandées.

C. Pourquoi utiliser ce protocole ?

Contrairement à Modbus RTU, Modbus ASCII et Modbus Serial, le protocole Modbus TCP / IP permet à plusieurs maîtres d'interroger simultanément avec le même appareil esclave. Ceci est permis car à travers le protocole Ethernet via TCP/IP, plusieurs messages peuvent être envoyés, mis en mémoire tampon et livrés sans la nécessité de passer des jetons ou de contrôler totalement le bus, ce qui est souvent le cas avec de nombreux protocoles tels que RS-485 et RS-422.

De plus, la vitesse de transmission est plus importante : 10 Mb, 100 Mb, 1 Gb.

Les supports de transmission sont d'un emploi beaucoup plus souple et leurs coûts moins élevés : fibre optique, liaison radio, etc.

III. Le fichier CSV

A. Qu'est ce qu'un fichier CSV ?

Le sigle CSV signifie Comma-Separated Values et désigne un fichier informatique de type tableur, chaque ligne du texte correspond à une ligne du tableau et chaque virgule correspond à une séparation entre les colonnes. Il est cependant possible d'utiliser un chiffre, un point-virgule, un espace, un onglet ou d'autres caractères.

Même si ce format n'est pas le plus connu qui soit, il est intrinsèquement lié à l'exportation ou l'importation de bases de données.

Exemple de conversion au format .CSV

Version texte du .csv :

40007;1;FFFF;	Rx;	Rotation 0x (roulis), en degrés;	Int;	0
40009;1;FFFF;	Ry;	Rotation 0y (tangage), en degrés;	Int;	0
40011;1;FFFF;	Rz;	Rotation 0z (lacet), en degrés;	Int;	0

Version tableau :

40007	1	FFFF	Rx	Rotation 0x (roulis)	en degrés	Int	0
40009	1	FFFF	Ry	Rotation 0y (tangage)	en degrés	Int	0
40011	1	FFF	Rz	Rotation 0z (lacet)	en degrés	Int	0

B. À quoi servent les fichiers CSV ?

Le format CSV est un format de texte simple qui est utilisé dans de nombreux contextes lorsque de grandes quantités de données doivent être fusionnées sans être directement connectées les unes aux autres.

L'extension de ce type de fichiers est .csv, et ils peuvent être utilisés entre différents outils informatiques et bases de données, lorsqu'on souhaite déployer le contenu d'une base de données sur une feuille de calcul.

Des tableurs tels qu'Excel (Microsoft) ou Calc (LibreOffice) et des bases de données telles que MySQL et Modbus sont capables d'importer et exporter des fichiers CSV. Toutefois, en raison de sa structure basique, le format de fichier CSV ne convient que pour des données structurées simples.

C. Utilisation de ce fichier dans notre projet

J'utilise le fichier en format CSV pour le protocole Modbus. Il sera utilisé pour la configuration du serveur et des clients grâce à la classe QamModusMap.

Le fichier doit respecter les contraintes suivantes :

- lignes de commentaire introduites par un symbole # en première colonne ;
- lignes vides acceptées ;
- adresse IP du serveur (étiquette HOST) : nécessaire en mode Client. En cas d'absence de cette information, le serveur est considéré local à la machine hôte du client ;
- port de service (étiquette PORT) : nécessaire en modes Client et Serveur. En cas d'absence de cette information, le service est supposé sur le port 502 ;
- description du serveur sous forme de texte libre (étiquette INFO) ;
- entrées de table spécifiées par exactement 7 champs :
- champ 1 : numéro dans la table (décimal) ;
- champ 2 : taille en nombre de mots (décimal) en cohérence avec le format d'affichage, peut être laissé à 0 pour les données de taille 1 bit ;
- champ 3 : masque (hexadécimal), FFFF pour les données primaires, peut être laissé à 0 pour les données de taille 1 bit ;
- champ 4 : nom (texte libre) ;
- champ 5 : commentaire (texte libre) ;
- champ 6 : format d'affichage, à choisir parmi Hex, Bool, UInt, Int, Ascii, Bcd ; Float, Long ; Str8 ou Str16 ;
- champ 7 : valeur, exprimée conformément au format d'affichage.

```
HOST;127.0.0.1
PORT;502
INFO;FLIGHTSIM HEXAPOD

# hexapod template

40201;1;FFFF;    baseRadius; Rayon ancrages base, em mm ;           Int;  500
40202;1;FFFF;    baseGap;   1/2 écart ancrages base, em mm ;           Int;  86
40203;1;FFFF;    topRadius; Rayon ancrages platine, em mm ;          Int; 300
40204;1;FFFF;    topGap;    1/2 écart ancrages platine, em mm ;          Int;  35
40205;1;FFFF;    minLen;    longueur min. (vérin rentré), em mm ;        Int; 560
40206;1;FFFF;    maxLen;    longueur max. (vérin sorti), em mm ;        Int; 860
40207;1;FFFF;    maxAngle;  angle absolu max. Rx,Ry,Rz, en degrés ;      Int;  15
40208;1;FFFF;    maxTrans;  déplacement abs. max. Tx,Ty, en mm ;        Int;  40
40209;1;FFFF;    maxSpeed;  vitesse linéaire max., em mm/s ;           Int;  30
```

Aperçu du fichier CSV utilisé dans le projet Hexapod

IV. QamModbus

La librairie Qam a été créée par mon professeur d'informatique, M. Menu. Il a fait toute une librairie appelée Qam3D qui permet la modélisation 3D d'objets. En parallèle, il a aussi fait en 2014 la librairie QamModbus qui permet la gestion d'un serveur et d'un client Modbus. C'est cette librairie que j'utilise.

A. Diagramme UML/SYML

Diagramme de classes

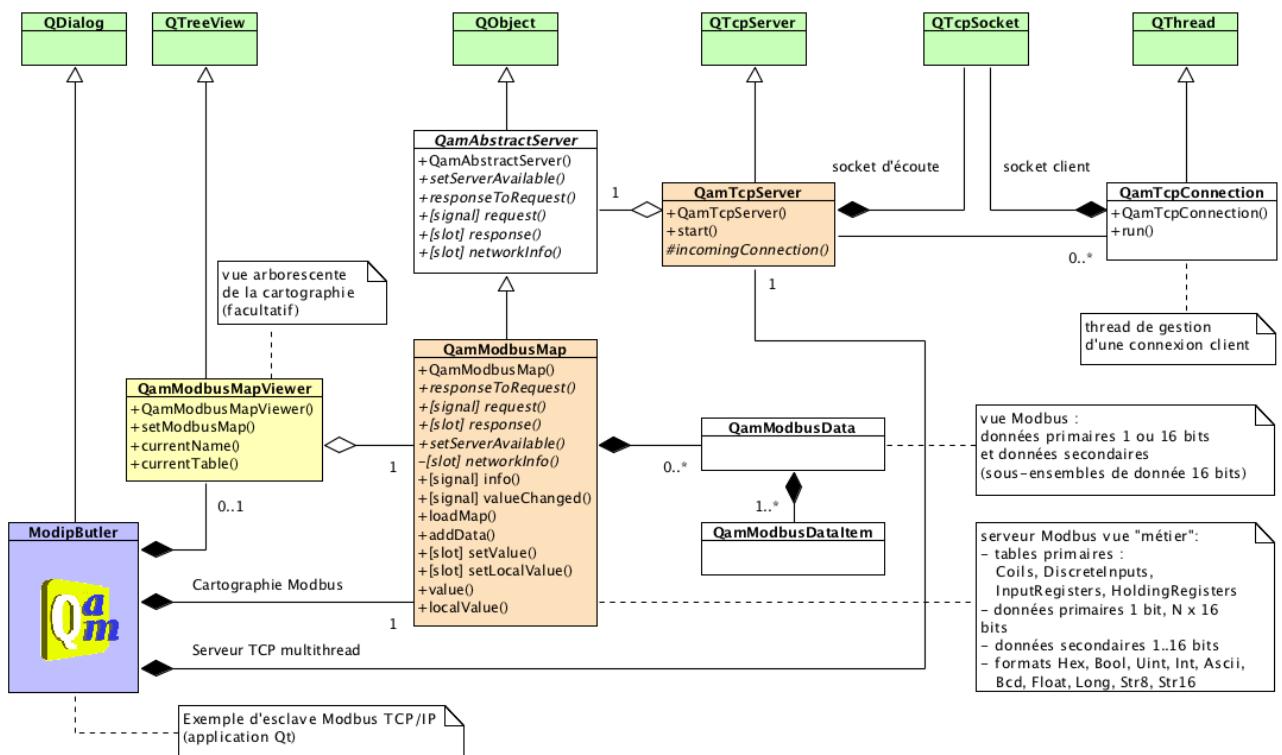


Diagramme de classes du serveur modbus

Diagramme de séquence

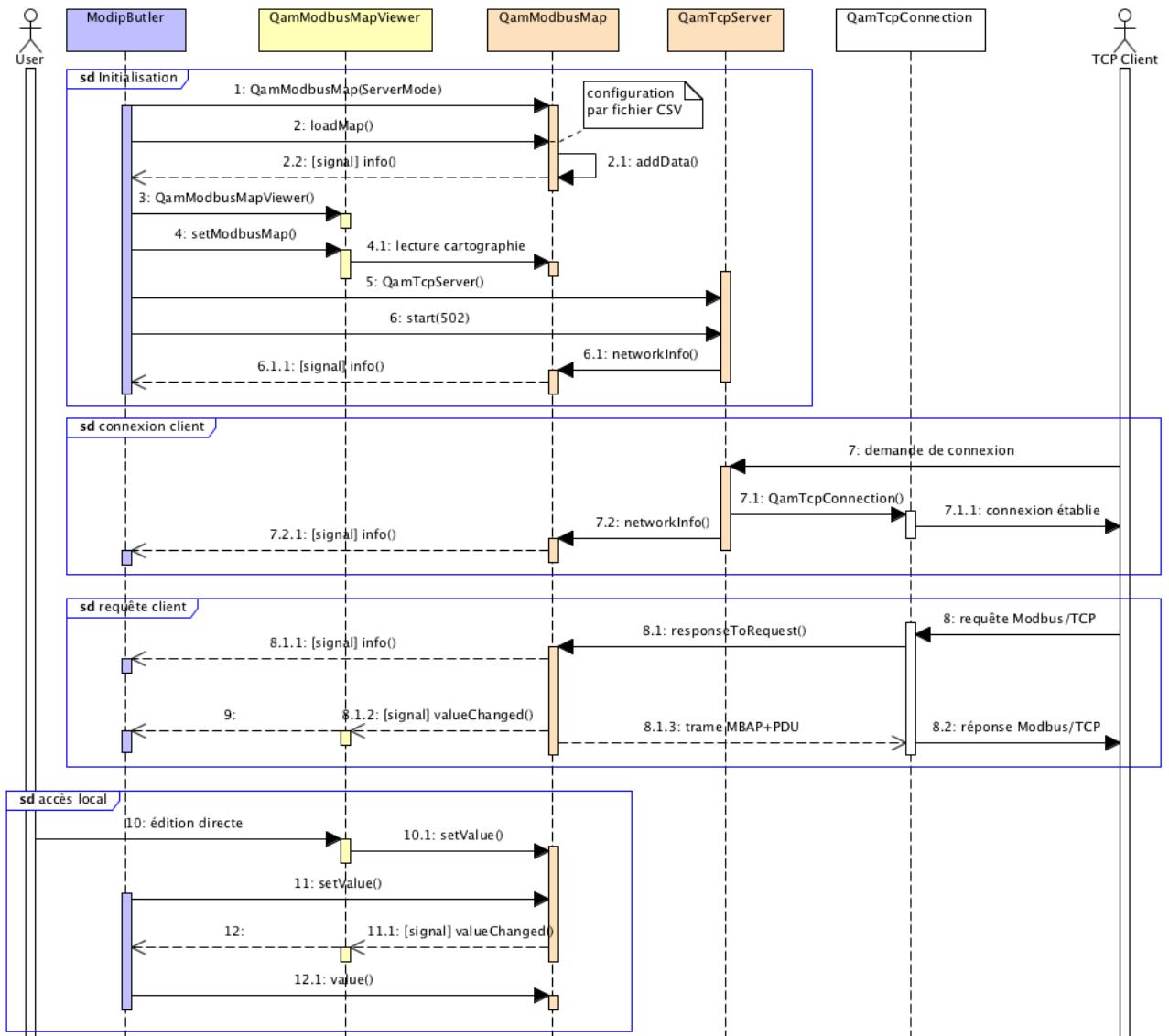


Diagramme de séquence du serveur Modbus

B. Descriptions des classes

Voici les classes qui sont utilisées dans ce projet :

QamAbstractServer :

La classe QamAbstractServer sert de modèle de développement pour l'interlocuteur des classes QamTcpServer et QamTcpConnection.

