Rapport de projet

Simulateur de vol

Etudiant 4 : Chammami Ismaïl

Académies : Besançon-Dijon | Session : 2019

Lycée ou Centre de formation : Lycée Gustave Eiffel

Ville : Dijon

Nom du projet : Simulateur de vol : plateforme de réalité virtuelle

Projet industriel : ~~OUI~~ NON

Équipe de développement.

Professeur : Joël Moutoussamy

Etudiant 1 : Mehdi Mokrane

Etudiant 2 : Julien Cordier

Etudiant 3 : Yvan Tran

Etudiant 4 : Chammami Ismaïl

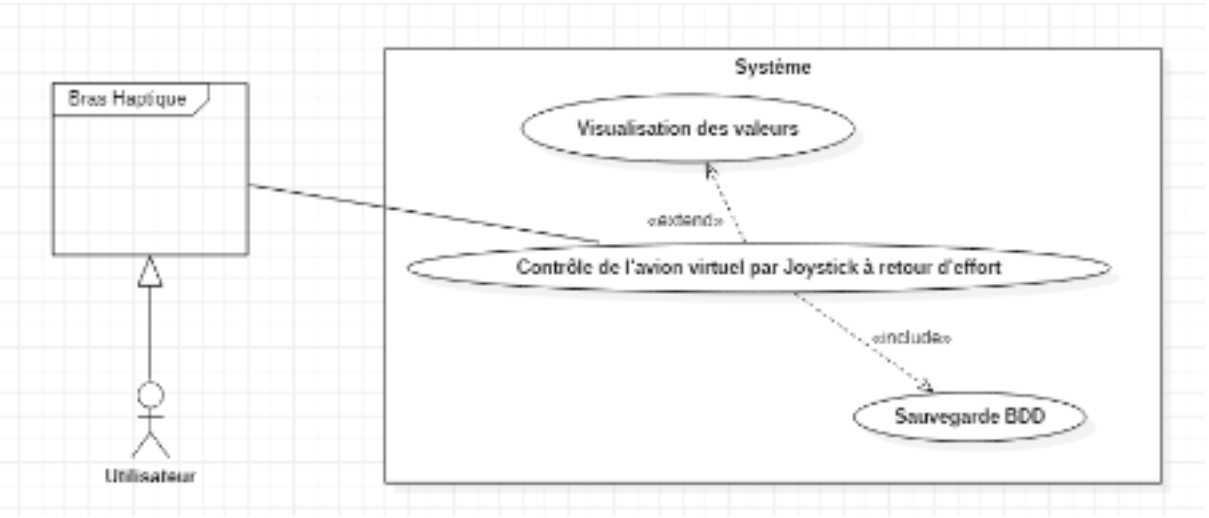
Lunettes immersives 3D

Bras haptique

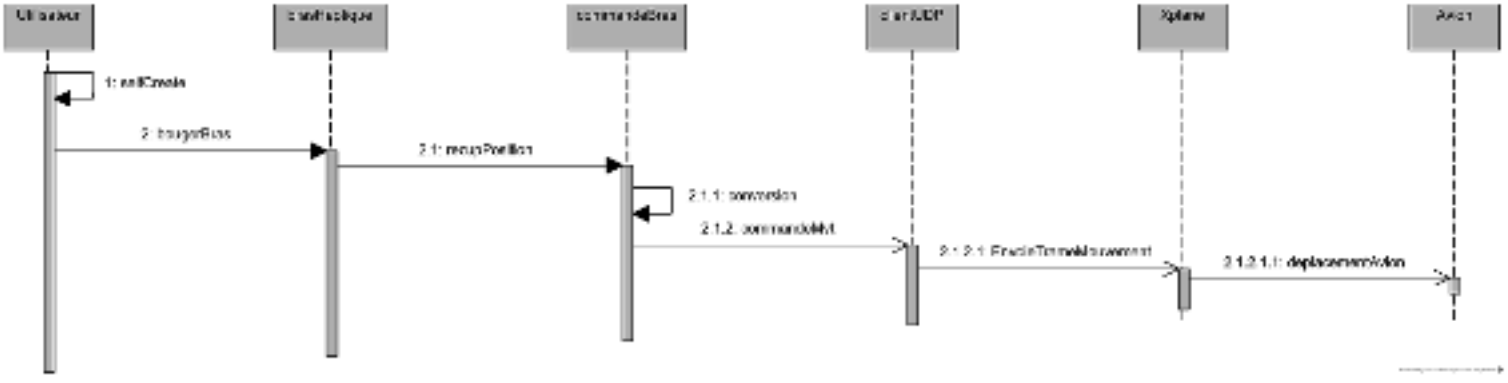
Environnement virtuel

Présentation du projet

Pour ma part je devais m'occuper de programmer l'application de pilotage de l'avion à l'aide du joystick à retour d'effort et de la sauvegarde chronologique des données du bras sur la base de données.



Ainsi l'application permettrait de piloter l'avion à l'aide du joystick à retour d'effort. Les positions (x,y) du bras sont transformées en angles roll/pitch et envoyés au serveur UDP de XPlane. Ces données sont sauvegardées périodiquement dans une base de données, pouvant ainsi être affiché sur un tableau de bord en ligne.

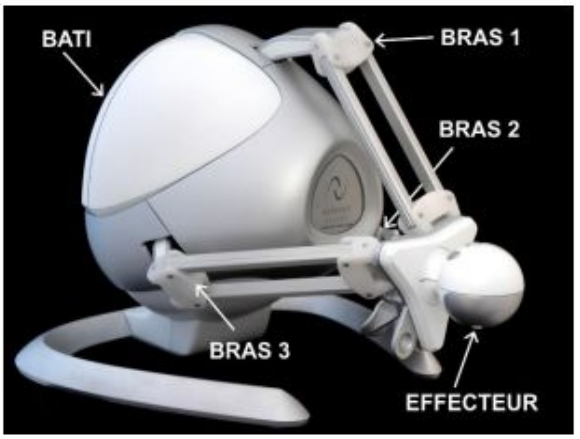


Contraintes et ressources utilisées

Selon le cahier des charges les contraintes logicielles et matérielles étaient les suivantes :

