Génie électrique / électronique

semestre 2023-2024

Travail de diplôme

Simulateur dynamique pour siège gaming

2411

Réalisé par :

Jonathan Shifteh

À l'attention de :

M. Gavin

M. Pereira

M. Bommottet

Début

19 août 2024

Fin

24 septembre 2024

Glossaire

DOF : Degree of Freedom – degré de liberté

uC : Microcontrôleur

Wiper moteur : Moteur d'essuie-glace

Monitoring : Système de surveillance ou contrôle en temps réel

Dps : Degrés par seconde - mesure de la vitesse angulaire

G : Gravité - unité de mesure pour l'accélération

PID : Proportionnel Intégral Dérivé - algorithme de contrôle

PCB : Printed Circuit Board - carte à circuit imprimé

BOM : Bill of Materials - liste de matériel

PWM : Pulse Width Modulation - modulation de largeur d'impulsion

UART : Universal Asynchronous Receiver-Transmitter - Récepteur-Émetteur asynchrone universel

Rx : Réception - pour la communication série

Tx : Transmission - pour la communication série

I2C : Inter-Integrated Circuit - Protocole de communication

ADC : Analog-to-Digital Converter - Convertisseur analogique-numérique

OC : Output Compare - Comparateur de sortie

ReMapping : Recalibrage ou transformation d'une plage de valeurs

SimTools : Outil de simulation pour la gestion des jeux vidéo et des simulateurs

Sabertooth : Type de driver moteur - Sabertooth 2x32

Table des matières

1 Cahier des charges 6
2 Introduction 6
2.1 Contexte 6
2.2 But du projet 6
2.3 Organisation 6
3 Base mécanique 7
Pré-étude 7
4 Planning 7
5 Description du système 7
5.1 Schéma bloc 7
5.2 Description des blocs 8
5.2.1 Alimentation / régulateurs 8
5.2.2 Microcontrôleur 8
5.2.3 Driver moteur 8
5.2.4 Wiper moteur 8
5.2.5 Accéléromètre / gyroscope 8
5.2.6 Capteur de position 9
5.2.7 Jeu de course 9
5.2.8 Application Simtools 9
5.2.9 Application C# 9
6 Choix des composants 9
6.1 Alimentation 9
6.2 Régulateur 24V - 5V 9
6.3 Régulateur 5V - 3.3V 9
6.4 Microcontrôleur 10
6.5 Driver moteur 10
6.6 Wiper moteur 11
6.7 Potentiomètre 12
6.8 Accéléromètre / gyroscope 12
6.9 Capteur de position 12
7 Estimation de la consommation en courant 13
Design 13
8 Réalisation du schéma 13
8.1 Communication Simtools – uC 14
8.2 Régulateur 5 V et 3.3 V 14

8.2.1 Régulateur 5 V 15
8.2.2 Régulateur 3.3 V 15
8.3 Commande du driver moteur (Sabertooth 2x32) 16
8.3.1 Lignes de commandes A1 et A2 16
8.3.2 Lignes de commandes S1 et S2 17
8.4 uC 18
8.5 Connecteurs 19
8.5.1 Commande du Sabertooth 2x32 19
8.5.2 Elément de contrôle afin de régler le PID 19
8.5.3 PINs inutilisés 20
9 BOM 20
Réalisation 20
10 Réalisation du PCB 20
11 Coût total du projet 20
12 Vue du PCB avec les différents blocs 21
13 Vues des couches de mon PCB 21
14 Vue 3D de mon PCB 22
15 Test le hardware 22
15.1 Modification à faire 22
Software/Firmware 23
16 Configuration Simtools 23
17 Configuration d'Harmony (MPLAB) 25
17.1 Réglage des pins du uC 25
17.2 Réglage Harmony 26
17.2.1 Timers 26
17.2.2 OC 28
17.2.3 UART 28
18 Réalisation du code 29
18.1 Diagramme d'état du logiciel 29
18.2 App_Tasks 29
18.3 Fonction PID 31
18.4 Fonction FeedbackPot 31
18.5 Fonction Limite 32
18.6 Fonction ReMapping 32
18.7 Modification librairie ADC 33
18.7.1 Mc32DriverAdc.c 33
18.7.2 Mc32DriverAdc.h 33

Mesures et problèmes rencontrés 33
19 Matériel de mesure....33
20 Mesure sur le système 34
20.1 Mesure des tensions d'alimentations sur le PCB....34
20.2 Mesure de la trame UART....34
20.2.1 Schéma de mesure 34
20.2.2 Protocole de mesure 34
20.2.3 Résultat....34
21 Problèmes rencontrés 36
22 Etat actuel du projet 37
22.1 Tâches effectuées....37
22.2 Tâches restantes....37
Déroulement du diplôme 37
23 Apports du travail de diplôme 37
24 L'environnement social....37
25 Environnement naturel 37
26 Leadership et développement personnel....38
Suivi du journal, PV et conclusion final 38
27 Journal de travail....38
28 Procès-verbal....38
29 Conclusion 38
30 Bibliographie (Datasheets et ChatGPT)....40
31 Annexe....41
Annexe 1 : Cahier des charges....41
Annexe 2 : Planification 41
Annexe 3 : Journal de travail....41
Annexe 4 : Procès-Verbaux 41
Annexe 5 : Schématique....41
Annexe 6 : Plan d'assemblage....41
Annexe 7 : BOM 41
Annexe 8 : Software 41
Annexe 9 : Mesure Timer1....41
Annexe 10 : Feuille de modification 41
Annexe 11 : Mode d'emploi du système....41
Annexe 12 : Construction partie mécanique....41
Annexe 13 : Tableau des différents DoF....41

1 Cahier des charges

Le cahier des charges se trouve en annexe N° 1.

2 Introduction

2.1 Contexte

Pour finir mes études à l'ETML-ES dans la filière génie électrique, il est demandé de créer un projet qui est le travail de fin de diplôme afin de valider toutes les nouvelles connaissances apprises. Pour ce faire, nous avons 5 semaines dédiées à ceci.
J'ai voulu profiter de cette occasion pour faire un projet qui me tenait à cœur, donc j'ai proposé un projet en donnant une première version du cahier des charges. Suite à plusieurs discussions avec mes professeurs, ce projet a été validé et m'a été attribué.
Mon maître de diplôme est M. Gavin, suivi de deux experts mandatés par l'ETML-ES, qui sont M. Pereira et M. Bommottet.

2.2 But du projet

Durant ce travail de diplôme, mon but est de créer un simulateur dynamique pour un siège gaming afin de ressentir les sensations de mouvement telles qu'elles sont dans le jeu vidéo.

Ce genre de simulateur peut avoir jusqu'à 6 degrés de liberté (vous pouvez trouver un tableau expliquant les différents DOF avec les mouvements qu'ils proposent en annexe N° 13), cependant, pour commencer, j'ai décidé de partir sur un 2 DOF, ce qui va me permettre de reproduire 2 styles de mouvements appelés Pitch et Roll :

- Le pitch est un des mouvements les plus importants car il permet de ressentir les accélérations et les freinages, ce qui offre de bonnes sensations. Du coup, il utilisera l'axe Y.
- Le roll est aussi un mouvement clé pour offrir de bonnes sensations, car celui-ci est basé sur la simulation des virages. Il utilisera l'axe X.
- Finalement, grâce à ce système, on pourra ressentir les virages, l'accélération, le freinage et aussi les variations de la route, donc les montées, descentes, et si la route est penchée.

Pour le cadre de l'école, je dois réaliser le mouvement nommé Pitch. Puis le mouvement Roll sera réalisé hors travail de diplôme.

Le choix de mes 2 moteurs s'est penché sur des Wiper moteurs, car déjà le coût de ces moteurs est généralement bon marché, ils ont un couple élevé, ce qui va être bénéfique pour recréer les mouvements réalistes du jeu. Ils sont dits réversibles car ils peuvent changer rapidement de direction, ce qui est essentiel.

2.3 Organisation

Pour réaliser un tel projet lors du travail de diplôme, j'ai dû au préalable, construire la structure métallique ainsi que commander mes 2 moteurs et le driver qui me permettra de les commander.

Durant ce projet, j'avais une séance hebdomadaire avec M. Gavin afin de vérifier et d'examiner l'évolution du projet.

3 Base mécanique

Pour la création de la base mécanique, j'ai réalisé le plan mécanique avec mon père, qui est ingénieur, et aussi en parallèle avec les consignes du créateur du site simukit.com. Ce créateur passionné de simulateurs m'a envoyé les 2 moteurs ainsi que le driver et l'alimentation, c'est pourquoi il nous a guidés dans cette conception mécanique.

Une fois cela fait, j'ai commandité une entreprise pour me créer cette structure, qui sera divisée en 2 parties : la base, qui ne bougera pas, puis la deuxième partie, qui reflétera les mouvements sur le siège gaming.

Vous pouvez retrouver quelques photos de cette conception mécanique en annexe N° 11.

Pré-étude

4 Planning

À la suite du cahier des charges, je me suis fait un planning afin de gérer au mieux le temps à disposition.

Durant le projet, je viendrai compléter ce planning avec les vraies tâches effectuées durant la journée.

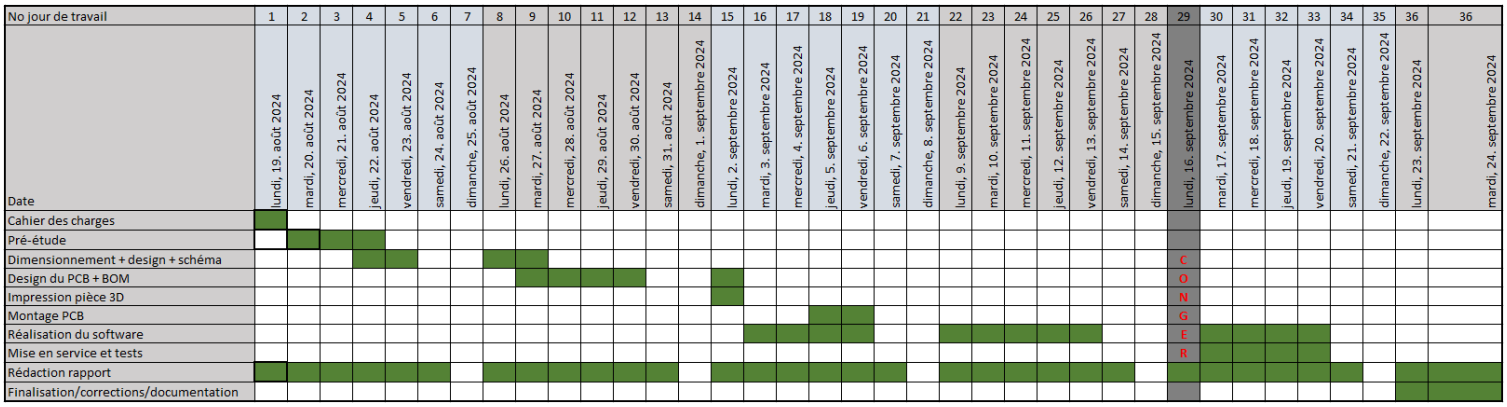


Figure 1 Planning

5 Description du système

5.1 Schéma bloc

Le schéma bloc ci-dessous représente les différents composants primaires du projet, facilitant ainsi la conception et la compréhension de celui-ci.

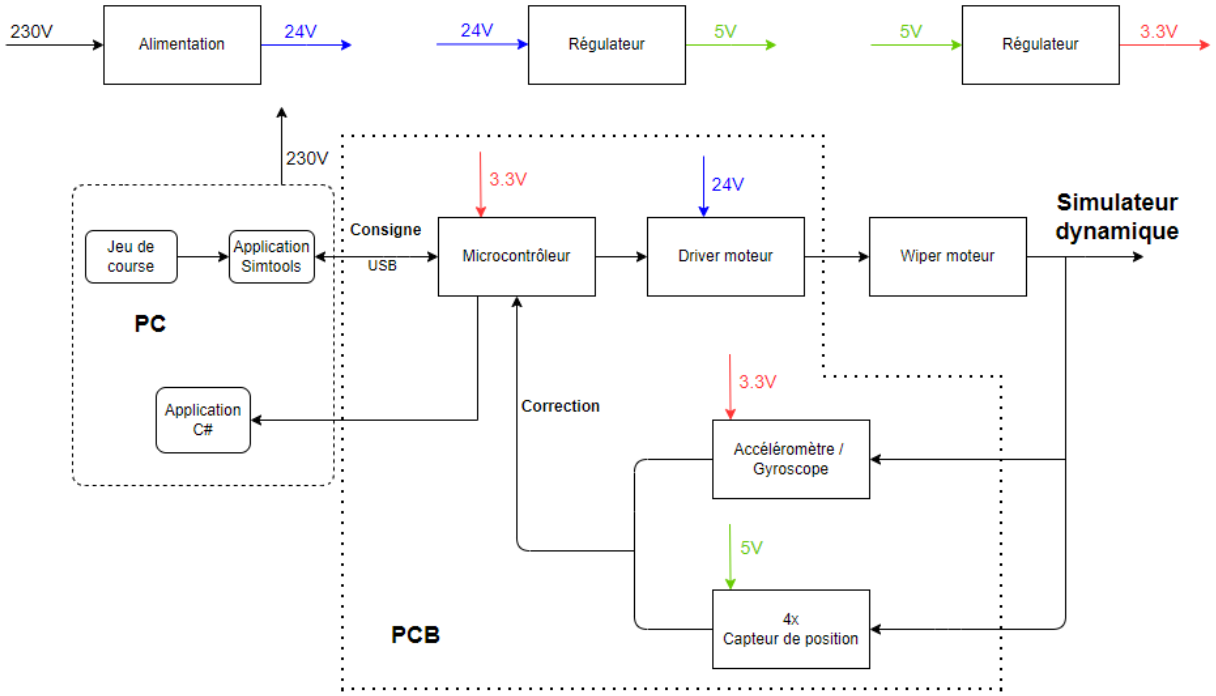


Figure 2 Schéma bloc final

Vous pouvez trouver une version plus grande en annexe N° 5.

5.2 Description des blocs

Voici une petite description de l'utilité de chacun des blocs.

5.2.1 Alimentation / régulateurs

Pour alimenter mon système, je vais avoir besoin d'une alimentation 24VDC qui me délivre assez de puissance pour gérer mes moteurs.

Puis, mon système va être capable, grâce aux régulateurs, de fournir du 5V et du 3.3V pour les différents composants de mon projet.

5.2.2 Microcontrôleur

Ce composant est le cerveau du système, il va permettre de gérer les différents composants sur le PCB et il va contrôler les moteurs via des commandes envoyées au driver moteur.

5.2.3 Driver moteur

Ce driver moteur sert à contrôler et alimenter les wiper moteurs avec un signal d'entrée géré par le uC. Ce qui nous permet de réguler la vitesse et la position des moteurs.

5.2.4 Wiper moteur

Les wiper moteurs serviront à faire bouger le siège afin d'avoir le ressenti d'un simulateur.

Le choix des moteurs est important car ils doivent être coupleux afin d'avoir de bons mouvements et notamment être réversible afin de changer rapidement de direction.

