

Robot holonome Robocup

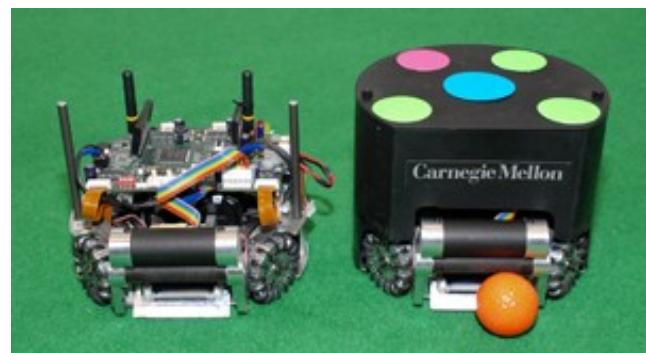
Gestion déplacements

Clément LAVERGNE

2017-2018

Sous la supervision de :

- MICLO Aloïs
- Steve N'Guyen



Ynov Aéronautique et systèmes embarqué
89 Quai des Chartrons 33000 Bordeaux

RÉSUMÉ

Ce rapport traitera d'un aspect de la conception d'un robot footballeur holonomique correspondant aux normes de la Robocup. Il relatera les recherches effectuées dans le cadre du projet de fin d'année sur 3 mois. Nous nous intéressons plus particulièrement la partie de gestion des déplacements aux travers l'utilisation de diverses capteurs et de PID. La conception de ce robot s'est fait dans les locaux de l'école YNOV aéronautique avec à notre disposition, PC, cartes d'évaluation avec micro-processeur MC9S08QE8 ainsi qu'un atelier de fabrication de carte électronique. Il s'agissait de la première itération d'un projet de plus grande envergure englobant tous les élèves actuels de l'école dans l'objectif de proposer un projet aboutie pour la Robocup 2020 se déroulant à Bordeaux. Le cahier des charges à était revu à minima au vu du temps et de l'équipe réduite (4 personnes) nous n'avons pas eu à traiter la partie intelligence artificielle pour la gestion de l'équipe.

Nous avons utilisé des capteurs optiques de type ADNS-3080 ainsi qu'un module MPU6050 pour observer et mesurer les déplacements du robot. Les informations ont été traitée sur un PC de contrôle avec un processus de PID sur les axes X et Y et sur la rotation. Nous avons également été amené à déterminer les puissances à envoyer aux moteurs quant à l'obtention de la direction et de la rotation souhaitées avec également un paramètre de vitesse.

Certains problèmes rencontrés ont empêchés de mener à bien le projet, la réception des valeurs en provenance des capteurs est fonctionnelle, les processus de PID ont été développés mais non calibrés, de plus les capteurs optiques et d'accélération n'ont pas pu être testés en condition réelles. Ce rapport pose de solides bases pour l'exploitation de tous les systèmes dont il traite sans être entré en phase de test pourtant crucial dans tout projet.

SOMMAIRE

Sommaire.....	3
Tables des illustrations.....	4
Introduction.....	5
I.Protocoles de communication	6
A.I2C.....	6
B.SPI.....	9
II.Registres utiles du micro-contrôleur	16
A.Broches.....	16
B.Timer.....	17
C.Portabilité du code vers QE 128.....	18
III.Communications au sein de la partie	19
IV.ADNS – Capteur optique	21
A.Présentation du module.....	21
B.Branchement du module.....	21
C.étude de registre.....	23
D.Utilisation et mise en place.....	24
E.Carte connecteur.....	26
V.Centrale inertielle	28
A.Choix.....	28
B.Présentation du module.....	28
C.Étude de registre.....	29
D.Mise en place.....	31
VI.PID	32
A.Principe.....	32
B.Mise en œuvre.....	33
C.Calibration.....	34
VII.Calculs pour les déplacements	36
A.Calculs théoriques.....	36
B.Exemple pratique.....	40
C.Mise en place.....	40
Conclusion.....	42
Bibliographie.....	43
Annexes.....	45

TABLES DES ILLUSTRATIONS

Index des illustrations

Logo RoboCup 2020.....	5
Exemple de trame I2C annotée.....	6
Représentation commande de restart.....	7
Représentation des commandes ACK et NACK.....	7
Câblage d'un système multi-esclaves.....	10
Variations des lignes en fonction de la polarité (CPOL) et de la phase (CPHA).....	10
Représentation de mode de communication expliqué.....	11
Schématisation de l'obtention de la fréquence SPI.....	13
Représentation d'une lecture sur l'esclave.....	15
Représentation des flux d'information de la partie.....	19
Capteur optique choisi.....	21
Vu de derrière du module.....	22
Connections conseillés pour le capteur.....	22
Représentation du placement des capteurs.....	24
Démonstration d'un capteur optique.....	25
Calcul de simplification de la conversion.....	26
Graphique de la sensibilité du capteur.....	26
face avant de la carte de connexion.....	27
face arrière de la carte de connexion.....	27
Schémas de la carte de connexion (une seul face).....	27
Photo du capteur choisi.....	28
Utilisation des bits DLPF_CFG.....	29
Démonstration pour la centrale.....	31
Principe du PID.....	32
Représentation du chemin des données au travers des calculs de PID.....	33
Exemple des variations de e(t) en fonction de Kp (facteur du paramètre proportionnel).....	35
Représentation des vecteurs unitaires des moteurs.....	36
Bref définition du produit scalaire.....	37
Formule du produit scalaire.....	37
Représentation fonction cosinus.....	37
Représentation du vecteur et de ses composantes. « Zoom » sur la représentation précédente.....	40

Index des tableaux

Système classique d'adressage.....	7
Séquence de communication pour une écriture sur le capteur.....	12
Séquence de communication pour une lecture sur le capteur.....	12
Choix des bits MODFEN et SSOE pour le réglage du fonctionnement du SS.....	17
Codage des valeurs de SPPRx.....	18
Codage des valeurs de SPRx.....	18
Listes des registres pour la gestion des broches.....	21
Tableau pour le choix des bits PS.....	22
Signification des différentes valeurs des bits FS_SEL.....	40
Signification des différentes valeur des bits AFS_SEL.....	40

INTRODUCTION

Nous devons réaliser un robot joueur de football pouvant participer à la Robotcup dans la catégorie f180. Sur les 4 mois où s'étend le projet il faudra mettre en place toute la partie matériel et logiciel d'un robot holonome commandé grâce à une manette, capable de se déplacer dans toutes les directions et d'effectuer des rotations sur lui-même. Enfin il doit être capable de garder la balle prêt de lui et de tirer toujours sous le contrôle de la manette.

Comme expliqué dans le rapport de groupe, je suis responsable de la partie déplacement du robot. J'ai donc la responsabilité de transformer les informations en provenance de la manette en informations qui actionneront les moteurs. Aussi il faudra déployer une série de capteurs pour ajuster la trajectoire et pour se renseigner sur les déplacements réels du robot.