- Langage de programmation : C++
- Environnement de développement : VisualStudio 2015, QT Creator
- Système de gestion de base de données : InfluxDB
- Bras Haptique : Novint Falcon
- Système d'exploitation : Windows 10

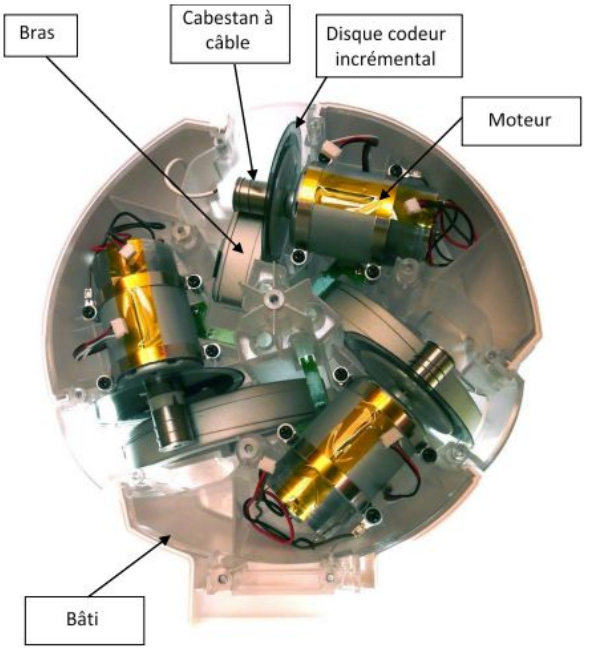
Présentation du bras haptique



Le joystick à retour d'effort est un Novint Falcon, contrôleur de jeu USB de la marque Novint Technologies. Il est doté d'un grip à 4 boutons, 3 bras reliés à une base conique reposant sur un pied en forme de U.

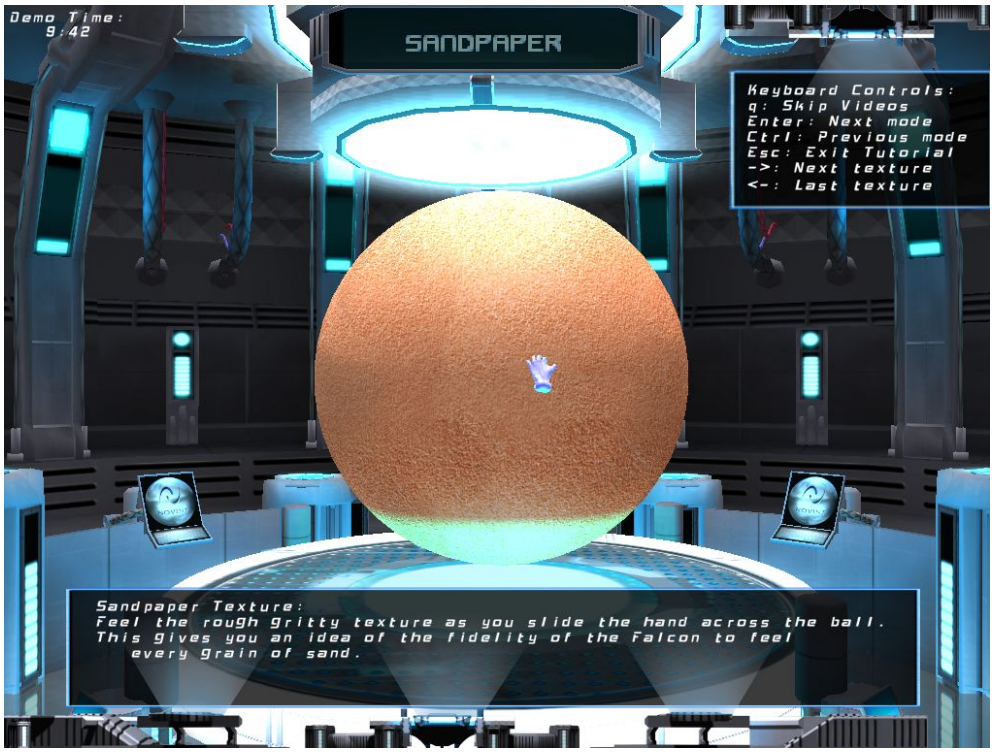
Un moteur entraîne chacun des bras via un réducteur à cabestan. De plus, l'axe de chaque moteur est équipé d'un disque gradué nécessaire au traitement d'un codeur incrémental.

Cela permet à l'aide d'une fonction elliptique de Jacobi de calculer la position d'un curseur tridimensionnel basé sur la position des bras, qui est ainsi déplacé par l'effecteur et permet au Falcon d'appliquer des forces sur l'utilisateur en réponse à l'environnement virtuel.



