



4nov



Responsable pédagogique :
Sagonéro Cyril
Aubry Pierre

Intervenant :
Oukrat Rémi
Nazim Zakari Saibi

Véhicule autonome et interactif

CAPTATION DES OBSTACLES A 360°

Table des matières

Introduction	2
Planning des tâches	3
Previsonnel	3
Reel.....	3
Cahier des charges	4
Fonction d'usage	5
Recherche des solutions techniques	5
Capteur	5
<i>Ultrason</i>	5
<i>Infrarouge</i>	11
<i>Laser</i>	13
Asservissement Moteur	16
<i>Etude du HS – 311</i>	17
Remise a zero	17
<i>Capteur capacitif</i>	17
<i>Capteur infrarouge</i>	18
<i>Capteur à effet Hall</i>	18
Carte electronique.....	19
<i>Schéma fonctionnel Niveau 1</i>	19
<i>Schéma fonctionnel Niveau 2</i>	20
<i>Schema structurel</i>	22
<i>Calcul des résistances</i>	23
<i>Filtrage des tensions d'alimentation</i>	24
<i>Tests, problèmes et solutions</i>	27
<i>Nomenclature</i>	29
<i>Plan de connexion</i>	30
Typons	30
<i>Bottom layer</i>	30
<i>Top layer</i>	31
<i>Plan de perçage</i>	31
<i>Schéma d'implantation</i>	32
Support physique	33
<i>Problème et solution :</i>	36
Programme	37

Mise en œuvre du VL53L0X.....	37
<i>Etude de fonction I2C</i>	37
<i>Initialisation et utilisation</i>	41
Mise en œuvre du servomoteur	45
<i>Etude de fonction PWM</i>	45
<i>Etude de fonction Timer</i>	46
<i>Asservissement de position</i>	48
<i>Initialisation position 0</i>	50
<i>Arrêt du servomoteur</i>	50
<i>Problèmes et solutions</i>	51
Conclusion.....	51

Introduction

Le projet dont nous sommes responsables est un sujet se portant sur un robot interactif, celui-ci devant répondre à un certain nombre de contraintes. Le système doit pouvoir être contrôlé sur 100m à champ dégagé pendant au moins 40 min. Il doit également être autonome si désiré par l'utilisateur. Dans ce cas, il suivra une ligne noire tracée sur le sol dont les dimensions sont données et bien précises et détecter les obstacles susceptibles de le gêner dans sa mission. Le projet a donc été décomposé en plusieurs fonctions principales tel que Communiquer et Capter.

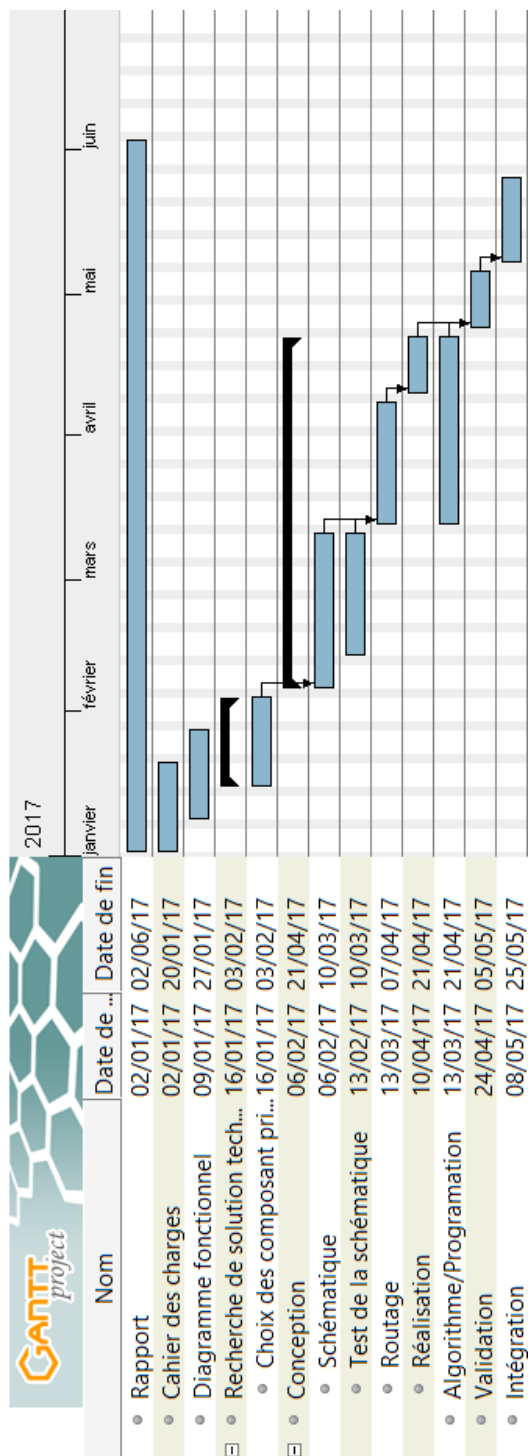
Me chargeant de cette dernière fonction, ma mission fut d'élaborer un module permettant de rendre possible la captation des obstacles. Dans un premier temps, j'ai construit un planning prévisionnel afin d'obtenir une vue d'ensemble sur le travail attendu. Suite à ce planning, je me suis posé différentes questions afin de trouver des solutions rentrant dans les limites du cahier des charges.

Avant d'entrer dans le projet en lui-même, différents points doivent être éclaircis afin de déterminer les cas qui seront traités et ceux volontairement écartés. Le cahier des charges stipule certaines obligations à respecter, cependant, il reste vague à certains endroits. Par exemple il est clairement mentionné une captation à 360° mais aucun détail n'est fourni en ce qui concerne les dimensions minimums des objets. Il est aussi mentionné une captation éventuelle de sa géolocalisation.

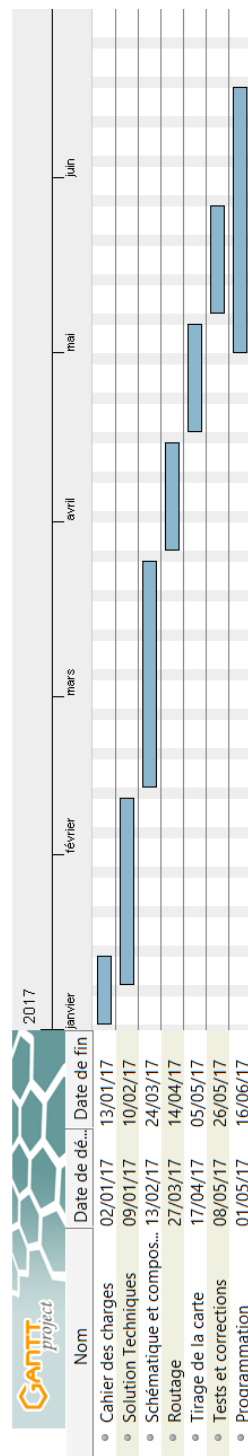
Nous verrons par la suite que ces détails seront chiffrés par un nouveau cahier des charges dit personnel, dans lequel je compléterai les données de celui qui nous a été fournis grâce à des recherches notamment en matière de coût, de dimension, de consommation, etc. Pour enfin conclure sur une solution technique définitive et aboutir à la concrétisation de ce projet avec une intégration du module sur le robot final.

Planning des tâches

PREVISIONNEL



REEL



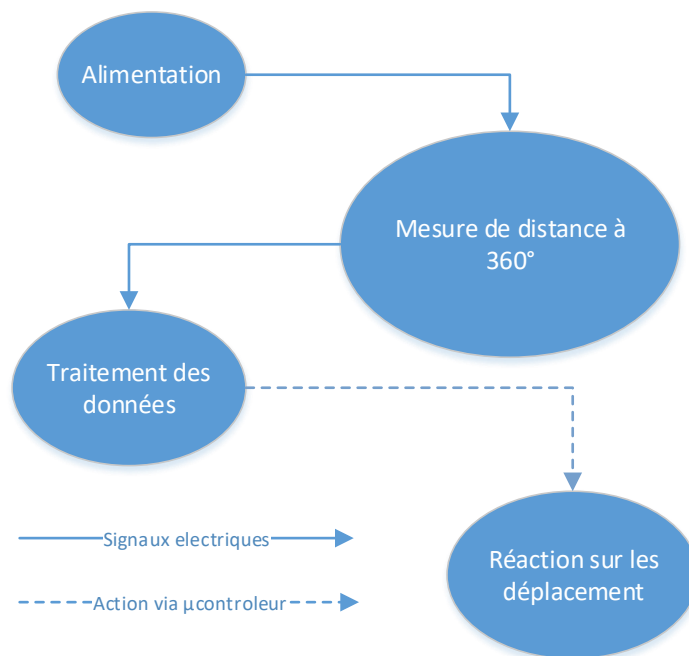
On peut constater un retard conséquent par rapport au diagramme prévisionnel notamment dû au fait que j'ai exécuté les tâches une par une au début, là où j'aurais pu en effectuer plusieurs de façon parallèle. J'ai donc accumulé du retard étapes par étapes. De plus, j'ai sous-estimé certaines tâches qui m'ont finalement demandé un temps conséquent comme l'étude des solutions techniques.

De plus certaines étapes ont été totalement confondu et ont découlé l'une de l'autre, il m'a ainsi paru plus sensé de les réunir.

Cahier des charges

Critère	Cahier des charges initial	Cahier des charges modifié
Température de fonctionnement	NC	0 – 50°C
Gamme de microcontrôleur	CISC	CISC
Prix de ma carte	<300€	<50€
Tension d'alimentation	NC	≤5V
Consommation de ma carte	NC	≤300 mA
Etanchéité de la carte	NC	Non
Dimension de la carte maximum	NC	75*75*15 mm
Distance max de détection	NC	20 cm à partir du bord du Rover
Pas de détection de distance	NC	Entre 0.5 mm et 5mm
Matières à détecter	NC	Toutes
Couleurs à détecter	NC	Toutes
Taille minimum de l'obstacle	NC	5cm
Largeur min de l'obstacle	NC	2cm
Fréquence de captation	NC	>20 Hz
Poids total	NC	<200g

FONCTION D'USAGE



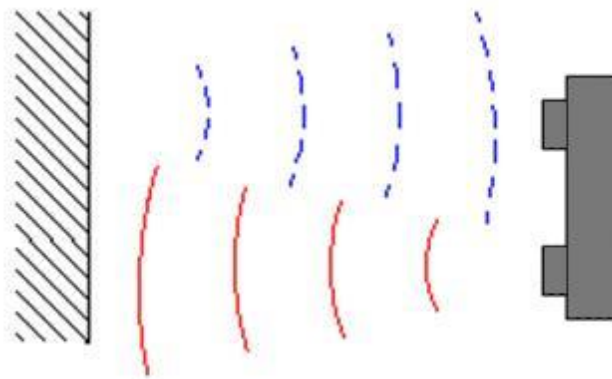
Recherche des solutions techniques

CAPTEUR

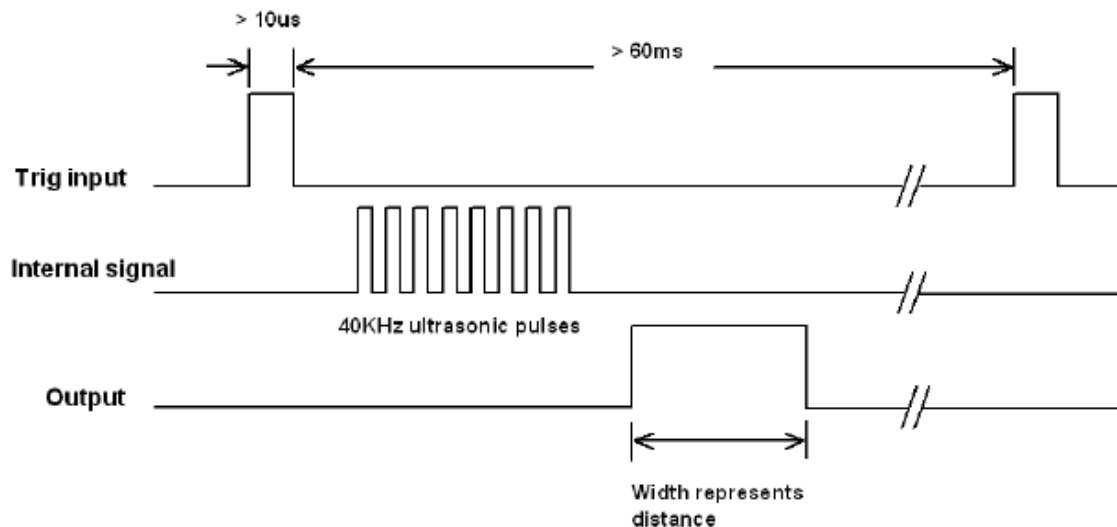
Dans cette partie nous allons étudier quelles solutions existent afin de solutionner la caractéristique principale du module : capter un obstacle. Pour chaque technologie étudiée, il sera estimé un prix, et une solution possiblement envisageable sera développée.

Ultrason

La captation via ondes ultrason, largement utilisé dans la robotique amateur de par son bas prix, permet une captation d'obstacle jusqu'à 4m pour la plupart des modules. Voici le fonctionnement :



Lorsque le module envoie des salves d'ultrason à 40 kHz, ces dernières se retrouvent propagées dans l'air, couvrant un angle relativement directif dépendant des modèles. Ces ondes rebondissant sur l'obstacle placé devant, retournent vers le module intégrant, en plus de l'émetteur, un capteur ultrason. Le temps de propagation de l'onde sonore étant connu (340 m/s variant en fonction de la température et de la pression) il est possible de calculer la distance séparant le module et l'obstacle.

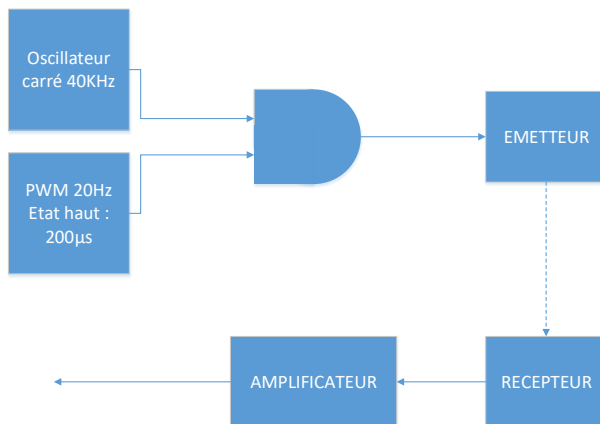


Ainsi lorsque la broche Trigger est enclenché, une salve de d'ultrason est envoyée. Le module commence donc le calcul de temps grâce à un μ contrôleur intégré. Sur ma grande majorité des modules, la sortie est représentée grâce à une largeur d'impulsion en sortie, permettant ainsi une précision qui n'est pas limitée par le module en lui-même mais par le choix de la conversion numérique. Il existe également des modules incluant des sorties analogiques ou numériques. Cependant le prix de ces ajouts fait grandement monter le prix de ces capteurs, passant alors d'une dizaine d'euro à une trentaine. Cependant, bien qu'étant bon marché, cette technologie présente certains inconvénients. Le temps entre chaque salve d'ondes ne peut se faire qu'après un délai d'au moins 50ms, afin de ne pas être perturbé par le retour de la salve précédente. Ensuite, toutes les matières ne retournent pas les ondes sonores, la mousse en particulier absorbe la totalité des ondes et rend le capteur totalement aveugle.

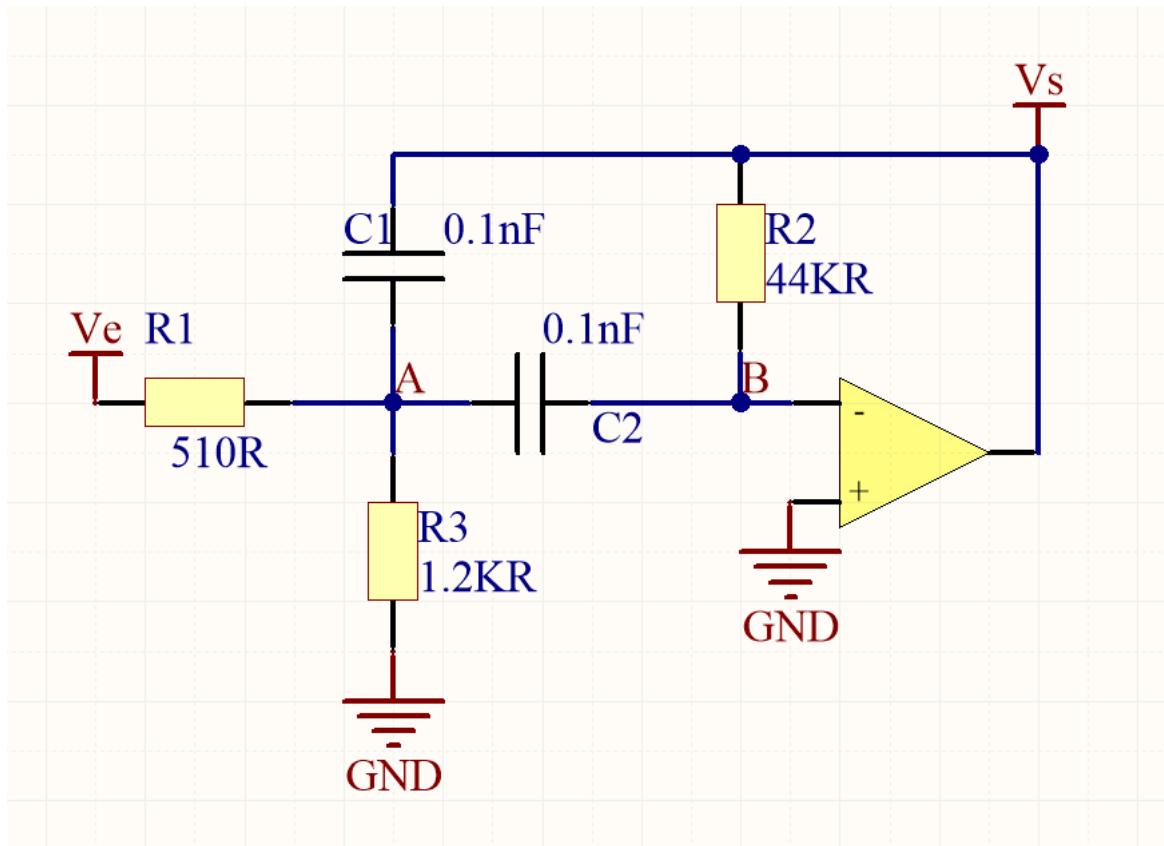
	Avantage	Inconvénient
<i>Prix</i>	10 – 20€	
<i>Couleur détectable</i>	Toutes	
<i>Matière détectable</i>		Uniquement les matières rigides
<i>Pas</i>	Dépend de la mise en place	
<i>Fréquence d'utilisation</i>		20 Hz
<i>Directivité</i>	30-60°	

La mise en place de ce type de capteur étant relativement simple, il serait tentant de l'utiliser en tant que détecteur d'obstacle. Cependant, sa sensibilité pose un problème. En effet, pour obtenir une couverture à 360°, en supposant qu'un capteur couvre au maximum 60°, nous aurions besoin de 6 capteurs, soit un minimum de 60€, uniquement pour les capteurs, d'autant plus que les modules les moins chères de ces gammes sont moins fiables. Nous arrivons donc à une carte tournant au minimum à 70€, ce qui est bien trop élevé compte tenu du budget du Rover s'élevant à 300€, cela représentera environ 23% du budget, et nous parlons bien ici d'un minimum.