5.2.5 Accéléromètre / gyroscope

Nous allons récolter les données du gyroscope afin d'avoir les axes X et Y, ce qui nous servira pour la partie réglages des moteurs.

5.2.6 Capteur de position

Le choix est d'implémenter un capteur de distance sur chaque coin de la structure qui sera fixe afin de voir les mouvements de la partie qui sera affectée par les mouvements de la voiture.

Je pensais aussi utiliser ces informations comme protection. (Exemple, si les moteurs s'emballent, nous allons voir un trop grand (ou trop faible) écart entre mes 2 structures métalliques donc il faut arrêter les mouvements pour ne pas avoir un choc entre ceux-ci).

5.2.7 Jeu de course

Le jeu de course enverra les données de la voiture à Simtools.

5.2.8 Application Simtools

Simtools va venir convertir les données reçues par le jeu pour les envoyer au uC via USB.

5.2.9 Application C#

En option, mais si le temps le permet, avoir une application qui retransmet les mouvements ainsi que les données du siège grâce aux capteurs et au gyroscope.

6 Choix des composants

Tous les liens vers les fiches techniques des composants primaires utilisés durant le projet se trouvent dans la bibliographie.

6.1 Alimentation

Pour mener à bien mon projet, j'ai besoin d'une alimentation qui doit fournir assez de puissance sur mon système. Donc en premier lieu, il faut choisir les moteurs, à la suite de ça, nous savons la puissance dont j'aurai besoin.

En commandant les deux moteurs chez Simukit (Site qui a fermé mais toujours possible de prendre contact avec le créateur de la boutique en ligne), il m'a proposé aussi une alimentation à découplage qui délivrait 24VDC/1kW ce qui est parfait pour faire un simulateur 2DOF avec les deux wiper moteurs en question.

Caractéristiques de l'alimentation 24VDC/1kW	
Tension de sortie	24 VDC (précision ajustable)
Puissance de sortie	1000 W
Courant de sortie	~41,6 A
Précision ajustable	20 – 26.4V
Tension d'entrée	230 VAC
Température protégée	Jusqu'à 70 °C
Dimensions et poids	291 x 132 x 68 mm et 2.35 kg

6.2 Régulateur 24V - 5V

Ayant une alimentation de 24 V, je suis parti sur un régulateur step-down qui me sort du 5 V.

J'ai choisi le composant : MIC4680-5.0YM, qui est un step-down avec une tension fixée à 5 V et qui peut sortir jusqu'à 1.3 A. Ce qui est largement suffisant, mais sachant pas jusqu'à où je vais aller en projet personnel, je préférais être large afin de pouvoir rajouter un deuxième driver moteur pour avoir d'autres mouvements.

6.3 Régulateur 5V - 3.3V

J'ai choisi le composant : 173950336, qui est également un step-down auquel je vais mettre une tension d'entrée de 5 V.

Ce composant a une tension de sortie fixée à 3.3 V avec un courant de 500 mA, ce qui va me permettre d'alimenter mon uC.

6.4 Microcontrôleur

Pour le choix du microcontrôleur, j'ai voulu partir sur un Arduino, mais vis-à-vis de l'école, je devais avoir un uC à programmer. Du coup, pour savoir mes besoins, j'ai réalisé un tableau ci-dessous qui permet de déterminer le nombre de pins et les communications nécessaires pour mener à bien mon projet.

Système à commander	Nombre d'I/O	Type d'I/O
Driver	4	Sortie analogique
Capteur de position	4	Entrée analogique
Gyroscope	2	I2C
Application Simtools	2	Serial

Suite à ça, j'ai décidé de partir sur la même famille du uC utilisé dans l'école. N'ayant pas besoin de beaucoup d'entrées/sorties, j'ai pris celui qui a 44 pins. C'est le PIC32MX230F256D-IPT.

J'ai pris un de 256 Ko en taille programmable, étant donné que c'est un projet personnel que je vais continuer après le travail de diplôme. Je préférais ne pas être limité pour la taille.

6.5 Driver moteur

J'ai décidé de partir avec le Sabertooth 2x32, qui est un driver moteur conçu pour contrôler deux moteurs DC à haute puissance et de manière simultanée. Il peut fournir 32 ampères par canal (avec des pics pouvant atteindre jusqu'à 64 A).

Nous pouvons communiquer avec le Sabertooth de différentes manières : par R/C, avec une entrée analogique, par communication série ou par USB.

Il possède une large variété de modes de fonctionnement paramétrables avec les DIP Switches. Il intègre également des fonctions de protection et de diagnostic pour être sûr de la sécurité et de la fiabilité du système.

Caractéristiques Sabertooth 2x32	
Tension d'entrée	6 - 33.6 V_{DC} max
Courant par canal	32 A en continue et 64 A pour des pic
Protection	Contre les surcharges de courant et la chaleur
Connexion	USB pour le contrôle et aussi le monitoring
Entrée (commande)	R/C, Analogique, Serial, USB
Vitesse de communication	Vitesse max = 115200 Baud

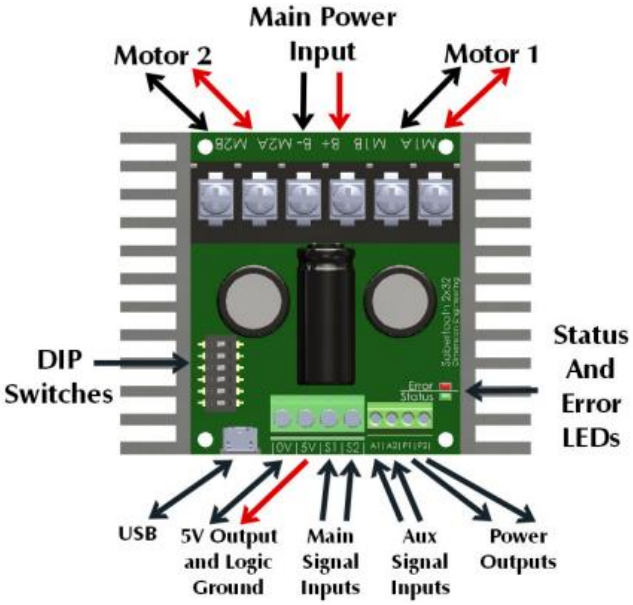


Figure 3 Overview de la carte Sabertooth 2x32

Pour configurer la carte, nous allons utiliser l'application DEScribe :

- Télécharger DEScribe et connecter la carte via son port USB
- Fixer les paramètres pour le mode Analog
- Fixer la limite de courant pour nos moteurs sous Motor Outputs
- Configurer le DIP Switches afin d'avoir le mode 'Analog'

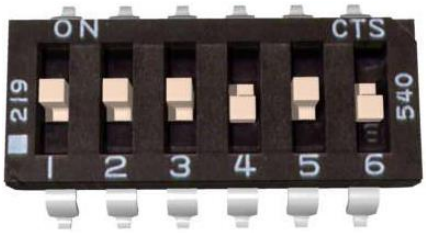


Figure 4 DIP switch settings for Analog Mode

Ici, nous sommes en analog mode, mais en mettant le switch 4 à OFF, nous gérons indépendamment nos moteurs (S1 gère le moteur 1 et S2 gère le moteur 2). Si nous décidons de gérer nos moteurs ensemble, nous mettons le switch 4 à ON ; l'entrée S1 contrôlera la vitesse des deux moteurs et l'entrée S2 contrôlera le sens de rotation des deux moteurs.

En mettant le switch 6 à OFF, nous gérons le sens de rotation des moteurs avec A1 et A2, afin que si nous mettons 0 V aux S1-S2, les moteurs s'arrêteront. Sinon, le 0 V représente la vitesse maximale en sens inverse du moteur, et c'est 2,5 V qui feront arrêter les moteurs (pas idéal si mon uC plante et me sort 0 V aux PINs reliés à S1-S2).

6.6 Wiper moteur

Le fonctionnement d'un wiper moteur en deux points :

- Le moteur électrique est composé d'une partie tournante (l'induit), d'inducteurs à aimant permanent, et de deux balais sur le collecteur d'induit qui permettent de transmettre l'énergie électrique.
- Le mécanisme de transmission de la rotation du moteur est constitué d'une vis sans fin au bout d'induit, ce qui réduit la vitesse de rotation tout en augmentant le couple ;

donc il transmet le mouvement à un pignon de plus grande taille pour démultiplier la vitesse de rotation.

Caractéristique du wiper moteur	
Tension d'alimentation	24 V_{DC}
Puissance	400 W
Couple	57 Nm
Tours par minute	50 ± 10 % RPM
Poids	5 kg

6.7 Potentiomètre

J'utiliserai des potentiomètres FCP22E, qui sont des potentiomètres sans butée mécanique afin d'éviter qu'ils se cassent si les moteurs s'emballent.

Ces potentiomètres seront montés sur les moteurs avec un jeu de roues dentées afin de connaître la position exacte des deux moteurs, ce qui me permettra de renvoyer ces valeurs au uC, qui lui traitera les informations.

Les plages électriques sont de 320° avec ± 5 %.

Ils seront mes principaux capteurs de position afin de réguler le PID.

6.8 Accéléromètre / gyroscope

Il sera utilisé afin de visualiser les déplacements du siège. Et si le temps me le permet, gérer aussi les accélérations afin de les retranscrire sur une application C#. Ces données me serviront dans un premier temps comme paramètres de correction du siège et, deuxièmement, si le temps me le permet, de visualiser tous ces mouvements sur une application C#.

Mon capteur est un gyroscope numérique à 3 axes de la série Grove de chez Seeed Studio (ref : 105020012) et peut également être utilisé en accéléromètre. Ce chip permet d'être utilisé dans différentes applications de détection d'inclinaison et de mouvement.

Ce gyroscope est un LSM6DS3 à faible consommation et il présente une sensibilité élevée qui peut être configurée pour différentes plages de mesure de la vitesse angulaire et différents niveaux de sensibilité pour l'accéléromètre.

Caractéristiques LSM6DS3 (105020012)	
Tension d'alimentation	3.3 V_{DC}
Consommation	0.42 mA (en low power), 0.9 mA (en normal) et 1.25 mA (en haute performance)
Communication	I2C
Plage de mesure de l'angle	±125 / ±245 / ±500 / ±1000 / ±2000 dps
Plage de mesure de l'accélération	±2 / ±4 / ±8 / ±16 g
Sortie	6 données de mouvement DOF (3 axes gyroscope + 3 axes accéléromètre)
Dimension du capteur	24 mm x 24 mm x 15 mm

6.9 Capteur de position

Les capteurs de distance serviront à mesurer, à chaque extrémité de la structure du simulateur, la distance entre le cadre du bas et celui du haut.

Les capteurs utilisés sont des GP2Y0A21YK0F de type capteur infrarouge de chez Sharp.

Caractéristiques GP2Y0A21YK0F Sharp	
Tension d'alimentation	4.5 – 5.5 VDC
Consommation	30 mA
Plage de distance	10 à 80 cm
Sortie	Analogique

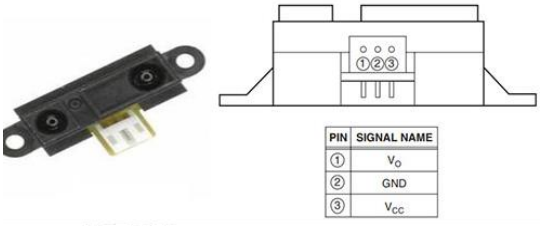


Figure 5 Capteur GP2Y0A21YK0F et branchement

Ma plage d'utilisation est environ entre 30 à 50 cm, donc ce capteur est parfait pour l'utilisation.

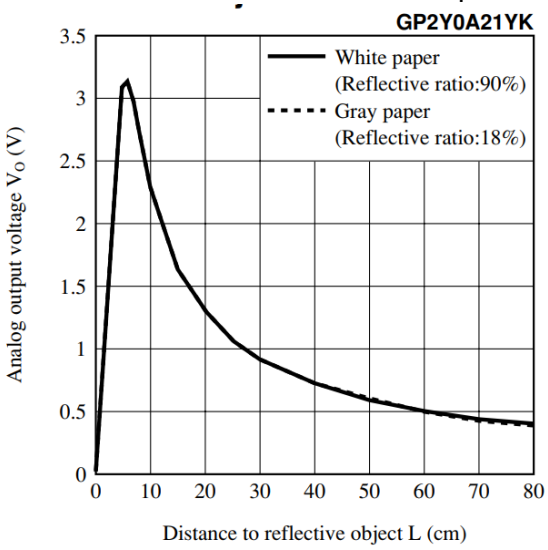


Figure 6 Caractéristiques de la tension de sortie par rapport à la distance

7 Estimation de la consommation en courant

Consommation en courant sans les moteurs				
Composant	Quantité	3.3 V	5 V	24 V
MIC4680-5.0YM	1	-	-	7 – 12 mA
PIC32MX230F256	1	20 – 30 mA	-	-
LSM6DS3	1	0.42 (LP), 0.9 (NP) et 1.25 (HP) mA	-	-
GP2Y0A21YK0F	4	-	30 mA	-
	Total	20.42 mA – 31,25 mA	120 mA	7 – 12 mA

Design

8 Réalisation du schéma

Pour réaliser le schéma, je me suis basée sur ma pré-étude.

Vous pouvez retrouver le schéma complet en annexe N° 5.

8.1 Communication Simtools – uC

Pour recevoir les données du jeu vidéo (les données envoyées par Simtools), j'ai dû choisir un composant qui va me convertir de l'USB en UART afin de pouvoir communiquer avec Simtools via le Tx et le Rx.

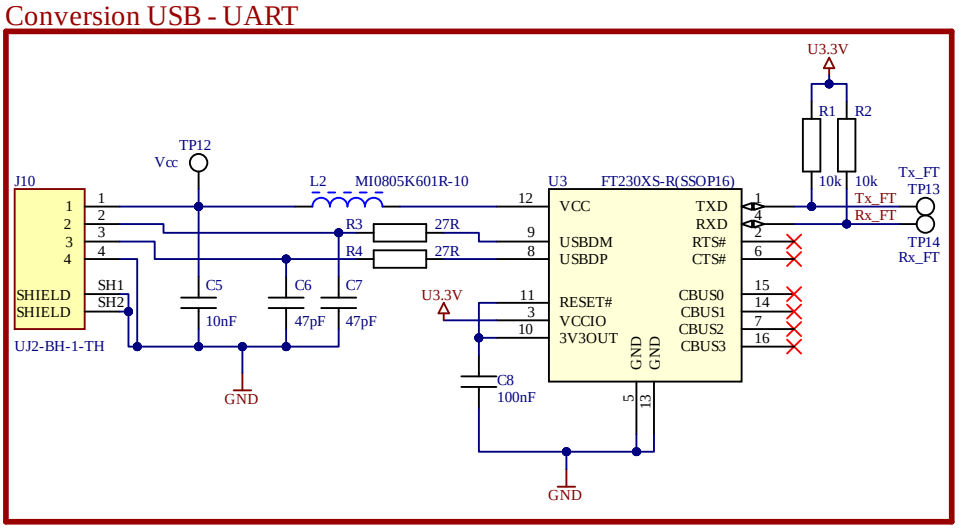


Figure 7 Conversion USB – UART

Je suis parti sur le FT230XS-R, auquel j'ai pu retrouver les valeurs des résistances et des capacités.