Prise en main et test du bras haptique

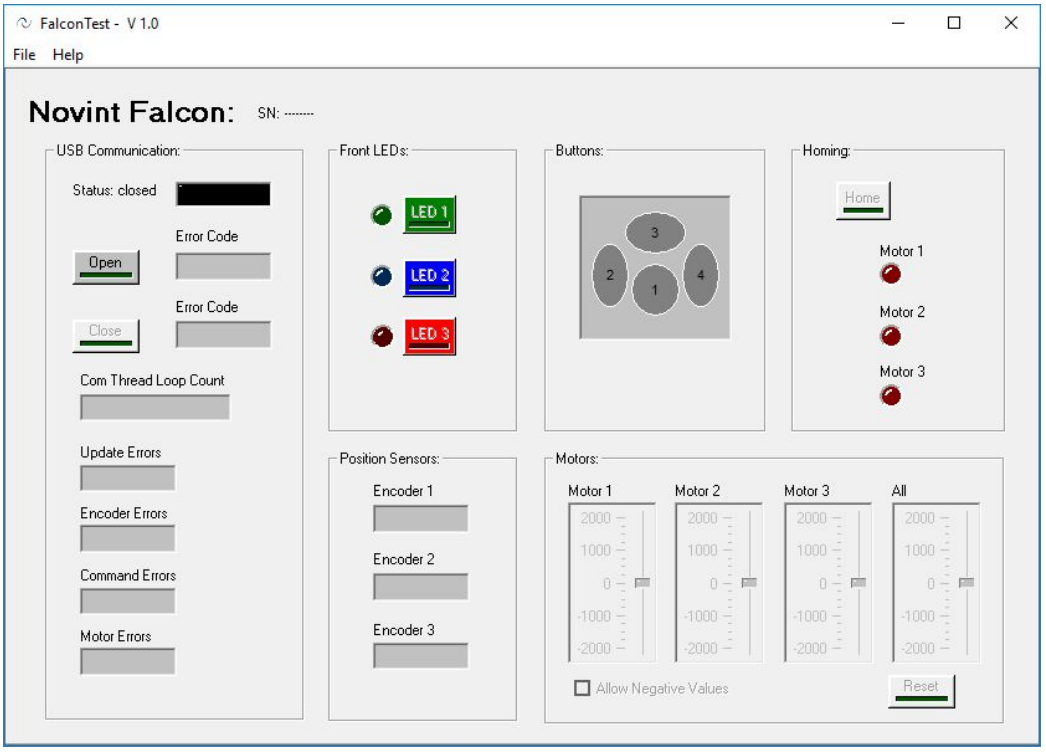
Tutoriel Novint Falcon



Sur internet on peut trouver le driver du Novint Falcon, contenant un logiciel de tutoriel d'utilisation et d'application du bras haptique. Ce tutoriel contient plusieurs mini-jeux :

- le premier démontre la possibilité de ressentir différentes surfaces à l'aide du grip : lisse, rugueux, collant, glissant, cahoteux etc.
- le second permet de ressentir des sensations dynamiques à l'aide d'une balle accrochée à un élastique dont on peut ressentir le point et l'inertie en la déplaçant.
- le troisième consiste à attraper des balles ce qui permet de ressentir la force de poussée.
- le quatrième consiste à tirer des billes avec un lance pierre ce qui permet de ressentir la résistance de l'élastique lors de sa flexion.

Logiciel de monitoring



Il contient également un logiciel de monitoring du bras permettant de tester le bon fonctionnement de celui ci :

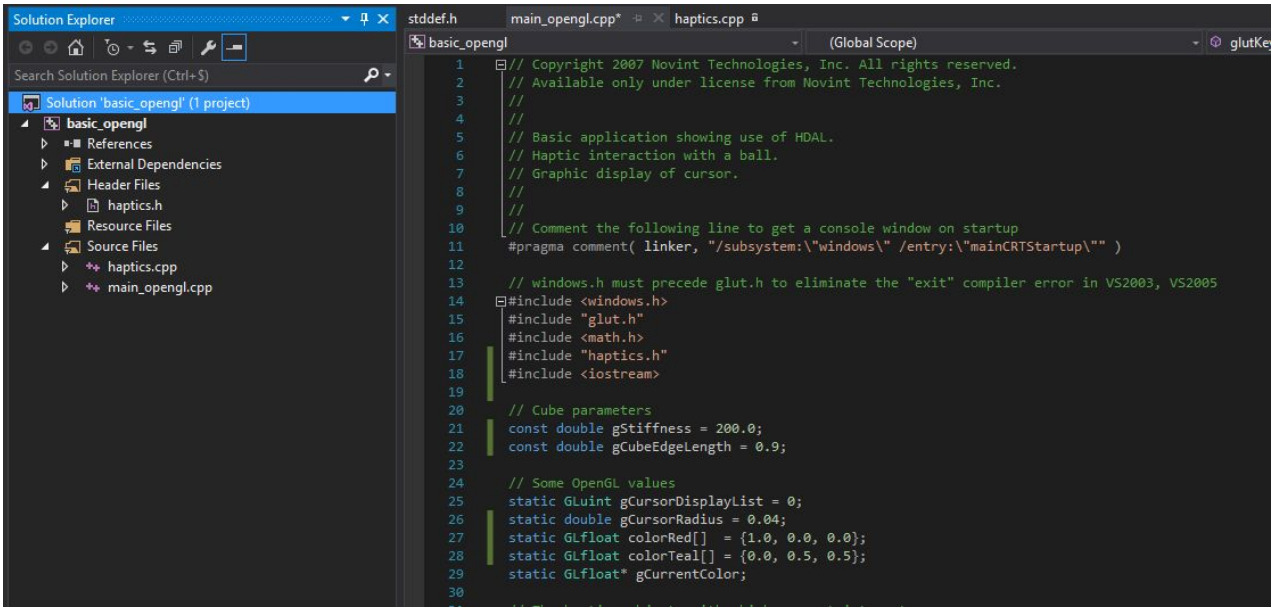
- on peut tester les différentes lumières et les différents boutons et moteurs
- on peut voir les valeurs des encodeurs de chaque bras

HDAL SDK 3.0

Afin de programmer l'application récupérant les positions x et y du grip j'ai utilisé le Kit de développement HDAL 3.0. Ce kit de développement utilise la librairie graphique OpenGL permettant d'afficher l'environnement 3D et le curseur dans celui-ci.

.vs	03/02/2020 16:50	Dossier de fichiers	
Backup	08/11/2019 11:10	Dossier de fichiers	
Debug	04/02/2020 15:45	Dossier de fichiers	
basic_opengl.sln	08/11/2019 14:45	Solution Visual St...	1 Ko
basic_opengl.VC.db	03/02/2020 16:51	Data Base File	220 Ko
basic_opengl.VC.VC.opendb	03/02/2020 16:51	Fichier OPENDB	0 Ko
basic_opengl.vcproj	09/02/2009 16:39	VC++ Project	7 Ko
basic_opengl.vcxproj	04/02/2020 15:15	VC++ Project	8 Ko
basic_opengl.vcxproj.filters	08/11/2019 11:10	VC++ Project Filte...	2 Ko
basic_opengl.vcxproj.user	08/11/2019 11:21	Fichier d'options ...	1 Ko
dhd_messages.log	04/02/2020 15:45	Document texte	3 Ko
dhdIcLog.log	04/02/2020 15:45	Document texte	1 Ko
hdal_messages.log	04/02/2020 15:45	Document texte	1 Ko
UpgradeLog.htm	08/11/2019 11:10	Chrome HTML Do...	41 Ko

Pour la prise en main du SDK j'ai donc utilisé un template de programme sur Visual Studio.