J'ai également émis l'hypothèse de fabriquer un module par moi-même, grâce aux émetteurs / récepteurs présents à l'école.



De cette façon, l'émetteur envoie des salves d'ultrason à 40KHz pendant 200µs, toutes les 50ms. Le délai entre chaque salve peut paraître long mais il est nécessaire afin que les ondes envoyées précédemment ne se répercutent pas dans le récepteur après un autre coup d'horloge, ce qui fausserait le résultat. Cependant, j'ai constaté qu'en sortie, le signal était entièrement bruité, et totalement illisible. J'ai donc ajouté un filtre du second ordre en sortie de l'amplificateur.



$$C1 = C2 = C$$

$$i_{C2} + i_{R2} = 0$$

$$A = \frac{\frac{V_e}{R1} + V_s j\omega + B j\omega + \frac{0}{R3}}{\frac{1}{R1} + \frac{1}{R3} + j\omega + j\omega} = \frac{V_e R3 + V_s j R1 R3 C \omega}{R3 + R1 + 2j R1 R3 C \omega}$$

$$\rightarrow A(R3 + R1 + 2j R1 R3 C \omega) = V_e R3 + V_s R1 R3 C \omega$$

$$\rightarrow \frac{A - B}{\frac{1}{j\omega}} + \frac{V_s - B}{R2} = 0 \rightarrow A j\omega + \frac{V_s}{R2} = 0$$

$$\rightarrow A j\omega = -\frac{V_s}{R2} \rightarrow A = -\frac{V_s}{j R2 C \omega}$$

$$\rightarrow -\frac{V_s}{j R2 C \omega} (R3 + R1 + j R1 R3 C \omega) = V_e R3 + V_s j R1 R3 C \omega$$

$$\rightarrow -V_s \left(\frac{R3 + R1 + 2j R1 R3 C \omega + (j\omega)^2 R1 R2 R3 C^2}{j R2 C \omega} \right) = V_e$$

$$\frac{V_s}{V_e} = T = -\frac{(j\omega) R2 R3 C}{(j\omega)^2 R1 R2 R3 C^2 + (j\omega) 2 R1 R3 C + (R3 + R1)}$$

$$= -\frac{2}{2 R1} * \frac{(j\omega) \frac{R2 R3 C}{R3 + R1}}{(j\omega)^2 \frac{R1 R2 R3 C^2}{R3 + R1} + (j\omega) \frac{2 R1 R3 C}{R3 + R1} + 1}$$

$$\equiv GM \frac{2mj \frac{\omega}{\omega_0}}{\left(j \frac{\omega}{\omega_0}\right)^2 + 2mj \frac{\omega}{\omega_0} + 1}$$

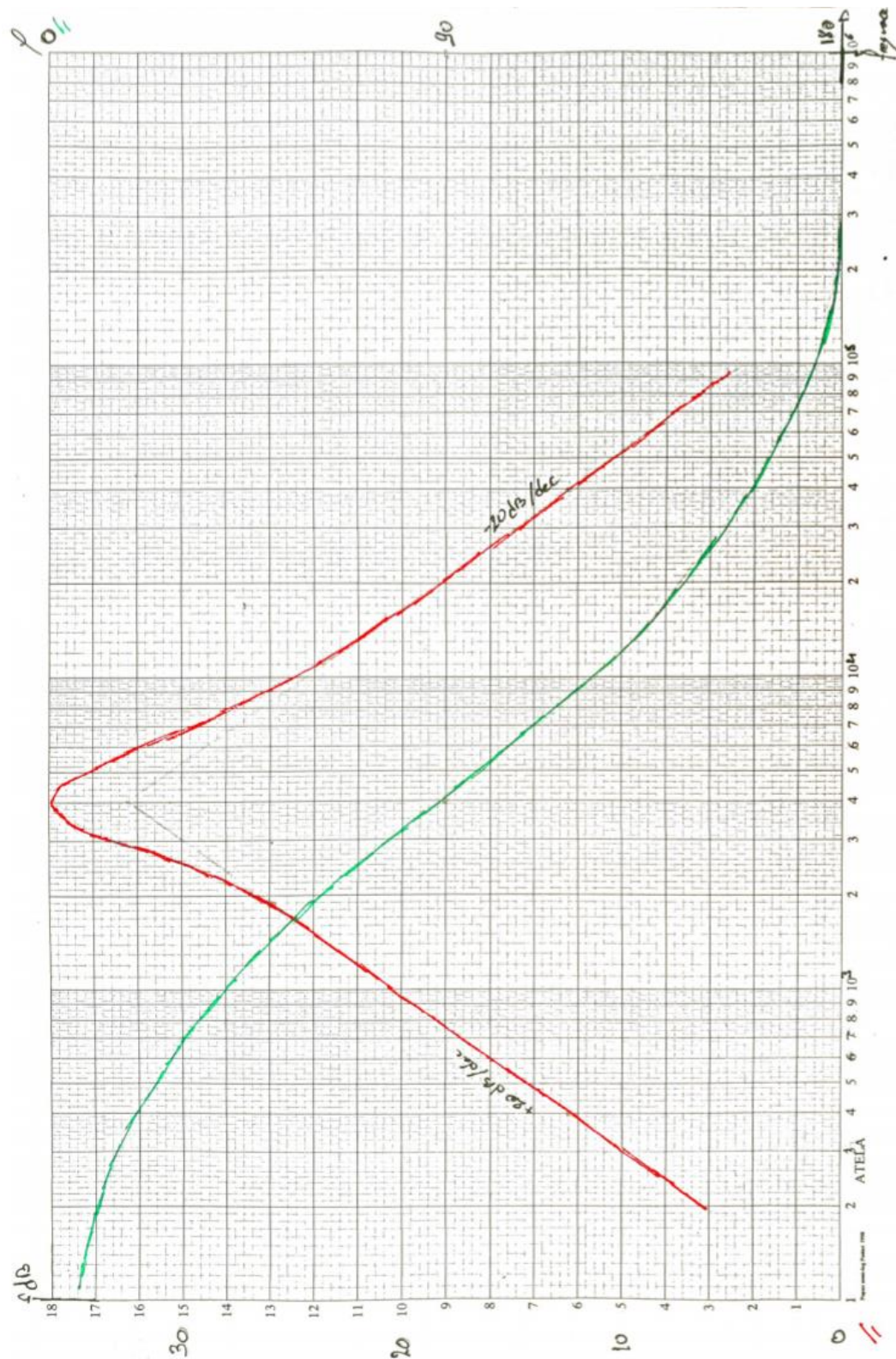
Filtre passe bande du second ordre

$$\frac{1}{(\omega_0)^2} = \frac{R1 R2 R3 C^2}{R3 + R1} \rightarrow \omega_0 = \frac{1}{C} \sqrt{\frac{R3 + R1}{R1 R2 R3}} = 251997.369$$

$$f_0 = \frac{\omega_0}{2\pi} = 40106 \text{ Hz}$$

$$\frac{2m}{\omega_0} = \frac{2 R1 R3 C}{R3 + R1} \rightarrow m = \frac{R1 R3}{\sqrt{(R3 + R1) R1 R2 R3}} = 0.0901$$

$$Gm = -\frac{R2}{2 R1} = -43.13$$



Malgré ce filtre passe-bande sélectif, un bruit bien trop important est présent, de plus, l'amplitude du signal varie grandement en fonction de la distance de la « cible », ce qui rend la donnée intraitable.

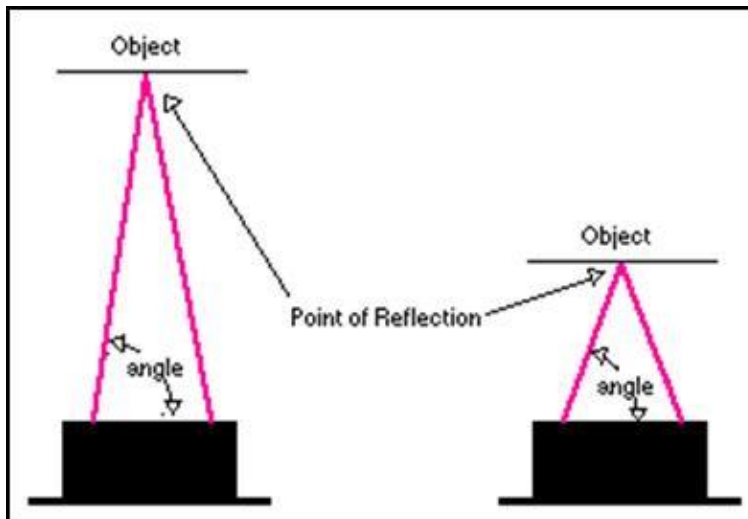
Une des solutions pour réduire le nombre de capteurs serait de monter un voire deux modules sur un servomoteur, permettant ainsi une couverture bien plus large pour une nombre de modules bien réduit. Le budget descendrait donc à 26€, soit 8% du budget. Bien que cette méthode puisse sembler intéressante, elle est inapplicable avec ce type de capteur. En effet, nous parlions plus haut d'une hypersensibilité au bruit, et notamment aux vibrations. Le capteur d'ultrason embarqué sur le module est calibré pour détecter une onde largement affaiblie par son trajet dans l'air, et un moteur produit largement assez de vibration pour perturber totalement un tel dispositif.

Le principe d'asservissement en position d'un moteur afin de limiter le nombre de capteur reste donc envisageable, mais pas avec ce type de capteur. On préférera se diriger vers un capteur optique.

Infrarouge

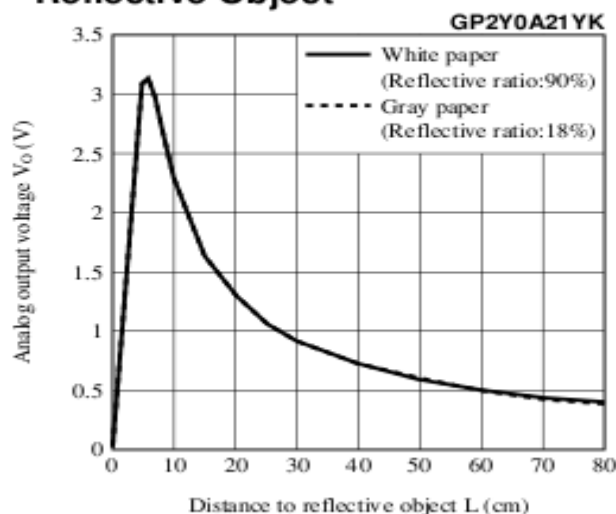
Il existe deux types de capteur infrarouge. Le premier est dit Tout ou Rien, tout simplement car il délivrera une tension (ou non) lorsque la lumière infrarouge émise par la diode embarquée sera réfléchiée et retournée au phototransistor, lui aussi embarqué. Ce type de capteur, très pratique de par sa simplicité, n'est pas ce que nous cherchons ici. En supposant que cette technologie soit choisie, le rover n'aura alors aucun moyen de savoir à quelle distance de lui se trouve l'obstacle, et les manœuvres d'évitements deviendraient alors impossibles.

Heureusement un deuxième type de capteur existe. Celle-ci donne un retour de distance via une sortie analogique.



Comme montré sur l'illustration ci-contre, le capteur émet une lumière infrarouge et capte le retour de jet lumineux rebondissant contre un obstacle. En calculant l'angle un retour de distance devient possible. Mais ce type de capteur possède deux gros désavantages : il est extrêmement directif et le résultat se retrouve brouillé par un déplacement de l'obstacle, et le retour analogique n'est pas linéaire.

Fig.5 Analog Output Voltage vs. Distance to Reflective Object



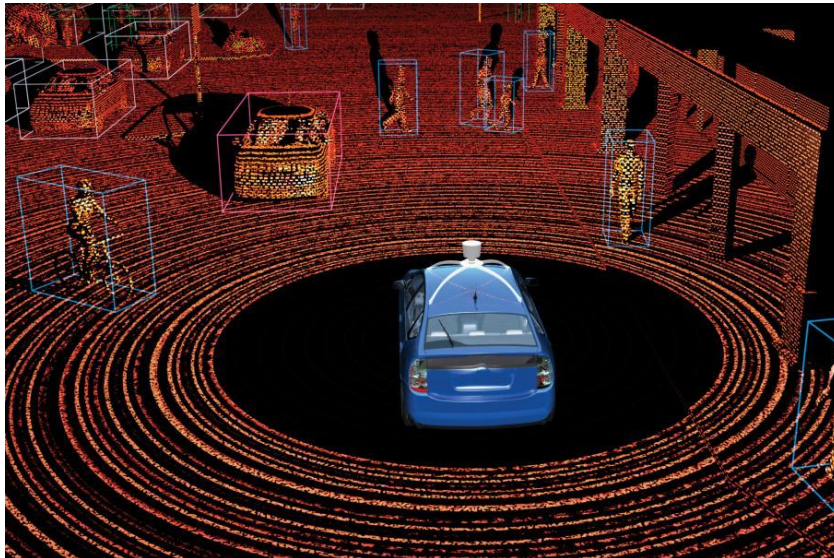
Ainsi, il est impossible d'établir une différence entre un objet situé très près d'un objet situé loin. Mais on peut aisément contourner ce problème en plaçant le capteur au centre du Rover et ainsi rendre inutile les 5 ou 10 premiers centimètres.

	Avantage	Inconvénient
<i>Prix</i>	10 – 20€	
<i>Couleur détectable</i>	Toutes	
<i>Matière détectable</i>	Toutes	
<i>Pas</i>	Dépend de la mise en place	
<i>Fréquence d'utilisation</i>		23Hz
<i>Directivité</i>		5°

Ce type de capteur est donc à proscrire dans ce cas, en vue du déplacement du véhicule. Il sera brouillé en permanence, et encore plus monté sur un servomoteur.

Laser

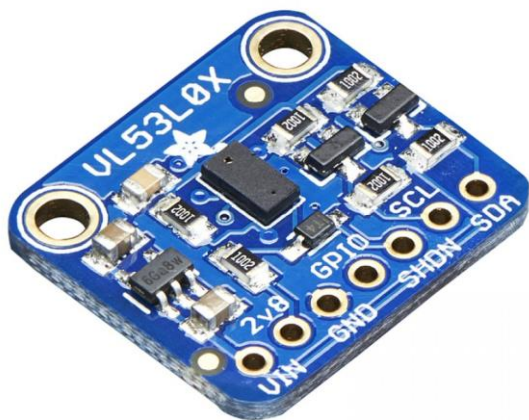
La technologie laser utilise principalement le même principe que l'infrarouge, à savoir qu'un rayon laser est émis, puis capté et son déphasage est ensuite calculé, donnant ainsi une indication de distance. Il a l'avantage d'avoir un temps de réponse ultra rapide. C'est ainsi que sont créés les LIDAR



Un capteur de distance laser placé sur une tête rotative envoie un rayon à intervalle régulier. Les données sont stockées et cela permet de cartographier l'espace sur deux dimensions. Multiplier les rayons sur l'axe vertical suffit à ajouter une troisième dimension à la carte. Ainsi le véhicule sait en permanence tout ce qui l'entoure.

Ce type de capteur est donc particulièrement adapté à la détection d'obstacle des objets en déplacement de par leur temps de réponse. Ce sont d'ailleurs des LIDAR qui sont utilisés dans les voitures autonomes. Ce type de montage coûte au-delà des 100€, mais il existe des alternatives moins coûteuses.

Modules ToF

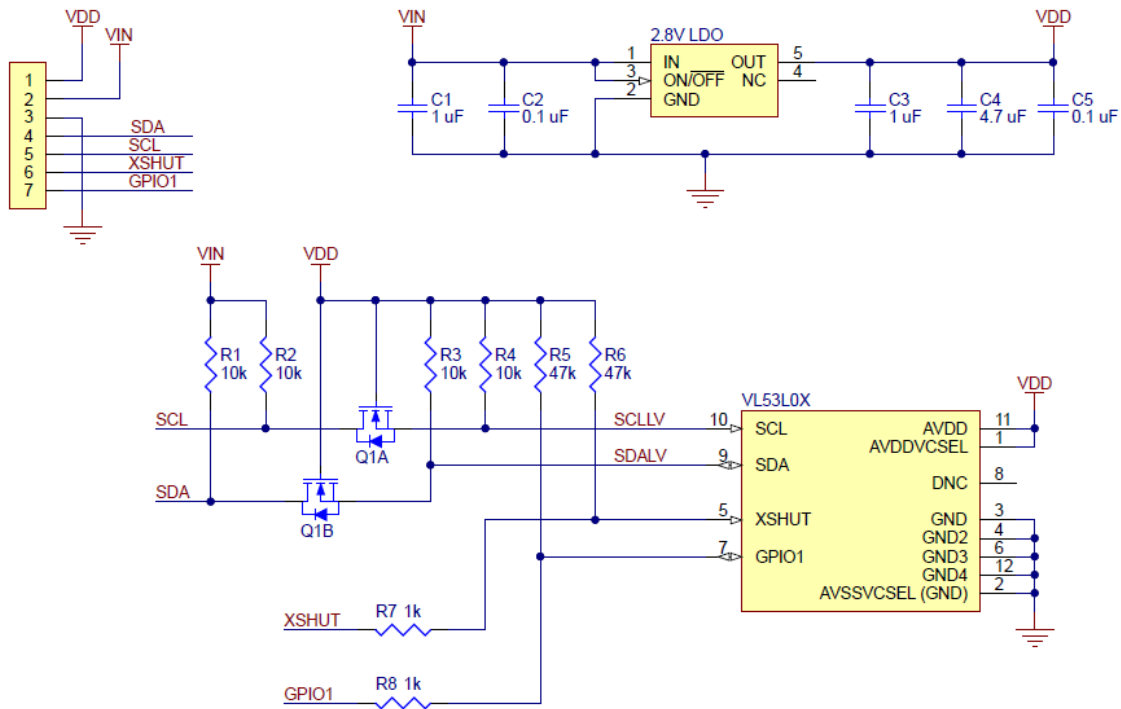


Un petit composant programmable embarquant son environnement sur une carte de la taille d’une pièce de monnaie est appelé Module Time Of Flight. Ces types de capteurs sont notamment utilisés pour détecter le temps de vol, d’où leur nom. Il en existe toute sorte, certain embarquant des caméras. Ce capteur laser possède un temps de réponse de 20m, cela peut paraître beaucoup mais contrairement aux capteurs ultrason il ne craint pas les vibrations et ne craint pas les déplacements comme le capteur IR. Sa sortie est numérique et il est même possible de choisir entre plusieurs modes tel que Longue Portée ou Haute rapidité. Toutes ces données en font un capteur idéal. Monté sur un servomoteur, il pourra assurer le rôle d’un LIDAR.

