RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Université Paris Saclay

Département d'électronique

Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en électronique

---

Étude et conception d'un système de contrôle d'une plateforme dynamique pour simulateur de vol

---

SI YOUCEF Amine

SOUALAH Mohand Tahar

Présenté et soutenue le 27/06/2020

Composition du jury :

Président	**M. Adel BELOUCHRANI**	**Prof.**	**ENP**
Promoteur	**M. Samir BOUAZIZ**	**Prof.**	**Paris Saclay**
Promoteur	**M. Mourad ADNANE**	**Prof.**	**ENP**
Examinateur	**M. Cherif LARBES**	**Prof.**	**ENP**

ENP 2020

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique

Université Paris Saclay

Département d'électronique

Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en électronique

---

Étude et conception d'un système de contrôle d'une plateforme dynamique pour simulateur de vol

---

SI YOUCEF Amine

SOUALAH Mohand Tahar

Présenté et soutenue le 27/06/2020

Composition du jury :

Président	**M. Adel BELOUCHRANI**	**Prof.**	**ENP**
Promoteur	**M. Samir BOUAZIZ**	**Prof.**	**Paris Saclay**
Promoteur	**M. Mourad ADNANE**	**Prof.**	**ENP**
Examinateur	**M. Cherif LARBES**	**Prof.**	**ENP**

ENP 2020

Dédicaces

On dédie ce travail à nos chers parents,

à nos frères et sœurs,

aux membres de nos familles,

à nos amis et connaissances,

et à toute personne chère à notre cœur.

Remerciements

Tout d'abord, on remercie Dieu pour nous avoir guidé et nous avoir donné la force et le courage nécessaire pour l'accomplissement de notre travail.

Ensuite, on remercie nos deux encadreurs **M. Samir BOUAZIZ** et **M. Mourad ADNANE** pour l'aide qu'ils nous ont apporté avant et durant tout notre projet, ainsi que **M. Rabah SADOUN** pour nous avoir initié à ce projet.

On tient également à remercier les membres du jury, **M. Adel BELOUCHRANI** et **M. Cherif LARBES** pour avoir accepté d'examiner notre travail et de l'enrichir avec leurs propositions.

On remercie aussi nos familles et nos amis respectifs pour leur soutien et leur aide continuelle.

Enfin, nos remerciements s'adressent à toute personne ayant participé de près ou de loin à la réalisation de ce travail.

ملخص

في الآونة الأخيرة، أصبحت أجهزة محاكاة الطيران أدوات أساسية لتدريب الطيارين. لكي يكونوا واقعيين قدر الإمكان، يجب أن يمتلكوا بعض الخصائص الديناميكية، والتي يتم توفيرها من خلال منصتهم الأساسية.

يحتوي هذا المشروع على دراسة وتطوير نموذج أولي لمنصة ديناميكية بالإضافة إلى الدائرة الإلكترونية التي تربطها ببرنامج المحاكاة والذي سيسمح لها بمحاكاة تحركات الطائرة بدقة.

الكلمات المفتاحية: محاكاة الطيران، منصة ديناميكية، موصل كان، مصفوفة البوابات المنطقية القابلة للبرمجة FPGA

Abstract

Flight simulators have become essential tools in the training of future pilots. For them to be as realistic as possible, they should possess some dynamic properties, which are provided by their basis platform.

This project consists of the study and the elaboration of a prototype of a dynamic platform as well as the electronic circuit that links it to the software simulator, which will allow it to simulate plane movements accurately.

Key words : Flight simulator, Dynamic platform, CAN Bus, FPGA.

Résumé

Les simulateurs de vol sont devenus des outils incontournables pour la formation de futurs pilotes. Pour qu'ils soient le plus fidèle possible à la réalité, ils doivent posséder des propriétés dynamiques que peut assurer la plateforme sur laquelle ils reposent.

Notre projet consiste en l'étude et l'élaboration d'un modèle réduit d'une plateforme dynamique ainsi que du circuit électronique la reliant au simulateur logiciel qui lui permettra de recréer le ressenti des mouvements de l'avion.

Mots clés : Simulateur de vol, Plateforme dynamique, Bus CAN, FPGA.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et acronymes

API : Application Programming Interface

CAN : Controller Area Network

CAO : Conception Assistée par Ordinateur

CS : Chip Select

EEPROM : Electrically Erasable Programmable Read Only Memory

FFS : Full Flight Simulator

FPGA : Field Programmable Gate Array

GPIO : General Purpose Input/Output

I2C : Inter Integrated Circuit bus

IMU : Inertial Measurement Unit

LCD : Liquid Crystal Display

LRU : Line Reusable Unit

P3d : Prepar3D

PLA : PolyLactic Acid

PWM : Pulse Width Modulation

RAM : Random Access Memory

SDK : Software Development Kit

SDRAM : Synchronous Dynamic Random Access Memory

SEV : Système d'Entrainement au Vol

SPI : Serial Peripheral Interface

UART : Universal Asynchronous Receiver Transmitter

USB : Universal Serial Bus

VHDL : Very high speed integrated circuit Hardware Description Language

Introduction générale

Compte tenu de la difficulté du pilotage des avions, on a vite eu recours à des méthodes pour entrainer les pilotes. Au fil du temps, divers moyens ont été mis à disposition pour permettre aux pilotes de s'exercer avant leur premier vol.

Dès les débuts de l'aviation on a tenté de restituer les effets aérodynamiques des commandes sur un avion. C'est ainsi que les simulateurs de vol sont nés. Après quelques modèles très simplifiés entre 1910 et 1950, l'approche scientifique a pris le dessus sur ce domaine en mettant en œuvre des modèles de vol sur calculateurs, analogiques d'abord et numériques par la suite. Ce qui a donné aux simulateurs la possibilité de représenter plus fidèlement le comportement d'un aéronef en vol.

Un simulateur de vol se compose principalement de deux parties. D'une part, on trouve l'intérieur de la cabine qui recrée le cockpit d'un avion avec des écrans pour simuler l'environnement extérieur, différents périphériques représentant les commandes de l'avion ainsi que les éléments du tableau de bord qui donnent des indications sur le vol. Cette partie repose sur un calculateur numérique qui effectue les opérations nécessaires et permet de fournir tous les paramètres de vols et les informations sur l'environnement.

D'autre part, on trouve l'extérieur de l'avion qui n'est autre que le mécanisme sur lequel repose la cabine. Il s'agit d'une plateforme dynamique pouvant effectuer des rotations ainsi que des mouvements verticaux dans le but de recréer le plus fidèlement possible les sensations éprouvées lors d'un vol.

Dans ce mémoire, nous allons nous intéresser à la seconde partie, et plus précisément, à la conception d'un système qui permettrait de contrôler une plateforme mobile sur laquelle reposerait une cabine de pilotage en se basant sur les paramètres de vols fournis par le simulateur virtuel.

Il s'agira donc de construire un modèle réduit d'une plateforme mobile d'un côté et de récupérer les paramètres de vol du simulateur existant d'un autre côté. Par la suite, il faudra trouver une fonction qui traduira les paramètres en commandes pour la plateforme et construire le circuit électronique qui se chargera d'assurer l'échange des données. Il faudra également ajouter des instruments permettant de vérifier le bon fonctionnement du système. Le but étant de valider le modèle afin de l'exporter ultérieurement vers un véritable simulateur de vol.

Nous verrons donc dans un premier temps une présentation générale de notre projet en explicitant l'objectif et les problèmes rencontrés pour avant de présenter la démarche de conception que nous avons suivi. Nous verrons par la suite les deux prototypes que nous avons construit en commençant par celui basé sur microcontrôleur puis celui basé sur FPGA en présentant dans chacun les éléments nécessaires à sa réalisation ainsi qu'une explication détaillée du fonctionnement du système. Nous terminerons par une évaluation des performances de notre système et une exposition du déroulement de notre projet et des perspectives de notre travail.

Chapitre 1

Présentation du projet

1.1 Introduction

Le but de ce premier chapitre est de faire une description générale de notre projet et de présenter son contexte.

On verra les notions fondamentales qui entourent le sujet et nous présenterons les objectifs, le cahier des charges et les problématiques rencontrées.

Enfin nous verrons une présentation des éléments électroniques et informatiques qui composent notre projet avant de présenter une démarche détaillée de la façon dont nous avons procédé pour réaliser notre travail.

1.2 Description générale

Dans le but de former efficacement de futurs pilotes, les simulateurs de vol se doivent d'apporter des sensations similaires à la réalité aux pilotes. Encouragés par l'arrivée à maturité de nombreuses technologies, notamment les environnements 3D, les simulateurs sont devenus très répandus car ils offrent de nombreux avantages tels que la sécurité, la diminution des coûts, l'augmentation de l'efficacité des formations et la reproductibilité de situations critiques afin d'entrainer l'équipage à mieux agir dans de telles situations.

Il existe plusieurs catégories de simulateurs de vol selon les fonctionnalités et le rendu qu'ils offrent. Les plus performants et les plus réalistes sont dans la catégories **Full Flight Simulator (FFS)**. Ces derniers incluent des capacités visuelles à travers des écrans qui simulent l'environnement extérieur, et des capacités dynamiques qui permettent au pilote de ressentir les mouvements de l'avion et les effets des accélérations. C'est la plateforme mobile qui est chargée de reproduire ce ressenti, ou plus précisément, le système qui est chargé de récupérer les paramètres du vol depuis le simulateur, de les traduire en commandes pour la plateforme et de les lui transmettre.

Notre projet consiste en l'élaboration d'un système électronique et informatique se chargeant de récupérer les paramètres de vol d'un simulateur déjà existant et fonctionnel qui n'est autre qu'une application sur ordinateur en trois dimensions pour l'entrainement au pilotage nommé Prepar3D, de créer une fonction de transfert entre ces paramètres de vols et les commandes d'une plateforme mobile qui ne sont autres que des moteurs contrôlant son mouvement.

Pour ce qui est de la plateforme, il nous fallait concevoir un modèle réduit qui nous permettrait de valider notre concept. Nous allons donc commencer par présenter le principe de ces plateformes et voir les différents types qui existent. Nous détaillerons la plateforme choisie dans le chapitre suivant.

1.3 Plateformes mobiles

Un simulateur de vol de type **FFS** est composé, comme on peut le voir dans la **Figure 1.1**, d'une plateforme mobile sur laquelle repose une cabine. Cette dernière dispose de sièges pour le pilote et le copilote, d'instruments de vol ainsi que des écrans d'affichages et de simulation de l'extérieur.

FIGURE 1.1 – Simulateur de vol de type FFS

1.3.1 Présentation

Une plateforme mobile est un dispositif mécanique qui permet de simuler les mouvements de l'avion afin d'apporter un ressenti au pilote à l'intérieur de la cabine. Il existe plusieurs types de plateformes, notamment selon le nombre de degrés de liberté.

1.3.2 Types de plateformes

Nous allons voir deux types de plateformes, une plateforme à six degrés de liberté ou plateforme de Stewart et des plateformes à trois degrés de liberté qui sont plus simples et plus facile à réaliser.

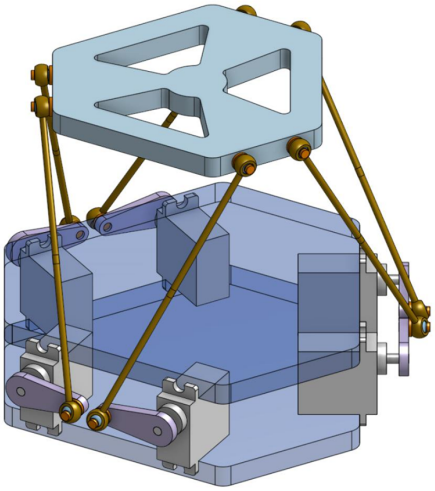
1.3.2.1 Plateforme de Stewart

La plateforme de Stewart est une plateforme à six degrés de liberté permettant trois déplacements linéaires selon trois axes orthogonaux et trois rotations autour des mêmes trois axes.

La plateforme repose sur six pattes identiquement construites comme on peut le voir dans la **Figure 1.2a**. Chaque patte est constituée d'un vérin et elle est terminée par une articulation sphérique et reliée à la base par des joints de cadran. Une autre version de cette plateforme existe comme on peut le voir dans la **Figure 1.2b**. Dans cette dernière, les vérins sont remplacés par des tiges de taille fixe et reliés à des rotations sur la base.

Cette plateforme est une des plus utilisées car elle offre les meilleures performances. Cependant, étant donné son coût et sa complexité, elle se limite aux applications de très haut niveau. Une alternative est donc les plateformes à trois degrés de liberté qui sont moins complexes et offrent tout de même des résultats corrects.

(a) Plateforme avec vérins



(b) Plateforme avec rotations

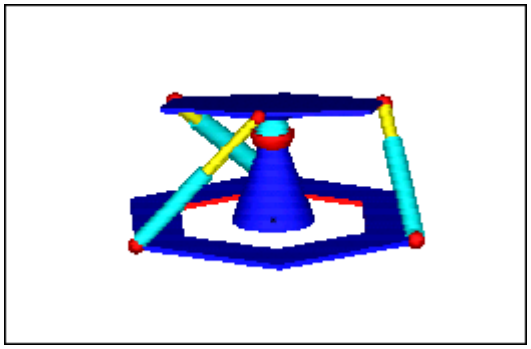
FIGURE 1.2 – Plateformes de Stewart

1.3.2.2 Plateformes à trois degrés de liberté

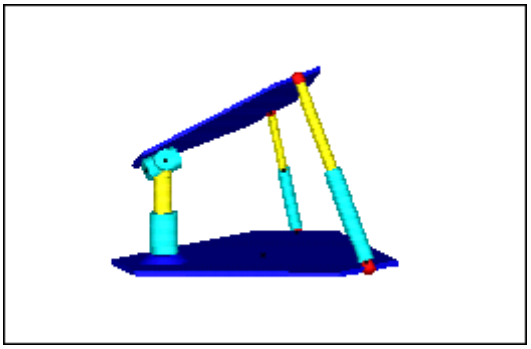
Dans les plateformes à trois degrés de liberté on renonce à trois possibilités de mouvement par rapport à la plateforme précédente. On retrouve donc deux cas :

- Les plateformes sphériques : on garde les trois rotations selon les trois axes, à savoir, le roulis, le tangage et le lacet (ou plus communément Roll, Pitch et Yaw en anglais).

- Les plateformes mixtes : on renonce à une rotation, à savoir, la rotation selon l'axe vertical ou lacet et on ne garde que le déplacement linéaire selon ce même axe.



(a) Plateforme sphérique



(b) Plateforme mixte

FIGURE 1.3 – Plateformes à trois degrés de liberté

Les deux architectures illustrées dans la **Figure 1.3** offrent l'avantage d'être plus simple et plus pratique pour des projets comme le nôtre. Selon l'application, on peut choisir l'une de ces deux architectures en renonçant aux trois degrés de liberté qui auront le moins d'impact sur le ressenti.

1.3.3 Plateforme SEVPro-5

Le SEVPro-5 est un système d'entrainement au Vol professionnel monoplace dynamique destiné à la formation en aéroclub ou centres de formation de pilote. Comme on peut le voir dans la **Figure 1.4**, il est composé d'un module SEV (Système d'Entrainement au vol) monté sur une plateforme mobile dynamique trois axes de type mixte.

FIGURE 1.4 – Simulateur de vol SEVPro-5

La plateforme est composé d'un châssis fixe sur lequel reposent trois servomoteurs positionnés en triangle. Chaque servomoteur est relié à un bras oscillant qui est à son tour relié à la plateforme sur laquelle reposera le module SEV.

La rotation des trois servomoteurs permet de déplacer les trois bras correspondant, ce qui crée les trois degrés de liberté de la plateforme supérieure.

Il nous a été spécifié dans le cahier des charges de travailler sur une plateforme du même type que celle du simulateur SEVPro-5, à savoir une plateforme mixte qui permet de simuler les rotations de l'avion (le roulis), son élévation (le tangage), ainsi que les déplacements sur l'axe vertical.

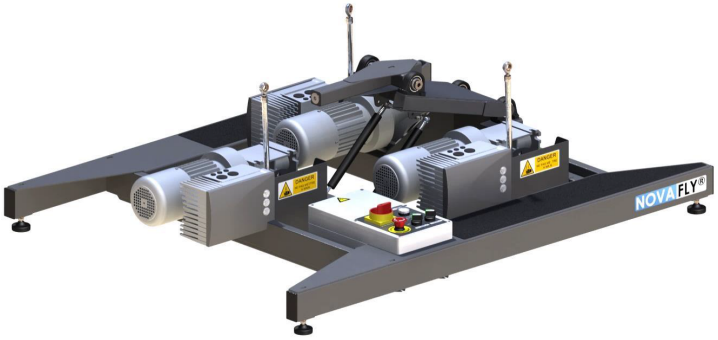


FIGURE 1.5 – Plateforme mobile du simulateur SEVPro-5

1.4 Spécifications du projet

1.4.1 Objectif et problématique

Afin de réaliser un simulateur de vol le plus réaliste possible, il ne suffit pas de permettre de manipuler des instruments et de voir l'effet sur des écrans. Il faudrait également que l'utilisateur ressente l'effet de ces manipulations. Le but est donc de placer la cabine du simulateur qui contient la réplique d'un cockpit sur une plateforme qui, suivant les mouvements de l'avion, va s'incliner de différentes façons : vers le haut quand l'avion monte, vers la gauche quand l'avion penche à gauche et bien d'autres situations plus complexes.

Dans ce but, il nous a été proposé de réaliser une plateforme mobile dynamique permettant de simuler le mouvement d'un avion vu de la cabine de pilotage afin de permettre aux apprentis pilotes d'avoir un ressenti le plus similaire possible à un avion réel.

Pour des raisons de sécurité, on réalisera d'abord un modèle réduit afin de valider le concept. Une fois le système fonctionnel, il suffira alors de le transposer sur le modèle grandeur nature en faisant les adaptations nécessaires.

La plateforme reposera sur 3 servomoteurs disposés en triangle qui permettront de simuler la pente (Tangage), l'inclinaison (Roulis) ainsi que l'élévation. Elle communiquera avec un simulateur de vol logiciel via un circuit électronique pour récupérer les données nécessaires pour la commande des servomoteurs.

La **Figure 1.6** montre une architecture globale du système à concevoir. Un bus de données global relie l'ordinateur qui fait tourner le simulateur de vol logiciel (Prepar3d), un microcontrôleur placé sur la plateforme permettant de contrôler les trois servomoteurs et un microcontrôleur qui contrôle une centrale inertielle, un écran LCD et un joystick.

Des informations sont récupérées depuis le simulateur et sont transmises sur le bus de données. Le premier microcontrôleur se charge de réceptionner ces informations et de les convertir en commandes pour les trois servomoteurs puis les envoie sur le bus CAN. Il permet également de recueillir les informations reçus de l'IMU (centrale inertielle) et de les afficher sur l'écran LCD ainsi que de recueillir les informations du joystick et de les transmettre sur le bus de données. Quant au dernier microcontrôleur positionné sur la plateforme, il réceptionne les commandes et les transmet aux servomoteurs.

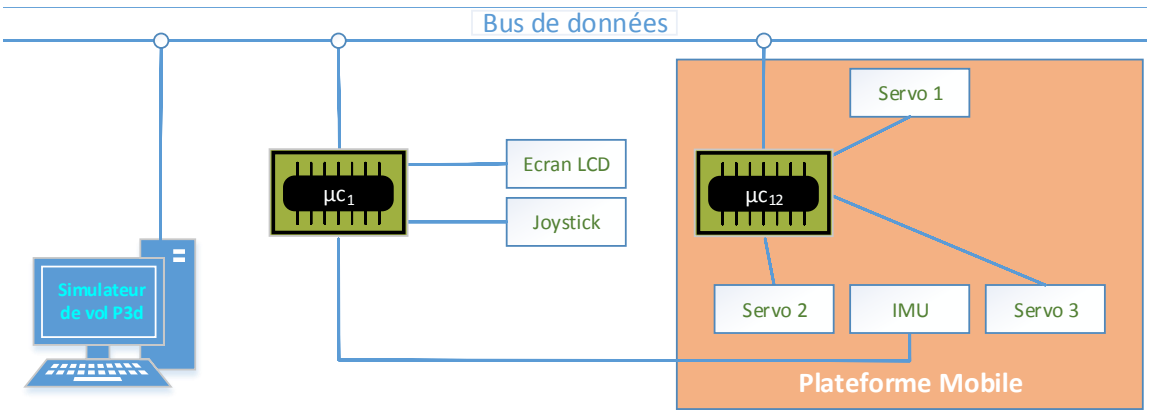


FIGURE 1.6 – Représentation en bloc du système à réaliser

Plusieurs points dans les paragraphes précédents sont à discuter, on les formule ici sous forme de problématiques :

- **Construire un modèle réduit d'une plateforme dynamique :**

Après avoir choisi un type de plateforme et avoir fixé les différentes spécifications de celle-ci (dimensions, forme, design), il fallait trouver le moyen de déterminer les bonnes pièces qui allaient constituer notre plateforme pour ensuite les concevoir. Il fallait également trouver la manière d'assembler ces dernières pour que la plateforme réponde aux critères de départ.

- **Extraire des données depuis le simulateur :**

Pour guider la plateforme à travers les mouvements de l'avion, il faut connaitre certains paramètres tels que les angles d'inclinaison instantanée de l'avion. De ce fait, il est nécessaire de trouver un moyen de se connecter au simulateur pour en récupérer les données dont on a besoin.

- **Concevoir le circuit électronique :**

Nous avons vu dans la **Figure 1.6** une architecture générale. Il faut alors trouver comment faire les différents branchements, à savoir : déterminer le type du bus de données principal, déterminer le moyen de communication entre le bus et l'ordinateur et entre le bus et le microcontrôleur et définir les différents protocoles de communications de tous les composants.

- **Traduire les données du simulateur en commandes pour les moteurs :**

Le simulateur donnera des angles, par exemple : le degré de rotation de l'avion par rapport à l'horizon. Ces angles doivent être traduit en commandes pour les moteurs qui contrôlent le mouvement de la plateforme. Il faut donc trouver une fonction de transfert depuis les angles aux commandes.

1.4.2 Cahier des charges

Nous allons présenter ici les exigences techniques de notre projet en commençant par la partie mécanique puis la partie électronique.

1.4.2.1 Partie mécanique

Pour la partie mécanique, il faut une plateforme à trois degrés de liberté de type mixte permettant deux rotations roulis et tangage ainsi qu'un déplacement linéaire selon l'axe vertical représentant l'élévation de l'avion.

Le modèle de la plateforme doit être inspiré de la plateforme du simulateur SEVPro-5 vu précédemment et doit en reprendre le principe de fonctionnement global.

Les trois bras de la plateforme doivent être contrôlés par des servomoteurs fixés sur la base.

1.4.2.2 Partie électronique

Le circuit électronique doit être conçu autour d'un bus CAN reliant les différentes parties du projet.

Concevoir un nœud connecté au simulateur sur ordinateur pour récupérer les données, un autre connecté à un microcontrôleur MBED pour réaliser la commande des trois servomoteurs et un dernier nœud également connecté à un microcontrôleur MBED pour le contrôle et la visualisation.

Utiliser le Kit de développement Simconnect pour se connecter au simulateur logiciel et récupérer les paramètres de vol qui passeront par une interface graphique à créer.

Réaliser deux modes de commande de la plateforme : un mode automatique avec des angles venant directement du simulateur et un mode manuel avec des angles récupérés d'un joystick que l'on peut manipuler.

Utiliser une centrale inertielle pour comparer les angles réels aux angles envoyés par le simulateur.

Implémenter une fonction de sécurité à deux niveaux par rapport aux servomoteurs pour qu'ils ne dépassent pas des valeurs critiques : un niveau software avec une fonction à programmer et un niveau physique en mettant des capteurs de fin de course et en déclenchant la fonction de sécurité qui interrompt le programme lorsque ces capteurs s'activent.

Pour que le projet soit adapté au modèle grandeur nature, utiliser trois microcontrôleurs pour la commande des trois servomoteurs (un microcontrôleur par servomoteur). Il s'agit du standard LRU (Line Reusable Unit).

Remplacer partiellement le microcontrôleur MBED par une carte FPGA (DE0-nano). Connecter cette dernière au bus CAN via le contrôleur CAN SJA1000.

1.5 Outils électroniques et informatiques

On passe maintenant à la présentation des différents outils et notions électroniques et informatiques utilisés dans ce projet.