Le programme utilise la classe “haptics.h” qui prend en charge le Novint Falcon. Afin de prendre en main la classe, mon but était d’obtenir et afficher sur la console les positions x, y et z du curseur lorsqu’on appuie sur un des 4 boutons du grip, mais également pouvoir appliquer des forces sur les différents axes.

Pour obtenir les positions le programme utilise une fonction callback (fonction de rappel). Une fonction de rappel est une fonction qui est passée en argument à une autre fonction. Ainsi, cette dernière peut faire l’usage de cette fonction sans la connaître par avance. Cela permet d’obtenir les données du bras au rythme maximum possible.

Fonction appelante exécutée à chaque déplacement du grip

```
// Set up callback function
m_servoOp = hdlCreateServoOp(ContactCB, this, bNonBlocking);
if (m_servoOp == HDL_INVALID_HANDLE)
{
    MessageBox(NULL, "Invalid servo op handle", "Device Failure", MB_OK);
}
testHDLError("hdlCreateServoOp");
```

Fonction appelée

```
// Continuous servo callback function
HDLServoOpExitCode ContactCB(void* pUserData)
{
        // Get pointer to haptics object
        HapticsClass* haptics = static_cast<HapticsClass*>(pUserData);
```

Ainsi à chaque déplacement du grip, la fonction “hdlCreateServoOp” appelle la fonction callback “ContactCB”, qui récupère ainsi les données du bras.

Obtention et affichage des valeurs de position

```
// Continuous servo callback function
HDLServoOpExitCode ContactCB(void* pUserData)
{
        // Get pointer to haptics object
        HapticsClass* haptics = static_cast<HapticsClass*>(pUserData);

        // Get current state of haptic device
        hdlToolPosition(haptics->m_positionServo);

        float posx = (haptics->m_positionServo[0]) * 1000;
        float posy = (haptics->m_positionServo[1]) * 1000;
        float posz = (haptics->m_positionServo[2]) * 1000;

        bool clic = haptics->m_buttonServo;

        print_position(posx, posy, posz, clic);
```

On peut récupérer les positions x, y et z du grip à l'aide de la fonction "hdlToolPosition" dans le tableau de 3 float "m_positionServo" (0 pour x, 1 pour y et 2 pour z).

```
// Get current state of haptic device
hdlToolPosition(haptics->m_positionServo);
```

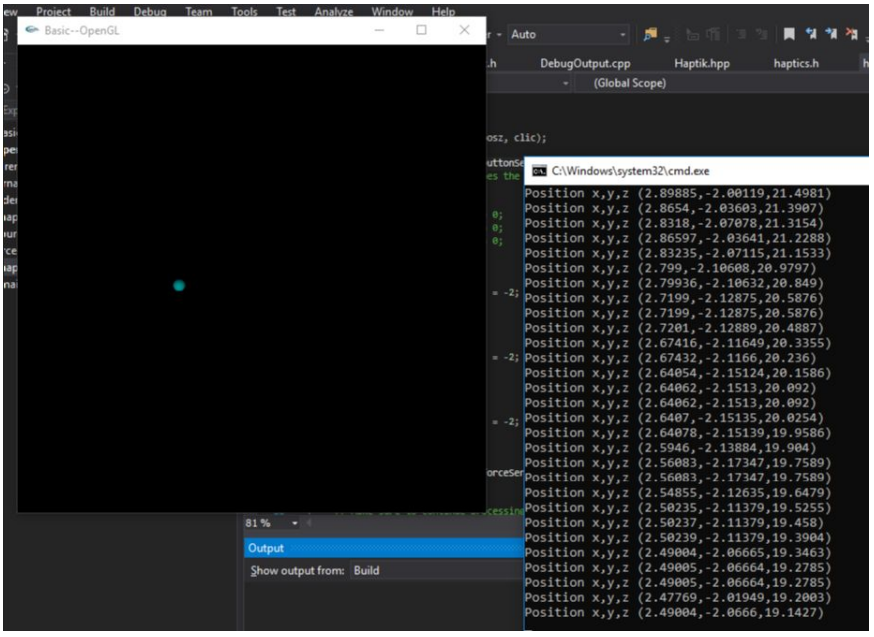
Les valeurs de position comprises entre -50 et +60 sont alors stockées dans les variable posx, posy et posz.

```
float posx = (haptics->m_positionServo[0]) * 1000;
float posy = (haptics->m_positionServo[1]) * 1000;
float posz = (haptics->m_positionServo[2]) * 1000;
```

Afin d'afficher la position lors du clic d'un des 4 boutons, on récupère la valeur 0 ou 1 (0 : pas de clic / 1 : clic) à l'aide de la méthode "m_buttonServo" que l'on stocke dans un booléen clic. Ces variables sont alors passées en paramètre à la fonction "print_position".

```
void print_position(float x, float y, float z, bool condition)
{
        if (condition)
        {
                cout << "Position x,y,z (" << x << "," << y << "," << z << ")" << endl;
        }
}
```

Résultat



Ainsi lorsque l'on clique sur un bouton du bras, les valeurs de positions sont affichées dans la console.

