RAPPORT DE PROJET

Rapport n°2

Jeudi 30 Avril 2020

CHAN Jayvin, GOOR Aurélien GE4

PROJET GE 2

AÉROGLISSEUR RADIOCOMMANDÉ

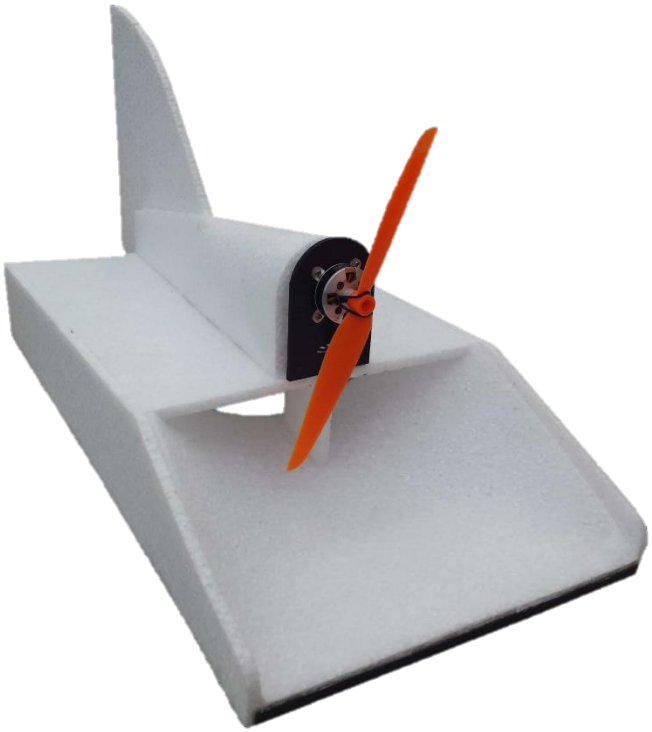


Table des matières

I. Analyse fonctionnelle

Dans le cadre de ce projet, nous allons réaliser un aéroglisseur. Nous n'utiliserons pas le système de coussin d'air, mais nous exploiterons la portance due à la circulation de l'air.

Pour cela nous utiliserons un moteur brushless doté d'une hélice qui servira à générer le flux d'air. Ce flux d'air sera utilisé d'une part pour faire flotter l'aéroglisseur, et d'autre part pour le mettre en mouvement. L'engin sera dirigé par une gouverne de direction actionné par une servomoteur. Le tout sera alimenté par une batterie Li-Po (3S)

L'aéroglisseur devra être piloté par Bluetooth depuis un smartphone, il disposera de 5 niveaux de vitesse et de 5 niveaux pour la direction.

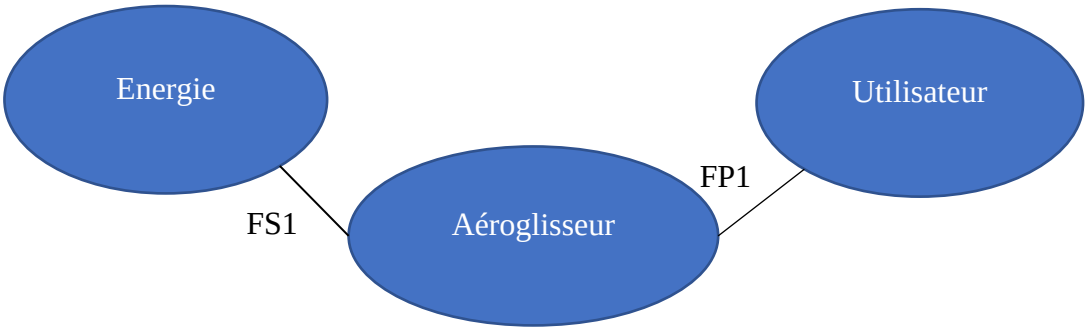


Figure 1 : Diagramme Bête à corne

FP1 : Contrôler les déplacements de l'aéroglisseur
FS1 : Alimenter l'aéroglisseur

Diagramme FAST

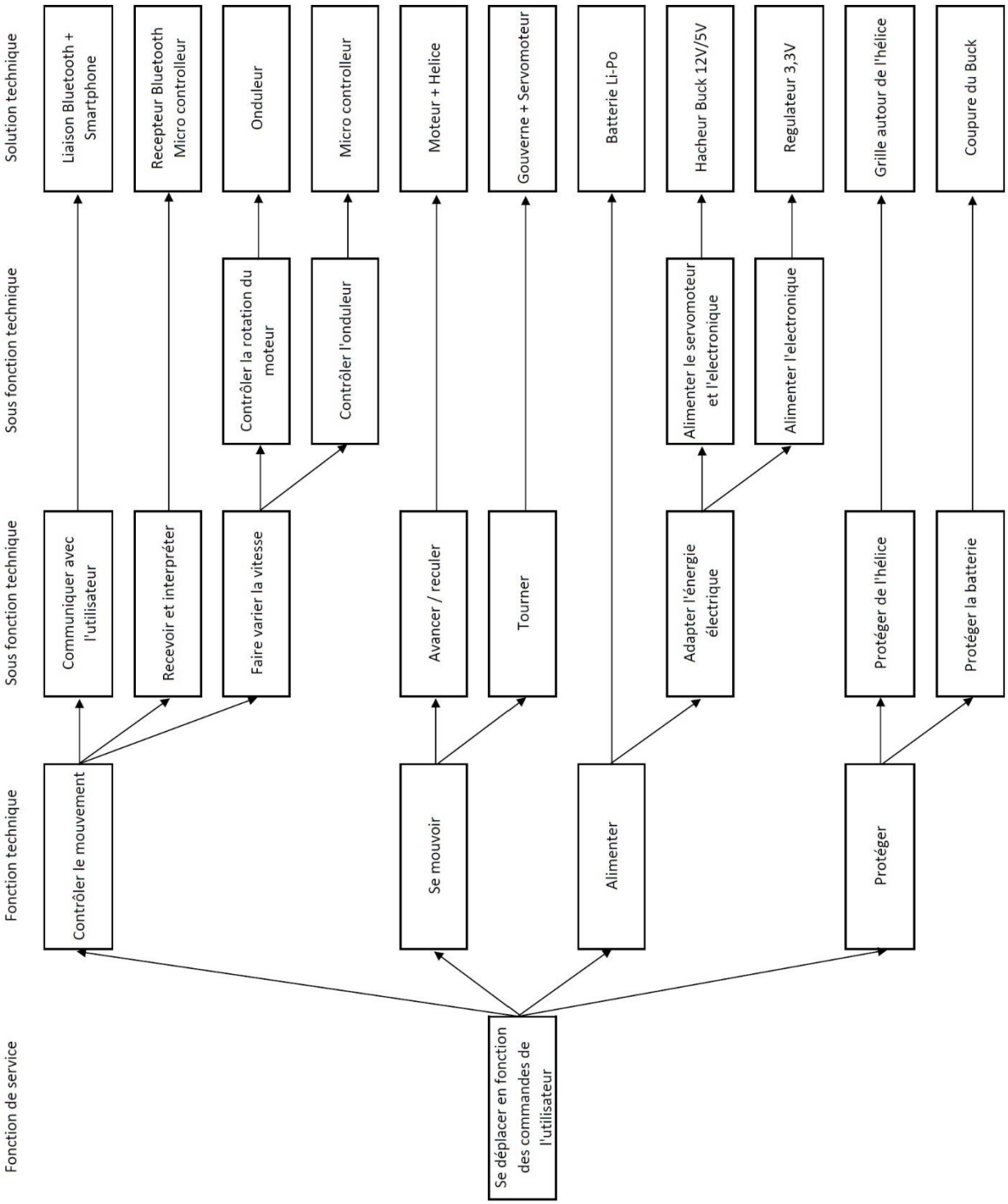


Figure 2 : Diagramme FAST

II. Cahier des charges différentes parties

1. Pilotage du moteur brushless

Dans un premier temps il faut déterminer les caractéristiques du moteur brushless (que nous verrons dans la pré-étude). Pour le faire tourner de façon optimale, il faudrait avoir accès à la position du rotor, cependant il ne dispose pas de capteurs. Nous exploiterons donc les tensions des trois phases pour remonter à V_A la tension simple. Le passage de cette tension par 0 nous donnera une information sur la position du rotor, ainsi nous alimenterons complémentairement les autres phases B et C.

2. Onduleur triphasé

Nous disposons d'une batterie de 11.1V, pour alimenter le moteur triphasé, nous réaliserons un onduleur à bras de pont qui prendra comme source continue la batterie et qui alimentera les trois phases du moteur. Les interrupteurs commandés seront des MOSFET canal N, pilotés par des drivers de demi pont eux même contrôlé par un microcontrôleur dsPIC30F2010 qui est spécialement conçu pour faire du pilotage d'un moteur brushless.

3. Convertisseur DC 5V

Le convertisseur DC 5V sert à adapter la tension délivrer par la batterie pour alimenter le servomoteur, le convertisseur 3.3V ainsi qu'une partie de l'électronique.

Le convertisseur qui sera réaliser par un hacheur devra répondre au cahier des charges suivant :

- Composant imposé : LM22672 de Texas Instrument
- Tension nominale d'entrée : 11.1V
- Tension minimale d'entrée : 10V (Pour éviter que la batterie ne passe en décharge profonde)
- Courant nominal de sortie : 1A
- Oscillation de courant en sortie : $< 30\%$ du courant nominal de sortie
- Tension nominale de sortie : 5V
- Oscillation maximale de la tension de sortie : 5% de la tension nominale de sortie

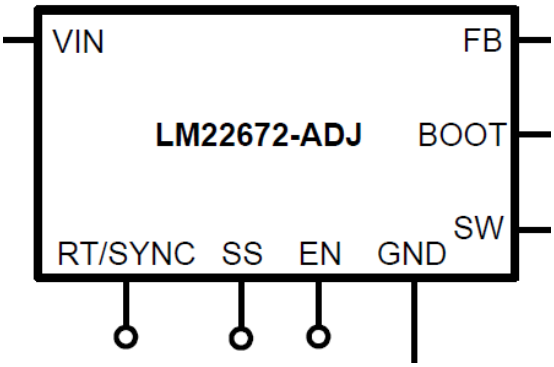


Figure 3 : Schéma du LM22672

Le pin SS (Soft start) de démarrage en douceur permet de contrôler la constante de temps de montée de la tension.

Le pin SYNC permet de fonctionner à une fréquence autre que celle de l'oscillateur interne.

La broche EN permet d'activer ou non la conversion suivant la tension à cette borne, on pourra l'exploiter pour protéger la batterie contre la décharge profonde.

La broche FB est le retour de l'asservissement en tension afin d'obtenir en sortie une tension stable.

4. Convertisseur DC 3.3V

Ce convertisseur permettra à partir des 5V en entrée d'obtenir 3.3V en sortie pour le DsPIC et la carte de commande générale. Il s'agira d'un régulateur de tension linéaire (LDO), nous utilisons le régulateur de tension MCP1826S de chez Microchip.

Il devra respecter le cahier des charges suivants :

- Composant imposé : MCP1826S de Microchip
- Tension nominale d'entrée : 5V
- Tension nominale de sortie : 3.3V
- De plus la tension de sortie doit être la plus stable et « propre » possible car le module Bluetooth est très sensible au bruit et au surtension
- Oscillation maximale de la tension de sortie : 5% de la tension nominale de sortie

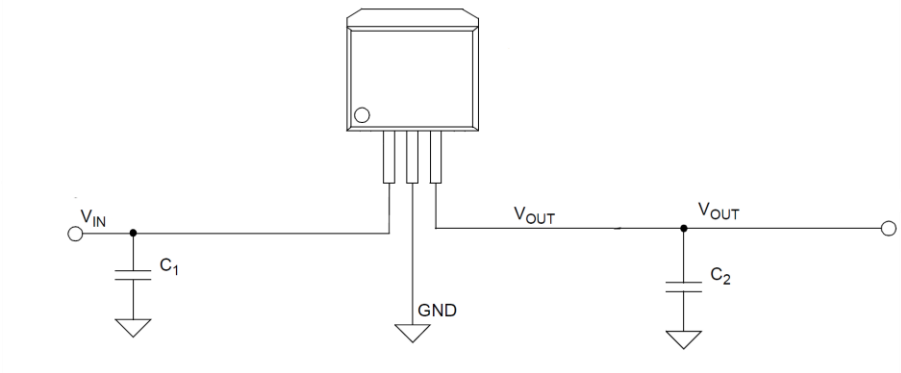


Figure 4 : Schéma de câblage du régulateur 3.3V

Etant donné que l'alimentation des microcontrôleurs et du module Bluetooth est sensible aux variations nous utiliserons un régulateur où nous n'avons plus qu'à câbler les condensateurs d'entrée/sortie.

5. Carte de commande générale

La carte de commande générale est l'organe qui va coordonner toutes les actions. On retrouvera sur cette carte le microcontrôleur PIC16F1619. Elle devra gérer :

- La communication en Bluetooth avec le smartphone de l'utilisateur
- Le servomoteur servant à diriger l'aéroglisseur
- Le microcontrôleur qui gère les transistors de l'onduleur en lui transmettant via une liaison qui sera vraisemblablement en SPI la fréquence électrique du moteur.

Pour communiquer en Bluetooth nous utiliserons un module HC05 ZS-040 qui utilise la norme Bluetooth Low Energy.

Il faudra mettre en place un interrupteur qui coupera l'alimentation de l'électronique et du moteur. Un autre système de sécurité permettra d'arrêter le moteur si la savonnet ne reçoit pas d'information de la télécommande pendant plus de 200ms. Enfin un système de coupure générale facile d'accès permettra de couper l'alimentation globale.

6. Télécommande et interface de communication

Le contrôle de l'aéroglisseur se fera via une application sur smartphone connecté en Bluetooth à la carte de commande générale. L'interface comportera au minimum d'un commande de direction (droite/gauche) proportionnel à 5 niveaux, d'une commande de propulsion également proportionnel à 5 niveaux et d'un retour de la tension de la batterie. On utilisera vraisemblablement l'application Bluetooth Electronics de keuwlsoft disponible sous Android©.

7. Servomoteur

Le servomoteur servira à actionner une gouverne afin de diriger le flux d'air, et donc l'aéroglisseur. Pour cela on utilisera un servomoteur standard.

Pour le commander on utilisera un signal en PWM, dont la durée entre la fin d'une impulsion et le début de la suivante n'excède pas 20ms.

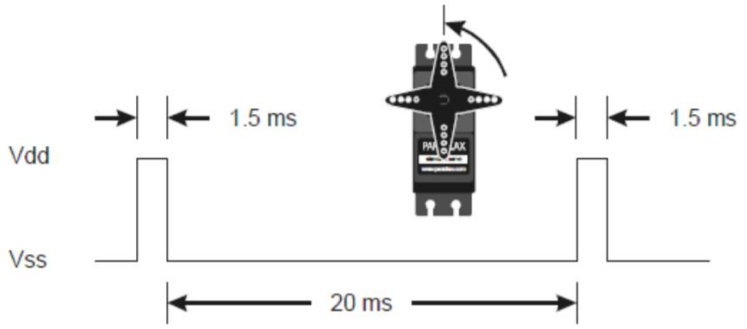


Figure 5 : Diagramme du signal de commande d'un servo moteur

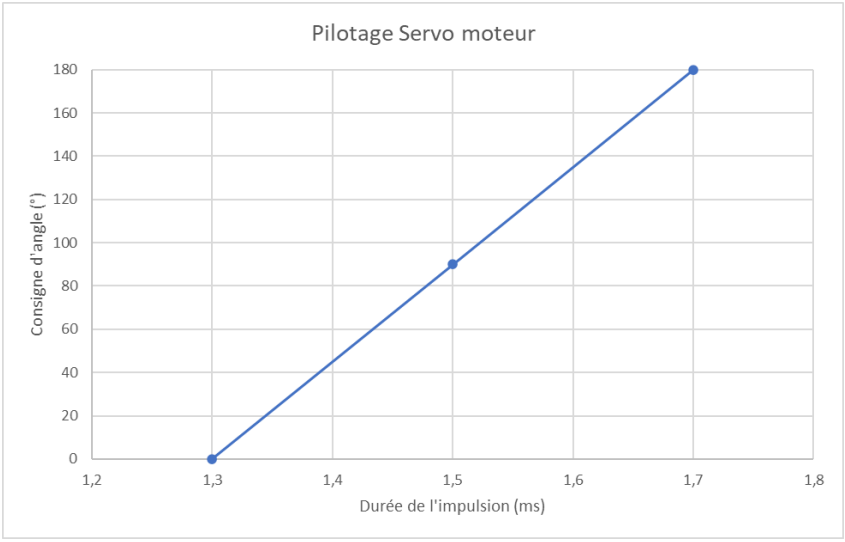
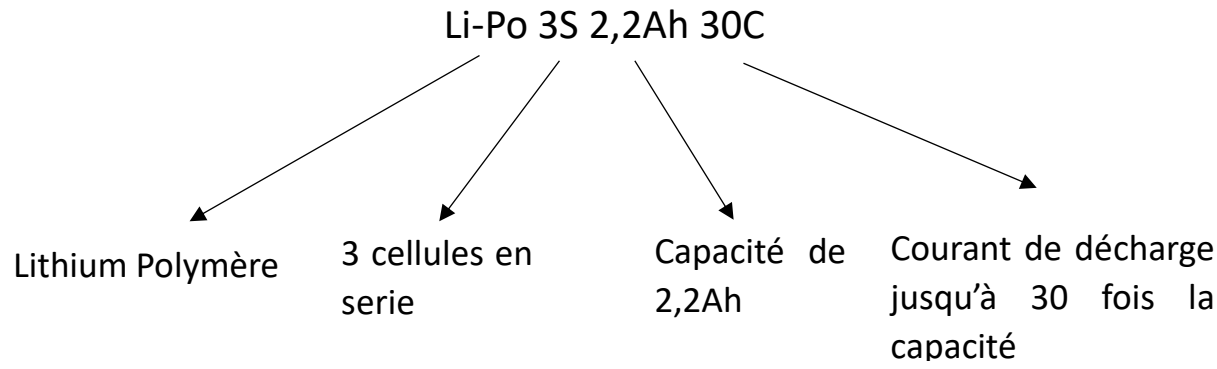


Figure 6 : Position du servomoteur en fonction de la durée d'impulsion

8. Batterie Li Po 3s

Nous déterminerons les caractéristiques de la batterie à l'aide de la datasheet afin de la modéliser dans PSIM.

Nous mettrons en œuvre un système permettant de couper l'alimentation lorsque la tension de la batterie passera en dessous d'un certain seuil afin de préserver la batterie.



9. Structure mécanique de l'aéroglisseur

Nous nous inspirons du patron déjà fourni par un constructeur, mais nous adapterons les dimensions vis-à-vis de la taille de l'hélice. Il faudra de même réaliser une protection des personnes contre l'hélice tout en laissant passer l'air, en réalisant par exemple une grille.

III. Pré-étude

1. Détermination des paramètres du moteur brushless triphasé

1.1. Paramètres électriques

Nous commençons par déterminer les éléments caractéristiques du schéma équivalent du modèle de Behn-Eschenburg.

Nous disposons d'un banc d'essai composé de deux moteurs du même type, l'un dispose d'une hélice.