Nous allons nous intéresser aux capteurs, au choix effectués, à leur fonctionnement ainsi qu'à leur mise en place logiquement et au sein du projet globale. Mais avant cela, nous étudierons le fonctionnement des protocoles de communication qu'ils utilisent. Ensuite de verront comment fonctionne les processus de PID et comment obtenir les paramètres utilisés pour mener à bien les différentes étapes. Bien-sûr les calculs nécessaires à l'obtention de déplacement seront expliqués dans un premier temps théoriquement puis dans leur mise en place logicielle.

Toute la parties emparquée est supportée par les micro-contrôleur de la famille NXP nous prendrons quelques lignes pour comprendre les fonctions utile des ses systèmes. Mais avant tout cela nous prendrons le temps de comprendre les communications au sein du robot et plus particulièrement celle passant au travers de notre partie afin de mieux comprendre l'enjeu de chaque chapitre du rapport.



Illustration 1: Logo RoboCup 2020

I. PROTOCOLES DE COMMUNICATION

A. I²C¹

Durant le projet nous avons été amené à utiliser le protocole de communication I²C pour échanger avec la centrale inertie. Dans cette partie nous allons étudier le fonctionnement de ce protocole ainsi que les registres du micro-contrôleur utiles à la mise en place de cette communication.

i. Principe

L'I²C est un protocole de communication synchrone inventé en 1982. Il nécessite seulement deux lignes nommées SDA² (ligne de donnée) et SCL³ (ligne d'horloge). A partir de deux lignes il est possible de communiquer avec de nombreux systèmes : il faut choisir un maître qui contrôlera les communications notamment en générant le signal d'horloge sur la ligne SCL. Les autres systèmes sont donc nommés esclave et sont différenciés par un système d'adresse. Le système d'adresse standard se fait sur 7 bits, un 8ème bit sélectionne si l'on souhaite lire ou écrire sur les registres de l'esclave avec lequel on communique.

I ² C address field								R/W'
Byte 1								
7	6	5	4	3	2	1	0	
7	6	5	4	3	2	1	0	
MSB					LSB	1 = Read		
						0 = Write		

Tableau 1: Système classique d'adressage

La communication I²C utilise des lignes dites « open-drain » ce qui signifie que lorsque qu'aucune action n'est en cours sur les lignes celle-ci sont alimentées et lorsque qu'un composant souhaite communiquer il va passer les lignes à la masse.

Le protocole I²C possède plusieurs combinaisons d'action codant pour des commandes précises (commencer la transmission, la finir, faire une pause etc.) nous allons en aborder certaine au travers d'un exemple :

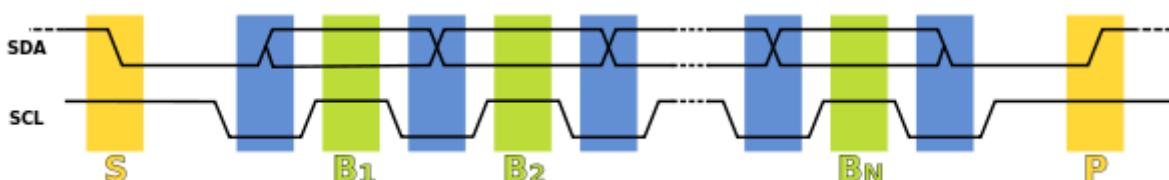


Illustration 2: Exemple de trame I²C annotée

1 Inter-Integrated Circuit

2 Serial Data Line

3 Serial Clock Line

toucher à la ligne SCL (action du maître)

Règle de transmission : le maître comme les esclaves doivent changer la valeur de la ligne SDA uniquement lorsque la ligne SCL est à l'état bas, ainsi des lectures peuvent être effectuées en toute confiance lorsque SCL est à l'état haut. Notons que cette règle est enfreinte par certaine commande (bit de start par exemple). C'est ainsi que l'on peut voir sur le schémas Bn bits être transmis.

Bit de STOP : repéré par un P sur le schémas le bit de stop est caractérisé par un relâchement de la ligne SDA (état haut) alors que la ligne SCL est à l'état haut.

Il existe aussi le bit de restart qui comme son nom l'indique permet d'éviter d'avoir un faire un bit de start suivi d'un bit de stop. Pour l'effectuer il suffit d'effectuer un bit de start après avoir ramené la ligne SCL à l'état haut.



Illustration 3: Représentation commande de restart

Enfin la majorité des systèmes utilisent un codage de bit de ACK et de NACK qui correspondent respectivement à un accusé de réception et à un non-accusé de réception, leurs usages diffèrent selon les systèmes.

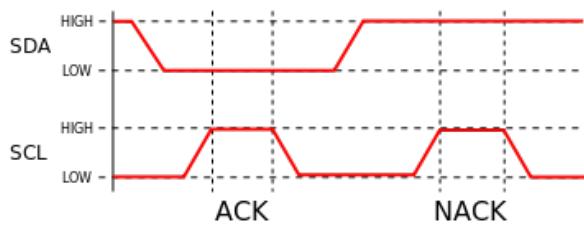


Illustration 4: Représentation des commandes ACK et NACK

ii. Étude de registre

Le micro-contrôleur que nous utilisons met à notre disposition des registres qui permettent d'automatiser la couche de base du codage de l'I2C, nous allons les étudier pour comprendre comment les utiliser.

Registre IICC1 – registre de contrôle 1

	7	6	5	4	3	2	1	0
R W	IICEN	IICIE	MST	TX	TXAK	0	0	0
Reset	0	0	0	0	0	0	0	0

bit n° :

7 IICEN : permet d'activer le module I2C

6 IICIE : Permet de déterminer si l'I2C à besoin d'une interruption

5 MST : Sélection du mode maître ou esclave

4 TX : Sélectionne le sens de propagation des informations

0 : mode réception

1 : mode transition

3 TXAK : Permet d'envoyer des ACK (0) et des non-ACK (1) en basculant le bit sur la bonne valeur

2 RSTA : Écrire un 1 sur ce bit permet de générer une commande restart

Registre IICS – registre d'états

	7	6	5	4	3	2	1	0
R	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
W					0	0	0	0

Reset 1 0 0 0 0 0 0 0 0

bit n° :

7 TCF : Ce bit permet de savoir si la transmission est terminée

0 : transfert en cours

1 : transfert terminé

6 IAAS : Ce bit est activé pour un usage du micro-contrôleur en esclave (ce qui n'est pas notre cas)

5 BUSY : Permet de savoir si le bus de transmission est occupé

0 : bus libre

1 : bus occupé

4 ARBL : Ce bit permet de gérer les pertes d'arbitrage notion que nous n'avons pas abordé car elle n'est utile que lorsque que plusieurs maîtres partagent la ligne.