	Avantage	Inconvénient
Prix	16 €	
Couleurs détectables	Toutes	
Matières détectables	Toutes	
Pas	1 mm	
Fréquence d'utilisation	50Hz	
Directivité		< 1°

Cette solution sera donc retenue pour le capteur de distance. Son pris et sa directivité empêche d’en mettre sur chaque face du module. Il sera donc monté sur un servomoteur afin de n’en avoir qu’un à gérer et à mettre en œuvre.

Voici la schématique du module ToF de Pololu, contenant le circuit VL53L0X de la marque ST :



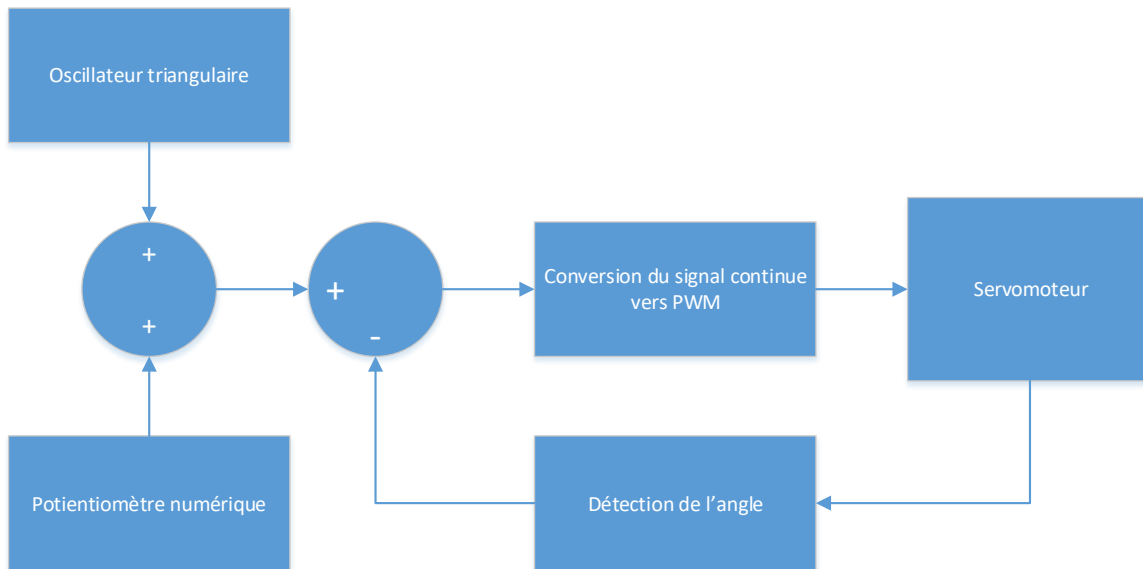
R1 ; R2 ; R3 ; R4	Résistances de tirage pour I2C
R5 ; R6	Résistance de tirage préconisé par la datasheet du VL53L0X
Q1B ; Q1A	Transistor Mosfet permettant le changement de voltage entre le maitre et l'esclave.
2.8 LDO ; C1 ; C2 ; C3 ; C4 ; C5	Régulateur permettant d'alimenter le VL53L0X même sous une tension allant de 2.6v à 5.5v

Ce capteur ne possède pas de registres à étudier, mais une API fournis par le constructeur. Il faudra alors extraire de l'API les registres à configurer et ainsi connaître exactement les données à échanger pour une initialisation et pour la réception des données. De plus, sa sortie GPIO configurable permet de prévenir de la disponibilité d'une nouvelle conversion. Cette dernière est envoyée sous 2 octets et exprimé en mm.

Asservissement Moteur

Le servomoteur est en fait un asservissement en position d'un moteur. Il regroupe donc en son sein un moteur, un potentiomètre et un circuit de contrôle. Ils sont souvent commandés par PWM mais on peut en trouver à commande numérique.

J'ai voulu dans un premier temps automatiser l'oscillation du moteur de façon matériel afin de soulager le microcontrôleur de cette tâche redondante. Mais cela aurait nécessité la manufacture d'engrenage précis et couteux.



La captation des obstacles doit sur faire sur les 360° ; or des servomoteurs asservis sur 360° sont rare, mais il existe des servomoteurs à rotation continue. Ce sont en fait servomoteurs à qui on supprime le potentiomètre. Ils passent donc en rotation continue et se comportent comme un moteur avec motoréducteur, dont la vitesse et le sens se commandent par PWM. Etant donné le poids du module très réduit n'excédant pas 10g, le couple importe peu. Un servomoteur à rotation continue de modélisme fera donc l'affaire. Le but étant de calculer le temps que met le servomoteur pour effectuer une rotation et donc de savoir combien de temps le faire tourner dans un sens ou dans l'autre afin d'arriver au point voulu. On obtient ainsi un asservissement à 360°.

Il existe également un type de moteur appelé Pas à Pas qui permet un positionnement extrêmement précis. Il n'est pas utilisé ici de par leur vitesse faible, mais il est notamment utilisé dans les imprimantes 3Ds.

Le servomoteur choisis est un Fitec FS90R. Pesant 9g, il est particulièrement utilisé pour le modélisme. Sur notre projet, le poids est un facteur important car une masse excessive entrainera une surconsommation des moteurs.

Malheureusement, Robotshop nous a livré le servomoteur dans sa version asservie en position. J'ai donc été contraint d'utiliser un des servomoteurs présents à l'école qui est cependant plus lourds, plus large et possède plus de couple. Il s'agit du servomoteur HS-311. Tous les calculs seront donc à faire avec celui-ci.



Etude du HS – 311

Le HS – 311 est un servomoteur à rotation continue

Vitesse : 0.19 / 60° à 4.8V
Poids : 43g
Tension d'alimentation : 4.8 à 6V
Consommation course libre : 300mA
PWM : 50Hz
Temps état haut à la position neutre : 1.5ms



Après des tests réalisés grâce à une carte d'évaluation, voici les nouveaux temps état haut pour des vitesses équivalentes et maximum dans un sens ou dans l'autre :

Sens horaire : 2ms

Sens antihoraire : 1ms

REMISE A ZERO

L'utilisation d'un servomoteur à rotation continue implique que toute notion d'asservissement en position n'existe plus. Or, lors du démarrage du Rover, et en outre, si quelque chose ou quelqu'un force sur le moteur, sa notion de repérage sera perdue. Il faudra donc prévoir une procédure de reset du servomoteur ainsi qu'un circuit détectant les pics de courant résultant d'une force extérieure. Un capteur de proximité tout ou rien donnera l'information au Microcontrôleur que le servomoteur est revenu à sa position d'origine.

Capteur capacitif

Ces types de capteurs réagissent à tous type de matériaux à proximité d'eux en créant une capacité équivalente. Il n'est pas préconisé ici car les fils d'alimentation peuvent se retrouver face à ce dernier ce qui pourrait activer le capteur et fausser les données.

Capteur infrarouge

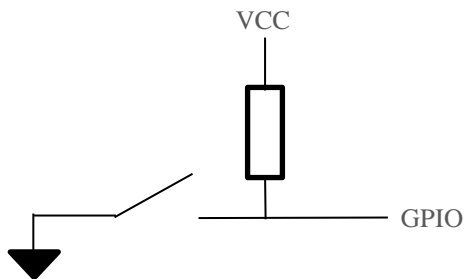
Ces capteurs envoient une lumière infrarouge et possèdent un récepteur. Lorsque la lumière infrarouge reçu est suffisamment forte, cela sature le transistor embarqué. Ceux-ci aussi peuvent être déclenché par n'importe quel objet passant devant eux. Ils ne seront donc pas utilisés ici encore une fois à cause des fils susceptibles de passer devant.

Capteur à effet Hall

Ces modules détectent un champ magnétique. Ils sont donc uniquement activables par un champ magnétique fort à proximité du capteur. Un aimant passant devant lui suffit à l'activer. Il est donc possible d'accrocher un aimant au servomoteur et de laisser le capteur fixe sur le châssis. Une position zéro est ainsi détectable lors du reset ou lors du premier démarrage du Rover. Ce sera donc ce type de capteur qui sera retenu, associé à un petit aimant.

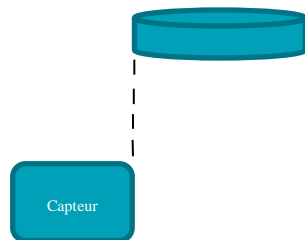


Etude du capteur



Les capteurs REED agissent comme un bouton poussoir, mais uniquement activable par champ magnétique. Il sera donc nécessaire d'y placer une résistance de tirage afin d'éviter les états indéterminés

Une résistance de 10k Ω offrira un courant de 33 μ A.



A une distance allant de 0 à 5 mm entre l'aimant et le capteur, ce dernier devient passant dès lors que l'aimant est au-dessus. Ainsi, dès que l'aimant franchit les pointillés, le capteur sera activé. Au-delà de 5mm de distance, le capteur ne détecte plus le champ électromagnétique.

CARTE ELECTRONIQUE

Schéma fonctionnel Niveau 1

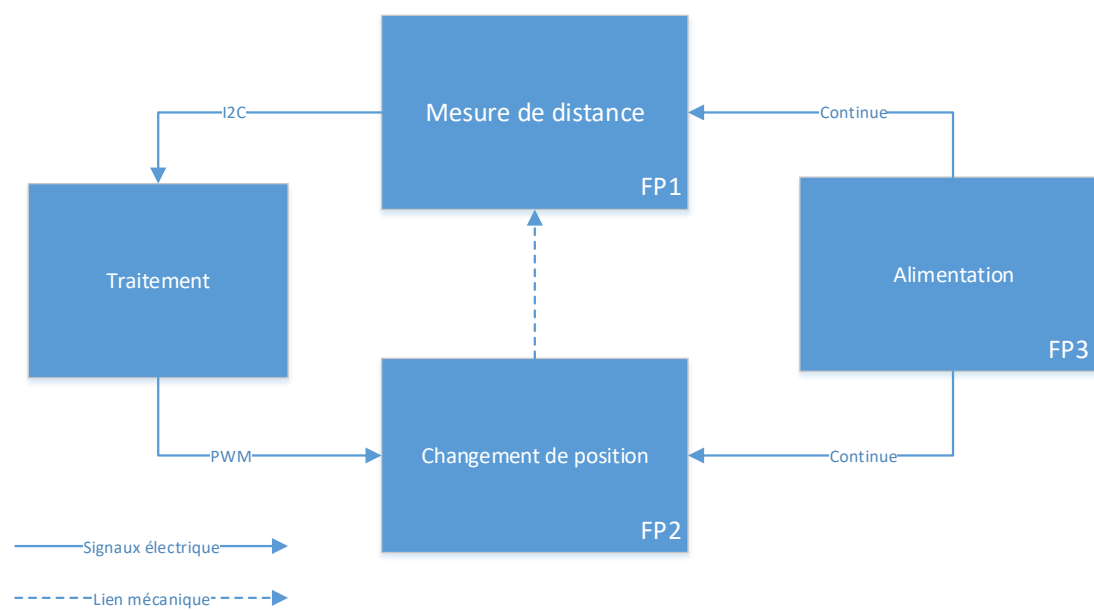
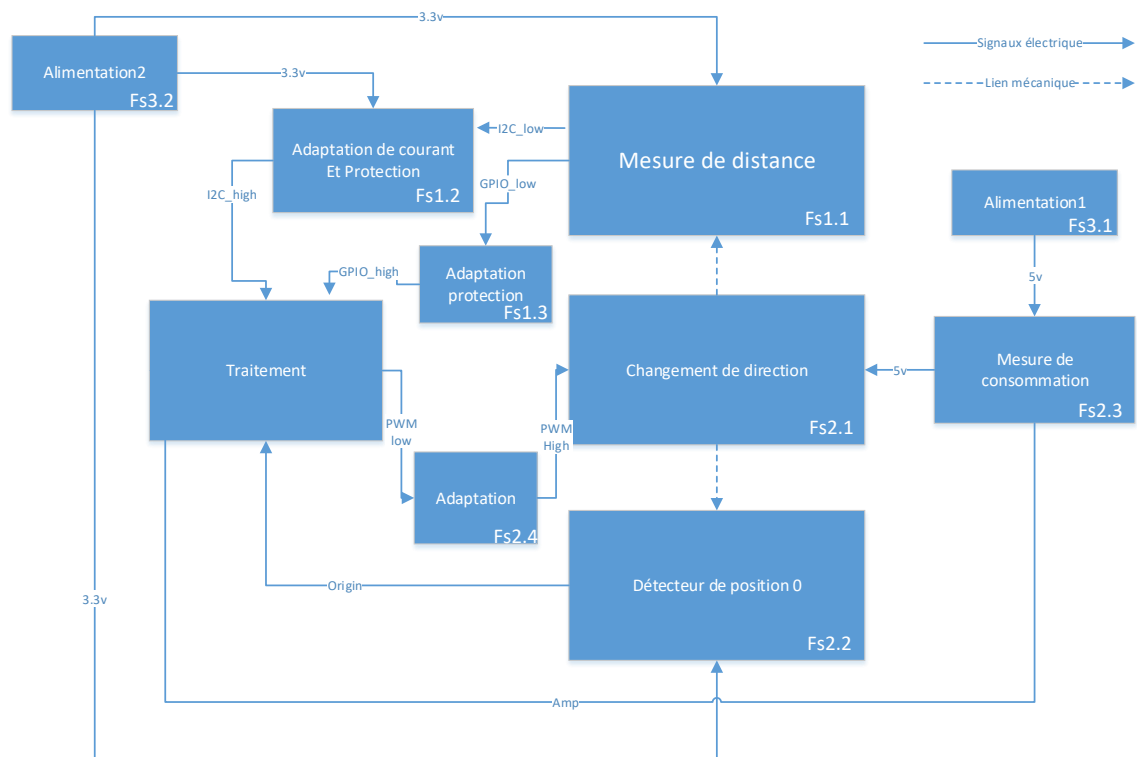


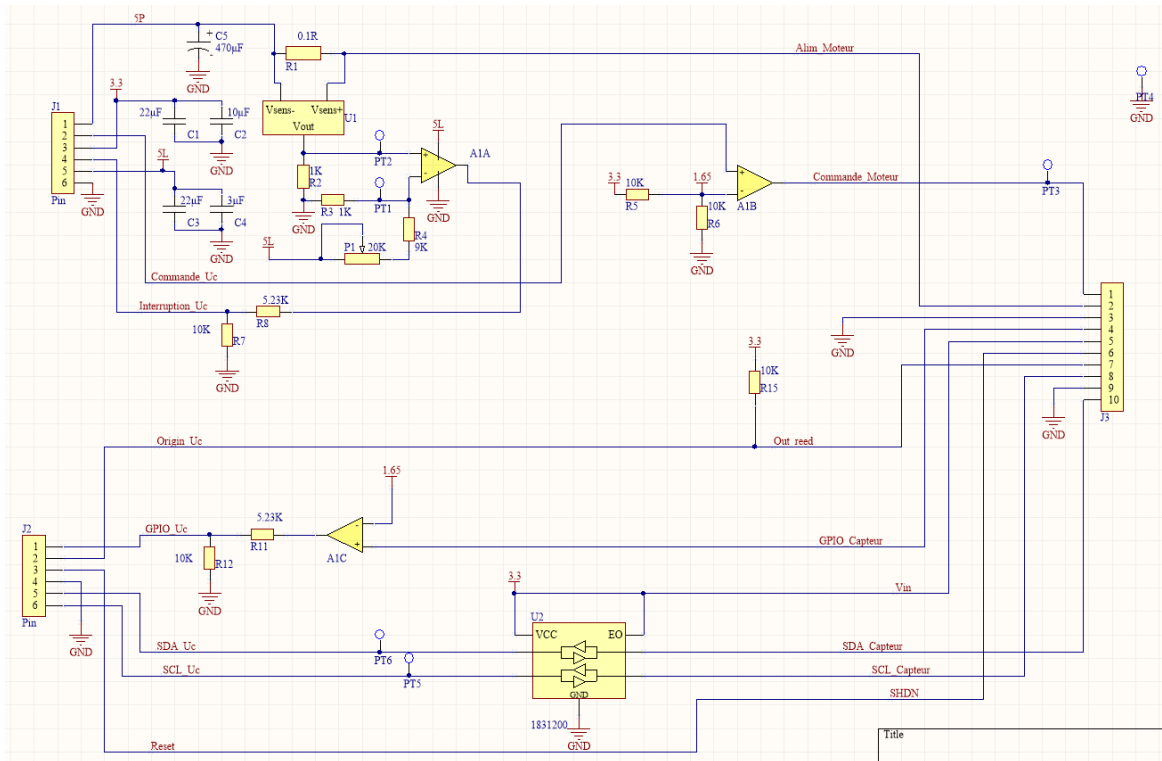
Schéma fonctionnel Niveau 2



3.3V	Signal continue d'alimentation 3.3V
5V	Signal continue d'alimentation 5V
I2C_low	Signal I2C
I2C_high	Signal I2C
Amp	Signal tout ou rien, le point de basculement se faisant à partir d'une certaine consommation afin d'arrêter le moteur
PWM_low	Signal PWM envoyé par le servomoteur, 3.3v max
PWM_high	Signal PWM reçu par le servomoteur, 5v max
Origin	Signal tout ou rien apparaissant lorsque le moteur se trouve en position 0
GIOP_cap	Signal tout ou rien prévenant de la présence d'une nouvelle mesure disponible

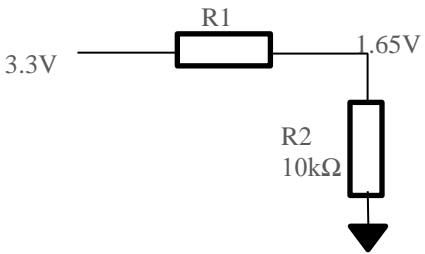
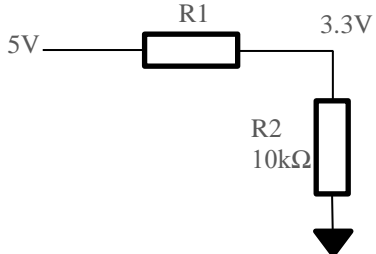
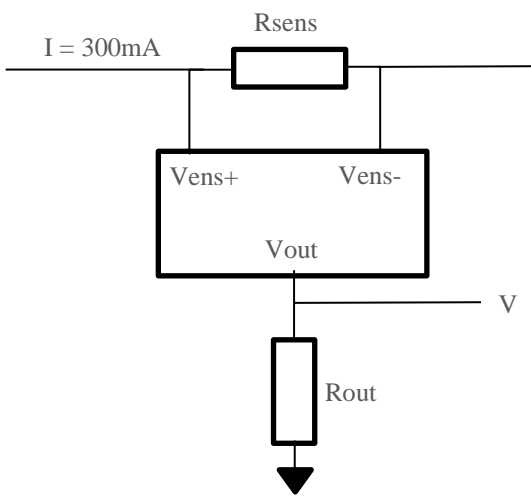
Fs1.1	Analyse la distance depuis le capteur jusqu'au prochain obstacle dans sa direction. Renvois les informations via I2C
Fs1.2	Système de protection I2C afin d'éviter toute surcharge sur le microcontrôleur et sur le capteur
Fs1.3	Adaptation de la tension de sortie du capteur pour être lue par le microcontrôleur. Sert de protection au même titre que la Fs1.2
Fs2.1	Servomoteur pouvant changer la direction du capteur de distance
Fs2.2	Capteur reed permettant de déterminer une position de démarrage fixe
Fs2.3	Capteur de courant permettant d'avertir le contrôleur d'une surcharge afin de réinitialiser le servomoteur à la position fixe
Fs2.4	Adaptation de la tension en sortie du microcontrôleur pour être lu par le servomoteur
Fs3.1	Filtrage de l'alimentation 5V provenant de la carte d'alimentation
Fs3.2	Filtrage de l'alimentation 3.3V provenant de la carte d'alimentation