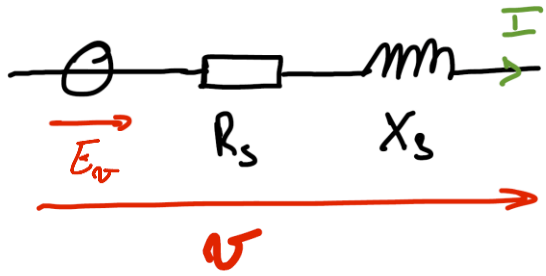


Figure 7 : Schéma équivalent de la machine synchrone

Résistance statorique

On détermine la résistance série statorique par la méthode volt-ampèremétrique, étant donné que nous n'avons pas accès au point neutre, on réalise les mesures entre deux phases, d'où :

$$R_s = \frac{U_{ph-ph}}{2 * I}$$

Mesures DC										
U (mV)	55,9	78,4	100,3	111,1	122,8	146,1	167,3	223,5	334,8	
I (A)	0,5	0,7	0,9	1	1,1	1,31	1,5	2	3	Moy
2R(mΩ)	111,8	112,0	111,4	111,1	111,6	111,5	111,5	111,8	111,6	111,6

Ainsi $\boxed{R_S = 55{,}8m\Omega}$

Essai à vide

Lors de l'essai à vide, on fait tourner l'un des moteurs, on relève la tension entre deux phases du générateur que l'on nomme E_{vcomp} pour plusieurs vitesses de rotation. Cela en prévision de l'essai en court-circuit.

Etant donné que le moteur est couplé en étoile $\boxed{E_V = \frac{E_{Vcomp}}{\sqrt{3}}}$

Essai à vide															
N	348	600	900	1430	1985	2690	2845	3230	4080	4610	5310	5890	6940	7555	
Ev comp	0,257	0,436	0,655	1,026	1,6	1,9	2,1	2,4	2,93	3,5	3,9	4,3	4,9	5,5	Moy
Ev/N	4,26E-04	4,20E-04	4,20E-04	4,14E-04	4,65E-04	4,08E-04	4,26E-04	4,29E-04	4,15E-04	4,38E-04	4,24E-04	#####	4,08E-04	4,20E-04	4,24E-04

Essai en court-circuit

Le système est assimilable à un transformateur, on court-circuite les phases au niveau de l'alternateur. On augmente progressivement le potentiomètre de commande de la vitesse du moteur. On se place à une vitesse N dont on connaît la fem E_{Vcomp}, sans que le courant de court-circuit ne dépasse le courant nominal, on mesure le courant de court-circuit et sa fréquence pour en déterminer L_{CS} l'inductance cyclique, ces mesures seront réalisées à l'aide d'une sonde de courant.

$$Z = \frac{E_V}{I_{cc}} = \sqrt{R_s^2 + (L_{cs} * \omega)^2}$$

$$Z = \frac{0.235}{3.9} = 0.0603$$

$$L_{CS} = \frac{\sqrt{Z^2 - R_S^2}}{2 * \pi * f} = \frac{\sqrt{0.0603^2 - 0.0558^2}}{2 * \pi * 46{,}2}$$

$$\boxed{L_{CS} = 78\mu H}$$

On obtient donc le modèle de Behn-Eschenburg suivant pour une phase du stator :

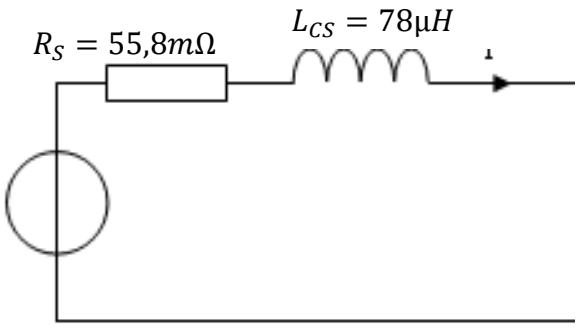


Figure 8 : Modèle de Behn-Eschenburg

1.2. Paramètres mécaniques

On souhaite ensuite déterminer les paramètres mécaniques. Pour cela on effectue un essai de lâcher, qui consiste à faire tourner le moteur à une certaine vitesse, puis on coupe l'alimentation et on relève la décroissance de la vitesse. Pour cela on peut observer les tensions aux bornes du même moteur entraîné et ainsi déterminer la constante de temps mécanique, et ensuite le moment d'inertie J en faisant attention car dans nos mesures nous retrouverons **deux fois J** étant donné que le moteur et l'alternateur du même type partagent le même arbre.

L'équation régissant le fonctionnement mécanique du moteur est :

$$J\frac{d\Omega}{dt} = -C_s - K_v\Omega$$

$$\frac{d\Omega}{dt} + \frac{K_v}{J}\Omega = -\frac{C_s}{J}$$

La courbe de vitesse de rotation obtenue lors de l'essai devrait donc se superposer avec la modélisation solution de l'équation différentiel ci-dessus.

$$\Omega(\mathrm{t}) = \frac{C_s}{K_v}e^{-t\cdot\frac{K_v}{J}}$$

Où J est le moment d'inertie, Kv le coefficient de frottement visqueux et Cs le couple de frottement sec.

A l'aide de l'essai lâché à vide on a pu déterminer $\boxed{\tau = 720ms}$, ω_0 la vitesse initiale était de 4150 tr/min.
On a alors essayé d'en tracer la courbe représentative. L'écart observé vient probablement des incertitudes de mesure, mais surtout du fait que l'hélice soit déjà présente sur le moteur alors que notre équation différentielle de vitesse ne la prend pas en compte.

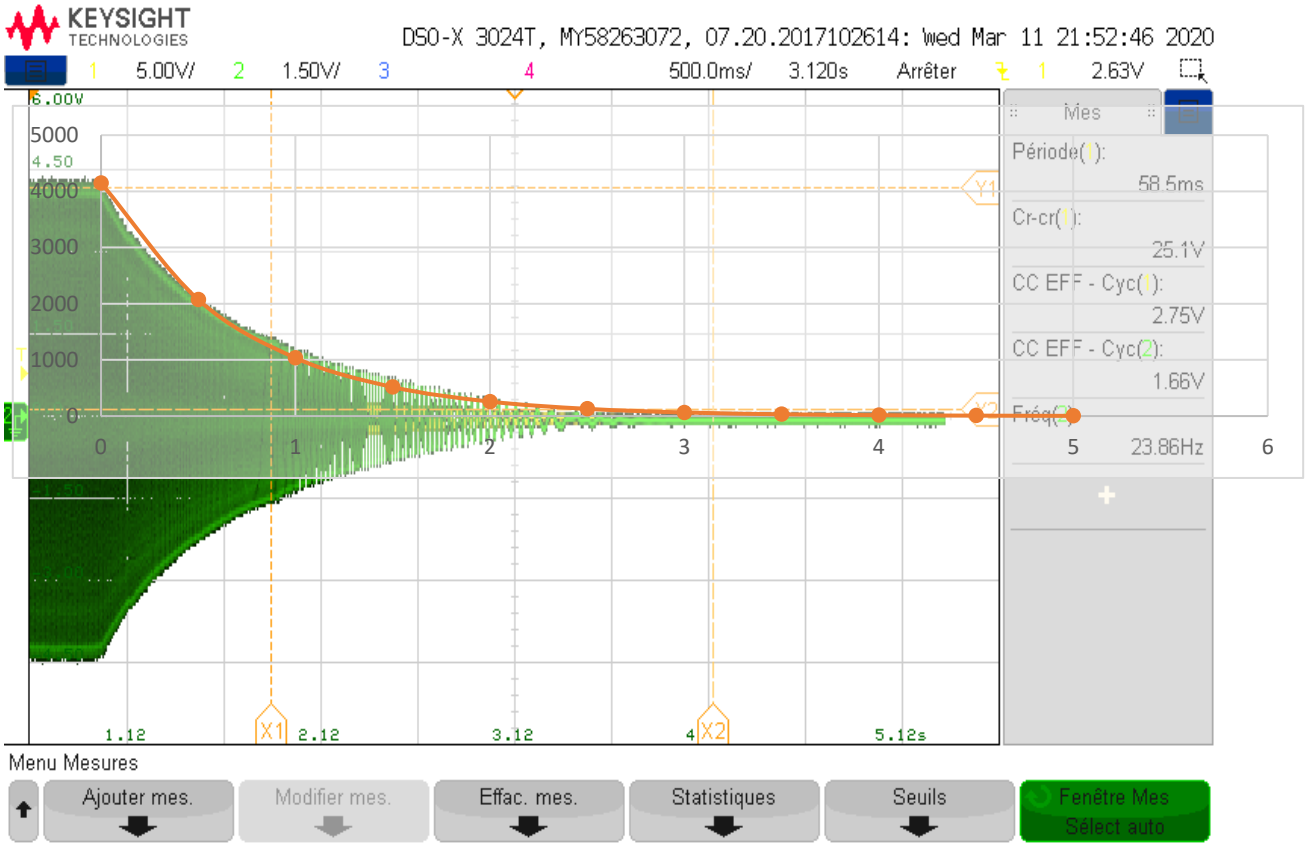


Figure 9 : Courbe théorique de la vitesse lors de l'essai lâché à vide superposée à celle des mesures

Pour déterminer les paramètres manquants, nous réaliserons un essai lâché en charge pour obtenir une autre équation.

Nous nous sommes rendus compte que nous avons fait l'erreur de réaliser ces essais sur un banc de moteurs contenant l'hélice, cela créée un certain écart vis-à-vis des vrais paramètres du moteur. Néanmoins la méthode est correcte, il suffira de les appliquer à nouveau sur le banc en pensant à enlever l'hélice.

2. Détermination des paramètres de la batterie

D'après la datasheet de la batterie choisie les caractéristiques sont les suivantes :

Batterie Li-Po 3S 2,2Ah 30C

- $V_{Nom} = 3 \cdot 3.7 = 11.1V$
- $V_{Max} = 3 \cdot 4.2 = 12.6V$
- $I_{Max} = 30 \cdot 2.2 = 66A$
- Il est recommandé de ne pas décharger la batterie en-dessous de 3V par cellule soit 9V au total, mais il nous est recommandé de nous limiter à 10V.

3. Alimentation

3.1. Convertisseur 5V

Compte tenu du cahier des charges et en se référant à la datasheet, voici comment le LM22672 sera branché.

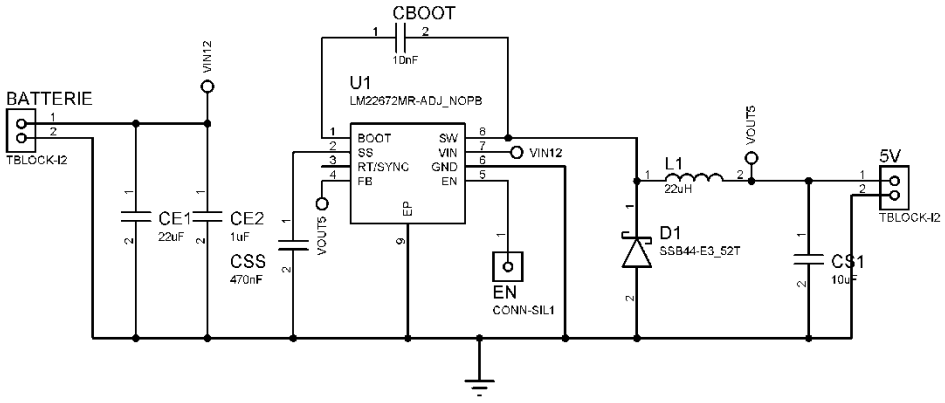


Figure 10 - Schéma de câblage du convertisseur 5V

Condensateurs d'entrée

Les condensateurs **CE1** et **CE2** sont des condensateurs d'entrée visant à stabiliser et enlever le bruit que l'on peut avoir sur l'alimentation. CE2 joue le rôle de découplage, et CE1 assure de fournir la charge nécessaire lors d'un appel de courant du composant.

D'après le cours d'électronique de puissance sur le hacheur série, on a :

$$Ce = \frac{Ie.(1-\alpha)}{\Delta Ve.F} = \frac{Vs.\alpha.(1-\alpha)}{R.\Delta Ve.F}$$

Ainsi avec $Vs = 5V, R = 5\Omega,\ \Delta\mathrm{Ve} = 30\mathrm{mV}, \mathrm{F} = 500\mathrm{kHz}, \alpha = 0.5$ lorsque la tension de la batterie atteint la limite de 10V, on obtient

$$Ce = 16{,}7\mu F$$

N'oublions pas les imperfections de la batterie et es fils d'alimentation : $R_{bat} + R_{fils}, L_{fils}$ qui nous amènent à prendre une valeur de Ce plus élevée.

On choisit alors **Ce = 22µF**

La fréquence de découpage est de 500kHz, la datasheet nous suggère de mettre en parallèle un condensateur céramique de $1\mu F$. Lors du choix des composants nous choisissons donc des types X7R car la valeur de capacité reste plutôt stable, avec une tenue en tension de 25V et avec la résistance série équivalente (ESR) la plus faible.

Cboot

Le condensateur Cboot est un condensateur de « boot-strap » qui permet de fournir à la grille du MOSFET le courant de grille nécessaire. La datasheet nous conseille d'utiliser un condensateur céramique de $\boldsymbol{10nF}$.

Css

La broche SS permet un démarrage en douceur (Soft Start) du régulateur, le temps de montée se calcule avec la formule suivante :

$$T_{ss} \approx 26 \cdot 10^3 \cdot C_{ss}$$

Ainsi un condensateur $\boldsymbol{Css} = 470nF$ nous parait adapté, donnant alors $T_{ss} \approx 12ms$.
Nous utilisons la fréquence interne de $F = 500kHz$, on ne connecte donc pas l'entrée RT/SYNC.

FB

Le composant choisi LM22672MR-5.0/NOPB est fait pour délivrer 5V en sortie, ainsi on reboucle directement la sortie sur l'entrée Feedback.

Condensateur de sortie

En prenant $V_e = 12V,\ V_s = 5V,\ I_s = 1A$
Pour un hacheur série on sait que :

$$V_s = \alpha V_e$$

D'où $\alpha_{nom} = \frac{V_{Snom}}{V_e} = 0.42$

On dimensionne L pour que les variations de I_L ne dépassent pas 30% de l'intensité de sortie

$$\Delta I_L = I_L(\alpha T) - I_L(0)$$

$$= \frac{V_e - V_s}{L}\alpha T = \frac{\alpha(1-\alpha)V_e}{LF}$$

$$L = \frac{\alpha(1-\alpha)V_e}{F\,\Delta I_L}$$

Soit une inductance de $19.4\mu H$, on prend alors la valeur normalisée de $\boldsymbol{22\mu H}$.

Nous souhaitons limiter l'ondulation de la tension de sortie à 25mV. Pour cela on fait l'hypothèse que la variation de Vs est petite devant Vs et que $i_S = i_c + I_s$

$$\Delta Q = C_S \cdot \Delta V_s$$

Or $\Delta Q = \frac{1}{2}\frac{\Delta I_L}{2}\frac{T}{2} = \frac{1}{8F}\frac{\alpha(1-\alpha)V_e}{LF}$

D'où $C_S = \frac{\alpha(1-\alpha)V_e}{8LF^2\Delta V_s}$

Considérons la valeur maximale pour Ve=12V, on a alors $\alpha = 0.42$. De plus L=22μH, F=500kHz, et avec $\Delta V_S = 10mV$, le calcul donne

$$C_S = 6.6\mu F$$

Pour prévoir une marge, on choisit la valeur de **Cs=10μF** c condensateur céramique

3.2. Régulateur linéaire 3.3V

Le convertisseur 5V → 3.3V est plus simple au niveau de la structure, il suffit d'y câbler les condensateurs d'entrée et de sortie, respectivement de 4.7μF et 2.2μF

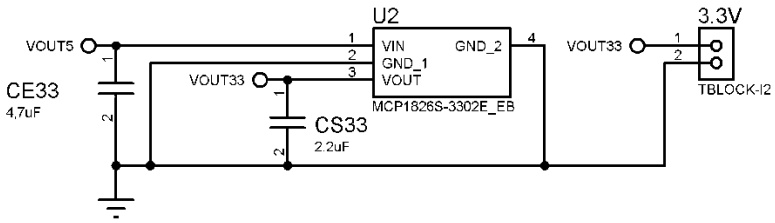


Figure 11 - Câblage du convertisseur 3.3V

3.3. Simulation

Nous simulons le hacheur série sur PSIM en utilisant des composants de niveau 2 afin de renseigner les valeurs de résistance série équivante ESR pour les condensateurs afin de simuler les imperfections. Pour cela en ayant choisi des condensateurs chez le fabricant KEMET, un outil de simulation (K-SIM) permet d'estimer la valeur d'ESR du condensateur choisi, en se plaçant à la fréquence de travail.

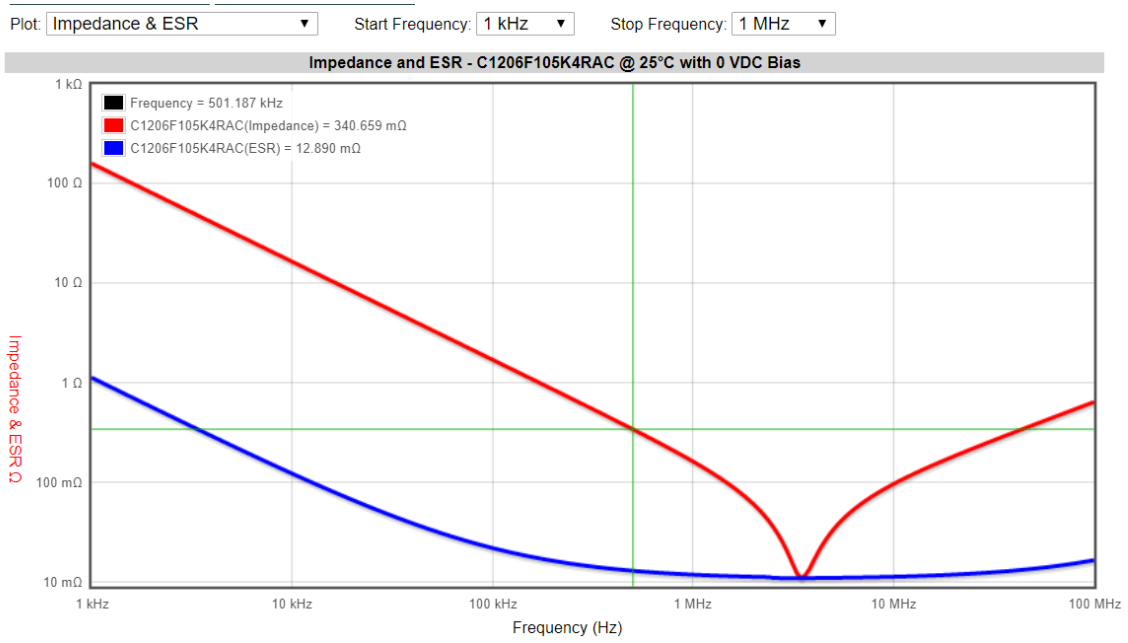


Figure 12 Détermination de la valeur de ESR sur l'outil K-SIM

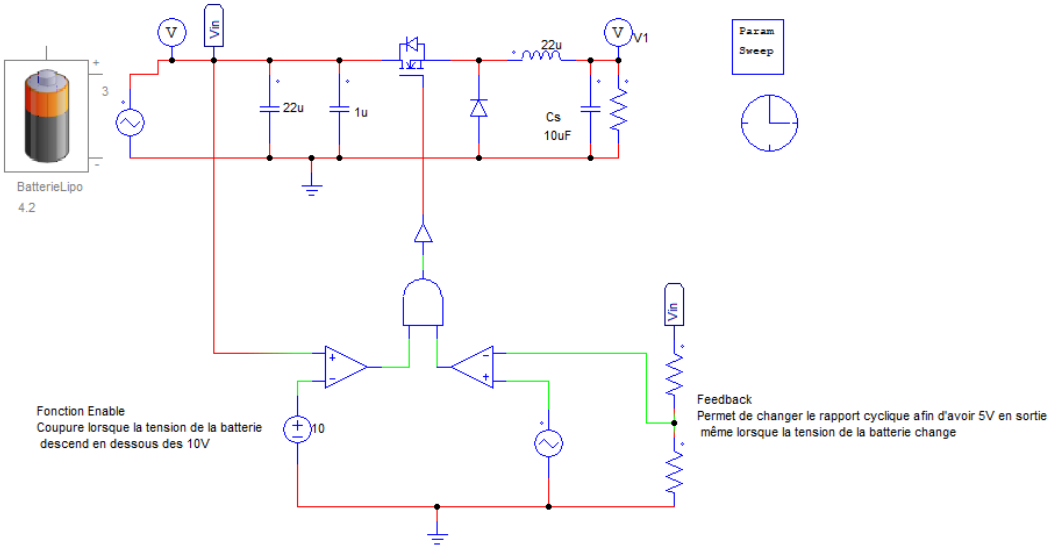


Figure 13 : Modélisation du convertisseur 5V et de certaines de ses fontions sous PSIM