2 SRW : Ce bit est activé pour un usage du micro-contrôleur en esclave (ce qui n'est pas notre cas)

1 IICIF : Permet de gérer les interruptions du module, nous ne nous en servirons pas.

0 RXAK : Ce bit permet de savoir si l'on a reçu une commande d'ACK de la part d'un esclave

0 : ACK reçu

1 : Pas d'ACK reçu

Registre IICD – registre de donnée

	7	6	5	4	3	2	1	0
R					DATA			
W								

Reset 0 0 0 0 0 0 0 0 0

Il faut écrire dans ce registre lorsque l'on est en mode TX pour que la donnée écrite soit envoyée. De la même manière, il s'agit du registre à lire en mode RX pour réceptionner des données.

iii. Mise en place

La mise en place de la communication I2C se fait sous forme de couche, du plus bas niveau au plus haut. La première couche que l'on avait déjà réalisée en « *bit-banging*⁴ » lors de cette année scolaire est gérée par le micro-contrôleur grâce aux registres, nous commençons donc par la couche supérieure. Dans un premier temps il faut coder les différentes commandes que nous avons évoquées, ces fonctions sont assez simple grâce aux registres à notre disposition. Le seul ajout par

⁴ Façon de coder une fonction de communication par basculement et lecture de borche I/O

rapport à ces bits de contrôle est la vérification de l'état du bus. La fonction init() met en application ce que nous avons vu plus tôt.

Enfin nous terminons pour cette couche avec les fonctions de lecture et d'écriture de la même manière ces deux fonctions reprennent les principes expliqués plus tôt. Nous avons également mis en place une variable nommée « status » qui permet de garder un œil sur les transmissions lorsque que l'on met en place le code. La gestion de cette variable prend un nombre important de lignes mais nous a fait gagner du temps lors des tests. Finalement, le paramètre nommé NACK permet de choisir d'envoyer un ACK ou un NACK suivant nos besoins.

La couche encore supérieure est constituée de deux autres fonctions pour la lecture et l'écriture I2C, cette fonction est adaptée au capteur, en effet on trouve dans la fiche technique le tableau suivant :

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Tableau 2: Séquence de communication pour une écriture sur le capteur

On suit donc les informations fournis par le constructeur, on commence par une commande de start grâce à la fonction prévue à cet effet. Il faut ensuite écrire l'adresse du capteur sur le bus avec un 1 en LSB pour signifier que souhaite écrire comme expliqué dans la partie sur l'I2C. On retrouve l'adresse du module dans la fiche technique : 110 1000 ou 110 1001 selon l'état de la broche AD0.

La suite du protocole de communication est assez basique, on suit chaque étape grâce à nos fonctions prévues à cet effet.

De la même manière on met en place la fonction de lecture à partir du tableau suivant :

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Tableau 3: Séquence de communication pour une lecture sur le capteur

B. SPI

i. Principe

Le protocole de communication SPI⁵ est également un protocole de communication synchrone qui utilise au minimum quatre lignes de communication. Nous avons utilisé le protocole SPI pour communiquer avec les capteurs optiques. De la même manière que le protocole I2C, il faut choisir un système maître qui va communiquer avec au moins un esclave. Chaque nouvel esclave nécessite une nouvelle ligne SS dans un câblage classique que nous utiliserons.

Comme on peut le remarquer, les trois lignes SCLK, MOSI et MISO sont communes à tous les esclaves. Ce nombre important de lignes permet des communications en full duplex⁶, regardons chacune de ces lignes :

⁵ Serial Peripheral Interface bus

⁶ Une communication full duplex permet aux deux acteurs de « parler » en même temps sans se gêner

- SCLK : il s'agit d'une ligne d'horloge générée par le maître sur laquelle toutes les communications vont être cadencées.
- MOSI (Master Out, Slave In) : il s'agit de la ligne où le maître va écrire les données à transférer
- MISO (Master In, Slave Out) : à l'inverse de la ligne MOSI, cette ligne va servir à l'esclave pour écrire et c'est ici que le maître va lire des données en provenance des esclaves.
- SSx (slave select) : Ces lignes servent à sélectionner l'esclave avec lequel on souhaite communiquer (de manière classique, un à la fois).

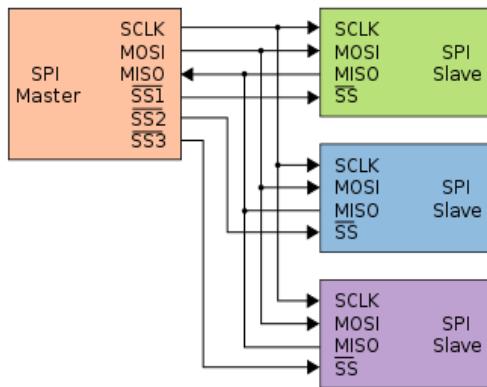


Illustration 5: Câblage d'un système multi-esclaves

Il existe deux paramètres importants pour réussir la communication : la polarité qui correspond à l'état par défaut de l'horloge et qui va donc décider de enchaînements des différentes positions. Le second paramètre est la phase, ce paramètre règle les moments où les lignes MISO et MOSI sont basculées ainsi que le moment où ils peuvent être lues par rapport à l'horloge. Les modifications apportées sont flagrantes au départ de la communication.

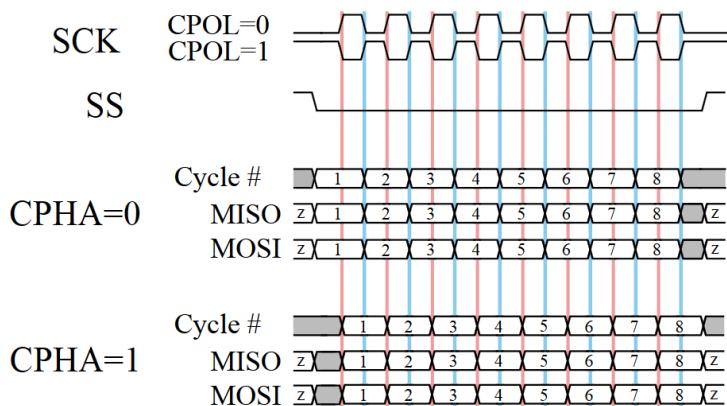


Illustration 6: Variations des lignes en fonction de la polarité (CPOL) et de la phase (CPHA)

La communication SPI se fait au travers d'un buffer circulaire⁷ commun entre le maître et l'esclave, ainsi bien que les deux communiquent en full-duplex uniquement un registre de donnée par système est nécessaire. Chaque système va donc écrire dans son registre la donnée qu'il veut

⁷ Un buffer circulaire est une manière de stoker des données. Il peut être représenté par un tableau dont la fin est suivie par le début du tableau lui-même.

transmettre. Ensuite les deux systèmes vont « échanger » le contenu de leur registre pour effectuer la transmission.