Application de force sur les axes du bras haptique

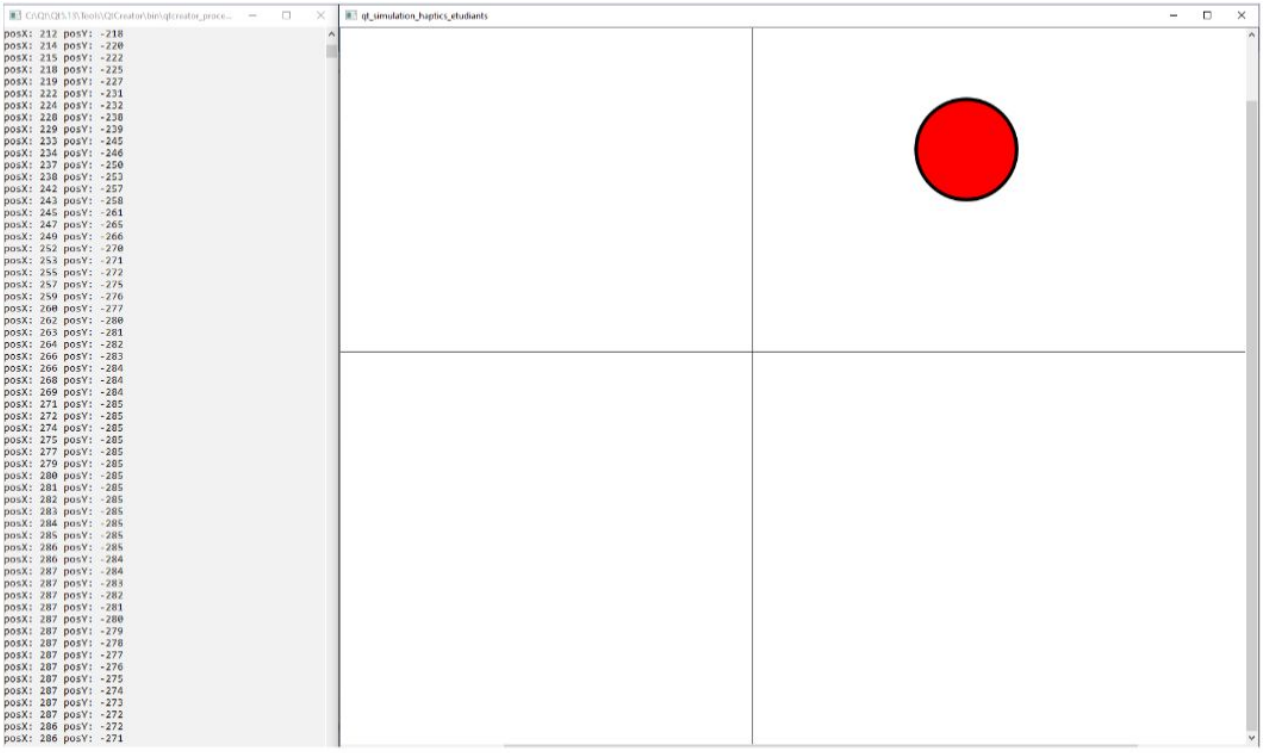
```
if (posx <= 10)
{
        haptics->m_forceServo[0] = -2;
}

if (posy <= 10)
{
        haptics->m_forceServo[1] = -2;
}

if (posz >= 0)
{
        haptics->m_forceServo[2] = -2;
}
```

Afin d'appliquer des forces sur les axes, on attribue une valeur (positive pour une force inverse au sens du mouvement, négative pour une force dans le sens du mouvement) à m_forceServo[x] (0 pour l'axe x, 1 pour l'axe y, 2 pour l'axe z). Dans cette capture d'écran, les valeurs sont attribués avec des conditions de position du curseur, ce qui permet de créer des zones de résistance positive ou négative.
Ces forces sont alors appliquées à l'aide de la fonction "hdlSetToolForce".

```
    // Send forces to device
hdlSetToolForce(haptics->m_forceServo);
```

Contraintes du confinement

En raison du confinement, il y avait des contraintes matérielles. Ne pouvant plus utiliser le bras haptique j'ai donc dû utiliser un simulateur de joystick fourni par le professeur.



Ce simulateur de bras haptique est un projet QT Creator qui contient :

- la classe CGrip simulant le grip
- la classe CBrasHaptique qui est composée de CGrip, simulant le bras haptique
- la classe principale ClientBrasHaptique qui est composée de CBrasHaptique, permettant d'afficher et manipuler les valeurs du bras

Comparaison avec le réel

Le système SIGNAL/SLOTS de Qt s'apparente au principe d'une fonction callback.

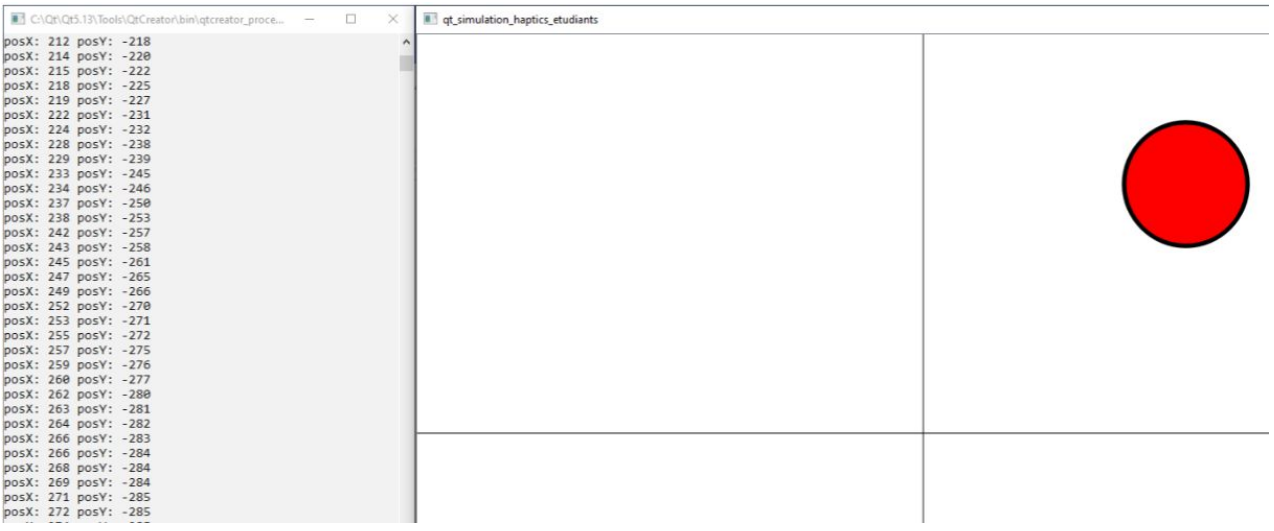
```
signals:
    //Permet de récupérer la position du grip lors
    //du déplacement de la souris
    void posGripChanged(QPoint p_posGrip);
```