Les fonctions Enable et Feedback détaillées sur le schéma ont aussi été réalisées. En utilisant un triangle allant de 9V à 12V en entrée, on obtient le résultat ci-dessous :

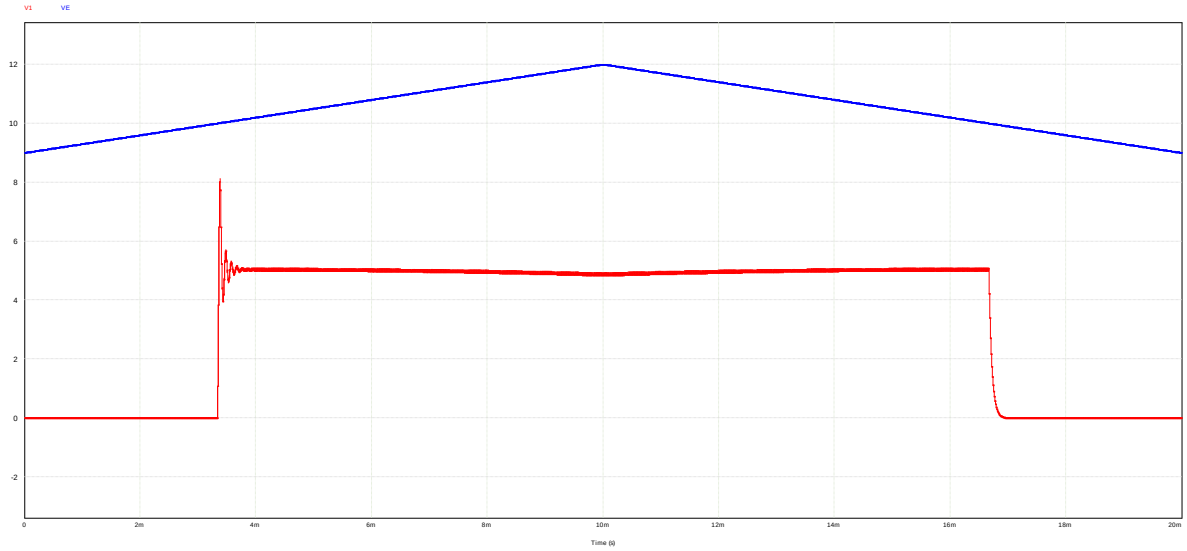


Figure 14 : Résultat de la simulation

La courbe bleue correspond au signal d'entrée, et la rouge au signal de sortie. Le pic correspond à la fonction feedback modélisée, ce ne sera pas le cas en réalité. Ce qui nous intéresse c'est l'ondulation de la tension de sortie qui reste bien inférieure à 10mV, cela valide le choix des composants.

3.4. Circuit imprimé

Nous avons décidé de rassembler sur une même carte d'alimentation les convertisseurs 5V et 3.3V. Ces derniers disposent de zones de pad thermique connectés au GND, c'est pourquoi il est judicieux de générer un plan de masse pour évacuer la chaleur générée. L'objectif est de rendre la carte la plus compacte, tout en essayant d'espacer les convertisseurs. Les composants externes doivent être placés au plus proche des convertisseurs.

Il est important de placer des LEDs au niveau de chaque bornier, cela permettra d'indiquer le bon fonctionnement de la carte, et de trouver plus facilement le souci en cas de problème.

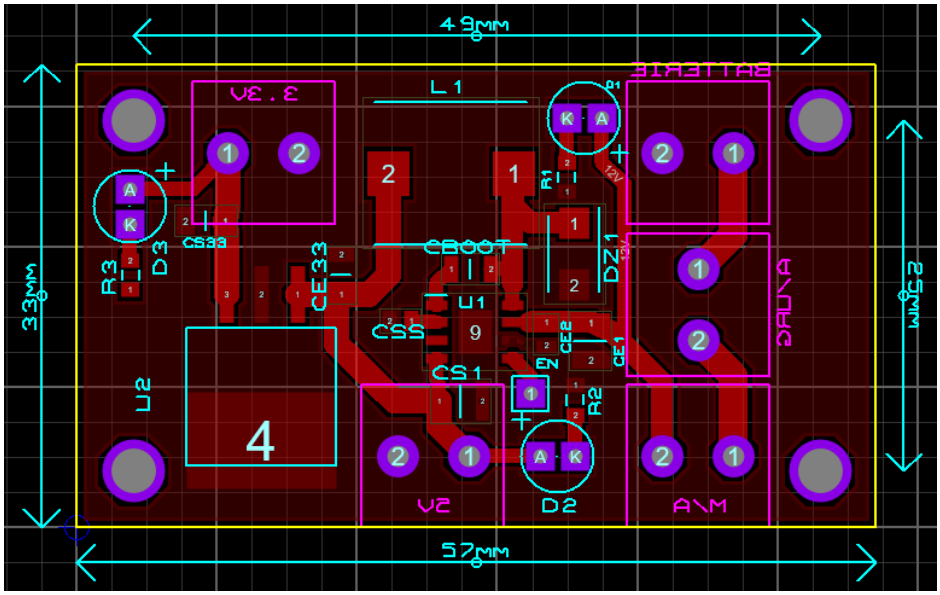


Figure 15 - PCB alimentation

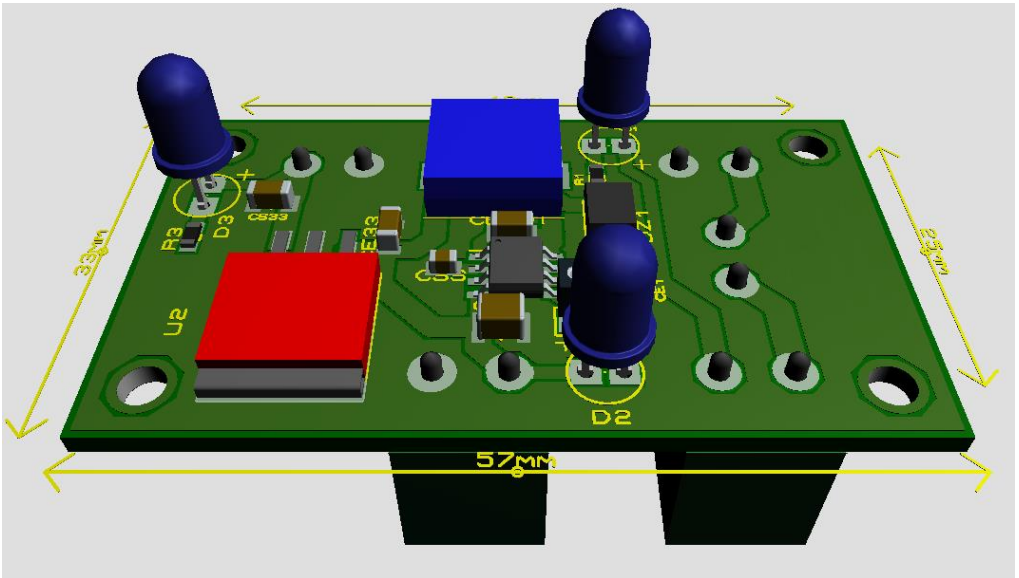
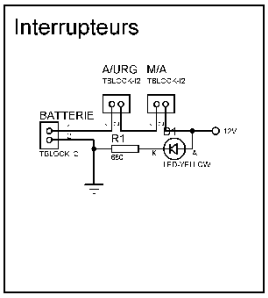


Figure 16 - Aperçu de la carte d'alimentation

Les connecteurs permettront l'arrivée de la tension issue de la batterie et l'accessibilité aux tensions converties.



Deux autres connecteurs ont été ajoutés, ils permettront de brancher les interrupteurs de Marche/Arrêt et d'arrêt d'urgence comme demandé dans le cahier des charges.

4. Onduleur

4.1. Pilotage moteur

Sur le schéma ci-dessous issu du document de Microchip, on dispose d'une représentation simplifiée d'un moteur brushless triphasé inner runner (le rotor à l'intérieur et le stator à l'extérieur). Notre moteur sera un outer runner donc on inverse la position du rotor et du stator mais le fonctionnement reste le même.

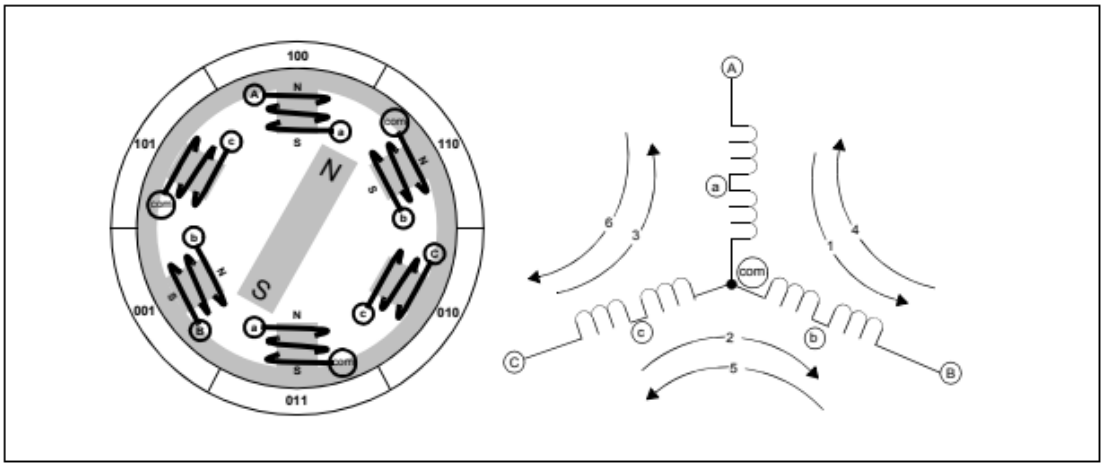


Figure 17 : Représentation simplifiée d'un moteur brushless triphasé

En alimentant à tour de rôle chaque bobine du stator, on peut voir dans le schéma suivant comment mettre le rotor en mouvement.

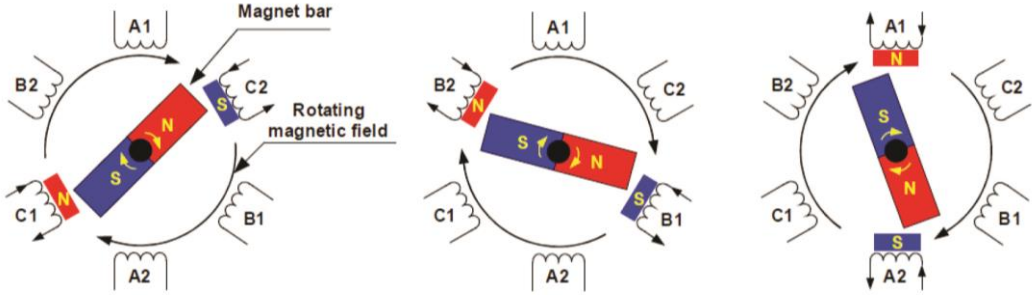


Figure 18 : Etapes de fonctionnement du moteur brushless

Source : EE Publishers – Brushless DC motors gain in popularity

Il est plus optimal d'alimenter **deux phases sur les trois** pour attirer et repousser l'aimant dans le même sens afin de générer une bonne valeur de couple.

Ainsi en alimentant les phases selon une certaine séquence on arrive à faire tourner le moteur, mais jusqu'à une certaine vitesse, car au-delà il n'y a plus synchronisme entre rotor et stator, le moteur décroche. Il est possible de maintenir ce synchronisme en connaissant la position angulaire du rotor, ce qui n'est pas donné directement car notre moteur ne dispose pas de capteurs de courant.

Une autre possibilité consiste à exploiter force contre-électromotrice que nous détaillerons par la suite.

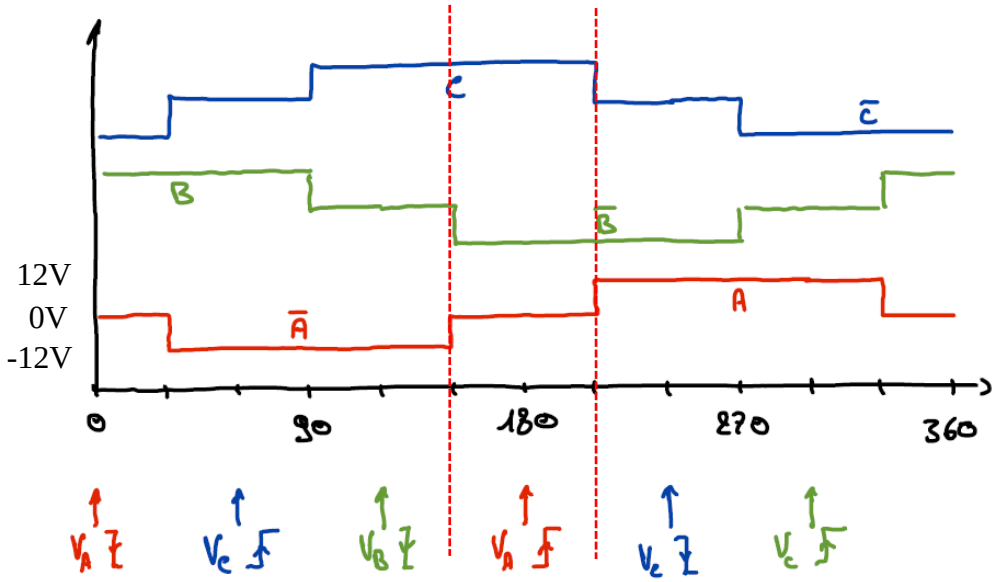


Figure 19 : exemple de séquence d'alimentation des phases du moteur

Pour alimenter les 3 phases du moteur au bon moment et donc faire tourner le moteur de façon nominale jusqu'à de très grandes vitesses, il faut réaliser un onduleur qui convertit la tension continue de la batterie en un tension triphasée alternative.

4.2. Angle d'avance/ largeur d'impulsion

Outre les paramètres intrinsèques au moteur, l'angle d'avance et la largeur du créneau de courant sont deux paramètres qui vont beaucoup modifier le fonctionnement du moteur en simulation et en réalité.

Pour avoir une bonne valeur de couple moyen, il faut que le courant soit déjà établi lorsque le flux est positif, donc que la tension varie de manière positive. L'angle d'avance permet d'indiquer de combien de degrés en avance faut-il commencer à faire croître le courant. Il sera entre 0 et 30°. La largeur d'impulsion est par défaut à 120°, mais en réalité il est intéressant de le faire varier entre 100° et 140°.

Nous cherchons à optimiser le fonctionnement du moteur afin d'avoir le meilleur rendement avec une vitesse de rotation proche de celle voulue, pour cela il faut trouver un compromis entre la valeur des deux paramètres. En effet, avec trop d'avance, on risque encore d'avoir un $\frac{dv}{dt} < 0$ lors de l'établissement du courant, ce qui fait baisser le couple. Si le créneau est trop large, il y a encore du courant dans la phase lors du passage d'un $\frac{dv}{dt} > 0$ à un $\frac{dv}{dt} < 0$, ce qui génère un couple négatif.

A l'aide de l'outil « parameter sweep » de PSIM, on simule les différents couples

(angle d'avance ; largeur d'impulsion).

Et on relève les résultats ci-dessous en régime permanent :

Av	pw	Ve VF	Ie VF	Pélec calculée (W)	Couple VF (Nm)	Vitesse (rad.s) VF	Vitesse (tr/min) VF	Pmeca (W)	Pmeca/Pelec
0	100	11,5	15,7	180	1,58E-01	7,46E+02	7129	118	65%
0	110	11,2	21,7	243	1,91E-01	8,28E+02	7907	158	65%
0	120	11,1	28,5	317	2,26E-01	9,04E+02	8639	204	64%
0	130	11,0	28,2	310	2,18E-01	8,87E+02	8468	193	62%
0	140	11,0	26,3	288	1,99E-01	8,46E+02	8084	169	59%
10	100	11,5	15,8	181	1,62E-01	7,58E+02	7237	123	68%
10	110	11,2	22,1	247	1,98E-01	8,43E+02	8048	167	68%
10	120	11,1	29,2	324	2,36E-01	9,25E+02	8837	218	67%
10	130	11,0	31,3	344	2,43E-01	9,41E+02	8987	229	67%
10	140	11,1	29,2	324	2,30E-01	9,13E+02	8716	210	65%
20	100	12,5	0,0	0	7,17E-09	8,48E-04	0	0	0%
20	110	11,1	22,8	254	2,03E-01	8,55E+02	8166	174	68%
20	120	11,0	30,4	335	2,43E-01	9,40E+02	8979	229	68%
20	130	10,9	33,1	361	2,55E-01	9,64E+02	9205	246	68%
20	140	10,9	32,4	355	2,51E-01	9,56E+02	9133	240	68%
30	100	12,5	0,0	0	7,17E-09	8,48E-04	0	0	0%
30	110	12,5	0,0	0	7,17E-09	8,48E-04	0	0	0%
30	120	11,0	31,9	350	2,48E-01	9,50E+02	9075	236	67%
30	130	10,8	35,0	379	2,62E-01	9,79E+02	9347	257	68%
30	140	10,8	35,7	385	2,66E-01	9,86E+02	9416	262	68%

Le couple ($\theta_{avance} = 30°$, $largeur_{créneau} = 140°$) est le plus optimal, nous garderons ces paramètres pour la suite.

4.3. Bras de pont

4.3.1. Structure

L'onduleur est constitué de 3 bras de pont, les interrupteurs commandés seront des N-MOSFETs Optimos de chez Infineon, pilotés par un dsPIC par l'intermédiaire de drivers. Chaque phase du moteur brushless sera connectée à une carte de bras de pont.

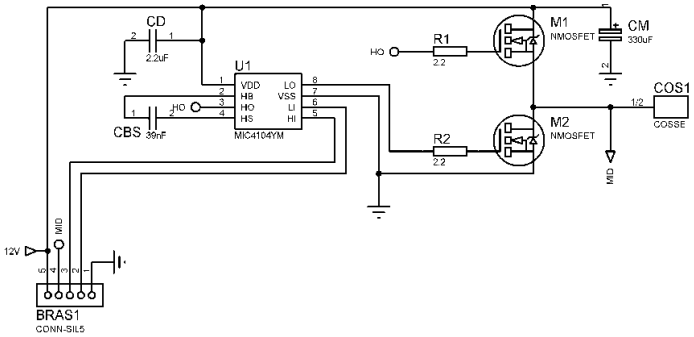


Figure 20 Schéma électrique d'un bras de pont

La valeur du condensateur de bootstrap CBS est déterminée à partir de la formule :

$$C_B \geq \frac{Q_{Gate}}{\Delta V_{HB}}$$

Où Q_{Gate} est la charge de grille nécessaire au Mosfet utilisé pour conduire. $Q_{Gmax} = 35nC$ pour nos N-Mosfets. ΔV_{HB} correspond à la chute de tension aux bornes de HB, nous la fixons à 1V. On obtient donc une valeur de 35nF pour C_B, et on choisit la valeur normalisée de **39nF**.