1.5.1 Simulateur de vol Prepar3d

1.5.1.1 Présentation

Le simulateur de vol que nous avons utilisé au cours de notre projet se nomme Prepar3D et est développé par Lockheed Martin qui est un constructeur d'avions, c'est donc un simulateur certifié. Il succède en 2010 à Microsoft Flight Simulator après avoir acheté les droits de l'application à Microsoft.

Il s'agit d'un simulateur de vol pour pc qui simule le fonctionnement d'un avion ainsi que son interaction avec l'environnement extérieur. Il permet d'effectuer des vols avec des conditions de réalisme poussées au maximum. Il permet également d'intégrer de la réalité augmentée avec un SDK nommé Oculus.

On retrouve à l'intérieur du cockpit d'un avion choisi exactement les mêmes commandes et instruments que dans un avion réel, comme on peut le voir dans la **Figure 1.7** et avec un fonctionnement identique.

FIGURE 1.7 – Capture d'écran du simulateur Prepar3d

1.5.1.2 SDK

Un kit de développement logiciel ou plus couramment SDK (Software Development Kit) est un ensemble d'outils logiciels destinés aux développeurs, facilitant le développement d'un logiciel sur une plateforme donnée.

Le SDK de Prepar3d fourni à l'utilisateur un ensemble de programmes lui permettant d'interagir avec le simulateur. Il permet de développer, créer ou modifier la simulation.

Il est composé de quatre parties qui traitent chacune une catégorie de la simulation :

- Core Utilities Kit : il couvre la commande des avions, les données instantanées, la configuration camera et la modification des évènements imposés par le client.
- SimObject Container Kit : il couvre la création de panneaux et les autres objets pouvant apparaitre dans l'environnement extérieur (animaux, véhicules ...).
- Environment Kit : il comprend la création de terrains, de scènes et d'effets spéciaux.
- Mission Creation Kit : il permet la création de missions à effectuer dans l'application.

Ce qui nous intéresse c'est de récupérer des informations de vol de l'avion, on s'intéresse donc au Core Utilities Kit et plus précisément à SimConnect.

SimConnect est la partie du SDK qui permet à des applications tierces de communiquer avec le simulateur de vol pour récupérer des données instantanées ou transmettre des commandes depuis la machine où s'exécute l'application ou sur une autre machine en réseau.

SimConnect offre des API en langage C/C++ ou encore en C# nous permettant d'interagir en temps réel avec les paramètres du vol.

Les API sont des interfaces de programmation regroupant un ensemble de fonctions permettant d'accéder aux services d'une application par l'intermédiaire d'un programme.

1.5.1.3 Interface graphique

Afin de faciliter le visionnement des informations tirés du simulateur, nous avons décidé de créer une interface graphique avec la bibliothèque Qt en C++. Cette interface affiche les informations dont on a besoin en temps réel et c'est depuis le même programme que cette interface que l'on envoie les informations vers le bus CAN.

1.5.2 Microcontrôleur

Un microcontrôleur est un circuit intégré comportant un processeur, une mémoire et quelques périphériques. Leur taille peu encombrante ainsi que leur faible consommation d'énergie en font des composants principaux dans les systèmes embarqués.

Nous avons travaillé avec le MBED LPC1768 de ARM. C'est un microcontrôleur basé sur un processeur ARM Cortex-M3 32 bits avec une horloge de 96 MHz. Il inclut une mémoire flash de 512 Ko et une mémoire RAM de 32 Ko.

Figure 1.8 – MBED LPC1768

Il comprend diverses interfaces d'entrées sorties tels que : Ethernet, USB, SPI, I2C, UART, CAN, 6 sorties PWM, 6 Convertisseurs analogiques numériques et des ports GPIO.

Cette carte est programmable en langage C/C++. Il suffit de la brancher à un ordinateur en USB et de lui transmettre le programme compilé. On peut écrire et compiler le programme sur le logiciel **Keil microvision** qui lui est dédié ou bien directement sur un compilateur en ligne, le **MBED Compiler**.

1.5.3 FPGA

Dans la seconde partie de notre conception, nous avons eu recours à une carte FPGA, la DE0-Nano de Terasic.

Une carte FPGA est un circuit programmable destiné au développement et au prototypage rapide. On y retrouve généralement une horloge, une mémoire, des pins d'entrées/sorties programmables et surtout une partie logique programmable.

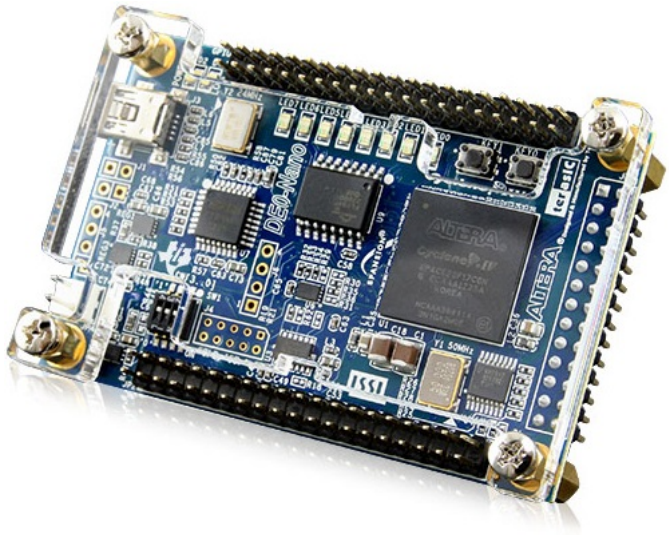


FIGURE 1.9 – Carte FPGA DE0-Nano

La DE0-Nano (**Figure 1.9**) est basé sur un Cyclone IV de la compagnie Intel FPGA (auparavant Altera), il dispose d'une horloge de 50 MHz, de plus de 22000 circuits logiques, d'une mémoire SDRAM de 32 Mo et d'une mémoire EEPROM de 2 Ko. Elle inclut huit leds vertes, quatre switchs, deux boutons poussoirs, 80 pins GPIO programmables ainsi que 26 autres GPIO sur l'autre face de la carte. On y retrouve également un accéléromètre trois axes, un convertisseur analogique numérique à huit canaux de 12 bits.

La programmation de cette carte passe par le logiciel Quartus de Intel. Il faut brancher la carte à un ordinateur hôte via une connexion USB.

1.5.4 Capteurs et instruments

On passe maintenant aux différents instruments et capteurs que nous avons utilisés dans notre projet.

1.5.4.1 Servomoteurs

Un servomoteur est un ensemble mécanique et électronique comprenant un moteur à courant continu et un dispositif électronique d'asservissement permettant au moteur d'effectuer des rotations entre 0° et 180°. On peut accrocher à l'axe de rotation différents types d'outils terminaux selon le besoin.

Le servomoteur possède trois pins, un pour l'alimentation (généralement 5 volts), un autre pour la masse et enfin un troisième qui peut se connecter à une sortie numérique d'un microcontrôleur afin de lui donner une commande correspondante à un angle précis.

Il faut envoyer au servomoteur un signal codé en largeur d'impulsions PWM. En général, la période de ces impulsions est de 20 millisecondes et la durée de l'état haut de l'impulsion détermine l'angle. La durée à l'état haut varie linéairement en général entre 1 ms pour 0° et 2 ms pour 180°.

FIGURE 1.10 – Servomoteur HS-56HB

FIGURE 1.11 – Servomoteur FUS009

Nous avons utilisé trois servomoteurs dans notre projet pour les trois bras de la plateforme : deux servomoteurs identiques (**Figure 1.10**) et un autre légèrement différent (**Figure 1.11**).

1.5.4.2 Centrale inertielle

Le miniMU-9 v2 de Polulu (**Figure 1.12**) est une unité de mesure inertielle qui comprend un accéléromètre, un gyromètre et un magnétomètre de trois axes chacun, ce qui permet d'avoir neuf mesures indépendantes.

Une interface I2C est comprise dans le capteur ce qui permet de le programmer avec notre microcontrôleur. Il possède cinq pins : deux pour l'interface I2C (SDA et SCL), un pour la masse, et deux pour l'alimentation (VIN entre 2.5 et 5.5 volts ou VDD à 3.3 volts si on travaille à cette tension).

Le but de l'utilisation de ce capteur est de combiner les neuf mesures pour obtenir les angles de rotation absolus (Roulis et Tangage) de la plateforme. L'obtention de ces informations nous permet de les comparer aux angles attendus et ainsi de déterminer l'erreur et de la corriger.

FIGURE 1.12 – miniMU-9 v2

FIGURE 1.13 – Pmod JSTK2

1.5.4.3 Joystick

On utilise un joystick afin de commander de façon manuelle les mouvements de la plateforme en plus du mode automatique qui est basé sur le simulateur logiciel.

Le Pmod JSTK2 (**Figure 1.13**) possède un joystick sur deux axes avec au milieu un bouton poussoir ainsi qu'un autre bouton poussoir situé à l'avant. Une interface SPI est mise à disposition pour en récupérer les mesures.

Il possède six pins : un pour l'alimentation (3.3 volts), un autre pour la masse, un pour la sélection du périphérique (cs) et les trois autres pour l'interface SPI (miso, mosi, sck).

1.5.4.4 Écran LCD

Nous avons utilisé un écran LCD pour afficher certaines informations de base comme les angles de rotation de la plateforme.

L'écran utilisé est le DisplayTech 204A (**Figure 1.14**), il permet d'afficher 20 caractères par ligne sur quatre lignes. Il est accompagné d'un adaptateur I2C pour sa programmation.

FIGURE 1.14 – Ecran LCD DisplayTech 204A

1.5.5 Protocoles de communication

Afin de faire communiquer nos différents instruments, nous avons eu recours à plusieurs protocoles que nous présentons ici de façon succincte. Les détails de ces protocoles sont présentés en **Annexe A**.

- **Bus CAN :** C'est un protocole de communication série, bidirectionnel et half duplex permettant de connecter un nombre important de composants tous autant que maitre qui peuvent communiquer à tour de rôle. Il s'agit du bus principal qui relie le simulateur et les deux microcontrôleurs, il utilise différents circuits pour qu'il soit connecté aux périphériques, à savoir : un module USB-CAN pour se connecter en USB à l'ordinateur, un circuit MCP2551 pour se connecter au microcontrôleur MBED et un circuit SJA1000 pour se connecter au FPGA.

- **Bus I2C :** C'est un protocole de communication série, synchrone, bidirectionnel et half duplex permettant la communication entre certains périphériques.

- **Bus SPI :** C'est un protocole de communication série, synchrone, bidirectionnel et full duplex. Il permet de relier le joystick au premier microcontrôleur.

1.5.6 Logiciels et interfaces de programmation

Nous avons utilisé plusieurs logiciels et interfaces de programmation pour réaliser notre travail, on présente les plus importants ici :

- **Simulink :** Nous avons eu besoin de visualiser rapidement des données qui circulent sur le bus CAN. Nous avons donc connecté notre microcontrôleur MBED en série avec l'ordinateur et avec Simulink, ce qui nous a permis de dessiner des graphes et d'enregistrer les données pour une utilisation ultérieure.

- **PcanView :** PcanView est un logiciel fourni avec le driver de l'adaptateur USB-CAN qui permet d'observer en temps réel l'échange des trames. On peut y lire les données, leur taille, leur identifiant, la période d'envoi ainsi que le nombre de fois qu'une donnée a été transmise avec cet identifiant. Il permet également d'envoyer des données sur le bus en spécifiant l'identifiant, la taille du message, le message et la période entre chaque envoi. Nous l'avons utilisé principalement pour faire du monitoring de toutes les données qui circulent sur le bus.

- **SolidWorks :** SolidWorks est un logiciel de CAO (Conception assistée par ordinateur) développé par Dassault Systèmes SE. Il s'agit d'un logiciel permettant la conception en 3D de pièces avec une multitude de paramètres. Nous l'avons utilisé pour concevoir des pièces dont nous avions besoin pour notre projet. A la fin de la conception, le logiciel génère, entre autre, un fichier que l'on peut passer à une imprimante 3D pour obtenir la pièce désirée.

- **Keil microvision :** Afin de programmer notre MBED, nous avons utilisé le logiciel Keil microvision de ARM qui est un environnement de développement complet permettant d'éditer le code et de le compiler. Une fois la compilation terminée, un fichier .bin est généré et transféré sur la carte. Il faut ensuite flasher cette dernière pour que le programme soit chargé dans sa mémoire flash.

- **Quartus :** Intel Quartus Prime est un logiciel de conception de dispositifs logiques programmables. Il nous permet d'écrire du code VHDL, de le vérifier, d'assigner des ports de notre carte VHDL, de faire de la simulation et enfin de transférer notre code vers la carte.

1.5.7 Systèmes temps-réels

Un système temps réel est un système informatique dont le fonctionnement est soumis à l'évolution dynamique d'un procédé extérieur qui lui est connecté. Ainsi, un système temps réel prend en compte des contraintes temporelles.

Un système temps réel n'est pas forcément un système « rapide/qui va vite » mais un système qui répond à certaines contraintes temporelles. Ces contraintes varient selon le domaine de l'application, elles peuvent aller de quelques millisecondes dans le contrôle d'un avion à plusieurs heures pour ajuster des prévisions météorologiques par exemple.

On distingue trois catégories de systèmes temps réels :

- Temps réel dur ou critique (Hard real-time) : le non-respect des contraintes temporelles entraine la panne du système.

- Temps réel souple (Soft real-time) : le respect des contraintes temporelles est important mais leur non-respect n'entraine pas de graves conséquences.

- Temps réel ferme (Firm real-time) : le non-respect des contraintes temporelles rend le résultat inutile.

1.6 Démarches de conception

Nous avons divisé notre projet en cinq parties principales que nous présentons ici dans l'ordre.

1.6.1 Partie mécanique

• Réalisation d'un modèle réduit d'une plateforme pour simulateur de vol en respectant la proportionnalité par rapport aux dimensions de la plateforme réelle du simulateur SEVPro-5.

• Impression des pièces réalisées sous SolidWorks sur une imprimante 3D.

• Vérification du bon assemblage des pièces et de la position du centre de gravité de la plateforme supérieure.

1.6.2 Partie électronique

• Réalisation d'un circuit basé sur un bus CAN.

• Mettre en place un nœud de récupération de données du simulateur (les angles Roulis et Tangage) et l'élévation.

• Mettre en place un nœud MBED pour le traitement des données avec un Joystick et un switch permettant de choisir le mode de contrôle de la plateforme (Manuel : via le joystick ou Automatique : via le simulateur).

• Mettre en place un autre nœud MBED pour la commande. On le connecte aux trois servomoteurs.

1.6.3 Fusion des deux parties

• Réalisation d'une fonction de transfert qui prend en entrée les angles Pitch et Roll et qui fournit en sortie la commande des trois servomoteurs.

• Définir les valeurs que les servomoteurs prendront au démarrage du système.

1.6.4 Amélioration du modèle

• Introduction d'une boucle de retour : on ajoute une IMU sur la plateforme supérieure, ce qui permettra d'avoir un retour d'expérience et de déterminer les retards, les temps de réponses et les corrections à apporter notamment en observant les différentes courbes de réponses.

• Ajouter une sécurité Software : angles maximales à ne pas dépasser sur le MBED.

• Ajouter une sécurité Hardware : bouton microswitch qui sera activé si les angles maximales définis sont dépassés. Son activation provoquera une interruption et l'arrêt de l'envoi des commandes.

• Passage en standard LRU (Line Reusable Unit) : chaque servomoteur sera commandé par un MBED.

1.6.5 Migration sur FPGA

• Implémentation de la commande des servomoteurs et de la sécurité sur un FPGA (DE0-nano) via un automate pour accélérer la prise de décision. Le FPGA interfacera avec le MBED en I2C (implémentation d'une brique I2C sur le FPGA).

• Utilisation du circuit SJA1000 pour connecter la FPGA directement au bus CAN.

1.6.6 Tableau des tâches

Afin d'avancer dans notre travail de façon optimale, on a eu recours à un outil de gestion de projet. Nous avons utilisé le diagramme de Gantt pour nous représenter nos différentes tâches avec les durées de chacune. La **Figure 1.15** dans la page suivante, présente le diagramme de Gantt prévisionnel établit au début du projet. Nous verrons dans la dernier chapitre le diagramme effectif et nous expliquerons les raisons des changements du planning.

1.7 Conclusion

Nous avons vu à travers ce chapitre un aperçu clair du but du projet ainsi que des démarches de sa réalisation et des contraintes auxquelles il doit répondre. Nous avons également vu les différents éléments qui le constitue.

Il est à présent temps de passer à la partie pratique. Partie que nous allons traiter dans les deux chapitres suivants en commençant par le premier prototype basé sur microcontrôleur puis sur celui basé sur FPGA et nous finirons par discuter les résultats et les perspectives de notre travail dans le dernier chapitre.

	Tâche	Date de début	Date de fin	Durée
1	**Conception de la plateforme**	**09/02/2020**	**18/02/2020**	**8**
1.1	Modélisation de la plateforme sur SolidWorks	09/02/2020	13/02/2020	5
1.2	Impression des pièces avec une imprimante 3D	16/02/2020	18/02/2020	3
2	**Réalisation du circuit électronique : Nœud Plateforme**	**16/02/2020**	**29/03/2020**	**7**
2.1	Commande des servomoteurs avec un MBED	16/02/2020	18/02/2020	3
2.2	Implémentation de la fonction d'initialisation sur MBED	24/03/2020	25/03/2020	2
2.3	Implémentation de la fonction de sécurité sur MBED	26/03/2020	29/03/2020	2
3	**Réalisation du circuit électronique : Nœud Simulateur**	**19/02/2020**	**08/03/2020**	13
3.1	Etude du SDK Simconnect	19/02/2020	24/02/2020	4
3.2	Récupération d'informations depuis le simulateur	25/02/2020	03/03/2020	6
3.3	Transfert des données sur le bus CAN	04/03/2020	08/03/2020	3
4	**Réalisation du circuit électronique : Nœud Contrôle**	**09/03/2020**	**16/03/2020**	6
4.1	Choix du mode (manuel ou automatique)	09/03/2020	10/03/2020	2
4.2	Récupération des angles du joystick	11/03/2020	11/03/2020	1
4.3	Récupération des angles du gyroscope, affichage sur LCD	12/03/2020	16/03/2020	3
5	**Réalisation de la fonction de transfert**	**17/03/2020**	**05/04/2020**	11
5.1	Etude du modèle cinématique de la plateforme	17/03/2020	19/03/2020	3
5.2	Implémentation de la fonction de transfert sur MBED	22/03/2020	23/03/2020	2
5.3	Etude et implémentation du filtre de Washout	30/03/2020	05/04/2020	5
5.4	Passage en logique LRU	05/04/2020	05/04/2020	1
6	**Migration sur FPGA : connexion en I2C avec MBED**	**07/04/2020**	**28/04/2020**	16
6.1	Conception d'une brique I2C sur FPGA	07/04/2020	12/04/2020	4
6.2	Connexion du FPGA au MBED en I2C	13/04/2020	14/04/2020	2
6.3	Brique d'initialisation et brique de sécurité	15/04/2020	22/04/2020	6
6.4	Brique de commande des servomoteurs	23/04/2020	28/04/2020	4
7	**Migration sur FPGA : connexion en CAN avec le circuit**	**29/04/2020**	**25/05/2020**	19
7.1	Etude du circuit SJA1000	29/04/2020	12/05/2020	10
7.2	Brique d'écriture et de lecture des registres	13/05/2020	17/05/2020	3
7.3	Envoie et récéption de trames du Bus CAN	18/05/2020	25/05/2020	6
8	**Finalisation du projet**	**26/05/2020**	**18/06/2020**	18
8.1	Analyse des temps de réponses	26/05/2020	01/06/2020	5
8.2	Rédaction du mémoire	02/06/2020	18/06/2020	13

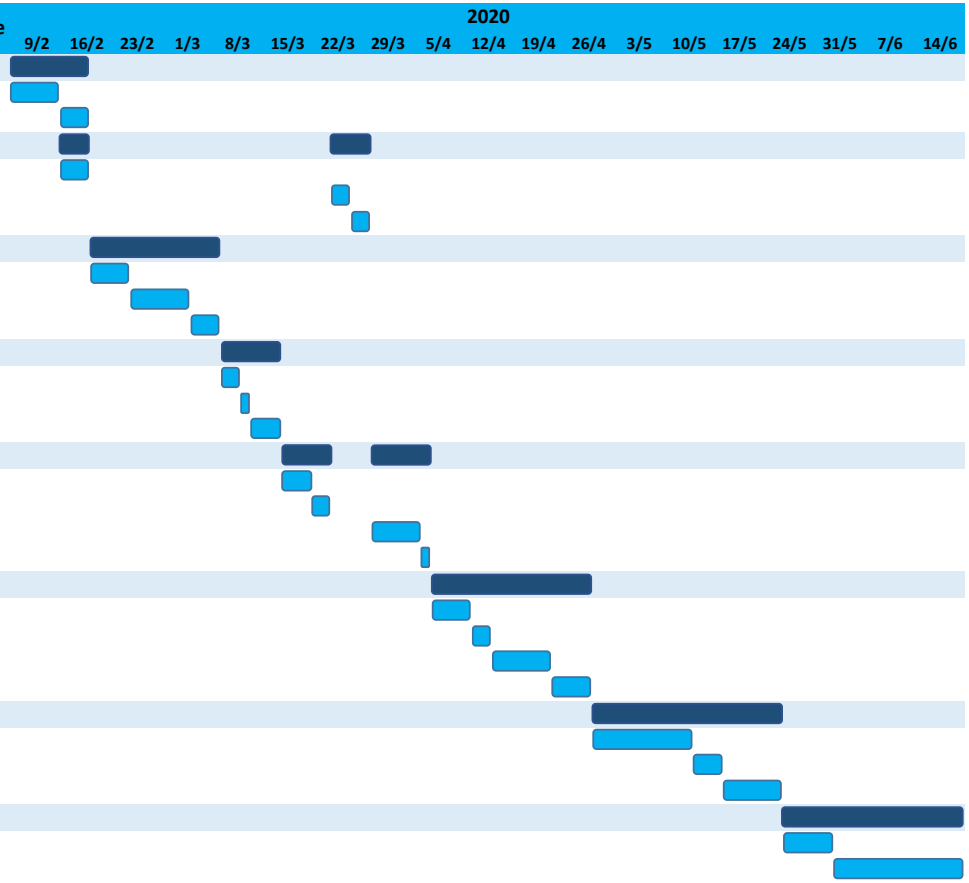


FIGURE 1.15 – Diagramme de Gantt prévisionnel

Chapitre 2

Conception du prototype basé sur microcontrôleur

2.1 Introduction

Le présent chapitre est consacré à la première partie de notre travail qui concerne le premier prototype construit à base de microcontrôleurs.

On y abordera d'abord la construction de la plateforme mobile dynamique depuis sa modélisation sur ordinateur jusqu'à la définition d'un modèle cinématique en passant par son impression et son assemblage.

Par la suite on passera au circuit électronique en détaillant le rôle de chaque composant et en explicitant les procédés de communications entre les périphériques.

2.2 Construction de la plateforme

On commence ce chapitre en nous intéressant à la construction de la plateforme depuis le choix de son design et de ses paramètres jusqu'à l'étude de son modèle cinématique.

2.2.1 Choix du modèle

Le choix du modèle de la plateforme mobile a été imposé par le cahier des charges. En effet, il doit s'inspirer de la plateforme du simulateur de vol SEVPro-5 vu au premier chapitre.

Le modèle est constitué de deux plateformes, la première étant fixe et la seconde mobile. Ces deux plateformes sont reliées par des bras dont les mouvements sont contrôlés par des servomoteurs fixés sur la plateforme fixe.

Une disposition de ces servomoteurs en triangle permet d'avoir trois degrés de liberté sur le mouvement de la plateforme : deux rotations tangage et roulis et une translation selon l'axe vertical, tout comme la plateforme du simulateur SEVPro-5.

Le SEVPro-5 utilise une plateforme à trois degrés de liberté car il s'agit d'un simulateur de formation catégorie simple, on n'a donc pas besoin de tous les degrés de liberté. On garde les trois degrés cités plus haut car ce sont ceux qui auront le plus d'impact sur le mouvement de la plateforme.

La plateforme inférieure (fixe) est sous la forme d'un rectangle de dimensions (210 x 190 mm) ce qui correspond à peu près à un redimensionnement de la plateforme grandeur nature avec une échelle de (1/4).

La plateforme supérieure est sous forme d'un trapèze de dimensions légèrement inférieures au rectangle (le trapèze est contenu dans le rectangle). Elle est reliée à la plateforme inférieure par des bras qui sont reliés aux servomoteurs.