De façon similaire à la fonction callback du programme Visual Studio expliquée plus haut, lorsque le grip virtuel se déplace le signal "posGripChanged" est émis avec la transmission de la position. Le SLOT printPosGrip récupère cette position.

```
QObject::connect(bras,&CBrasHaptique::posGripChanged,this,&ClientBrasHaptique::printPosGrip);
```

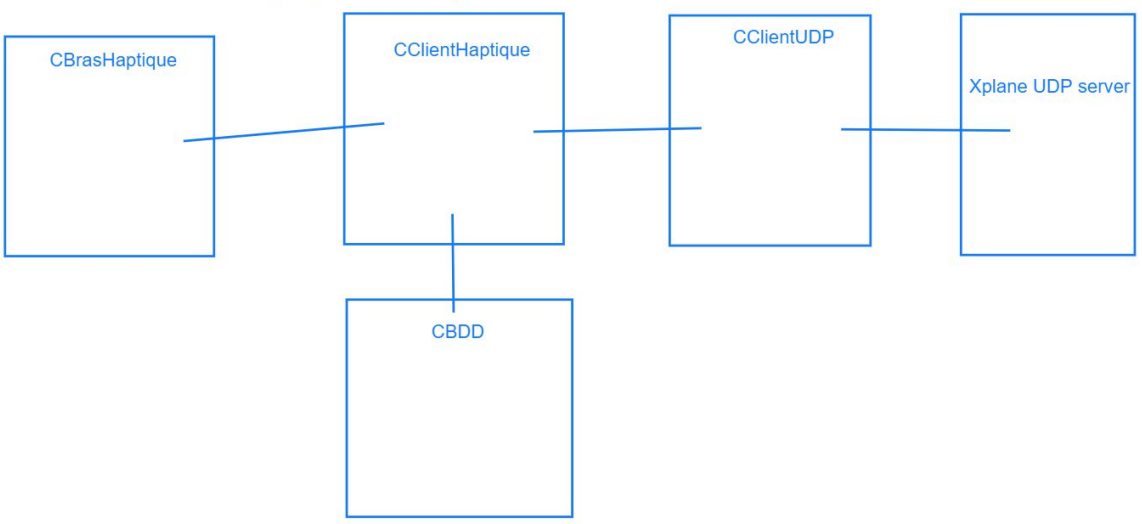
Ainsi on peut afficher la position du grip lorsque celui ci est déplacé :

```
void ClientBrasHaptique::printPosGrip(QPoint p_posGrip)
{
    qDebug()<<"posX:"<<p_posGrip.x()<<"posY:"<<p_posGrip.y();
}
```



Conception préliminaire

Avec les nouvelles contraintes du confinement, le but était donc de contrôler l'avion sur XPlane avec le simulateur de bras haptique, toujours en UDP.

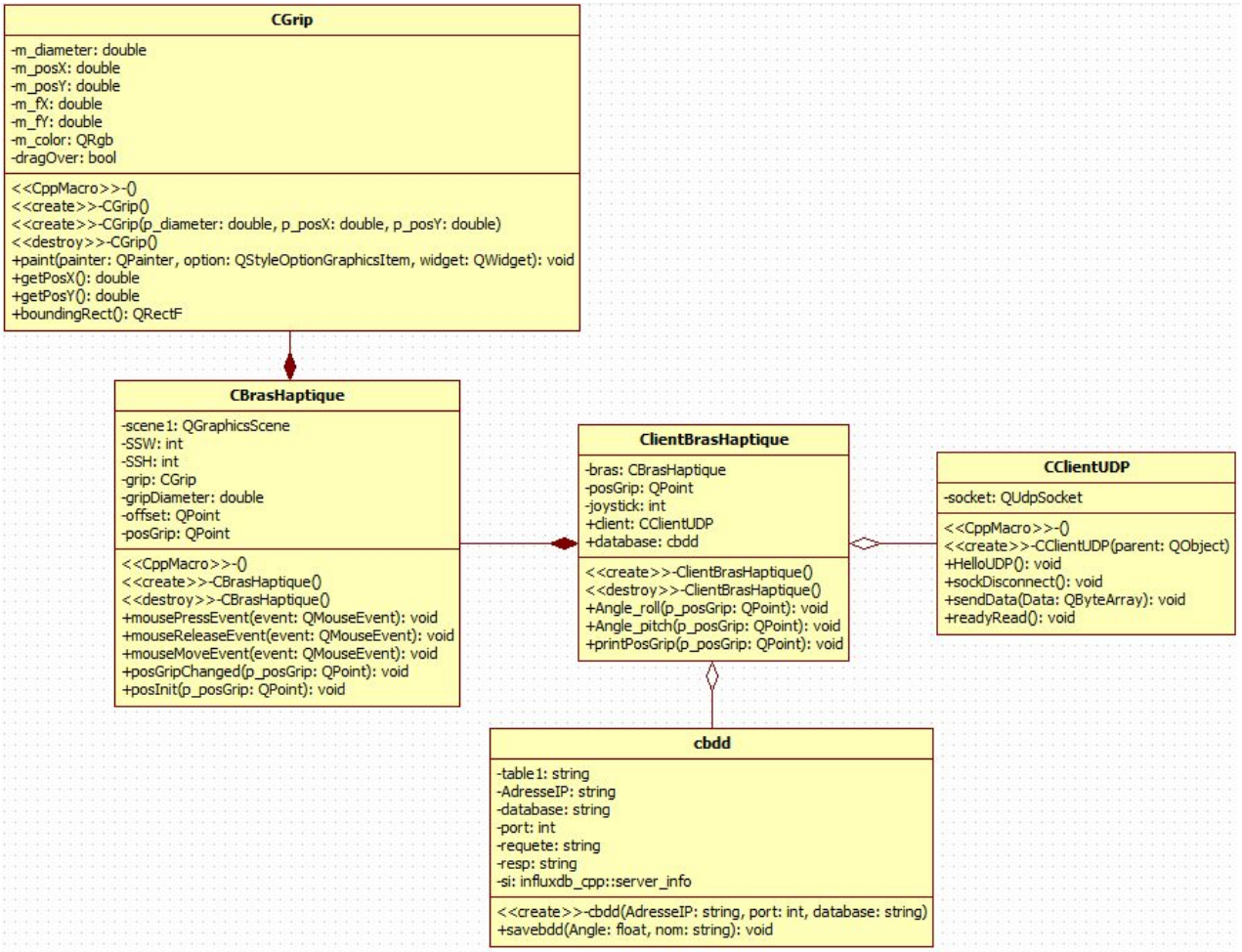


La classe CClientHaptique récupère les données de position du simulateur de bras haptique, les transformes en angle roll/pitch, sauvegarde ceux-ci dans la base de données InfluxDB, et les envoies sur le serveur UDP d'XPlane.
On a donc :