La résistance de grille doit être assez faible afin de limiter les temps de commutations, sans être trop faible en termes de limite de courant, c'est pourquoi la valeur de 2,2Ω a été choisie.

La phase du moteur sera connectée au milieu du bras de pont à travers une cosse, le signal sera aussi renvoyé au dsPIC à travers le connecteur.

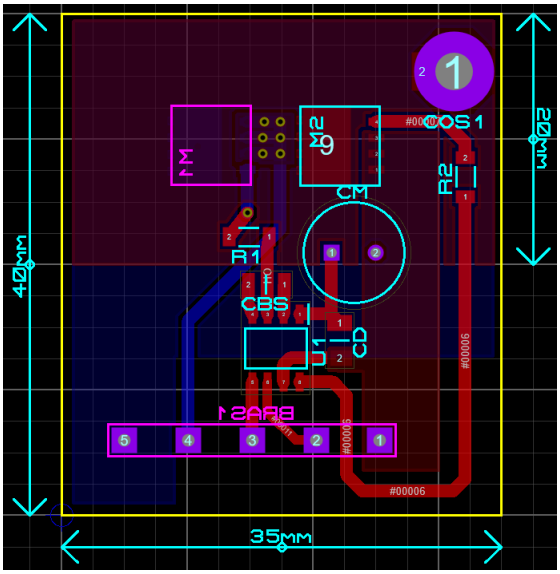


Figure 21 Routage de la carte de bras de pont

Pour conduire la puissance, nous avons choisi d'utiliser les zones. Le condensateur de découplage CM est placé au plus proche des deux transistors de manière à limiter l'inductance de la maille de commutation. La carte est routée en double face, chaque Mosfet sur une face. L'idée est de dissiper au mieux la chaleur, de plus l'objectif est de placer les cartes de bras de pont entre le moteur et la gouverne afin d'intercepter le flux d'air généré, et donc de dissiper plus rapidement la chaleur.

Le transistor high side est commandé en PWM à 20kHz , et génèrera plus de chaleur que le low side. L'utilisation de vias pour connecter la source du high side au drain du low side permet aussi de transférer la chaleur vers la zone de dissipation de l'autre face.

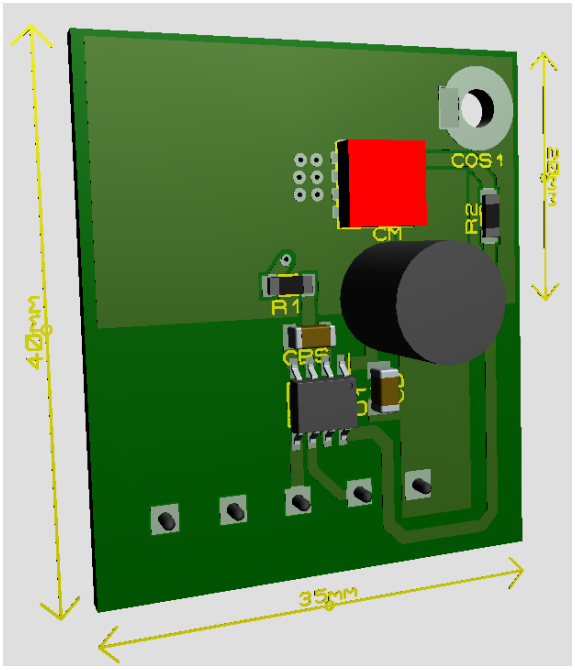


Figure 22 Aperçu d'une carte de bras de pont

La carte de bras de pont sera imprimée en double face afin de pouvoir dissiper la chaleur générée par les transistors à l'aide des zones de conduction. Les 3 cartes seront connectées électriquement et mécaniquement sur la carte de dsPIC. Elles seront branchées orthogonalement de manière à ce qu'elles soient verticales.

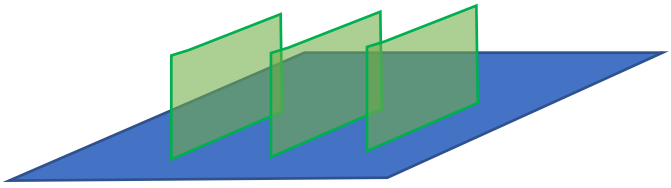
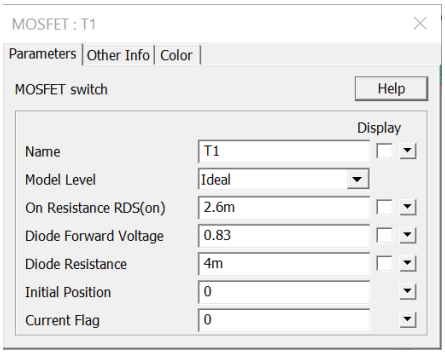


Figure 23 : Disposition de la carte supportant le dsPIC et les bras de l'onduleur

4.3.2. Pertes au niveau du transistor

Afin d'estimer les pertes générées par les MOSFET et d'assurer une bonne dissipation, on renseigne les imperfections des MOSFETs dans le schéma de simulation, à savoir :



La résistance drain-source du transistor, la tension seuil de la diode, ainsi que sa valeur de résistance dynamique

$r_D = \frac{1-0.83}{44} \simeq 4m\Omega$ renseignées dans la Datasheet.

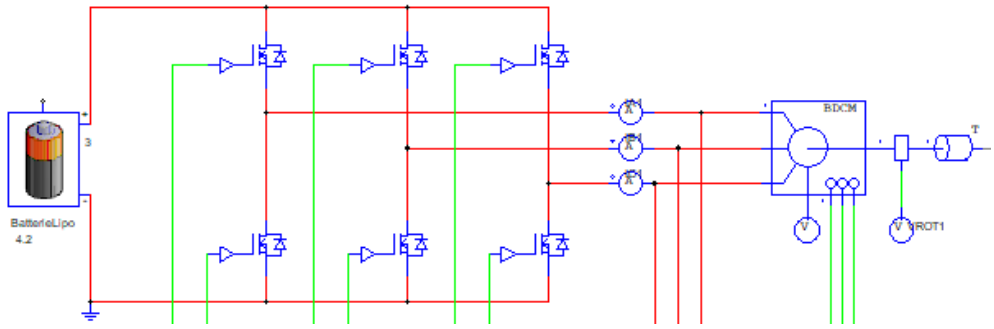


Figure 24 : Schéma de simulation de l'onduleur

Le transistor high-side génèrera plus de pertes que le low-side, car il commute à la fréquence de la PWM lors de la conduction. Estimons alors les pertes maximales dans un transistor high-side avec un rapport cyclique de 0,99 à la vitesse maximale:

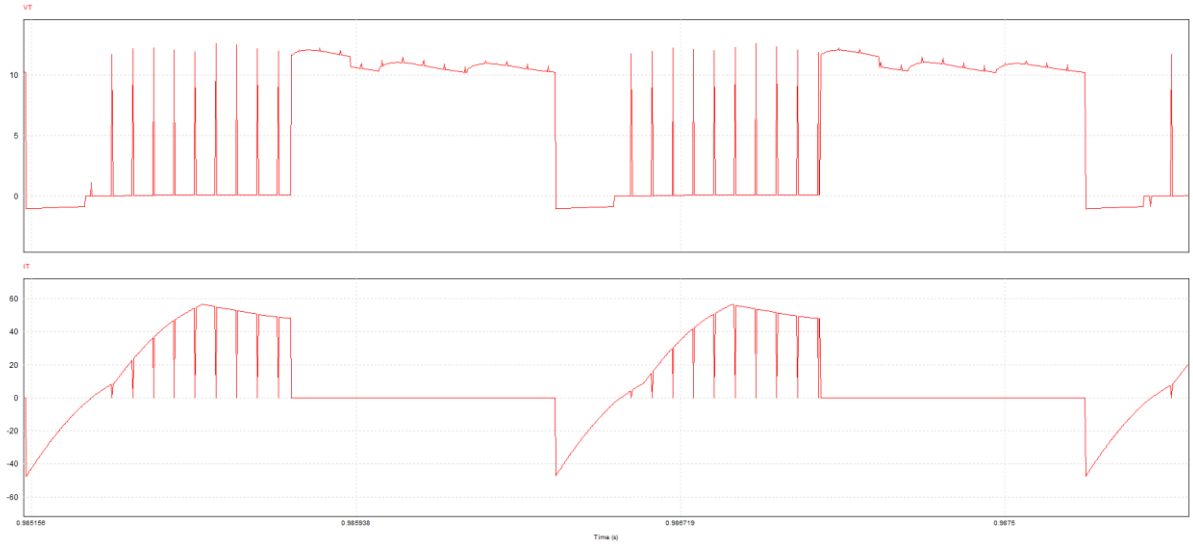
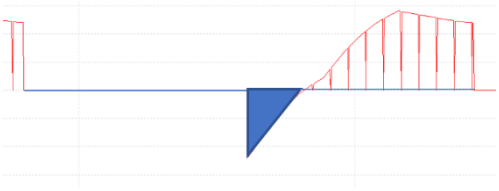


Figure 25: Allure de la tension (vT) et du courant (iT)dans le high side

Les valeurs négatives concernent la diode de structure, nous séparons alors les calculs :

***Pour la diode**, les pertes de mise en conduction sont quasiment nulles. On remarque qu'il n'y a pas de tension inverse, les pertes au blocage sont nulles.

Pour les pertes en conduction, on étudie le courant dans la diode sur une période pour les calculs de valeur moyenne et efficace :



Ainsi $P_{cond} = V_{T0} * I_{Dmoy} + r_T * I_{Deff}^2$

On obtient $\boldsymbol{P_{cond} = 2,91W}$

***Pour le transistor**, pour le calcul de la valeur efficace on trace le carré du courant :

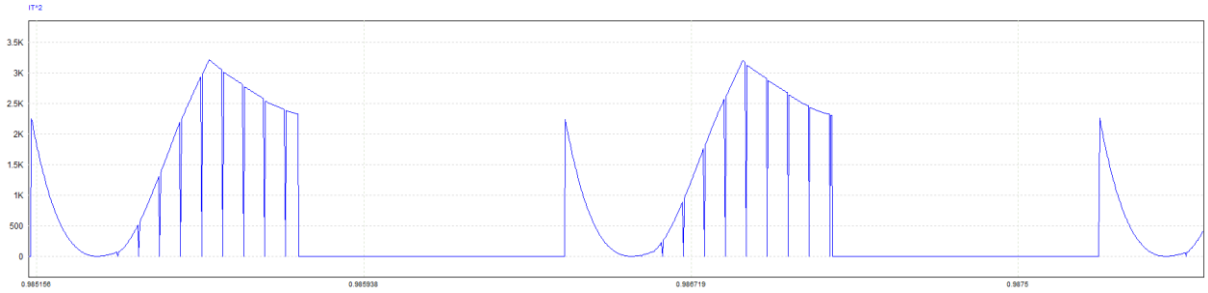


Figure 26 : Allure du carré du courant circulant dans le transistor

Et on fait le calcul d'intégrale en ne considérant pas les commutations, bien que cela surestime les pertes.

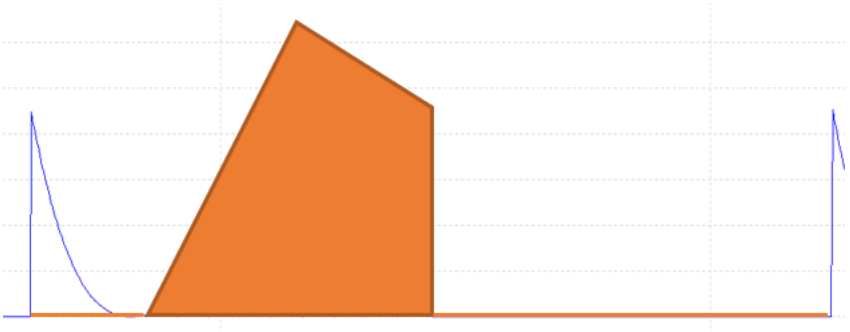


Figure 27 : Illustration du calcul de perte dans le transistor

L'aire du polygone orange vaut

$$A = 0.5 * (3170 * 0.000203 + (2320 + 3170) * 0.000211)$$

$$I_{Deff} = \sqrt{\frac{1}{T} * A} = \sqrt{\frac{1}{0.001277} * 0.5 * (3170 * 0.000203 + (2320 + 3170) * 0.000211)}$$

$$I_{Deff} = 26.56A$$

Ensuite, étant donné que la simulation ne prend pas en compte l'inductance de la maille de commutation, donc pas de surtension au blocage, selon notre routage, nous estimons une inductance de la maille d'une valeur de 40nH. Puis avec la courbe de courant dans la simulation, on obtient une valeur de surtension d'environ 0.95V.

Pour le transistor $\begin{cases} P_{cond} = R_{DSon} * I_{Deff}^2 \\ P_{on} = \frac{F}{2}(E(I + I_{RRM}) * (t_{on} + t_{rr})) \\ P_{off} = \frac{F}{2}(I(E + \Delta V_{DS}) * t_{off}) \end{cases}$

Pour le calcul des pertes lors de la commutation du transistor à la fréquence de la PWM, on fait les calculs ci-dessus et on les multiplie par un ratio entre la durée réelle pendant laquelle il y a commutation à 20kHz et la période.

On obtient les résultats suivants :

Pertes MOSFET									
High side							Diode		
RDSON	2,60E-03	Ω	**Pcond**	**1,83**	**W**		VT0	0,83	V
Intégrale IT^2	0,90095		**Pon**	**3,32**	**W**		Idmoy	2,86	A
Ideff	26,56	A	**Poff**	**3,61**	**W**		rT	4,00E-03	Ω
T	1,27E-03		**PTH**	**8,77**	**W**		Ideff	11,56	A
F	7,87E+02	Hz					**Pcond**	**2,91**	**W**
PWM	20000	Hz					**Pdon**	**0**	**W**
E	10,6	V	duree	0,000414	s		**Pdoff**	**0**	**W**
I	48	A	durant laquelle a lieu la commutation				**PDH**	**2,91**	**W**
IRRM	0	A							
ton	2,00E-06	s							
trr	0,00E+00	s							
di/dt	2,38E+07	A/s	blocage						
lambda	0,00000004	H							
ΔVDS	0,95	V							
toff	2,00E-06	s					**Ptotale H**	**11,67**	**W**

Cette puissance à dissiper est énorme, et les 6cm² de zone de dissipation conseillés ne suffiraient pas, il faudrait un dissipateur CMS pour limiter la température de jonction du transistor.

Cela dit, une erreur a probablement été faite au niveau des hypothèses, nous continuerons le raisonnement avec la surface de dissipation de 6cm² pour chaque MOSFET.

4.4. La carte de dsPIC

Voici le schéma réalisé pour la carte de dsPIC :