En pratique, elle modélise la couche TCP/IP 'Application'

QamModbusData

Modélisation d'une donnée de table primaire, 1 ou 16 bits.

Cela représente une ligne sur le fichier CSV, avec ses différentes colonnes.

QamModbusDataItem

Modélisation d'une donnée primaire ou d'un champ partiel de donnée 16 bits.

Elle encapsule les caractéristiques d'une donnée : nom, commentaire, masque, format et valeur.

Cela représente une colonne sur le fichier CSV.

QamModbusMap

Modélisation de la cartographie d'un équipement Modbus.

La classe QamModbusMap est conçue de manière à permettre à deux applications Qt jouant respectivement les rôles d'esclave et de maître sur un réseau Modbus TCP/IP de partager un seul et unique fichier de description de cartographie (format CSV).

Cette classe permet de modéliser d'un point de vue métier une cartographie Modbus.

La vue métier permet de manipuler les données d'une cartographie non plus directement par leur adresse mais par un nom de rôle (champ 4 du fichier CSV).

Les données primaires constituant la cartographie d'un équipement sont réparties dans quatre tables :

- Table Coils : données 1 bit accessibles en lecture et écriture ;
- Table Discrete Inputs : données 1 bit en lecture seule ;
- Table Input Registers : données 16 bits en lecture seule ;
- Table Holding Registers : données 16 bits en lecture/écriture.

J'utilise la table Holding Registers qui prend comme adresse de 40001 à 105536 et qui permet la lecture et l'écriture de données de 16 bits.

Cette classe permet d'accéder au fichier CSV, il est possible de lire et écrire en tant que serveur ou client via cette classe. Notamment grâce à ces méthodes :

- QString localValue (PrimaryTable table, const QString &name)
- QString remoteValue (PrimaryTable table, const QString &name)
- QString display (PrimaryTable table, const QString &name)

QamModbusMapView

La classe QamModbusMapView permet de visualiser sous forme d'arbre la cartographie d'un serveur Modbus encapsulée dans un objet de classe QamModbusMap

QamTcpClient

La classe QamTcpClient permet la mise en œuvre d'un client TCP/IP. La logique applicative d'envoi de requêtes et de traitement des réponses peut être assurée par un objet répondant au modèle de développement spécifié par la classe QamAbstractServer.

QamTcpConnection

Thread de gestion d'une connexion TCP.

La classe QamTcpConnection prend en charge l'établissement et la gestion d'une connexion TCP avec un client.

QamTcpServer

La classe QamTcpServer permet la mise en œuvre d'un serveur TCP/IP multi-clients.

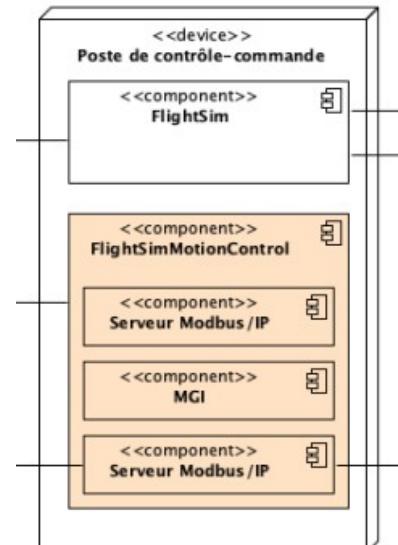
Les méthodes utilisées pour la gestion du serveur Modbus sont :

- QamTcpServer (QamAbstractServer *server, QObject *parent=0)
- void start (int listenPort=4000)

V. FlightSimMotionControl

FlightSimMotionControl est le nom de l'application sur laquelle Killian et moi travaillons. Il y a donc la liaison UDP entre X-Plane et cette application, ainsi que le serveur Modbus et le MGI.

Les applications *FlightSim* et *FlightSimMotionControl* sont hébergées par le même poste sous Linux. Dans le futur, il est probable que *FlightSimMotionControl* devienne une option intégrée à *FlightSim*.



A. Interface Homme Machine

L'application *FlightSimMotionControl* doit disposer d'une IHM montrant les données d'entrée, ainsi que la sortie du MGI.

Architecture

La première ligne affiche les données de rotation de l'hélicoptère reçues d'X-Plane via la liaison UDP.

La deuxième ligne affiche les six longueurs des vérins résultant du MGI.

En haut à droite, il y a l'état de connexion de la liaison UDP avec le progiciel X-Plane.

En bas, il y a le bouton "quitter" pour fermer l'application *FlightSimMotionControl*.



IHM de *FlightSimMotionControl*

Problématique

Killian souhaitait pouvoir afficher l'état de connexion de la liaison UDP sur l'IHM.

Cependant, l'UDP est sans connexion : le transport des données via le protocole UDP se démarque par le fait qu'il a lieu sans connexion existante entre l'expéditeur et le destinataire. Ce protocole de transport est adapté à une transmission rapide des données. Ceci résulte également du fait que la perte de paquets individuels impacte uniquement la qualité de la transmission.

N'ayant pas de trame réponse venant du destinataire, il est théoriquement impossible de savoir si la liaison est établie.

Résolution

Il y a un connecteur dans le constructeur du programme de Killian :

```
connect(socket, SIGNAL(readyRead()), this, SLOT(traitementDatagram()),Qt::QueuedConnection);
```

Un signal est émis quand X-Plane envoie les données de l'hélicoptère.

Ce signal appelle la méthode ‘traitementDatagram()’ qui gère ces données.

J'ai donc pensé qu'il serait possible de mettre un timer pour savoir si le signal n'est pas émis depuis trop longtemps et donc que la liaison serait déconnectée.

Pour ce faire, on utilise la classe : QTimer de qt.

Dans le constructeur on déclare la variable ‘timer’ et on lance le décompte de 100 ms.

```
timer = new QTimer();
timer->start(100);
```

Il y a ensuite un connecteur lié à la variable ‘timer’, ainsi que le signal ‘timeout()’ qui appelle la méthode ‘disconnected()’.

```
QTimer::connect(timer , SIGNAL(timeout()),this,SLOT(disconnected()));
```

Dans la méthode ‘disconnected()’, nous changeons le label de l'IHM pour afficher l'état déconnecté.

Si la liaison est établie et que les données sont reçues, dans la méthode ‘traitementDatagram()’, nous recommandons le timer à 100 ms et changeons le label dans l'IHM pour afficher l'état connecté.