- la classe CBrasHaptique pour le simulateur de bras haptique
- la classe CBDD pour la base de données InfluxDB
- la classe CClientUDP pour les envois UDP
- la classe principale CClientHaptique qui gère la récupération des données du bras, la création des angles et la sauvegarde et l'envoi de ceux ci

Conception détaillée

Diagramme de classes



Le programme final contient donc :

- La classe principale ClientBrasHaptique composée de CBrasHaptique elle même composée de CGrip
- Les classe CClientUDP et CBDD qui appartiennent à ClientBrasHaptique

Le programme

La classe principale

Le programme initialise un unique objet bras.

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    ClientBrasHaptique *bras=new ClientBrasHaptique;

    return a.exec();
}
```

La classe ClientBrasHaptique contient les objets “CBrasHaptique bras“, “cbdd database“ et “CClientUDP client“.

```
class ClientBrasHaptique : public QWidget
{

    private:
        //Un bras haptique est contenu
        CBrasHaptique *bras;
        //client UDP
        CClientUDP client;
        //influxdb
        cbdd *database;
```

Ces objets sont initialisés dans le constructeur avec la connection des SLOT de calcul des angles.

```
ClientBrasHaptique::ClientBrasHaptique()
{
    bras=new CBrasHaptique;
    bras->show();

    database=new cbdd("192.168.1.76",8086,"db");

    client.HelloUDP();

    QObject::connect(bras,&CBrasHaptique::posGripChanged,this,&ClientBrasHaptique::Angle_roll);
    QObject::connect(bras,&CBrasHaptique::posGripChanged,this,&ClientBrasHaptique::Angle_pitch);
}
```

Les fonctions Angle_roll et Angle_pitch permettent, à chaque fois que le joystick est déplacé, de calculer les angles, les sauvegarder dans la base de données et les envoyer au serveur UDP d’XPlane.

Création des angles

Les fonctions créant les angles sont les fonctions Angle_roll(QPoint p_posGrip) et Angle_pitch(QPoint p_posGrip).

```
void ClientBrasHaptique::Angle_roll(QPoint p_posGrip)
{
    qreal posx = p_posGrip.x();
    float angle;
```

Celles-ci sont similaires dans la création des angles. On récupère la valeur de position (x pour l'angle roll et y pour l'angle pitch) récupérée à l'aide de l'objet p_posGrip avec la méthode ".x()"pour la position x et ".y()" pour la position y. Pour déterminer l'angle on crée une longeur de joystick proportionnelle à la fenêtre appelée "joystick" afin d'obtenir des angles cohérents.

```
angle = qAtan2(posx,joystick);
```

A l'aide de la fonction "qAtan2" (arctangente) on peut donc obtenir des angles à l'aide de ces deux valeurs.

Sauvegarde des angles dans la base de données

J'utilise la même base de données et classe que mon partenaire Mehdi pour la sauvegarde, à la différence du constructeur et de la fonction de sauvegarde.

```
ClientBrasHaptique::ClientBrasHaptique()
{
    bras=new CBrasHaptique;
    bras->show();

    database=new cbdd("192.168.1.76",8086,"db");
```

La base de données est initialisée dans le constructeur de l'objet principal, avec pour seul paramètres ceux nécessaire à la connection avec le serveur.

```
cbdd::cbdd(string AdresseIP,int port,string database):si(AdresseIP,port,database)
```

Les angles sont sauvegardés après le calcul à l'aide de la fonction savebdd() qui prend en paramètre la valeur à sauvegarder et le nom afin de différencier les angles roll et pitch.

```
void cbdd::savebdd(float Angle, string nom)
{

    influxdb_cpp::builder()
        .meas(table1)
        .field(nom,Angle)
        .post_http(si,&resp);

    cout << resp << endl;

}
```

Exemple pour les angles roll :

```
void ClientBrasHaptique::Angle_roll(QPoint p_posGrip)
{
    qreal posx = p_posGrip.x();
    float angle;

    angle = qAtan2(posx,joystick);

    database->savebdd(angle, "Angle roll");
```

Envoi des données au serveur UDP d'XPlane

Afin de communiquer avec XPlane il faut envoyer une trame de 504 octets afin de modifier les datarefs.
La trame est de la forme "HEADER VALEUR NOM_DATAREFS"
HEADER : commande spécifique, DREF pour modifier un dataref
VALEUR : si float à coder en IEEE 754 little endian

Exemple :
DREF\x0\xCD\xCC\x4C\x3Fsim/joystick/yoke_roll_ratio
Cette commande applique un angle roll de 80°.

Construction de la trame

J'utilise un QByteArray pour la trame.

```
QByteArray Data_roll("DREF\x0\x00\x00\x00\x00sim/joystick/yoke_roll_ratio",38);
```

Afin de convertir la valeur de l'angle j'utilise la fonction memcpy avec un tableau de 4 char afin de copier la valeur de mémoire (en IEEE 754 little endian) de la variable "angle".

```
void ClientBrasHaptique::Angle_roll(QPoint p_posGrip)
{
    qreal posx = p_posGrip.x();
    float angle;

    angle = qAtan2(posx,joystick);                    ⚠ implicit conve

    database->savebdd(angle, "Angle roll");

    char IEEE[4];

    memcpy(IEEE, &angle, 4);

    QByteArray Data_roll("DREF\x0\x00\x00\x00\x00sim/joystick/yoke_roll_ratio",38);
```

Ensuite j'attribue les 4 valeurs du tableau à la place des "\x00".