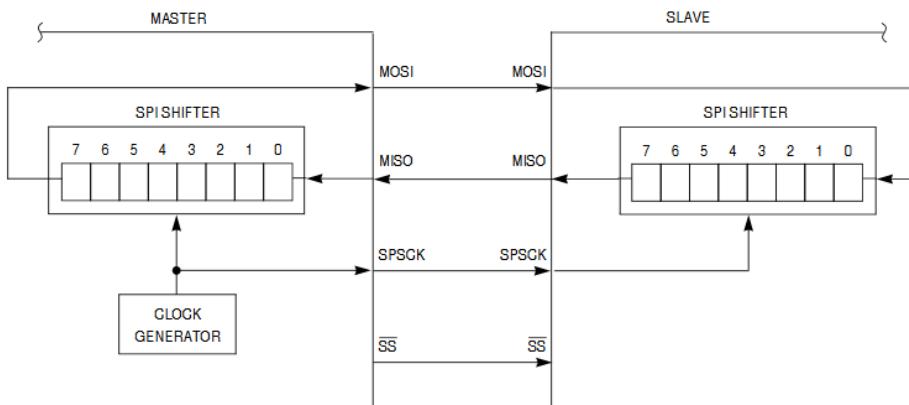


Figure 15-2. SPI System Connections

Illustration 7: Représentation de mode de communication expliquée

ii. Étude de registre

Registre SPIC1 – registre de contrôle 1

	7	6	5	4	3	2	1	0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset	0	0	0	0	0	1	0	0

Bit n° :

7 SPIE : Ce bit permet d'activer les interruptions générées lorsque le buffer de réception est plein, nous ne nous en servirons pas car notre usage de la SPI est plutôt d'aller interroger nous même les capteurs.

6 SPE : Ce bit est indispensable, il permet l'activation du système de SPI en y plaçant un 1.

5 SPTIE : Il s'agit d'un autre bit permettant d'utiliser une interruption, cette fois lorsque le drapeau SPTEF est à 1. Encore une fois, nous gérons ça manuellement.

4 MSTR : Permet de régler le micro-contrôleur en mode esclave ou maître

0 : Mode esclave

1 : Mode maître (ce que l'on choisit)

3 CPOL : Il s'agit du réglage pour le paramètre de polarité dont nous avons déjà parlé

0 : horloge active à l'état haut

1 : horloge active à l'état bas

2 CPHA : De la même manière il s'agit du réglage du paramètre de phase

0 : le premier changement sur l'horloge intervient au milieu du premier bit transféré

1 : le premier changement sur l'horloge intervient au début du premier bit transféré

1 SSOE : Ce bit est utilisé en le combinant avec le bit MODFEN (registre suivant) pour régler le mode de fonctionnement de la ligne SS.

0 LSBFE : Permet de sélectionner l'ordre dans lequel un bit est transféré

0 : bit de poids fort en premier

1 : bit de poids faible en premier

registre SPIC2 – registre de contrôle 2

	7	6	5	4		3	2	1	0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0	
W									
Reset	0	0	0	0	0	0	0	0	0

Bits n° :

4 MODFEN : Ce bit est utilisé en le combinant avec le bit SSOE (registre précédent) pour régler le mode de fonctionnement de la ligne SS suivant le tableau suivant :

MODFEN	SSOE	Master Mode
0	0	General-purpose I/O (not SPI)
0	1	General-purpose I/O (not SPI)
1	0	SS input for mode fault
1	1	Automatic SS output

Tableau 4: Choix des bits MODFEN et SSOE pour le réglage du fonctionnement du SS

Nous allons régler les deux bits sur 0, en effet après quelques tests il semble que la génération automatique du SS pose problème, peut-être n'est-elle pas adapter à notre module. Générer manuelle le SS est assez simple et résout le problème.

3 BIDIROE : Aucun effet dans notre cas

1 SPISWAI : Permet de savoir si l'on souhaite arrêter l'horloge lorsque le micro_contrôleur est en mode d'attente

0 SPC0 : Permet d'utiliser la SPI uniquement avec un fil pour transférer dans les deux sens, option que nous n'utilisons pas.

0 : usage d'un canal bidirectionnel

1 : usage classique de deux canaux

Registre SPIBR – Réglage du diviseur d'horloge

	7	6	5	4		3	2	1	0
R	0	SPPR2	SPPR1	SPPR0	SPR3	SPR2	SPR1	SPR0	
W									
Reset	0	0	0	0	0	0	0	0	0

Ce registre est utilisé pour régler la fréquence de l'horloge qui sera utilisée pour la communication SPI et qui donc décide de la vitesse de communication. Le calcul de cette fréquence prend en compte trois variables :

- La fréquence de l'horloge de micro-contrôleur qui sert de base.

Le prescaler, la valeur de l'horloge principale va être divisor par cette valeur. Il est réglé par les bits 6 à 4 de ce registre suivant le tableau suivant :

SPPR2:SPPR1:SPPR0	Prescaler Divisor
0:0:0	1
0:0:1	2
0:1:0	3
0:1:1	4
1:0:0	5
1:0:1	6
1:1:0	7
1:1:1	8

Tableau 5: Codage des valeurs de SPPRx

- Le baud rate divisor, la fréquence obtenue par le calcul précédent va à son tour être divisée par cette valeur codée par les bit 3 à 0 de ce registre suivant la règle suivante :

SPR3:SPR2:SPR1:SPR0	Rate Divisor
0:0:0:0	2
0:0:0:1	4
0:0:1:0	8
0:0:1:1	16
0:1:0:0	32
0:1:0:1	64
0:1:1:0	128
0:1:1:1	256
1:0:0:0	512
All other combinations	512

Tableau 6: Codage des valeurs de SPRx

On peut résumer les calculs précédents par un schéma :

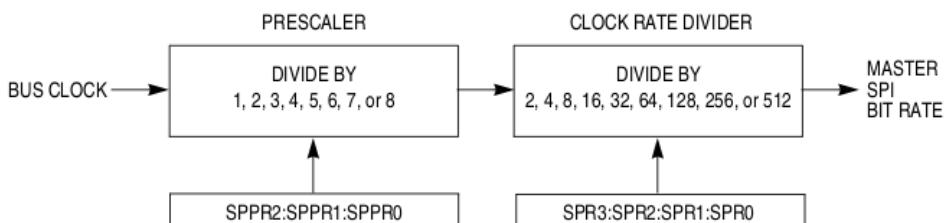


Illustration 8: Schématisation de l'obtention de la fréquence SPI

Registre SPIS – registre d'états

	7	6	5	4	3	2	1	0
R	SPRF	0	SPTEF	MODF	0	0	0	0
W								
Reset	0	0	1	0	0	0	0	0

Ce registre en lecture seul contient des drapeaux qui vont nous informer sur l'état de la communication. Il est important de noter que la lecture de ces bits à de réelles répercussions sur les communications.