2.2.2 Conception du modèle

Afin de modéliser cette plateforme, nous avons utilisé le logiciel de conception assistée par ordinateur SolidWorks.

2.2.2.1 Plateformes inférieur et supérieure

Tout d'abord nous avons modélisé la plateforme inférieure. Un rectangle contenant trois emplacement pour les servomoteurs ainsi que trois emplacement pour les trois capteurs de fin de course, situés chacun juste à côté de l'emplacement du servomoteur.

Ensuite nous sommes passé à la plateforme supérieure. Un trapèze avec au centre un emplacement consacré à la centrale inertielle. Un vide est également réalisé sur le bas de la plateforme pour nous permettre d'équilibrer cette dernière en contrôlant la position du point de gravité.

Enfin, nous avons modélisé les différentes pièces complémentaires, à savoir : les réceptacles des servomoteurs à fixer sur la plateforme inférieure et les bras, ceux qui seront fixés aux extrémités des servomoteurs et ceux qui seront fixés à la plateforme supérieure et reliés aux premiers.

Les **Figures 2.1, 2.2 et 2.3** montrent des vues d'angles différents du résultat de la modélisation de la plateforme sous le logiciel SolidWorks.

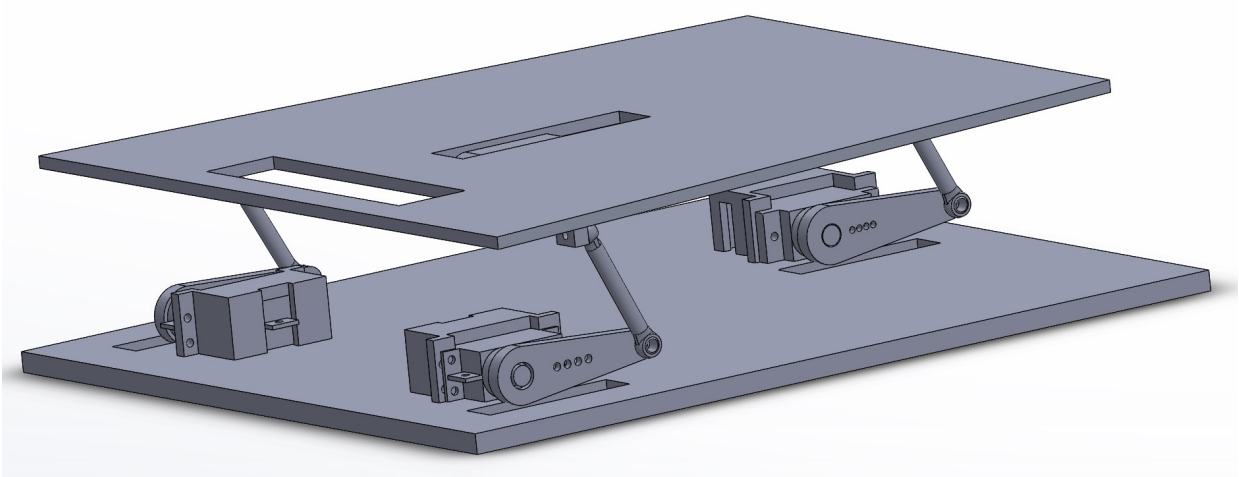


FIGURE 2.1 – Vue générale de la modélisation de la plateforme

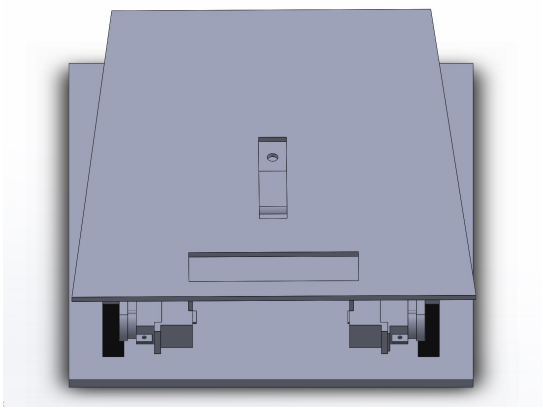


FIGURE 2.2 – Vue de haut de la plateforme

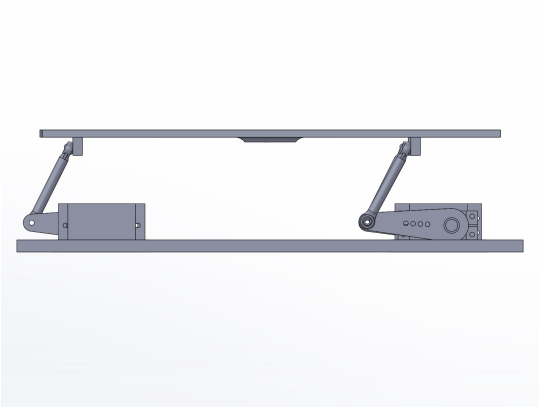


FIGURE 2.3 – Vue de côté de la plateforme

2.2.2.2 Bras de la plateforme

Il y a deux types de bras qui relient les deux plateformes. Le premier (**Figures 2.4**) est accroché à l'extrémité tournante du servomoteur via un engrenage. Cette pièce suit donc de façon exact les rotations du servomoteur. On trouve à l'autre extrémité de ce bras un trou permettant de le raccorder au second bras.

Le second bras est constitué d'un cylindre uniforme terminé de part et d'autre par des ouvertures identiques lui permettant d'être raccordé au premier bras d'un côté et à la plateforme supérieure de l'autre comme on peut le voir dans la **Figures 2.5**.

Le même couple de bras est réalisé pour chacun des trois servomoteurs.

FIGURE 2.4 – Bras du servomoteur

FIGURE 2.5 – Bras de la plateforme

2.2.3 Impression et assemblage

Après avoir terminé la modélisation de la plateforme sur SolidWorks, nous sommes passé sur une imprimante 3D pour imprimer le modèle.

Comme matière, nous avons utilisé une bobine PLA de 1,75 mm de diamètre. La durée totale de l'impression était d'environ 30 heures réparties sur plusieurs jours.

Nous avons commencé par les deux plateformes puis on est passé aux autres pièces, celles dédiées aux servomoteurs et les bras. Une fois l'impression terminée, nous sommes passé à l'assemblage. Nous avons commencé par fixer les réceptacles des servomoteurs par des vices dont l'emplacement était prévu dans la modélisation puis nous avons raccordé les servomoteurs aux bras puis à la plateforme supérieure.

On peut voir sur la **Figure 2.6** le résultat final de la plateforme assemblée.

2.2.4 Étude cinématique

Dans le but de contrôler le mouvement de la plateforme, nous avons besoin de déterminer les trois angles à donner aux trois servomoteurs à partir des données que nous allons récupérer du simulateur de vol, à savoir, les angles tangage et roulis.

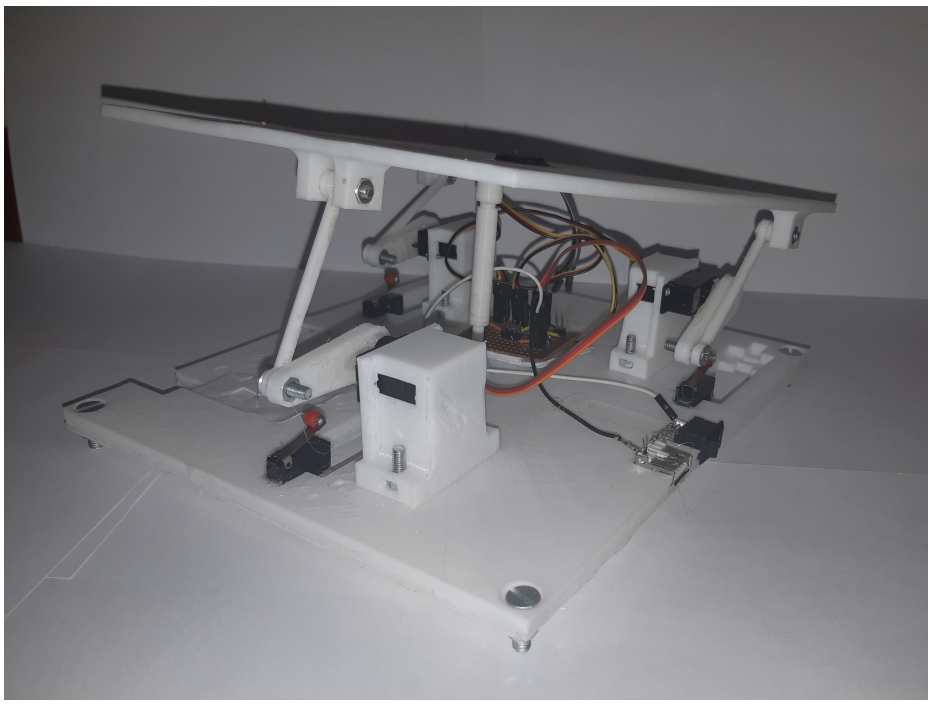


FIGURE 2.6 – Assemblage final de la plateforme mobile

A travers l'étude du modèle cinématique de la plateforme, l'objectif est de concevoir une fonction de transfert capable de convertir les deux angles tangage et roulis en trois commandes pour les trois servomoteurs.

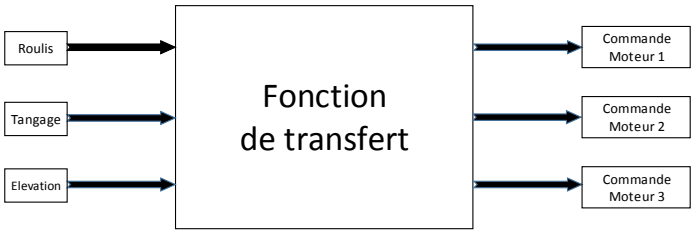


FIGURE 2.7 – Fonction de transfert entre les angles et les commandes

Pour ce faire, nous avons mené une étude cinématique de la plateforme et nous sommes sortis avec des résultats qui sont les équations des trois angles à donner au servomoteur en fonction des deux angles que l'on a. Cette étude est expliquée en **Annexe B**.

Cependant, l'implémentation de cette fonction serait trop couteuse en temps de calcul et nous ferait perdre l'aspect temps réel du système.

Nous avons donc réfléchi à exécuter cette fonction ailleurs pour différentes valeurs d'entrées (angles tangage et roulis) dans l'intervalle de leurs variations et enregistrer les sorties associées (angles à affecter aux servomoteurs). On pourrait alors enregistrer ceci dans un tableau que l'on gardera dans la RAM du microcontrôleur et à chaque fois que l'on reçoit deux angles tangage et roulis, on trouvera dans le tableau les deux valeurs les plus proches à ces deux angles et on affectera aux servomoteurs les trois valeurs associées depuis le tableau.

Là encore, cette solution présente l'inconvénient d'être gourmande en mémoire car si on veut avoir une bonne précision sur les angles d'entrés, il nous faudrait beaucoup de mémoire disponible sur le microcontrôleur.

Nous avons finalement opté pour une solution simplifié nous permettant de valider le système. Nous avons choisi d'affecter au premier servomoteur (positionné au centre et à l'avant de la plateforme) la valeur du tangage. Quant aux deux autres servomoteurs (à l'arrière de la plateforme), on donne la valeur du tangage moins celle du roulis pour celui de droite et la valeur du tangage plus celle du roulis pour celui de gauche.

2.3 Architecture du système

Maintenant que la plateforme est prête et que nous avons une fonction de transfert sur papier, il est temps de passer au circuit électronique qui va relier nos différents instruments afin de pourvoir apporter du mouvement à notre plateforme.

2.3.1 Schéma général

L'architecture globale du circuit électronique est basé sur un Bus CAN qui sera le bus de données principal et qui servira d'intermédiaire entre les différentes parties du système.

Une connexion à un bus CAN est appelée nœud. De ce fait, notre système est divisé en trois nœuds chacun se chargeant de tâches spécifiques.

Tout d'abord, on a l'ordinateur qui fait tourner le logiciel du simulateur de vol qui doit également être relié au bus CAN via un module USB-CAN. Il s'agit de notre premier nœud que l'on nommera nœud simulateur.

Ensuite, nous avons un MBED auquel sont reliés un joystick, un écran LCD, un switch et une IMU se trouvant sur la plateforme. Le MBED est relié au bus CAN à travers un adaptateur de ligne (le circuit MCP2551) ce qui formera notre deuxième nœud appelé nœud contrôle.

Enfin, on trouve sur la plateforme trois servomoteurs que l'on doit relier à un microcontrôleur MBED. Ce MBED est également relié au bus CAN à travers un autre circuit MCP2551 ce qui formera notre troisième nœud appelé nœud commande.

La **Figure 2.8** montre un schéma global détaillé contenant tous les composants utilisés, les différentes connexions entre eux ainsi que les protocoles ou les circuits permettant de relier ces composants.

2.3.2 Explication des nœuds

- **Nœud Simulateur :**

Sur ce nœud nous avons un ordinateur sur lequel est installé l'application Prepar3d (ou plus couramment P3d). On communique avec cette application à travers le SDK SimConnect qui nous permet de récupérer les données qui nous intéressent.

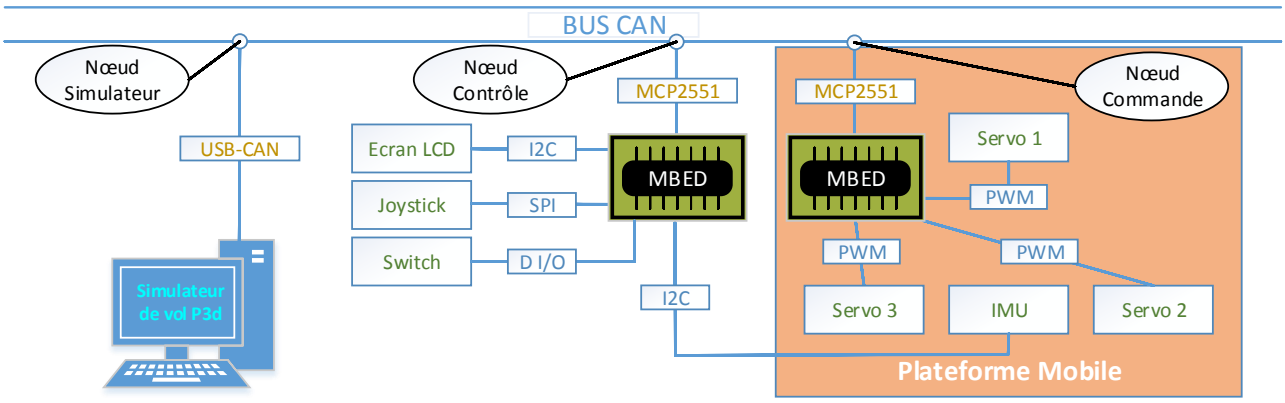


FIGURE 2.8 – Architecture global du système

Une fois en possession de ces données, on les fait passer par une interface graphique. Cette dernière nous permet de visualiser en temps réel les données que l'on souhaite mais également de contrôler l'envoi ou non de ces données sur le bus CAN.

Ces données sont ensuite transmises sur le bus à travers le module USB-CAN à l'aide de son API qui nous permet de programmer la façon dont ces données sont envoyées sur le bus.

- **Nœud Contrôle :**

Dans ce nœud un microcontrôleur MBED est connecté au bus CAN à travers un transceiver de ligne. Plusieurs composants sont connectés à ce microcontrôleur :

- Centrale inertielle : placé sur la plateforme, il permet de récupérer les angles instantanés des inclinaisons de cette dernière selon deux axes, nous avons donc le tangage et le roulis.

- Joystick : permet de diriger la plateforme en mode manuel, ses deux angles selon ses deux axes sont récupérés et seront par la suite traduits en commandes pour les servomoteurs de la plateforme.

- Switch : un bouton à deux états qui permet de basculer entre le mode automatique (angles venant du simulateur) et le mode manuel (angles venant du joystick).

- Écran LCD : il nous permet d'afficher des informations qui sont en l'occurrence les deux angles récupérés grâce à l'IMU et le mode de contrôle actuel.

- **Nœud Commande :**

De même que pour le nœud précédent, celui-ci contient également un MBED. Sauf que le rôle de ce microcontrôleur est différent du précédent. Celui-ci lit des commandes depuis le bus CAN et les transmet aux servomoteurs. Ce qui permet au final le mouvement de la plateforme suivant les données reçues sur le bus CAN.

2.4 Fonctionnement du système

Nous allons maintenant détailler le fonctionnement de chaque nœud et son interaction avec les autres.

2.4.1 Nœud Simulateur

Dans ce nœud, il y a trois étapes principales : se connecter au simulateur et en extraire des données, afficher ces données sur une interface graphique et enfin transmettre ces données sur le bus CAN pour une utilisation ultérieure par les autres nœuds.

2.4.1.1 Connexion au simulateur avec SimConnect

La **Figure 2.9** montre les différentes étapes permettant de récupérer des données depuis le simulateur à travers le SDK SimConnect. Chaque étape fait appel à certaines fonctions spécifiques.

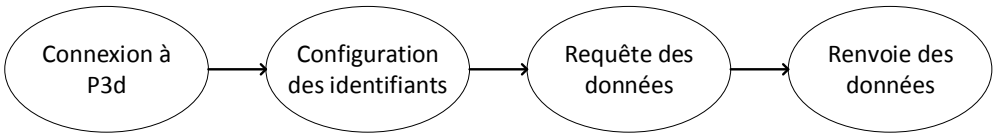


FIGURE 2.9 – Organigramme de la récupération de données avec SimConnect

D'abord, la connexion est établie entre le programme et l'application P3d grâce aux fonctions **P3DStart** et **P3DConnect**.

Ensuite, il faut configurer l'échange des données en spécifiant les identifiants relatifs aux données que l'on souhaite récupérer. Ceci est permis grâce à la fonction **P3DConfig**.

Après, il s'agit de demander les données aux simulateurs en faisant une requête basée sur la configuration faite précédemment à travers la fonction **P3DRequestData**.

Enfin, le simulateur répond avec les données demandées. Ce processus est répété tant qu'on demande ces données.

2.4.1.2 Interface graphique pour la gestion des données

Maintenant que nous avons réussi à récupérer les deux angles, nous allons les afficher sur une interface graphique.

Pour la création de cette interface, et étant donné que nous travaillons déjà en C++, nous avons choisi la bibliothèque Qt qui permet de créer des interfaces graphiques. C'est une bibliothèque très complète et qui subvient largement à nos besoins.

Comme on peut le voir sur la **Figure 2.10**, on peut commander à travers la fenêtre principale la connexion ou la déconnexion au simulateur et également l'envoi ou non des données sur le bus. On peut également observer en temps réel les valeurs des angles tangage et roulis récupérés du simulateur.

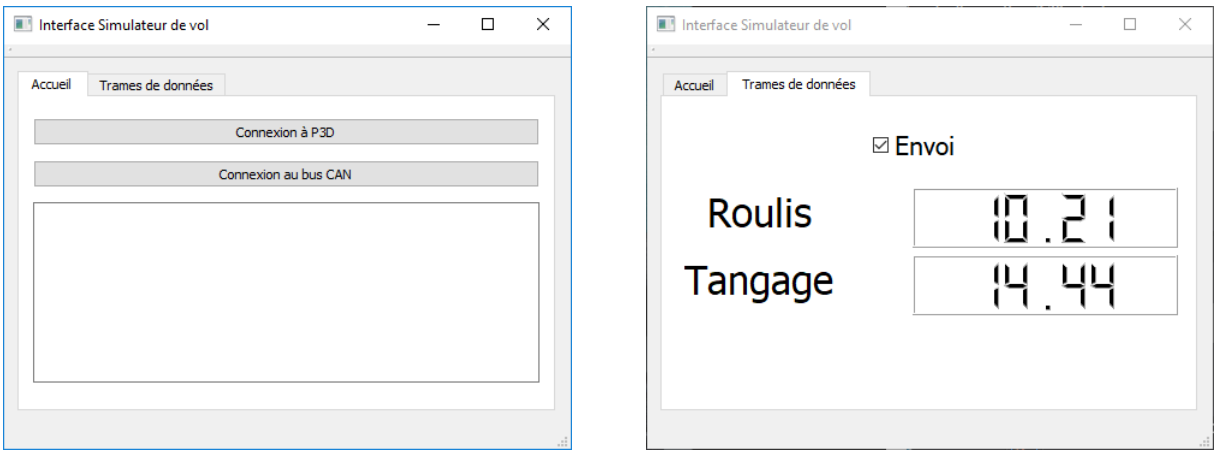


FIGURE 2.10 – Captures d'écran de l'interface graphique

En outre, nous avons également utilisé le logiciel PCANView pour visualiser toutes les données qui circulent sur le bus CAN avec les valeurs et l'identifiant de chaque trame comme on peut le voir dans la **Figure 2.11**. Il nous permet également d'envoyer des trames de façon personnalisée afin de vérifier le bon fonctionnement du système.

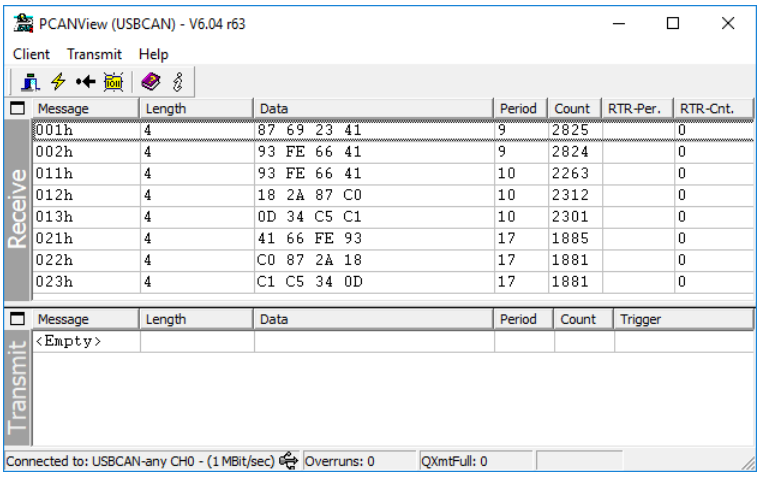


FIGURE 2.11 – Capture d'écran du logiciel PCanView

2.4.1.3 Transfert des données sur le bus CAN

La **Figure 2.12** montre les différentes étapes permettant l'échange de données entre le programme sur ordinateur et le bus CAN à travers l'adaptateur USB-CAN. Chaque étape fait appel à certaines fonctions spécifiques.

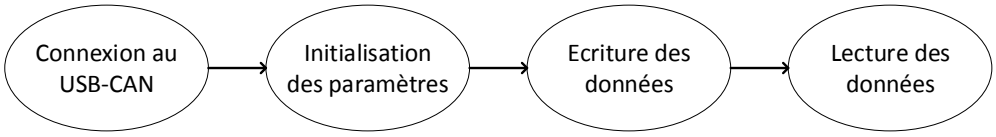


FIGURE 2.12 – Organigramme de l'échange de données à travers l'USB-CAN

D'abord, une connexion à l'adaptateur est créée grâce aux fonctions **UsbCanStart** et **UsbCanConnect** avant d'initialiser les différents paramètres de l'échange tels que la fréquence ou les identifiants.

Ensuite, on transmet les données du programme vers le bus CAN à travers la fonction **UsbCanSend**. On peut également lire les données présentes sur le bus CAN à travers la fonction **UsbCanReceive**.

2.4.2 Nœud Contrôle

Le nœud de contrôle est chargé, à travers son MBED, de lire les données envoyées par le nœud précédent depuis le bus CAN et de lire les données du joystick. Le MBED est aussi chargé de lire l'état du switch pour décider de prendre en considération les données du simulateur dans le cas du mode automatique ou les données du joystick dans le cas du mode manuel.

Par la suite les données choisies passeront par une fonction de transfert qui générera les commandes pour les trois servomoteurs. Ces commandes seront alors envoyées dans le bus CAN pour qu'elles soient récupérées par le nœud suivant.

Enfin, le MBED est également chargé de lire les données de la centrale inertielle et de les convertir en deux angles représentant l'inclinaison de la plateforme. Ces deux angles seront affichés sur l'écran LCD ainsi que le mode choisi.

Le schéma de la **Figure 2.13** permet de visualiser les différentes connexions du nœud ainsi que les types de connexions entre chacun.