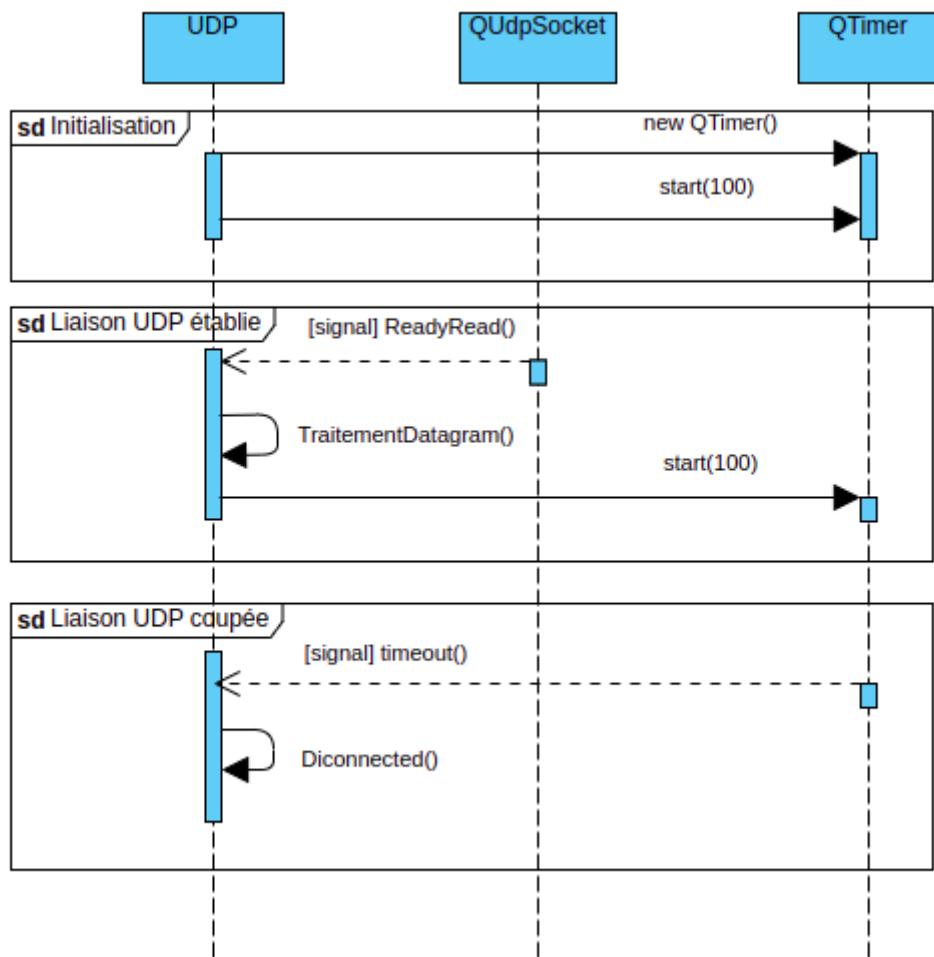


Diagramme de séquence de l'état de connexion

B. Serveur Modbus

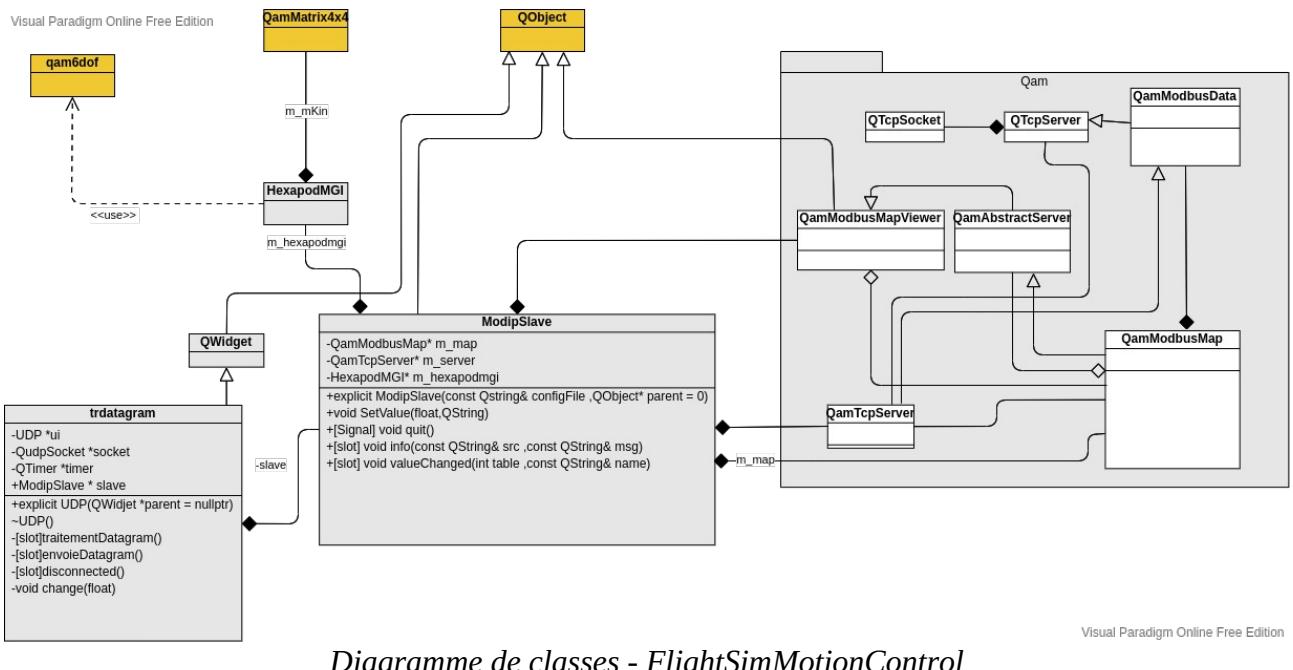


Diagramme de classes - FlightSimMotionControl

La classe ModipSlave s'occupe de gérer le serveur Modbus.
Elle est composée de plusieurs classes de QamModbus.

```
ModipSlave::ModipSlave(const QString& configFile, QObject* parent )
: QObject(parent)
{
    // cartographie Modbus

    m_map      = new QamModbusMap( QamModbusMap::ServerMode, this ) ;
    m_map->setVerbose( false ) ;

    connect( m_map, SIGNAL(info(QString(QString)), this, SLOT(info(QString(QString))) ) ;
    connect( m_map, SIGNAL(valueChanged(int,QString)), this, SLOT(valueChanged(int,QString)) ) ;

    m_map->loadMap( configFile ) ;

    // serveur TCP

    m_server = new QamTcpServer( m_map, this ) ;
    m_hexapodmgi = new HexapodMGI();
    m_server->start( m_map->port() ) ;
}
```

Constructeur de ModipSlave

Dans le constructeur de la classe ModipSlave :

- On crée un objet de la classe QamModbusMap en mode Serveur
Ayant le même fichier CSV c'est ce qui différenciera le programme serveur et client
- On charge un fichier de configuration
- On crée un objet de la classe QamTcpServer avec en argument l'objet de configuration
- On crée un objet de la classe HexapodMGI qui servira à implémenter le MGI.
- On lance le server sur le port spécifié dans le fichier de configuration (502)

VI. MGI

Le modèle géométrique inverse (MGI) est un modèle mécanique utilisé en robotique pour les bras manipulateurs.

Ce modèle permet de déterminer la configuration des liaisons, en fonction de la configuration (position et orientation) d'un robot. Il existe 2 types de liaisons :

- les liaisons pivots (Des angles : Rx, Ry, Rz)
- les liaisons glissières. (Des translations : Tx, Ty, Tz)

Le modèle géométrique inverse est utilisé pour caractériser le fonctionnement d'un bras manipulateur.

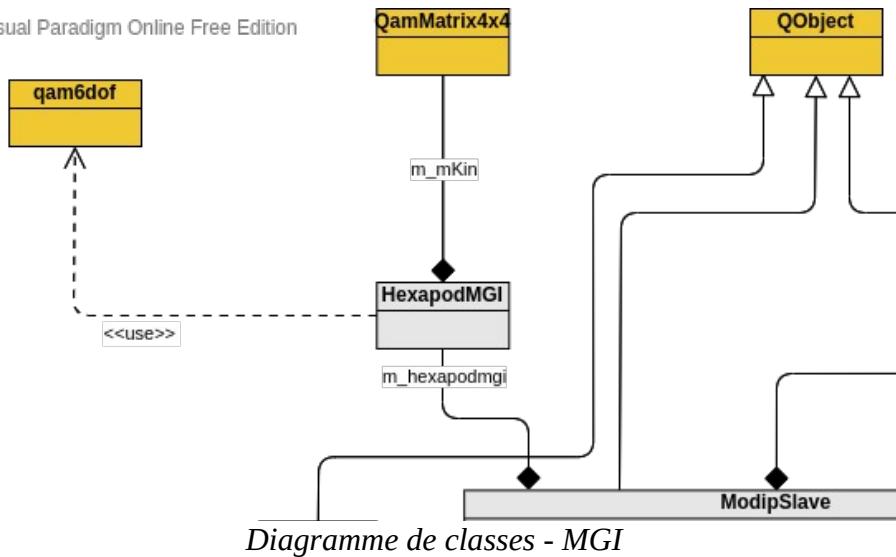
Concrètement, dans notre projet, le modèle géométrique inverse permet de calculer la position de chaque vérin de l'hexapode en fonction de la position et de l'orientation de l'hélicoptère.

La fonction inverse qui consiste à déterminer la position et l'orientation de l'hélicoptère en fonction de la configuration de l'hexapode est le modèle géométrique direct (MGD).

Sur l'application *FlightSimMockup*, qui permet de simuler la configuration de l'hexapode, l'implémentation du MGI et MGD est déjà faite.

Ce qu'il a fallu que je fasse c'est l'implémentation du MGI sur FlightSimMotionControl pour pouvoir convertir les angles reçus du logiciel X-Plane (Rx, Ry, Rz) en six longueurs de vérins dans une matrice 6x1.

J'ai pour cela utilisé la classe HexapodMGI qui est composée de la classe QamMatrix4x4 et qui inclut le fichier qam6dof qui possède la classe Qam matrix 1x6.



```
void HexapodMGI::setMGI(QamMatrix6x1 vKin )
{
    m_vKin = vKin ;

    m_mKin.setToIdentity(); // calcul matrice 4x4

    m_mKin.translate( QVector3D(m_vKin(0), m_vKin(1), m_vKin(2) ) );
    m_mKin.rotate(m_vKin(3), 1, 0, 0 ) ;
    m_mKin.rotate(m_vKin(4), 0, 1, 0 ) ;
    m_mKin.rotate(m_vKin(5), 0, 0, 1 ) ;

    for ( int i = 0 ; i < 6 ; ++i ) vlength(i); // maj m_vLen
}
```

La conversion utilise des matrices 6x1 pour les 6 données de l'hélicoptère (Tx, Ty, Tz, Rx, Ry, Rz). Puis des matrices de translations et de rotations sont faites en fonction de ces données.

Nous n'implémentons pas l'accélération pour le moment, nous mettons donc les valeurs de translations à 0.

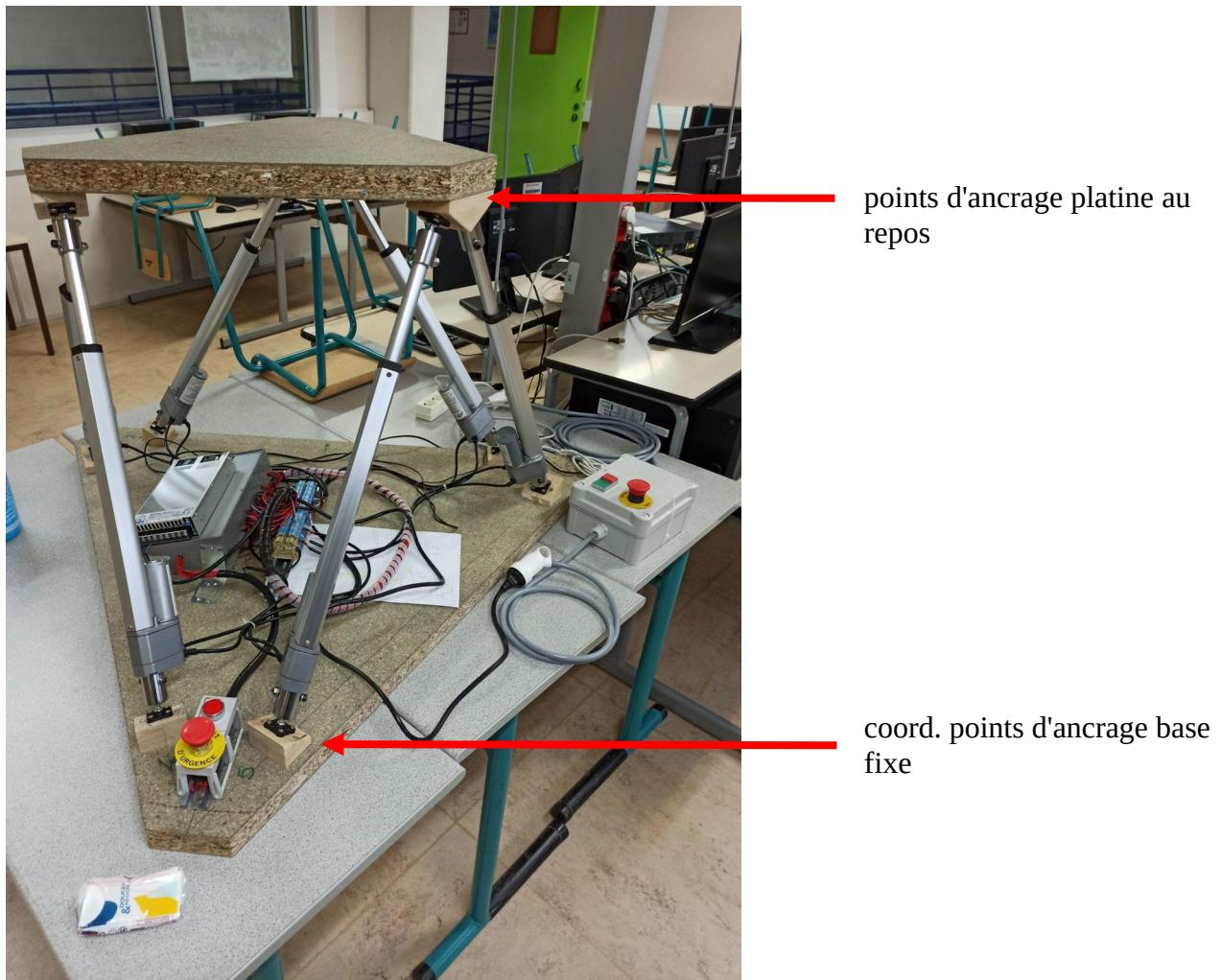
La nouvelle matrice est passée en argument dans la méthode vlength qui convertira en longueurs de vérins.

```
float HexapodMGI::vlength(int i )
{
    QVector3D v = m_mKin.map( m_pAncRef[i] ) - m_bAnc[i] ;

// float eps = (m_pMaxAlt - m_pMinAlt) * 0.01 ;
// bool err = ( v.length() > m_vMax + eps )||( v.length() < m_vMin - eps ) ;
    m_vLen(i) = v.length() ;

    return v.length();
}
```

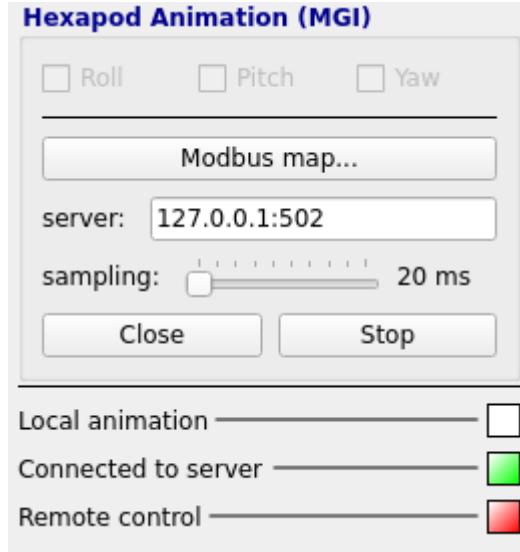
Un vecteur est créé en soustrayant la position du vérin en haut de l'hexapode, moins la position en bas de l'hexapode.