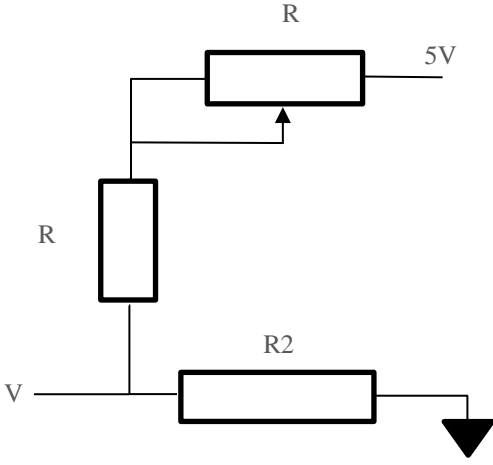
Schema structurel



Fs1.1	Non présent sur la carte de traitement car fixé sur le servomoteur
Fs1.2	U2
Fs1.3	A1C ; R11 ;R12
Fs2.1	Non présent sur la carte de traitement car fixé sur le haut du Rover
Fs2.2	Le Capteur Reed en lui-même est placé à côté du servomoteur. R15
Fs2.3	R1 ; U1 ; R2 ;R3 ;R4 ;P1 ;A1A ;R8 ;R7
Fs2.4	A1B ;R5 ;R6
Fs3.1	C5
Fs3.2	C1 ;C2 ;C3 ;C4

Calcul des résistances

<p>Pont diviseur</p> 	$1.65 = 3.3 \frac{R2}{R1 + R2}$ $R1 = 10k\Omega$ <p>Donc $I = 165\mu A$</p>
<p>Pont diviseur</p> 	$3.3 = 5 \frac{R2}{R1 + R2}$ $R1 = 5151\Omega$ <p>Donc $I = 328\mu A$</p>
<p>ZXCT1009</p> 	<p>Pour $R_{sens} = 0.1 \Omega$ Et $R_{out} = 1k \Omega$</p> <p>D'après la datasheet :</p> $V_{out} = 0.01 * V_{sens} * R_{out}$ <p>Ainsi pour $I=0.3A$; $V_{out} = 0.3V$</p> <p>Donc $I_{out} = 3\mu A$</p> <p>Le capteur est linéaire et suit l'équation $V = I$</p>

<p style="text-align: center;">Diviseur de tension réglable</p> 	$V = 5 \frac{R2}{R + R1 + R2}$ <p>On pose $R2 = k \Omega$</p> <p>Pour un pont diviseur allant de 200mV à 300mV :</p> $V = 200mV$ $V = 5 \frac{R2}{R + R1 + R2}$ $R1 + R = 24k\Omega$ $V = 300mV$ $R1 + R = \frac{R2(5 - V)}{V} = 9k\Omega$ <p>Donc $R1 = 9k \Omega$ et $R = 15k \Omega$</p> <p>$I_{max} = 5\mu A$</p> <p>$I_{min} = 2\mu A$</p>
--	--

Filtrage des tensions d'alimentation

Pour cela, il faut déterminer le courant utilisé par la carte, j'ai donc analysé chaque partie et déterminé le courant maximum et minimum.

Donc $I_{3.3_max} = 25.4mA$

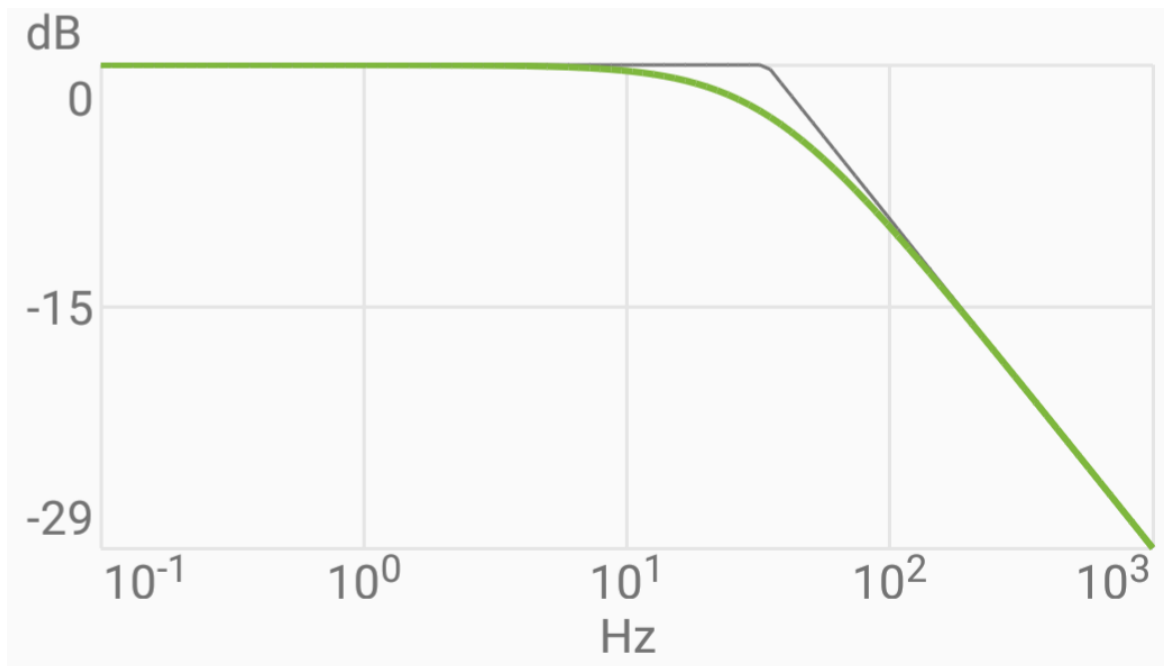
$I_{3.3_min} = 21.5mA$

Ce qui donne une résistance d'entrée : $Re_{3.3_max} = \frac{V}{I} = 155 \Omega$

$$Re_{3.3_min} = \frac{V}{I} = 130 \Omega$$

Donc avec $C = C1 + C2 = 10\mu F + 22\mu F = 32\mu F$

$$f_{0.3.3} = \frac{1}{2\pi Re_{3.3_max} C} = 32Hz$$



Donc $I_{5_max} = 9.18\text{mA}$

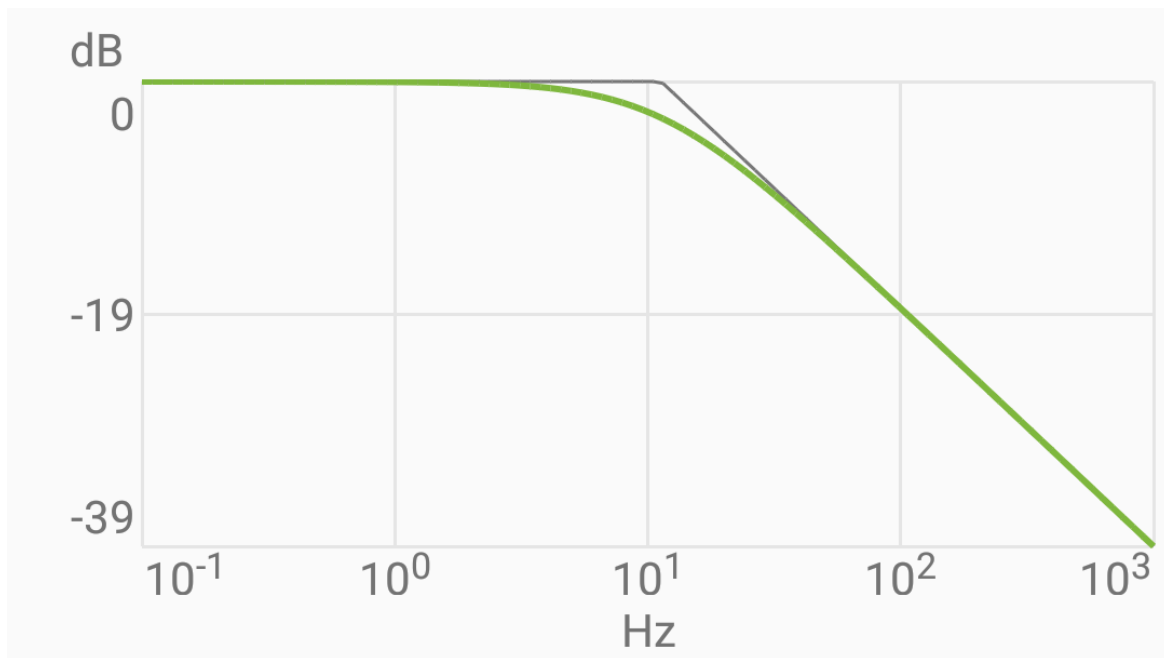
$I_{5_min} = 8.46\text{mA}$

Ce qui donne une résistance d'entrée : $Re_{5_max} = \frac{V}{I} = 591 \Omega$

$$Re_{5_min} = \frac{V}{I} = 544 \Omega$$

Donc avec $C = C3 + C4 = 3\mu\text{F} + 22\mu\text{F} = 25\mu\text{F}$

$$f_{0.5} = \frac{1}{2\pi Re_{5_max} C} = 10\text{Hz}$$



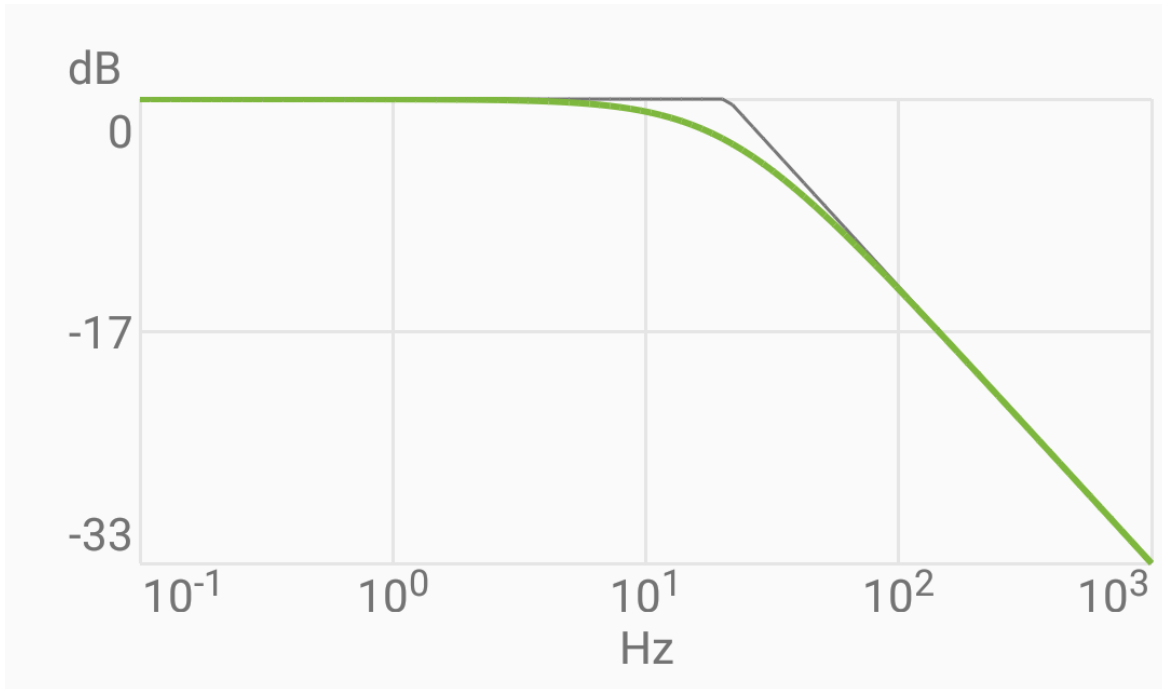
Concernant l'alimentation du servomoteur :

$I_{max} = 300\text{mA}$

Donc $R_e = \frac{V}{I} = 17\ \Omega$

Avec $C = 470\mu\text{F}$

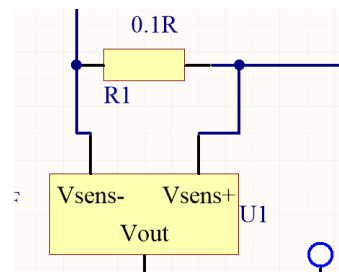
$$f_0 = \frac{1}{2\pi R_e C} = 20\text{Hz}$$



Tests, problèmes et solutions

Les tests de continuités ont révélé une microcoupure due au processus de fabrication. Une reprise grâce à de l'étain a suffi à régler le problème.

Lors de l'élaboration de la schématique, j'ai interverti les broches Vsens- et Vsens+. Ainsi, le Vsens+, censé se retrouver du côté où la tension est la plus haute, se retrouve du côté où la tension est la plus basse. J'ai donc été obligé de faire deux reprises filaires afin d'invertir ces deux broches.

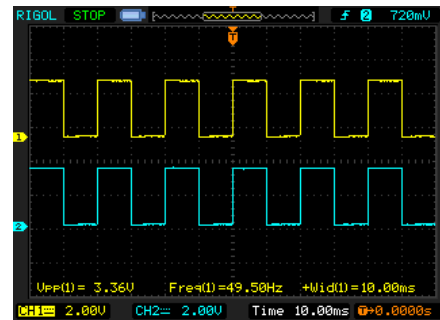


Des problèmes de soudures ont créés des courts circuits sur plusieurs points, ce qui a grillé le quadruple AOP. Après changement une refonte de l'étain des résistances et une vérification minutieuse, j'ai dû remplacer l'AOP.

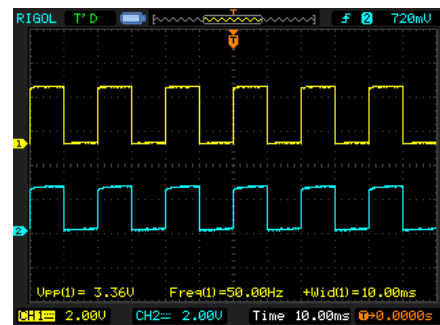
Tests ohmiques :

Entrée mesuré	Résistance	Courant
3.3v	15.2k Ω	217 μ A
5v logique	19.34k Ω	259 μ A

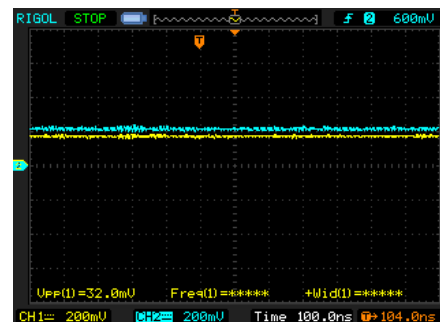
On voit ici une tram I2C entrante, en bleu, généré par un oscillateur externe. La tram est reproduite à l'identiques de l'autre côté du buffer (courbe jaune). Ma carte ne possédant pas de résistance de tirage, j'ai dû en ajouter afin de faire les tests. Ces dernières sont implémentées directement sur la carte mère dans un but économique, n'étant pas le seul à utiliser l'I2C.



La PWM générée ici par un oscillateur externe est représentée en bleu, et possède une amplitude crête à crête de 3.3V. Le signal de sortie en bleu, possède les mêmes propriétés mais avec comme caractéristique une amplitude plus importante puisqu'elle passe à 5V crête à crête.

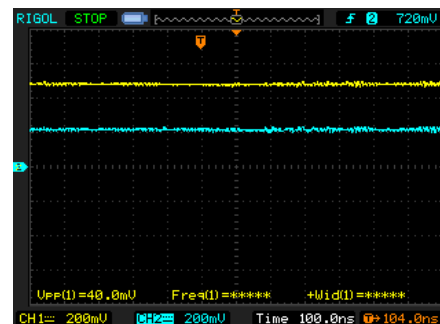


J'ai simulé la consommation du courant par deux résistances en parallèle de 50 Ω ce qui donne une résistance de 25 Ω , alimenté sous 5v, génère un courant de 200mA. La courbe bleue représente le capteur de courant et on peut voir qu'il atteint les 200mV.



La courbe jaune représente la tension de seuil réglé à son minimum.

Ici même condition sur la consommation, mais le potentiomètre est réglé à son maximum, il accepte donc 500mA maximum, sans quoi une alerte sera envoyée au μ Controller afin de stopper le moteur.