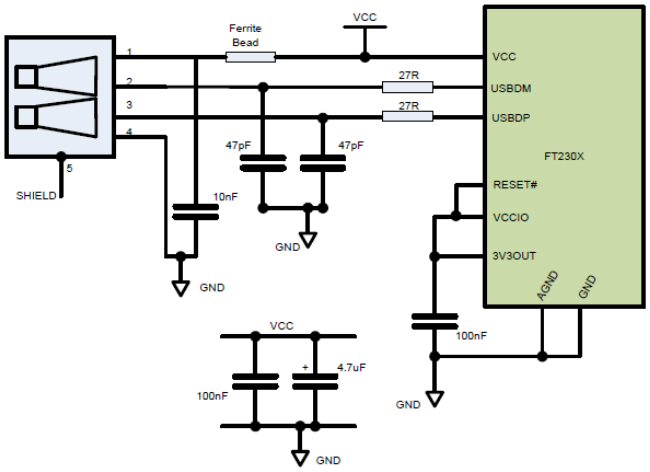


Figure 8 Schéma tiré de la datasheet

Il y a une différence sur mon schéma : c'est la PIN Vccio que j'ai directement reliée à mon régulateur 3.3V.

J'ai mis aussi 2 PULL-UP de 10 kΩ sur la ligne Rx et Tx ; cette valeur est une norme dans les PULL-UP.

8.2 Régulateur 5 V et 3.3 V

Pour dimensionner mes régulateurs 5 V et 3.3 V, je suis allé regarder dans leurs datasheets respectives afin de voir s'il y avait un schéma typique.

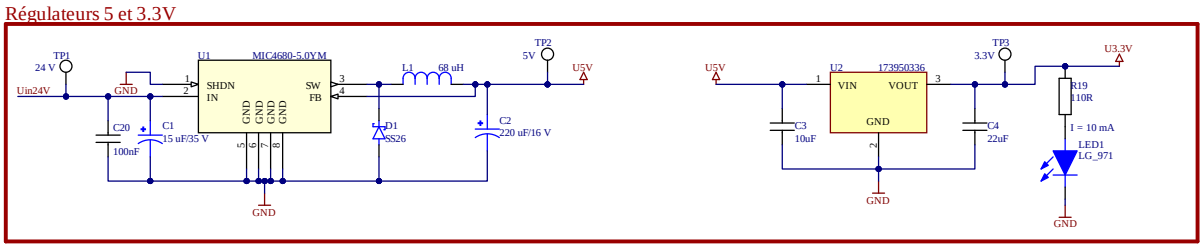


Figure 9 Schéma de mes 2 régulateurs

J'ai mis aussi une LED de couleur verte à la sortie du régulateur 3.3 V afin de savoir visuellement si mon circuit est bien alimenté ou non.

Pour la résistance, je suis venu calculer en fonction d'un courant souhaité, donc 10 mA.

$$R19 = \frac{U - Uled}{I} = \frac{3.3 - 2.2}{0.01} = 110\Omega$$

8.2.1 Régulateur 5 V

Pour le régulateur MIC4680-5.0BM, la datasheet me donnait un schéma typique.

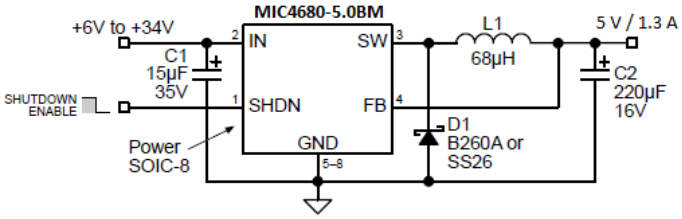


Figure 10 MIC4680-5.0BM schéma typique

Du coup, j'ai recopié ces valeurs sur mon schéma.

8.2.2 Régulateur 3.3 V

Pour ce régulateur, j'ai d'abord regardé un graphique qui me montre l'efficacité suivant la tension qu'on lui met en entrée.

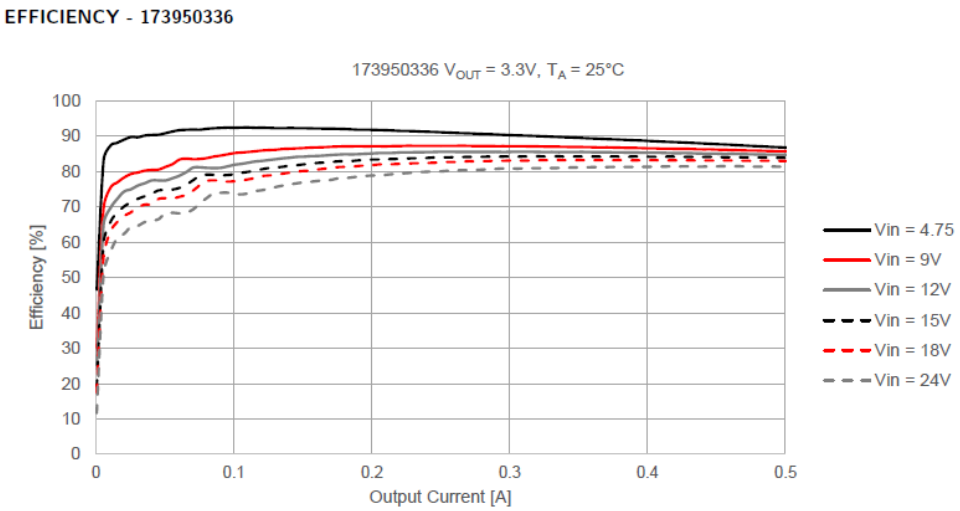
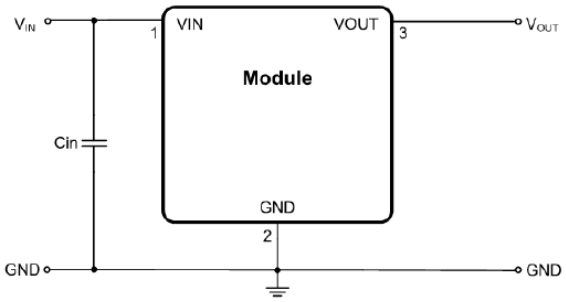


Figure 11 Graphique d'efficacité

Suite à ça, je suis partie sur un Vin de 5 V, ce qui était la meilleure option.

Puis, j'ai trouvé un tableau qui m'indiquait quelle capacité je devais mettre en Vin et GND.



SYMBOL	NUMBER	TYPE	DESCRIPTION
VIN	1	Power	The supply input pin is a terminal for an input voltage source. It is recommended to use a 10μF/50V input capacitor.
GND	2	Power	Ground pin; reference for V_{IN} and V_{OUT}.
VOUT	3	Power	Regulated output voltage pin. There is no need for an external output capacitor.

Figure 12 Valeur de Cin

Et pour la capacité au Vout, j'ai trouvé cette phrase qui expliquait que si je voulais avoir une faible ondulation en sortie, je devais mettre une capacité de 22 uF : *For low output voltage ripple, it is recommended to use an additional 22 μF external capacitor at the output of the module.*

8.3 Commande du driver moteur (Sabertooth 2x32)

Afin de bien commander mon Sabertooth, il faut savoir que les lignes de commandes S1, S2, A1 et A2 sont commandées en 5 V. Donc, je dois élever la tension que me procure la PIN du uC à 5 V.

Pour élever la tension sur un signal analogique, je vais utiliser un amplificateur opérationnel en montage non-inverseur. Et pour un signal série, je suis venu m'inspirer de ce que font SparkFun afin de créer un Logic Level Converter de 3.3 V à 5 V.

8.3.1 Lignes de commandes A1 et A2

Les lignes A1 et A2 du Sabertooth seront utilisées pour gérer le sens de rotation du moteur. Il faudra mettre 0 ou 5 V afin d'aller en avant ou en arrière.

Cela me permet d'envoyer 0 V sur S1 ou S2 pour que les moteurs s'arrêtent. Sinon, j'aurais dû mettre 2.5 V pour les arrêter, ce qui peut poser un problème si j'ai un souci dans mon code et que ma PIN de commande se met à 0 V, car je serai en Full Speed en arrière.

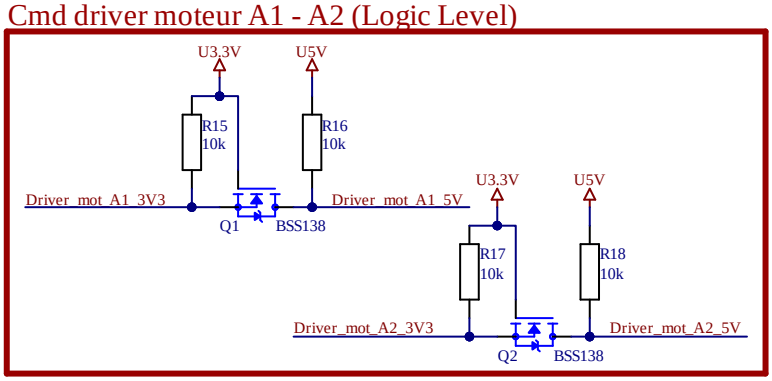


Figure 13 Logic Level Converter 3.3 V à 5 V

Voici deux tests faits sur Lushprojects.com afin de m'assurer du bon fonctionnement de mes Logic Level.

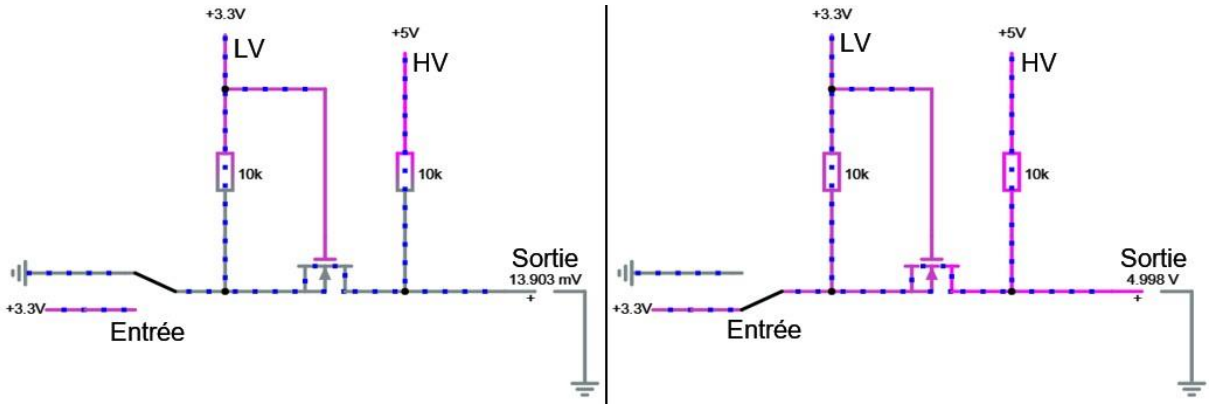


Figure 14 Simulation du montage Logic Level Converter

Donc, si je mets 0 V en entrée, notre sortie, qui sera reliée au driver moteur, aura bien 0 V.

Puis, si je mets un signal à 3.3 V en entrée, la sortie sera bien configurée en 5 V, ce qui est parfait pour les commandes du Sabertooth.

8.3.2 Lignes de commandes S1 et S2

Les lignes S1 et S2 servent à gérer la vitesse des moteurs 1 et 2.

Pour la ligne S1, j'ai décidé d'avoir deux options possibles, car je peux aussi bien commander mes moteurs en série en envoyant le Tx au S1 qu'en analogique afin de les contrôler indépendamment. Du coup, à l'aide d'un jumper (P6), je pourrai choisir entre une ligne série qui passera dans mon circuit Logic Level Converter ou une ligne analogique qui passera dans le non inverseur.

Et le S2 sera seulement commandé en analogique, donc suivi du non inverseur.

8.3.2.1 Calcul

R10 (R1) = 9.1k, R9 (R2) = 4.7k.

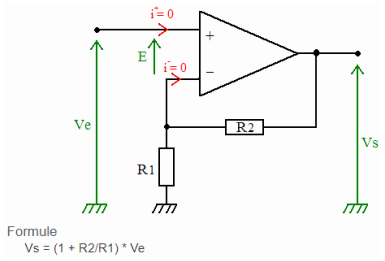


Figure 15 Schéma générique d'un non-inverseur

$$Vs = \left(1 + \frac{R2}{R1}\right) * Ve = \left(1 + \frac{4.7}{9.1}\right) * 3.3 = 5.0\,V$$

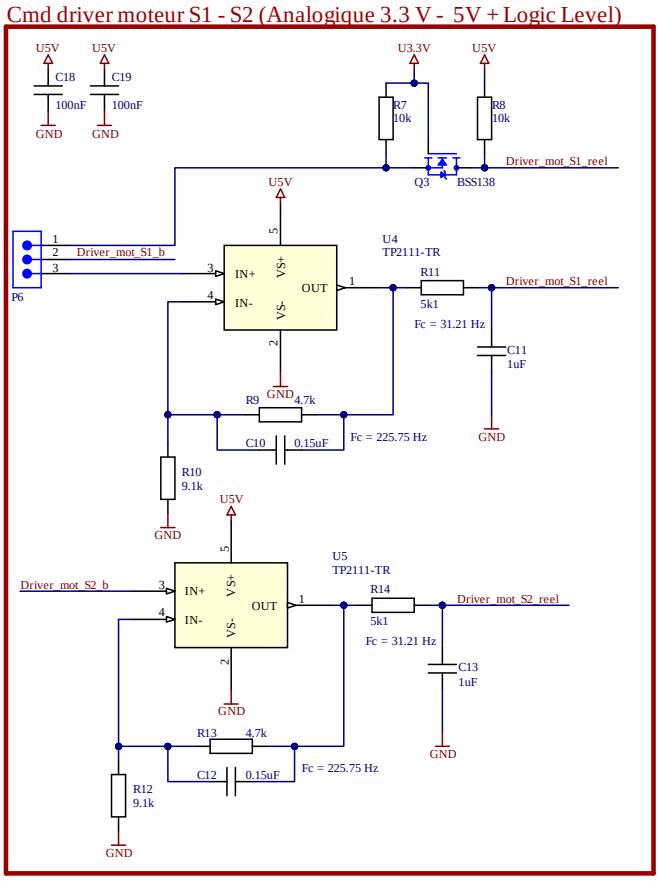


Figure 16 Montage pour les commandes S1 et S2 du Sabertooth

8.4 uC

Pour le uC, je suis venu chercher, dans sa datasheet, les capacités que je devais mettre afin d'avoir le bon fonctionnement de celui-ci.