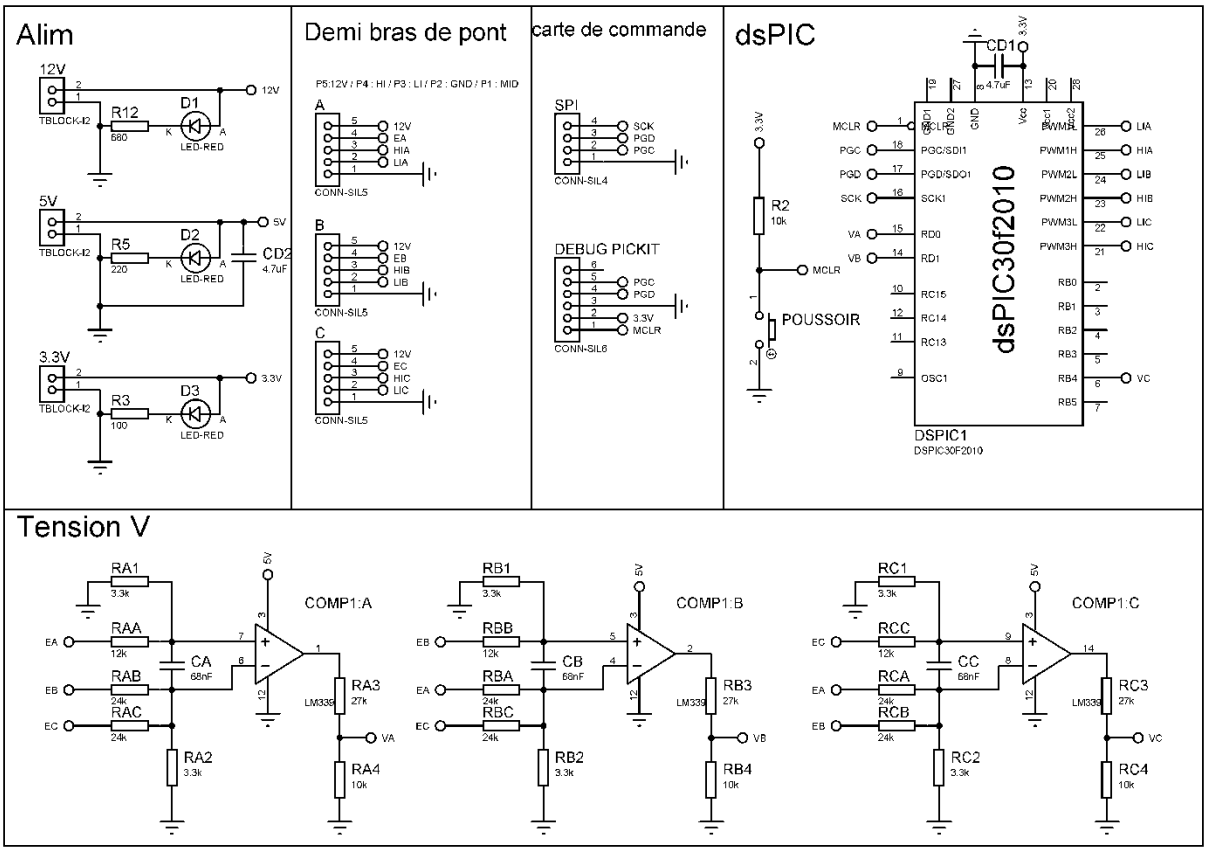


Figure 28 : Schéma de la carte supportant le dsPIC

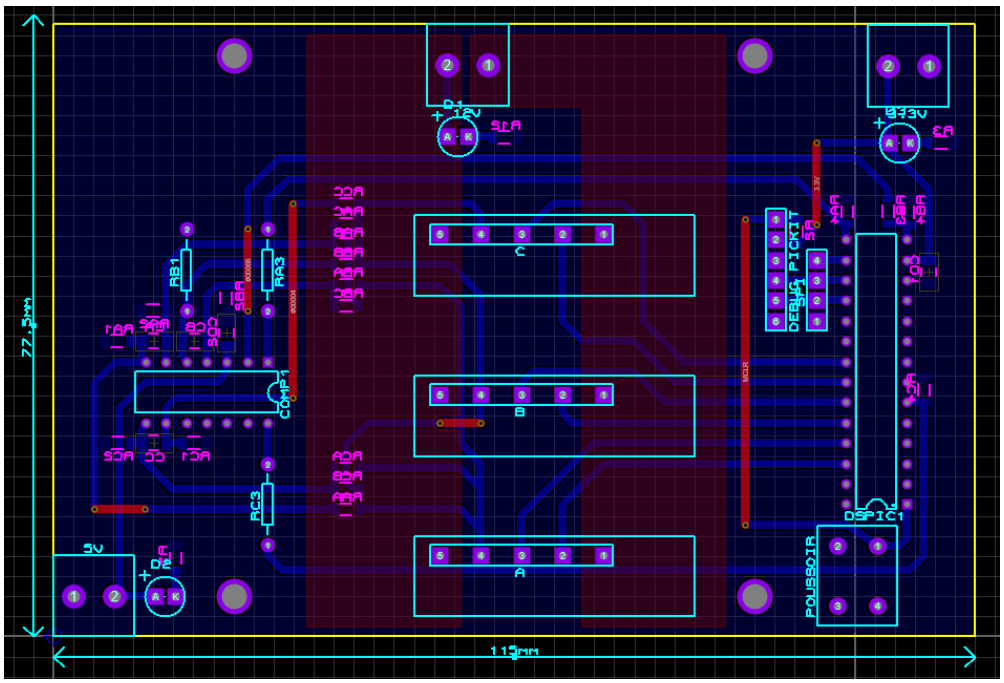


Figure 29 : PCB de la carte du dsPIC

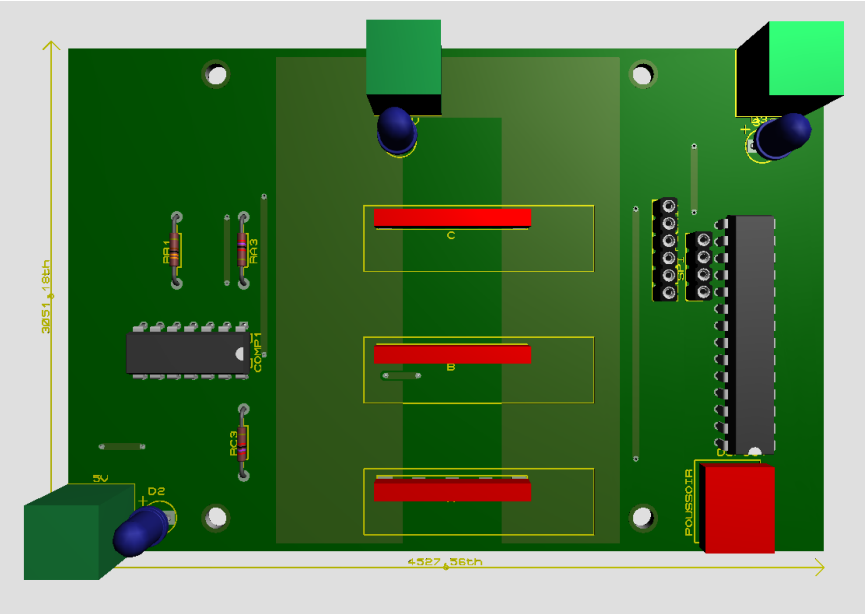


Figure 30 : Vue 3D de la carte du dsPIC

Il est possible de faire tourner le moteur brushless en mode pas à pas jusqu'à une vitesse autour de 4000tr/mi. Au-delà, le moteur décroche avec ce type de commande. Etant donné que nous visons des vitesses autour de 10 000tr/min, il faudra utiliser le mode autopiloté. Cela nécessite de connaître la position du moteur pour afin d'alimenter au bon moment chaque phase du moteur.

Le moteur à disposition ne dispose pourtant pas de capteur de position, ou de sondes de Hall. Le « Zero cross detection » est une solution, cela consiste à déterminer la position du rotor à partir du passage par zéro d'une tension simple du moteur, résultant de la force contre électromotrice. La position sera déterminée selon le type de front.

Comme l'accès au neutre n'est pas possible :

$$\begin{cases} U_{ab} = E_a - E_b \\ U_{bc} = E_b - E_c \\ U_{ca} = E_c - E_a \end{cases}$$

Et

$$\begin{cases} U_{ab} = V_a - V_b \\ U_{bc} = V_b - V_c \\ U_{ca} = V_c - V_a \end{cases}$$

Soit $U_{ab} - U_{ca} = 2V_a - (V_b + V_c) = 2V_a - (-V_a) = 3V_a$

D'où $V_a = \frac{1}{3}(U_{ab} - U_{ca}) = \frac{1}{3}(E_a - E_b - E_c + E_a)$

$$\boxed{V_a = \frac{1}{3}\left(2E_a - (E_b + E_c)\right)}$$

On utilisera alors un comparateur afin de mettre en évidence les changements d'états, les fronts seront détectés par le dsPIC.

Voici le schéma correspondant :

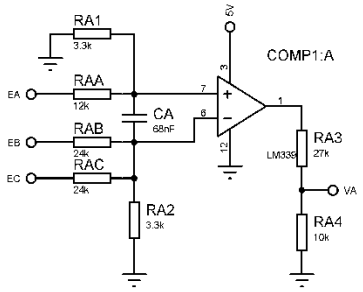


Figure 31 : Schéma de la détection du passage à zéro

En ce qui concerne les valeurs de composants : soit s la sortie du comparateur :

$$s = \frac{RA1}{RAA + RA1} E_a - \frac{\frac{E_b}{RAB} + \frac{E_c}{RAC}}{\frac{1}{RAB} + \frac{1}{RAC} + \frac{1}{RA2}}$$

Dans un premier temps, nous pensions choisir les valeurs de résistance afin de suivre l'équation de V_a. Nous avons ensuite préféré choisir des valeurs de sorte à ce qu'on garde

$(2E_a - (E_b + E_c))$ et en choisissant RA1 et RA2 afin de limiter la différence aux bornes du comparateur.

Le condensateur permet de filtrer les signaux parasites de haute fréquence générés par la PWM à 20kHz, mais il génère aussi un retard, c'est pourquoi il faudrait mesurer ce retard et l'intégrer dans le dsPIC pour la commande du moteur.

Pour déterminer la valeur du condensateur, à partir du schéma précédent, on fait l'étude théorique avec la fonction de transfert (en supposant juste pour le filtrage, un régime sinusoïdal) :

$$\boxed{\frac{\underline{S(p)}}{\underline{E(p)}} = K * \frac{1 + \frac{p}{\omega_n}}{1 + \frac{p}{\omega_d}}}$$

avec la sortie S correspondant à l'entrée V+ du comparateur et l'entrée E correspondant à l'arrivée de Ea la phase du moteur

Où $K = \frac{RA2}{RA2+RA1}$; $\omega_n = \frac{1}{RA2*CA}$; $\omega_d = \frac{1}{\left(RA2+\frac{RAA*RA1}{RAA+RA1}\right)*CA}$

Si on considère une vitesse maximale avoisinant 9000tr/min, cela correspondrait à une fréquence de750Hz. La valeur de CA à 68nF permettrait une deuxième fréquence de cassure fd à 710Hz

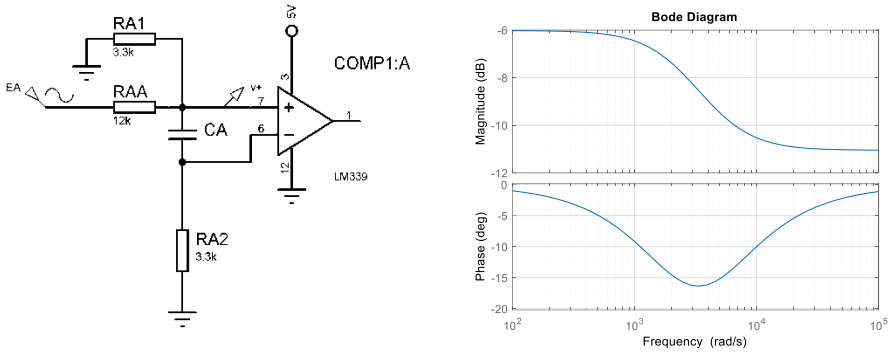


Figure 32 : Tracé du diagramme de Bode

Ensuite, afin de déterminer la valeur du retard, on couple d'une part un deuxième moteur sur le même arbre, avec les mêmes caractéristiques, sauf l'angle d'avance à 0° et la largeur d'impulsion à 180°. Ce deuxième moteur fonctionnera alors en générateur, on branche en étoile trois résistances de forte impédance aux bornes de ses phases. Cela permet d'observer la valeur de la tension simple générée. D'autre part, on a le Montag proposé précédemment pour le zero-cross detection. Voici une illustration :

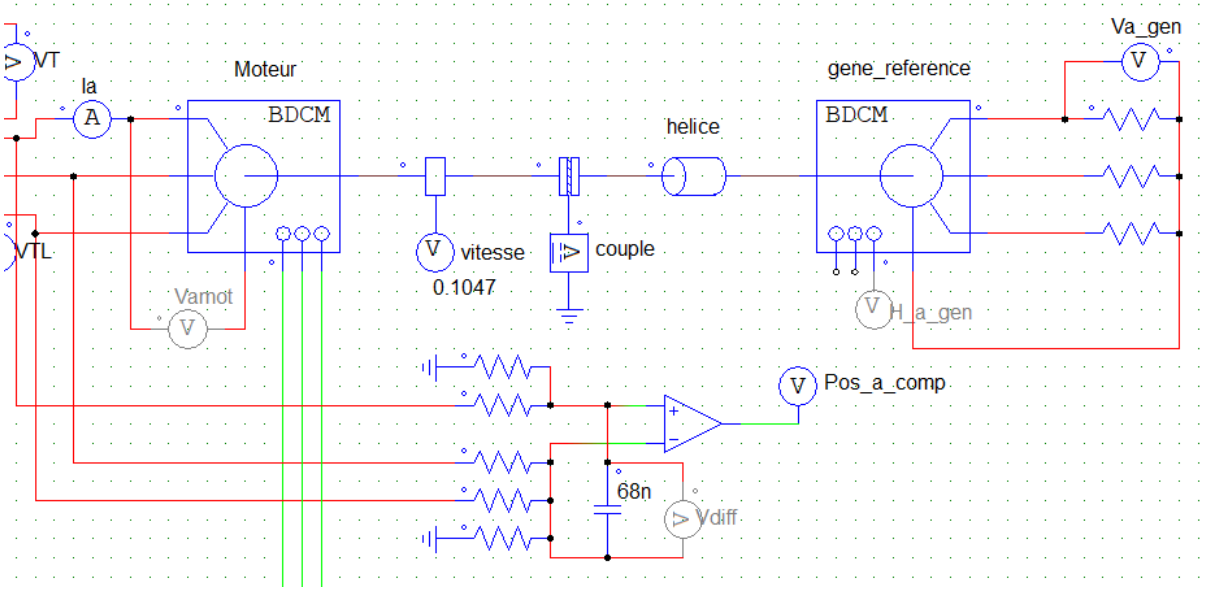


Figure 33 : Composants de simulation pour la mesure du déphasage

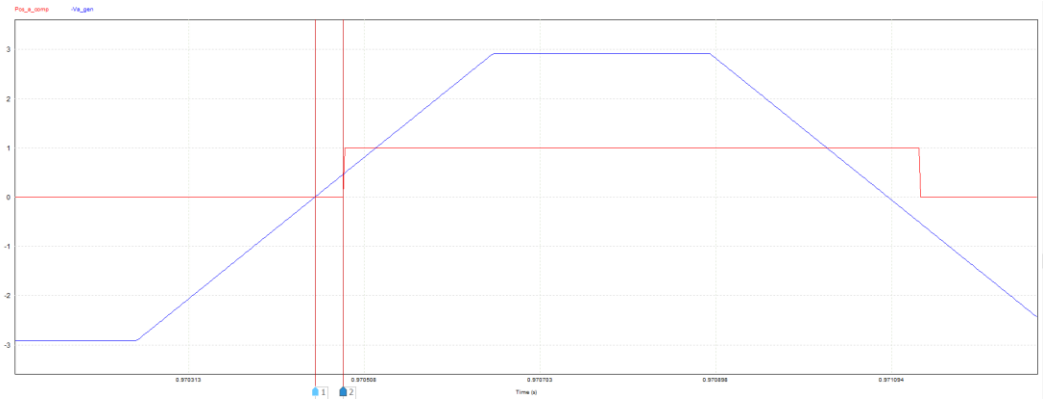


Figure 34 : Mesure du déphasage

En choisissant une même phase, on mesure le décalage entre le passage par zéro de la tension simple issue du générateur, et le passage par zéro de la tension simple reconstituée à partir des trois phases du moteur. On fais ensuit le rapport entre ce décalage et la période électrique, pour chaque valeur du rapport cyclique.

Voici les résultats obtenus :

	C = 68nF		
alpha	T	Δt retard	Retard °
0,4	2,111E-03	8,170E-05	13,9
0,5	1,843E-03	5,980E-05	11,7
0,6	1,645E-03	5,226E-05	11,4
0,7	1,492E-03	4,322E-05	10,4
0,8	1,391E-03	3,905E-05	10,1
0,9	1,307E-03	3,580E-05	9,9
0,99	1,272E-03	3,649E-05	10,3

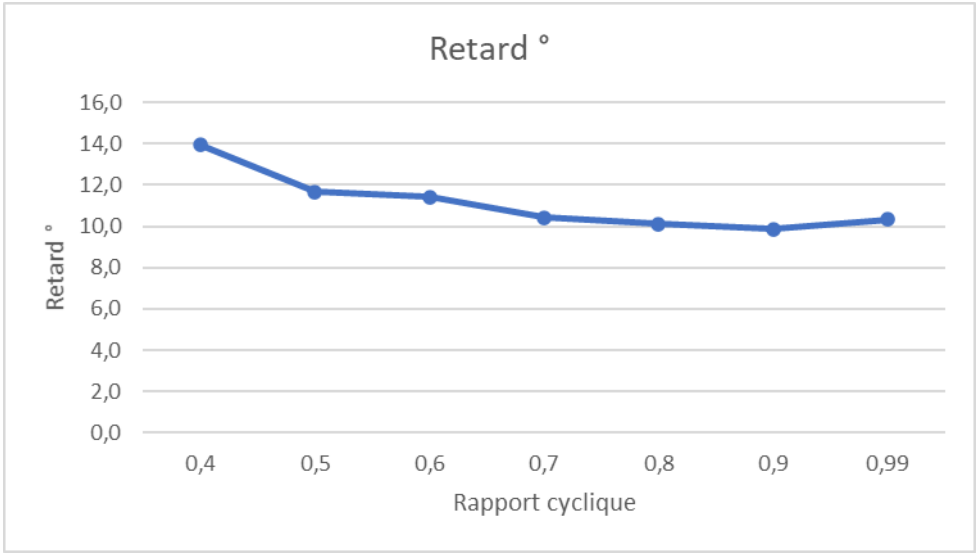


Figure 35 : Estimation du déphasage selon le rapport cyclique

On peut donc estimer un déphasage d'environ 10° généré par le condensateur dans le filtre passe-bas. Ce déphasage sera alors pris en compte par le dsPIC lors de la commande du moteur brushless.

4.5. La carte PIC:

Une autre carte principale sera celle du PIC16F1619, elle permettra de brancher le module bluetooth, d'assurer la communication avec le dsPIC à travers la liaison SPI. Des connecteurs sont aussi prévus pour l'alimentation ainsi que la commande du servo moteur et la programmation du PIC. Les tensions de 5V et 3,3V y seront accessibles pour l'alimentation des composants, la tension de la batterie y sera aussi connectée et permettra d'estimer son état de charge.

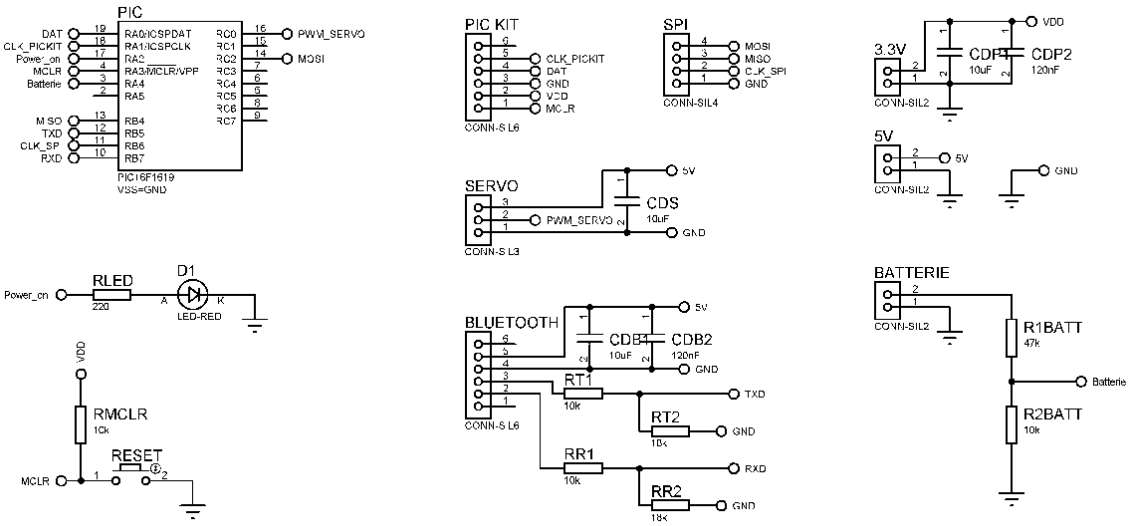


Figure 36 : Schéma électrique de la carte PIC

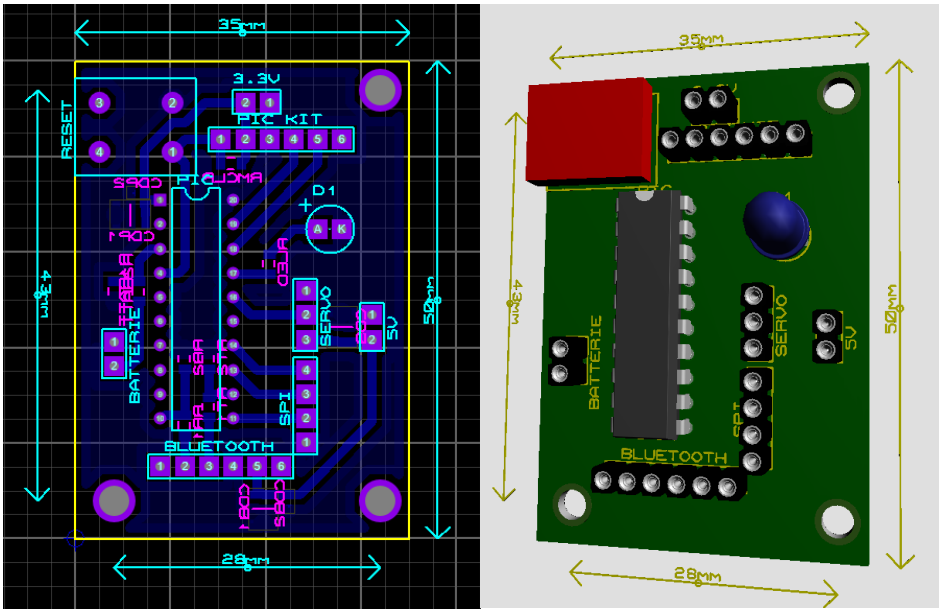


Figure 37 : Routage et vue 3D de la carte PIC

IV. Electronique numérique

1. Interface smartphone

Le contrôle de l'aéroglisseur se fera via une application sur smartphone connecté en Bluetooth à la carte de commande générale. L'interface comportera un pad de commande de direction (droite/gauche) proportionnel et de propulsion également proportionnelle et d'un retour de la tension de la batterie ainsi que d'un bouton On/Off. On utilisera l'application Bluetooth Electronics de keuwlsoft disponible sous Android© qui se présente de la façon suivante :