VII. FlightSimMockup

A. Caractéristiques

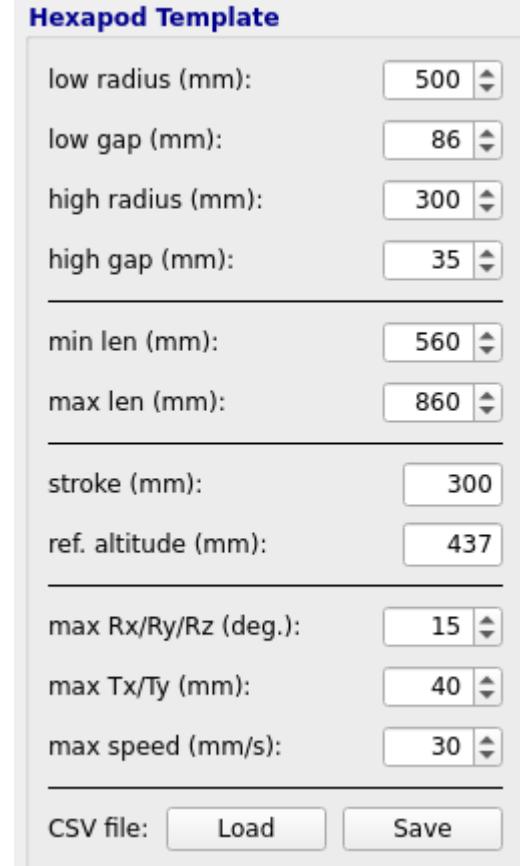
L'application FlightSimMockup a été faite par mon professeur d'informatique, M. Menu. Elle permet de simuler l'hexapode virtuellement.



Cette application possède un client Modbus qui se connecte à mon serveur.

Il charge la cartographie Modbus qui est la même que le serveur (grâce à QamModbusMap).

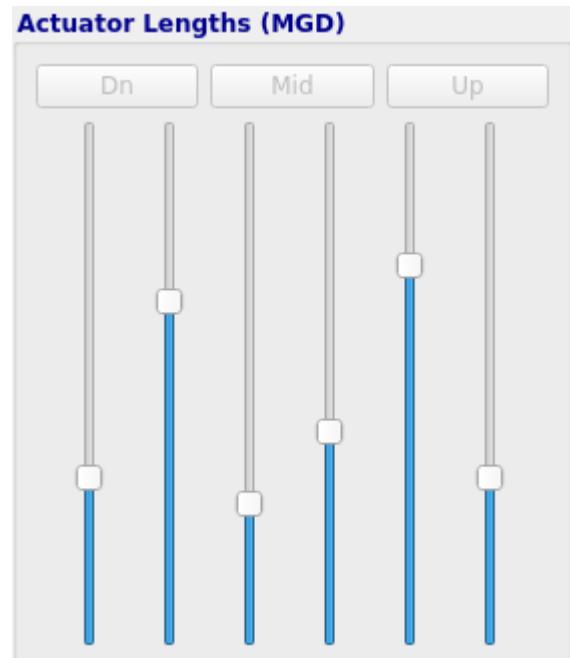
Il prend l'adresse IP du serveur, ainsi que son port et envoie des requêtes pour recevoir les angles de l'hélicoptère (Rx, Ry, Rz).



Il y a la possibilité de pouvoir changer les caractéristiques de l'hexapode directement sur l'application. Telles que :

- la longueur minimale et maximale des vérins
- le degré de liberté des rotations
- la vitesse maximale

On peut aussi charger ou sauvegarder une configuration personnalisée.



Hexapod

L'application à déjà un MGI intégré, il n'est pas nécessaire d'avoir les longueurs de vérins, il peut les convertir sur le moment.

De plus, il y a un MGD (modèle géométrique direct). Les 6 curseurs correspondent aux 6 vérins. En bougeant ces curseurs, l'hexapode s'anime en temps réel.

En haut à gauche, il y a la possibilité de pouvoir changer les caractéristiques de l'hexapode directement sur l'application qui partagent le même fichier CSV que le serveur.



Grâce à cette application mise à notre disposition, j'ai pu constater si mon application *FlightSimMotionControl* fonctionnait correctement.

Cela me permet de voir si le serveur est en état de marche et reçoit les requêtes clients.

B. Débogage

Problématique

Quand cette application nous a été donnée, elle ne marchait pas.

Lorsque nous la lancions avec Killian, l'application ‘crashait’, s’arrêtait en mettant le message d’erreur suivant :

Please check your inputs (1)

J’ai perdu beaucoup de temps à essayer de le déboguer. J’ai fini par prendre une version antérieure sans MGI/MGD, ni client Modbus, juste pour pouvoir voir le rendu de l’hexapode virtuellement.

Après avoir fini ma partie personnelle, je me suis remis à trouver le pourquoi de l’erreur.
J’ai essayé de passer à la version de QT 5.15.2 mais sans résultats.

De plus, ce programme ne marchait que sur le MAC de mon professeur alors que j’ai essayé ce programme sur une dizaine d’ordinateurs avec différents OS, ainsi que différentes versions de QT.

```
void Hexapod::draw(float globalScale )
{
    m_base->transformMatrix().setToIdentity() ;

    m_base->transformMatrix().scale( globalScale ) ;

    m_base->transformMatrix().translate( 0, 0, -m_pMinAlt - m_base->thickness() ) ;

    m_base->draw() ;

    m_platine->transformMatrix().setToIdentity() ;
    m_platine->transformMatrix().scale( globalScale ) ;
    m_platine->transformMatrix() *= m_mKinMockup ;
    m_platine->draw() ;
```

La méthode draw() de la classe Hexapod est supposée modéliser l’hexapode sur l’application grâce aux caractéristiques du fichier de configuration (CSV).

Cependant, en mettant des qDebug() un peu partout pour voir où le programme plantait, j’ai remarqué qu’il s’arrêtait à l’appel de cette fonction draw().

Résolution

Il se trouve que le concepteur de cette application, mon professeur d’informatique, avait mis en place un système d’inter session. Cela consiste à garder en mémoire les paramètres de l’hexapode lorsque l’on ferme le programme. Néanmoins, il fallait mettre des valeurs par défaut pour le premier lancement de l’application pour que le système d’inter session sauvegarde ces données.

Le programme plantait puisqu’il essayait de modéliser un objet avec des valeurs nulles.

Pour corriger ce problème, il suffisait de rajouter un constructeur de la classe hexapodData (qui stock les paramètres) qui déclare les variables avec des valeurs par défaut.

```

setData(HexapodData());
}

HexapodData()
{
    baseRadius = 500;
    baseGap = 86;
    topRadius = 300;
    topGap = 35;
    minLen = 560;
    maxLen = 860;
    maxAngle = 15;
    maxTrans = 40;
    maxSpeed = 30;
}

```

VIII. Conclusion

Ce projet d'envergure m'a permis de mettre en pratique mes connaissances, avec le serveur Modbus, ainsi que d'en acquérir de nouvelles, avec le MGI.

En tant que chef de projet, j'ai dû gérer son bon déroulement et aider mes camarades sur différentes problématiques.

C'est ainsi que j'ai pu constater l'importance du travail d'équipe et l'impact que peut avoir le retard de certaines tâches sur les parties d'autres personnes du groupe.

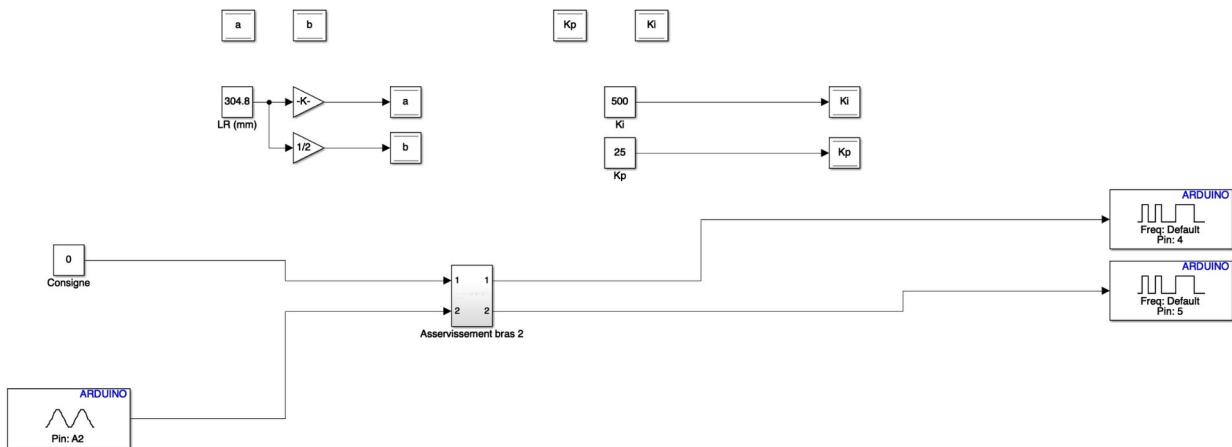
J'ai appris que le métier de développeur nécessitait parfois de déboguer le programme des autres et de le comprendre.