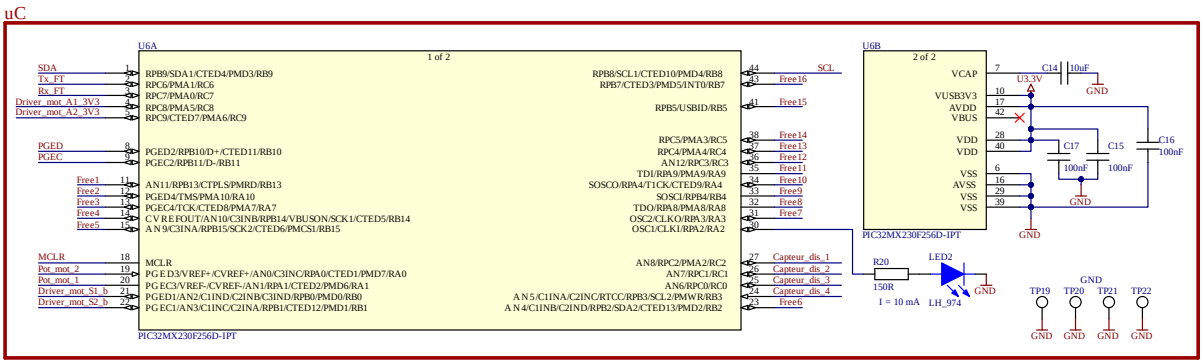
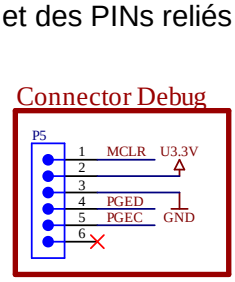
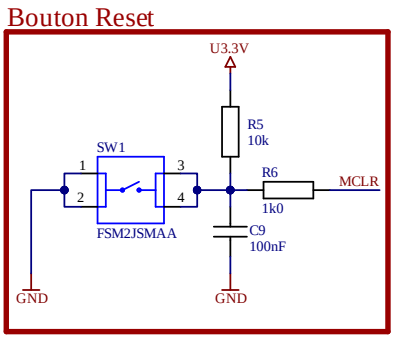


Figure 17 Schéma du uC

J'ai aussi mis à disposition un bouton reset afin de pouvoir le réinitialiser s'il y a un problème et des PINs reliés au PIN de programmation du uC.





J'ai mis aussi une LED de couleur rouge à l'un des pins de mon uC afin de pouvoir la commander.

Pour la résistance, je suis venu calculer en fonction d'un courant souhaité, donc 10 mA.

$$R20 = \frac{U - Uled}{I} = \frac{3.3 - 1.8}{0.01} = 150\Omega$$

8.5 Connecteurs

8.5.1 Commande du Sabertooth 2x32

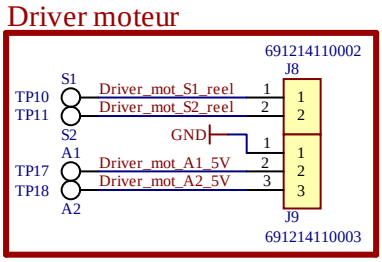
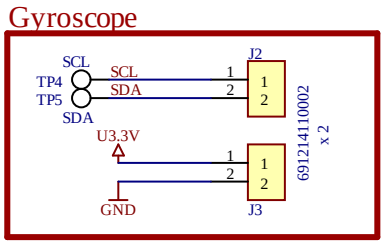
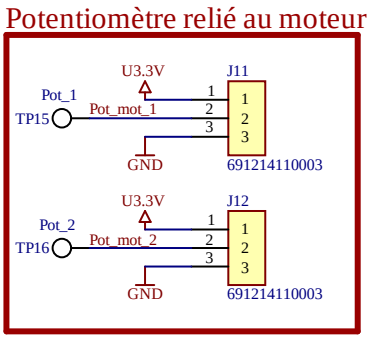


Figure 18 Connecteur pour les commandes du driver moteur

8.5.2 Elément de contrôle afin de régler le PID





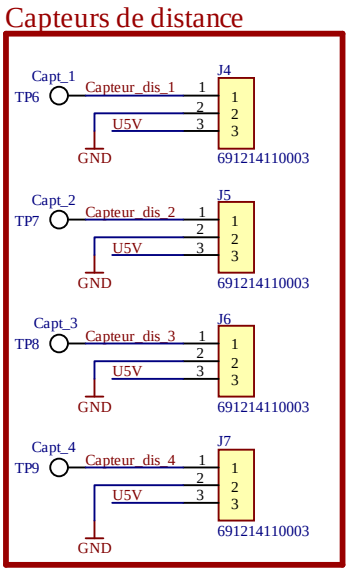


Figure 19 Connecteurs pour le gyroscope, les potentiomètres et les capteurs de distances

8.5.3 PINs inutilisés

Étant donné que je ne savais pas jusqu'où j'irai dans mon projet personnel lors de mon temps privé, j'ai rajouté toutes les PINs inutilisées sur 4 rangées de PINs afin de pouvoir être modulable.

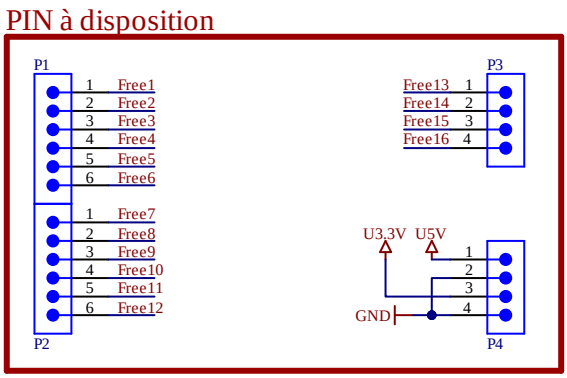


Figure 20 Pins à disposition

9 BOM

La BOM (la liste de composants/pièces) se trouve en annexe N° 7.

Réalisation

Maintenant que le schéma est réalisé, je vais commencer la réalisation du PCB.

10 Réalisation du PCB

Lors de la réalisation de mon PCB, n'ayant aucune contrainte de taille mise à part la taille du Sabertooth afin de le fixer sur le circuit, ce qui m'a donné la valeur de la largeur du PCB.

Les dimensions de mon PCB sont 208.28 x 69.85 mm.

J'ai mis des trous de M3 pour venir fixer le Sabertooth ainsi que pour mettre des pieds au circuit.

Pour le routage, toutes mes pistes sont en 17 mil d'épaisseur. Mon PCB contient 4 plans de masse, 2 du côté du driver moteur (TOP et BOTTOM) ainsi que 2 du côté de mes composants (TOP et BOTTOM), avec la possibilité de les connecter entre eux avec de la brasure. Des vias stitching sont implémentés afin de relier le TOP et le BOTTOM.

J'ai mis tous mes composants au TOP du PCB avec les connecteurs (Alim 24V, gyroscope, 4x capteurs de distance, 2x potentiomètres et lignes de commandes pour le driver moteur) implémentés de façon à ce que l'accès à ceux-ci se fasse facilement.

J'ai mis aussi à disposition plusieurs points de test SMD sur le PCB afin d'effectuer facilement les mesures.

Une fois mon PCB fini, j'ai vérifié qu'il correspondait bien à la classe 6C de Eurocircuits, ce qui était correct.

11 Coût total du projet

Pour avoir un cout total détaillé, veuillez-vous référer à la BOM à l'annexe N° 7.

Mais voici un petit résumé : pour les composants et le PCB, cela a couté à l'ETML-ES 108.67 fr + 64.14 fr donc 172.81 fr.
Pour la structure métallique, les moteurs ainsi que le driver moteur et la licence Simtools, ils ont été payés à mes frais et sa m'est revenu à environ 720 fr.

12 Vue du PCB avec les différents blocs

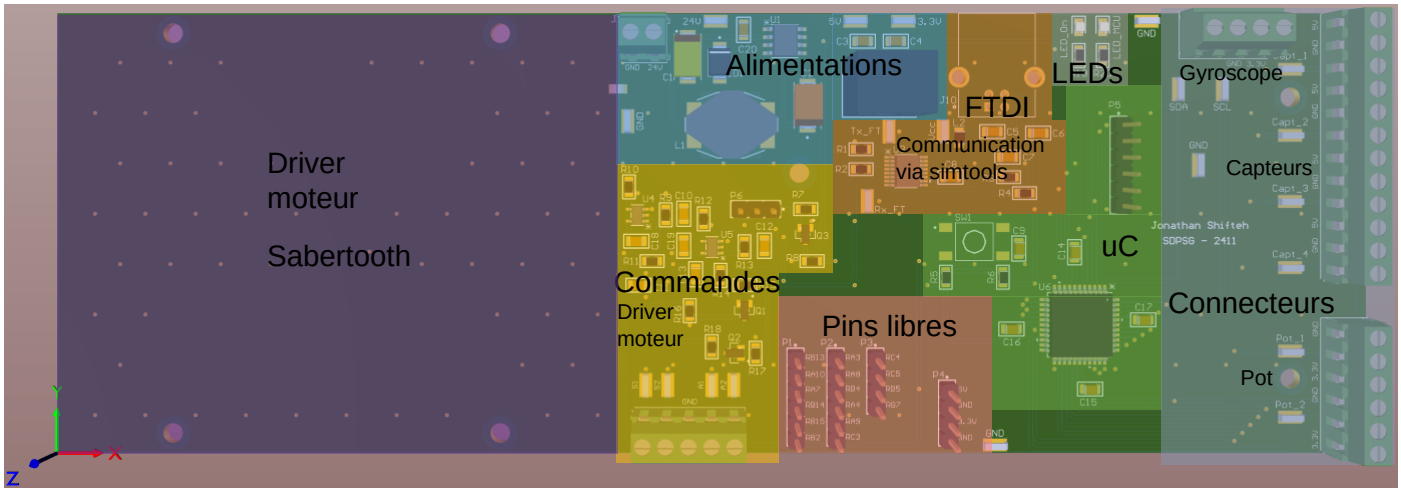


Figure 21 PCB avec vue sur les blocs

13 Vues des couches de mon PCB

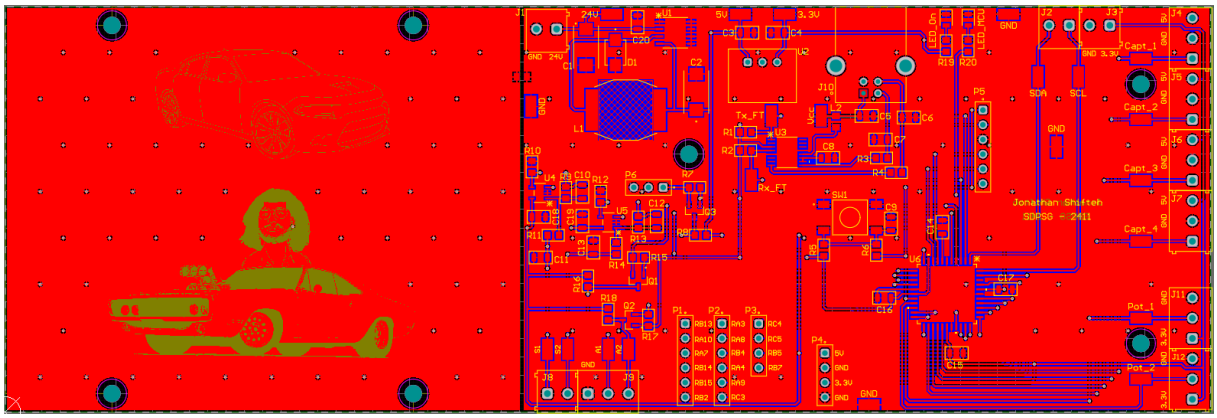


Figure 22 Vue du TOP

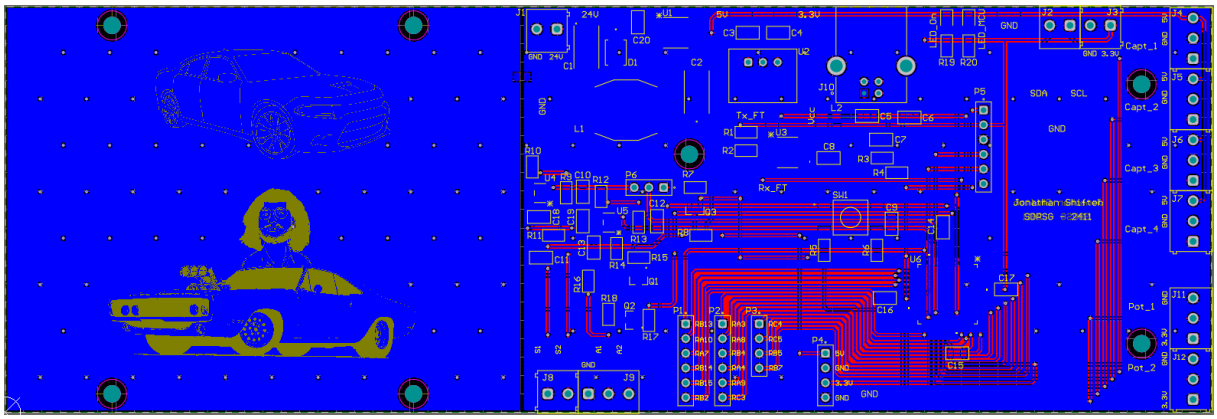


Figure 23 Vue du BOTTOM

14 Vue 3D de mon PCB

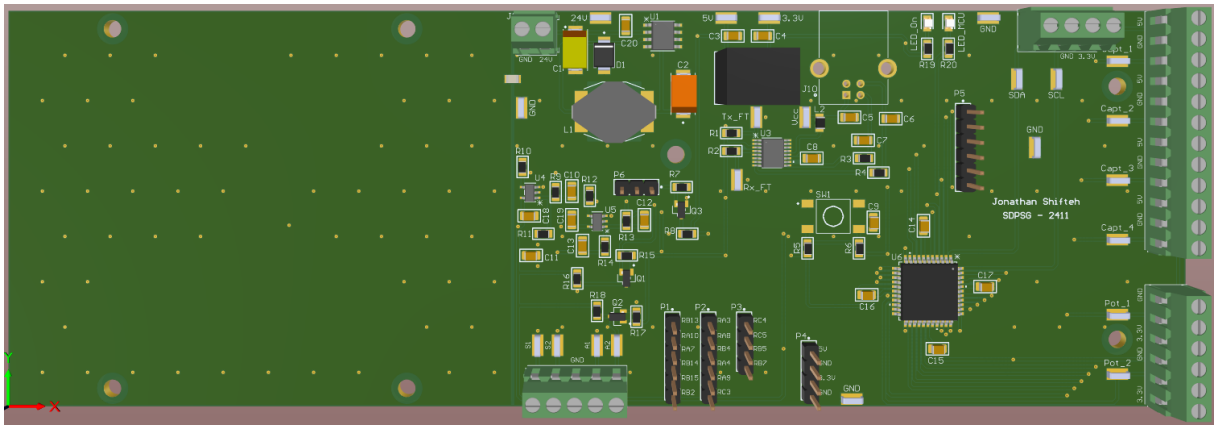


Figure 24 Vue du TOP en 3D

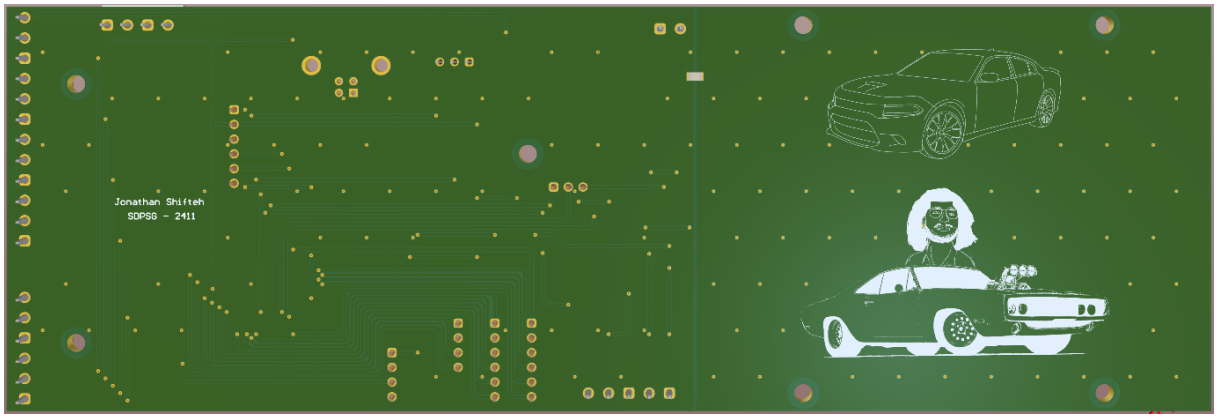


Figure 25 Vue du BOTTOM en 3D

15 Test le hardware

Pour tester mon hardware, je suis d'abord venu braser l'alimentation 5V avec ces composants. Une fois cela fonctionnel, je suis venu faire l'alimentation 3.3V. Après mes alimentations montées et fonctionnelles, je suis passé au uC et j'ai testé qu'il soit programmable.