Bit n° :

7 SPRF : Permet de savoir si il y a des valeurs à lire dans le buffer de donnée

0 : Rien à lire

1 : Une nouvelle donnée disponible

5 SPTEF : Ce bit permet de savoir si le buffer de donnée est vide et donc, disponible pour une écriture

0 : Nouvelle écriture impossible

1 : nouvelle lecture possible

4 MODF : Ce bit passe à 1 lorsque qu'un autre maître est détecté sur la ligne, ce qui ne nous arrivera pas.

Registre SPID – registre de donnée

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W	0	0	0	0	0	0	0	0

Reset

Il s'agit du registre où l'on va écrire ce que l'on veut communiquer aux esclaves et où l'on va lire pour obtenir les données en provenance des esclaves. Il y a quelques règles pour l'utilisation de ce registre que nous avons déjà abordé plus tôt :

- Il faut s'assurer que le bit SPTEF est à 1 avant d'écrire
- Il faut lire le bit SPRF et qu'il soit à 1 avant toute lecture

iii. Mise en place

La mise en place des fonctions SPI est en théorie un peu plus simple car nous avons moins à gérer le bas niveau qu'avec l'I2C (pas commande à créer etc...). La gestion de la SPI par notre micro-contrôleur fait qu'une seule fonction suffira pour coder les communications.

Bien que les fonctions une fois terminées paraissent très simples nous nous sommes heurtés à des difficultés pour mettre en place la communication. La mise en place d'une communication en full-duplex est compliquée et nous avons mis du temps à appréhendé le fonctionnement du micro-contrôleur. La principale difficulté été dût au fait qu'un esclave ne peut pas parler si le maître ne génère pas d'horloge, or le maître ne génère une horloge qui s'il est en train d'envoyer. Autre problème qui a agi en parallèle : le registre SPID, pierre angulaire de la communication ne peut pas recevoir de nouvelle valeur tant que l'on ne lit pas ce qu'il contient, hors, lorsque l'on effectue une écriture l'esclave abaisse le MISO ce qui est interprété par le micro-contrôleur comme une donnée (ce qui pourrait l'être en full-duplex) il faut donc lire le registre SPID après toute écriture pour vider le registre.

La solution trouvée est de renoncé à certains aspects du full-duplex, en effet nous avons mis en place une communication suivant ce prototype :

- Pour une écriture on commence par écrire sur le bus le registre dans lequel on veut écrire sur l'esclave, puis on lit le registre SPID pour le vider. On écrit ensuite la valeur que l'on souhaite écrire avant de lire une fois de plus le SPID pour le vider.

Pour une lecture il n'y a qu'une différence, lors de la seconde lecture du registre SPID nous gardons la valeur qui correspond à ce qui nous intéresse.

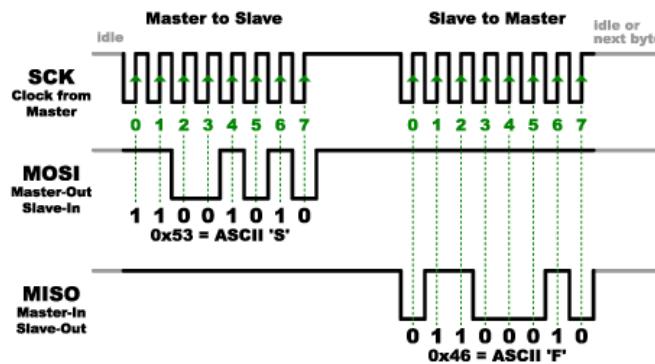


Illustration 9: Représentation d'une lecture sur l'esclave

Au vu de la similitude des protocoles de lecture et d'écriture une seule fonction fera l'affaire, on utilise un tableau de deux caractères pour les deux valeurs que l'on va écrire sur le bus, on mettra la deuxième à 0x00 pour une lecture. Le paramètre ‘read’ permettra de récupérer la valeur lu lors de la deuxième lecture du registre SPID et sera ignorée en cas d’écriture.

La dernière étape est de basculer la valeur des broches de sélection d’esclave avant et après la fonction.

II. REGISTRES UTILES DU MICRO-CONTRÔLEUR

A. Broches

La fonction la plus basique d'un micro-contrôleur est certainement la gestion de broche aux état allumé/éteint, deux familles de registre sont à notre disposition pour les usages les plus simples.

Registre PTAD – Données du port A

	7	6	5	4	3	2	1	0
R W	PTAD7	PTAD6	PTAD5 ¹	PTAD4 ²	PTAD3	PTAD2	PTAD1	PTAD0
Reset:	0	0	0	0	0	0	0	0

Ce registre permet de lire et de basculer les broches du port A.

Registre PTADD – Direction des données

	7	6	5	4	3	2	1	0
R W	PTADD7	PTADD6	PTADD5 ¹	PTADD4 ²	PTADD3	PTADD2	PTADD1	PTADD0
Reset:	0	0	0	0	0	0	0	0

Ce registre permet de configurer le sens d'utilisation des broches du port A, un 0 codant pour une broche en entré (lorsque l'on veut lire des valeurs) et un 1 codant une broche en sortie (lorsque l'on veut contrôler l'état une broche).

Ce que l'on explique ici pour le port A fonctionne de la même manière pour les port B,C et D qui possède des registres de nom proche comme on peut le voir ici :

Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x0000	PTAD	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0x0001	PTADD	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
0x0002	PTBD	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
0x0003	PTBDD	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
0x0004	PTCD	PTCD7	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
0x0005	PTCDD	PTCDD7	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
0x0006	PTDD	0	0	0	0	PTDD3	PTDD2	PTDD1	PTDD0
0x0007	PTDDD	0	0	0	0	PTDDD3	PTDDD2	PTDDD1	PTDDD0

Tableau 7: Listes des registres pour la gestion des broches.

Il est évidemment important de vérifier qu'une broche n'est pas utilisée par certain module du micro-contrôleur avant d'en faire usage, au risque de dérégler certaines fonctions.

B. Timer⁸

Un timer est un compteur qui débute en général de 0 et qui va s'incrémenter au rythme de l'horloge du micro-contrôleur, on peut changer le rythme d'incrémentation à l'aide de diviseur qui vont comme leur nom l'indique diviser le rythme de l'horloge pour l'incrémentation du timer. On peut ensuite choisir une valeur limite à laquelle le timer va effectuer l'action (souvent une interruption) manuellement ou le laisser se déclencher une fois arrivé à sa valeur max (une marge de sécurité est appliquée).

Registres utiles pour mettre en place un timer :

TPMxSC état et registre de contrôle

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
W	0							
Reset	0	0	0	0	0	0	0	0

Figure 16-7. TPM Status and Control Register (TPMxSC)

Bit n° :

7 : TOF → Il s'agit d'un drapeau qui passe à 1 lorsque que le timer arrive à son terme.

6 : TOIE → Lorsque le bit est activé, le timer déclenchera une interruption lorsque TOF passera à 1

5 : CPWMS → Permet de passer le timer en mode PWM, pas ce que l'on veut faire.

4-3 : CLKS → Permet de choisir la source d'horloge pour le timer, 01 correspondant l'horloge interne du micro-contrôleur

2-0 : PS → Permet de régler le diviseur de fréquence qui sera appliqué à l'horloge pour l'incrémentation du timer suivant la règle suivante :

Table 16-5. Prescale Factor Selection

PS[2:0]	TPM Clock Divided-by
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Tableau 8: Tableau pour le choix des bits PS

Plus on souhaite obtenir un timer qui s'active peu souvent, plus on choisira un diviseur important.