Etudiant 3 : Yohann RAIMBAULT

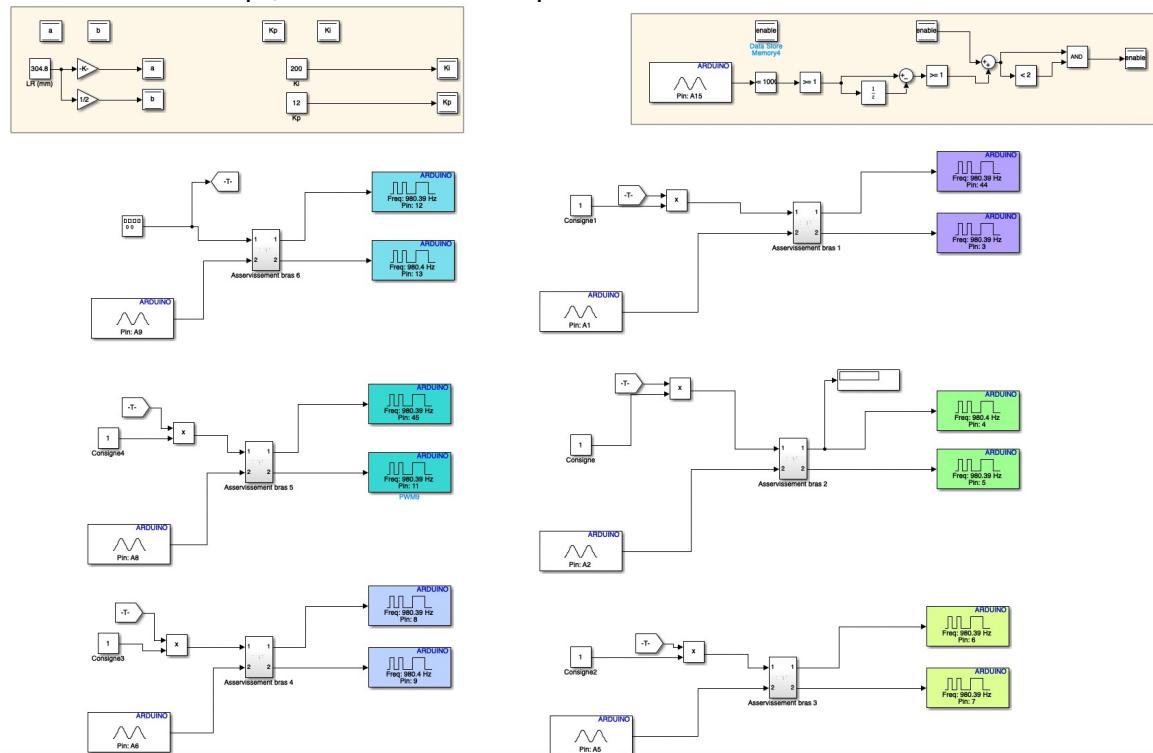
I. Mon cahier des charges

Dans ce projet, je devais réaliser un programme Matlab/Simulink afin de récupérer les informations de longueurs de vérins maintenues par le serveur Modbus. Je devais donc créer un client Modbus afin de récupérer ces informations.

Pour ce faire, j'ai dans un premier temps réalisé un schéma sur Simulink afin de réaliser la mise en mouvement d'un seul vérin.



J'ai dans un second temps, réalisé un schéma pour la mise en marche des six vérins.



II. Matlab/Simulink

A. Qu'est ce que Matlab?

Tout d'abord, Matlab est un langage de programmation. Il est utilisé pour des calculs numériques. Il permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++.

B. Qu'est ce que Simulink ?

Simulink est un logiciel de modélisation. Il est intégré dans Matlab, fournissant ainsi un accès immédiat aux nombreux outils de développement algorithmique, de visualisation et d'analyse de données de MATLAB.

C. Comment fonctionne-t-il ?

Tout d'abord, afin de pouvoir commencer à utiliser Simulink, vous devez lancer Matlab. Ensuite, une fois que nous sommes sur Simulink, nous avons les librairies qui nous permettent d'avoir accès à un très grand nombre de blocs afin de réaliser notre schéma. Une fois que nous avons réalisé notre schéma, nous pouvons vérifier si celui-ci fonctionne correctement.

III. Programme Simulink

Pour réaliser le programme Simulink afin que les vérins fonctionnent, j'ai dans un premier temps pris les informations des vérins. Pour ce faire, j'ai regardé la fiche technique de ces derniers.



Comme nous pouvons le constater sur la fiche technique, nous devons respecter certains critères.

Nous observons que la puissance d'entrée est de 12 Watts.

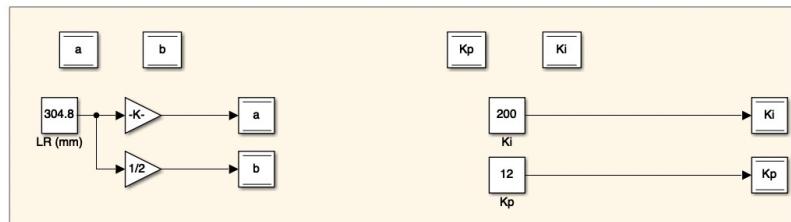
Le poids maximal pour chaque vérin est de 150 Lbs ce qui correspond à 68Kg.

Son rapport cyclique (durée pendant laquelle une charge ou un circuit est sous tension et la durée pendant laquelle il/elle est hors tension) est de 20%.

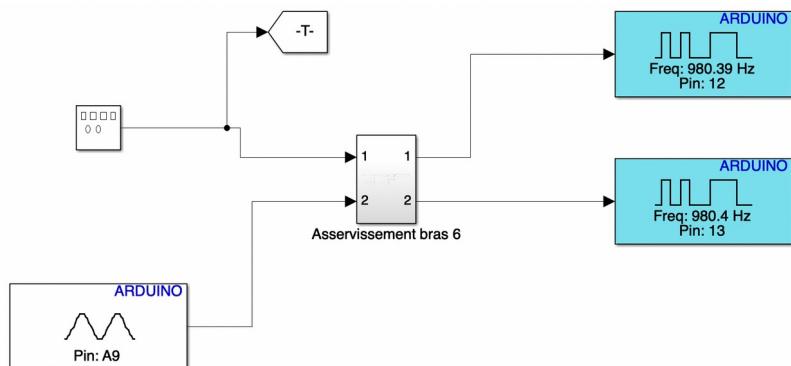
Nous remarquons également que son IP Protection (Indice de Protection) est 54, ce qui signifie qu'il est protégé contre les poussières qui pourraient interférer au bon fonctionnement du vérin, mais n'est pas complètement étanche à la poussière. Mais il est aussi protégé contre les éclaboussures d'eau sous tous les angles.

En bas de la fiche technique, nous pouvons également observer deux schémas. Le premier représente la résistance et le second le moteur.

Pour réaliser le programme Simulink et afin que celui-ci soit le plus clair possible, des mémoires de stockage de données ont été utilisées.



Comme nous l'avons constaté sur la fiche technique du vérin, celui-ci mesure 12 pouces, ce qui correspond à 30,48cm ou encore 304,8mm que nous retrouvons dans le schéma ci-dessus.

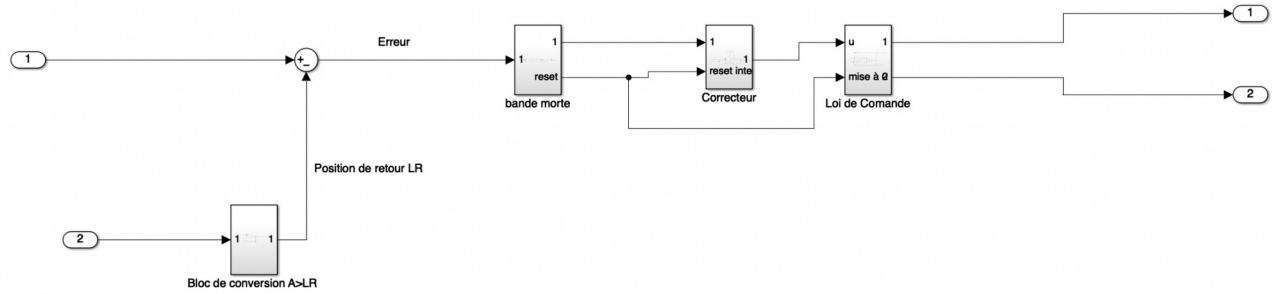


Dans le schéma ci-dessus, nous retrouvons une entrée Arduino sur le port Analogique 9, et nous retrouvons en sortie le port Numérique 12 et le port Numérique 13. On peut également y retrouver un bloc de sous-système nommé Asservissement bras 6 qui correspond au vérin numéro 6.

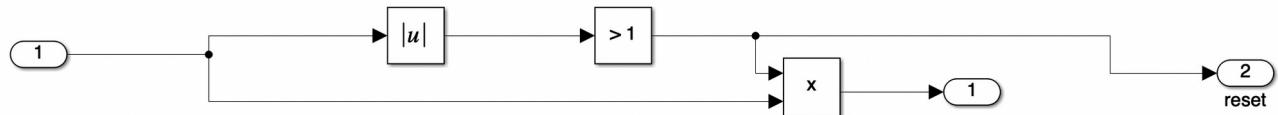
Nous retrouvons également en entrée un générateur de signal ayant pour amplitude 20 et pour fréquence 4.

Le bloc -T- correspond à un Goto, il transmet son entrée à ses blocs From correspondants. Cette entrée peut être un signal ou un vecteur à valeurs réelles ou complexes de n'importe quel type de données. Les blocs From et Goto vous permettent de passer d'un bloc à un autre sans les connecter.

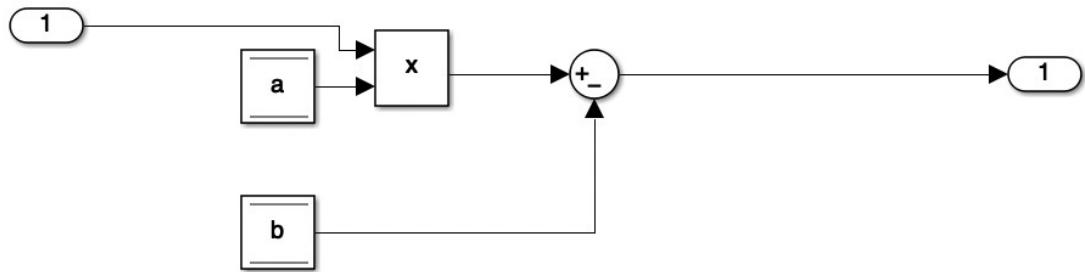
Lorsque nous rentrons dans le bloc de sous-système, nous retrouvons le schéma ci-dessous.



Dans ce schéma, nous apercevons plusieurs blocs. Tout d'abord, il y a un bloc bande morte (voir ci-dessous), qui permet d'arrêter le sifflement des vérins.

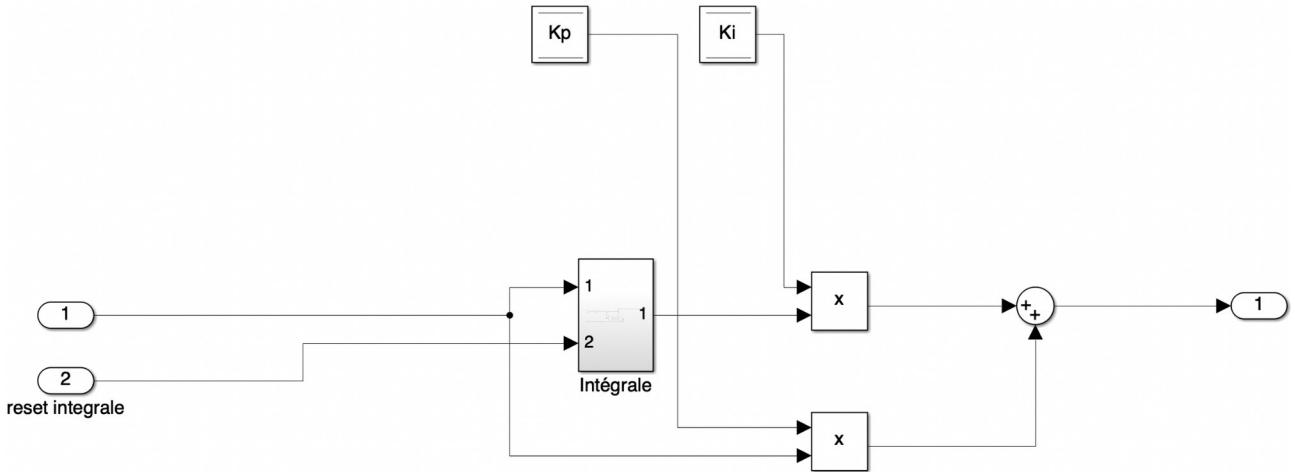


Ensuite nous avons un bloc de conversion (voir schéma ci-dessous) dans lequel nous retrouvons les mémoires de stockage de données **a** et **b** déclarées précédemment.