Pour le reste, j'ai pu tout monter directement.

Protocole de test			
Tâche n°	Montage à braser	OK/NOK	
1	Alim 5V	OK	Erreur au début, montage diode (D1) inversé
2	Alim 3.3V	OK	
3	uC	OK	Test programmable OK
4	Logic Level/cmd driver	OK	Après débrasage de R7/R8
5	Connecteur	OK	Test capteurs OK, test pots OK (PIN config en ADC NOK)
6	FT230XS	OK	Test communication avec Simtools OK

15.1 Modification à faire

J'ai eu un problème sur ma ligne de commande S1 car j'avais mis un jumper pour choisir si ma ligne de commande était en analogique ou en série. Cependant, j'aurais dû remettre un jumper à la sortie afin que mes 2 méthodes ne soient pas en contact direct, car là les pull-up R7 et R8 me tirent la tension vers le haut. Pour y remédier, j'ai débrasé ces 2 résistances.

Software/Firmware

Maintenant que le PCB est réalisé, je vais configurer et programmer ma carte ainsi que configurer SimTools et le driver moteur.

16 Configuration Simtools

SimTools est divisé en 3 applications distinctes. Donc voici une petite explication de comment programmer SimTools afin de recevoir les informations du jeu.

En premier lieu, il faut d'abord patcher le jeu qu'on veut utiliser afin qu'il communique avec SimTools. Pour ce faire, il y a une application nommée SimTools Plugin Updater où il suffit de glisser le fichier du patch.

Figure 26 SimTools Plugin Updater

Une fois le jeu patché, il faut lancer l'application SimTools Game Manager, celui-ci servira de passerelle entre le jeu et SimTools. Lorsque le jeu sera lancé, il faudra cliquer sur le bouton 1.

Figure 27 SimTools Game Manager

Mais avant qu'on reçoive une information sur SimTools Game Engine, il faut configurer les adresses IP entre ces deux applications. Vu que mes deux applications sont sur le même ordi, l'adresse IP qu'on va leur indiquer est '127.0.0.1'.

Sur Game Manager, il faut aller dans le menu Tools puis Engine IP.

Sur Game Engine, il faut aller dans le menu Tools puis Manager IP.

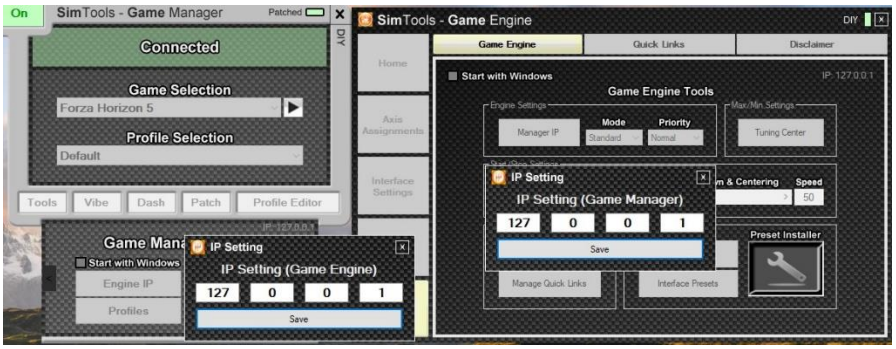


Figure 28 Set Address IP

Une fois cela bien configuré, sur SimTools Game Engine, nous pouvons configurer les différents pourcentages des mouvements sur l'axe en question.

Pour ma part, je préfère avoir une plus grande intensité sur les mouvements d'accélération/décélération (effet Surge) que sur le reflet de la pente (effet Roll).

Figure 29 Set Axis Assignments

Comme vous pouvez l'apercevoir, j'ai mis 70% sur l'effet Surge et 40% sur l'effet Roll. La case Dir sert à inverser si on s'aperçoit que le siège bouge dans le mouvement inverse souhaité.

Maintenant, il ne nous reste plus qu'à configurer la communication série afin de récolter les informations.

Cependant, lors de mes tests, ça ne fonctionnait pas et je ne comprenais pas pourquoi. Après plusieurs échecs, j'ai tout simplement changé l'interface utilisée, je suis passé de l'interface 1 à l'interface 2 avec les mêmes paramètres et ça fonctionnait. Donc ne jamais utiliser l'interface 1 car elle est buggée.

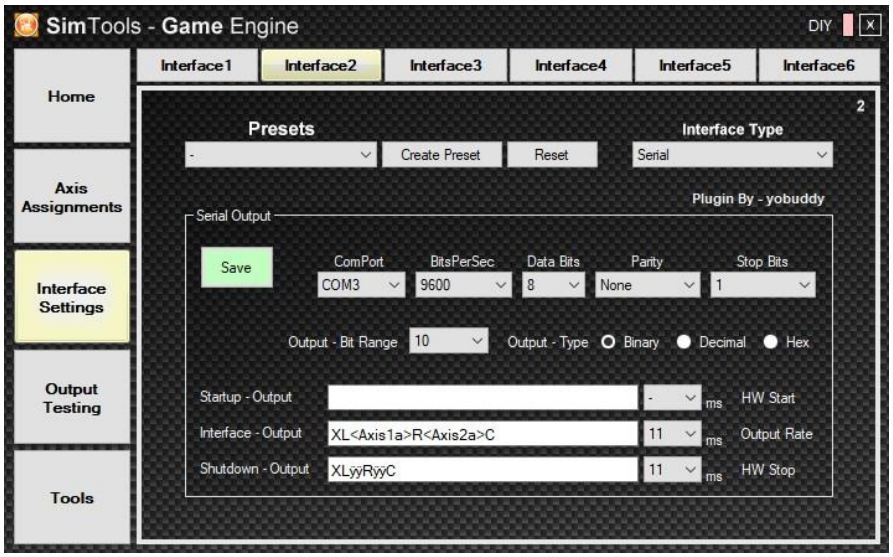


Figure 30 Interface Settings

Voici la configuration de l'interface. Ici, on vient mettre les paramètres de la communication série puis, nous pouvons choisir le style de trame qu'on veut sous Interface – Output. J'ai choisi la trame 'XL<valeurs de l'axe 1>R<valeurs de l'axe 2>C', ce qui me permettra, en identifiant les caractères XL-R-C, de décoder les valeurs des axes.

Le Shutdown – Output servira à remettre le siège dynamique en position initiale lors de la fin de la session gaming.

17 Configuration d'Harmony (MPLAB)

17.1 Réglage des pins du uC

Tout au début de la phase de programmation, je suis venu définir les différents pins de mon uC en fonction de mon étude et de mon schéma fini au préalable.

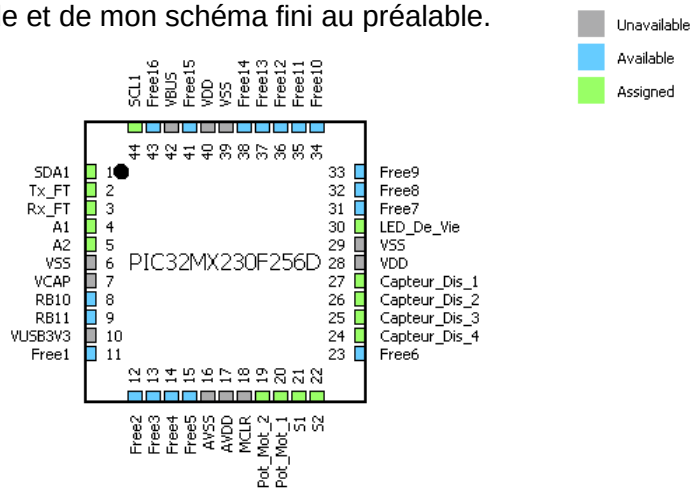


Figure 31 Pins diagram

Voici les paramètres mis à chaque PINs.

Pin Number	Pin ID	Voltage Tolerance	Name	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)	Change Notification (CNEN)	Pull Up (CNPU)	Pull Down (CNPD)
1	RB9	5V	SDA1	SDA1	n/a	n/a	☐	Digital	☐	☑	☐
2	RC6	5V	Tx_FT	U1RX	n/a	n/a	☐	Digital	☐	☐	☐
3	RC7	5V	Rx_FT	U1TX	n/a	n/a	☐	Digital	☐	☐	☐
4	RC8	5V	A1	GPIO_OUT	Out	Low	☐	Digital	☐	☐	☐
5	RC9	5V	A2	GPIO_OUT	Out	Low	☐	Digital	☐	☐	☐
8	RB10		PGED	Available	In	n/a	☐	Analog	☐	☐	☐
9	RB11		PGEC	Available	In	n/a	☐	Analog	☐	☐	☐
11	RB13		Free1	Available	In	n/a	☐	Analog	☐	☐	☐
12	RA10	5V	Free2	Available	In	n/a	☐	Digital	☐	☐	☐
13	RA7	5V	Free3	Available	In	n/a	☐	Digital	☐	☐	☐
14	RB14		Free4	Available	In	n/a	☐	Analog	☐	☐	☐
15	RB15		Free5	Available	In	n/a	☐	Analog	☐	☐	☐
18	MCLR	5V	MCLR		In	n/a	☐	Digital	☐	☐	☐
19	RA0		Pot_Mot_2	GPIO_IN	In	Low	☐	Analog	☐	☐	☐
20	RA1		Pot_Mot_1	GPIO_IN	In	Low	☐	Analog	☐	☐	☐
21	RB0		S1	OC3	n/a	n/a	☐	Digital	☐	☐	☐
22	RB1		S2	OC2	n/a	n/a	☐	Digital	☐	☐	☐
23	RB2		Free6	Available	In	n/a	☐	Analog	☐	☐	☐
24	RB3		Capteur_Dis_4	AN5	n/a	n/a	☐	Analog	☐	☐	☐
25	RC0		Capteur_Dis_3	AN6	n/a	n/a	☐	Analog	☐	☐	☐
26	RC1		Capteur_Dis_2	AN7	n/a	n/a	☐	Analog	☐	☐	☐
27	RC2		Capteur_Dis_1	AN8	n/a	n/a	☐	Analog	☐	☐	☐
30	RA2		LED_De_Vie	GPIO_OUT	Out	Low	☐	Digital	☐	☐	☐
31	RA3		Free7	Available	In	n/a	☐	Analog	☐	☐	☐
32	RA8	5V	Free8	Available	In	n/a	☐	Digital	☐	☐	☐
33	RB4		Free9	Available	In	n/a	☐	Analog	☐	☐	☐
34	RA4		Free10	Available	In	n/a	☐	Analog	☐	☐	☐
35	RA9	5V	Free11	Available	In	n/a	☐	Digital	☐	☐	☐
36	RC3		Free12	Available	In	n/a	☐	Analog	☐	☐	☐
37	RC4	5V	Free13	Available	In	n/a	☐	Digital	☐	☐	☐
38	RC5	5V	Free14	Available	In	n/a	☐	Digital	☐	☐	☐
41	RB5	5V	Free15	Available	In	n/a	☐	Digital	☐	☐	☐
43	RB7	5V	Free16	Available	In	n/a	☐	Digital	☐	☐	☐
44	RB8	5V	SCL1	SCL1	n/a	n/a	☐	Digital	☐	☑	☐

Figure 32 Pin settings

17.2 Réglage Harmony

17.2.1 Timers

Pour réaliser à bien mon projet, j'aurai besoin de 2 timers.

Le 1er servira à venir cadencer mon programme principal et le 2ième servira comme base de temps pour mes OCs.

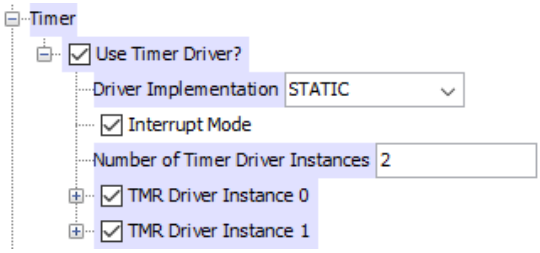


Figure 33 Activations des timers

17.2.1.1 Timer 1

Je suis venu alors activer et configurer le timer 1. C'est celui qui gèrera la machine d'état principal.

J'ai défini de mettre à jour la machine d'état toutes les 10 ms.

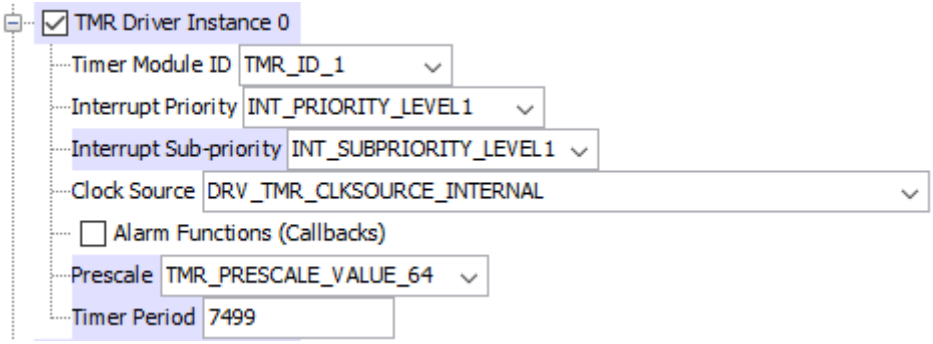


Figure 34 Configuration timer 1

Pour le configurer, j'ai dû venir calculer le Prescale minimum ainsi que le nombre de Timer Period (tics).

Pour le prescale :

$$t_{uc} = \frac{1}{f_{uc}} = \frac{1}{48 * 10^6} = 20.83 * 10^{-9}\ [s]$$

$$Prescaler = \left(\frac{t_{voulu}}{t_{uc} * Bits_{timer}}\right) = \left(\frac{10 * 10^{-3}}{20.83 * 10^{-9} * 2^{16}}\right) = 7.32\ [-]$$

Suivant ce calcul, je sais que je dois mettre au minimum un prescaler de 8. J'ai pris l'habitude de toujours prendre le prescaler au-dessus du minimum afin que le Timer Period soit plus petit. Donc je suis parti sur un prescaler de 64.