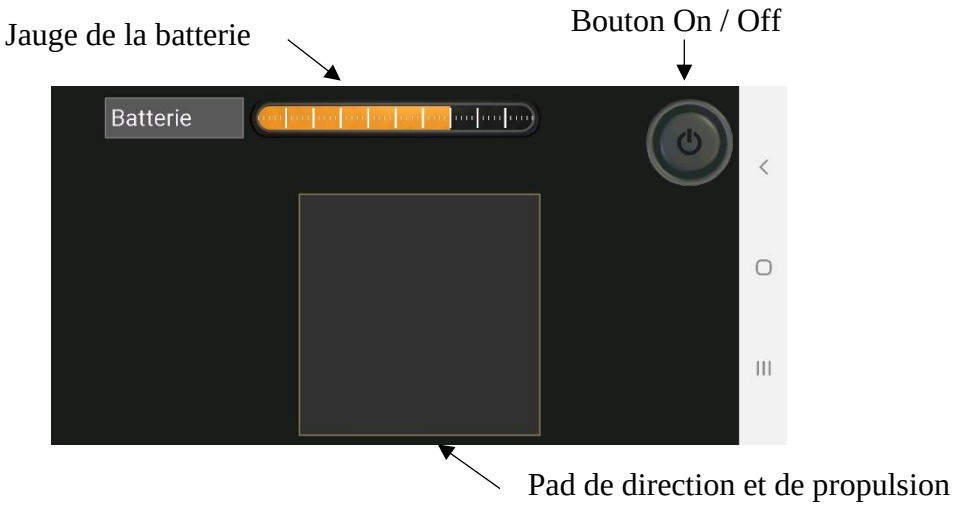


Figure 38 : Interface graphique du smartphone

Instructions :

Instruction du bouton On/Off : `H`=On, `h`=Off

Indicateur de la batterie : `*B__*` avec __ égale au pourcentage de la batterie

Pad de direction : `PX__Y__p` avec __ compris entre 0 et 255

2. Microcontrôleur principal

2.1. Schéma pins

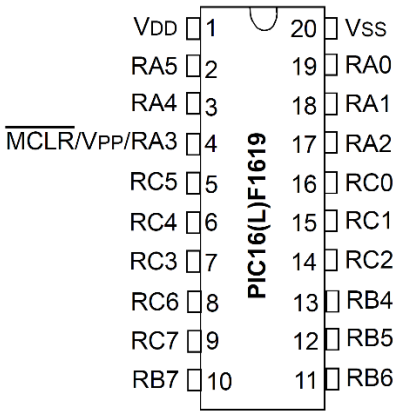


Figure 39 : Brochage du PIC

Affectation des pins :

- 20 VSS : Ground
- 1 VDD : 3.3V
- 19 RA0 : ICSPDAT
- 18 RA1 : ICSPCLK
- 4 RA3 : MCLR
- 17 RA2 : LED Power ON
- 16 RC0 : PWM pour servo moteur
- 12 RB5 : UART RX
- 10 RB7 : UART TX
- 13 RB4 : SDI/MISO
- 14 RC2 : SDO/MOSI
- 11 RB6 : SCK
- 3 RA4 : CAN tension batterie

2.2. Organisation du programme

Fichier	Fonction inclus	Fonctionnalité
Main.c	`init_Pin()`	Initialisation des pins I/O
	`init_Osc()`	Initialisation oscillateur à 4MHz
	`init_ADC()`	Initialisation CAN pour la batterie
	`arret()`	Procédure d'arrêt
	`niveau_batterie()`	Calcul et envoi au téléphone le niveau de batterie
SPI.c	`init_SPI()`	Initialisation de la liaison SPI
	`SPI_write(Data_SPI)`	Ecriture de la donnée sur la liaison SPI
PWM.c	`init_PWM()`	Initialisation de la PWM pour le servo
	`PWM_angle(angle)`	Actualisation du rapport cyclique en fonction de l'angle souhaité
Interruption.c	`init_interruption()`	Initialisation des interruption
	`__interrupt() Interuption()`	Fonction d'interruption
Bluetooth.c	`init_Bluetooth()`	Initialisation de la liaison UART avec le module Bluetooth
	`Bluetooth_write(taille,Data)`	Ecriture de la donnée sur la liaison UART
	`analyse_data_B_recu(Data)`	Analyse des données reçu du module Bluetooth

2.3. PWM

La PWM va nous servir à commander le servomoteur pour diriger l'aéroglisseur.

Nous commençons par régler la fréquence d'oscillation de l'oscillateur interne car la liaison UART pour communiquer avec le module Bluetooth et la liaison SPI nécessitent une fréquence de 4MHz.

Pour ce faire on change juste le registre suivant :

```
void init_Osc(void){
    //Oscillateur a 4 MHz
    OSCCONbits.IRCF = 0b1101;
}
```

Nous choisissons d'utiliser les modules Timer4 et PWM3

On commence par initialiser les registres suivants du Timer4 :

```
//Init TIMER4
T4CLKCONbits.CS = 0b0000;   //Frequence de ref = Fosc/4

T4CONbits.ON = 1;           //ON
T4CONbits.OUTPS = 0b0000;   //Postscaller INUTILE

T4HLTbits.PSYNC = 1;        //synchro sur l'horloge
T4HLTbits.CKPOL = 0;        //Sur front montant
T4HLTbits.CKSYNC = 1;       //Activation de la synchro sur l'horloge
T4HLTbits.MODE = 0b00000;   //Mode Free Run (Software gate)

T4RSTbits.RSEL = 0b1101;    //Reset sur la PWM3
```

D’après la datasheet :

$$F_{timer} = \frac{Fosc}{4 \cdot PS}$$

On se fixe un préscaler (PS) de 128 ainsi :

$$F_{timer} = 7812.5Hz, \qquad T_{timer} = 128\mu s$$

Pour un servomoteur l’impulsion contenant l’information de position doit arriver au maximum toutes les 20ms soit une fréquence de plus de 50Hz. La PWM devra donc avoir une période inférieure à $T_{PWM} = \frac{T}{T_{timer}} = \frac{20}{0.128} = 155$

```
PR4 = 154;                  //Definition de la periode
T4CONbits.CKPS = 0b111;     //Prescaller de 128
```

Nous initialisons ensuite la PWM :

```
PPSLOCK = 0;                //Debloque le PPS
PWM3CONbits.PWM3EN = 1;     //Activation
PWM3CONbits.PWM3OUT = 1;    //mettre a 1
PWM3CONbits.PWM3POL = 0;    //Actif a l'etat haut

CCPTMRSbits.P3TSEL = 0b01;  //PWM3 base sur Timer 4
RC0PPS = 0b00001110;        //PWM3 lie a RC0

PWM3DCH = 0b00000000;       //Rapport cycle (registre bits de poids forts)
PWM3DCL = 0b00000000;       //Rapport cycle (registre bits de poids faibles)

PPSLOCK = 1;                //Bloque le PPS
```

Pour commander le servo moteur, il faut faire varier le rapport cyclique pour avoir une impulsion comprise entre 1.3 et 1.7 ms. Sachant que le rapport cyclique à une valeur max de 1024, il faut donc qu’il soit compris entre $1024 \cdot \frac{1.3}{20} = 66$ et $1024 \cdot \frac{1.7}{20} = 87$

Il ne nous reste plus qu’à changer le rapport cyclique en modifiant les registres adéquates, pour des raisons de simplicité nous créons une fonction :

```
void PWM_angle(unsigned int angle)    //Angle code entre 0 et 255
{
    int DC = 66 + angle*21/255;
    PWM3DCH = (DC & 0b0000001111111100) >> 2;
    PWM3DCL = (DC & 0b0000000000000011) << 6;
}
```

Malheureusement sans oscilloscope pour observer le signal de sortie il est difficile de savoir si cela fonctionne correctement.

2.4. Liaison Bluetooth

La communication entre la carte et le module Bluetooth se fait via une liaison UART à une vitesse de 9600 bauds.

2.4.1. Réception

Nous souhaitions dans un premier temps implémenter la réception des données.

Définition des pins

```
PPSLOCK = 0;                //Debloque le PPS
//RX -> RB5  par defaut
//TX -> RB7
RB7PPS = 0b10010;
PPSLOCK = 1;                //Bloque le PPS
```

Ensuite on fait l'initialisation en configurant différents registres :

```
RC1STAbits.SPEN = 1;        //Reception ON
RC1STAbits.CREN = 1;        //Reception en continue
TX1STAbits.TXEN = 1;        //Transmission ON
TX1STAbits.SYNC = 0;        //Assynchrone
TX1STAbits.BRGH = 1;        //Haut debit

BAUD1CONbits.RCIDL = 0;     //
BAUD1CONbits.SCKP = 0;      //Pas d'inversion de la data
BAUD1CONbits.BRG16 = 1;     //Baud rate sur 16 bits
```

Pour régler ensuite le baud rate, la datasheet nous donne la formule suivante :

On pose $n = SP1BRGH \oplus SP1BRGL$ (concaténation)

$$n = \frac{F_{osc}}{4 \cdot Baud\ rate} - 1 = \frac{4\,000\,000}{4 \cdot 9600} - 1 = 103.16$$

Ainsi $n = 103 = 0x67$:

```
SP1BRGL = 0x67;             //Baud rate
SP1BRGH = 0;
```

Avec cette configuration nous avons un pourcentage d'erreur dans la liaison de :

$$Erreur\ = \frac{Br\ calculer - Br\ desirer}{Br\ desirer} = \frac{9615 - 9600}{9600} = 0.16\%$$

Lorsqu'une donnée est reçue, une interruption est déclenchée dans laquelle nous traitons l'information :

```
//Reception Bluetooth
if (PIR1bits.RCIF)
    {
        if (RC1REG == 'H'){              //Si on recoit un H on allume
            Etat = 1;
        }else if (RC1REG == 'h'){        //Si on recoit un h on arret
            arret();
        }else if (RC1REG == 'P'){        //Debut des intructions de position
            rg_data_B_recu = 0;

            //SPI_write(1, RC1REG);
        }else if (RC1REG == 'p' && rg_data_B_recu != 10){      //Fin des instructions
de position
            //SPI_write(1, RC1REG);
            Data_B_recu[rg_data_B_recu] = RC1REG;
            rg_data_B_recu = 10;
            analyse_data_B_recu(Data_B_recu);     //Analyse des donnees de position
recu
        }else if (rg_data_B_recu <10){  //Stockage des 10 caracteres suivant le debut
            //SPI_write(1, RC1REG);
            Data_B_recu[rg_data_B_recu] = RC1REG;
            rg_data_B_recu++;
        }
        PIR1bits.RCIF = 0;               //Reset du flag
    }
```

On peut recevoir un

- 'H' cela signifie que l'état est sur On, on envoi donc les ordres au dsPIC et au servo
- 'h' on entre en procédure d'arrêt
- Si on reçoit un 'P' cela signifie le début de la trame des consignes de propulsion et de direction
- On stock donc les 10 caractères suivant pour les analysés lorsque c'est fini c'est-à-dire lorsque un 'p' est reçu. La fonction d'analyse étant assez lourde et pas pertinente elle n'est pas décrite ici.

2.4.2. Transmission

Nous souhaitons envoyer des données au smartphone de l'utilisateur pour monitorer le niveau de charge de la batterie. Pour cela, on créé une fonction qui envoie un maximum de 6 caractères les uns après les autres.

```
void Bluetooth_write(const unsigned int taille , char Data_B_envoie[6])
{
    for (int i = 0; i<taille; i++){
        TX1REG = Data_B_envoie[i];       //Envoie d'un octet/caractere
        while(!TX1STAbits.TRMT) {}       //Attente de l'envoie
    }
}
```

2.5. SPI coté carte principale

Le but de la liaison SPI est d'envoyer à la carte de gestion de l'onduleur la fréquence de rotation de l'hélice, pour ce faire la valeur image de la vitesse est codé sur un octet soit entre 0 et 255 représentants respectivement une vitesse de 0 à 9000tr/min soit une résolution de 35.3tr/min. Ainsi nous nous concentrons dans un premier temps sur l'envoie de données.

2.5.1. Envoyer les données

Commençons par définir les pins utilisés

```
//PINs
//SDI -> RB4 (MISO)  Par default
PPSLOCK = 0;                //Debloque le PPS
//SCK -> RB6
RB6PPS = 0b10000;
//SS  -> RC5
//SDO -> RC2 (MOSI)
RC2PPS = 0b10001;           //RC2 relier a SDO
PPSLOCK = 1;                //Bloque le PPS
```

Ensuite on fait l'initialisation en configurant différent registre

```
//Forme de l'horloge
SSP1STATbits.CKE = 0;       //Actif sur front montant
SSP1CON1bits.CKP = 1;       //Polarité inversé

SSP1STATbits.SMP = 0;       //Enchantillonage a la fin
SSP1CON1bits.SSPOV = 0;     //Pas d'overflow
SSP1CON1bits.SSPM = 0b1010; //Master Mode Clock = Fosc/4

SSP1CON1bits.SSPEN = 1;     //Activation
```

Ensuite il est nécessaire de fixer un baud rate, d'après la datasheet en ayant une fréquence d'oscillateur de 4MHz on peut avoir un baud rate à 100kHz

$$SSP1ADD = \frac{F_{osc}}{Baude\ rate \cdot 4} - 1$$

Soit $SSP1ADD = \frac{4\,000\,000}{100\,000 \cdot 4} - 1 = 9$

```
SSP1ADD = 0x09;             //Baud rate
```

Les données sont envoyées à l'aide de la fonction suivante :

```
void SPI_write(int SS, char Data_SPI)
{
    SSP1BUF = Data_SPI;             //Envoie des donnees
    while(!PIR1bits.SSP1IF) {}      //Attente de la fin de l'envoie
}
```

2.6. Gestion de la batterie

La gestion de la batterie sera assurée par le microcontrôleur qui a pour mission de monitorer le niveau de charge et de l'envoyer sur le téléphone et également d'effectuer un arrêt si le niveau de charge est trop bas (en-dessous de 9V).

Pour pouvoir utiliser le CAN du microcontrôleur on utilise un pont diviseur :

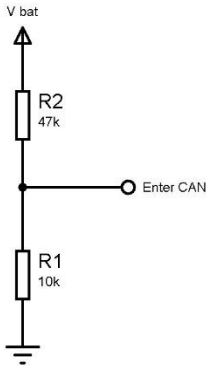


Figure 40 : Pont diviseur pour monitorer la tension de la batterie

Cas de figure	Batterie chargée à 100%	Batterie à tension nominal	Batterie déchargée
Tension de la batterie	12.6V	11.1v	10V
Tension en entrée du CAN	2.21V	1.95V	1.75V
Valeur pleine échèle du CAN	3.3V	3.3V	3.3V
Valeur après conversion	686	605	543

On supposera que la relation entre tension de la batterie et l'état de charge est linéaire même si ce n'est pas le cas en réalité :

$$Niv\ charge = \frac{Tension\ actuel - Tension\ min}{Tension\ Max - Tension\ min} \cdot 100$$

2.6.1. Lecture du niveau de batterie

Pour lire la tension délivrée par la batterie, on programme le CAN

Initialisation :

```
void init_ADC(void)
{
    ADCON0bits.CHS = 0b00011;           //AN3 -> RA4
    ADCON0bits.ADON = 1;                //ADC ON
    ADCON1bits.ADFM = 1;                //Resultat aligne a droite
    ADCON1bits.ADCS = 0b100;            //Fosc/4
    ADCON1bits.ADPREF = 0b00;           //Reference à Vcc
}
```

La lecture et l'envoi sont déclenchés dans le fonction `niveau_batterie()` appelé dans le programme principal

La lecture est déclenchée par la ligne suivante :

```
ADCON0bits.ADGO = 1;                          //Lecture du niveau de la batterie
```

Le programme entre alors dans l'interruption suivante dans laquelle le pourcentage de batterie est sauvegarder dans la variable globale : `Niv_Bat`

```
//Constante
#define Tension_min_Bat 550
#define Tension_max_Bat 685

//ADC
if (PIR1bits.ADIF)
{
    PIR1bits.ADIF = 0;                    //Reset du flag
    int res = ADRESH *256 + ADRESL;       //Lecture du resultat

    if (res > Tension_max_Bat)            //Calcul du "pourcentage" de charge
    {
        Niv_Bat = 100;
    }else{
        Niv_Bat = (res-Tension_min_Bat)*100/( Tension_max_Bat-Tension_min_Bat);
    }

    if (res < Tension_min_Bat)            //Arret si tension trop basse
    {
        arret();
    }
}
```

2.6.2. Envoie au téléphone

Le niveau de charge de la batterie est géré dans une fonction dédiée :

```
void niveau_batterie(void)    //Aquisition du niveau de la batterie + envoie au
telephone
{
    ADCON0bits.ADGO = 1;                          //Lecture du niveau de la batterie
    while (ADCON0bits.ADGO) {}                    //Attente de la lecture
    char data [6] = {'*', 'B', '0', '0', '0', '*'};
    if (Niv_Bat >=100)
    {
        data[2] = '1';
    }else{
        data[2] = '0';
        data[3] = Niv_Bat/10 +48;       //Bidouille de conversion d'un entier en
caractère ASCII
        data[4] = Niv_Bat%10 +48;
    }
    Bluetooth_write(6 , data);                    //Envoie au smartphone
}
```

2.7. Programme principal

La vitesse de consigne est soumise à une rampe, lors de l'essai lâché on observe une décélération maximum de 3000 tr/min/s.