Nomenclature

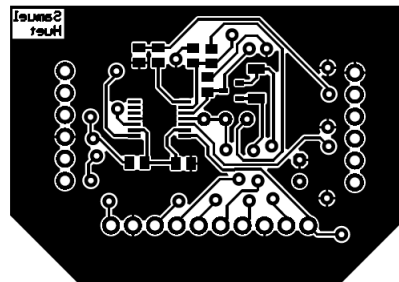
Désignation	Description	Caractéristique	Température de fonctionnement	Fournisseur	Référence	Code commande	Quantité	Prix UHT	Lot	Prix TTC
R1	Resistance 0,1R	0.1 ohm, 200 V, 1206 [3216 Metric], 500 mW, ± 1%	de -55°C à +150°C	Farnell	LR1206-R10F-W	1099914	1	0,587	5	3,522
U1	Captur de courant	SOT-23, 3 Broches], 2,5 V, 20 V	de -40°C à 85°C	Farnell	ZKCT1009F	1132757	1	0,964	1	1,1568
A1	AOP x4	1 MHz, 1 V/µs, 2.7V à 5.5V, TSSOP, 14 Broche(s)	de -40°C à 85°C	Farnell	LMV224PWR	8389276	1	0,67	1	0,804
U2	Buffer IC	2.3 V à 3.6 V, TSSOP-8	de -55°C à +125°C	Farnell	PCA9515ADP	2212062	1	1,33	1	1,596
P1	Potentiomètre 20kR	20 kohm, 500 mW, ± 20%, 1 Tour(s), Linéaire	de -55°C à +125°C	Farnell	3352H-1-208LF	2113963	1	2,02	1	2,424
J1/J2	Pins femelle x6	Connecteur carte-à-carte, Vertical 2,54 mm, 6 Contact(s), Traversant	/	Farnell	22115-066	1593415	2	0,341	1	0,8184
R5,R6,R7,R12,R13,R14,R15	resistance 10K	10 kohm, 150 V, 0805 [2012 Metric], 125 mW, ± 1%	de -55°C à +125°C	Farnell	RC0805FR-0710KL	9337755	1	0,014	10	0,168
R2,R3	Resistance 1K	1 kohm, 150 V, 0805 [2012 Metric], 125 mW, ± 1%	de -55°C à +125°C	Farnell	RC0805FR-071KL	9337496	1	0,014	10	0,168
R4	Resistance 9K	9.09 kohm, 150 V, 0805 [2012 Metric], 125 mW, ± 1%	de -55°C à +125°C	Farnell	FR0805FR091V	2057632	1	0,0269	10	0,328
R8, R11	Resistance 5,2K	5.23 kohm, 150 V, 0805 [2012 Metric], 125 mW, ± 1%	de -55°C à +125°C	Farnell	FR0805FR5231V	2303626	1	0,0269	10	0,328
U3	Captur reed	20 mm, 10 W, 200 Vac/dc, 0,5 A	de -20°C à 85°C	Farnell	MK6-5-C	1079480	1	1,11	1	1,332
C2	Condensateur 10µF	1206 [3216 Metric], 10 µF, 16 V, ± 10%	de -55°C à +125°C	Farnell	1206YC06KAT2A	1657943	1	0,18	5	1,08
C3	Condensateur 22µF	1206 [3216 Metric], 22 µF, 16 V, ± 10%, X5R	de -55°C à +85°C	Farnell	MC1206X226K160CT	2320923	1	0,285	5	1,71
C4	Condensateur 3µF	1206 [3216 Metric], 3.3 µF, 25 V, ± 10%	de -55°C à +125°C	Farnell	GRM31CR71E335KA88L	2494296	1	0,371	5	2,226
J3	Connecteur carte-fil	Connecteur fil-à-carte, 2,54 mm, 7 Contacts, 1 Rangée	de -55°C à +105°C	Farnell	103688-6	2429626	1	0,73	1	0,876
J4	Connecteur fil-carte	Connecteur carte-à-carte, 2,54 mm, 10 Contacts, Traversant, 1 Rangée	de -40°C à +125°C	Farnell	61301011121	1841229	1	0,84	1	1,008
C5	Condensateur 470µF	470 µF, 6.3 V, ± 20%, 6.3 mm	de -40°C à 105°C	Farnell	MCK56R3M471E115	2627681	1	0,0847	10	1,0164
J5	Connecteur fil-carte	Connecteur fil-à-carte, Angle Droit, 2,54 mm, 7 Contacts, Traversant	/	Farnell	640457-2	588763	1	0,317	5	1,902
S1	Servomoteur	Servomoteur Micro 9g Rotation Continue	/	Robotshop	RB-Fit-03		1	4,66	1	5,59
U4	Module ToF	Captur de Distance ToF VL53LOX - 50 à 1200mm	/	Semageek	ADA 3317		1	14,36	1	17,95
M	Aimant	1/4" X 1/16" Aimant au Neodymium	/	Robotshop	RB-Sha-30		1	0,89 €	1	1,07
							Total			47,0632

Plan de connexion

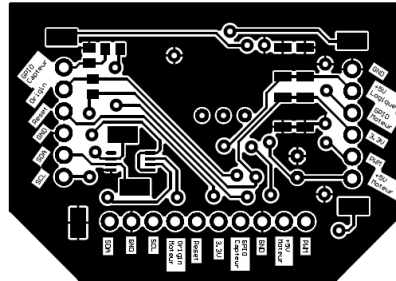
J1.1	Alimentation	+5v de puissance
J1.2	PWM	Pin 61 (TPM3CH5)
J1.3	Alimentation	+3.3v
J1.4	Surconsommation Moteur	Pin 50 (PTA0)
J1.5	Alimentation	+5v logique
J1.6	Masse	GND
J2.1	GPIO capteur	Pin 49 (PTA1)
J2.2	Origine moteur	Pin 62 (PTC4)
J2.3	Reset	Pin 63
J2.4	Masse	GND
J2.5	SDA capteur	Pin 48
J2.6	SCL capteur	Pin 47

TYPONS

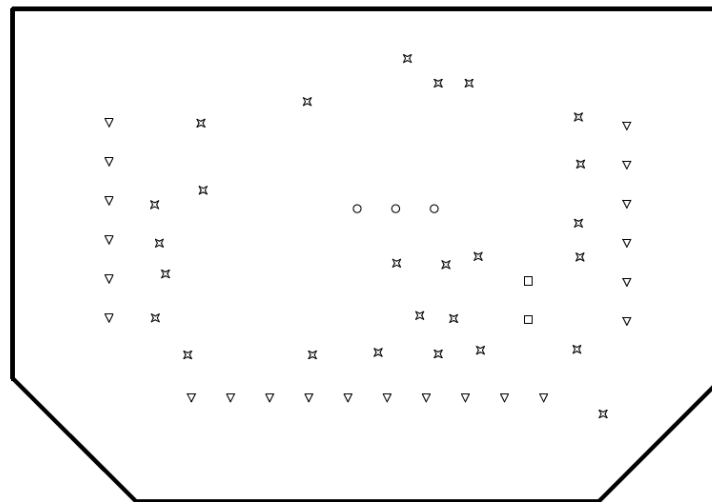
Bottom layer



Top layer



Plan de perçage






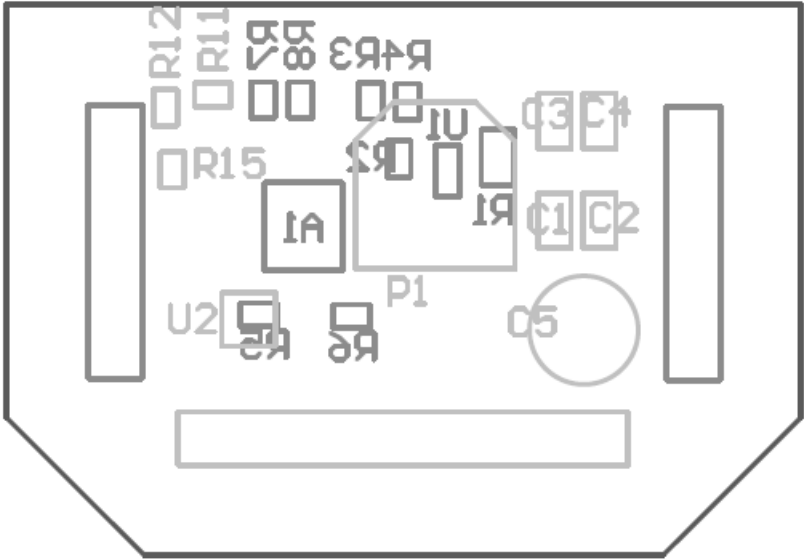
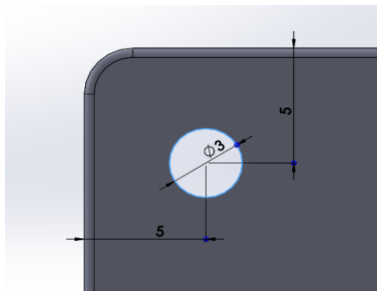
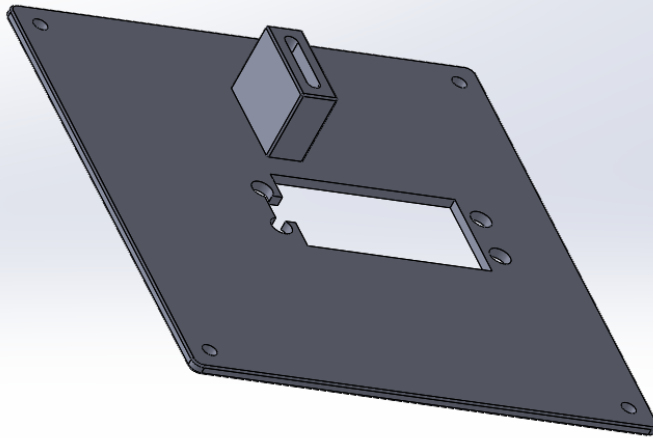
Symbole	Taille du trou
	0.6mm
	0.6mm
	1mm

Schéma d'implantation



Support physique

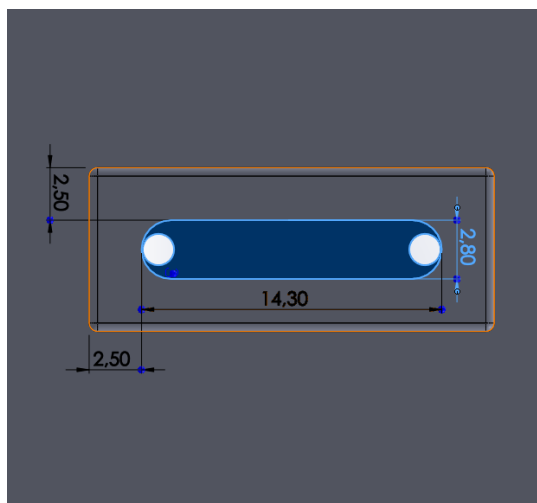
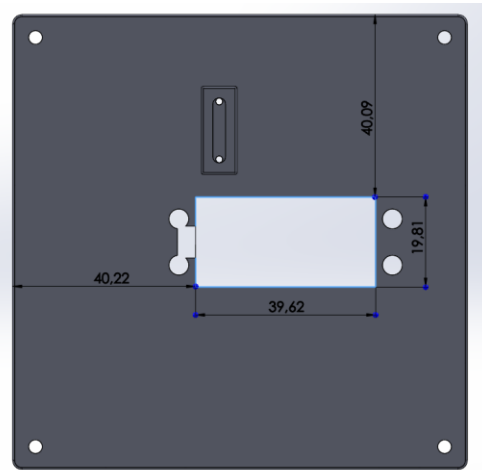
Afin de pouvoir intégrer le système de tourelle mis en place, il a été nécessaire de fabriquer un support physique. Pour cela j'ai donc pris support sur la carte de traitement du Rover, sur lequel a été placé 4 trous, séparé par 9cm en distance chacun, et dont le centre se trouve être le centre du Rover. J'ai donc opté pour une plaque de 10cm par 10cm, dont le centre serait le centre de rotation du servomoteur.



Sur une plaque de 10*10 cm, les trous sont donc à 5mm du bord. Ils font 3mm de diamètre afin d'être dans la capacité de hausser la hauteur grâce à des entretoises. La plaque fait 2mm d'épaisseur afin d'être suffisamment solide pour supporter un objet en mouvement mais aussi afin de pouvoir être imprimé correctement avec la majorité des imprimantes 3D.

Le trou central est destiné à accueillir le servomoteur, il est donc aux exact dimensions mentionnées sur la datasheet du HS-331, en prenant soin de placer au centre le pignon.

En revanche, le rectangle à droite n'est pas présent sur la documentation mais est nécessaire au passage des câbles du servomoteur, les dimensions ne laissant pas de marges de manœuvre.



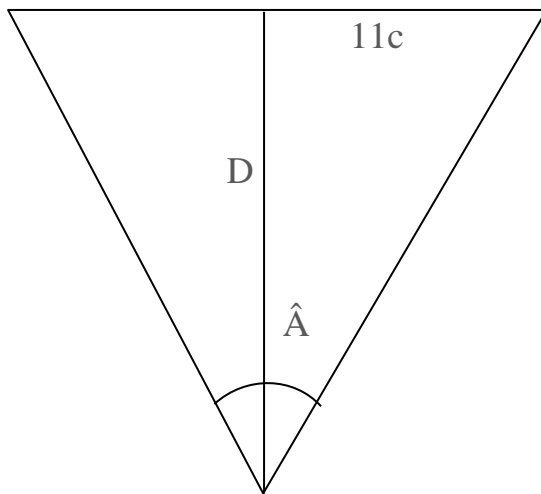
Ce promontoire accueille le capteur reed, destiné à la remise à zéro du servomoteur. Il est donc aux dimensions exactes mentionnées dans la datasheet du capteur. Les deux trous correspondent à l'emplacement des pattes du composant et font 1mm de diamètre pour que l'imprimante puisse l'imprimer.

Cette empreinte a ensuite été surélevée suivant les indications de la documentation du servomoteur, de 1.5 cm, ce qui permettra à l'aimant de frôler la surface du capteur.

Concernant la pièce se trouvant sur le sommet du servomoteur, il a fallu déterminer l'inclinaison du capteur vers le sol en prenant en compte plusieurs facteurs : le temps que le servomoteur mettra pour faire un balayage complet, la vitesse du Rover, ainsi que la distance minimum de détection.

$$\text{Temps aveugle} = 2 * \frac{0.19}{60} * \text{Angle de balayage}$$

Or pour une largeur du Rover de 22cm :



$$\tan \frac{A}{2} = \frac{0.11}{D}$$

$$\text{Donc } D = \frac{0.11}{\tan \frac{A}{2}} \text{ mètres}$$

Vitesse du Rover = 31.7 cm/s Donc 3.1545 s/m

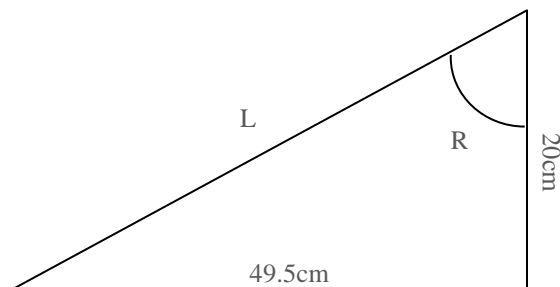
Or temps aveugle < D * 3.1545

$$\text{Donc } 2 * \frac{0.19}{60} * A < \frac{0.11}{\tan \frac{A}{2}} * 3.1545$$

Alors $A < 73^\circ$

Pour un balayage de 70° on a alors un temps aveugle de 445ms et une distance minimale de détection de 49,5cm. En dessous de cette distance, le Rover ne pourra garantir de détecter l'obstacle.

Grâce à ses valeurs, on peut calculer l'inclinaison du capteur vers le sol :



$$\tan R = \frac{49.5}{20}$$

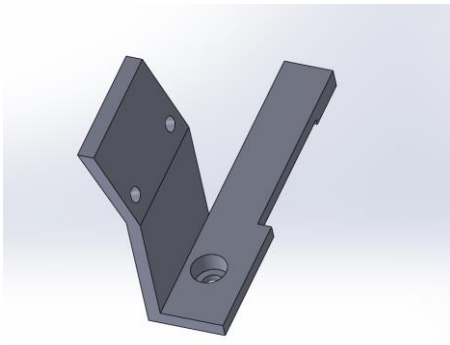
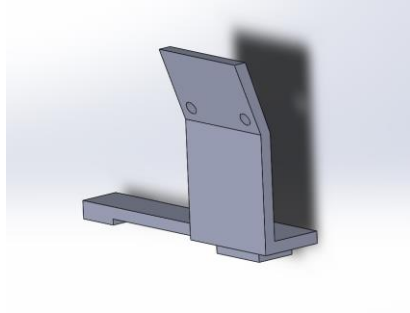
$$R = 67.9^\circ$$

Distance que le capteur doit donc détecter : $L^2 = 20^2 + 49.5^2$

Donc $L = 53.3\text{mm}$

Par précaution, nous utiliserons une distance de mesure de 50cm

J'ai créé une pièce 3D à placer sur le sommet du servomoteur, sensé accueillir le capteur. La place du capteur est coudé de façon à obtenir l'angle calculé précédemment, et en hauteur pour éviter que le capteur détecte son propre support.



Un trou est placé au centre afin de visser ce support au servomoteur. De plus, un ergo rentrera parfaitement dans le support fourni lors de l'achat du servomoteur. Le bras quant à lui accueille un aimant sensé activer le capteur Reed et donner une position 0 au moteur.

Problème et solution :

A l'impression de la pièce, les valeurs étaient trop justes. Il était donc impossible d'y faire rentrer le servomoteur et encore moins le capteur. J'ai donc limé les contours correspondant à l'emplacement du servo jusqu'à ce qu'il rentre, puis passé le trou du capteur à la dremel jusqu'à son intégration.

De plus, il se trouve que deux cartes branchées sur la carte mère, dont la mienne, se trouvaient dans la trajectoire des entretoises. J'ai donc coupé un petit morceau où, par chance, aucune piste passait, laissant ainsi place à l'entretoise. En revanche, pour la seconde carte, il est impossible de la couper. La tourelle tiendra donc sur 3 entretoises, ce qui lui confèrera une stabilité suffisante.

Programme

MISE EN ŒUVRE DU VL53LoX

Etude de fonction I2C

IICA

AD7	AD6	AD5	AD4	AD3	AD2	AD1	
-----	-----	-----	-----	-----	-----	-----	--

AD : Contient l'adresse du module IIC esclave

IICF

MULT	ICR
------	-----

MULT : A 00 pré divise la fréquence de bus par 1
Fbus = 4.197434 Mhz

ICR : A 001100, divise la clock par 12

IICC1

IICEN		MST	TX	TXAK	RSTA		
-------	--	-----	----	------	------	--	--

IICEN : A 1, active la fonction I2C

MST : A 1, le microcontrôleur est configuré en maitre

TX : A 1, active le mode réception
A 0, active le mode transmission

TXAK : A 0, active les acquittements

RSTA : A 1, génère un repeated start, ce bit est remis à zéro après sa lecture.

IICS

TCF	IAAS	BUSY	ARBL		SRW		RXAK
-----	------	------	------	--	-----	--	------

TCF : Flag, a 1, le transfert a été complété. RAZ par lecture de IICD, en mode réception, ou écriture en mode émission

IAAS : Flag, A 1, L'adresse de l'esclave et celle appelé correspondent. Ecrire dans IICC reset ce bit

BUSY : Flag, A 1, le bus est occupé, RAZ quand un stop est détecté

ARBL : flag, A 1, perte d'arbitrage. RAZ y écrivant un 1

SRW : flag, A 0, l'esclave envoie, le maître reçoit. L'esclave reçoit, le maître écrit

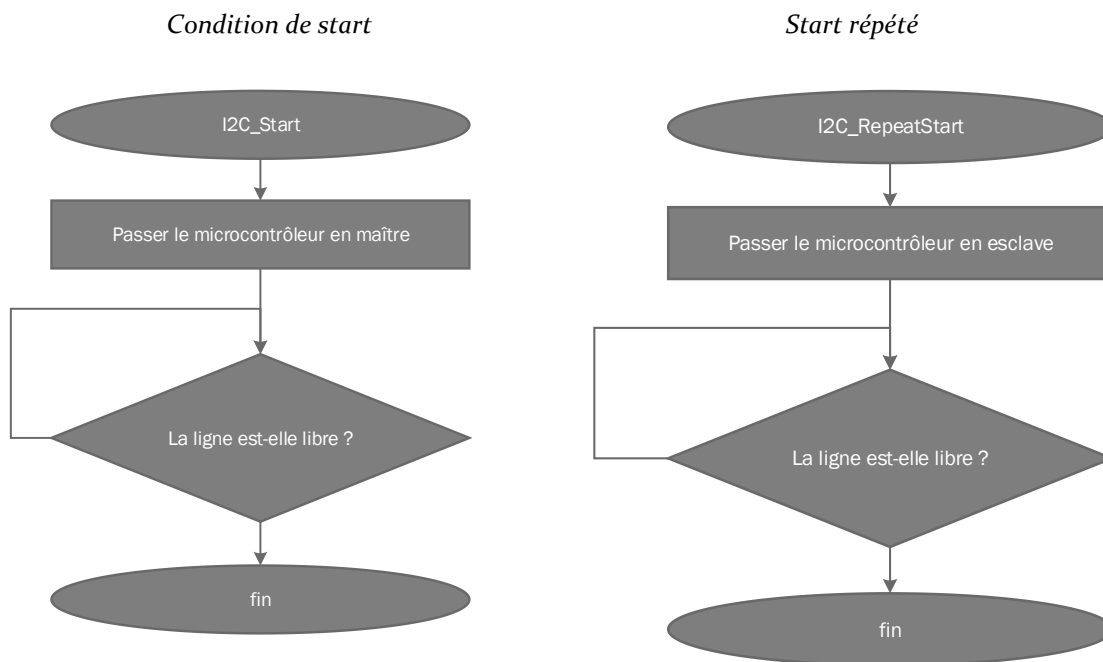
RXAK : A 0, un acquittement a été détecté.