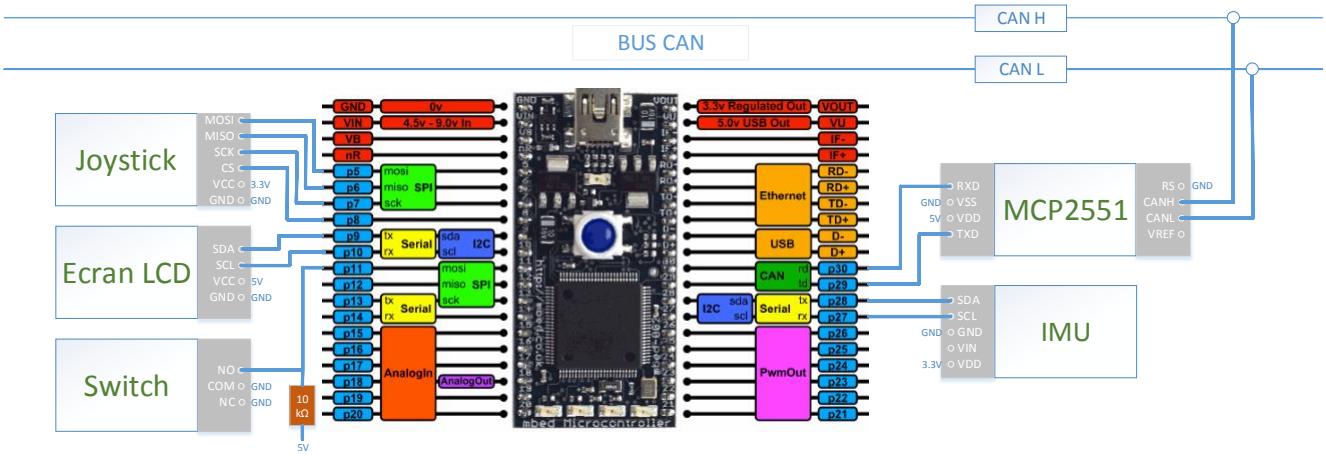


FIGURE 2.13 – Architecture du nœud de contrôle

2.4.2.1 Centrale inertielle

Le MinlMU-9 v2 est un composant électronique qui regroupe trois capteurs :

- Un accéléromètre : il mesure les accélérations linéaires selon les trois axes et nous permet donc de savoir dans quelle direction se déplace un objet.
- Un gyromètre : il mesure les accélérations de rotation autour des trois axes et il ne donne que des valeurs relatives.
- Un magnétomètre : il mesure les champs magnétiques autour de lui et peut être utilisé comme boussole.

Grâce à une interface I2C, on peut lire les neuf valeurs. Ce qu'on cherche c'est obtenir des valeurs absolues des deux angles d'inclinaison tangage et roulis. Une combinaison des valeurs de l'accéléromètre peut nous donner ces deux angles. Seulement, l'accéléromètre seul n'est pas efficace pour les hautes fréquences (variations brusques). Le gyromètre quant à lui n'est pas très efficace pour les faibles fréquences : avec peu de mouvements, il finit par dévier de la bonne valeur à cause de son principe d'intégration.

Une combinaison des six données de l'accéléromètre et du gyromètre nous permet donc de combler les failles de chacun. On obtient les angles roulis (θ) et tangage (ϕ) depuis l'accéléromètre avec les relations suivantes :

$$tan(\theta) = \frac{a_y}{a_z} \qquad tan(\phi) = -\frac{a_x}{\sqrt{a_y^2 + a_z^2}}$$

Avec : a_x, a_y et a_z les accélérations dans chaque direction.

Les mêmes angles peuvent être trouvés par le gyromètre à travers les équations suivantes :

$$\theta = \omega_\theta.\Delta t \qquad \phi = \omega_\phi.\Delta t$$

Avec : ω_θ et ω_ϕ les vitesses de rotation roulis et tangage mesurées dans un intervalle de temps Δt.

Afin de combiner ces angles, on utilise un filtre complémentaire qui, comme son nom l'indique, permet de compenser les problèmes de l'un des capteurs avec l'autre capteur. Il utilise donc l'avantage du gyromètre pour les fortes accélérations et compense sa déviation pour les faibles accélérations avec l'accéléromètre.

On donne ici l'équation du filtre qui est la suivante :

$$CF_{angle} = HP_{weight}.(CF_{angle} + Gyro_{angle}) + LP_{weight}.Accelero_{angle}$$

Avec :

- CF_{angle} : l'angle que l'on souhaite calculer avec le filtre complémentaire.
- $Gyro_{angle}$: l'angle calculé en se basant sur les mesures du gyromètre.
- $Accelero_{angle}$: l'angle calculé en se basant sur les mesures de l'accéléromètre.
- HP_{weight} (High Pass Weight) : constante qui représente le poids attribué au valeurs du gyromètre.
- LP_{weight} (Low Pass Weight) : constante qui représente le poids attribué au valeurs de l'accéléromètre.

Les deux constantes HP_{weight} et LP_{weight} sont déterminés expérimentalement. Leur somme doit être égale à 1 avec le poids de HP proche de 1 et celui de LP proche de 0.

2.4.2.2 Joystick

Notre joystick nous permet d'incliner la plateforme de différentes façons, il suffit de le diriger avec notre doigt.

Deux valeurs sont générées par le joystick représentant chacune un des deux axes. Grâce à une interface SPI, on récupère ces deux valeurs (chacune entre 0 et 1023) et on les étale entre $-90°$ et $90°$.

Ces deux angles sont équivalents aux deux angles envoyés depuis le simulateur de vol. Selon le mode de contrôle de la plateforme on choisira les uns ou les autres.

2.4.2.3 Switch

Le switch est tout simplement un bouton à deux états qui renvoie un « 1 » s'il est appuyé et un « 0 » s'il ne l'est pas.

On a choisi d'attribuer la valeur « 0 » au mode automatique et la valeur « 1 » au mode manuel. Ainsi, un couple d'angles est sélectionné selon l'état du bouton et sera par la suite traité pour être converti en commandes pour les servomoteurs.

2.4.2.4 Écran LCD

L'écran LCD que nous avons est un afficheur associé à un adaptateur I2C sur lequel nous avons affiché :

- Le tangage : celui récupéré de l'IMU sur la première ligne.
- Le roulis : celui récupéré de l'IMU sur la deuxième ligne.
- Le mode : automatique ou manuel sur la troisième ligne.

Ceci nous permet de visualiser en temps réel le mode dans lequel on se trouve ainsi que les angles d'inclinaison réels de la plateforme.

2.4.2.5 Gestion des données

Le rôle principal de ce nœud est de convertir les angles en commandes et de transmettre ces commandes sur le bus CAN.

Le MBED lit donc l'état du switch. Si l'état est en mode automatique, il lit les deux angles envoyés par le simulateur.

Si l'état du switch est sur le mode manuel, ces données ne seront pas lues et les angles pris en considération sont ceux du joystick.

Les deux angles passeront ensuite par la fonction de transfert vu dans le modèle cinématique de la plateforme pour être convertis en trois commandes pour les trois servomoteurs.

Ces trois commandes seront alors transmises sur le bus CAN sur trois trames différentes. La procédure d'envoi de données sur un bus CAN et celle de la lecture sont très similaires et sont résumées dans l'organigramme de la **Figure 2.14**.

Il est à noter que l'initialisation et le choix de la fréquence ne se fait qu'une fois au début du programme.

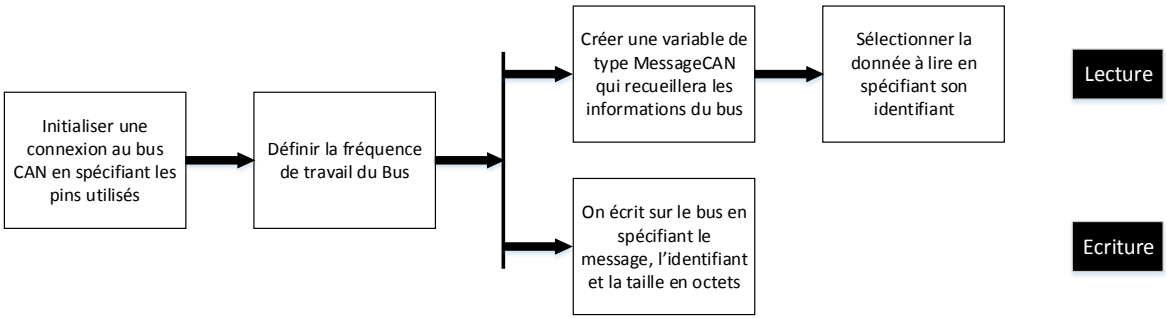


FIGURE 2.14 – Échange de données entre un MBED et un bus CAN

2.4.3 Nœud Commande

Le nœud de commande est chargé, à travers son MBED, de lire les commandes envoyées par le nœud précédent depuis le bus CAN et de les transmettre aux servomoteurs.

Avant de transmettre la première commande, une fonction d'initialisation des servomoteurs est appelée et tout au long du fonctionnement du système une fonction de sécurité pourra être déclenchée dans le cas où l'un des capteurs de fin de course est activé, ces derniers étant placés juste sous les bras des servomoteurs pour indiquer le dépassement de l'angle maximal permis.

Le schéma de la **Figure 2.15** permet de visualiser les différentes connexions du nœud ainsi que les types de connexions entre chacun des composants.

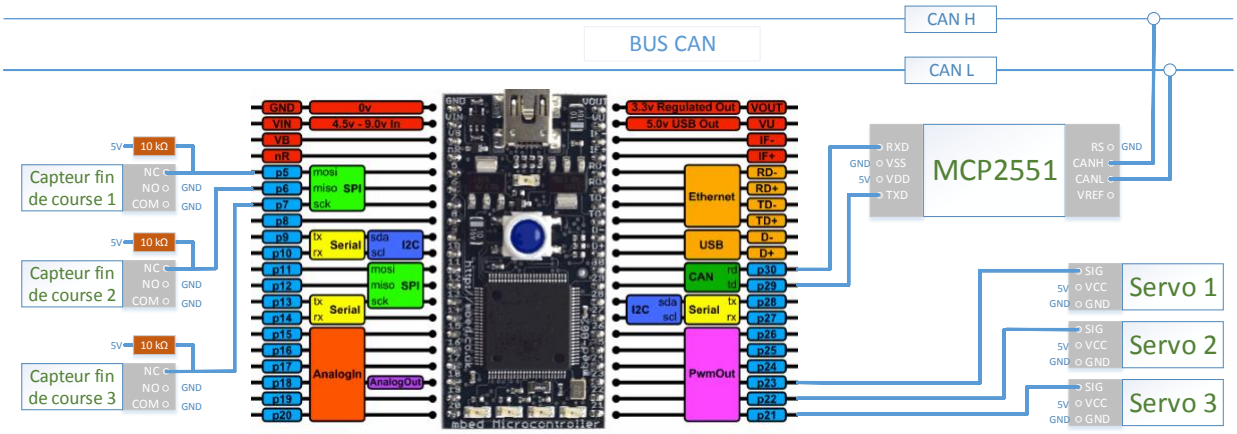


FIGURE 2.15 – Architecture du nœud commande

2.4.3.1 Commande des servomoteurs

Tout d'abord, les commandes sont récupérées du bus CAN par la même procédure expliquée dans le nœud précédant. Il suffit de sélectionner les trois identifiants correspondant à ces trois données.

Nos servomoteurs sont commandés par des impulsion PWM (modulation de largeur d'impulsions). Une période est d'abord définie, elle correspond à la période de rafraichissement de la commande du servomoteur. Durant cette période le signal envoyé est d'abord à l'état « 1 » pendant une certaine durée puis sur l'état « 0 » pour le reste de la période.

La durée de l'état haut défini l'angle que prendra le servomoteur et c'est justement la durée de l'état haut qui est transmise depuis le nœud précédent.

La période de nos servomoteurs est de 20 ms. Quant à la durée de l'état haut, elle varie entre 600 μs à 2650 μs. Il s'agit d'une variation linéaire qui correspond à la plage d'angles entre $-90°$ et $90°$.

2.4.3.2 Fonction d'initialisation

La fonction d'initialisation permet, comme son nom l'indique, d'initialiser la position des servomoteurs.

Avant que le système ne soit opérationnel, le nœud commande initialise les positions des servomoteurs. Il positionne tout d'abord les trois servomoteurs en position milieu puis, et ce pour chacun d'entre eux, il commence à abaisser l'angle du servomoteur jusqu'à ce qu'il touche son capteur de fin de course.

À ce moment-là, il relève le servomoteur de quelques degrés pour assurer le relâchement du bouton du capteur et enregistrera cet angle comme angle initial. Valeur dont la fonction de sécurité pourra se servir.

2.4.3.3 Fonction de sécurité

La fonction de sécurité est déclenchée dès que l'un des trois capteurs de fin de course est activé. Ce qui signifiera que le servomoteur correspondant a dépassé son angle maximal.

Dès lors, la fonction arrête la transmission des commandes aux servomoteurs et elle remet celui qui l'a déclenché à sa valeur initiale stockée depuis la fonction précédente.

2.4.3.4 Passage au standard LRU

Dans le but de faire correspondre notre travail sur le modèle réduit le plus possible au modèle réel afin qu'il soit transposable facilement par la suite, il nous a été demandé de travailler en standard LRU pour Line Reusable Unit. Cette méthode permet de traiter chaque composant à part et donc de remplacer facilement d'éventuelles pièces défectueuses.

Le modèle grandeur nature du simulateur de vol contient lui trois moteurs qui sont chacun commandé par un microcontrôleur.

Nous devons donc diviser notre nœud de commande en trois nœuds. Chaque nœud contiendra un MBED, un transceiver de ligne, un servomoteur et un capteur de fin de course. Chaque MBED lira du bus CAN la commande qui correspond à son servomoteur et renverra sur une autre trame la position réelle en cours du servomoteur.

2.4.4 Rétrospective des nœuds

La **Figure 2.16** résume l'activité des trois nœuds combinés en fonction du temps. Le processus décrit dans cette figure tourne en boucle tant que le système est en marche.

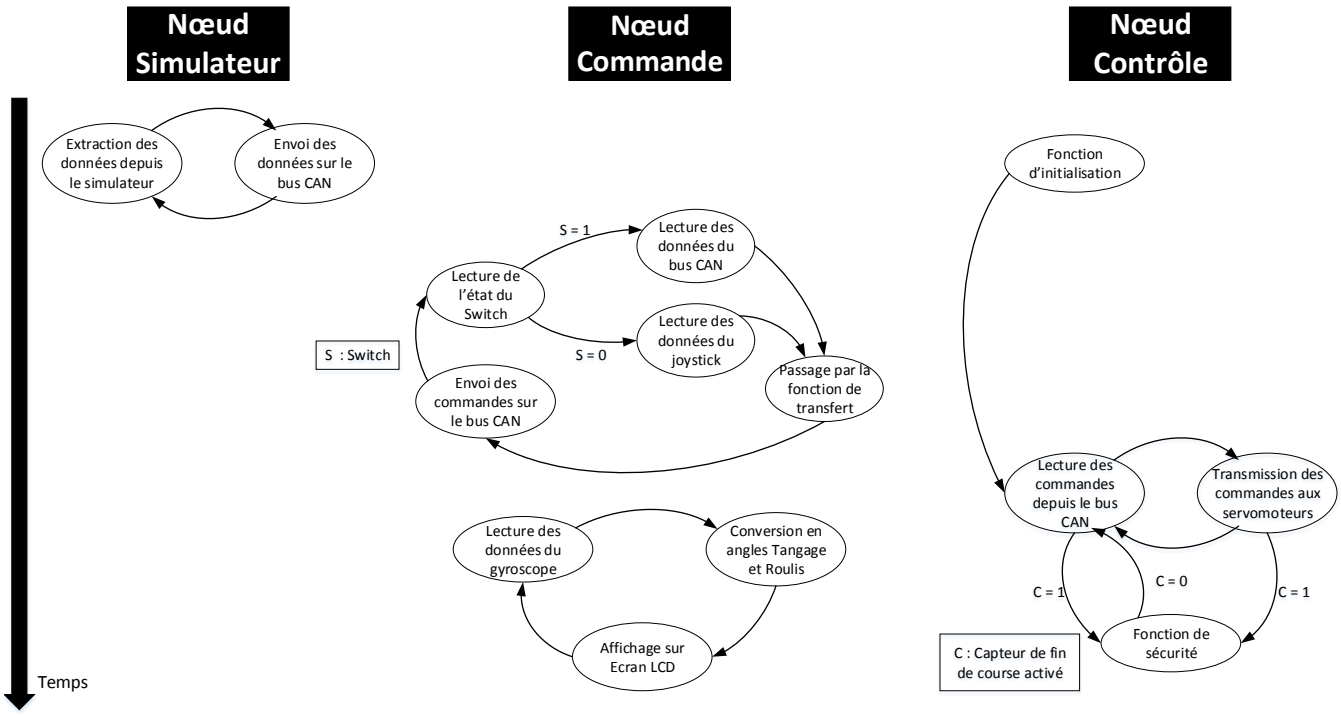


FIGURE 2.16 – Représentation globale des nœuds

2.4.5 Schéma électrique

Le schéma de la **Figure 2.17** montre les différentes connexions électriques qui permettent d'alimenter tout le circuit.

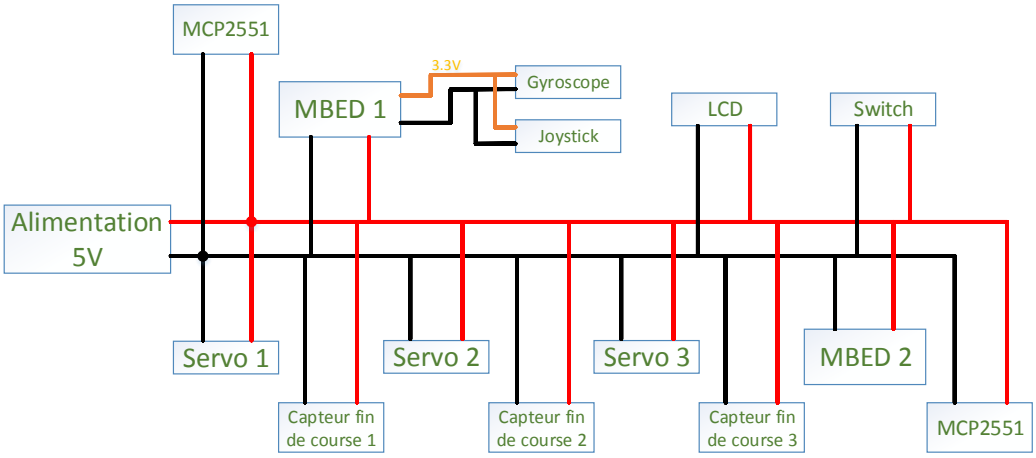


FIGURE 2.17 – Schéma électrique du système

2.5 Conclusion

Nous avons vu au cours de ce chapitre les détails de la conception et de l'étude de notre plateforme mobile.

Par la suite nous avons vu notre circuit électronique qui est autour de la plateforme et qui lui permet d'être mobile à travers les commandes que nous lui adressons.

Nous avons vu que le système est divisé en trois nœuds distincts communiquant tous à travers le bus CAN. Chaque nœud possède un rôle bien défini qu'on peut résumer par l'extraction des angles pour le nœud simulateur, le traitement de ces angles et la génération de commandes pour le nœud contrôle et enfin l'application de ces commandes pour le nœud commande.

Ce dernier nœud joue également un rôle très important qui est celui de veiller à ce que les servomoteurs ne dépassent pas des angles limites à travers sa fonction de sécurité. Dans le prochain chapitre nous allons essayer d'optimiser ce dernier nœud pour assurer une fiabilité maximale.

Chapitre 3

Migration vers un prototype basé sur FPGA

3.1 Introduction

Dans ce chapitre, nous allons nous concentrer sur l'amélioration du nœud commande en introduisant un FPGA.

Nous verrons d'abord le but du passage vers une architecture basée sur FPGA et nous expliquerons la procédure du changement du système.

Par la suite nous verrons en détails la nouvelle architecture en deux étapes successives.

3.2 Passage sur FPGA

Le nœud de commande joue un rôle très important. En plus d'être celui qui commande les servomoteurs et donc le mouvement de la plateforme, il est également chargé d'assurer la sécurité du système en veillant à ce que les servomoteurs ne dépassent pas les valeurs critiques.

Étant donné l'enjeu de ce nœud, il serait plus judicieux de l'optimiser au maximum. C'est la raison pour laquelle on souhaite passer d'une architecture basée sur un microcontrôleur à une architecture basée sur un FPGA.

En effet un FPGA étant bien plus proche du niveau physique et se basant sur une implémentation hardware, il nous offre de meilleures performances et permet d'avoir des temps de réponses minimales ce qui fait que notre système sera bien plus réactif.

En outre, le nœud de commande assure la sécurité du système, il serait donc plus raisonnable que cette fonction soit géré au plus bas niveau au lieu d'une fonction logicielle car si le système plante, cela pourrait avoir de grave conséquences sur la plateforme.

Le système global restera le même. Le changement n'interviendra que sur le nœud de commande où on remplacera le MBED par un FPGA. Ce dernier commandera les servomoteurs ainsi que les capteurs de fin de course associés comme on peut le voir dans la **Figure 3.1**.

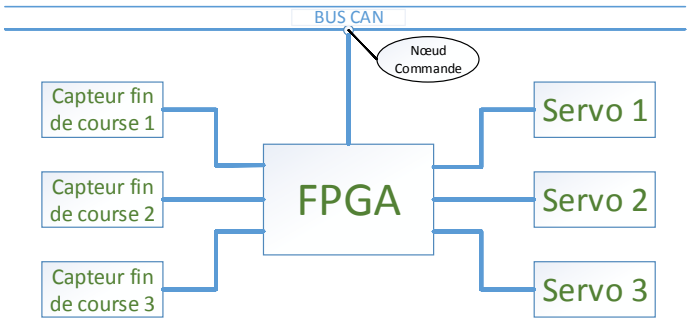


FIGURE 3.1 – Nouvelle architecture du nœud commande

Il faudra donc créer une brique pour la commande des servomoteurs, une brique de sécurité qui prendra en compte les capteurs de fin de course et surtout raccorder notre FPGA au bus CAN pour recevoir les trois commandes pour les servomoteurs.

Étant donné la difficulté de relier le FPGA directement au bus CAN, on se propose d'abord de laisser le MBED sur ce nœud pour qu'il se charge de récupérer les données depuis le bus CAN puis de les transmettre au FPGA. Une fois ces données sur le FPGA, on pourra vérifier que la commande des servomoteurs fonctionne correctement.

Par la suite, on supprimera définitivement le MBED et la communication entre le FPGA et le Bus CAN se fera à travers le circuit SJA1000 conçu dans ce but.

3.3 Architecture avec une interface I2C

La **Figure 3.2** montre la nouvelle architecture du nœud de commande. Le rôle du MBED est réduit à récupérer les données du bus CAN et à les transmettre au FPGA. Ce dernier s'occupe désormais de gérer la commande des servomoteurs et l'état des capteurs de fin de course.

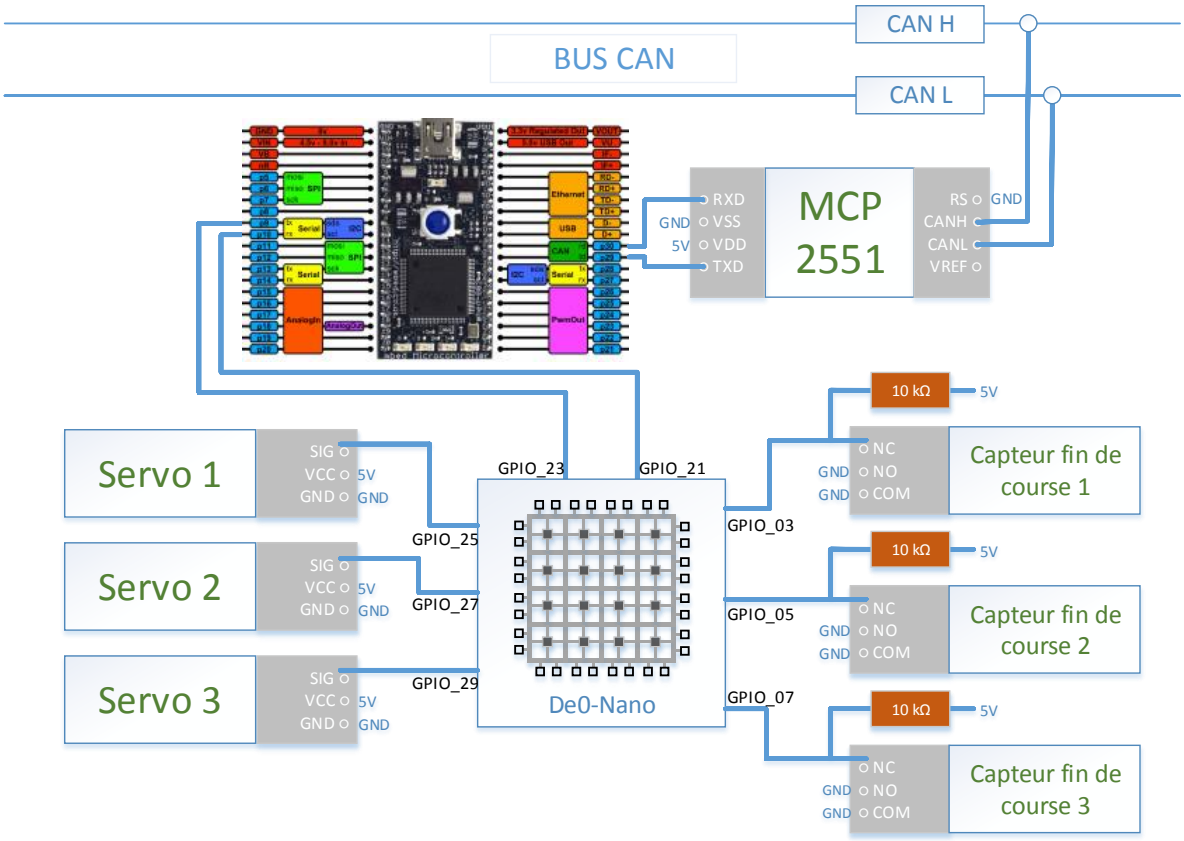


FIGURE 3.2 – Architecture avec un MBED

À l'intérieur du FPGA, on retrouve un système basé sur trois briques principales qui permettent l'échange avec le MBED d'un côté et la commande des servomoteurs de l'autre. La **Figure 3.3** montre le schéma en blocs détaillé du système sur FPGA.