Pour ce faire la vitesse de consigne (entre 0 et 255) sera augmenté ou diminué d'une unité à la fois (soit plus ou moins 35.3 tr/min) et envoyer au dsPIC toutes les 11.7ms on prendra enfaite 10ms.

```
void main(void) {
    //Initialisation
    init_Pin();
    init_Osc();
    init_PWM();
    init_Bluetooth();
    init_interruption();
    init_SPI();
    init_ADC();

    //Boucle infinie
    while(1)
    {
        if(Etat)                        //Etat en marche
        {
            LATAbits.LATA2 = 1;         //LED signalant l'etat de marche

            PWM_angle(direction);       //MAJ de la direction
        }else{
            vitesse_consigne = 0;
        }

        if (vitesse_consigne > vitesse_actuelle)
        {
            vitesse_actuelle++;
        }else if (vitesse_consigne < vitesse_actuelle)
        {
            vitesse_actuelle--;
        }
        SPI_write(1, vitesse_actuelle);       //Transmission de la vitesse voulu au
dsPIC
        niveau_batterie();              //MAJ du niveau de la batterie
        __delay_ms(10);
    }
    return;
}
```

La procédure d'arrêt qui est appelé lorsque l'utilisateur appuie sur le bouton d'arrêt ou que le niveau de la batterie est trop faible et la suivante :

```
void arret(void)            //Procedure d'arret
{
    Etat = 0;
    LATAbits.LATA2 = 0;

    vitesse_consigne = 0;
    direction = 128;
    PWM_angle(direction);
}
```

3. Microcontrôleur de gestion de l'onduleur

3.1. Schéma pins

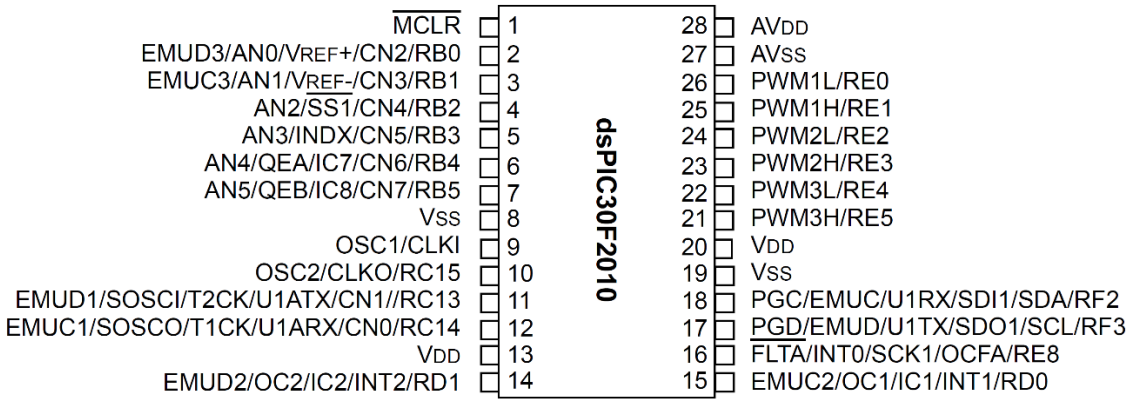


Figure 41 : Brochage du dsPIC

Affectation des pins :

- 8, 19 Vss : GND
- 13, 20 Vdd : 5V
- 1 MCLR
- 17 PGD / SDO1
- 18 PGC / SDI1
- 16 SCK1
- 25 PWM1H : T1
- 26 PWM1L : T2
- 23 PWM2H : T3
- 24 PWM2L : T4
- 21 PWM3H : T5
- 22 PWM3L : T6
- 2 RB0 : Led de débogage (allumée en mode autopiloté)
- 15 IC1 : Va
- 14 IC2 : Vb
- 6 IC7 : Vc

3.2. Organisation du programme

Fichier	Fonction inclus	Fonctionnalité
Main.c	`init_pin ()`	Initialisation des pins I/O
SPI.c	`init_SPI()`	Initialisation de la liaison SPI
	`__attribute__ ((interrupt (auto_psv)))_SPI1Interrupt (void)`	Fonction d'interruption lors de la reception de donné de la liaison SPI
PWM.c	`init_PWM()`	Initialisation de la PWM
	`Alim()`	Mise a jour des sorties de PWM en fonction du secteur
Gestion_ moteur.c	`Mode_pas_pas()`	Gestion du mode pas à pas
	`Mode_autopilote()`	Gestion du mode autopilote
	`void init_timer()`	Initialisation des Timer 1 et 2
	`Init_IC1()`	Initialisation du module input capture 1
	`Init_IC2()`	Initialisation du module input capture 2
	`Init_IC7()`	Initialisation du module input capture 7
	`__attribute__((__interrupt__, __auto_psv__)) _T2Interrupt()`	Fonction d'interruption du Timer 2
	`__attribute__((interrupt, no_auto_psv )) _IC1Interrupt(void)`	Fonction d'interruption de input capture 1
	`__attribute__((interrupt, no_auto_psv )) _IC2Interrupt(void)`	Fonction d'interruption de input capture 2
	`__attribute__((interrupt, no_auto_psv )) _IC7Interrupt(void)`	Fonction d'interruption de input capture 7

3.3. SPI

La liaison SPI sur le dsPIC nous sert à recevoir les ordres sur la vitesse que doit avoir l'hélice.

Initialisation de la liaison SPI :

```
void init_SPI(void)
{
    SPI1CONbits.MODE16=0;       //Donnees sur 8 bits
    SPI1CONbits.CKP=1;          //Polarite clock inverser
    SPI1CONbits.CKE=0;          //Actif sur front montant
    SPI1CONbits.SSEN=0;         //Pas d'utilisation de SS
    SPI1CONbits.MSTEN=0;        //Mode esclave
    SPI1CONbits.SMP=0;          //A 0 en mode esclave
    SPI1STATbits.SPIROV=0;      //Pas d'overflow

    SPI1STATbits.SPIEN=1;       //Activation du SPI

    //Interruption SPI
    IFS0bits.SPI1IF=0;          //Flag a 0
    IPC2bits.SPI1IP=0b101;      //Priorite 5
    IEC0bits.SPI1IE=1;          //Activation de l'interruption
}
```

Pour réceptionner la donnée on utilise l'interruption suivante :

```
void __attribute__((__interrupt__(auto_psv))) _SPI1Interrupt(void)
{
    IFS0bits.SPI1IF = 0;
    vitesse = SPI1BUF;
}
```

La donnée reçue est stockée dans la variable globale `vitesse` pour être ensuit utilisée pour régler la vitesse.

3.4. PWM

Le dsPIC comporte trois modules de PWM indépendants relier chacun à deux pins de sorties, chacun des modules gèrera un bras de pont de l'onduleur.

Voici l'initialisation des PWMs :

```
void init_PWM(void)
{
    PTCONbits.PTMOD = 0;         //Mode freerun
    PTCONbits.PTCKPS = 0b00;     //Prescaller a 1
    PTMRbits.PTDIR = 0;          //Dirrection croissante

    PTMR = 0;                    //Initialisation du compteur a 0
    PTPER = FCY/(FPWM) -1;       //=91 si Fpwm = 20kHz

    PWMCON1bits.PMOD1 = 1;       //L et H independant
    PWMCON1bits.PEN1H = 1;       //Activation de H
    PWMCON1bits.PEN1L = 1;       //Activation de L
    PDC1 = PTPER/1024*512;       //Initialisation du DC a 50%

    PWMCON1bits.PMOD2 = 1;       //Idem 1
    PWMCON1bits.PEN2H = 1;
    PWMCON1bits.PEN2L = 1;
    PDC2 = PTPER/1024*512;

    PWMCON1bits.PMOD3 = 1;       //Idem 1
    PWMCON1bits.PEN3H = 1;
    PWMCON1bits.PEN3L = 1;
    PDC3 = PTPER/1024*512;

    OVDCON = 0;                   //Initialisation de la commande de chacun des pins
(rien en sortie)

    PTCONbits.PTEN = 1;          //Activation du module
}
```

Ensuite il suffit d'actualiser le registre OVDCON grâce à la fonction suivante :

```
void alim(void)
{
    //Activation des sorties de PWM en fonction du secteur
    switch (secteur)
    {
        case 0:
            OVDCON = 0b0000001000000100; //secteur 0 1H et 2L
            break;
        case 1:
            OVDCON = 0b0000001000010000; //secteur 1 1H et 3L
            break;
        case 2:
            OVDCON = 0b0000100000010000; //secteur 2 2H et 3L
            break;
        case 3:
            OVDCON = 0b0000100000000001; //secteur 3 2H et 1L
            break;
        case 4:
            OVDCON = 0b0010000000000001; //secteur 4 3H et 1L
            break;
        case 5:
            OVDCON = 0b0010000000000100; //secteur 5 3H et 2L
            break;
    }
}
```

3.5. Mode pas à pas

Pour simplifier le programme et surtout en limité sa complexité, on a découpé une période électrique en 6 secteurs de 60° électrique chacun :

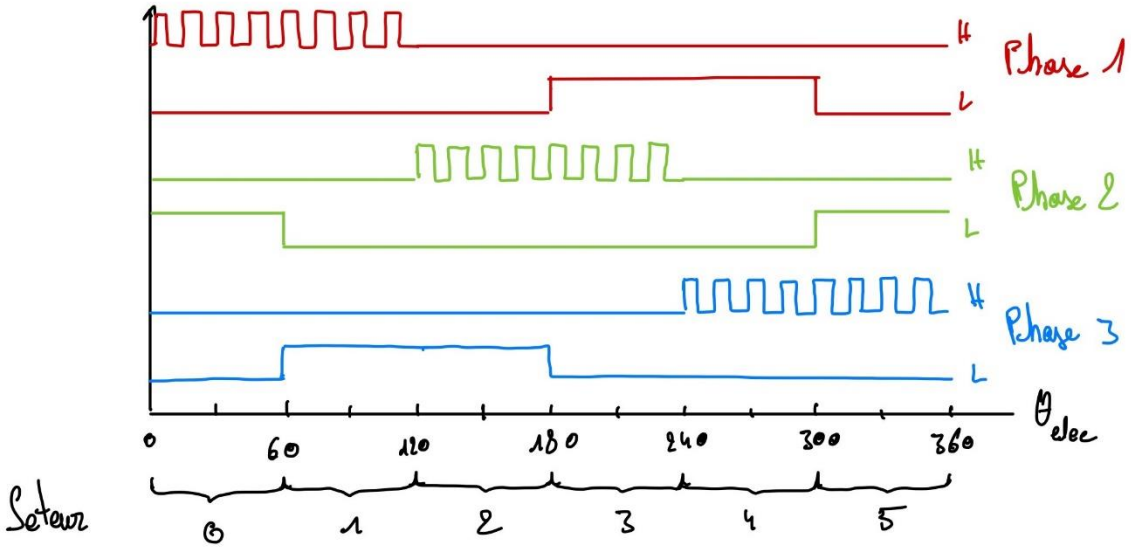


Figure 42 : Découpage des secteurs électrique

On a choisi d'applique la PWM uniquement sur les transistors haut, car si nous appliquons la PWM aux deux transistors et qu'ils ne sont pas parfaitement synchronisés, on risque de ne pas avoir le rapport cyclique souhaité, de plus cela entraine des pertes évitables.

Au début de la période électrique, la fréquence électrique et la valeur des rapports cycliques seront actualisées, et au début de chaque secteur, on actualise quelle sortie est actif ou pas.

Pour avoir une temporisation précise nous utiliserons le Timer 1, plutôt qu'un « delay », pour attendre entre le passage d'un secteur à un autre. Sa période est définie par

$$T = \frac{PS \cdot Per}{Fcy}$$

Avec PS le préscaller (fixé à 8), Per la période & Fcy la fréquence des cycles d'instruction (définie à 1 842 500 Hz)

La variable vitesse dans le programme est codée sur 1 octet donc entre 0 et 255, ainsi pour une vitesse maximale de 9000 tr/min sa résolution est de $q = \frac{9000}{255} = 35.3$ tr/min

La vitesse minimum étant de 1 ce qui correspond à une vitesse de rotation de 35.3 tr/min soit une fréquence électrique de 2.94Hz donc une période électrique de 340ms et donc un changement de secteur tous les 56.6ms

$$Per = \frac{T \cdot Fcy}{PS} = \frac{0.0566 \cdot 1842500}{8} = 13036$$

Ainsi la période du Timer sera définie de la manière suivante :

$$Per = \frac{13036}{Vitesse}$$

Initialisation du Timer 1

```
//Initialisation Timer 1 pour le Mode Pas a Pas
    T1CONbits.TCS = 0;                  //Selection de l'hologe
    T1CONbits.TGATE = 0;

    T1CONbits.TCKPS = 0b01;             //Predivision par 8 (=0b01)
    PR1 = 13035;                         //Periode Max

    T1CONbits.TSYNC = 1;                //Synchronisation sur l'horloge
    T1CONbits.TON = 0;                  //Desactivation

    IFS0bits.T1IF = 0;                  //Flag a 0
    IEC0bits.T1IE = 0;                  //Desactivation de l'interruption
```

Le rapport cyclique nous permet de modifier la tension moyenne aux bornes du moteur. Le rapport cyclique est proportionnel à la vitesse. Quand la vitesse est nulle le rapport cyclique est a 0 et quand la vitesse est à 9000 tr/min (vitesse maximal) le rapport cyclique est à 1.

Fonction du mode pas à pas

```
void Mode_pas_pas(void)
{
    T1CONbits.TON = 1;                  //Activation du Timer 1
    T2CONbits.TON = 0;                  //Desactivation du Timer 2
    IEC0bits.T2IE = 0;                  //Desactivation de l'interruption du Timer 2

    //Desactivation de la detection de Va Vb Vc
    IEC0bits.IC1IE = 0;
    IEC0bits.IC2IE = 0;
    IEC1bits.IC7IE = 0;

    if (vitesse == 0)
    {
        OVDCON = 0;
    }else{
        //Actualisation en debut de tour de la periode et du DC
        PR1 = 13035/vitesse;
        int DC = PTPER*vitesse/255;
        PDC1 = DC;
        PDC2 = DC;
        PDC3 = DC;


        for (secteur=0; secteur<6; secteur++)   //Balayage des secteurs de 0 a 5
        {
            while(!IFS0bits.T1IF){}         //Tempo timer1
            IFS0bits.T1IF = 0;
            alim();                  //MaJ du secteur dans lequel on se trouve
        }
    }
}
```

3.6. Mode autopiloté

Le mode autopiloté sera activé lorsque la consigne de vitesse de rotation dépassera les 4000tr/min.

La période électrique est toujours divisée en 6 secteurs, mais le passage d'un secteur à l'autre est conditionné au passage à zéro des tensions V_a, V_b et V_c comme résumé si dessous :

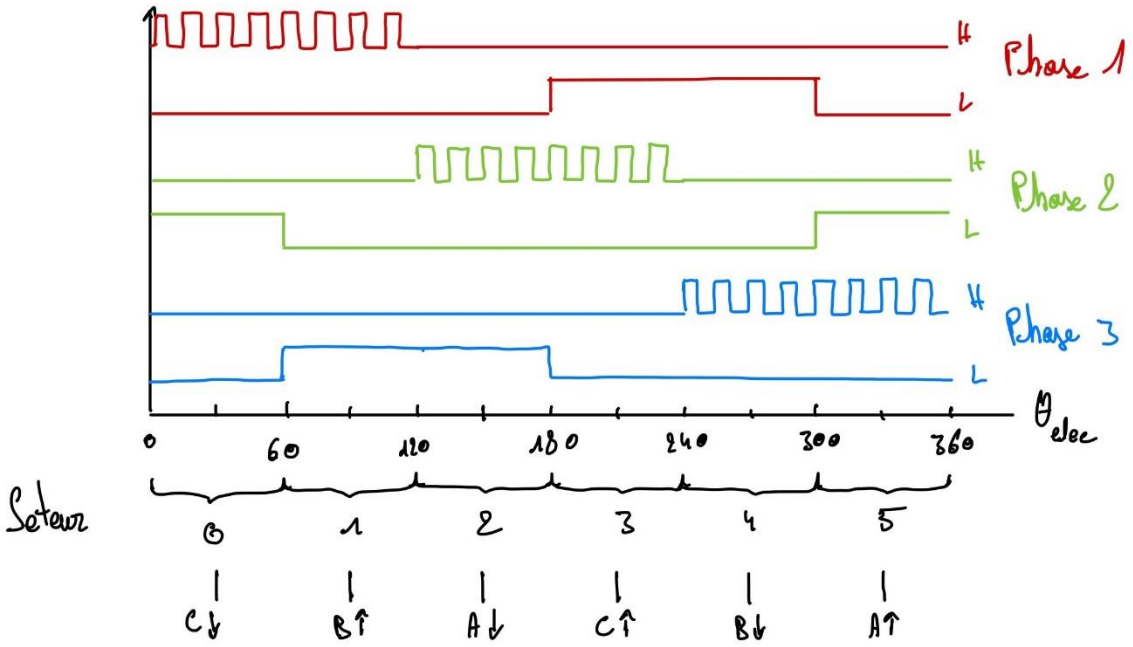


Figure 43 : Découpage des secteurs électrique

S'il n'y a pas de déphasage les fronts montants et descendant sur Va, Vb et Vc se font au milieu du secteur, cela signifie qu'à la détection d'un front il faut attendre au moins 30° avant de passer au secteur suivant. Mais avec le déphasage il suffira d'attendre $\theta = 30° - déphasage$, dans notre cas il est de 10°. Cette temporisation est effectuée avec le Timer 2 :

$$Per_{T2} = T \cdot \frac{F_{CY}}{PS}$$

Avec T le temps à attendre, F_{CY} la fréquence cyclique et PS le prescaller (=1)

Le temps d'attente est égal au nombre de degrés électrique à attente θ multiplié par la durée d'un degré T_θ :

$$T = \theta \cdot T_\theta$$

La durée d'un degré électrique vaut :

$$T_\theta = \frac{1}{F_{elec} \cdot 360} = \frac{1}{\frac{q \cdot vitesse \cdot Nb\ poles}{60} \cdot 360} = \frac{1}{30 \cdot q \cdot vitesse}$$

Avec $vitesse$ la vitesse de consigne entre 0 et 255, q est la résolution de 35.3 tr/min et le nombre de pair de pôles du moteur (=5)

Au final on obtient :

$$Per_{T2} = \theta \cdot \frac{F_{CY}}{30 \cdot 35.3 \cdot vitesse}$$

Fonction d'initialisation du Timer 2 :

```
//Initialisation Timer 2 pour le Mode Autopilote
    T2CONbits.TCS = 0;                  //Selection de l'hologe
    T2CONbits.TGATE = 0;

    T2CONbits.TCKPS = 0b00;             //Predivision par 1 (=0b00)
    PR2 = 10000;                        //Periode

    T2CONbits.T32 = 0;                  //Timer 2 et Timer 3 independants
    T2CONbits.TON = 0;                  //Desactivation

    IFS0bits.T2IF = 0;                  //Flag a 0
    IEC0bits.T2IE = 0;                  //Desactivation de l'interruption
```

Interruption du Timer 2 :

```
void __attribute__((__interrupt__, __auto_psv__)) _T2Interrupt(void)
{
    IFS0bits.T2IF = 0;          //Flag a 0
    alim();                     //Mise a jour de OVDCON
}
```

La détection des fronts montants et descendants pour le mode autopiloté ce fait avec le module input capture qui à chaque fois qu'un front est détecté le programme va entrer dans une interruption, dans laquelle le Timer 2 est déclanché.

Fonctions d'initialisation :

```
void Init_IC1(void)                  //Initialisation de la detection du passage a 0
de Va
{
    IEC0bits.IC1IE = 0;             //Desactiver l'interruption IC1
    IFS0bits.IC1IF = 0;             //flag a 0
    IPC0bits.IC1IP = 7;             //priorité = 7
    IC1CONbits.ICM = 0b001;         //Detection des front montant et descendant
}

void Init_IC2(void)                  //Initialisation de la detection du passage a 0
de Vb
{
    IEC0bits.IC2IE = 0;             //Desactiver l'interruption IC2
    IFS0bits.IC2IF = 0;             //flag a 0
    IPC1bits.IC2IP = 7;             //priorité = 7
    IC2CONbits.ICM = 0b001;         //Detection des front montant et descendant
}

void Init_IC7(void)                  //Initialisation de la detection du passage a 0
de Vc
{
    IEC1bits.IC7IE = 0;             //Desactiver l'interruption IC7
    IFS1bits.IC7IF = 0;             //flag a 0
    IPC4bits.IC7IP = 7;             //priorité = 7
    IC7CONbits.ICM = 0b001;         //Detection des front montant et descendant
}
```

Fonctions d'interruption :

```
void __attribute__((interrupt, no_auto_psv )) _IC1Interrupt(void)
{
    if ((PORTDbits.RD0 == 0) && (secteur == 2))              //Front descendant
    {
        secteur = 3;            //Passage au secteur suivant
        TMR2 = 0;               //quand le retard sera combler

    }else if ((PORTDbits.RD0 == 1) && (secteur == 5))        //Front Montant
    {
        secteur = 0;            //Passage au secteur suivant
        TMR2 = 0;               //quand le retard sera combler
    }
    IFS0bits.IC1IF = 0;         //on abaisse le flag
}

void __attribute__((interrupt, no_auto_psv )) _IC2Interrupt(void)
{
    if ((PORTDbits.RD1 == 0) && (secteur == 4))              //Front descendant
    {
        secteur = 5;            //Passage au secteur suivant
        TMR2 = 0;               //quand le retard sera combler

    }else if ((PORTDbits.RD1 == 1) && (secteur == 1))        //Front Montant
    {
        secteur = 2;            //Passage au secteur suivant
        TMR2 = 0;               //quand le retard sera combler
    }
    IFS0bits.IC2IF = 0;         //on abaisse le flag
}

void __attribute__((interrupt, no_auto_psv )) _IC7Interrupt(void)
{
    if ((PORTBbits.RB4 == 0) && (secteur == 0))              //Front descendant
    {
        secteur = 1;            //Passage au secteur suivant
        TMR2 = 0;               //quand le retard sera combler

    }else if ((PORTBbits.RB4 == 1) && (secteur == 3))        //Front Montant
    {
        secteur = 4;            //Passage au secteur suivant
        TMR2 = 0;               //quand le retard sera combler
    }
    IFS1bits.IC7IF = 0;         //on abaisse le flag
}
```

Fonction du mode autopilotage :

```
void Mode_autopilote(void)
{
    T1CONbits.TON = 0;                  //Desactivation du Timer 1
    T2CONbits.TON = 1;                  //Activation du Timer 2
    IEC0bits.T2IE = 1;                  //Activation de l'interruption du Timer 2

    //Actualisation en debut de tour de la periode et du DC
    int retard = 20;                    //Retard du au filtage (en degres)
    PR2 = FCY/1059*retard/vitesse;
    int DC = PTPER*vitesse/255;
    PDC1 = DC;
    PDC2 = DC;
    PDC3 = DC;

    //Activation de la detection de Va Vb Vc
    IEC0bits.IC1IE = 1;
    IEC0bits.IC2IE = 1;
    IEC1bits.IC7IE = 1;
}
```

4. Conclusion de la partie électronique numérique

En conclusion, la programmation n'a pas été quelque chose de simple entre débogage et documentation technique pas toujours claire, nous sommes néanmoins parvenus à un résultat fonctionnel au moins en théorie, comme l'atteste cette vidéo : https://drive.google.com/file/d/1k8cJqvJhrkV4Fhq3UR0ANMaXSCO_8ENK/view?usp=sharing.