IICD

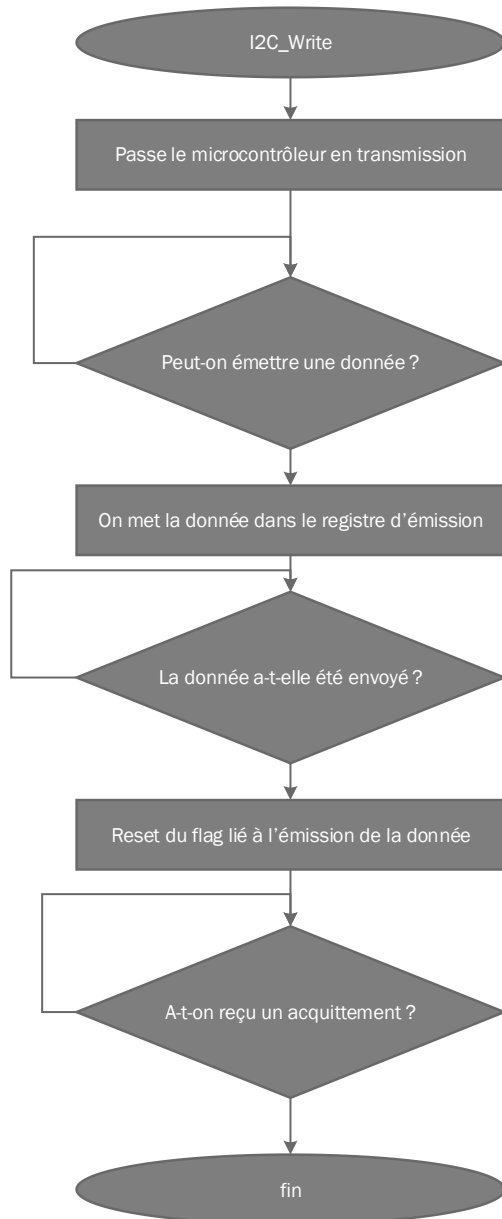


Registre d'émission et de réception de la fonction I2C

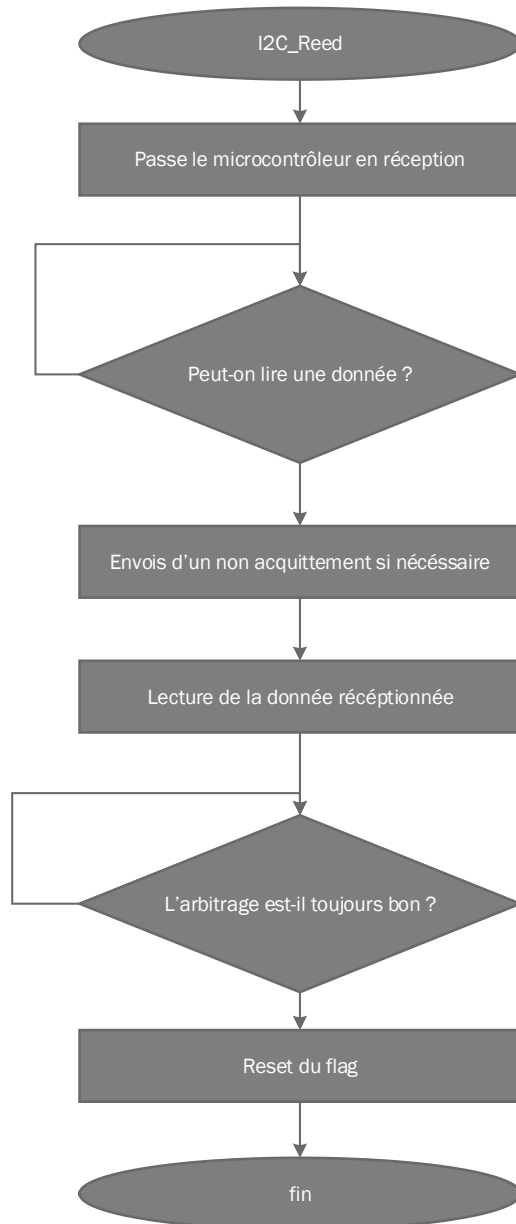
Les bits et registres non étudiés sont laissés dans leur état par défaut.



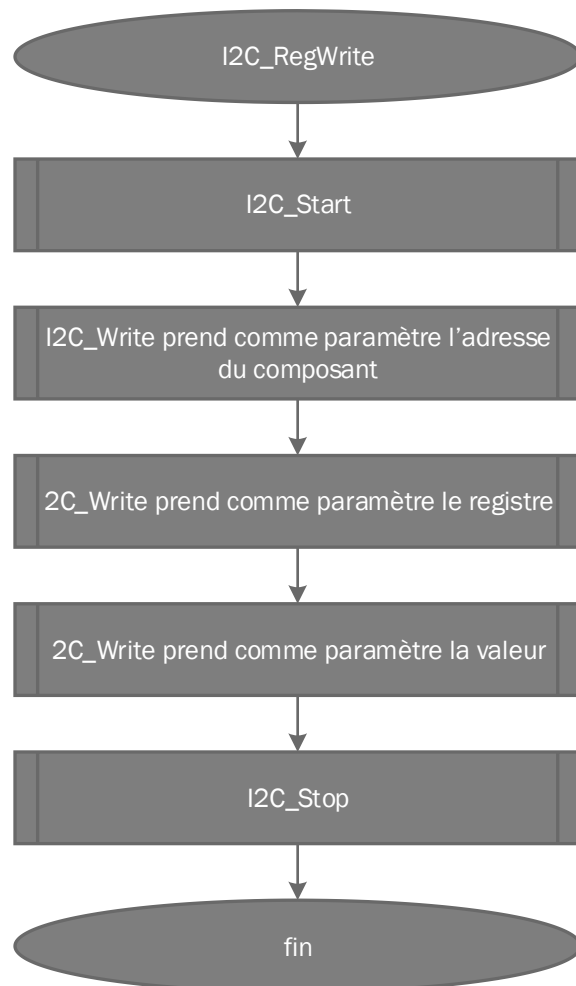
Ecriture simple



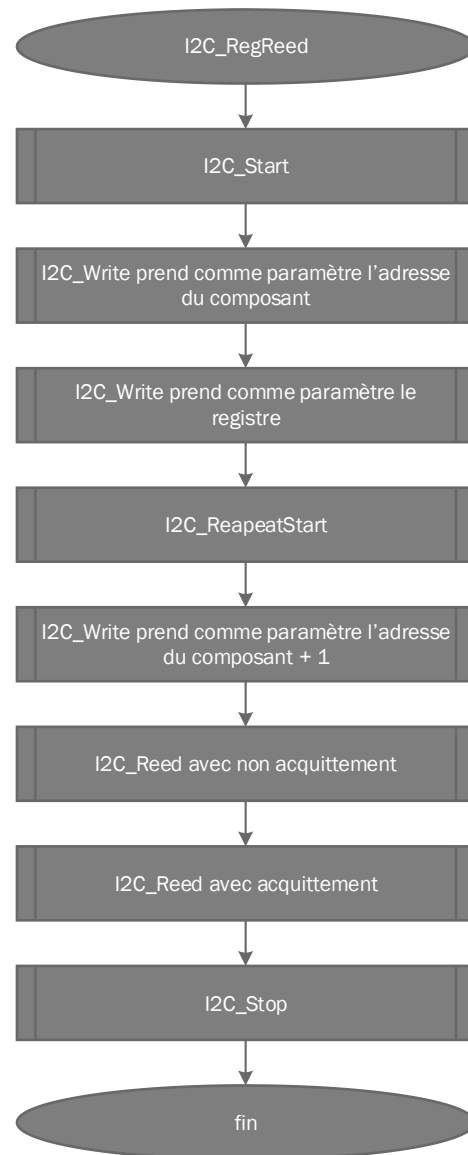
Lecture simple



Fonction écriture



Fonction lecture



Initialisation et utilisation

Extraction des registres

Après étude de l'API fournis par ST Microelectronics, je n'ai pas réussi à extraire les registres à modifier afin d'obtenir les paramètres souhaités. J'ai donc décidé d'agir par retro engineering.

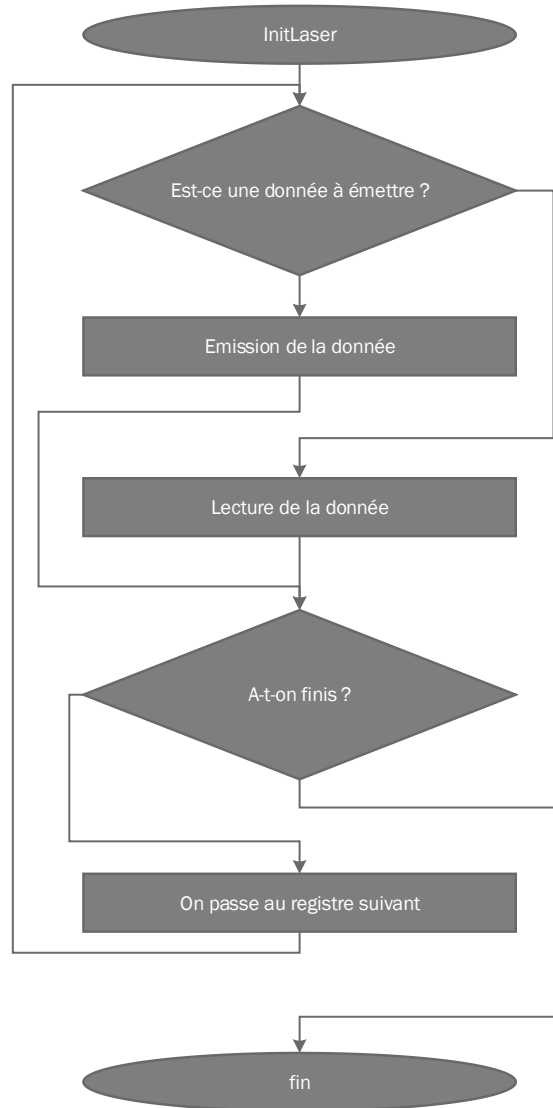
Grâce à un code Arduino trouvé sur internet, ainsi qu'un analyseur logique de la marque Saleae, j'ai pu identifier les échanges effectués entre l'Arduino et le module.



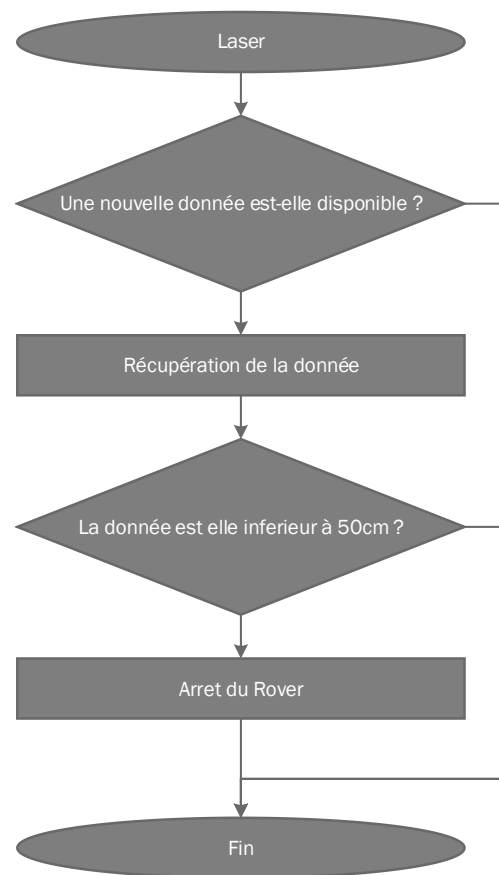
Registres	Données	Registres	Données
0x89,	Lecture	0x66,	0xA0,
0x89	0x01,	0xFF,	0x01,
0x88,	0x00,	0x22,	0x32,
0x80,	0x01,	0x49,	0xFF,
0xFF,	0x01,	0x47,	0x14,
0x00,	0x00,	0x4A,	0x00,
0x91,	Lecture	0xFF,	0x00,
0x00,	0x01,	0x7A,	0x0A,
0xFF,	0x00,	0x7B,	0x00,
0x80,	0x00,	0x78,	0x21,
0x60,	Lecture	0xFF,	0x01,
0x60,	0x12,	0x23,	0x34,
0x44,	0x00, 0x20,	0x42,	0x00,
0x01,	0xFF,	0x44,	0xFF,
0x80,	0x01,	0x45,	0x26,
0xFF,	0x01,	0x46,	0x05,
0x00,	0x00,	0x40,	0x40,
0xFF,	0x06,	0x0E,	0x06,
0x83,	Lecture	0x20,	0x1A,
0x83,	0xDD,	0x43,	0x40,
0xFF,	0x07,	0xFF,	0x00,
0x81,	0x01,	0x34,	0x03,
0x80,	0x01,	0x35,	0x44,
0x94,	0x6B,	0xFF,	0x01,
0x83,	0x00,	0x31,	0x04,
0x83,	Lecture	0x4B,	0x09,
0x83,	0x01,	0x4C,	0x05,
0x92,	Lecture	0x4D,	0x04,
0x81,	0x00,	0xFF,	0x00,
0xFF,	0x06,	0x44,	0x00,
0x83,	Lecture	0x45,	0x20,
0x83,	0xDD,	0x47,	0x08,
0xFF,	0x01,	0x48,	0x28,
0x00,	0x01,	0x67,	0x00,
0xFF,	0x00,	0x70,	0x04,
0x80	0x00,	0x71,	0x01,
0xB0,	Lecture 6 octets	0x72,	0xFE,
0xFF,	0x01,	0x76,	0x00,
0x4F,	0x00,	0x77,	0x00,
0x4E,	0x2C,	0xFF,	0x01,
0xFF,	0x00,	0x0D,	0x01,
0xB6,	0xB4,	0xFF,	0x00,
0xB0,	0x3F, 0x00, 0x00, 0x00, 0x00, 0x00,	0x80,	0x01,
0xFF,	0x01,	0x01,	0xF8,
0x00,	0x00,	0xFF,	0x01,
0xFF,	0x00,	0x8E,	0x01,
0x09,	0x00,	0x00,	0x01,
0x10,	0x00,	0xFF,	0x00,
0x11,	0x00,	0x80,	0x00,
0x24,	0x01,	0x0A,	0x04,
0x25,	0xFF,	0x84,	Lecture
0x75,	0x00,	0x84,	0x01,
0xFF,	0x01,	0x0B,	0x01,
0x4E,	0x2C,	0x01,	Lecture
0x48,	0x00,	0x50,	Lecture
0x30,	0x20,	0x46,	Lecture
0xFF,	0x00,	0x51,	Lecture 2 octets
0x30,	0x09,	0x70,	Lecture
0x54,	0x00,	0x71,	Lecture 2 octets
0x31,	0x04,	0x01,	0xE8,
0x32,	0x03,	0x01,	Lecture
0x40,	0x83,	0x50,	Lecture
0x46,	0x25,	0x46,	Lecture
0x60,	0x00,	0x51,	Lecture 2 octets
0x27,	0x00,	0x70,	Lecture
0x50,	0x06,	0x71,	Lecture 2 octets
0x51,	0x00,	0x71,	0x02, 0x94,
0x52,	0x96,	0x01,	0x01,
0x56,	0x08,	0x00,	0x41,
0x57,	0x30,		
0x61,	0x00,		
0x62,	0x00,		
0x64,	0x00,		
0x65,	0x00,		

PAGE 42

Initialisation



Mise en œuvre



Problème

Après de nombreux essais, nous n'avons pas réussi à faire fonctionner le module ToF. Une erreur subvient au moment de l'échange de donnée, le capteur n'envoie pas d'acquiescement après l'envoi de certaines données, ce qui fausse son initialisation. De plus, en ne prenant pas en compte ces erreurs, le capteur renvoie alors systématiquement la valeur maximum soit 1FFF. J'ai travaillé sur ce point avec Axel, responsable du suivi de ligne, et à ce jour, nous n'avons toujours pas trouvé la solution.

MISE EN ŒUVRE DU SERVOMOTEUR

Etude de fonction PWM

L'objectif de cette étude de fonction est de générer une PWM à 50Hz sur le timer 3, channel 3. Cette PWM pourra prendre 3 rapports cycliques différents, correspondant à une rotation horaire, antihoraire et une position neutre du servomoteur.

$F_{bus} = 16.777216 \text{ Mhz}$

TPM3SC

		CPWM	CLKS	PS
--	--	------	------	----

CPWM : A 1, tous les channels sont utilisés en PWM, aligné sur le centre

CLKS : A 01, utilise la fréquence de bus pour générer la PWM

PS : A 010, divise la fréquence de bus par 4, on obtient donc une fréquence de 4.1943 MHz

TPM3CNT

Double registre 8 bits servant au comptage des Timerticks

TPM3MOD

Contient le nombre de timerticks correspondant à la demi période

$TPM3MOD = 41943_{(10)} = A3D7_{(16)}$

TPM2C3V

Contient le nombre de Timerticks correspondant à la moitié du temps état haut

Rotation horaire : $4194_{(10)} = 1062_{(16)}$	taux d'erreur = 0.007%
Rotation antihoraire : $2097_{(10)} = 831_{(16)}$	taux d'erreur = 0.007%
Position neutre : $3146_{(10)} = C4A_{(16)}$	taux d'erreur = 0.008%

Les fonctions I2C ont été réalisés par Axel dont la caméra doit être initialisé grâce à l'I2C. J'ai donc réutilisé ses fonctions.

Etude de fonction Timer

L'étude de la fonction a pour but de déclencher une interruption toutes les 0.5ms.