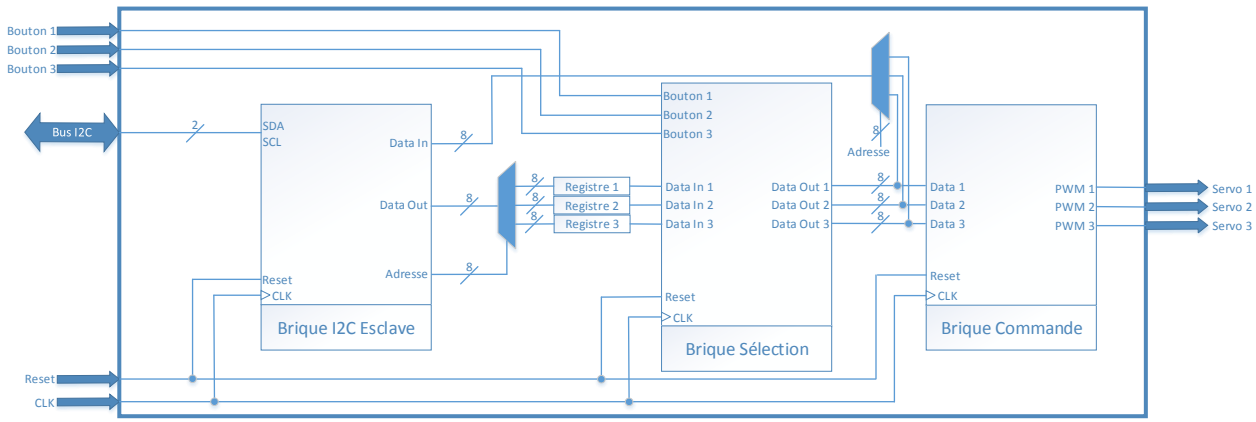


FIGURE 3.3 – Schéma en blocs sur FPGA

La première brique est une brique I2C esclave permettant de récupérer les angles transmis par le MBED et de lui renvoyer les angles réels instantanés.

La deuxième brique sélectionne l'état dans lequel se trouve le système. Soit l'état d'initialisation, soit l'état d'activation de la fonction de sécurité, soit l'état normal où les servomoteurs seront contrôlés par les commandes reçus du MBED.

Enfin, la troisième brique se charge de générer le signal PWM qui permet de commander les servomoteurs.

Nous allons maintenant regarder de plus près le fonctionnement de chaque brique.

3.3.1 Brique I2C

La brique I2C esclave est programmée pour reproduire le fonctionnement de ce protocole comme expliqué en annexe (Section A.2).

Le MBED récupère une commande depuis le bus CAN puis la transmet en I2C à cette brique avec une adresse particulière. La même procédure est suivie pour les deux commandes suivantes.

Ces trois commandes seront enregistrées sur des registres entre les deux premières briques selon leur adresse. En outre, et toujours selon l'adresse, les valeurs de sorties de la brique suivante seront renvoyées à la brique I2C pour qu'elles soient ramenées vers le MBED pour une éventuelle utilisation ultérieure.

Le **Tableau 3.1** montre les différentes adresses pour les trois commandes reçues et les trois commandes renvoyées.

3.3.2 Brique sélection

La brique sélection contrôle le fonctionnement du système et gère la sécurité. Elle se sert des données reçues par la brique précédente et de l'état des boutons (capteurs de fin de courses) et génère les sorties qui vont être envoyées à la brique de commande et également être renvoyées au MBED.

TABLE 3.1 – Adresses I2C entre MBED et FPGA

Adresse	Contenu
0x00	Commande servomoteur 1
0x01	Commande servomoteur 2
0x02	Commande servomoteur 3
0x03	Position servomoteur 1
0x04	Position servomoteur 2
0x05	Position servomoteur 3

Cette brique est basée sur une machine d'état principale dont les états engendrent d'autres machines d'états. La **Figure 3.4** montre la machine d'état principale.

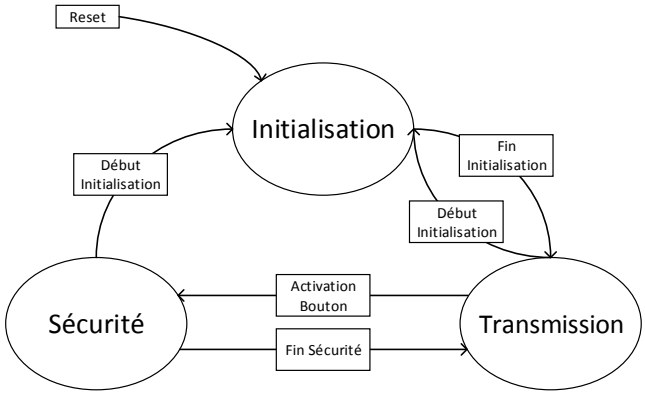


FIGURE 3.4 – Machine d'état principale du bloc Sélection

3.3.2.1 État d'initialisation

Cet état est déclenché au démarrage du système, lors de l'appui sur le bouton Reset ou si on appuie sur le bouton d'initialisation. Il suit la machine d'état de la **Figure 3.5**.

Tout d'abord, les trois servomoteurs sont mis sur leurs positions centrales en envoyant la valeur du milieu de l'intervalle à la brique suivante.

Ensuite, on cherche pour chaque servomoteur sa position limite qui permet de ne pas déclencher le capteur de fin de course tout en étant très proche de celui-ci. La procédure suit la machine d'état de la Figure 3.6. La valeur qui définit la position du servomoteur est décrémenté (ou incrémenté, selon l'orientation du servomoteur) tant que ce dernier n'a pas déclenché le capteur de fin de course. Une fois que cela arrive, on ajuste cette valeur en ajoutant une valeur équivalente à 10° à cette dernière valeur pour assurer le relâchement du capteur de fin de course et on gardera le résultat comme valeur minimale.

Une fois cette opération faite pour les trois servomoteurs, la tâche de cet état sera terminée et on pourra passer à l'état suivant.

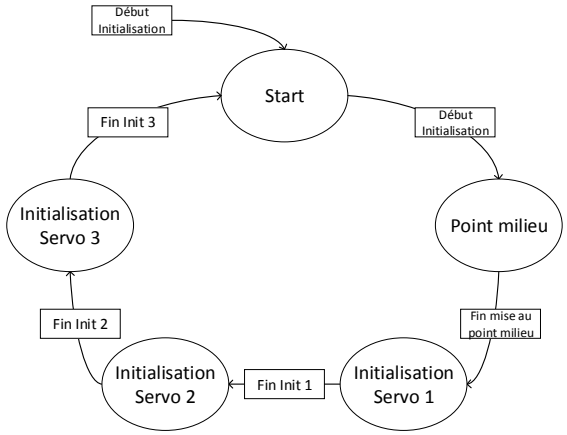


FIGURE 3.5 – Machine d'état d'initialisation

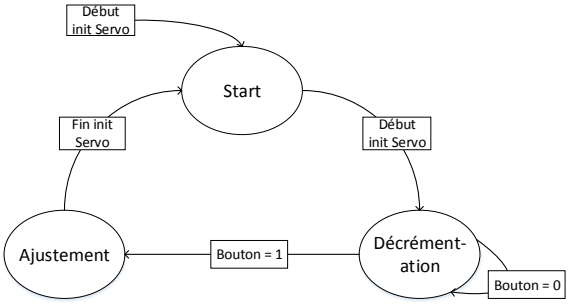


FIGURE 3.6 – Initialisation des servos

3.3.2.2 État de transmission

Une fois l'initialisation terminée, on passe à l'état « Transmission » dans lequel cette brique de sélection se contente de fournir au bloc suivant les commandes reçues depuis le bloc précédent. Ce qui permet aux servomoteurs de créer le mouvement de la plateforme.

La machine d'état reste dans cet état tant que l'un des deux événements suivants ne se déclenche pas :

- Début initialisation : un bouton est prévu permettant de refaire une initialisation à tout moment, dans ce cas l'état passe à l'état d'initialisation le temps que cette dernière se passe avant de revenir à l'état de transmission.

- Activation d'un bouton de fin de course : dans ce cas, on passe à l'état de sécurité puis on revient à l'état de transmission dès que le système sera rétabli.

3.3.2.3 État de sécurité

Le dernier état est l'état de sécurité. Il est déclenché par l'activation d'un capteur de fin de course, ce qui signifie que le servomoteur correspondant a touché ce capteur. De ce fait, ce servomoteur est ramené à sa valeur minimale enregistré dans l'état d'initialisation.

Une fois la remise à l'état minimale des servomoteurs concernés effectuée, on revient à l'état de transmission à moins que le bouton d'initialisation ne soit actionné pendant cet état.

3.3.3 Brique commande

Cette brique permet la commande des trois servomoteurs à travers la génération de trois signaux PWM.

Sachant que la période des servomoteurs est de 20 ms, on lance un compteur qui se réinitialise toutes les 20 ms. Il reste alors à définir la durée de l'état haut du signal.

On sait que la durée de l'état haut varie entre 0.5 ms et 2.5 ms (valeurs obtenues après des tests). Ce qui nous laisse un intervalle de variation de 2 ms codé sur 8 bits (entre 0 et 255), donc chaque valeur vaudra 7,8 μs.

Les intervalles de variation ne sont autres que les trois valeurs envoyées depuis la brique précédente qui sont elles-mêmes celles qui sont lues depuis le bus CAN.

Tout d'abord, un registre stocke une valeur correspondante à une durée de 0.5 ms, ce qui correspond à la durée minimale pour la commande d'un servomoteur. Ensuite, un compteur stocke la valeur correspondante à une durée de 20 ms, ce qui correspond à la période de rafraichissement des servomoteurs.

Une fois ces deux valeurs initialisées, on commence la partie du programme qui tourne en boucle.

À partir de la valeur reçue depuis la brique précédente (entre 0 et 255) on déduit une durée puis la valeur du compteur correspondante à cette durée que l'on ajoutera au registre précédent. Cette durée est en fait la durée complémentaire au 0.5 ms de départ pour aboutir sur la durée de l'état haut du signal PWM.

On lance alors le compteur qui sera réinitialisé toutes les 20 ms. La sortie sera mise à l'état « 1 » tant que ce compteur n'est pas arrivé à la valeur du registre. Une fois cette valeur atteinte, la sortie est mise à « 0 » pour le reste de la période.

Lors de la réinitialisation du compteur, le registre reprend la valeur correspondante à 0.5 ms avant que la nouvelle valeur ne lui soit additionné.

La **Figure 3.7** résume la procédure de la commande des servomoteurs par cette brique.

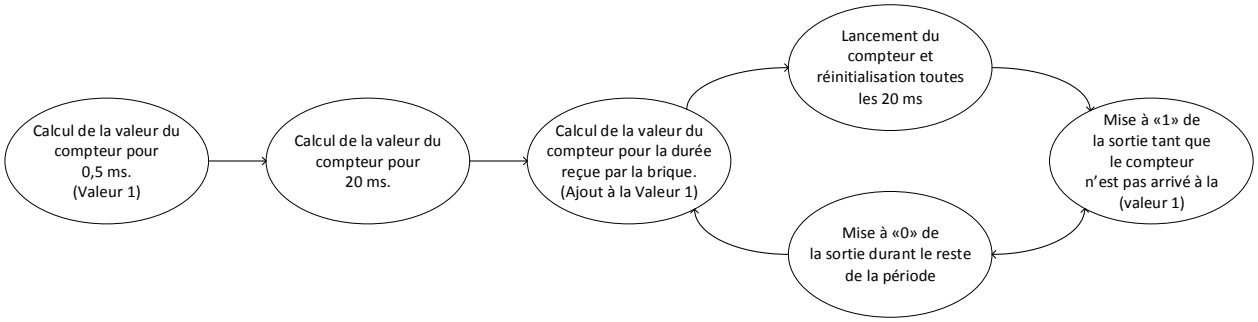


FIGURE 3.7 – Procédure de la commande des servomoteurs

3.4 Architecture avec le circuit SJA1000

Toujours en quête de meilleures performances, il serait intéressant de ne plus passer par un MBED pour se connecter au bus CAN.

Une alternative est d'utiliser un gestionnaire de protocole ou plus précisément un contrôleur CAN. Nous avons utilisé une carte électronique basé sur le contrôleur SJA1000. Cette carte nous a été fourni car la conception d'une telle carte nécessite de disposer de tous les outils et d'une expertise pour sa réalisation. Cette carte a été réalisé au laboratoire SATIE à Paris, elle est destinée à la conception d'une architecture LRU. Nous sommes les premiers utilisateurs de cette carte, avec l'aide de notre tuteur.

Le SJA1000 est un contrôleur CAN de type « standalone », c'est à dire qu'il ne nécessite pas un microcontrôleur externe mais il en intègre directement un. Il n'a besoin que d'un quartz qu'on lui connecte.

Ce circuit se connecte au FPGA d'un côté et au bus CAN de l'autre à travers un transceiver de ligne et permet l'échange de trames entre eux.

3.4.1 Adaptation de l'architecture

Tout d'abord, on s'intéresse à la nouvelle architecture, qui représente d'ailleurs l'architecture finale de notre projet. Dans le circuit de la **Figure 3.2**, il s'agit de remplacer le MBED par le circuit SJA1000. Le reste du circuit demeure inchangé.

Comme on peut le voir dans la **Figure 3.8**, le circuit SJA1000 comporte 28 connexions dont 22 seront utilisées.

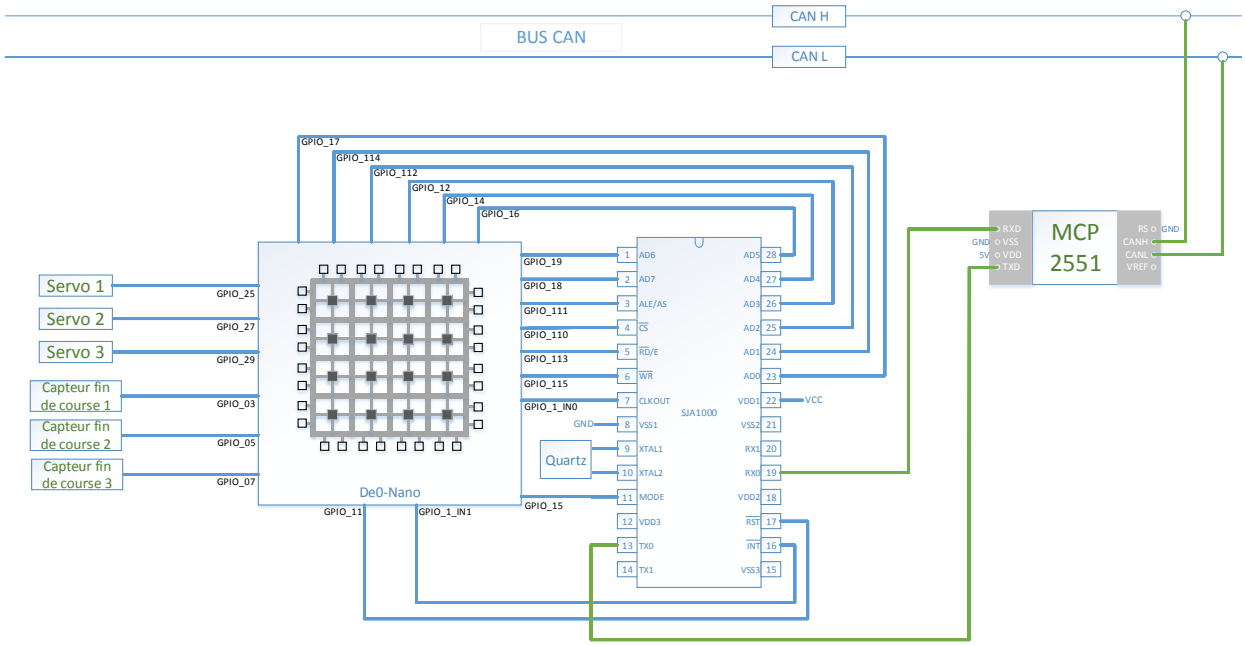


FIGURE 3.8 – Architecture avec un SJA1000

Les pins **VDD1** et **VSS1** sont respectivement reliés à VCC et GND pour l'alimentation du circuit. Les pins **TX0** et **RX0** sont connectés aux pins correspondant sur le MCP2551. Un quartz de 24 MHz est relié aux pins **XTAL1** et **XTAL2**. Les 16 autres pins sont reliés à des pins GPIO du FPGA.

Les pins de **AD0** à **AD7** représentent un bus servant à entrer des adresses ou des données. Le pin **ALE/AS** représente l'entrée du signal. Le pin $\overline{\textbf{CS}}$ permet de sélectionner le périphérique à utiliser. Les pins $\overline{\textbf{RD}}$**/E** et $\overline{\textbf{WR}}$ représentent respectivement le signal de lecture et le signal d'écriture.

Le pin **CLKOUT** est la sortie de l'horloge divisée du SJA1000 pour son microcontrôleur. Le pin **MODE** permet de choisir le mode « Intel » ou « Motorola ». Le pin $\overline{\textbf{INT}}$ est la sortie des interruptions et enfin le pin $\overline{\textbf{RST}}$ est le pin du reset.

À l'intérieur du FPGA, le schéma en bloc et le fonctionnement reste le même à la différence que la brique I2C est remplacée par la brique SJA1000 qui consiste en une machine d'état que l'on expliquera plus bas.

3.4.2 Modèle de programmation du SJA1000

Le modèle de programmation du SJA1000 repose sur la configuration de plusieurs registres avec chacun leur rôle.

Ces registres doivent être soigneusement remplis et dans un certain ordre afin de permettre la communication entre le FPGA et le bus CAN.

3.4.2.1 Procédure à suivre

La **Figure 3.9** montre le diagramme des étapes à suivre pour faire fonctionner le SJA1000.

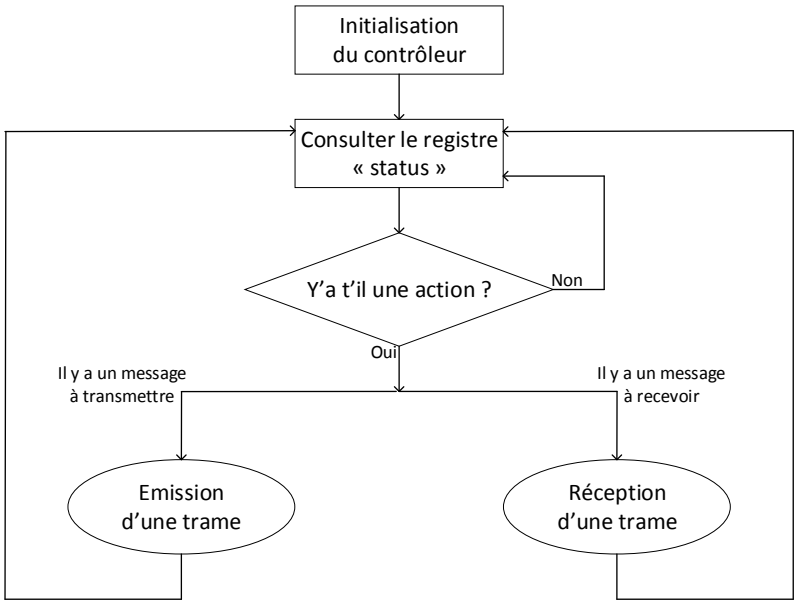


FIGURE 3.9 – Procédure d'utilisation du SJA1000

La première étape consiste en l'initialisation du contrôleur qui elle-même comprend plusieurs étapes que nous verrons plus bas dans la phase d'initialisation.

Une fois l'initialisation effectuée, on consulte le registre « status » et le timer d'envoi qui nous informeront s'il y a une activité en cours. On continue à consulter ce registre en boucle tant qu'il n'y a pas d'actions prévues.

Si un message à recevoir est signalé, on passe à l'étape de la réception de trame et si un message à envoyer est signalé, on passe à l'étape d'émission de trame. Tout cela consiste en la seconde phase dite phase d'utilisation.

3.4.2.2 Phase d'initialisation

La phase d'initialisation est une suite d'opérations à effectuer afin de préparer la communication en spécifiant certains paramètres. La suite des étapes est représentée dans la **Figure 3.10**.

On commence par désactiver les interruptions que nous n'allons pas utiliser dans notre projet. Puis on se met sur le mode « RESET » qui correspond au mode d'initialisation en mettant le bit CR.0 du registre « Control » à « 1 » .

Avant de continuer, on vérifie que nous sommes bien dans le mode « RESET » en consultant le registre « Control » , si ce n'est pas le cas, on recommence la procédure précédente jusqu'à ce que ça se fasse.

Ensuite on choisit la fréquence qui sort du pin **CLKOUT** sur le registre « Clock Divider ». On laisse la valeur par défaut 0x00 qui correspond à la moitié de la fréquence du quartz.

Par la suite, on configure le filtre d'acceptation en spécifiant les identifiants qui seront permis à la réception et qui correspondent à ceux que nous allons utiliser. Cette configuration se fait sur les registres « Acceptance Code » et « Acceptance Mask ».

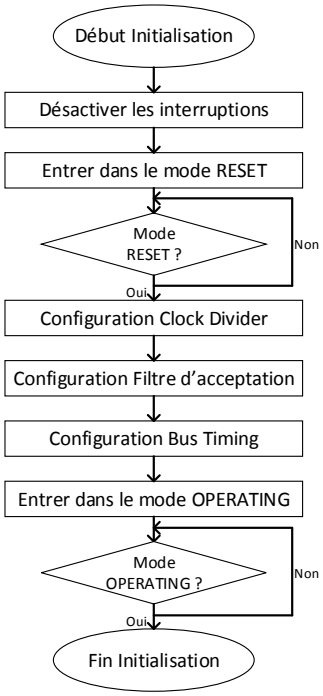


FIGURE 3.10 – Organigramme de la phase d'initialisation

On définit ensuite la fréquence du bus CAN qui est en l'occurrence de 1 Mbps sur les deux registres « Bus Timing 0 » et « Bus Timing 1 ».

À ce stade, l'initialisation est terminée et la dernière étape est de s'assurer qu'on passe au mode opérationnel pour pouvoir transmettre et recevoir des trames. Ceci est fait en mettant le bit **CR.0** du registre « Control » à 0. On s'assure qu'on est bien passé dans ce mode avant que la phase d'initialisation ne soit terminée.

3.4.2.3 Phase d'utilisation

Une fois l'initialisation effectuée, on peut passer à la phase d'utilisation après être passé en mode opérationnel.

Dans ce mode on peut soit transmettre des trames sur le bus, soit recevoir des messages transmis sur le bus. Le bit **SR.5** du registre « Status » nous permet de savoir s'il y a une trame en cours de transmission sur le bus et le bit **SR.4** du même registre nous permet de savoir s'il y a une trame reçue depuis le bus.

Dans le cas de la transmission, le buffer de transmission est rempli avec la trame que l'on souhaite envoyer puis une demande de transmission est émise. S'il n'y a pas d'erreurs, la trame sera alors transmise.