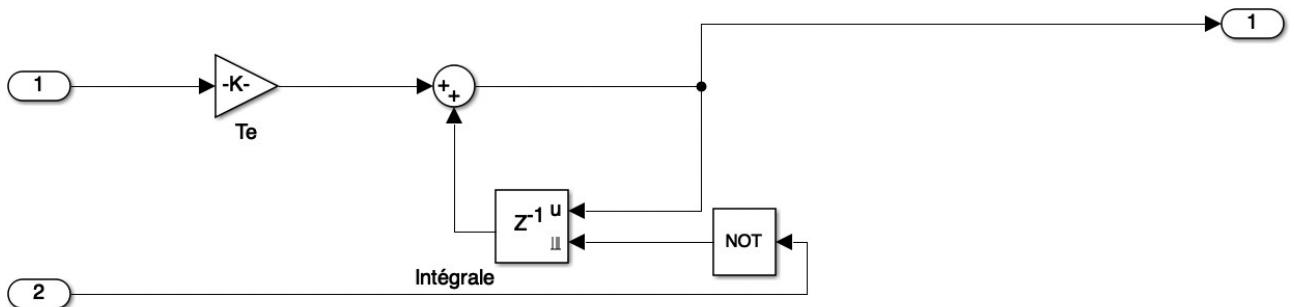


Bloc de conversion A0 -> Lr

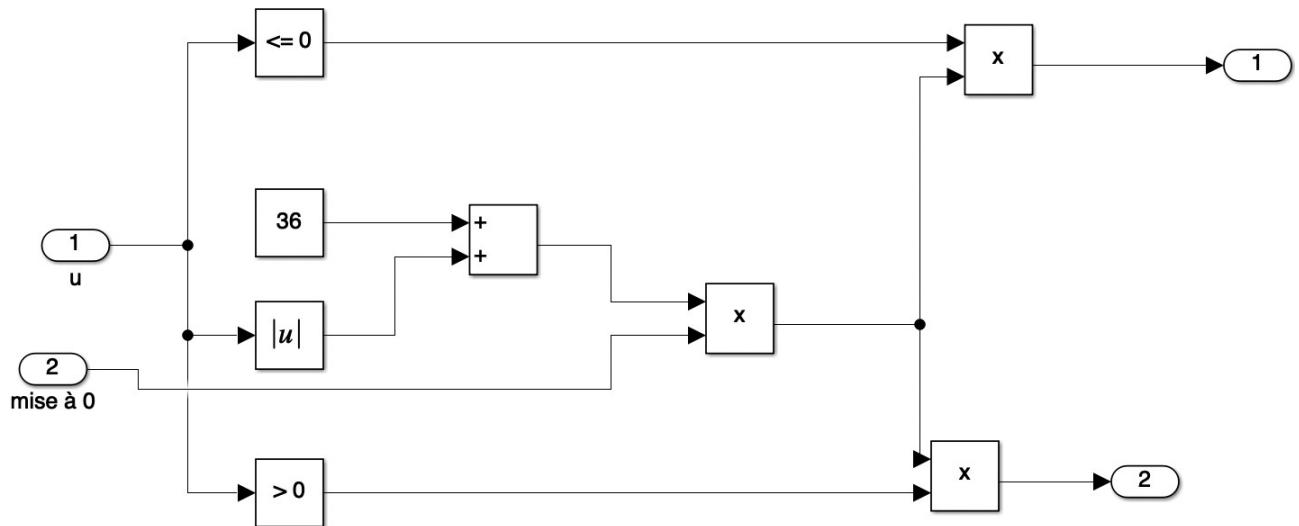
Nous avons également le bloc correcteur (voir ci-dessous) dans lequel nous retrouvons d'autres mémoires de stockage de données et un bloc d'intégrale.



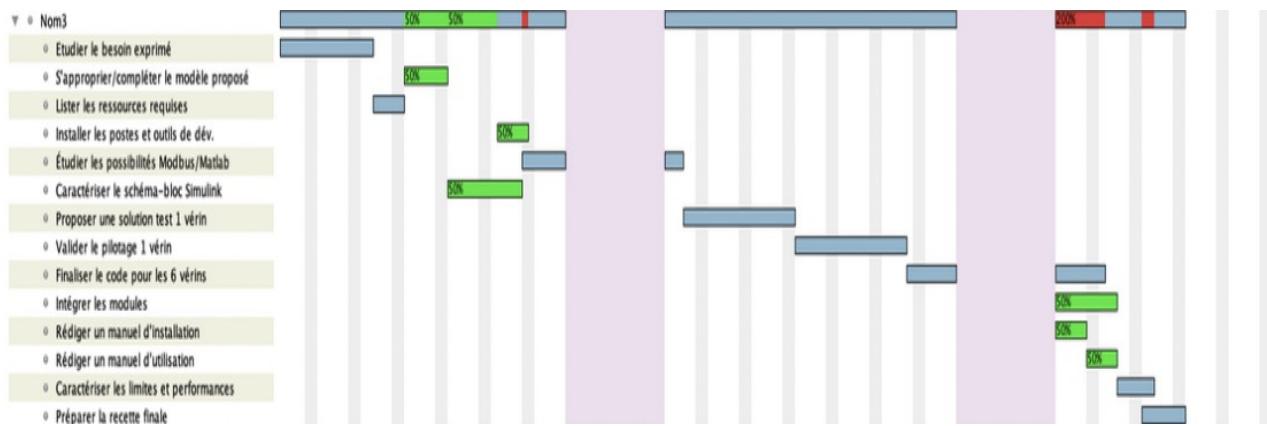
Dans le bloc Intégrale (voir ci-dessous), nous retrouvons un retard ainsi qu'une addition.



Nous avons aussi le bloc de loi de commande (voir ci-dessous) dans lequel nous retrouvons des additions et multiplications afin de réaliser la mise en mouvement des vérins.



IV. Gantt



V. Conclusion

Ce projet a été très bénéfique d'un point de vue personnel car cela m'a permis de mieux savoir utiliser Simulink et de découvrir de nouvelles fonctions. J'ai également appris à travailler en équipe afin de réaliser ce projet.

Grâce à ce projet, j'ai également pu approfondir mes connaissances sur le fonctionnement d'un hélicoptère, notamment sur les différentes commandes de celui-ci.

Etudiant 4 : Aurélien FERREIRA

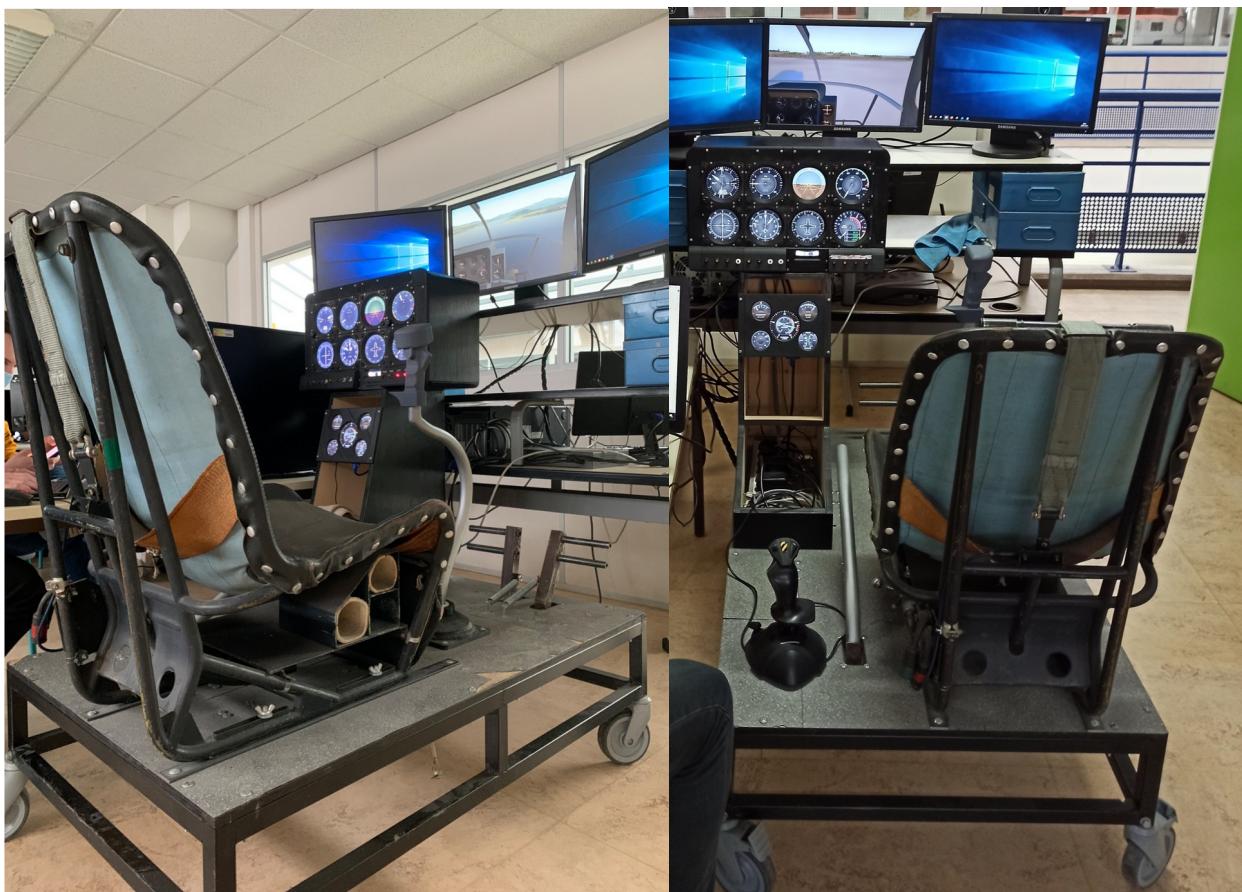
I. Cahier des charges

Dans ce projet, j'étais chargé de développer un modèle 3D de la maquette du lycée avec d'un côté les parties fixes (console d'instruments, siège, pilote...) et de l'autre, les parties mobiles (commandes de vol).

J'avais à disposition l'application “*FlightSimMockup*” donc l'hexapode virtuel. C'est un modèle 3D paramétrable d'hexapode développé en C++/Qt et basé sur la bibliothèque opensource Qam3D/GLam. L'hexapode virtuel peut être commandé en spécifiant la longueur des vérins ou en spécifiant son allure par des déplacements longitudinaux, transversaux et verticaux et des mouvements angulaires de roulis, tangage et lacet .

Celle-ci est composée d'une classe se nommant “*FSmuFrame*” qui représente le support mécano-soudé de la maquette (ce support est animé en fonction de l'allure de l'hexapode).

Puis ensuite il faut mettre au point des classes *FSmuFrame* de modélisation des autres éléments fixes de la maquette, soit essentiellement la console d'instruments et le siège du pilote.



Le développement de ces classes tout comme *FlightSimMockup* sont basé sur la

bibliothèque Qam3D/GLam version 0.8 ou supérieure. Cette bibliothèque regroupe un moteur d'animation 3D OpenGL-ES (shaders) et un ensemble de classes de modélisation de solides 3D de géométrie simple (cube, cylindre...) ou plus complexe (engrenage, ressort...). Le travail à donc débuté par une phase d'apprentissage de manière à s'approprier les ressources offertes par la bibliothèque Qam/GLam.