Fréquence de bus du Microcontrôleur : $f_{bus} = 16.777216 \text{ Mhz}$

TPM2SC

TOF	TOIE	CPWM	CLKS	PS
-----	------	------	------	----

TOF : (flag) A 1, le compteur a dépassé la valeur programmée dans le modulo. RAZ par lecture de TPM1SC lorsque TOF est à 1, suivis d'une écriture d'un 0 logique dans TOF.

TOIE : A 11, active une interruption lorsque TOF est a 1.

CPWM : A 0, préconfigure le timer en mode output compare

CLKS : A 01, sélectionne l'horloge de bus interne pour la fonction Timer

PS : A 000, pas de pré division de l'horloge de bus interne (1 Timertick = 59.6046 ns)

TPM2CNT

Double registre 8 bits servant au comptage des Timerticks

TPM2MOD

Double registre 8 bit contenant la limite du compteur. Après avoir atteint cette valeur, TPM1CNT sera reset et recommencera à compter.

Pour un délai de 0.5ms, $TPM1MOD = 8389_{(10)} = 20C5_{(16)}$

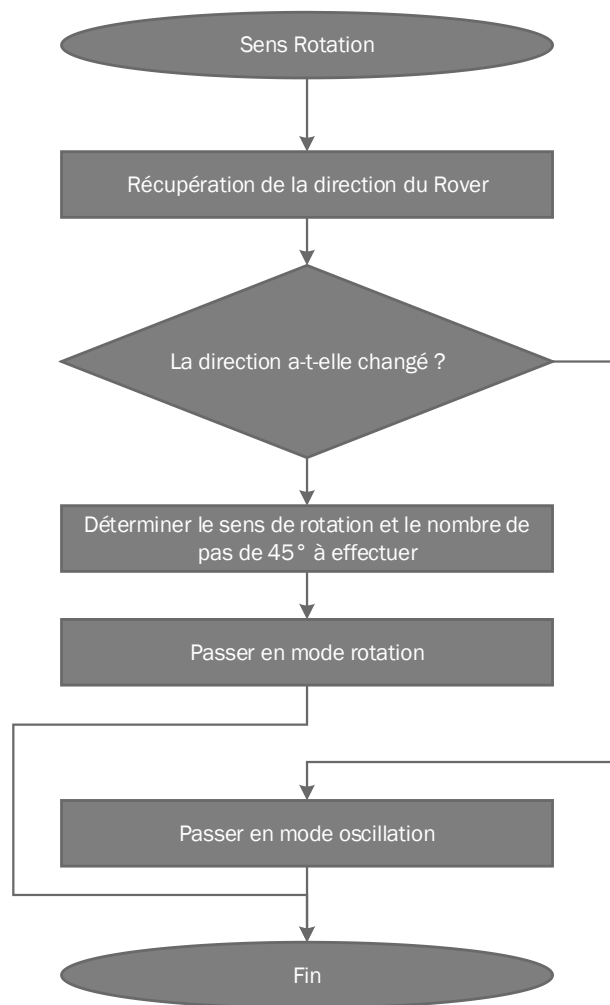
$$\text{Taux d'erreur} = \left(1 - \frac{0.5 \cdot 10^{-3}}{8389 \cdot 59.6046 \cdot 10^{-9}}\right) * 100 = 0.0045\%$$

Les bits et registres non étudiés sont laissés dans leur état par défaut.

Asservissement de position

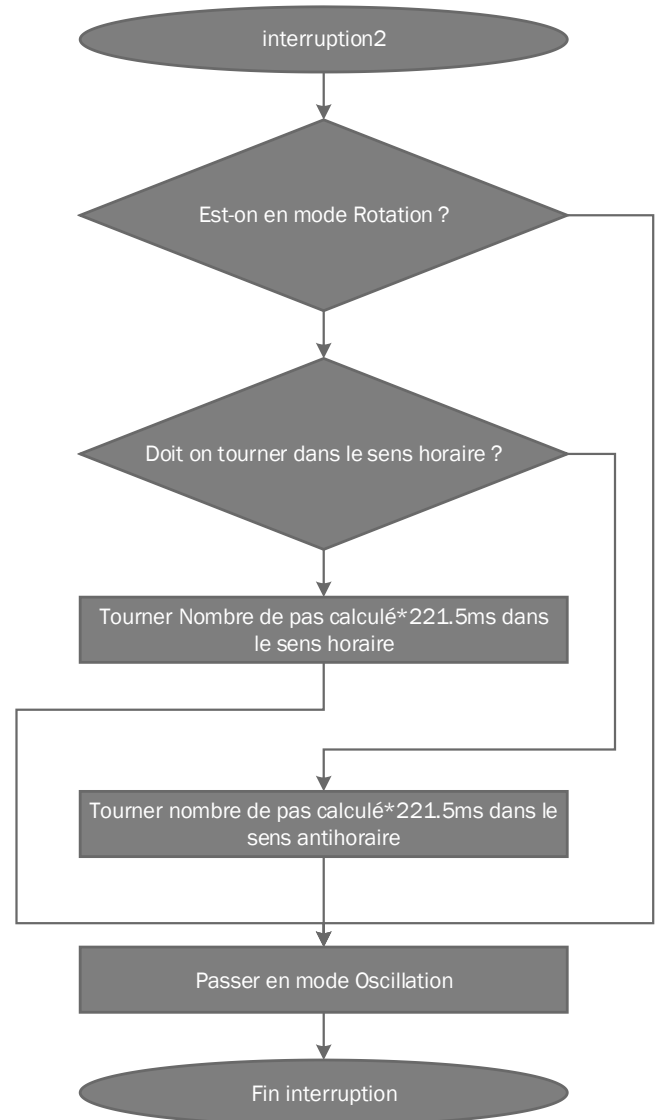
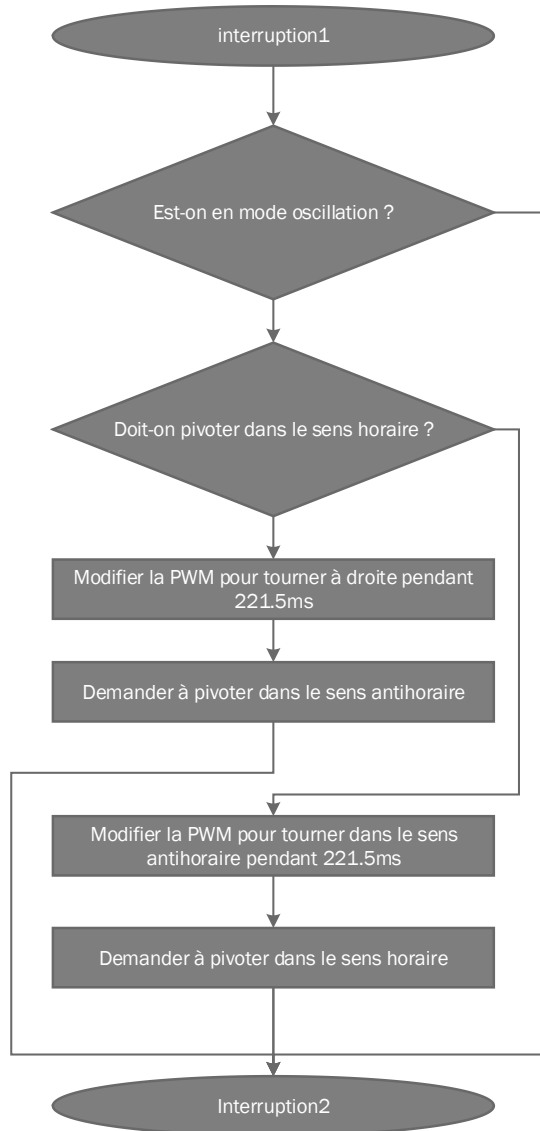
Paramétrage

L'asservissement en position se fait en fonction du nombre de tour par seconde du servomoteur donné dans la datasheet.

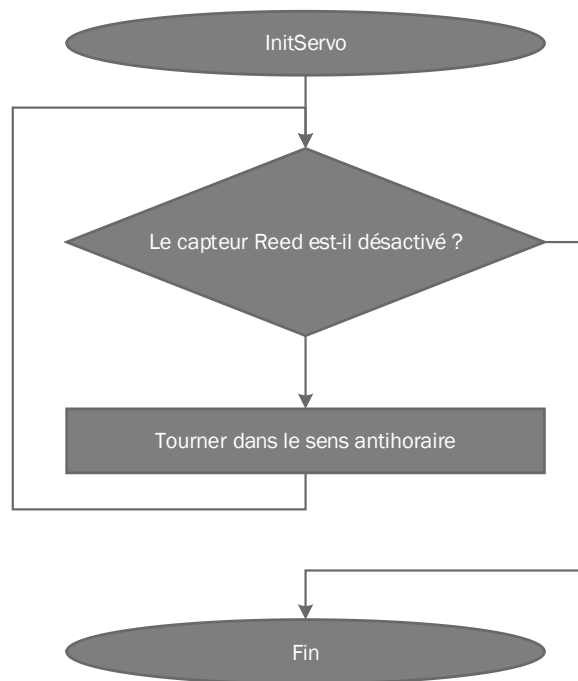


Interruption

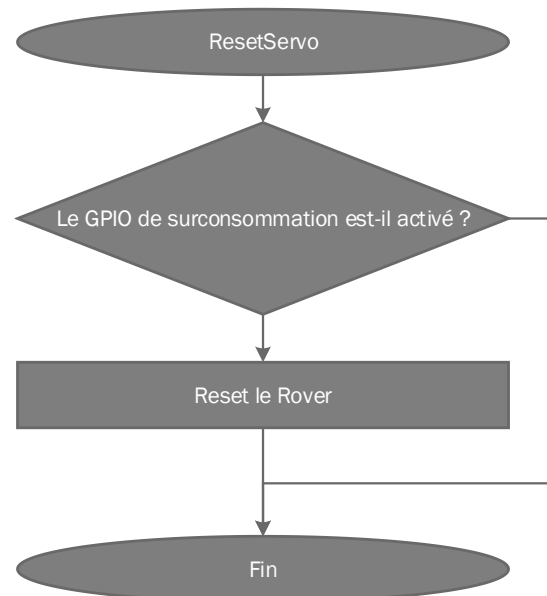
Ce code d'interruption est déclenché toutes les 0.5ms



Initialisation position o



Arrêt du servomoteur



Problèmes et solutions

Lors du changement de direction de la tourelle, à savoir quand la direction du rover change, la variable sensé s'incrémenté semble être étonné et les conditions déduites s'en trouvent fausses. Ainsi, il est impossible de faire changer de direction le laser.

Un problème similaire a été aperçu lors de la première version du code pour lequel j'avais conçu une fonction placée dans l'interruption, engendrant un dépassement de pile. Aucune solution n'a encore abouti mais une recherche est activement en cours.

Conclusion

On peut observer une désynchronisation lors du balayage probablement due aux taux d'erreurs mais aussi basé sur le fait que le servomoteur n'est pas conçu pour avoir une course parfaitement régulière. Ainsi un moteur pas à pas aurait été un choix plus judicieux bien que moins aisé à mettre en œuvre. On aurait pu imaginer placer autant de capteur reed qu'il y a d'angle de déplacement afin de ne jamais perdre la position souhaitée.

De plus, le capteur choisi s'est révélé peu relativement peu documenté, et n'ayant eu que deux semaines pour le mettre en œuvre du au retard de la livraison, il a été difficile de trouver les paramètres nécessaires et l'initialisation. Cependant il se révèle efficace et rapide.

J'ai choisi pour cette carte de pouvoir la brancher sur la carte mère grâce à une série de header pour afin de pouvoir effectuer mes tests indépendamment et pouvoir tirer la carte dès que nécessaire. La carte mère n'ayant été tiré que bien plus tard, ce fut un choix judicieux. Cependant, le prix des headers a rendu la carte plus chère et nous pourrions imaginer une intégration directe sur la carte mère pour une version définitive.

Le projet dans son ensemble fut une expérience enrichissante qui m'a montré l'importance du travail de groupe et de la coopération, que ce soit dans le groupe télécommande ou dans le groupe Rover. Chacun personne a ses points forts sur lesquels il est bon de miser.

ANEXES

Programme principale :

```
unsigned char OldP = 0x00;
unsigned char NewP = 0x00;
volatile signed char Diff;
volatile unsigned int AbsDiff;
volatile unsigned char Oscil;
unsigned char tram;
unsigned char Oldtram;

volatile unsigned int OscilTimer = 0x00;
volatile unsigned int DepTimer = 0x00;
unsigned char var = 0x00;

unsigned char ADC = 0x00;

extern unsigned char reset = 0x00;

void main(void) {

    /* Write your local variable definition here */

    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
    PE_low_level_init();
    /*** End of Processor Expert internal initialization. ***/

    /* Write your code here */

    TimerInit();                //Initialisation des Timer
    ServoInit();                //Position 0 du servomoteur

    //tram = 0b10101110;        //exemple de tram
    Oldtram = 0x00;

    for(;;) {

        SensRotation(tram, &NewP, OldP, &Diff, &AbsDiff, &Oscil);
        Savetram (&tram, &Oldtram, Oscil);
        ResetServo (&reset);
    }
}
```

Tourelle.c :

```
void TimerInit (void){
    TPM2MOD= 0x20C5;           //Limite du compteur (0.5ms)
    TPM2COSC = 0x00;           //Config du compteur 1(pas d'utilisation des channel)
    TPM2SC = 0x28;             //Config compteur 1(activation)

    TPM3C5SC = 0x28;           //PWM
    TPM3MOD = 0xA3D7;
    TPM3C5V = 0xC4A;           // PWM en position neutre
    TPM3SC = 0x2A;

    PTADD_PTADD0 = 0;          //Configuration Surconsommation
    PTCDD_PTCDD4 = 0;          //Configuration Origin moteur
}

void ServoInit (void){
    while (PTCD_PTCD4)         //Tant que le port correspondant à la position 0 est à 1
    {
        TPM3C5V = 0x0831;      //Sens antihorraire
    }
    TPM3C5V = 0x0C4A;          //point milieux
}

void ResetServo (unsigned char* reset){
    if (PTAD_PTAD0){           //Si le bit de surconsommation est à 1
        *reset = 0xFF;         //On le signal à l'ordonnanceur
    }
}

void SensRotation (unsigned char tram,unsigned char* NewP, unsigned char OldP, signed char* Diff, unsigned int* AbsDiff, unsigned char* Oscil){
    *NewP =(tram & 0x70)>>4;    //Masquage sur la tram

    if (OldP != *NewP){         //Si la position doit changer
        *Diff = OldP - *NewP;   //On calcule la différence
        if (*Diff > 0x00){       //Si différence positive
            *AbsDiff = *Diff;    //valeur absolue
        }else if(*Diff < 0x00){
            *AbsDiff = - (*Diff); //valeur absolue
        }
        *Oscil = 0xFF;          //On change de cas dans l'interruption TIMER1.
    }
    else {
        *Oscil = 0x00;
    }
}

void Savetram (unsigned char* tram, unsigned char* Oldtram, unsigned char Oscil){
    if (*tram != *Oldtram && Oscil == 0xFF){ //Si la tourelle est en déplacement et que la tram change
        *tram = (*Oldtram & 0x70)+(*tram & 0x8F); //Alors on l'empêche de changer.
    }
}
```

Tourelle.h :

```
void TimerInit (void);
void ServoInit (void);
void ResetServo (unsigned char*);
void SensRotation (unsigned char,unsigned char*, unsigned char, signed char*, unsigned int* , unsigned char*);
void Savetram (unsigned char*, unsigned char*, unsigned char);
```

Interruption :

```
extern unsigned char OldP;
extern unsigned char NewP;
extern signed char Diff;
extern unsigned char AbsDiff;
extern unsigned char Oscil;
extern unsigned int OscilTimer;
extern unsigned int DepTimer;
extern unsigned char var;
extern unsigned char Oldtram;
extern unsigned char tram;

ISR(TMP1OVERFLOW)
{
    // NOTE: The routine should include the following actions to obtain
    // correct functionality of the hardware.
    //
    // The TPM1SC register should be read and
    // TOF bit must be set to 0 to clear the interrupt request.
    // Example: TPM1SC;
    // TPM1SC_TOF = 0;

    if(Oscil == 0x00){
        ++ OscilTimer;
        if (OscilTimer == 0x1BB){
            var = ~(var);
            if (var==0xFF){
                TPM3CSV = 0x0831;
            }
            else if (var==0x00){
                TPM3CSV = 0x1062;
            }
            OscilTimer = 0x00;
        }
    }
    if (Oscil == 0xFF){
        ++ DepTimer;
        if (Diff > 0x00 && DepTimer < (AbsDiff*0x11D)){
            TPM3CSV = 0x0831;
        }
        else if (Diff < 0x00 && DepTimer < (AbsDiff*0x11D)){
            TPM3CSV = 0x1062;
        }
        else if (DepTimer >= (AbsDiff*0x1BB)){
            Oscil = 0x00;
            OldP = NewP;
            DepTimer = 0x00;
            Oldtram = tram;
        }
    }

    TPM1SC_TOF = 0;
}
```

Tests I2C :

```
/* User includes (#include below this line is not maintained by Processor Expert) */
const unsigned char Register[]={
    0x89, //lecture
    0x89,
    0x88,
    0x80,
    0xFF,
    0x00,
    0x91, //lecture
    0x00,
    0xFF,
    0x80,
    0x60, //lecture
    0x60,
    0x44, //emission 2 octets(0x0020)
    0x01,
    0x80,
    0xFF,
    0x00,
    0xFF,
    0x83, //lecture
    0x83,
    0xFF,
    0x81,
    0x80,
    0x94,
    0x83,
    0x83, //lecture
    0x83,
    0x92, //lecture
    0x81,
    0xFF,
    0x83, //lecture
    0x83,
    0xFF,
    0x00,
    0xFF,
    0x80,
    0xB0, //lecture 6 octets
    0xFF,
    0x4F,
    0x4E,
    0xFF,
    0xB6,
    0xB0, //Emission 6 octets
    0xFF,
    0x00,
    0xFF,
    0x09,
    0x10,
    0x11,
    0x24,
    0x25,
    0x75,
```