Dans le cas de la réception, le message est lu depuis le buffer de réception. Ce dernier est ensuite vidé et libéré pour la suite.

Les **Figures 3.11 et 3.12** montrent les organigrammes de la transmission et de la réception. Les deux buffers ont une taille de 10 octets chacun, huit pour les données et deux pour l'identifiant et la taille du message.

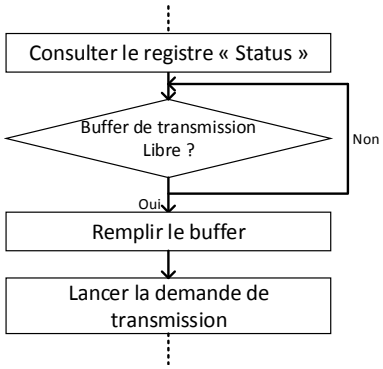


FIGURE 3.11 – Organigramme de transmission

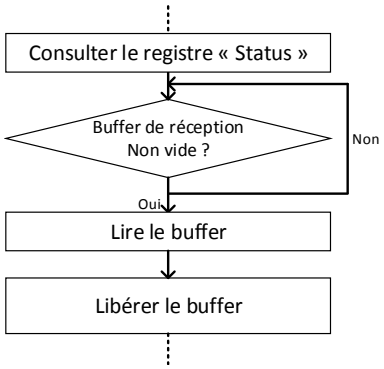


FIGURE 3.12 – Organigramme de réception

3.4.2.4 Écriture dans un registre

Que ce soit dans la phase d'initialisation ou dans la phase d'utilisation, nous avons eu besoin d'écrire dans un registre et de lire depuis un registre.

Le chronogramme de la **Figure 3.13** présente la procédure d'écriture dans un registre. On commence par mettre l'adresse du registre dans lequel on souhaite écrire dans les pins **AD0** à **AD7** puis on valide en mettant le pin **ALE** à « 1 » pendant un certain temps avant de le remettre à « 0 ».

On met ensuite le pin $\overline{\textbf{CS}}$ à « 0 » puis le pin $\overline{\textbf{WR}}$ à « 0 » pour pouvoir écrire. On met alors la valeur que l'on souhaite mettre dans le registre dans les pins de **AD0** à **AD7**.

On pourra ensuite désactiver le pin d'écriture $\overline{\textbf{WR}}$ et le pin $\overline{\textbf{CS}}$ et après un certain temps on pourra modifier la valeur des pins **AD0** à **AD7** afin de faire une nouvelle écriture.

Toutes ces opérations doivent être faites dans l'ordre et doivent respecter un temps minimal et/ou un temps maximal comme on peut le voir dans le chronogramme. Les valeurs des durées sont définies dans le Datasheet du SJA1000.

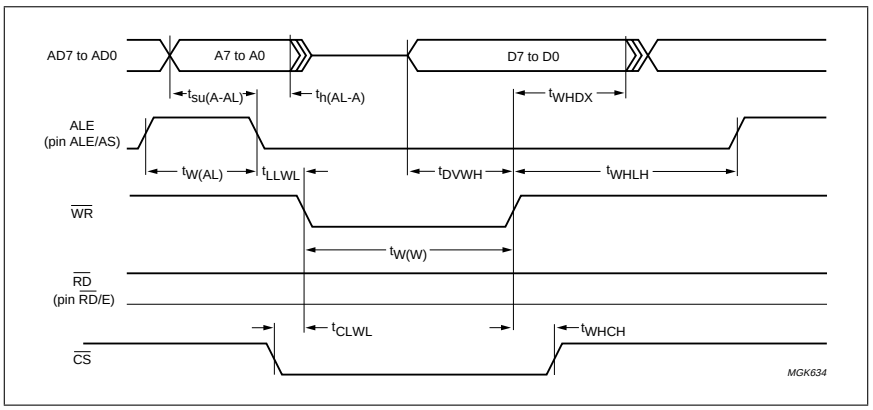


FIGURE 3.13 – Chronogramme d'écriture dans un registre

3.4.2.5 Lecture d'un registre

De même que pour l'écriture, le chronogramme de la **Figure 3.14** présente la procédure de lecture depuis un registre.

On commence par mettre l'adresse du registre que l'on souhaite lire dans les pins **AD0** à **AD7** puis on valide en mettant le pin **ALE** à « 1 » pendant un certain temps avant de le remettre à « 0 ».

On met ensuite le pin $\overline{\textbf{CS}}$ à « 0 » puis le pin $\overline{\textbf{RD}}$ à « 0 » pour activer la lecture. On pourra alors après un certain temps récupérer le contenu du registre dans les pins de **AD0** à **AD7**.

On pourra ensuite désactiver le pin de lecture $\overline{\textbf{CS}}$ et le pin $\overline{\textbf{RD}}$ et après un certain temps on pourra modifier la valeur des pins **AD0** à **AD7** afin de faire une nouvelle lecture.

Encore une fois, toutes ces opérations doivent être faites dans l'ordre et doivent respecter un temps minimal et/ou un temps maximal comme on peut le voir dans le chronogramme. Les valeurs des durées sont définies dans le Datasheet du SJA1000.

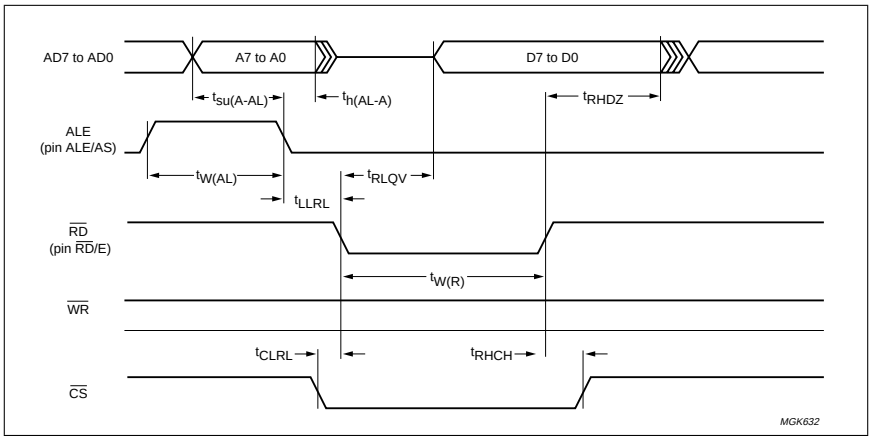


FIGURE 3.14 – Chronogramme de lecture depuis un registre

3.4.3 Système implémenté

Le système que nous avons implémenté dans la brique du SJA1000 sur le FPGA repose sur une machine d'état principale fonctionnant avec les procédures expliquées plus haut.

La **Figure 3.15** montre cette machine d'état qui commence par un état « Start » qui est l'état de départ et auquel on sera redirigé en cas de Reset.

On passe ensuite à l'état « Initialisation » qui reprend les étapes de la phase d'initialisation que l'on a vu plus haut.

On passe alors à l'état « Vérification » qui se charge, selon ce qu'il lit dans le registre « Status » et sur le timer d'envoi, de décider s'il passe à l'état « Transmission » ou « Réception » ou de rester dans le même état.

On revient ensuite à l'état « Vérification » une fois l'action terminée.

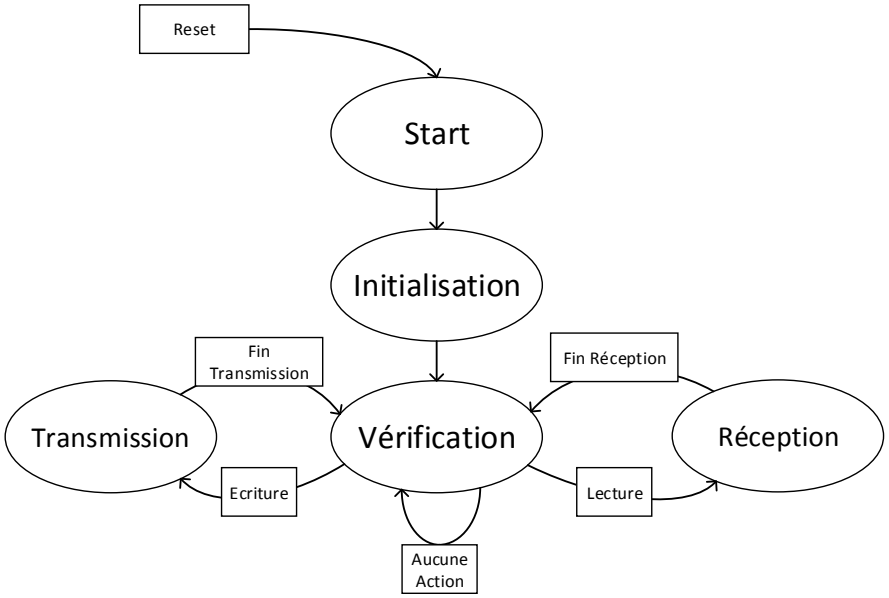


FIGURE 3.15 – Machine d'état du programme principale

3.5 Conclusion

Nous avons vu dans ce chapitre l'architecture finale de notre système après avoir optimisé le nœud de commande.

Nous avons introduit un FPGA pour remplacer le microcontrôleur et nous avons introduit le circuit SJA1000 qui nous a permis de communiquer avec le bus CAN directement depuis le FPGA.

Maintenant que nous avons notre prototype final, on passe, dans le chapitre suivant, à l'étude de ce système, de ses performances et de sa fiabilité ainsi que d'autres points concernant le déroulement du projet en général.

Chapitre 4

Analyse du projet

4.1 Introduction

Nous avons vu au cours des deux derniers chapitres la conception de notre système. Nous avons vu que nous avons d'abord construit un système basé sur microcontrôleur puis nous l'avons amélioré en introduisant un FPGA.

Dans ce dernier chapitre, nous allons étudier et évaluer les performances de notre système final et nous allons le comparer aux systèmes précédents.

En outre, ce chapitre nous permettra d'évoquer l'aspect gestion de projet de notre travail avant de terminer par développer les perspectives relatives à notre projet.

4.2 Analyse des performances

Afin d'évaluer les performances de notre système, nous devons observer comment il réagit par rapport aux commandes que l'on lui donne et également mesurer les temps de réponses des différents systèmes.

Nous allons commencer par introduire un nouveau nœud qui sera chargé de cette tâche puis nous présenterons les différents résultats.

4.2.1 Nœud analyse

Nous avons repris chacun des trois systèmes vus dans ce projet, et nous avons à chaque fois sorti l'IMU du nœud commande vers un nouveau nœud nommé nœud Analyse.

Nous avons effectué ce changement pour que les mesures de l'IMU soient en temps réel et coïncident avec les commandes envoyées.

On présente donc l'architecture de la **Figure 4.1** qui est la même que précédemment avec un nouveau nœud comprenant un MBED relié au bus CAN par un transceiver de ligne. Ce MBED gère l'IMU qui est posé sur la plateforme.

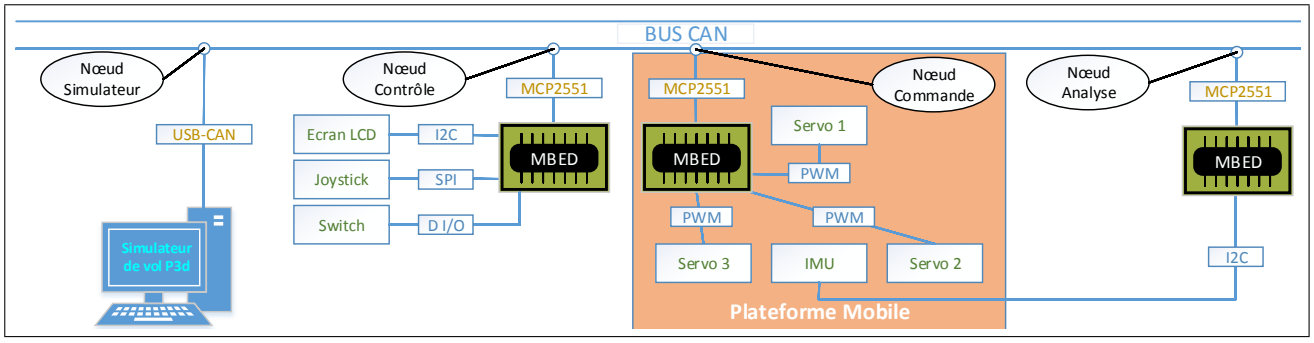


FIGURE 4.1 – Nouvelle architecture pour l'analyse

Le principe de notre analyse est d'envoyer différents types de commandes (sinusoïde, rampe, impulsion) depuis le nœud analyse. Ces commandes se traduiront au final par des rotations des servomoteurs qui donneront du mouvement à la plateforme.

La centrale inertielle se trouvant sur cette plateforme mesure les angles de la plateforme et enregistre dans un fichier les résultats obtenus.

Avec ces données, on peut tracer un graphe de la réponse du système et la comparer à la commande envoyée et ça nous permettra de voir si notre système est bien fidèle à la commande et aussi de calculer les temps de réponses de chaque système et de les comparer par la suite.

On se concentre pour notre analyse sur une rotation qui est le tangage. On envoie depuis le nœud Analyse des commandes sur le tangage sous différentes formes et on observe les réponses du système.

On envoie au total 3000 commandes espacées par une durée de $5ms$. De l'autre côté on relève également 3000 échantillons espacées par la même durée sur l'IMU, ce qui nous permettra de dessiner dans la même figure la commande et sa réponse et ainsi d'évaluer le temps de réponse du système.

4.2.2 Système sur microcontrôleur

On commence par l'analyse du premier système présenté dans le **Chapitre 2**. Pour rappel, ce système est basé sur une commande par microcontrôleur.

La **Figure 4.2** montre la réponse du système à une sinusoïde. On peut y remarquer que la réponse est assez proche de la commande avec un léger retard que l'on évaluera plus bas.

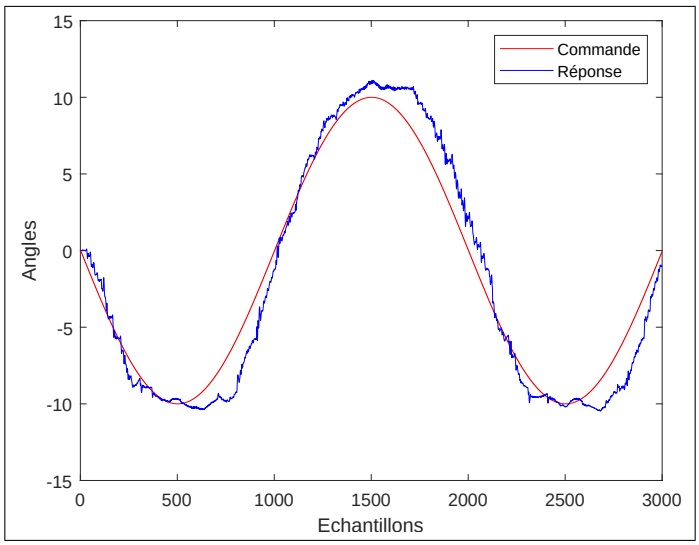


FIGURE 4.2 – Réponse du premier système à une sinusoïde

La **Figure 4.3** montre quant à elle la réponse pour une rampe. Là encore la réponse est assez fidèle à la commande avec quelques fluctuations de l'ordre d'un demi degré. Ceci s'explique par les vibrations provoquées par les servomoteurs, ce qui peut être résolu en prenant des servomoteurs plus performant.

Enfin, la **Figure 4.4** montre la réponse du système à une impulsion. La commande envoie un angle de 0° puis un angle de 16°. On peut alors évaluer le temps de réponse du système au moment où la réponse atteint la commande. On peut déterminer cette valeur sur la figure, elle est de $290ms$ (58 échantillons avec une période d'échantillonnage de $5ms$).

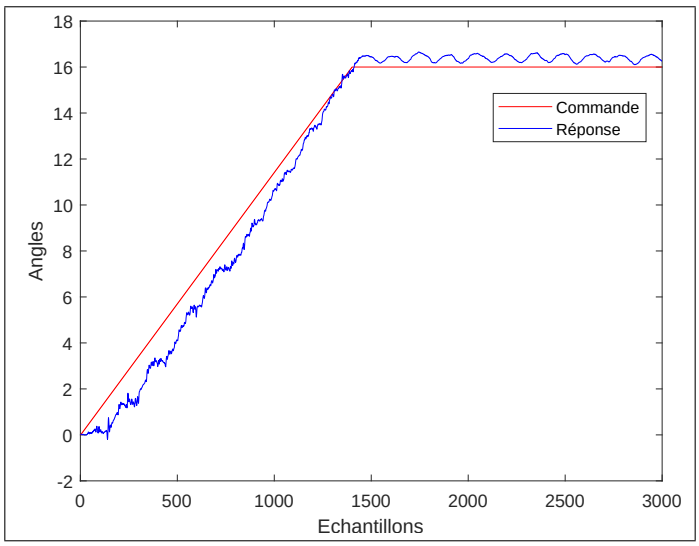


FIGURE 4.3 – Réponse du premier système à une rampe

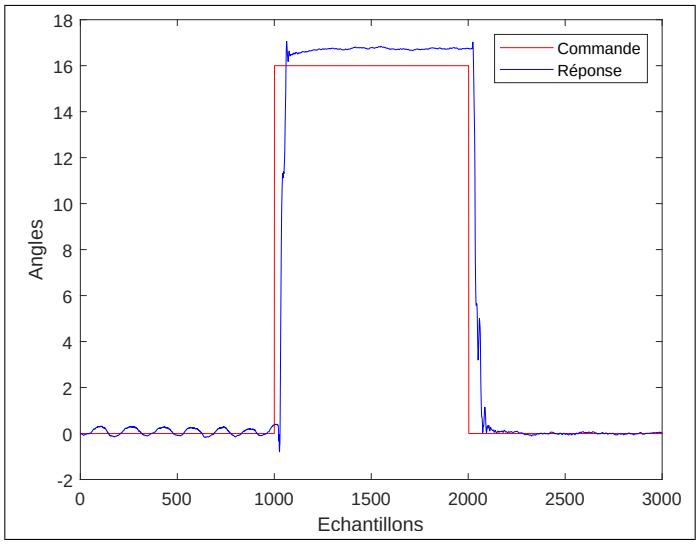


FIGURE 4.4 – Réponse du premier système à une impulsion

4.2.3 Système sur FPGA avec I2C

Le deuxième système est en fait le système intermédiaire entre le passage d'un microcontrôleur à un FPGA pour le nœud de commande. On effectue également des analyses sur ce système pour voir si nous avons une amélioration par rapport au précédent.

La **Figure 4.5** montre la réponse du système à une sinusoïde où l'on peut voir que cette réponse est plus fidèle à la commande par rapport au système précédent particulièrement lors des pics de la sinusoïde ou le retard n'est plus que de $300ms$ alors qu'il atteignait les $500ms$ dans le premier système.

La **Figure 4.6** montre la réponse du système à une rampe. On voit que la réponse est plus fidèle à la commande par rapport à celle du système précédent ($200ms$ par rapport à $350ms$ pour le système précédent). Une fois de plus on remarque l'impact des vibrations des servomoteurs sur la réponse du système.

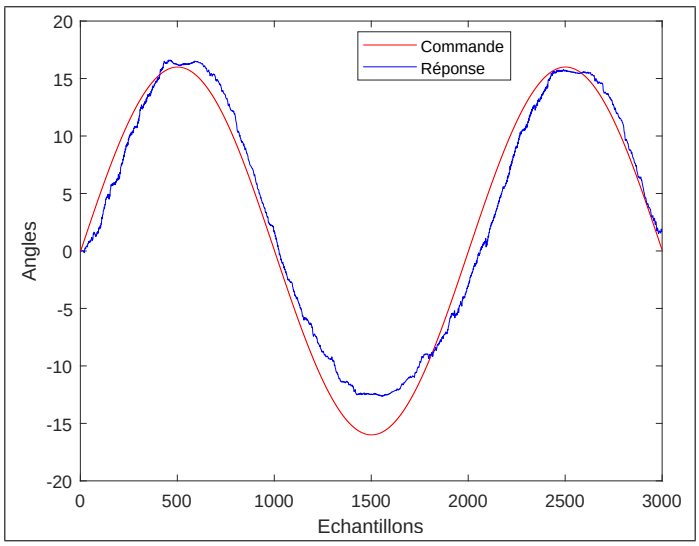


FIGURE 4.5 – Réponse du deuxième système à une sinusoïde

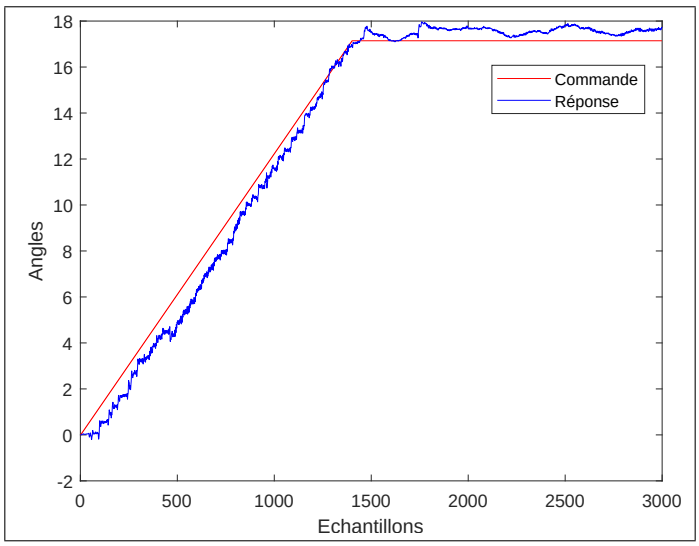


FIGURE 4.6 – Réponse du deuxième système à une rampe

Sur la **Figure 4.7**, on peut observer la réponse de notre deuxième système à une impulsion. On peut en déduire le temps de réponse de notre système qu'on évalue ici à $130ms$ (26 échantillons avec une période d'échantillonnage de $5ms$). Ce système est déjà plus performant que le précédent car le FPGA peut envoyer les commandes aux servomoteurs beaucoup plus vite que le MBED et que la génération des signaux PWM sur MBED est remplacée par une transmission des données à travers l'I2C.

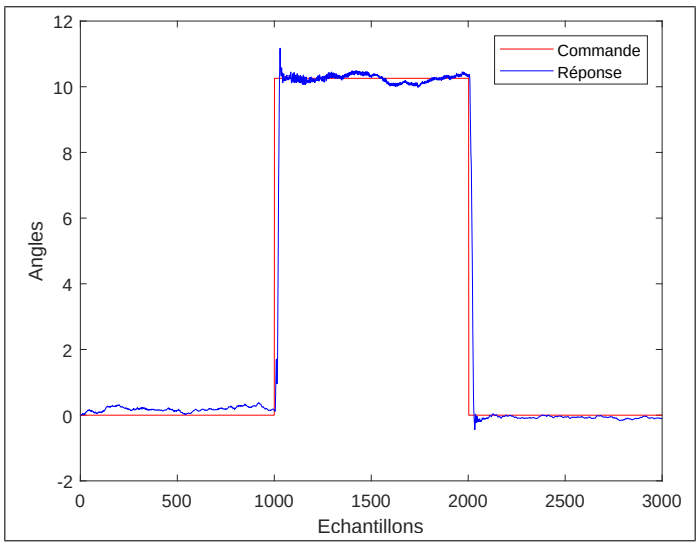


FIGURE 4.7 – Réponse du deuxième système à une impulsion

4.2.4 Système sur FPGA avec SJA1000

Pour ce dernier système, le MBED laisse place au SJA1000, ce qui promet un échange plus rapide de données avec le bus CAN. On s'attend donc à ce que la réponse soit meilleure par rapport aux systèmes précédents.

La **Figure 4.8** montre la réponse du système à une sinusoïde où l'on peut observer une amélioration considérable de la fidélité de la réponse à la commande avec un retard aux pics de la sinusoïde évalué à $125ms$, à part au niveau du pic inférieure à cause de la limitation des mouvements de la plateforme.

On observe également une amélioration dans la **Figure 4.9** représentant la réponse à une rampe. Elle est bien plus stable que pour les systèmes précédents et les fluctuations très faibles.