TPMxCNT (H et L)

Ce double registre (sur 16bits donc) permet de stocker la valeur actuelle du compteur, elle ne peut être modifiée directement dans ce registre, juste lu, ce qui ne présente pas une fonctionnalité intéressante dans notre cas.

⁸ Timer se traduit par minuteur mais cette terminologie n'est pas utilisée.

TPMxMOD (H et L)

Ce double registre de 16 bites va servir à indiquer la valeur à laquelle on souhaite que le timer s'arrête et se remette à zéro. Plus cette valeur sera importante, plus la période du timer sera longue.

TPMxCnSC Registre de configuration

	7	6	5	4	3	2	1	0
R	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
W	0							
Reset	0	0	0	0	0	0	0	0

Bit n° :

7 :ChnF → drapeau pour informer d'événement « input capture » ou « output compare » que nous n'utilisons pas.

6 : ChnIE → Permet d'activer le déclenchement d'une interruption lors de la remise à zéro du timer
 5-2 → Registre pour les diverses configuration et utilisation du timer résumé dans un tableau,
 l'usage assez simple de notre timer nous fait laisser tous ces bits à 0.

C. Portabilité du code vers QE 128

Tous les registres présentés ici ainsi que tous ceux qui le seront concernant le micro-contrôleur portent sur le modèle MC9S08QE8 que nous utilisons sur les carte d'évaluation à notre disposition. Nous avions prévu d'utiliser un MC9S08QE128 sur le robot sur la carte centrale principalement pour bénéficier deux modules SPI. Après avoir étudier la fiche technique de ce dernier, la portabilité du code semble assez simple, en effet presque que tous les registres dont j'ai besoin sont analogue sur la version 128bits. Les seules différences notables concernent les registre SPI où il faut juste ajouter le numéro du module concerné. Par exemple le registre SPID devient SPI1D ou SPI2D. La portabilité pourra donc être assuré sans problème.

III. COMMUNICATIONS AU SEIN DE LA PARTIE

Les principales communications au sein du robot ont probablement déjà été abordées dans la partie commune, il paraît néanmoins indispensable de détailler les flux d'informations passant au travers de cette partie. Cette explication permettra de comprendre les tenants et aboutissants de chaque partie de ce rapport. Nous allons décrire le cheminement de l'information de la manette jusqu'au déplacement de robot. Notons que l'on s'intéresse uniquement à cette partie du projet, de plus, nous allons ici uniquement survoler des notions plus au moins complexes qui seront expliquées dans les parties suivantes.

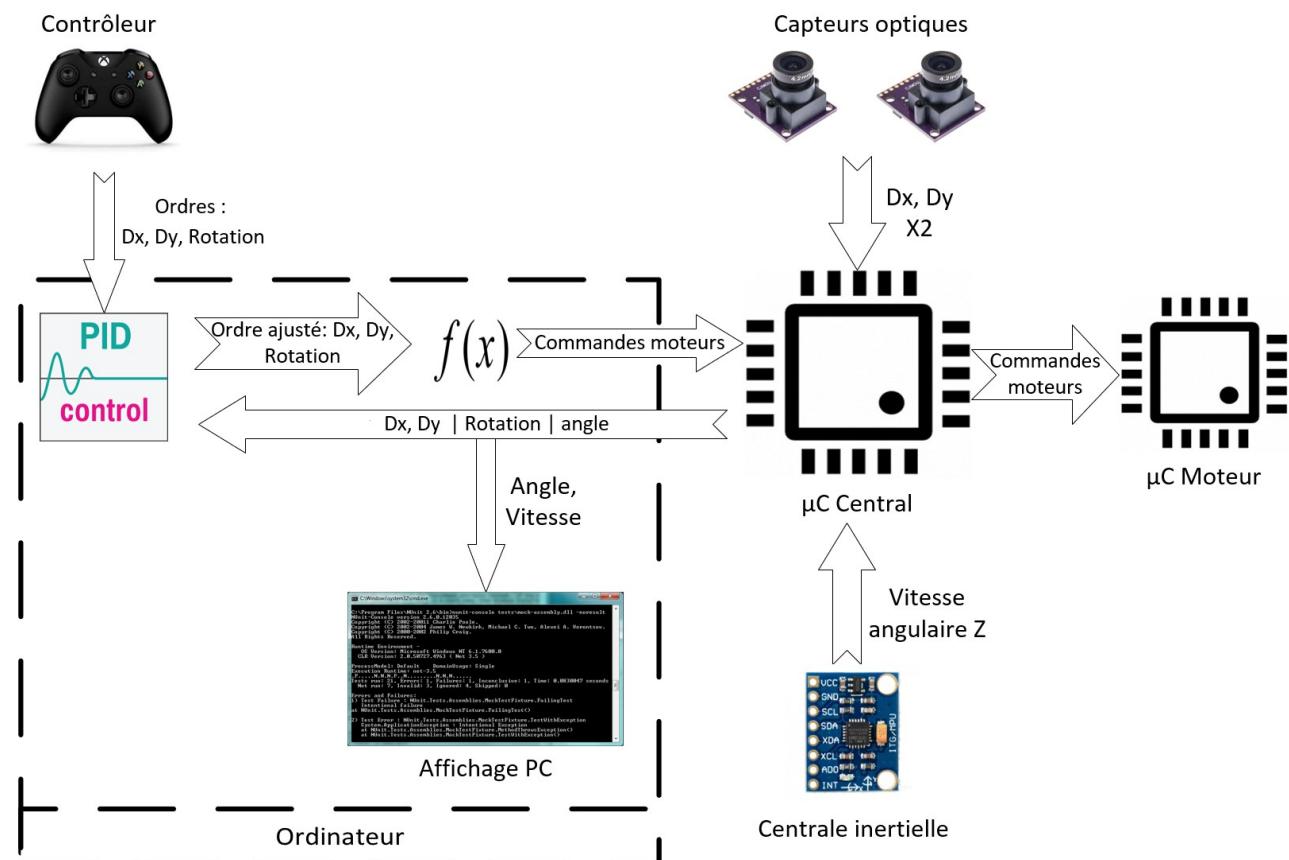


Illustration 10: Représentation des flux d'information de la partie

Comme évoqué précédemment le début de la chaîne de traitement de l'information débute par un vecteur dx, dy ainsi qu'une valeur de rotation en provenance de la manette. Il s'agit de l'ordre utilisateur pour les déplacements du robot, ces trois données vont toutes passer au travers d'un processus de PID qui va les ajustées pour atteindre la commande plus rapidement et efficacement (chaque processus à besoin d'une seconde entrée, nous reviendrons à la fin de cette partie). Les trois processus de PID vont fournir une nouvelle valeur pour chaque composante : dx, dy et rotation.