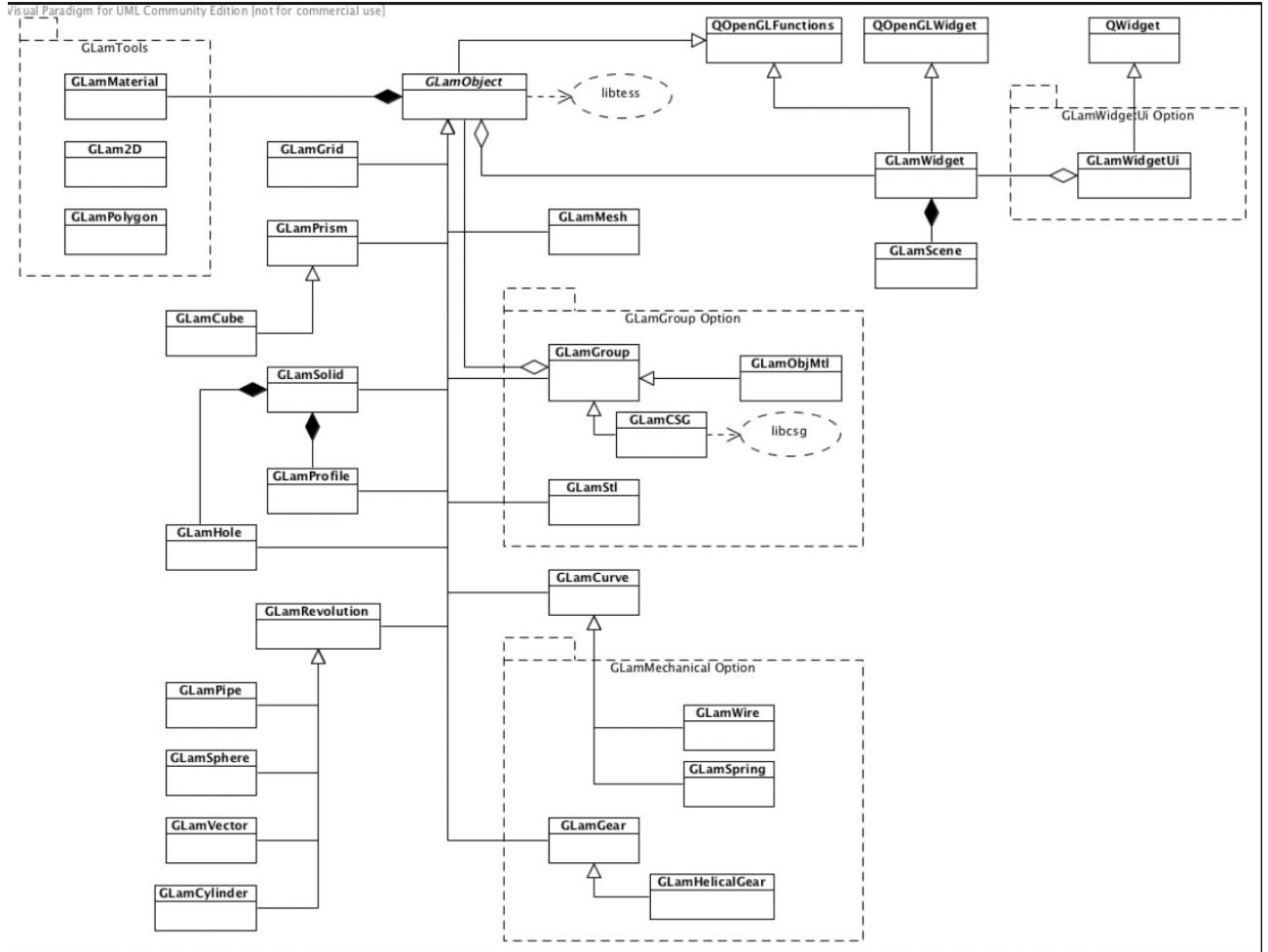
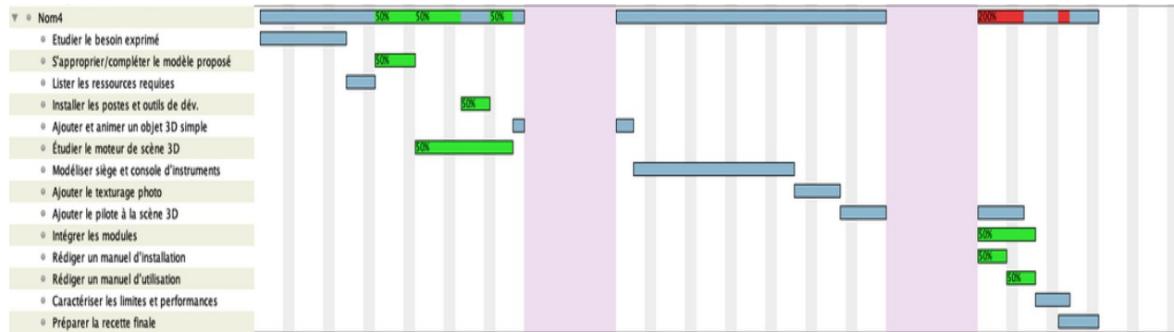


Diagramme de classe GLamTester

II. Mes différentes tâches

Voici ci-dessous l'ensemble de toutes mes tâches représentées sur un diagramme de Gantt.

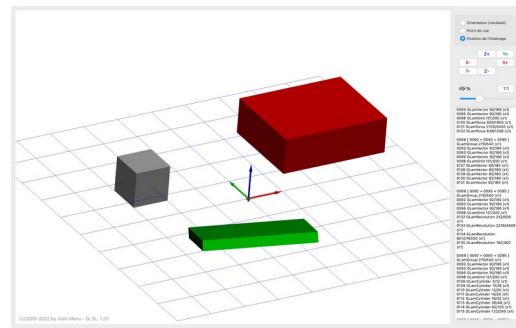


J'ai tout d'abord regardé comment je pouvais m'y prendre en regardant le siège et ses consoles d'instruments en physique. Puis j'ai récupéré et téléchargé toutes les ressources nécessaires comme par exemple la bibliothèque Glam puis j'ai essayé de m'approprier ces ressources pour après créer un simple cube en 3D.

A. Exemple d'un simple cube

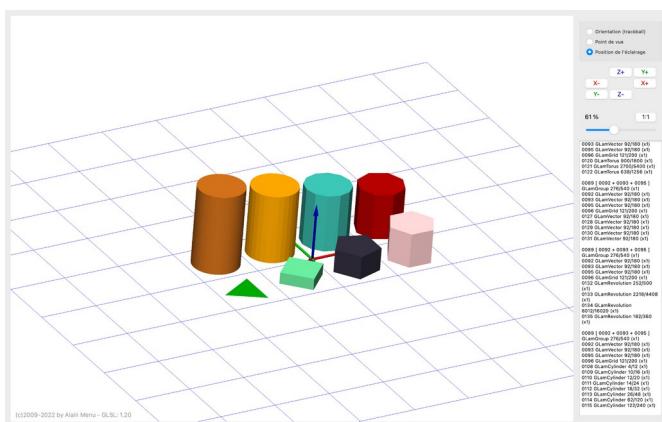
Mon professeur d'informatique m'a mis à disposition une application sur le framework QT se nommant "GLamTester" pour pouvoir tester la librairie GLam. Cela m'a beaucoup aidé à créer différents objets comme un cube ci-dessous.

```
01
62 // construction
63     m_cube[0] = new GLamCube ;
64     m_cube[0]->setZCentered() ;
65     m_cube[0]->defaultMatrix().translate(-1, 1 ) ;
66
67     m_cube[1] = new GLamCube(1.5, 1.2, 0.5 ) ;
68     m_cube[1]->defaultMatrix().translate( 1.5, 1 ) ;
69     m_cube[1]->material().setColor(0.8, 0.1, 0.0 ) ;
70
71     m_cube[2] = new GLamCube( *m_cube[1] ) ;
72     m_cube[2]->setThickness(0.02) ;
73     m_cube[2]->defaultMatrix().setToIdentity() ;
74     m_cube[2]->defaultMatrix().translate( -0.5, -1 ) ;
75     m_cube[2]->defaultMatrix().rotate(-30, 1, 0, 1 ) ;
76     m_cube[2]->material().setColor(0.0, 0.8, 0.2 ) ;
77
78
```



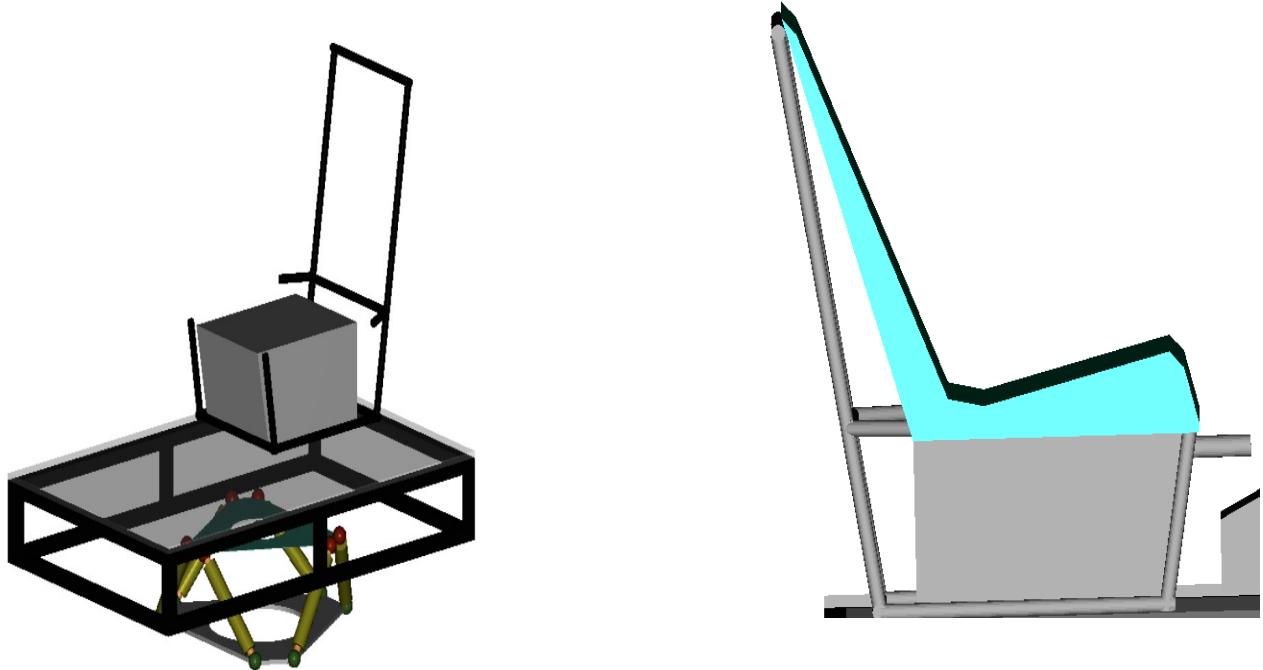
Les lignes de code pour la construction d'un cube à comme caractéristique (longueur, largeur, hauteur, l'emplacement et la couleur).