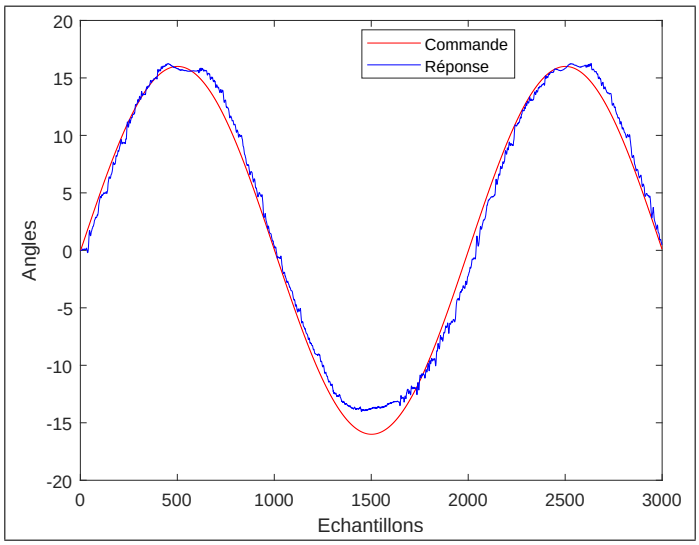


FIGURE 4.8 – Réponse du troisième système à une sinusoïde

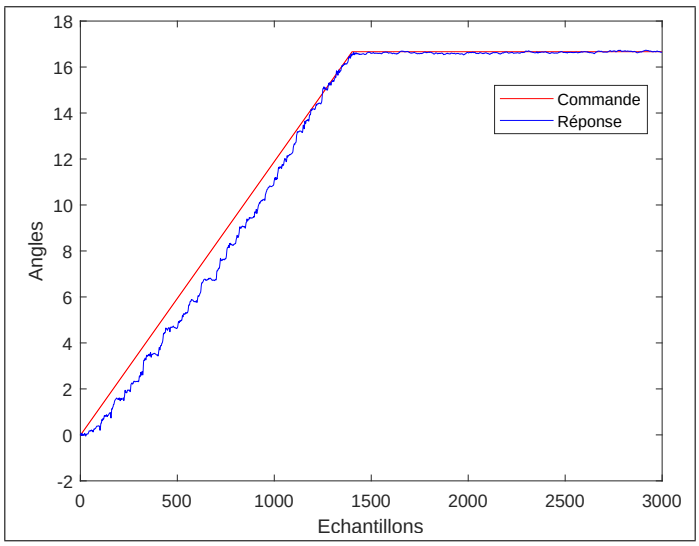


FIGURE 4.9 – Réponse du troisième système à une rampe

Concernant le temps de réponse de ce dernier système, la **Figure 4.10** montre la réponse du système à une impulsion où l'on peut évaluer le temps de réponse qui est ici de $95ms$ (19 échantillons avec une période d'échantillonnage de $5ms$).

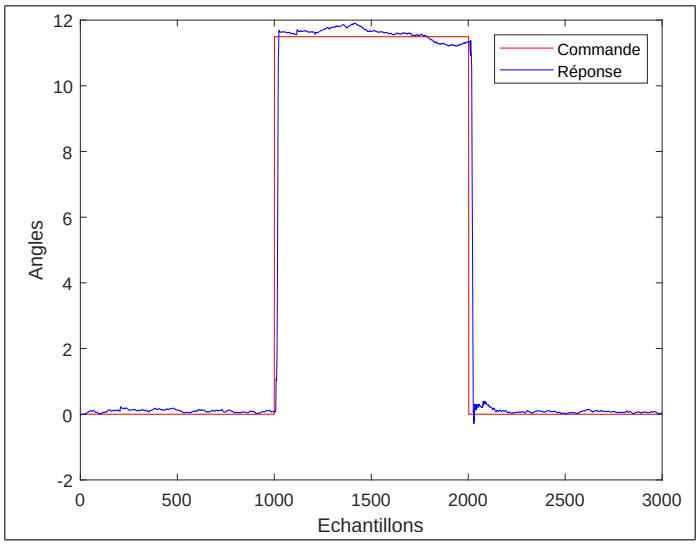


FIGURE 4.10 – Réponse du troisième système à une impulsion

4.3 Diagramme de Gantt

Nous avons vu dans le premier chapitre un outil nous permettant d'organiser nos tâches et de les visualiser dans le temps. Il s'agissait d'un diagramme de Gantt prévisionnel.

À la fin de notre projet, nous avons ajusté ce diagramme selon l'avancement réel de notre travail. La **Figure 4.11** montre le diagramme de Gantt effectif spécifiant les différences par rapport au diagramme initial qui ont eu lieu principalement dans la seconde partie du projet.

Initialement le projet devait se dérouler en première partie au niveau de l'École Nationale Polytechnique et en seconde partie en mobilité de deux mois à l'Université Paris Saclay. Globalement la première partie s'est déroulée comme prévue.

À la fin mars, et suite à l'accentuation de la pandémie du COVID-19, l'École a dû fermer ses portes et nous nous sommes mis en confinement. Nous avons été bloqué durant les deux premières semaines du mois d'Avril jusqu'à la confirmation de l'annulation de la mobilité où nous avons repris le travail à distance.

La suite du projet nous a pris un peu plus de temps que prévu. En effet le travail à distance nous a causé beaucoup de problèmes de synchronisations, sans oublier notre incapacité d'accéder à l'école. Nous avons donc manqué de matériel ce qui nous a beaucoup retardés.

En outre, nous avons eu besoin d'apporter des modifications à la plateforme mobile en imprimant de nouvelles pièces. Nous avons eu du mal à trouver une imprimante 3D et celle que l'on a finalement trouvé présentait des problèmes d'impression principalement pour les grandes pièces, ce qui fait que nous lui avons consacré beaucoup de temps pour avoir nos nouvelles pièces.

4.4 Perspectives

Au terme de notre projet de fin d'études, nous sommes arrivés à un résultat prometteur. Il reste néanmoins des améliorations à lui apporter pour qu'il soit encore plus performant avant de passer au modèle grandeur nature.

On cite ici quelques une des perspectives que nous envisageons :

- **Migration totale sur du FPGA :**

Après avoir remplacé le microcontrôleur du nœud de commande par un FPGA, nous avons remarqué une nette amélioration du temps de réponse. Il serait donc intéressant de faire de même pour le nœud contrôle.

On pourrait garder la même architecture en implémentant les deux nœuds sur un même FPGA. On aurait des performances considérablement optimisées et une meilleure exploitation du FPGA.

- **Implémentation du filtre de Washout :**

Les rotations de la plateforme étant physiquement limitées, il ne serait pas possible de simuler parfaitement le ressenti d'un pilote dans un avion.

Pour y remédier, il est possible d'utiliser le filtre de *Washout*[12] qui permet de générer des mouvements pour la plateforme en fonction des mouvements de l'avion afin de trouver un compromis entre les limitations de la plateforme et le ressenti qu'elle peut produire.

- **Passage au modèle grandeur nature :**

Maintenant que le système est validé sur modèle réduit, il est désormais possible de passer au modèle grandeur nature en faisant les adaptations nécessaires.

Il faudrait asservir les moteurs de la plateforme réelle dans un premier temps, pour ensuite pouvoir lui appliquer une version adaptée du système que nous avons conçu.

Nous envisageons de poursuivre nos études dans le master Systèmes Embarqués et Traitement de l'Information à l'Université de Paris Saclay ou nous sommes déjà admis. Nous pourrons alors accomplir ces différentes perspectives et apporter des améliorations diverses.

4.5 Conclusion

Ce chapitre nous a permis de faire une analyse globale des trois systèmes que nous avons conçu. Nous avons pu observer l'amélioration des performances en mesurant les temps de réponses de chaque système.

En outre nous avons vu les différents changements au niveau du déroulement du projet par rapport à ce qui était initialement prévu, et nous avons terminé par présenter les perspectives de notre travail.

	Tâche	Date de début	Date de fin	Durée
1	**Conception de la plateforme**	**09/02/2020**	**19/02/2020**	**9**
1.1	Modélisation de la plateforme sur SolidWorks	09/02/2020	13/02/2020	5
1.2	Impression des pièces avec une imprimante 3D	16/02/2020	19/02/2020	4
2	**Réalisation du circuit électronique : Nœud Commande**	**18/02/2020**	**26/02/2020**	**7**
2.1	Commande des servomoteurs avec un MBED	18/02/2020	20/02/2020	3
2.2	Implémentation de la fonction d'initialisation sur MBED	23/02/2020	24/02/2020	2
2.3	Implémentation de la fonction de sécurité sur MBED	25/02/2020	26/02/2020	2
3	**Réalisation du circuit électronique : Nœud Simulateur**	**27/02/2020**	**11/03/2020**	10
3.1	Etude du SDK Simconnect	27/02/2020	02/03/2020	3
3.2	Récupération d'informations depuis le simulateur	03/03/2020	04/03/2020	2
3.3	Création de l'interface graphique	05/03/2020	08/03/2020	2
3.3	Transfert des données sur le bus CAN	09/03/2020	11/03/2020	3
4	**Réalisation du circuit électronique : Nœud Contrôle**	**12/03/2020**	**23/03/2020**	8
4.1	Choix du mode (manuel ou automatique)	12/03/2020	16/03/2020	3
4.2	Récupération des angles du joystick	17/03/2020	18/03/2020	2
4.3	Récupération des angles du gyroscope, affichage sur LCD	19/03/2020	23/03/2020	3
5	**Réalisation de la fonction de transfert**	**24/03/2020**	**16/04/2020**	8
5.1	Etude du modèle cinématique de la plateforme	24/03/2020	29/03/2020	4
5.2	Implémentation de la fonction de transfert sur MBED	30/03/2020	01/04/2020	3
5.4	Passage en logique LRU	02/04/2020	02/04/2020	1
	Arrêt imprévu suite au COVID-19, annulation de la mobilité	05/04/2020	16/04/2020	10
6	**Migration sur FPGA : connexion en I2C avec MBED**	**19/04/2020**	**30/04/2020**	10
6.1	Conception d'une brique I2C sur FPGA	19/04/2020	21/04/2020	3
6.2	Connexion du FPGA au MBED en I2C	22/04/2020	26/04/2020	3
6.3	Brique d'initialisation et brique de sécurité	27/04/2020	28/04/2020	2
6.4	Brique de commande des servomoteurs	29/04/2020	30/04/2020	2
7	**Migration sur FPGA : connexion en CAN avec le SJA1000**	**03/05/2020**	**21/05/2020**	15
7.1	Etude du circuit SJA1000	03/05/2020	10/05/2020	6
7.2	Brique d'écriture et de lecture des registres	11/05/2020	14/05/2020	4
7.3	Envoie et récéption de trames du Bus CAN	17/05/2020	21/05/2020	5
8	**Reconstruction du modèle mécanique**	**24/05/2020**	**02/06/2020**	8
8.1	Ajustement de la modélisation sur SolidWorks	24/05/2020	25/05/2020	2
8.2	Impression des nouvelles pièces sur imprimante 3D	26/05/2020	02/06/2020	6
9	**Finalisation du projet**	**03/06/2020**	**18/06/2020**	17
9.1	Analyse des temps de réponses	03/06/2020	04/06/2020	2
9.2	Rédaction du mémoire	07/06/2020	18/06/2020	10
9.3	Préparation de la présentation	21/06/2020	25/06/2020	5

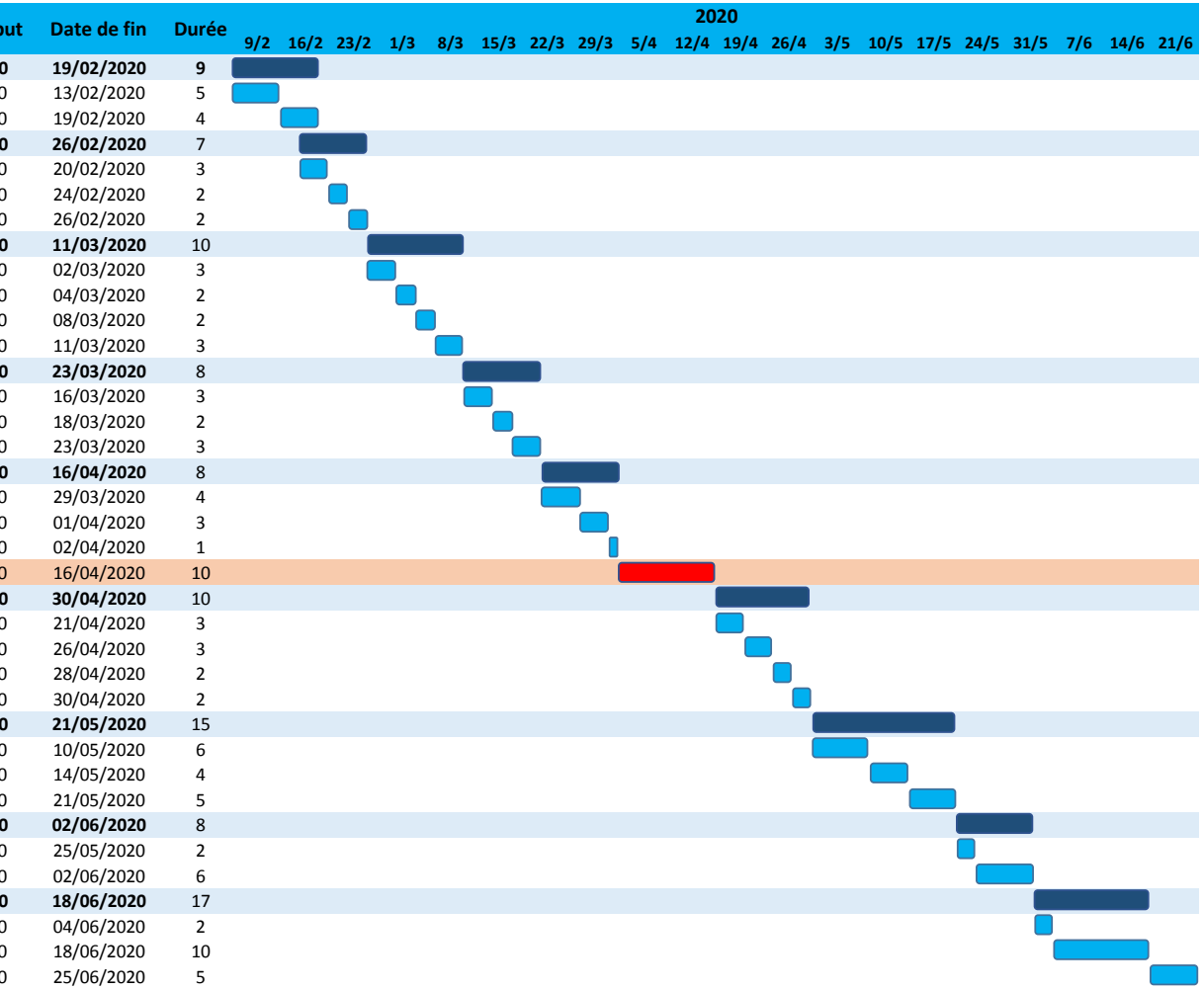


FIGURE 4.11 – Diagramme de Gantt effectif

Conclusion générale

Les simulateurs de vol représentent un enjeu capital dans la formation des pilotes. Ils permettent de rendre le pilotage plus accessible et d'offrir des formations de qualité. Ils permettent également de simuler des situations exceptionnelles et de préparer les pilotes à ces dernières, ce qui aura un impact positif sur l'aspect sécurité et économique. Pour qu'un simulateur soit performant au plus haut point, la cabine doit recréer le plus précisément possible le cockpit d'un avion pour que le pilote soit totalement immergé dans le vol. En outre, le pilote devrait ressentir les secousses et les différents mouvements d'un avion en vol afin de se rapprocher le plus possible de la réalité.

Dans notre projet de fin d'études, on a étudié un modèle de plateforme dynamique sur laquelle reposerait la cabine de pilotage d'un simulateur de vol. Ensuite nous sommes passé à la conception d'un modèle réduit de plateforme et à la réalisation des circuits électroniques autour d'elle. Cela est réalisé afin d'apporter des déplacements en fonction des mouvements du modèle de l'avion dans le simulateur logiciel.

Dans le premier chapitre nous avons vu un aperçu global sur le sujet, ce qui nous a permis de comprendre le but du projet et la procédure de sa conception tout en présentant les outils qui ont été nécessaires pour sa réalisation. Le deuxième chapitre était consacré à la modélisation et à la conception de la plateforme dynamique puis à la réalisation du premier système électronique, basé sur des microcontrôleurs, reliant la plateforme au simulateur de vol.

Dans le troisième chapitre on s'est consacré au second système basé sur FPGA. Nous avons d'abord expliqué les raisons qui nous ont menées à passer vers ce nouveau système puis nous sommes passés à sa réalisation en deux parties. Dans le dernier chapitre, nous avons d'abord analysé les performances de notre système avant d'analyser notre projet en général pour terminer par évoquer les perspectives envisagées à notre travail.

À la fin de notre travail, nous avons observé des résultats prometteurs, la plateforme suivait bien les mouvements de l'avion dans le simulateur et les temps de réponses, particulièrement ceux du système final, ont été satisfaisant et nous ont donc permis d'envisager d'exporter ce système à un modèle grandeur nature de simulateur de vol en faisant les adaptations nécessaires.

Bibliographie

[1] Marcel BERNARD. Real Learning through Flight Simulation The ABCs of ATDs [en ligne]. FAA Safety Briefing, 2012 [consulté le 17/02/2020].
Disponible sur :
https ://www.faa.gov/news/safety_briefing/2012/media/SepOct2012ATD.pdf.

[2] Emilie SABRIE. Analyse d'un mécnaisme de simulation de vol sphérique et son controle en temps réel [en ligne]. Mémoire de maitrise science : génie mécanique. Faculté des études supérieures de l'Université Laval., [consulté le 20/02/2020].
Disponible sur :
https ://robot.gmc.ulaval.ca/fileadmin/documents/Memoires/emilie_sabrie.pdf.

[3] Nicolas POULIOT. Analyse, optimisation et conception de mécanismes de simulation de mouvement à trois degrés de liberté [en ligne]. Mémoire de maitrise science : génie mécanique. Faculté des études supérieures de l'Université Laval., [consulté le 21/02/2020].
Disponible sur :
https ://robot.gmc.ulaval.ca/fileadmin/documents/Memoires/nicolas_pouliot.pdf.

[4] Assia SAFSAFI. Controleur CAN sur une carte DE2 d'ALTERA [en ligne]. Mémoire d'ingéniorat : Electronique, Ecole Nationale Polytechnique., [consulté le 23/02/2020].
Disponible sur :
http ://mt.biblio.intranet.enp.edu.dz/pub/Pfe/Electronique/Annee_2011/SEFSAFI.Assia /SEFSAFI.Assia.pdf.

[5] Patrice KADIONIK. LE BUS CAN [en ligne]. Bordeaux : Ecole nationale supérieure, 2001, 32 p., [consulté le 25/02/2020].
Disponible sur :
http ://geea.org/IMG/pdf/canbus_enseirb.pdf.

[6] Walid TALABOULMA. Conception d'un simulateur de vol matériel ZLIN142 [en ligne]. Mémoire d'ingéniorat : Electronique, Ecole Nationale Polytechnique., [consulté le 19/03/2020].
Disponible sur :
http ://mt.biblio.intranet.enp.edu.dz/pub/Pfe/Electronique/Annee_2012/TALABOULMA .Walid/TALABOULMA.Walid.pdf.

[7] Tran LONG. Data Fusion with 9 Degrees of Freedom Inertial Measurement Unit To Determine Object's Orientation [en ligne]. Senior Project : Electrical Engineering Department. California Polytechnic State University., [consulté le 19/03/2020].
Disponible sur :
https ://digitalcommons.calpoly.edu/cgi/viewcontent.cgi ?referer=https ://www.google .com/&httpsredir=1& article=1422&context=eesp.

[8] D. Patel, R. Kalla, H. Tetik, G. Kiper, and S. Bandyopadhyay. Computing the safeworking zone of a 3-RRS parallel manipulator [en ligne]. In : Wenger, P., Flores, P. (eds.) NewTrends in Mechanism and Machine Science. pp. 113–120. Springer InternationalPublishing, Cham (2017), [consulté le 17/02/2020].
Disponible sur :
https ://www.researchgate.net/publication/305939925.

[9] Wang J Li J and et al. Chou W. Inverse kinematics and dynamics of the 3-RRS parallel platform [en ligne]. In : Proceedings 2001 ICRA IEEE international conference on robotics and automation , Seoul, Korea, 21–26 May 2001, pp. 2506–2511. IEEE., [consulté le 17/02/2020].
Disponible sur :
https ://www.researchgate.net/publication/3902626.

[10] D. Tomaszewski, J. Rapinski, and R. Pelc-Mieczkowska. Concept of AHRS Algorithm Designed for Platform Independent Imu Attitude Alignment [en ligne]. In : Geodesy and Geoinformatics, 104(1). pp 33–47., [consulté le 19/03/2020].
Disponible sur :
https ://www.researchgate.net/publication/322668267_Concept_of_AHRS_Algorithm_ Designed_for_Platform_Independent_Imu_Attitude_Alignment.

[11] Terasic Technologies Inc. DE0-Nano User Manual [en ligne]. Taiwan : 2013, [consulté le 19/04/2020].
Disponible sur :
https ://www.terasic.com.tw/attachment/archive/941/DE0-Nano-SoC_User_manual.pdf .

[12] Asadi Houshyar, Mohamed Shady, and Nelson Kyle. Human Perception-Based Washout Filtering Using Genetic Algorithm [en ligne]. In : International Conference on Neural Information Processing, ICONIP 2015, pp.401-411., [consulté le 24/03/2020].
Disponible sur :
https ://www.researchgate.net/publication/304452514 .

[13] Philips Semconductors. SJA1000 Stand-alone CAN controller [en ligne]. [consulté le 03/05/2020].
Disponible sur :
https ://www.nxp.com/docs/en/data-sheet/SJA1000.pdf.

[14] NXP Semiconductors. UM10360 LPC176x/5x User manual [en ligne]. [consulté le 16/02/2020].
Disponible sur :
https ://www.nxp.com/docs/en/user-guide/UM10360.pdf.

[15] SYSTEC electronic AG. USB-CANmodul System Manual [en ligne]. [consulté le 16/02/2020].
Disponible sur :
https ://www.systec-electronic.com/fileadmin/Redakteur/Unternehmen/Support/Downloadbereich/Handbuecher/L-487e_02_05_USB-CANmodul.pdf.

[16] Microchip Technology Inc. MCP2551 High-Speed CAN Transceiver [en ligne]. [consulté le 16/02/2020].
Disponible sur :
http ://ww1.microchip.com/downloads/en/devicedoc/21667e.pdf.

[17] NOVA FLY. Dossier technique [en ligne]. [consulté le 09/02/2020].
Disponible sur :
http ://www.novafly.fr/docprodB.html [accès pour clients].

[18] STMicroelectronics. L3GD20H. [consulté le 19/03/2020].
Disponible sur :
https ://www.pololu.com/file/0J731/L3GD20H.pdf.

[19] STMicroelectronics. LSM303DLHC. [consulté le 19/03/2020].
Disponible sur :
https ://www.st.com/resource/en/datasheet/DM00027543.pdf.

Annexe A

Protocoles de communications

A.1 Bus CAN

A.1.1 Présentation du bus

Le bus CAN (Controler Area Network) est un protocole de communication série basé sur le multiplexage sur lequel plusieurs calculateurs peuvent communiquer. Son principal avantage c'est qu'il permet de connecter un nombre important de calculateurs et de les faire communiquer à haut débit et haute fiabilité. Son concept de multiplexage permet d'économiser un coût non négligeable en ce qui concerne le câblage.

Un bus CAN consiste en deux fils représentant deux canaux : un canal High et un canal Low comme on peut le voir dans la **Figure A.1**, terminé par des résistances dont la valeur dépend de la longueur de la ligne (en général 120 Ω). Chaque connexion à un périphérique représente un nœud. Chaque nœud est un maitre au sein du réseau, la notion d'esclave n'existe pas dans ce bus.

Ainsi tous les nœuds peuvent communiquer à tour de rôle selon une topologie bidirectionnelle half duplex. Chaque message présent dans le canal possède un identifiant unique et la priorité est donnée au message avec l'identifiant le plus haut.

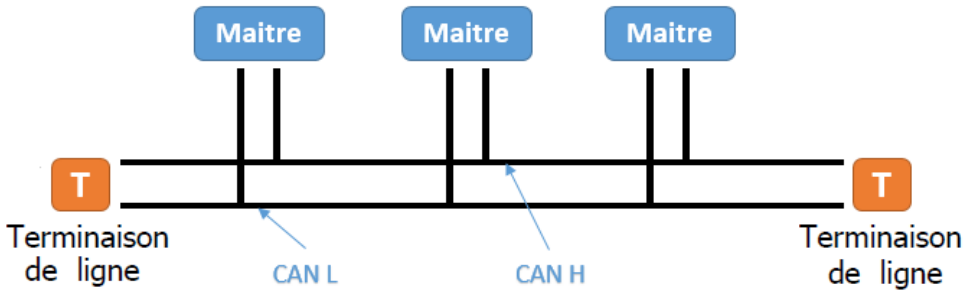


FIGURE A.1 – Architecture du bus CAN

Le bus CAN peut fonctionner à plusieurs débits, il est important de veiller à travailler avec le bon débit selon la longueur du bus. Le **Tableau A.1** résume les distances maximales entre deux nœuds selon le débit.