Ces trois valeurs vont ensuite être traitées pour obtenir quatre nouvelles données correspondant aux puissances souhaitées de chaque moteur du robot. Toutes les actions précédentes

se déroulent au sein du PC de contrôle. C'est à partir de là que l'on quitte le PC, les valeurs vont être envoyées par radio au micro-contrôleur central du robot.

Dans un souci de précision nous avons défini un standard de conversion des données de la manette et des commandes moteur : une valeur de 128 pour une commande moteur correspond à 55 tour/s de la roue soit 10m/s. De la même manière une valeur 32767 envoyée par la manette correspond à cette vitesse max théorique.

Le micro-contrôleur central va alors faire office de relais et va communiquer ces valeurs au micro-contrôleur dédié au moteur. Aussi le composant sert de relais avec le micro-contrôleur du kiteur, transmettant l'information de tir et de position de la balle.

Sur le micro-contrôleur central sont directement relié trois capteurs : deux capteurs optiques qui scrutent le sol et fournissent un déplacement ainsi qu'une centrale inertuelle qui communique des informations d'accélération et de vitesse angulaire. Toutes les informations en provenance des capteurs sont transmises au PC de contrôle.

Sur le PC ces informations après traitement vont être remis sur la même échelle que les entrés provenant de la manette, elles servent de deuxième entré aux processus de PID.

La représentation suivante résume ce qui a été expliqué plus tôt, nous avons omis les informations hors de la partie que nous traitons dont nous sommes les relais. La vitesse angulaire obtenue grâce à la centrale inertuelle est utilisé pour obtenir l'angle absolu de robot, dit autrement, la direction dans laquelle il regarde.

IV. ADNS⁹ – CAPTEUR OPTIQUE

Afin d'obtenir les déplacements réels du robot ce dernier sera équipé de deux capteurs optiques comparables à ceux présent dans les souris optiques, ces capteurs fourniront au pc de contrôle en temps réel les déplacements du robot.

Lorsque l'on s'intéresse au marché des souris optiques et plus particulièrement aux capteurs utilisés, la majorité semble appartenir à la gamme ADNS, il s'agit de capteurs produits par Aligent largement démocratisés. Néanmoins après quelques recherches, bien que ces capteurs soient courant dans le monde de l'industrie il est en même temps très difficile de s'en procurer pour le grand public.

C'est ainsi que nous avons trouvé un article sur Amazon contenant un module ADNS :

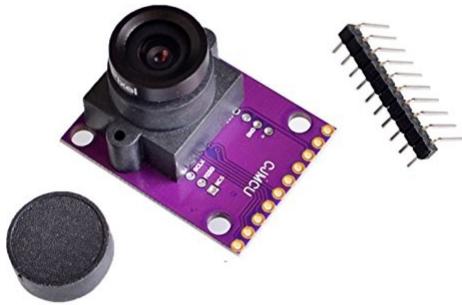


Illustration 11: Capteur optique choisi

Avantages :

1. Le module est vendu avec un objectif assez qualitatif qui pourrait permettre une très bonne qualité de mesure.
2. La carte est déjà faite avec une qualité de fabrication industrielle.
3. La documentation semble indiquer que la mise en œuvre sera assez simple grâce à des registres « dx,dy » qu'il suffit de lire.

Inconvénients :

1. Le prix, plus de 15€ par module
2. Des tests seront nécessaires pour déterminer si l'objectif est adapté à notre utilisation, en effet il semble prévu pour de longues distances
3. Pas de fiche technique pour le module

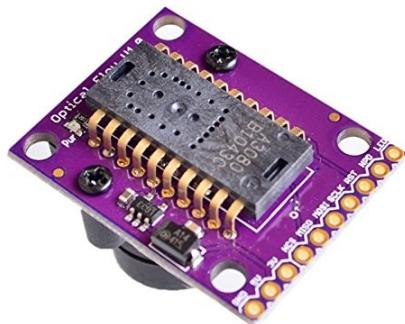
A. Présentation du module

Le module présent sur le composant est donc un ADNS-3080 un capteur très performant pouvant mesurer jusqu'à 100cm par seconde. Le fonctionnement théorique est relativement simple à comprendre, le capteur prend des séries de cliché (jusqu'à 9400 images par seconde) grâce au capteur haute résolution de 900pixels, c'est à partir de ces images que le capteur va mathématiquement définir le mouvement observé.

B. Branchement du module

L'absence de fiche technique pour le module que nous avons commandé ne sera pas un problème majeur étant donné que les broches du module sont annotées :

⁹ Automated Digital Network System, il s'agit d'une gamme de capteurs.



Comme nous pouvons le voir sur cette image le module possède les broches suivante :

GND – 3V – 5V – NCS – MISO – MOSI – SCLK – RST – NPD - LED

Illustration 12: Vu de derrière du module

De base le capteur possède 20 broches, on trouve dans la fiche technique un câblage conseillé :

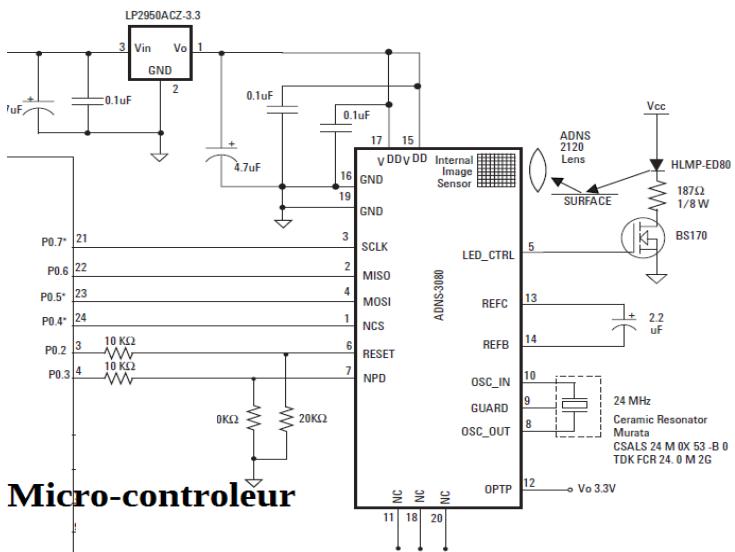


Illustration 13: Connexions conseillés pour le capteur

On retrouve donc toutes les broches précédemment citées. La connexion des broches : GND – 3V – 5V – NCS – MISO – MOSI – SCLK sera assez triviale, nous auront juste à les relier aux broches correspondantes sur notre micro-contrôleur en notant que l'on branche au choix l'une des broches de tentions.

La broche RST doit être reliée à une sortie I/O de notre micro-contrôleur pour permettre de remettre à zéro le capteur en cas de besoin.

La broche NPD permet de mettre ou non le module en marche, l'éteindre pourrait permettre d'économiser de la batterie sur notre robot mais nous ne pouvons-nous permettre de stopper les modules tant les informations qu'ils procurent sont indispensable. Le fonctionnement de cette broche est un peu particulier, en effet lorsque la broche est alimentée (3V) le module est mis en marche et lorsqu'elle ne l'est plus (GND) le module est mise en veille. Par commodité on relira cette broche au VCC.