Grâce à ça, je peux venir calculer le nombre de tics que j'aurai dans mon Timer Period.

Pour le Timer Period :

$$timer\ period = \left(\frac{t_{voulu}}{t_{uc} * Prescaler}\right) - 1 = \left(\frac{10 * 10^{-3}}{20.83 * 10^{-9} * 64}\right) - 1 = 7'499[-]$$

J'ai mis ce timer en interrupt priorité de level 1 et sub-priorité level 1 car c'est le timer principal de mon programme, l'autre timer vient juste comme référence au OCs donc n'est pas prioritaire sur le timer 1 et aura des priorités de level 0. Mais j'ai aussi la communication série auquel je vais venir mettre une plus grande priorité afin que mes trames UART ne soient jamais coupées.

Vous pouvez retrouver une capture d'oscilloscope afin de démontrer les 10 ms (annexe N° 9).

17.2.1.2 Timer 2

Ici, je suis venu activer et configurer le timer 2. C'est celui-ci qui gèrera la base de temps de mes OCs.

Le temps que fait le timer est défini par la fréquence des PWMs qui vont commander mes moteurs.

Suite à une discussion avec Monsieur Gavin, on a défini qu'une fréquence de 20 kHz était parfaite pour cette utilisation. Une fréquence de 20 kHz permet d'éviter les bruits indésirables et d'assurer un bon contrôle.

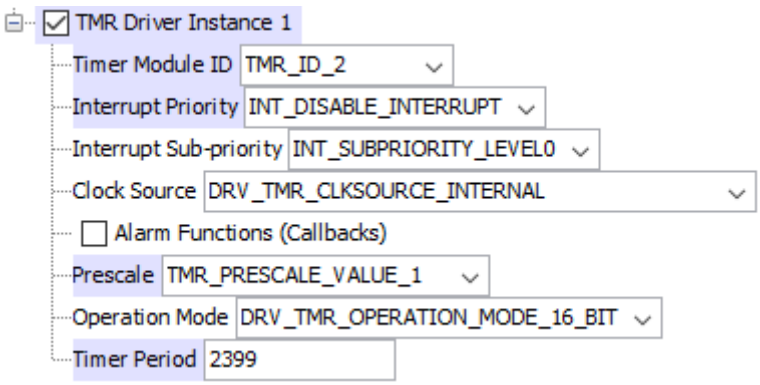


Figure 35 Configuration timer 2

Pour le configurer, j'ai dû venir calculer le prescale minimum ainsi que le nombre de Timer Period.

Pour le prescale :

$$t_{uc} = \frac{1}{f_{uc}} = \frac{1}{48*10^6} = 20.83 * 10^{-9}\ [s] \qquad t_{oc} = \frac{1}{f_{oc}} = \frac{1}{20*10^3} = 50 * 10^{-6}\ [s]$$

$$Prescaler = \left(\frac{t_{voulu}}{t_{uc} * Bits_{timer}}\right) = \left(\frac{50 * 10^{-6}}{20.83 * 10^{-9} * 2^{16}}\right) = 36.62 * 10^{-3}\ [-]$$

Suivant ce calcul, je sais que je dois mettre au minimum un prescaler de 1.

Grâce à ça, je peux venir calculer le nombre de tics que j'aurai dans mon Timer Period.

Pour le Timer Period :

$$timer\ period = \left(\frac{t_{voulu}}{t_{uc} * Prescaler}\right) - 1 = \left(\frac{50 * 10^{-6}}{20.83 * 10^{-9} * 1}\right) - 1 = 2'399[-]$$

J'ai mis ce timer en interrupt priorité désactivée car je n'ai pas besoin d'interrupt afin de gérer mes OCs.

17.2.2 OC

Voici la configuration de l'OC2. Pour l'OC3, la configuration reste identique.

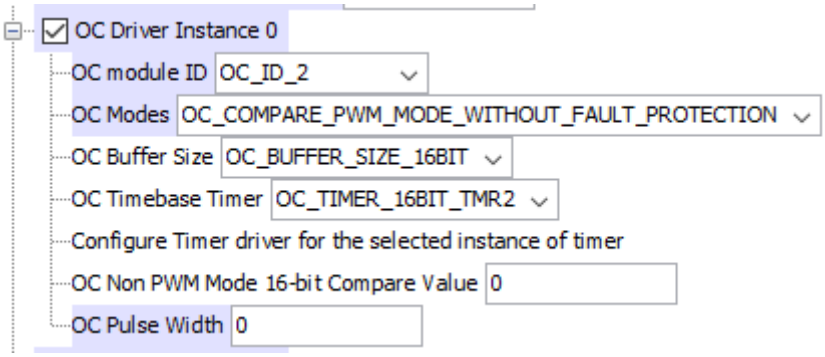


Figure 36 Configuration OC2

L'OC 2 a été configuré sur la base de temps du timer 2.

J'ai mis le mode 'OC_COMPARE_PWM_MODE_WITHOUT_FAULT_PROTECTION' pour pouvoir varier le rapport cyclique dans le code sans que la période ne change.

17.2.3 UART

Le bus USART_ID_1 me servira afin de communiquer avec Simtools pour recevoir la télémétrie du jeu vidéo. Je l'ai configuré par rapport aux configurations de l'interface de Simtools que vous pouvez retrouver au point 16.

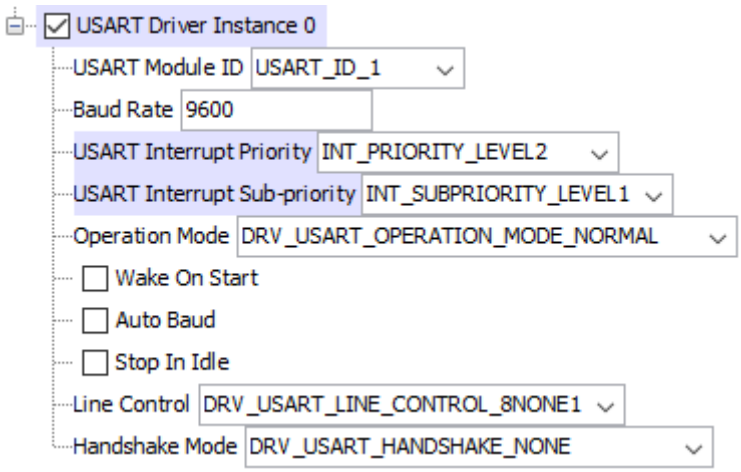


Figure 37 Configuration UART

18 Réalisation du code

18.1 Diagramme d'état du logiciel

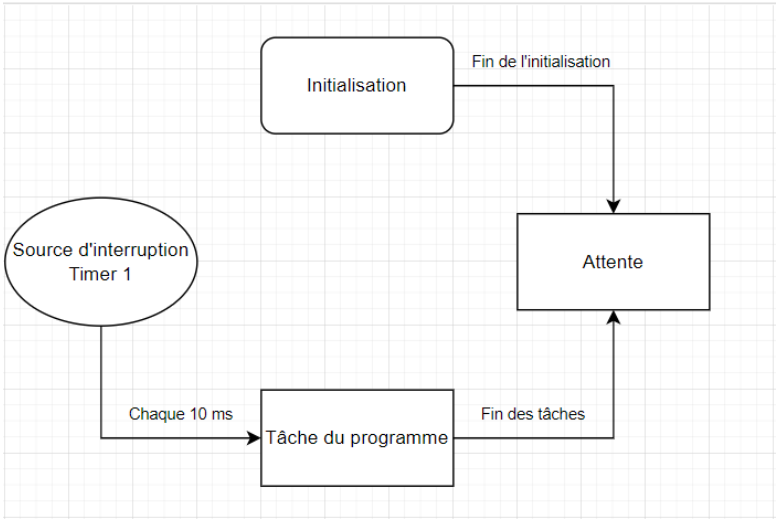


Figure 38 Diagramme d'état de mon logiciel

18.2 App_Tasks

J'ai une machine d'état qui est prise du modèle de base de l'ETML-ES dans le fichier app.c.

Le premier état sera l'APP_STATE_INIT (Initialisation) qui me servira à initialiser mes différents périphériques ainsi que les variables.

Une fois que cet état a fini son exécution, la machine d'état se mettra en mode APP_STATE_WAIT (Attente). Dans cet état, on attend et on ne fait rien d'autre.

Puis, à chaque 10 ms, grâce à l'interrupt du timer 1, le programme se mettra dans l'état APP_STATE_SERVICE_TASKS (Tâche du programme). Dans cet état, on exécutera les différentes commandes pour avoir le bon fonctionnement du projet.

Voici un structogramme du code implémenté avec ces différents états.

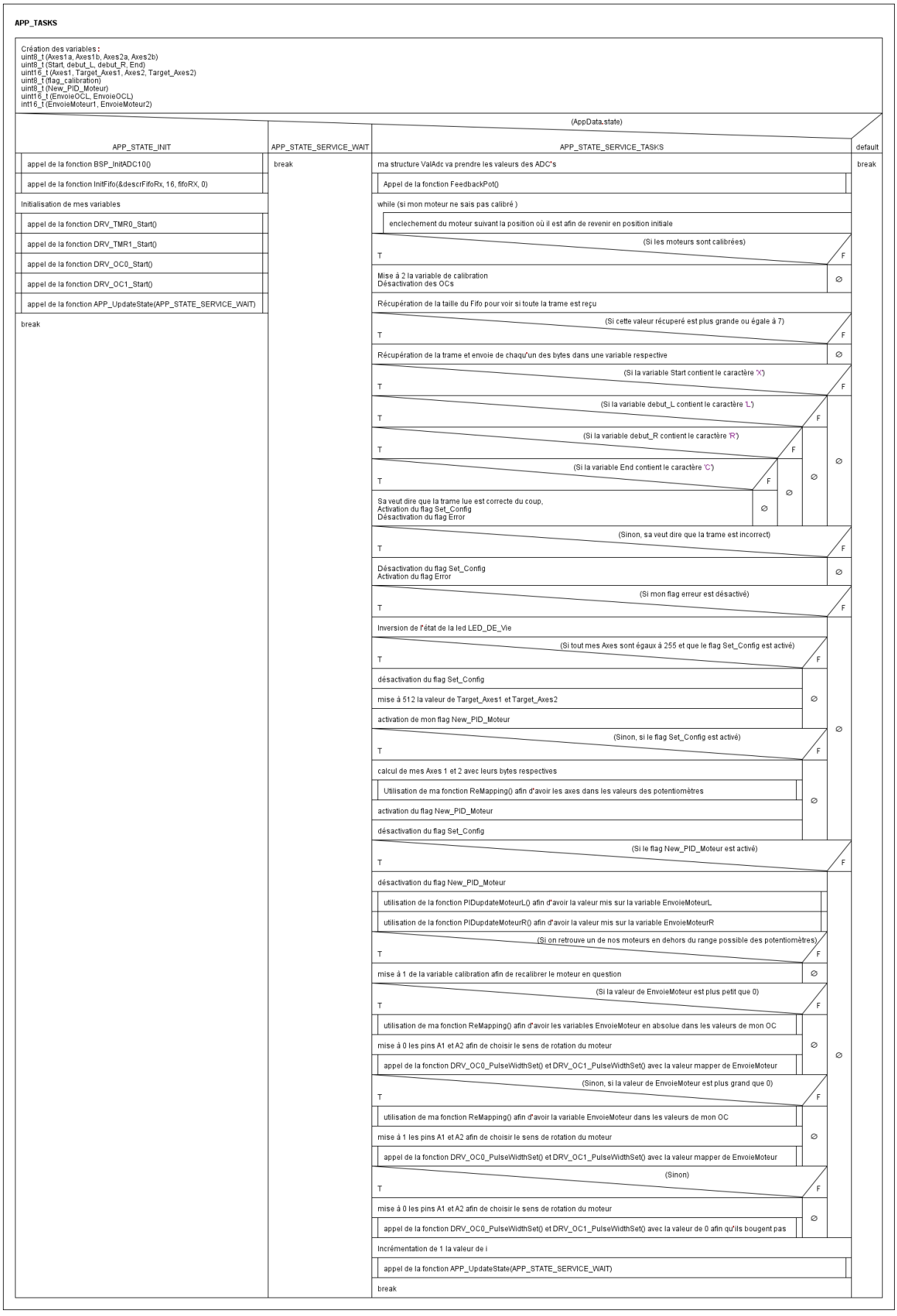


Figure 39 Structogramme de APP_TASKS

18.3 Fonction PID

Cette fonction PIDupdateMoteur me servira à calculer le PID et celle-ci me renverra une valeur qui lui gérera la vitesse à envoyer à l'OC. Cette fonction est doublée, une pour chacun des moteurs dont le nom sera PIDupdateMoteurL et PIDupdateMoteurR.

Pour paramétrer les différentes valeurs des gains PI utilisés, je suis venu faire comme dans les différents TP, donc par tâtonnement.

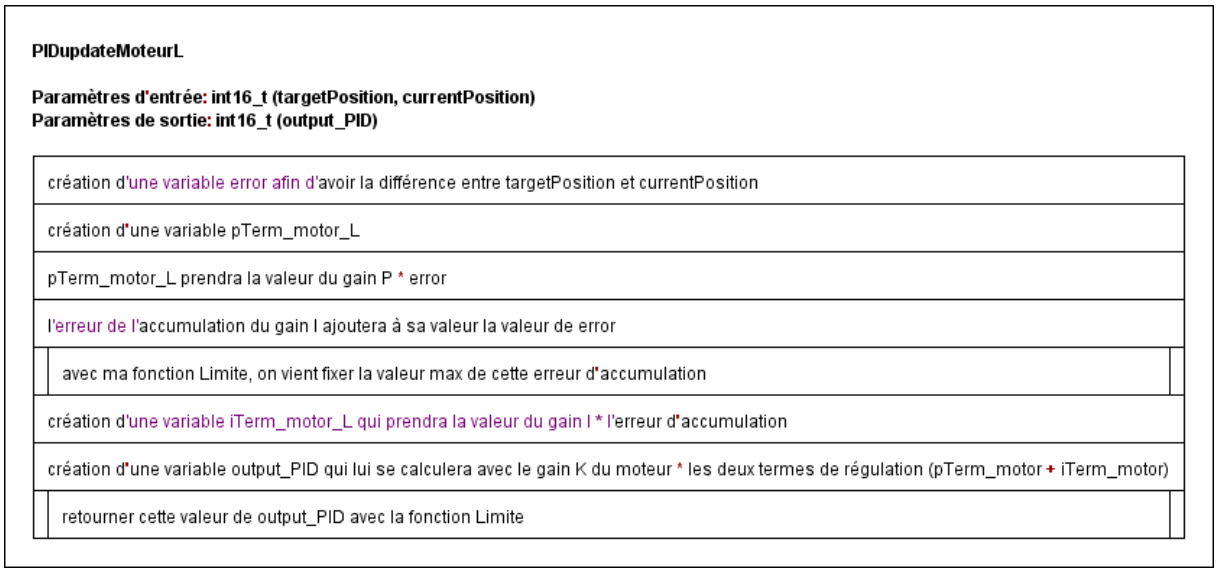


Figure 40 Fonction PIDupdateMoteur

18.4 Fonction FeedbackPot

Cette fonction FeedbackPot me servira afin de récolter l'information de la position du moteur avec le jeu de roues dentées. Du coup, cette fonction me permet de lire 10 fois les potentiomètres et d'en faire une moyenne afin de retourner la valeur brute de ceux-ci.