```
    int i;
    for(i=0;i<4;i++)
    {
        Data_roll[5+i]=IEEE[i];
    }
```

Enfin j'effectue un bourrage de trame afin qu'elle fasse 504 octets avec une boucle for, puis elle est envoyée à l'aide de la fonction sendData.

```
for(i=0;i<509-38;i++)
{
    Data_roll.append(0x20);
}

client.sendData(Data_roll);
```

La fonction sendData prend en paramètre un QByteArray et l'envoie sur le socket au port de réception 49000 du serveur UDP XPlane.

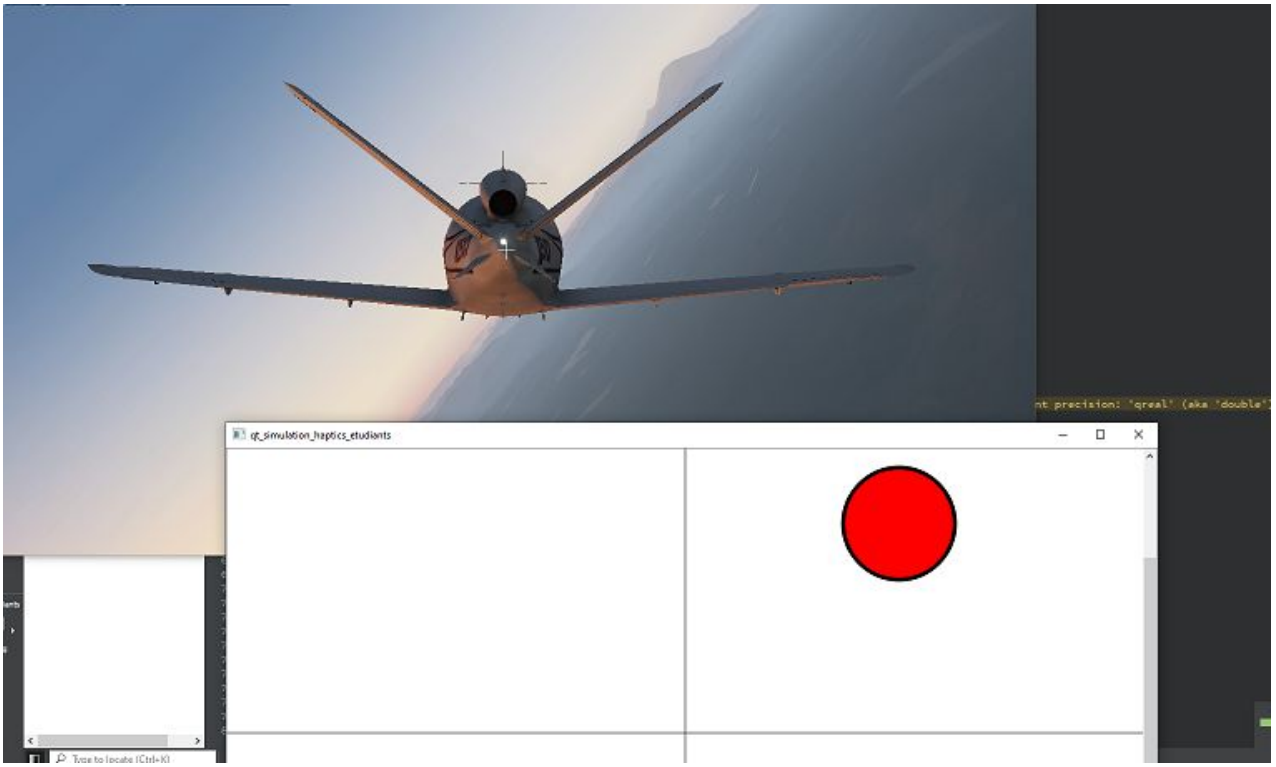
```
void CClientUDP::sendData(QByteArray Data)
{
    socket->writeDatagram(Data, QHostAddress::LocalHost, 49000);
}
```

La fonction Angle_pitch est similaire à la différence du nom du dataref.

```
void ClientBrasHaptique::Angle_pitch(QPoint p_posGrip)
{
    qreal posy = p_posGrip.y();
    float angle;

    angle = qAtan2(posy,joystick);                    ⚠ implicit convers

    database->savebdd(angle, "Angle pitch");

    char IEEE[4];

    memcpy(IEEE, &angle, 4);

    QByteArray Data_pitch("DREF\x0\x00\x00\x00\x00sim/joystick/yoke_pitch_ratio",39);

    int i;
    for(i=0;i<4;i++)
    {
        Data_pitch[5+i]=IEEE[i];
    }

    for(i=0;i<509-39;i++)
    {
        Data_pitch.append(0x20);
    }

    client.sendData(Data_pitch);
}
```

Résultats

Pilotage de l'avion



Sauvegarde BDD

```
> SELECT * from simuVol
name: simuVol
time                Angle pitch Angle roll
----                ----------- ----------
1591358460625879300             0.0017
1591358460738140600 0.0033
1591358460750255900             -0.4
1591358460753254600 -0.038
1591358460760534800             -0.43
1591358460764532700 -0.043
1591358460770736000             -0.46
1591358460774239900 -0.047
1591358460781324000             -0.51
1591358460827829200 -0.057
1591358460833647000             -0.59
1591358460837065500 -0.11
1591358460845364800             -0.61
1591358460860790200 -0.12
1591358460864839700             -0.62
1591358460869354800 -0.13
1591358460874864200             -0.62
```

Bilan

Difficultés rencontrées

- Il m'était difficile de trouver des driver, SDK etc pour le Novint Falcon car l'outil est un peu ancien et les sites et liens officiels de Novint n'existent plus.

Conclusion

Ce projet m'a permis de travailler avec un outil très intéressant que je n'avais jamais manipulé avant. Cela m'a aussi permis de consolider mes connaissances en développement en revoyant des bases très importantes que nous avons vu pendant les 2 ans de BTS :

- les bases de données
- les communications UDP
- la programmation orienté objet et les relations entre classes

J'ai également trouvé le fait de travailler sur le simulateur XPlane11 assez divertissant.