Des ajustements sont surement à faire pour que cela devienne réellement fonctionnel. Des améliorations sont possibles pour peut-être optimiser le code et ajouter des fonctions supplémentaires comme un asservissement en vitesse.

V. Modèle 3D

Comme la réalisation du projet n'est plus d'actualité, voici un model 3D avec les plans de la disposition des différentes parties sur l'aéroglisseur. Nous avons fait le choix de repartir des plans fournis en réduisant d'environ 20% la longueur de l'aéroglisseur pour respecter le cahier des charges.

Pour des raisons de simplicité nous n'avons modélisé que les pièces principales de la structure de l'aéroglisseur.

Disposition en vue 3D :

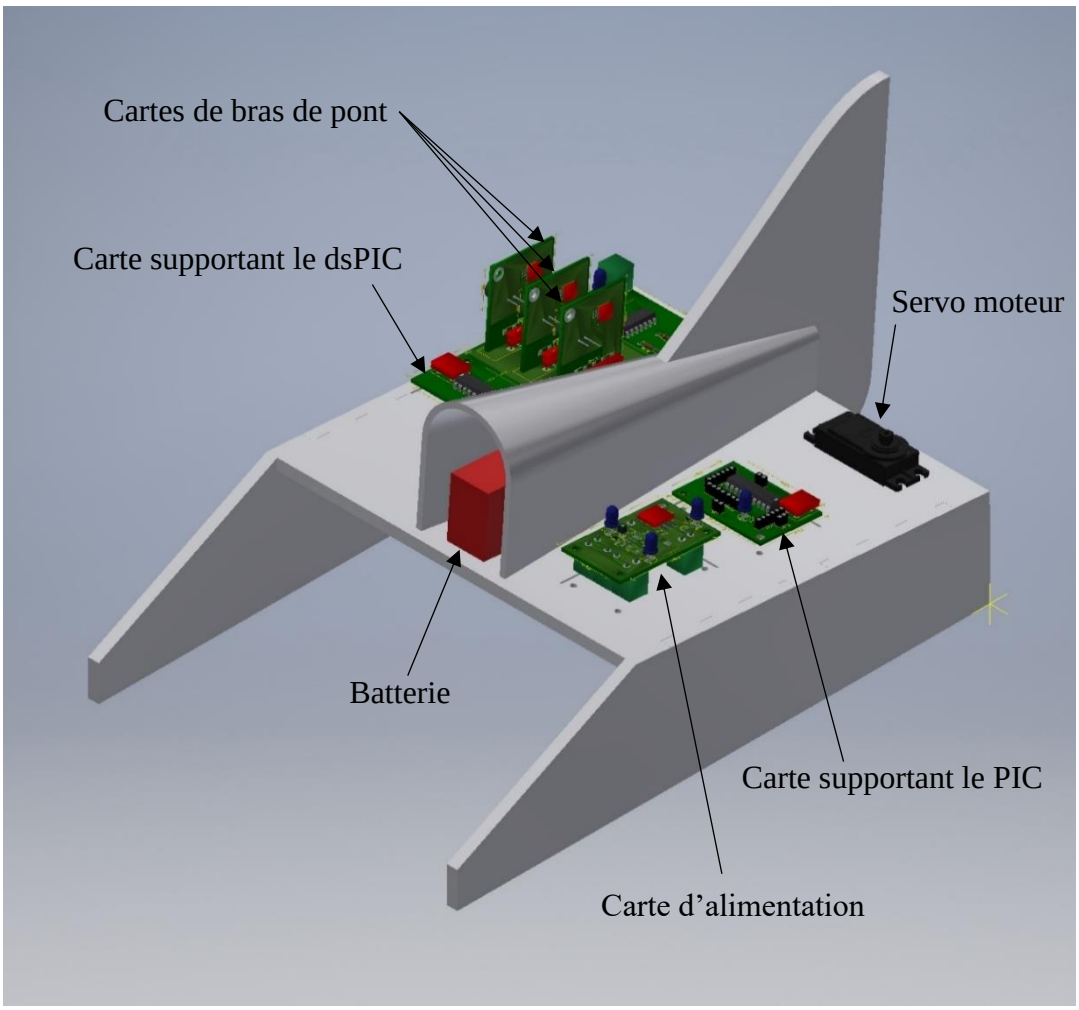


Figure 44 : Vue 3D annotée

Nous avons fait le choix de mettre de part et d'autre les cartes ainsi que le servo moteur et de mettre la batterie sous la voute centrale car c'est l'élément le plus lourd, il était donc important de la centré pour ne pas déséquilibré l'aéroglisseur, par chance cela rentre tout juste.

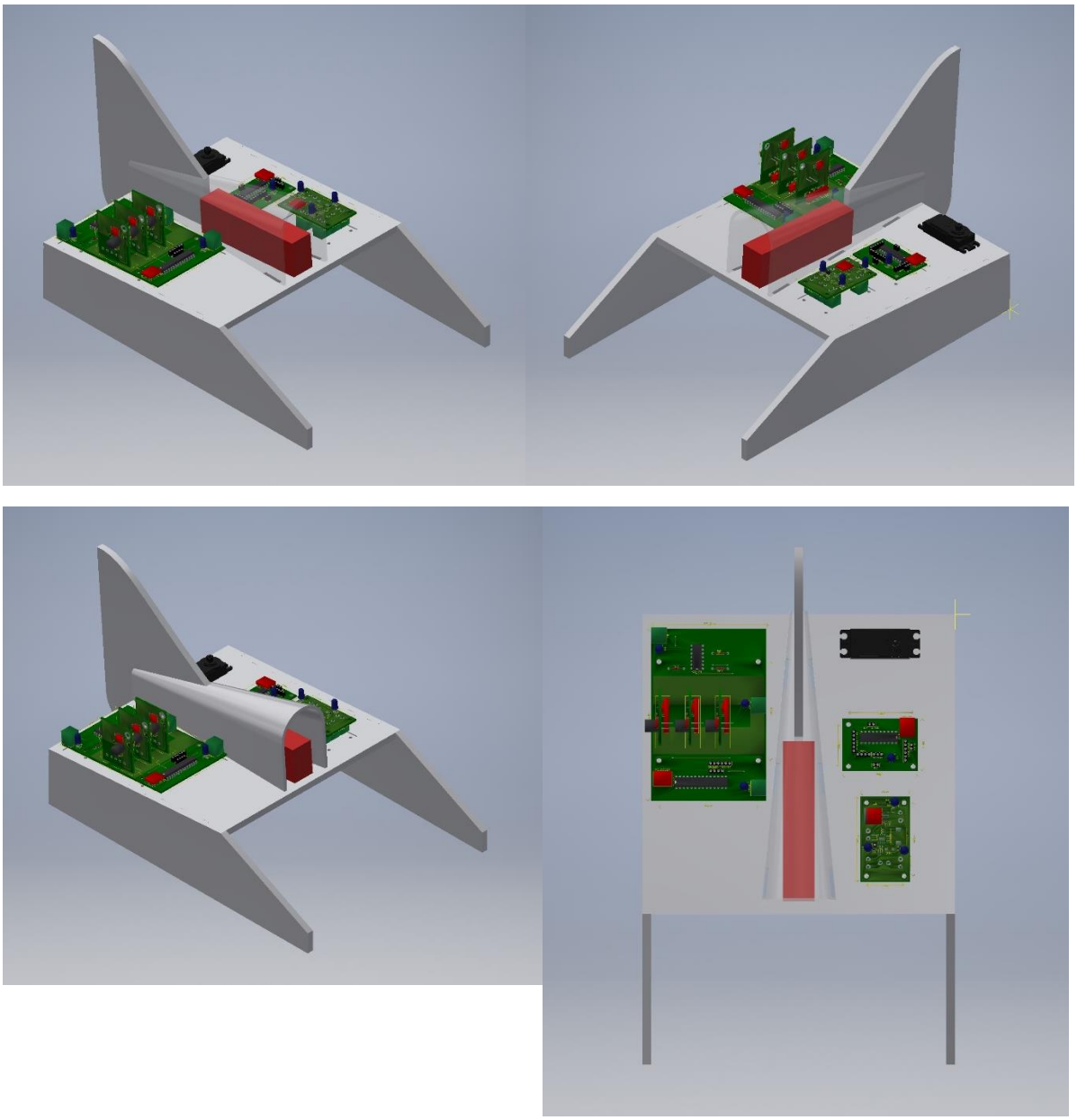
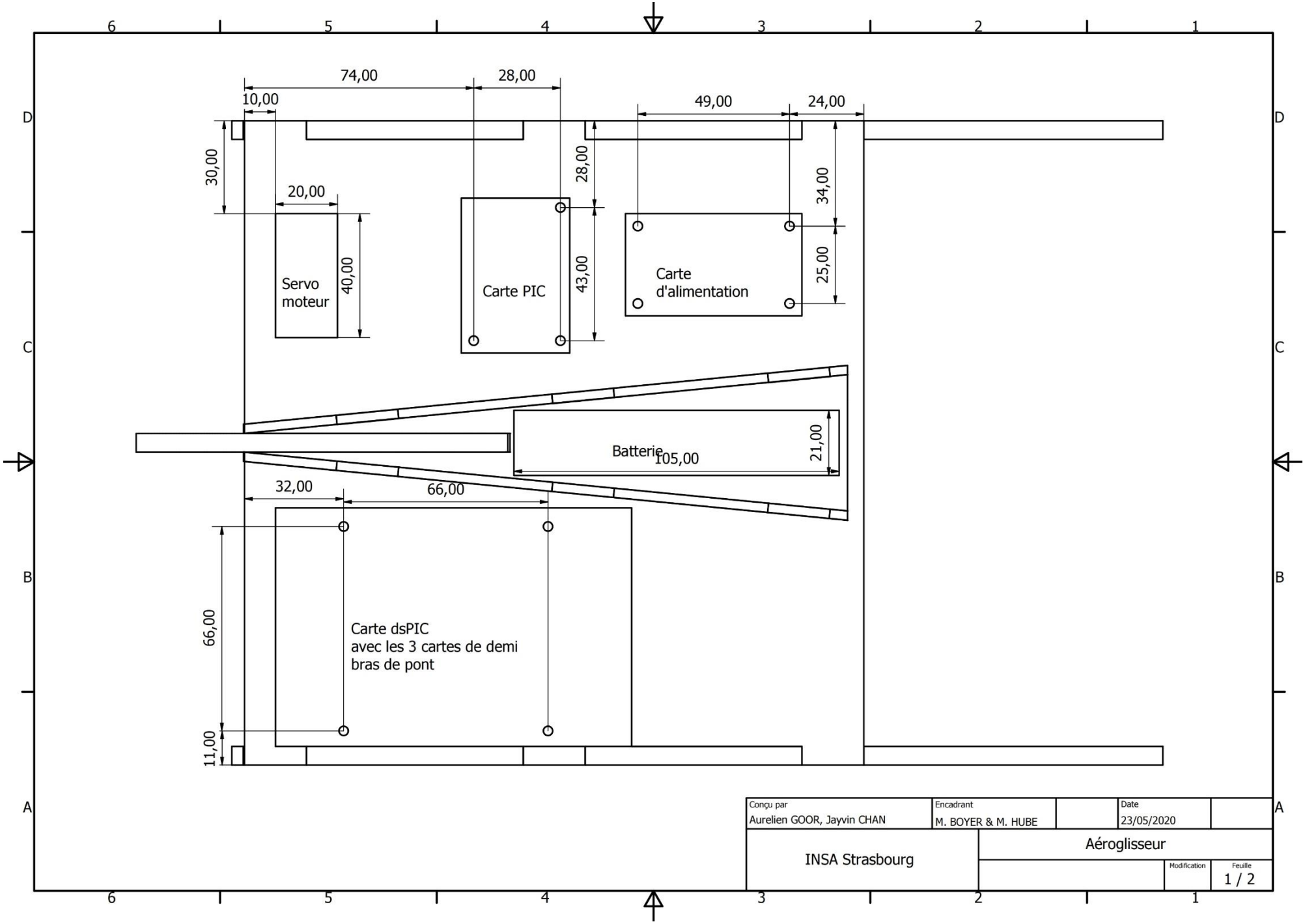
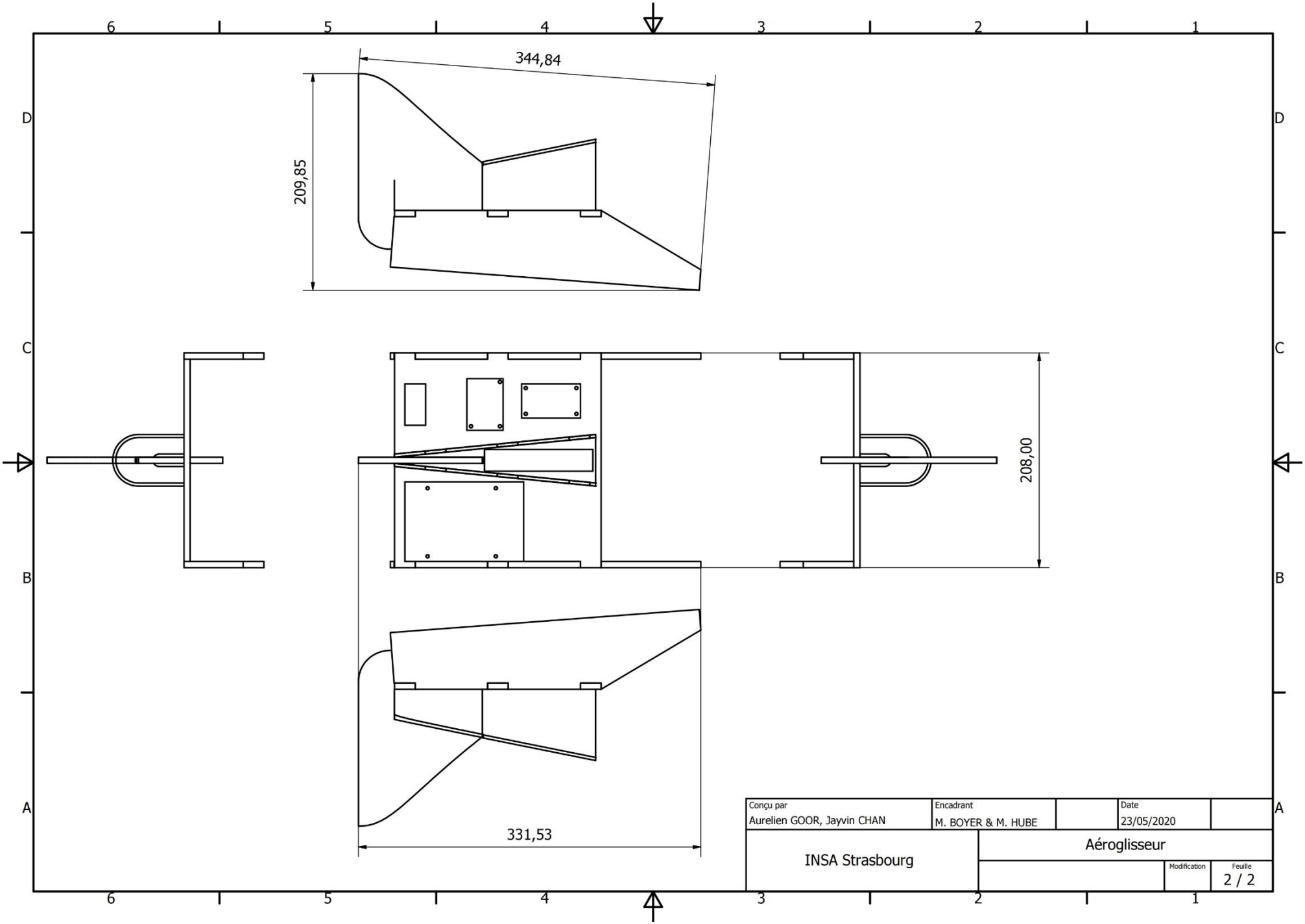


Figure 45 : Vues 3D


74,00
28,00
10,00
49,00
24,00
30,00
28,00
34,00
20,00
Servo
moteur
40,00
Carte PIC
43,00
Carte
d'alimentation
25,00
Batterie
105,00
21,00
32,00
66,00
66,00
Carte dsPIC
avec les 3 cartes de demi
bras de pont
11,00
Conçu par
Aurelien GOOR, Jayvin CHAN
Encadrant
M. BOYER & M. HUBE
Date
23/05/2020
INSA Strasbourg
Aéroglisseur
Modification
Feuille
1 / 2


344,84
209,85
208,00
331,53
Conçu par
Aurelien GOOR, Jayvin CHAN
Encadrant
M. BOYER & M. HUBE
Date
23/05/2020
INSA Strasbourg
Aéroglisseur
Modification
Feuille
2 / 2

VI. Conclusion

Ce projet sur l'aéroglisseur commandé par Bluetooth est très intéressant. Il nous a permis de balayer plusieurs domaines, et de mettre en application la théorie enseignée.

Au niveau technique, nous avons eu à refaire fréquemment le choix des composants, le routage des cartes, ainsi que le programme afin de les améliorer. Cela nous a permis de cibler nos faiblesses et ainsi de mieux comprendre le fonctionnement. Les logiciels de simulations nous ont été d'une grande utilité.

Suite à notre travail, nous espérons qu'après impression des cartes, soudure et téléversement des programmes, l'aéroglisseur puisse fonctionner.

Avec les étapes du projet et en l'exploitant pour la communication et l'anglais appliqué au projet, on se rapproche de plus en plus des attentes du métier d'ingénieur.

Nous remercions les professeurs pour leur accompagnement malgré la situation dans laquelle s'est déroulée le travail.

Table des Illustrations