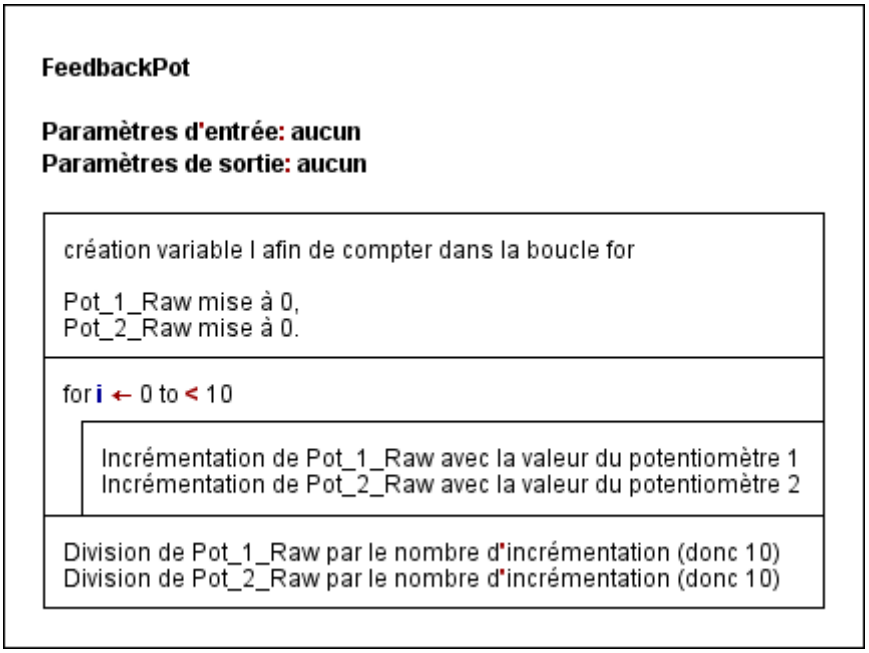


Figure 41 Structogramme de la fonction FeedbackPot

18.5 Fonction Limite

Cette fonction me permet de mettre une limite par rapport à la valeur et aux paramètres d'entrée min/max afin de forcer, en cas de dépassement, à retourner la valeur maximum respective.

Donc, si la valeur est plus petite que le minimum, elle va me retourner la valeur min passée en paramètre. Sinon, si la valeur est plus grande que le maximum, elle me retourne la valeur max. Sinon, elle retourne juste la valeur passée en paramètre sans modification de celle-ci.

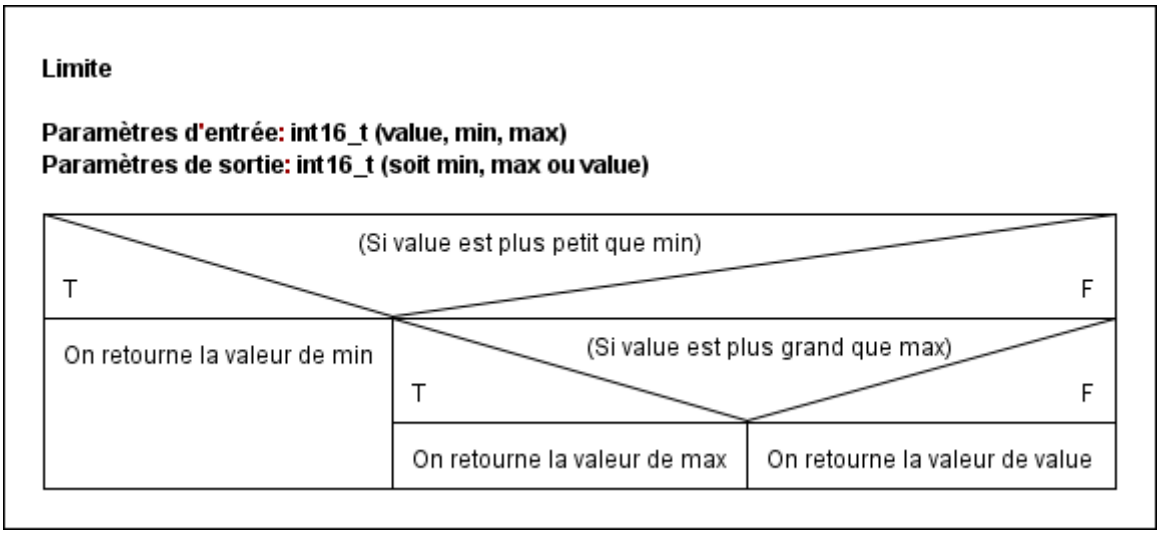


Figure 42 Structogramme de la fonction Limite

18.6 Fonction ReMapping

Cette fonction permet de transformer n'importe quelle plage de valeurs dans une autre plage de valeurs choisie. Pour en démontrer le fonctionnement, je vais vous faire un petit exemple en transformant la valeur de 1.65 V, qui est la moitié d'une plage de 0 - 3.3 V, dans une plage de 0 – 5 V.

Exemple : les valeurs passées en paramètres d'entrée sont X = 1.65, in_min = 0, in_max = 3.3, out_min = 0, out_max = 5.

Donc [(1.65 – 0) * (5 – 0)] / (3.3 – 0) + 0 = 2.5, ce qui coïncide parfaitement car la moitié de la plage de 0 - 5 V est 2.5 V.

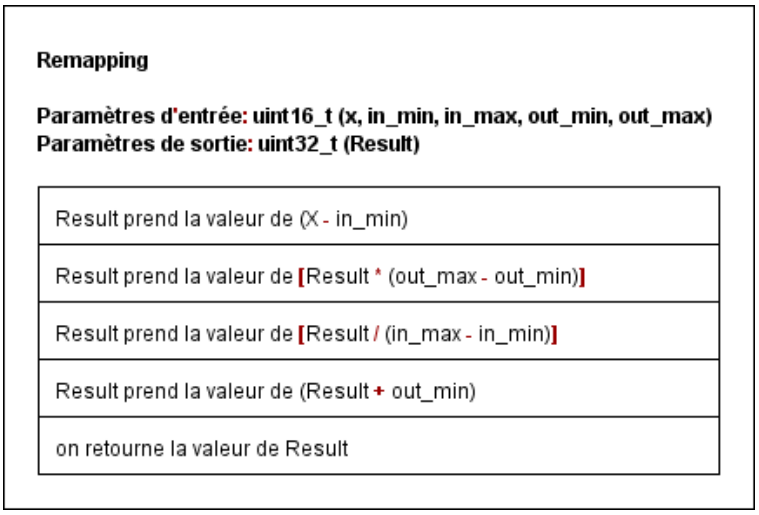


Figure 43 Structogramme de la fonction ReMapping

18.7 Modification librairie ADC

J'ai dû venir modifier les fichiers de notre librairie de l'école afin de lire les bons ADC's du uC pour récolter les valeurs de mes potentiomètres et de mes capteurs.

18.7.1 Mc32DriverAdc.c

Tout d'abord, je suis venu modifier le configscan afin qu'il détecte tous mes ADC's.

```
#define configscan  0x01E3      // SCAN AN8 AN7 AN6 AN5 AN1 AN0 (Pots du kit)
```

Une fois cela fait, j'ai dû modifier le nombre de samples afin qu'il lise les 6 ADC's.

```
// Rem CHR le nb d'échantillon par interruption doit correspondre au nb d'entrées
// de la liste de scan
PLIB_ADC_SamplesPerInterruptSelect(ADC_ID_1, ADC_6SAMPLES_PER_INTERRUPT);
```

Puis, je suis venu chercher les valeurs de la lecture pour les envoyer sur ma structure.

```
if (BufStatus == ADC_FILLING_BUF_0TO7) {
    result.Chan0 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 0);
    result.Chan1 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 1);
    result.Chan2 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 2);
    result.Chan3 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 3);
    result.Chan4 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 4);
    result.Chan5 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 5);
} else {
    result.Chan0 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 8);
    result.Chan1 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 9);
    result.Chan2 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 10);
    result.Chan3 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 11);
    result.Chan4 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 12);
    result.Chan5 = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 13);
}
```

18.7.2 Mc32DriverAdc.h

Dans ma structure, je suis venu créer les 6 ADC's afin de pouvoir les utiliser.

```
typedef struct {
    uint16_t Chan0;
    uint16_t Chan1;
    uint16_t Chan2;
    uint16_t Chan3;
    uint16_t Chan4;
    uint16_t Chan5;
} S_ADCResults;
```

Mesures et problèmes rencontrés

19 Matériel de mesure

Durant ce projet, j'ai utilisé plusieurs appareils de mesure pour tester le fonctionnement de mon prototype. Voici une liste des appareils utilisés au cours des cinq dernières semaines :

Marque	Type	Caractéristique	N° Inventaire
GW Instek	GDM-396	Multimètre	ES.SLO2.00.00.77
Rohde&Schwarz	RTB2004	Oscilloscope	ES.SLO2.05.01.03

20 Mesure sur le système

20.1 Mesure des tensions d'alimentations sur le PCB

Je suis venu mesurer mes différentes tensions grâce aux pointes de mesures préalablement implémentées sur mon PCB. Du coup, dans le tableau, vous pouvez retrouver les valeurs attendues et les valeurs mesurées.

Puis, j'ai pu venir mesurer grâce aux pointes de mesures. Les pointes de mesures pour le GND sont TP19/TP20/TP21/TP22.

Tension Voulue	Tension mesurée	OK/NOK	Pointes de mesures
24.0 V	23.93 V	OK	TP1 (24V)
5.0 V	5.0 V	OK	TP2 (5V)
3.3 V	3.30 V	OK	TP3 (3.3V)

20.2 Mesure de la trame UART

Afin de savoir ce que SimTools m'envoie comme trame, je suis venu faire une mesure de celle-ci avec l'oscilloscope et son mode protocole afin qu'il décode celle-ci.

20.2.1 Schéma de mesure

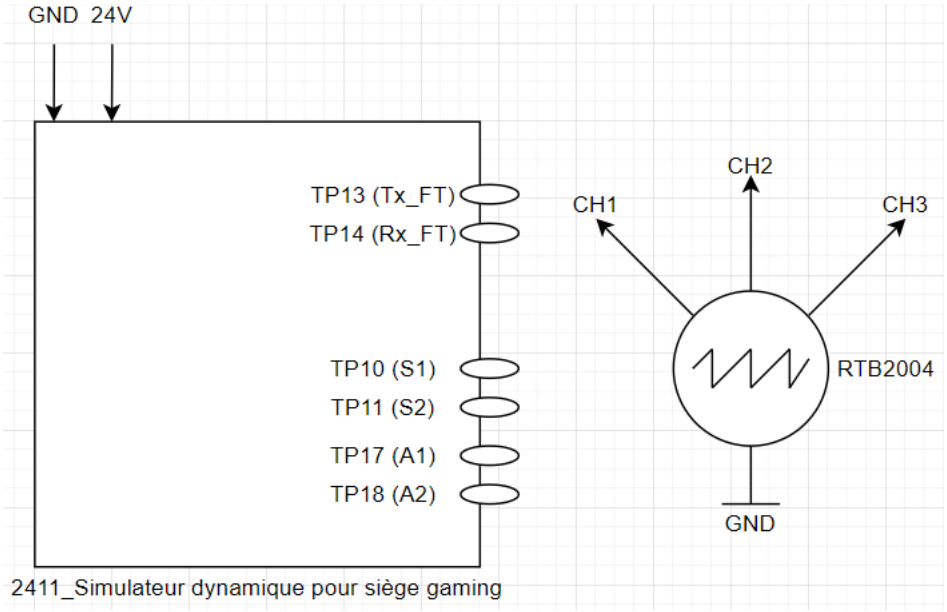


Figure 44 Schéma de mesure

20.2.2 Protocole de mesure

Pour venir mesurer ma trame, j'ai mis le canal 1 de l'oscilloscope sur TP14 et le canal 2 sur TP13. Puis, dans le mode protocole de l'oscilloscope, je suis venu configurer le mode Serial et, je suis venu notamment configurer les canaux afin que le 1 soit la configuration RX et le canal 2 TX. J'ai dû aussi mettre la même parité que sur SimTools, ainsi que les bits de stop et le baudrate.

20.2.3 Résultat

Petit rappel de la trame attendue : XL<valeurs de l'axe 1>R<valeurs de l'axe 2>C

J'ai fait plusieurs tests mais voici 2 des tests effectués : une fois la trame lorsque mes curseurs sur SimTools étaient au milieu de leurs valeurs, donc je dois pouvoir voir la valeur de 511 dans mes 2 axes. Et aussi la trame que m'envoie SimTools lors de l'arrêt de celui-ci, qui est

normalement deux fois le caractère ÿ par axe, car je savais que les informations du jeu ne pouvaient jamais me renvoyer 2 bytes à la suite valant ÿ (la valeur maximum est 1023).

20.2.3.1 Test n°1

Sachant que ÿ vaut 255 dans le tableau ascii, j'ai pu décoder la trame comme ceci :

0x01h = 1, ÿ = 255. $Valeur\ de\ l'axe = (0x01h * 256) + ÿ = (1 * 256) + 255 = 511$

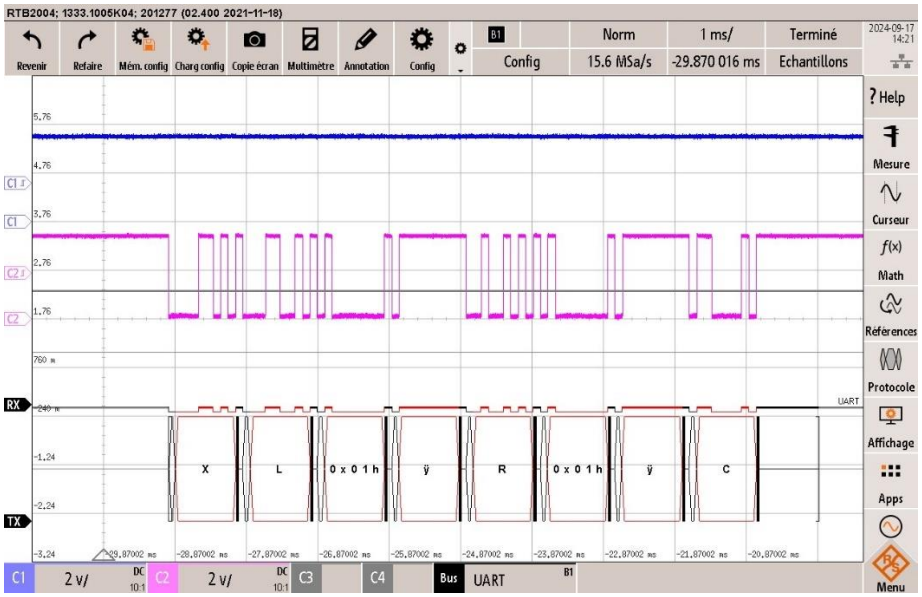


Figure 45 Mesure du test N°1

20.2.3.2

Voici la trame lors de la fermeture de la communication de Simtools.