Mais bien sûr il est possible de faire plusieurs formes ou différents objets en 3D que j'ai utilisés pour faire le siège comme des cylindres, des sphères, des formes avec des vecteurs...



B. La partie du siège

En première partie il a été important de commencer par le siège et son squelette pour avoir une bonne base dès le début. Pour modéliser le cube en dessous du siège, la classe que j'ai utilisée est celle de GLamSolid. Puis j'ai créé en tout, 12 tubes de différentes tailles et orientés différemment pour pouvoir y ajouter le dossier et l'assise du siège. Et c'est la classe GLamCylinder que j'ai appliquée.



Voici quelques lignes de code pour la modélisation du siège, on voit les deux première ligne (shape QPoint) qui sert à modéliser un objet en 3D en déclarant une suite de points qui permet de tracer la forme du siège

```
// Back
shape.clear();
shape <<QPoint( 100,220 ) <<QPoint ( 500,220 ) <<QPoint ( 500,230 ) <<QPoint ( 480,300 ) <<QPoint(460,320)
    <<QPoint(200,260) <<QPoint(150,270) <<QPoint(-50,720) <<QPoint(-70,740) ;

m_back = new GLamSolid (shape,470) ;
m_back->defaultMatrix().rotate(90, 1, 0, 0) ;

m_back->setMaterial( GLamMaterial::CyanPlastic );

this->addObject( m_back ) ;
```

Puis il est possible de changer la couleur de l'objet en ajoutant la classe “GLamMaterial” avec la couleur qui suit donc pour ce cas là, le cyan.

C. La partie de la console

Voici ensuite la prochaine étape réalisée, la console centrale. La classe utilisée est la même précédemment, GLamSolid. Il y a plusieurs objets 3D qui créent l'ensemble de la console avec des différents cubes modélisés de différentes manières.

On peut retrouver comme avant la base de code avec différents points pour créer le socle de la console.

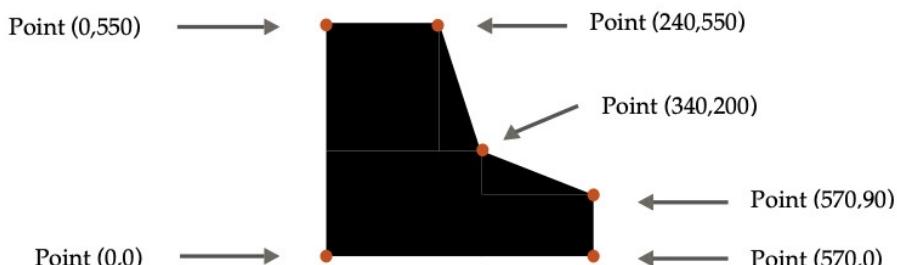
Puis ensuite il fallait ajouter l'épaisseur sur la troisième ligne et qui est égale à 220 mm donc 22 cm.

```
// Socle
shape <<QPoint( 0,0 ) <<QPoint ( 570,0 ) <<QPoint ( 570,90 )
<<QPoint(340,200) <<QPoint( 240,550) <<QPoint(0,550) ;

m_socle = new GLamSolid (shape,220) ;
m_socle->defaultMatrix().rotate(90, 1, 0, 0) ;
m_socle->defaultMatrix().rotate(180, 0, 1, 0) ;

//m_socle->setMaterial( GLamMaterial::Obsidian ) ;

this->addObject( m_socle ) ;
```



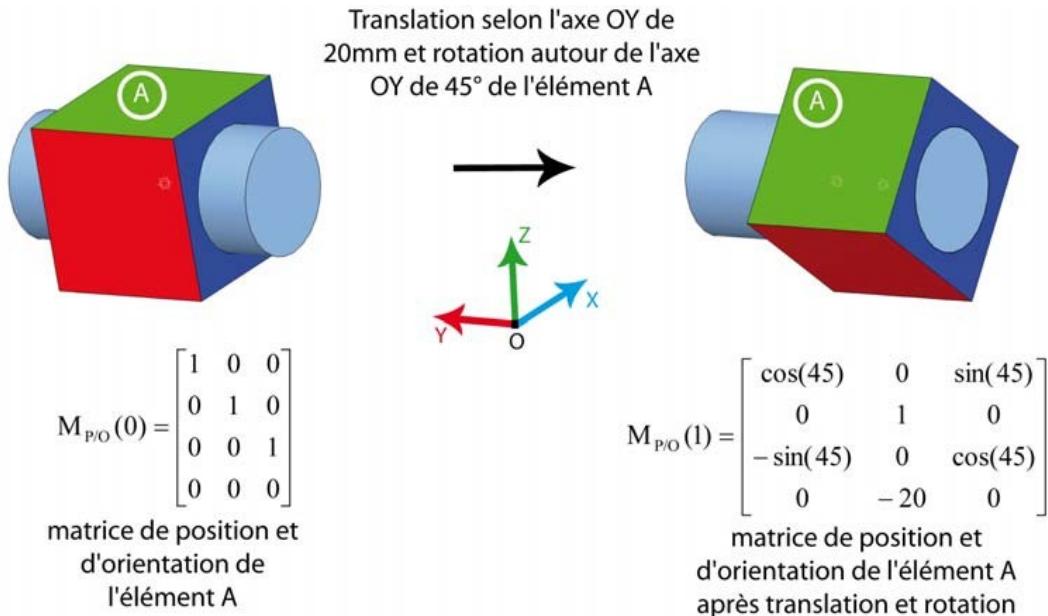
Et pour placer nos objets sur le socle posé sur les vérins, il a fallut utiliser la matrice de translation et de rotation.

Une translation: consiste à faire "glisser" un ensemble de points sur un "rail" (un vecteur).

Et voilà une matrice de translation :

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Une rotation: consiste à faire pivoter un ensemble de points d'un angle Thêta par rapport à un point.



D. La partie des commandes de vol

Tout d'abord il fallait comprendre le fonctionnement des commandes et à quoi elles servaient. Il y a au total trois commandes différentes sur le simulateur.

La première qui est le manche de variation du pas cyclique, qui permet de varier l'inclinaison des pales pour faire monter ou descendre l'hélicoptère.

Puis la deuxième qui est le levier de variation du pas collectif qui permet de varier l'inclinaison du rotor principal.

Et pour terminer, le palonnier qui lui permet de varier l'inclinaison des pales du rotor arrière.

Il y a trois différentes classes qui constituent ces commandes, FSmuHandle, FSmuRudder, FSmuCollective.

Les classes GLamSolid et GLamCylinder, vu précédemment ont été utilisées pour faire la commande du palonnier et du levier.

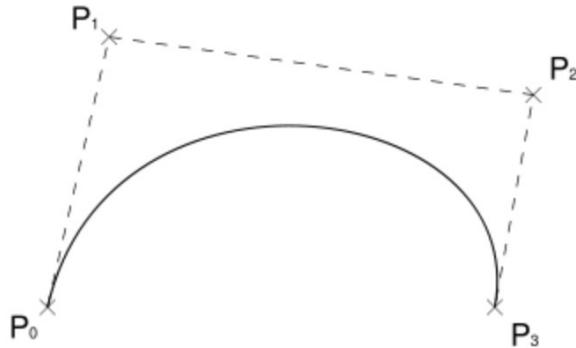
Mais pour faire la commande du manche, j'ai utilisé une nouvelle classe qui est GLamWire.

Cette classe a pour particularité d'utiliser les courbes de Bézier pour déformer les tubes.

Courbe de Bézier : Les courbes de Bézier sont des courbes polynomiales paramétriques décrites pour la première fois en 1962 par l'ingénieur français Pierre Bézier qui les utilisa Hexapod

pour concevoir des pièces d'automobiles à l'aide d'ordinateurs. Elles ont de nombreuses applications dans la synthèse d'images et le rendu de fontes. Elles ont donné naissance à de nombreux autres objets mathématiques.

Technique : Quatre points P_0 , P_1 , P_2 et P_3 définissent une courbe de Bézier cubique. La courbe se trace en partant du point P_0 , en se dirigeant vers P_1 et en arrivant au point P_3 selon la direction P_2-P_3 . En général, la courbe ne passe ni par P_1 ni par P_2 : ces points sont simplement là pour donner une information de direction. La distance entre P_0 et P_1 détermine la " longueur " du déplacement dans la direction de P_1 avant de tourner vers P_3 .



```

FSmuHandle::FSmuHandle()
: GLamGroup()
, m_diameter( 27 )
{
    qreal E = 370 ;
    qreal e = 230 ;
    qreal a = -20 ;

    QVector3D p0( 0, 0, 0 ) ;
    QVector3D v0( a, 0, 0 ) ;
    QVector3D p1( 0, 0, E ) ;
    QVector3D v1( e, 0, 0 ) ;
    m_wire = new GLamWire(BezierCurve(p0,v0,p1,v1), m_diameter ) ;
    m_wire->defaultMatrix().translate(0,0,100) ;
    //m_wire->material().setColor(Qt::red) ;
    this->addObject( m_wire ) ;
}

```

Voici les quatres lignes qui ont servi de déformer le tube de façon à arrondir le manche et qu'il se rapproche de l'original en physique. Cela fonctionne avec la courbe de Bézier évidemment.

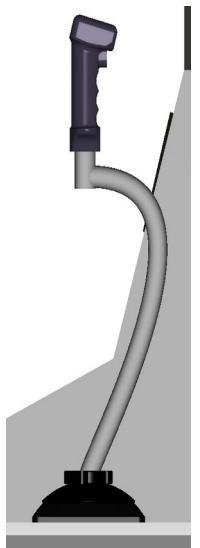
Ensuite pour créer la poignée du dessus du manche, j'ai juste importé un fichier type ".stl". Car le manche de la maquette avait été déjà fabriqué auparavant par une imprimante 3D.

```

m_stlImport4 = new GLamStL(":Users/Aurélien/Documents/BTS/2emeannee/Informatique/Projet/Manche-stl.stl") ;
m_stlImport4->defaultMatrix().scale(1) ;
m_stlImport4->defaultMatrix().translate(-30,40,500) ;
m_stlImport4->defaultMatrix().rotate(-90, 0, 0, 1) ;
m_stlImport4->material().set(GLamMaterial::Obsidian) ;

GLamMesh mesh4( m_stlImport4->toTrimesh() ) ;
this->addObject( &mesh4 ) ;

```



E. Texturage photo

J'ai utilisé une nouvelle classe, GLamPhoto qui a été important pour ajouter des textures. Comme par exemple les instruments de vol ci-dessous. Le principe est simple, il suffit de déposer une image sur une partie de l'objet choisi de type GLamCube et de donner une texture au cube.

```
// Texture Instrumentation 2
```

```

m_texture2 = new GLamPhoto(220,150) ;
m_texture2->defaultMatrix().rotate(90, 0, 0, 1) ;
m_texture2->defaultMatrix().rotate(-75, 1, 0, 0) ;
//m_texture->defaultMatrix().rotate(90, 1, 0, 0) ;
m_texture2->transformMatrix().translate( 110,-340, 370) ;

m_texture2->setTexture(":Users/Aurélien/Documents/BTS/2emeannee/Informatique/Projet/Image/photo2-img.jpg")
//m_texture->defaultMatrix().scale(1) ;

```



Conclusion

Ce projet m'a appris certaines choses d'une part de mon travail personnel et du travail en équipe, il a fallu respecter les délais de nos tâches respectives. Et du côté personnel j'ai dû apprendre certaines connaissances et surtout appliquer les connaissances que j'ai apprises tout au cours de l'année comme par exemple en programmation C++.