0x75 ,
0xFF ,
0x4E ,
0x48 ,
0x30 ,
0xFF ,
0x30 ,
0x54 ,
0x31 ,
0x32 ,
0x40 ,
0x46 ,
0x60 ,
0x27 ,
0x50 ,
0x51 ,
0x52 ,
0x56 ,
0x57 ,
0x61 ,
0x62 ,
0x64 ,
0x65 ,
0x66 ,
0xFF ,
0x22 ,
0x47 ,
0x49 ,
0x4A ,
0xFF ,
0x7A ,
0x7B ,
0x78 ,
0xFF ,
0x23 ,
0x42 ,
0x44 ,
0x45 ,
0x46 ,
0x40 ,
0x0E ,
0x20 ,
0x43 ,
0xFF ,
0x34 ,
0x35 ,
0xFF ,
0x31 ,
0x4B ,
0x4C ,
0x4D ,
0xFF ,
0x44 ,
0x45 ,
0x47 ,
0x48 ,

```
0x67,  
0x70,  
0x71,  
0x72,  
0x76,  
0x77,  
0xFF,  
0x0D,  
0xFF,  
0x80,  
0x01,  
0xFF,  
0x8E,  
0x00,  
0xFF,  
0x80,  
0x0A,  
0x84, //lecture  
0x84,  
0x0B,  
0x01, //lecture  
0x50, //lecture  
0x46, //lecture  
0x51, //lecture 2 octets  
0x70, //lecture  
0x71, //lecture 2 octets  
0x01,  
0x01, //lecture  
0x50, //lecture  
0x46, //lecture  
0x51, //lecture 2 octets  
0x70, //lecture  
0x71, //lecture 2 octets  
0x71, //Emission 2 octets  
0x01,  
0x00,  
-};  
  
const unsigned char Data[]={  
    0x01,  
    0x00,  
    0x01,  
    0x01,  
    0x00,  
    0x01,  
    0x00,  
    0x00,  
    0x00,  
    0x12,  
    0x00,0x20, //Envois 2octets (0x44)  
    0xFF,  
    0x01,  
    0x01,  
    0x00,  
    0x06,  
    0xDD,
```

```
0x01,
0x00,
0x06,
0xDD,
0x07,
0x01,
0x01,
0x6B,
0x00,
0x01,
0x00,
0x06,
0xDD,
0x01,
0x01,
0x00,
0x00,
0x01,
0x00,
0x01,
0x00,
0x2C,
0x00,
0xB4,
0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, //Emission 6 octets (0xB0)
0x01,
0x00,
0x00,
0x00,
0x00,
0x00,
0x01,
0xFF,
0x00,
0x01,
0x2C,
0x00,
0x20,
0x00,
0x09,
0x00,
0x04,
0x03,
0x83,
0x25,
0x00,
0x00,
0x06,
0x00,
0x96,
0x08,
0x30,
0x00,
0x00,
0x00,
0x00,
0xA0,
- -
```

```
0x01,  
0x32,  
0x14,  
0xFF,  
0x00,  
0x00,  
0x0A,  
0x00,  
0x21,  
0x01,  
0x34,  
0x00,  
0xFF,  
0x26,  
0x05,  
0x40,  
0x06,  
0x1A,  
0x40,  
0x00,  
0x03,  
0x44,  
0x01,  
0x04,  
0x09,  
0x05,  
0x04,  
0x00,  
0x00,  
0x20,  
0x08,  
0x28,  
0x00,  
0x04,  
0x01,  
0xFE,  
0x00,  
0x00,  
0x01,  
0x01,  
0x00,  
0x01,  
0xF8,  
0x01,  
0x01,  
0x01,  
0x00,  
0x00,  
0x04,  
0x01,  
0x01,  
0xE8,  
0x02, 0x94, //Emission 2 octets (0x71)
```

```
        0x01,  
        0x41,  
};  
  
unsigned char index=0;  
unsigned char Adresse=0x52;  
unsigned char *address=&Adresse;  
unsigned char *regis=&Register[0];  
unsigned char *data=&Data[0];  
unsigned char *data2=0;  
unsigned char *data3=0;  
unsigned char *data4=0;  
unsigned char *data5=0;  
unsigned char *data6=0;  
unsigned char dummy=0;  
unsigned char receive_data=0x1e;  
unsigned char receive_test=0x13;  
unsigned char test=0;  
unsigned char high_range=0;  
unsigned char low_range=0;  
  
/*  
  
unsigned char Address_laser = 0x52;  
unsigned char *address = &Address_laser;  
unsigned char dummy=0;;  
  
const unsigned char preregister []={0x89, 0x84, 0x60};  
unsigned char *reg=0;  
unsigned char *val=0;  
unsigned char i=0;;  
unsigned char index=0;  
  
unsigned char data=0x91;  
unsigned char receive_test=0x13;  
unsigned char receive_data=0x1E;  
unsigned char test=0;  
  
unsigned char *stockage=0;  
  
extern unsigned char *array=0;  
unsigned char low_range=0;  
unsigned char high_range=0;  
  
const unsigned char register_laser[]={  
    0x88,    //0x00  
    0x80,    //0x01  
    0xFF,    //0x01  
    0x00,    //0x00  
    //Lire 0x91 et retenir  
    0x00,    //0x01  
    0xFF,    //0x00  
    0x80,    //0x00      I2C standard mod
```

```
0x01, //0xFF (SYSTEM_SEQUENCE_CONFIG)

0xFF, //0x01
0x4F, //0x00 (DYNAMIC_SPAD_REF_EN_START_OFFSET)
0x4E, //0x2C (DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD)
0xFF, //0x00
0xB6, //0xB4 (GLOBAL_CONFIG_REF_EN_START_SELECT)
//----- tunning
0xFF, //0x01
0x00, //0x00

0xFF, //0x00
0x09, //0x00
0x10, //0x00
0x11, //0x00

0x24, //0x01
0x25, //0xFF
0x75, //0x00

0xFF, //0x01
0x4E, //0x2C
0x48, //0x00
0x30, //0x20

0xFF, //0x01
0x30, //0x09
0x54, //0x00
0x31, //0x04
0x32, //0x03
0x40, //0x83
0x46, //0x25
0x60, //0x00
0x27, //0x00
0x50, //0x06
0x51, //0x00
0x52, //0x96
0x56, //0x08
0x57, //0x30
0x61, //0x00
0x62, //0x00
0x64, //0x00
0x65, //0x00
0x66, //0xA0

0xFF, //0x01
0x22, //0x32
0x47, //0x14
0x49, //0xFF
0x4A, //0x00

0xFF, //0x00
```

```
0xFF, //0x01
0x22, //0x32
0x47, //0x14
0x49, //0xFF
0x4A, //0x00
```

```
0xFF, //0x00
0x7A, //0x0A
0x7B, //0x00
0x78, //0x21
```

```
0xFF, //0x01
0x23, //0x34
0x42, //0x00
0x44, //0xFF
0x45, //0x26
0x46, //0x05
0x40, //0x40
0x0E, //0x06
0x20, //0x1A
0x43, //0x40
```

```
0xFF, //0x00
0x34, //0x03
0x35, //0x44
```

```
0xFF, //0x01
0x31, //0x04
0x4B, //0x09
0x4C, //0x05
0x4D, //0x04
```

```
0xFF, //0x00
0x44, //0x00
0x45, //0x20
0x47, //0x08
0x48, //0x28
0x67, //0x00
0x70, //0x04
0x71, //0x01
0x72, //0xFE
0x76, //0x00
0x77, //0x00
```

```
0xFF, //0x01
0x0D, //0x01
```

```
0xFF, //0x00
0x80, //0x01
0x01, //0xF8
```

```

        0xFF, //0x01
        0x8E, //0x01
        0x00, //0x01
        0xFF, //0x00
        0x80, //0x00

        //-----

        0x0A, //0x04 (SYSTEM_INTERRUPT_CONFIG_GPIO)
        0x0B, //0x01 (SYSTEM_INTERRUPT_CLEAR)

        0x01, //0xE8 (SYSTEM_SEQUENCE_CONFIG)
        0x01, //0x01 (SYSTEM_SEQUENCE_CONFIG)

        0x01, //0x02 ou 0x08 (SYSTEM_SEQUENCE_CONFIG)

        //Start continuous

        0x80, //0x01
        0xFF, //0x01
        0x00, //0x00
        0x91, //Ancienne valeur de 0x91
        0x00, //0x01
        0xFF, //0x00
        0x80, //0x00
        0x00 // 0x02 (SYSRANGE_START)

};

unsigned char data_laser[]={
    0x00,
    0x01,
    0x01,
    0x00,
    //Lire 0x91 et retenir
    0x01,
    0x00,
    0x00, //I2C standard mod

    0xFF, //(SYSTEM_SEQUENCE_CONFIG)

    0x01,
    0x00, //(DYNAMIC_SPAD_REF_EN_START_OFFSET)
    0x2C, //(DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD)
    0x00,
    0xB4, //(GLOBAL_CONFIG_REF_EN_START_SELECT)
    //----- tunning
    0x01,
    0x00,

```


0x00,
0x00,
0x00,
0x00,

0x01,
0xFF,
0x00,

0x01,
0x2C,
0x00,
0x20,

0x01,
0x09,
0x00,
0x04,
0x03,
0x83,
0x25,
0x00,
0x00,
0x06,
0x00,
0x96,
0x08,
0x30,
0x00,
0x00,
0x00,
0x00,
0xA0,

0x01,
0x32,
0x14,
0xFF,
0x00,

0x00,
0x0A,
0x00,
0x21,

0x01,
0x34,
0x00,
0xFF,
0x26,
0x05,
0x40,
0x06,
0x1A,
0x40,

```
0x00,  
0x03,  
0x44,  
  
0x01,  
0x04,  
0x09,  
0x05,  
0x04,  
  
0x00,  
0x00,  
0x20,  
0x08,  
0x28,  
0x00,  
0x04,  
0x01,  
0xFE,  
0x00,  
0x00,  
  
0x01,  
0x01,  
  
0x00,  
0x01,  
0xF8,  
  
0x01,  
0x01,  
0x01,  
0x00,  
0x00,  
  
//-----  
  
0x04, //(SYSTEM_INTERRUPT_CONFIG_GPIO)  
0x01, //(SYSTEM_INTERRUPT_CLEAR)  
  
0xE8, //(SYSTEM_SEQUENCE_CONFIG)  
0x01, //(SYSTEM_SEQUENCE_CONFIG)  
  
0x02, //ou 0x08 (SYSTEM_SEQUENCE_CONFIG)  
  
//Start continuous  
  
0x01,  
0x01,  
0x00,  
0x00, //Ancienne valeur de 0x91  
0x01,  
0x00,  
0x00,  
0x02 //(SYSRANGE_START)
```

```
void main(void)
{
    /* Write your local variable definition here */

    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
    PE_low_level_init();
    /*** End of Processor Expert internal initialization. ***/

    /* Write your code here */
    /* For example: for(;;) { } */
    /*reg = &preregister[0];

    //lire VH_V_CONFIG_PAD_SCL_SDA__EXTSUP_HV (0x89) et | 0x01
    dummy = I2C_RegRead(adress,reg);
    dummy = dummy | 0x01;
    val = &dummy;
    I2C_RegWrite(adress,reg,val);

    //lire GPIO_HV_MUX_ACTIVE_HIGH (0x84) et faire & ~0x10
    reg = &preregister[1];
    dummy = Samuel(adress,reg);
    dummy = dummy & ~0x10;
    val = &dummy;
    I2C_RegWrite(adress,reg,val);

    //lire MSRC_CONFIG_CONTROL (0x60) et faire | 0x12
    reg = &preregister[2];
    dummy = I2C_RegRead(adress,reg);
    dummy = dummy | 0x12;
    val = &dummy;
    I2C_RegWrite(adress,reg,val);

    reg = &register_laser[0];
    val = &data_laser[0];

    while (index != 106){

        if (reg == &register_laser[4]){
            stockage=reg;
            reg=&data;
            dummy = I2C_RegRead(adress,reg);
            data_laser[65]= dummy;
            reg=stockage;
        }

        I2C_RegWrite(adress,reg,val);

        reg ++;
        val ++;
        index++;
    }
    */
}
```

```

for(;;) {

    if (index!=142){
        if (index==0 || index==6 || index==10 || index==18 || index==25 || index==27 ||
            index==30 || index==124 || index==127 || index==128 || index==129 || index==131 || index==134 || index== 135 || index==136 || index==138){
            // lecture de 1 octet
            dummy=Read_one_octet(address,regis);
        }
        else if (index==130 || index==132 || index==137 || index==139){
            //lecture de 2 octets
            I2C_RegRead(address,regis);
        }
        else if (index==34){
            //lecture 6 octets
            Read_six_bytes(address,regis);
        }
        else if (index==12 || index==140){
            //émission de 2 octets
            data2=data+1;
            Write_two_bytes(address,regis,data,data2);
            data=data2;
        }
        else if (index==42){
            //émission de 6 octets
            data2=data+1;
            data3=data2+1;
            data4=data3+1;
            data5=data4+1;
            data6=data5+1;
            Write_six_bytes(address,regis,data,data2,data3,data4,data5,data6);
            data=data6;
        }
        else {
            I2C_RegWrite(address,regis,data);
        }
        index++;
        data++;
        regis++;
    }
    else{
        regis=&receive_test;
        test=Read_one_octet(address,regis);
        if (test == 0x44){
            regis=&receive_data;
            I2C_RegReadinfini(address,regis);

            low_range;
            high_range;
            asm(
                NOP
            );
        }
    }
}
    
```

I2C.c

```
unsigned int timeout=0;
unsigned char b=0;
unsigned char Read=0;
unsigned int timer=0;

void Write_two_bytes(unsigned *address, unsigned char *reg, unsigned char *val1, unsigned char *val2){
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_Write(*val1); // Send Value
    I2C_Write(*val2); // Send Value
    I2C_Stop(); // Send Stop
}

void Write_six_bytes(unsigned char *address, unsigned char *reg, unsigned char *val1, unsigned char *val2, unsigned char *val3, unsigned char *val4, unsigned char *val5, unsigned char *val6){
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_Write(*val1); // Send Value
    I2C_Write(*val2); // Send Value
    I2C_Write(*val3); // Send Value
    I2C_Write(*val4); // Send Value
    I2C_Write(*val5); // Send Value
    I2C_Write(*val6); // Send Value
    I2C_Stop(); // Send Stop
}

void Read_six_bytes (unsigned char *address, unsigned char *reg){
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_RepeatStart(); // Send Repeat Start
    I2C_Write(*address+0x01); // Send IIC "Read" Address
    b=I2C_Read(0); // Dummy read starts read transaction
    b=I2C_Read(0); // Read Register Value
    b=I2C_Read(0);
    b=I2C_Read(0); // Read Register Value
    b=I2C_Read(0);
    b=I2C_Read(0);
    b=I2C_Read(1);
    I2C_Stop(); // Send Stop
}

void I2C_Start(){
    IICCL_MST=1; // Mise en marche de l'I2C
    while (!IICS_BUSY);
}
```

```
void I2C_Stop(){
    IICC1_MST=0; //Arret de l'I2C
    timeout=0;
    while (IICS_BUSY && (timeout<500)){
        ++timeout;
    }
}

void I2C_RepeatStart(){
    IICC1_RSTA=1; //Repetition du start
    while (!IICS_BUSY);
}

void I2C_Write (unsigned char write){
    timeout=0;
    IICC1_TX=1;
    while ((!IICS_TCF) && (timeout<500)){
        ++timeout;
    }
    IICD=write;
    timeout=0;
    while ((!IICS_IICIF) && (timeout<500)){
        ++timeout;
    }
    IICS_IICIF=1;
    timeout=0;
    while (IICS_RXAK && (timeout<500)){
        ++timeout;
    }
}

unsigned char I2C_Read (unsigned char NACK){
    timeout=0;
    IICC1_TX=0;
    while ((!IICS_TCF) && (timeout<500)){ //Test si reception de données
        ++timeout;
    }
    IICC1_TXAK=NACK; //Activation du No Acknowledge
    Read=IICD; //Stockage de la donnée
    timeout=0;
    while((!IICS_IICIF) && (timeout<500)){
        ++timeout;
    }
    IICS_IICIF=1;
    return Read; //Retour au sous programme
}
```

```
void I2C_RegWrite(unsigned char *address, unsigned char *reg, unsigned char *val)
{
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_Write(*val); // Send Value
    I2C_Stop(); // Send Stop
    while (timer!=0x3FF){
        timer++;
    }
    timer=0;
}

unsigned char I2C_RegRead(unsigned char *address, unsigned char *reg)
{
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_RepeatStart(); // Send Repeat Start
    I2C_Write(*address+0x01); // Send IIC "Read" Address
    b = I2C_Read(0); // Dummy read starts read transaction
    b = I2C_Read(0); // Read Register Value
    b = I2C_Read(1);
    I2C_Stop(); // Send Stop
    while (timer!=0x3FF){
        timer++;
    }
    timer=0;
    return b;
}

unsigned char Read_one_octet (unsigned char *address, unsigned char *reg){
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_RepeatStart(); // Send Repeat Start
    I2C_Write(*address+0x01); // Send IIC "Read" Address
    b = I2C_Read(0); // Dummy read starts read transaction
    b = I2C_Read(0); // Dummy read starts read transaction
    b = I2C_Read(1); // Read Register Value
    I2C_Stop(); // Send Stop
    while (timer!=0x3FF){
        timer++;
    }
    timer=0;
    return b;
}
```

```
unsigned char Samuel (unsigned char *address, unsigned char *reg){
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_RepeatStart(); // Send Repeat Start
    I2C_Write(*address+0x01); // Send IIC "Read" Address
    b = I2C_Read(0); // Dummy read starts read transaction
    b = I2C_Read(0); // Read Register Value
    b = I2C_Read(0);
    b = I2C_Read(1);
    I2C_Stop(); // Send Stop
    while (timer!=0x3FF){
        timer++;
    }
    timer=0;
    return b;
}

/*
void I2C_RegReadN(unsigned char *address, unsigned char *reg)
{
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_RepeatStart(); // Send Repeat Start
    I2C_Write(*address+0x01); // Send IIC "Read" Address
    *array= I2C_Read(0); // Dummy read
    *array= I2C_Read(0);
    ++array;
    *array = I2C_Read(1);
    I2C_Stop(); // Send Stop
    while (timer!=0x3FF){
        timer++;
    }
    timer=0;
}

*/
void I2C_RegReadinfini(unsigned char *address, unsigned char *reg)
{
    I2C_Start(); // Send Start
    I2C_Write(*address); // Send IIC "Write" Address
    I2C_Write(*reg); // Send Register
    I2C_RepeatStart(); // Send Repeat Start
    I2C_Write(*address+0x01); // Send IIC "Read" Address
    high_range= I2C_Read(0); // Dummy read
    high_range= I2C_Read(0);
    low_range = I2C_Read(1);
    I2C_Stop(); // Send Stop
    while (timer!=0x3FF){
        timer++;
    }
    timer=0;
}
```


I2C.h

```
extern unsigned char low_range;
extern unsigned char high_range;

void I2C_Start(void);
void I2C_Stop(void);
void I2C_RepeatStart(void);
void I2C_Write(unsigned char);
unsigned char I2C_Read(unsigned char);
void I2C_RegWrite(unsigned char *, unsigned char *, unsigned char *);
unsigned char I2C_RegRead(unsigned char *, unsigned char *);
unsigned char Samuel(unsigned char *, unsigned char *);
unsigned char Read_one_octet (unsigned char *, unsigned char *);
void I2C_RegReadN(unsigned char *, unsigned char *);
void I2C_RegReadinfini(unsigned char *, unsigned char *);
void Write_six_bytes(unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *, unsigned char *);
void Write_two_bytes(unsigned *, unsigned char *, unsigned char *, unsigned char *);
void Read_six_bytes (unsigned char *, unsigned char *);
```

Note : Certains fonctions du programme I2C sont nommés de façon temporaire, de façon à affecter des tests.