Comme attendu, nous avons bien les 2 bytes ÿ. Grâce à ça, dans mon programme, je vais venir repositionner mon siège à l'angle de base.

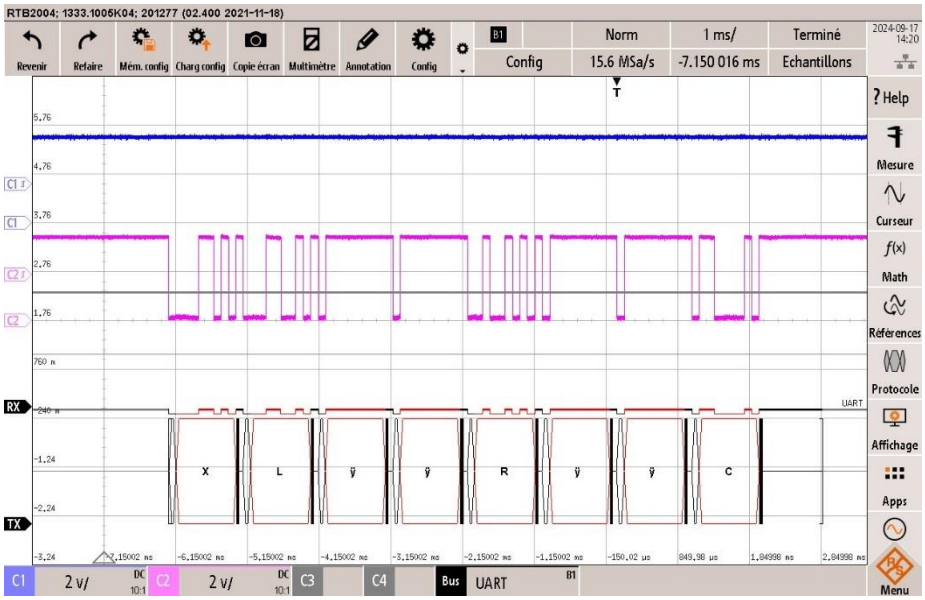


Figure 46 Mesure du test N°2

21 Problèmes rencontrés

Le premier problème est apparu lors du protocole de mesure. Dans mes lignes Logic Level, j'aurais dû mettre aussi un jumper sur la ligne Driver_mot_S1_reel afin de pouvoir choisir comment communiquer avec le driver moteur (analogique ou série). Pour y remédier, j'ai débrasé les résistances R7/R8, ce qui fait que je peux seulement communiquer en analogique.

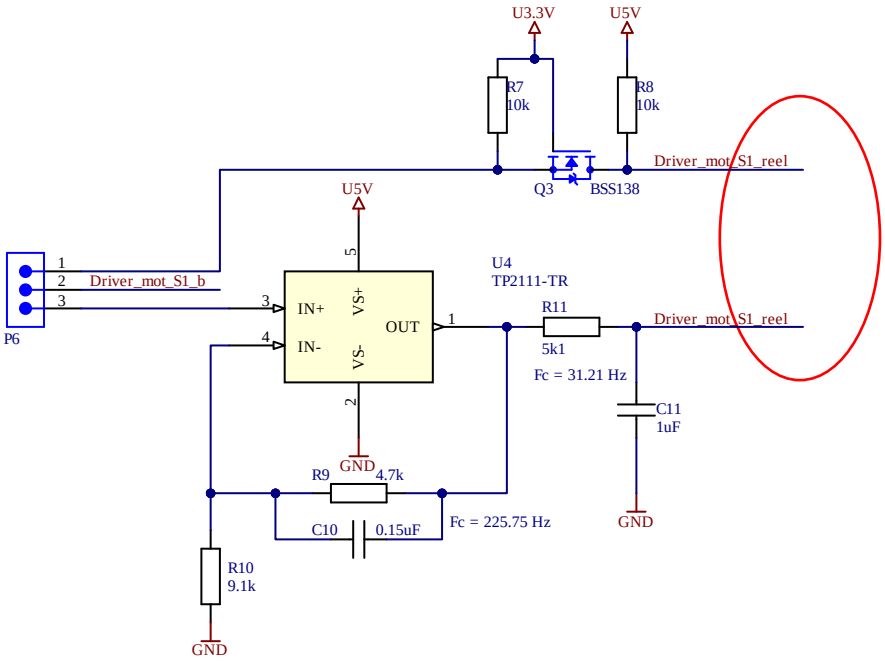


Figure 47 Schéma partie logic level

Le deuxième problème était lors de l'utilisation de SimTools, car d'abord, j'ai téléchargé la V3 de SimTools, sauf que la communication serial avec celle-ci ne fonctionnait pas. Pour pouvoir communiquer avec le jeu, j'ai dû télécharger la V2.4, mais suite à ça, c'était le plugin du jeu que je ne pouvais plus charger. J'ai dû en trouver un autre plugin de Forza Horizon 5 qui était fait pour la V2.4.

Le troisième problème survient lors de l'enclenchement de l'alimentation. Car une fois que le uC est alimenté, n'ayant pas mis de pull-down sur les pattes RB0/RB1, qui me sortent les OCs pour piloter la vitesse des moteurs, l'état de ces pins se retrouve en état flottant, ce qui provoque une tension à la sortie, donc ça fait tourner mes moteurs. Ce qui pose un problème, car j'ai des contraintes mécaniques.

J'ai eu aussi un grand délai d'attente pour la conception de mes pièces 3D faites par un professeur de l'ETML-ES, notamment dû à des problèmes techniques d'imprimante. Donc, je n'ai pas eu le temps de perfectionner mon code après les différents tests effectués avant le rendu du rapport.

J'ai rencontré un autre problème, mais cela n'était pas très grave, car c'était une partie optionnelle. J'ai voulu mettre un gyroscope sur mon système, sauf que je n'arrivais pas à récolter les valeurs des axes X, Y et Z. Mais je communiquais bien avec celui-ci et je venais lire dans les bons registres. Pour ne pas perdre trop de temps avec ce problème, je suis passé aux éléments essentiels de mon système.

Malheureusement, dans les derniers jours du projet, ma carte graphique commençait à rendre l'âme. Lorsque je lance le jeu, ma carte graphique se bloque et l'ordinateur redémarre. J'ai essayé de corriger ça, mais rien à faire… je dois changer de carte graphique. C'est pourquoi,

si le problème n'a pas encore été résolu, je présenterai mon projet avec une simulation de données via Simtools. En utilisant Output Testing dans Simtools, il faut savoir que les valeurs sont plus agressives que lorsque les valeurs sont prises dans le jeu, donc la régulation peut être moins efficace.

22 Etat actuel du projet

22.1 Tâches effectuées

Durant ces 5 semaines, j'ai réussi à communiquer entre le jeu vidéo et Simtools, puis à envoyer cette trame sur mon PCB.

J'ai mis en place un programme qui récoltait cette trame puis la décodait afin de calculer le PID à envoyer au moteur.

Cependant, le PID n'est pas optimal et peut encore mieux se réguler avec différents tests en modifiant les gains PI. J'ai aussi supprimé le gain D, car il augmentait mes oscillations sur mes moteurs.

J'ai pu créer des fonctions pour l'utilisation des capteurs de distances GP2Y0A21YK0F.

22.2 Tâches restantes

Il me reste encore à ajuster ce PI afin que le simulateur soit plus réactif sans osciller et, si possible, à faire fonctionner ce gyroscope en récoltant les axes X, Y et Z.

Déroulement du diplôme

23 Apports du travail de diplôme

Grâce à ce travail de diplôme, j'ai pu acquérir des compétences dans la conception d'un simulateur avec ses contraintes mécaniques et améliorer celles de la conception d'un PCB.

Puis, j'ai pu mettre en pratique ce qu'on avait vu en leçon de réglage afin de gérer mes moteurs Wiper avec le retour de la position sur les potentiomètres.

24 L'environnement social

Mon projet pourrait avoir un impact sur les expériences des gamers avec les jeux de type simulateur, car il offrira de meilleures sensations et une immersion accrue dans ces jeux.

Normalement, sur le marché, ces simulateurs sont chers si on les achète déjà tout faits, donc ce système est un bon compromis pour faire un premier pas dans ce monde de simulation, offrant un coût raisonnable.

Le seul problème de mon système est de retrouver des moteurs Wiper suffisamment coupleux pour supporter de lourdes charges, car ces 400 W se font rares, mais j'ai vu des compromis avec des moteurs de 100 W.

25 Environnement naturel

Grâce à ce système qui est géré avec un PID, la consommation du courant sera moindre que celle des vrais simulateurs sur le marché. Voici un petit exemple :

Dans l'industrie des vrais simulateurs, comme les simulateurs de vol et ceux de conduite, on retrouve souvent des systèmes plus complexes avec une consommation de courant plus importante. Par exemple, les simulateurs de vol utilisent des systèmes de mouvement

hydrauliques ou électriques puissants pour gérer des plateformes avec 6 degrés de liberté, donc 6 DOF. Ces systèmes consomment une forte quantité d'énergie pour reproduire des mouvements réalistes sur de telles structures, et ils sont aussi très coûteux à produire et à entretenir.

Mon simulateur dynamique qui lui est basé sur des Wiper moteur qui sont bon marché, est bien plus économe en énergie tout en offrant aussi une expérience immersive cependant avec un nombre de degrés de liberté plus petit donc 2 DOF. Et les simulateurs qui sont faits par des marques connue comme par exemple D-BOX ou Next Level Racing Motion Platforms utilisent eux des systèmes plus coûteux avec une plus grande consommation énergétique, souvent basés sur des moteurs brushless équiper de systèmes à retour haptique.

Donc, on peut en déduire que mon simulateur a certes une approche plus simplifiée, mais de ce fait, il est plus accessible.

26 Leadership et développement personnel

Grâce à ce projet, n'ayant que 5 semaines, j'ai pu apprendre à organiser au mieux mon temps à disposition, à fixer mes objectifs, puis à suivre un planning et le maintenir, et au final à m'adapter aux imprévus survenus.

Suivi du journal, PV et conclusion final

27 Journal de travail

Durant les 5 semaines qui m'ont été données afin de réaliser le travail de diplôme, j'ai fait un journal de travail dans lequel j'ai écrit les tâches que j'ai accomplies durant la journée, ainsi que le nombre de périodes effectuées.

Vous pouvez voir ce journal en annexe N° 3.

28 Procès-verbal

Pendant cette réalisation, chaque mardi, j'ai eu une séance avec Monsieur Gavin pour s'assurer et parler de l'avancement du projet, des difficultés rencontrées et pour définir les futures tâches à réaliser jusqu'à la prochaine séance.

Vous pouvez voir les 5 procès-verbaux rédigés qui se trouvent en annexe N° 4.

29 Conclusion

Durant ce travail de diplôme, j'ai pu concevoir un simulateur dynamique pour un siège gaming dont il est capable de reproduire les mouvements d'un jeu sur un axe de mouvement appelé Pitch.

L'objectif principal, qui était de créer un simulateur ayant 2 degrés de liberté (2 DOF) en me focalisant uniquement sur le Pitch, a été en partie réalisé, avec une partie mécanique fonctionnelle et une communication réussie entre Simtools et le PCB de mon simulateur dynamique.

Les tests menés montrent un fonctionnement stable de mon système, bien que des ajustements soient encore nécessaires afin d'améliorer la régulation PI pour obtenir une fluidité optimale des mouvements, donc sans oscillation.

Durant ce projet, plusieurs compétences m'ont été acquises en conception mécanique, électronique, et gestion de projet. L'aspect de l'accessibilité de mon simulateur qui est piloté

avec des moteurs d’essuie-glace (Wiper moteurs) et d’un contrôle PI, montre qu'il est possible de créer une alternative bien plus économique que les simulateurs disponibles sur le marché tout en offrant une assez bonne expérience qui est immersive pour les gamers.

Malgré quelques défis techniques et des ajustements à réaliser dans le futur, mon projet se rapproche des objectifs initiaux. Maintenant, j’ai une base solide pour une continuation future à titre personnel, notamment en augmentant les possibilités de mouvement en ajoutant le mouvement Roll et en optimisant les performances globales de mon simulateur.

Lausanne, le 23.09.2024 Jonathan Shifteh

30 Bibliographie (Datasheets et ChatGPT)

J'ai ajouté les liens vers les datasheets que j'ai utilisées lors de mon projet :

- Régulateur 5V
  untitled (microchip.com)

- Régulateur 3.3V
  173950336.pdf (we-online.com)

- Sabertooth 2x32
  Microsoft Word - Sabertooth 2x32 Manual.doc (dimensionengineering.com)

- FT230XS-R
  FT230X (ftdichip.com)

- TP2111-TR
  Datasheet_TP2111-TP2112.pdf (3peak.com)

- BSS138
  BSS138 - N-Channel Logic Level Enhancement Mode Field Effect Transistor (onsemi.com)

- LED LG R971
  LG-R971.pdf (ams-osram.com)

- LED LH R974
  LH-R974.pdf (ams-osram.com)

- Module du gyroscope
  - Microsoft Word - Document1 (digikey.com)
  - Pour la datasheet du chip, untitled (arduino.cc)

- Capteur de distance GP2Y0A21YK0F
  GP2Y0A21YK0F_spec.pdf (socle-tech.com)

- uC
  PIC32MX1XX/2XX 28/36/44-pin Family Data Sheet (microchip.com)

Je vous ai mis aussi le lien de ma conversation avec ChatGPT car ayant des difficultés en français, je me suis aidé de ChatGPT afin de corriger les fautes d'orthographe.

https://chatgpt.com/share/66f12fc6-439c-8008-92c3-85b31d2f8420

31 Annexe

Annexe 1 : Cahier des charges
(Voir PDF : 2411_CDC_diplome_SimulateurDynamique)

Annexe 2 : Planification
(Voir PDF : 2411_SDPSG-PLANNING_v1

Annexe 3 : Journal de travail
(Voir PDF : 2411_SDPSG-JOURNAL_v1)

Annexe 4 : Procès-Verbaux
(Voir PDF : 2411_SDPSG_PV_v1)

Annexe 5 : Schématique
(Voir PDF : 2411_SDPSG_Schematic_PCB_v1)

Annexe 6 : Plan d'assemblage
(Voir PDF : 2411_SDPSG_PCB Top-Bot Assy Dwgs_Variant_v1)

Annexe 7 : BOM
(Voir PDF : 2411_SDPSG_Bill of Materials-ES_v1)

Annexe 8 : Software
(Voir structogramme et code v1)

Annexe 9 : Mesure Timer1
(Voir image : 2411_SDPSG_Timer1_10ms)

Annexe 10 : Feuille de modification
(Voir PDF : 2411_SDPSG-MOD-v1)

Annexe 11 : Mode d'emploi du système
(Voir PDF : 2411_SDPSG-ModeEmploi-v1)

Annexe 12 : Construction partie mécanique
(Voir dossier mec : Construction métallique)

Annexe 13 : Tableau des différents DoF
(Voir image : DifferenceDOFAxes)