La technique de transmission des bits sur un bus CAN est dite différentielle : les deux états logiques « 0 » et « 1 » sont déterminés par une différence de potentiel mesurée sur les deux canaux (CAN High et CAN Low).

Table A.1 – Longueur maximale du bus selon le débit

Débit (kbits/s)	Distance (m)
1000	30
800	50
500	100
250	250
125	500
20	2500
10	5000

Les nœuds sont câblés sur le bus sur le principe d'un « ET logique » ce qui veut dire qu'en cas d'émission simultanée de deux nœuds, la valeur 0 écrase la valeur 1. On dit donc que le « 0 » est le bit dominant et le « 1 » est le bit récessif.

L'encodage utilisé dans le bus CAN est le NRZ (non-retour à zéro).

Deux types de bus existent, à savoir, le CAN low speed qui travaille à un débit inférieur à 125 kbits/s et le CAN high speed qui travaille à un débit supérieur à 125 kbits/s et jusqu'à 1 Mbits/s.

Selon le type du CAN, le nœud doit appliquer différentes tensions sur chaque canal pour avoir des différences de potentiels spécifiques à des niveaux logiques « 0 » ou « 1 ». Le **Tableau A.2** résume les différences entre les deux types.

Table A.2 – Potentiel des niveaux logiques

Paramètres	CAN Low Speed	CAN Hihg Speed
Débit	<125 kbits/s	>125 kbits/s
Niveau récessif « 1 »	CAN H = 1.75 V CAN L = 3.25 V différence = -1.5 V	CAN H = 2.5 V CAN L = 2.5 V différence = 0 V
Niveau dominant « 0 »	CAN H = 4 V CAN L = 1 V différence = 3 V	CAN H = 3.5 V CAN L = 1.5 V différence = 2 V

Une trame de données transporte des données dans le bus. Elle est composée de sept champs différents :

- Début de trame SOF (Start Of Frame) (1 bit) : drapeau constitué d'un bit dominant qui indique le début d'un échange.

- Champ d'arbitrage (12 bits pour une trame standard ou 32 bits pour une trame étendue) : l'identificateur de la trame est codé dans ce champ. Cet identificateur sert à définir la priorité entre les trames.

- Champ de commande (6 bits) : détermine le nombre d'octets de données contenus dans le champ suivant (champ de données).
- Champ de données (0 à 64 bits) : la donnée peut varier entre 0 et 8 octets.

- Champ de CRC (Cyclic Redundancy Code) (16 bits) : contient une séquence calculée à partir des champs SOF, arbitrage, contrôle et données qui permet de s'assurer de la validité du message.

- Champ d'acquittement (2 bits) : contient la réponse des nœuds ayant reçu le message. Si aucune erreur n'est détectée après calcul du CRC, un bit dominant est envoyé sinon, une trame d'erreur est émise.

- Fin de trame EOF (End Of Frame)(7 bits) : ce champ est constitué de sept bits récessifs indiquant la fin de la trame.

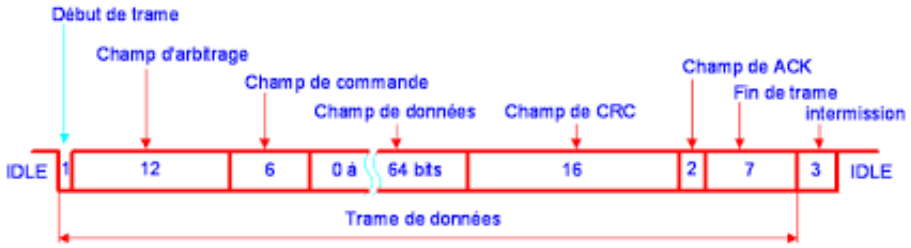


FIGURE A.2 – Champ d'une trame en bus CAN

Le support de transmission peut théoriquement être deux simples fils conducteurs. Cependant, pour des performances optimales, on utilise généralement des câbles coaxiaux, des paires torsadées ou même des fibres optiques.

A.1.2 USB-CAN

Afin de connecter l'application du simulateur de vol qui se trouve sur un ordinateur au bus CAN, on a besoin d'un adaptateur.

Le USB-CAN modul1 de Systec (**Figure A.3**) offre une solution d'interface entre le bus CAN et l'ordinateur via une connexion USB. Il est fourni avec un logiciel qui permet de suivre en temps réel l'échange des trames et également d'en envoyer.

Il est également accompagné de son API en C++ qui nous permet de le programmer.

A.1.3 Circuit MCP2551

Pour pouvoir communiquer entre le microcontrôleur MBED et le bus CAN, il est nécessaire de convertir les signaux de transmission et de réception du MBED en signaux spécifiques au bus CAN.

Le circuit MCP2551 (**Figure A.4**) est un adaptateur de ligne entre les signaux numériques du MBED et les signaux spécifiques au bus CAN tout en assurant une adaptation d'impédances entre les deux. Il suffit de connecter le CAN T du MBED à la broche RXD du circuit et le CAN R au TXD. Les broches CANH et CANL sont connectés au bus CAN. On met VDD à 5 V et VSS et Rs à la masse. On pourra alors programmer les trames sur le MBED.

FIGURE A.3 – Adaptateur USB-CAN modul1

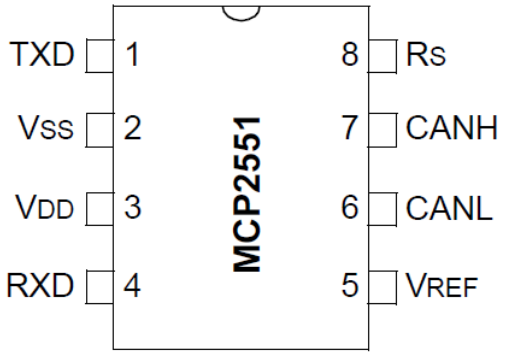


FIGURE A.4 – Circuit intégré MCP2551

A.1.4 Module SJA1000

Il existe divers circuits permettant au FPGA d'interagir avec le bus CAN. On s'intéresse au SJA1000 qui est un contrôleur CAN de type standalone développé par NXP Semiconductors.

Le modèle de programmation de ce contrôleur repose sur plusieurs registres tels que les registres MODE, COMMAND, STATUS, INTERRUPT et d'autres registres présentés plus en détails dans le **Chapitre 3**.

Globalement la procédure de communication au sein du SJA1000 se fait en l'initialisant en choisissant le mode (mode normal ou interruptible) et en configurant les différents registres puis en préparant le message à transmettre et en faisant une demande de transmission. Il peut également réagir à la réception d'un message ou d'une erreur.

A.2 Bus I2C

Le bus I2C (Inter Integrated Circuit) est un protocole de communication série, synchrone, bidirectionnel et half duplex (les données ne circulent que dans un sens à la fois). Plusieurs périphériques peuvent se connecter au bus en tant que maitre ou esclave.

Les échanges dans ce bus ont lieu entre un maitre et un ou tous les esclaves mais jamais entre deux maitres ou entre deux esclaves. Selon le circuit qui y est implémenté, un équipement peut avoir la possibilité de se comporter comme maitre et comme esclave tandis que d'autres ne peuvent avoir qu'un seul comportement.

L'architecture de ce bus est composée de deux lignes :

- Ligne SDA (Serial Data Line) : C'est la ligne sur laquelle transitent les données.
- Ligne SCL (Serial Clock Line) : C'est la ligne d'horloge qui permet la synchronisation des périphériques.

En plus de ces deux lignes, les composants doivent être reliés à une masse commune. Les deux lignes sont reliées à une tension VDD à travers des résistances de PULL UP. **La Figure A.5** montre l'architecture du Bus I2C.

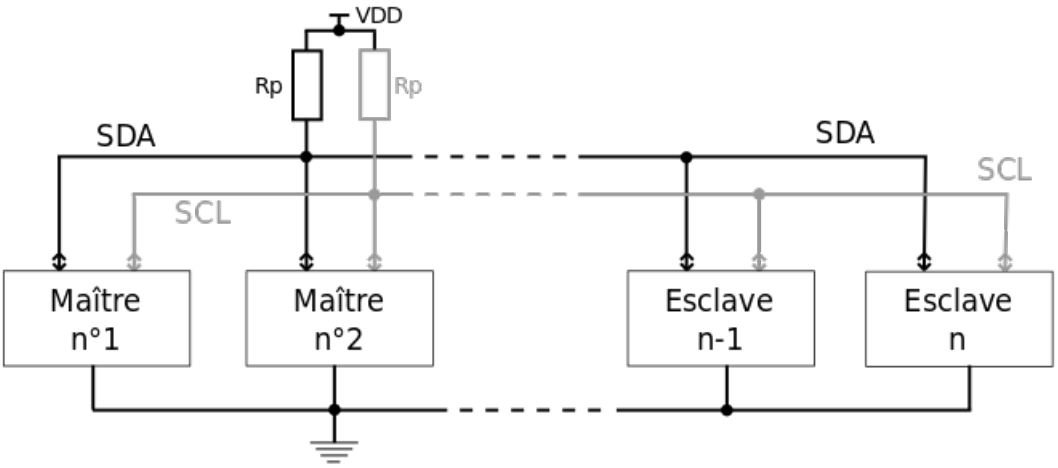


FIGURE A.5 – Architecture du bus I2C

Chaque équipement possède une adresse unique sur 7 bits, ce qui limite le nombre maximal d'équipements à 128.

De même que pour le bus CAN, les équipements sont connectés sur le principe d'un ET Logique, ce qui fait que le niveau « 0 » est dit dominant et le niveau « 1 » est dit récessif. Pour qu'un état HIGH ou LOW soit lu, la ligne SDA doit être maintenu à un état stable durant l'état HIGH de la ligne SCL.

Pour qu'un maitre prenne le contrôle du bus, ce dernier doit d'abord être en état de repos (Les lignes SDA et SCL au niveau haut). Le maitre génère alors le signal d'horloge qui synchronisera tous les esclaves et le départ est donné lorsqu'un front descendant a lieu sur la ligne SDA. L'esclave quant à lui a pour rôle de répondre aux requêtes du maitre s'il est adressé ou alors tout simplement d'ignorer tous les messages s'il ne l'est pas.

La condition de départ est un front descendant sur la ligne SDA pendant le niveau haut sur la ligne SCL et la condition d'arrêt est un front montant sur la SDA pendant le niveau haut sur la SCL.

L'échange entre le maitre et l'esclave peut être décomposé en deux partie. Dans la première partie, le maitre est émetteur et l'esclave récepteur :

- Le maitre transmet la condition de démarrage (1 bit) suivi d'un octet contenant l'adresse de l'esclave (7 bits) et le bit R/W à 0 signifiant que le maitre va envoyer des données.
- L'esclave répond par un bit d'acquittement ou de non acquittement suivi par une éventuelle pause sur un bit.

- Le maitre émet un octet de commande.
- L'esclave répond à nouveau par un bit d'acquittement ou de non acquittement suivi par une éventuelle pause sur un bit.
- Le maitre transmet un bit de RESTART signifiant le début d'une nouvelle trame dès la fin de la trame précédente sans passer par la condition d'arrêt, suivi d'un octet contenant l'adresse de l'esclave (7 bits) et le bit R/W à 1 signifiant que le maitre est prêt à recevoir des données.

Dans la seconde partie, le maitre est récepteur et l'esclave est émetteur :

- L'esclave émet un octet de données vers le maitre.
- Le maitre répond par un bit d'acquittement ou de non acquittement.
- Si la réponse du maitre est un acquittement alors l'esclave émet un autre octet de données et si la réponse du maitre est un non acquittement, ça sera considéré comme le signal pour mettre fin au dialogue. L'esclave pourra alors envoyer une éventuelle pause sur un bit.
- Le maitre transmet la condition d'arrêt (1 bit).

A.3 Bus SPI

Le bus SPI (Serial Peripheral Interface) est un protocole de communication série, synchrone, bidirectionnel et full duplex (les données circulent dans les deux sens au même temps). Plusieurs périphériques peuvent se connecter au bus, en général un seul en tant que maitre et plusieurs en tant qu'esclaves.

Les échanges dans ce bus ont lieu entre le maitre et un esclave. Si plusieurs esclaves existent, le maitre sélectionne l'esclave avec qui il veut communiquer avec le signal CS (Chip Select).

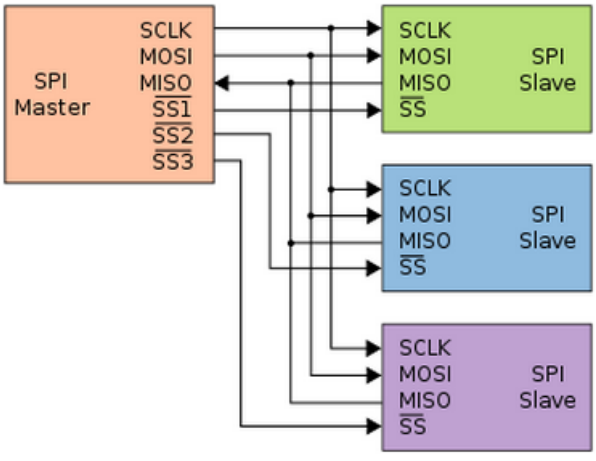


FIGURE A.6 – Architecture du bus SPI

L'architecture de ce bus est composée, comme on peut le voir sur la **Figure A.6**, de quatre lignes :

- Ligne SCLK (Serial Clock) : c'est la ligne d'horloge qui permet la synchronisation des périphériques. Elle est générée par le maitre.

- Ligne MOSI (Master Out Slave In) : c'est la ligne de transmission des données du maitre vers l'esclave.
- Ligne MISO (Master In Slave Out) : c'est la ligne de transmission des données de l'esclave vers le maitre.
- Ligne CS (Chip Select) : le maitre indique sur cette ligne avec quel esclave il communiquera.

L'échange entre le maitre et un esclave se fait de la manière suivante :

- Le maitre génère l'horloge et sélectionne l'esclave avec lequel il veut communiquer en envoyant un « 0 » sur la ligne CS de l'esclave.
- Le maitre envoie une requête vers l'esclave via la ligne MOSI et l'esclave répond via la ligne MISO.

Un bit est échangé à chaque coup d'horloge. Au bout de huit coups d'horloge, le maitre aura envoyé un octet à l'esclave et en aura reçu un également. La vitesse de l'horloge définit donc la vitesse de transmission et elle est réglée selon des caractéristiques propres aux périphériques.

Annexe B

Modèle cinématique

B.1 Notations du mécanisme

Dans le but de contrôler le mouvement de la plateforme, nous avons besoin de déterminer les trois angles à donner aux trois servomoteurs (θ_1, θ_2, θ_3) à partir des données que nous allons récupérer du simulateur de vol, à savoir, les angles tangage (α) et roulis (β).

Pour ce faire, on doit d'abord étudier le mouvement de la plateforme mobile par rapport à la plateforme fixe. Notre plateforme est constituée de trois degrés de liberté qui sont l'inclinaison horizontal, l'inclinaison vertical ainsi que l'élévation. On choisit les nominations suivantes qui sont également représentés dans la **Figure B.1** :

- α : inclinaison horizontale (Roulis).
- β : inclinaison verticale (Tangage).
- z : élévation selon l'axe verticale.
- a : distance entre le centre de gravité et le servomoteur '1' sur le plan de la plateforme.
- b : distance entre le centre de gravité et la projection du servomoteur '2' (ou '3') sur l'axe perpendiculaire à l'axe précédent.

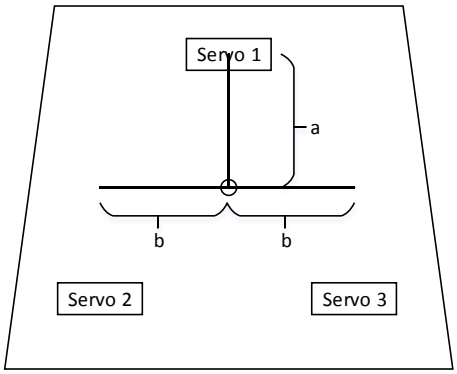


FIGURE B.1 – Représentation des distances sur la plateforme supérieure

Notre but est de pouvoir déterminer les angles des trois servomoteurs (θ_1, θ_2, θ_3) en fonction des trois degrés de liberté de la plateforme mobile (α,β,z), ce qui correspond au modèle cinématique inverse de cette plateforme.

Afin de pouvoir commencer notre étude, on doit définir les différents repères, un repère fixe relatif au centre de la plateforme fixe et un autre relatif au centre de gravité de la plateforme mobile.

Les trois servomoteurs sont positionnés sur la plateforme fixe sous forme de triangle et leur positions respectives sont B_1, B_2 et B_3.

$$B_1 = \begin{bmatrix} a \\ 0 \\ 0 \end{bmatrix} \qquad B_2 = \begin{bmatrix} -a \\ b \\ 0 \end{bmatrix} \qquad B_3 = \begin{bmatrix} -a \\ -b \\ 0 \end{bmatrix}$$

Le repère O est positionné au centre de la plateforme fixe avec l'axe Z pointé verticalement vers le haut et l'axe X pointe vers le premier servomoteur P_1. Ce repère est défini par les équations suivantes :

$$0 - XYZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De même, le second repère P' est positionné au centre de la plateforme mobile, avec l'axe Z' orienté verticalement vers le haut et l'axe X' vers la première rotule P'_1 . Ce repère est défini par les équations suivantes par rapport au premier repère :

$$P - XYZ = \begin{bmatrix} cos(\beta) & sin(\alpha).sin(\beta) & cos(\alpha).sin(\beta) & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ -sin(\beta) & sin(\alpha).cos(\beta) & cos(\alpha).cos(\beta) & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ainsi, un point p' positionné à des coordonnées (x', y', 0) sur la plateforme mobile est repéré par la matrice suivante :

$$p' = \begin{bmatrix} cos(\beta) & sin(\alpha).sin(\beta) & cos(\alpha).sin(\beta) & x'.cos(\beta) + y'.sin(\alpha).sin(\beta) \\ 0 & cos(\alpha) & -sin(\alpha) & y'.cos(\alpha) \\ -sin(\beta) & sin(\alpha).cos(\beta) & cos(\alpha).cos(\beta) & -x'.sin(\beta) + y'.sin(\alpha).cos(\beta) + z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pour nos calculs, on a besoin des positions des trois rotules de la plateforme mobile sur le référentiel de la base fixe. Or ces dernières sont positionnées sur la base mobile au positions (P_1^P P_2^P P_3^P), ainsi leur position dans le repère fixe est (P_1 P_2 P_3).

$$P_1 = \begin{bmatrix} a.cos(\beta) \\ 0 \\ -a.sin(\beta) + z \end{bmatrix}$$

$$P_2 = \begin{bmatrix} -a.cos(\beta) + b.sin(\alpha).sin(\beta) \\ b.cos(\alpha) \\ a.sin(\beta) + b.sin(\alpha).cos(\beta) + z \end{bmatrix}$$

$$P_3 = \begin{bmatrix} -a.cos(\beta) - b.sin(\alpha).sin(\beta) \\ -b.cos(\alpha) \\ a.sin(\beta) - b.sin(\alpha).cos(\beta) + z \end{bmatrix}$$

B.2 Modèle géométrique inverse

Afin de pouvoir étudier le mouvement de la plateforme mobile par rapport à la plateforme fixe, on doit déterminer les équations cinématiques du mouvement. Notre but est de pouvoir déterminer les angles des 3 servomoteurs (θ_1, θ_2, θ_3) en fonction des trois degrés de liberté de la plateforme mobile (α,β,z), ce qui correspond au modèle cinématique inverse de cette plateforme.

En utilisant les notations précédentes, le problème géométrique inverse peut-être résolu en écrivant l'équation de fermeture géométrique de chacune des chaines cinématiques actionnées.

Cette chaine est constituée par l'origine du repère de la plateforme fixe qui est le point O, le servomoteur r, les deux bras (u de longueur l_1 et v et longueur l_2) et de la rotule de la plateforme mobile p.

On explique le raisonnement pour le premier bras, qui s'appliquera également pour les deux autres.

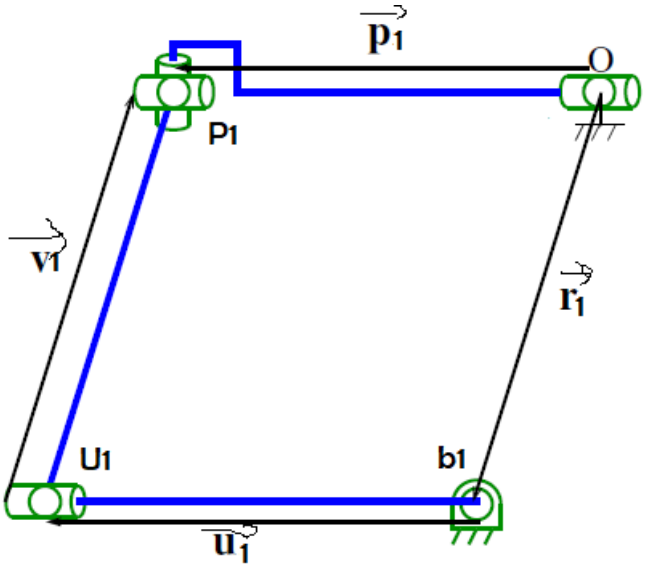


FIGURE B.2 – Représentation de la chaine cinématique

D'après le graphe, on a : $v_1 = P_1 - r_1 - u_1$

On ne peut pas définir l'angle de l'inconnu v_1 dû à son mouvement libre. Afin de résoudre ce problème, ce vecteur est mis à gauche de l'égalité et en passant au module le problème se résout ($||v_1|| = ||l_2||$).

L'équation devient donc :

$$v_1^T v_1 = l_2^2 = (P_1 - r_1 - u_1)^T (P_1 - r_1 - u_1) = (P_1 - r_1)^T (P_1 - r_1) - 2(P_1 - r_1)^T u_1 + l_1^2$$

Après arrangement, l'équation devient :

$$2(P_1 - r_1)^T u_1 + l_2^2 - l_1^2 - (P_1 - r_1)^T (P_1 - r_1) = 0$$

Avec :

- r_1 : position du servomoteur (fixe).
- P_1 : position de la rotule de la plateforme mobile.
- u_1 : vecteur qui s'écrit dans le repère O comme suit :

$$u_1 = \begin{bmatrix} l_1.cos(\theta_1) \\ 0 \\ l_1.sin(\theta_1) \end{bmatrix}$$

En remplaçant les valeurs des vecteurs, on obtient :

$$A_1.cos(\theta_1) + B_1.sin(\theta_1) - C_1 = 0$$

Avec :

- $A_1 = 2(P_1 - r_1)^T l_1.e_1$
- $B_1 = 2(P_1 - r_1)^T l_1.e_3$
- $C_1 = (P_1 - r_1)^T(P_1 - r_1) - l_2^2 + l_1^2$
- $e_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$
- $e_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

On effectue le changement de variable $t_1 = tan(\frac{\theta_1}{2})$

On aura donc :

$$cos(\theta_1) = \frac{1 - t_1^2}{1 + t_1^2} \qquad\qquad sin(\theta_1) = \frac{2.t_1^2}{1 + t_1^2}$$

Ainsi, notre équation prend la forme d'une équation quadratique en t_1 :

$$-(A_1 + C_1).t_1^2 + 2.B_1.t_1 + (A_1 - C_1) = 0$$

Cette équation conduit dans le cas général à deux solutions distinctes pour t_1 et donc en remplaçant des solutions, on trouve deux solutions pour l'angle du premier servomoteur :

$$\theta_1 = 2.arctan(\frac{B_1 - \sqrt{A_1^2 + B_1^2 - C_1^2}}{A_1 + C_1})$$

Avec le même raisonnement pour les deux autres angles, on trouve :

$$\theta_2 = 2.arctan(\frac{B_2 - \sqrt{A_2^2 + B_2^2 - C_2^2}}{A_2 + C_2})$$

$$\theta_3 = 2.arctan(\frac{B_3 - \sqrt{A_3^2 + B_3^2 - C_3^2}}{A_3 + C_3})$$