Enfin la broche LED permet de relier le module à une LED, cette dernière permettra d'éclairer la surface à observer. Relié la LED au capteur a un avantage, le capteur peut gérer la LED pour l'allumer uniquement en cas de besoin. Dans notre cas, la consommation d'une LED est assez

négligeable, nous utiliserons une LED mais elle sera directement connectée au VCC pour limiter les sources de dysfonctionnement.

C. étude de registre

On peut trouver tous les registres du capteur sur une page, malgré cela, peut d'entre eux nous seront utiles, nous allons donc nous recentrer sur ceux-là.

0x00 Product_ID - 0x01 Revision_ID - 0x3f Inverse Product ID

Ces trois registres ne sont pas utilisés dans la version finale du code, néanmoins ils ont été utile pour les tests préliminaires de la SPI pour vérifier notre capacité à lire des registres au travers de la SPI.

0x02 Motion

Access: Read	Reset Value: 0x00							
Bit	7	6	5	4	3	2	1	0
Field	MOT	Reserved	Reserved	OVF	Reserved	Reserved	Reserved	RES

Ce registre possède deux bits qui vont nous intéresser :

- MOT : Ce bit passe à 1 lorsqu'un mouvement a été détecté depuis la dernière lecture, lorsqu'il est relevé à 1, cela signifie que l'on peut aller lire les registre DX et DY pour récupérer des données.
 - OVF : Ce bit passe à 1 lorsque l'un des registre DX, DY (ou les deux) n'a pas été lu a temps causant un dépassement de pile

0x03 Delta_X & 0x04 Delta_Y

Access: Read Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0

Ces deux registres correspondent aux données qui nous intéressent, y est stocké la valeur signée sur huit bits des derniers déplacements. La signification de la valeur est déterminée par la résolution.

0x0a Configuration_bits

Access: Read/Write		Reset Value: 0x09							
Bit	7	6	5	4	3	2	1	0	
Field	0	LED_MODE	Sys Test	RES	Reserved	Reserved	Reserved	Reserved	Reserved

Il s'agit d'un registre assez classique et simple de configuration

- LED_MODE : Comme expliqué dans la partie connexion la LED est gérée indépendamment du capteur
 - Sys Test : 0 → On ne met pas en place le système de test simplement car nous n'avons pas le temps de lire des variables de CRC car nous devons préserver une vitesse de communication importante

- RES Permet de régler la résolution du capteur
 - 0 : 400
 - 1 : 1600

La résolution correspond au nombre (400 ou 1600) que donnera un déplacement de un pouce (2,54cm), dans notre cas nous prendrons une résolution de 400 afin d'éviter au maximum tout dépassement de pile.

D. Utilisation et mise en place

Ce capteur sera utilisé en deux exemplaires, un premier sera placé au centre du robot et nous indiqueront les déplacements sur les axes X et Y. Le second sera placé le plus à l'extérieur possible, au niveau des roues, aligné sur l'axe X avec le premier capteur. Ce capteur nous permettra de déduire la rotation effectuée par le robot. En effet, un capteur seul ne peut nous communiquer les informations de rotation, nous obtiendrons cette donnée simplement en soustrayant les valeurs de déplacement DY des deux capteurs, si cette soustraction est nulle, alors le robot a effectué un déplacement simple (ou pas de déplacement), sinon on obtient la valeur de cette rotation.

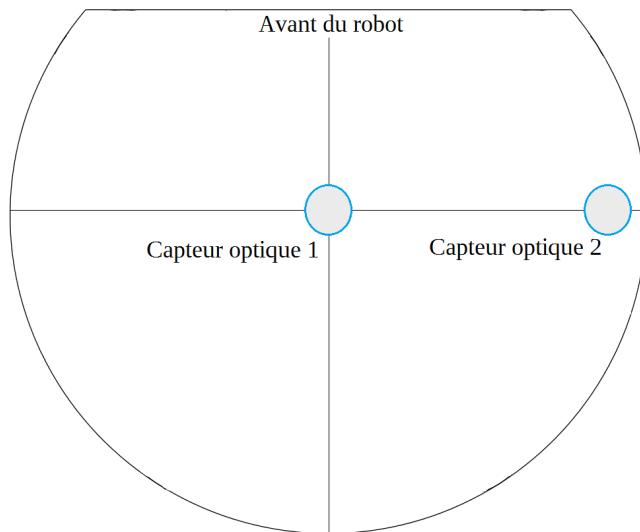


Illustration 14: Représentation du placement des capteurs

Nous allons donc mettre en place une communication SPI avec les deux capteurs en utilisant toutes les lignes en commun et deux broches pour la sélection de l'esclave avec lequel on souhaite communier¹⁰.

Fonction d'initialisation : Une fois que les fonctions de communication SPI fonctionne et avant de commencer à récupérer les déplacements du robot il faut configurer le capteur, cela est assez simple une fois l'étude de registre effectuée, nous allons donc configurer nos modules (un par un) grâce à une fonction d'initialisation¹¹.

Une fois l'initialisation effectuée la lecture de donnée est assez simple, il suffit de lire le registre motion qui va nous indiquer grâce au bit n°7 si il y a eu un mouvement, si il n'y a pas eu de mouvement on retourne 0, si il y a bien eu un mouvement nous allons donc lire les registres delta_X (0x03) et delta_y (0x04) pour obtenir le déplacement qui nous intéresse sur les deux capteurs, on

10 Voir partie SPI page 10

11 Voir annexe page 57

obtient donc trois valeurs : delta_X et delta_Y pour le capteur central pour les déplacement, enfin on obtient la valeur de rotation grâce à la soustraction expliquée plus tôt.

Il est important d'effectuer des relevés à intervalles réguliers, en effet, les données seront notamment utilisées pour l'asservissement en vitesse du robot, or, lorsque l'on s'intéresse à une vitesse le facteur temps est forcément très important c'est pour cela que l'on met en place un timer¹² qui déclenchera les lectures à intervalles réguliers.

Ces trois valeurs ainsi obtenues seront envoyées au PC de contrôle où elles seront traitées. Leur traitement sera double, dans un premier temps elles seront utilisées pour connaître la position absolue du robot, pour cela il suffit de stocker la somme des valeurs en provenance du capteur centrale après les avoir converties en cm. On se retrouvera donc avec deux valeurs, la somme des delta_X et la somme des delta_Y, ces deux sommes représentent donc le déplacement du robot depuis sa position de démarrage ce qui fait partie de notre cahier des charges.

La seconde utilisation des valeurs de ces capteurs sera l'asservissement en position, je reviendrais plus dans le détail dans la partie dédiée mais avant cela il faut savoir que pour que cela fonctionne il faut convertir l'information du capteur en valeur qui s'étend sur un 'signed short'¹³.

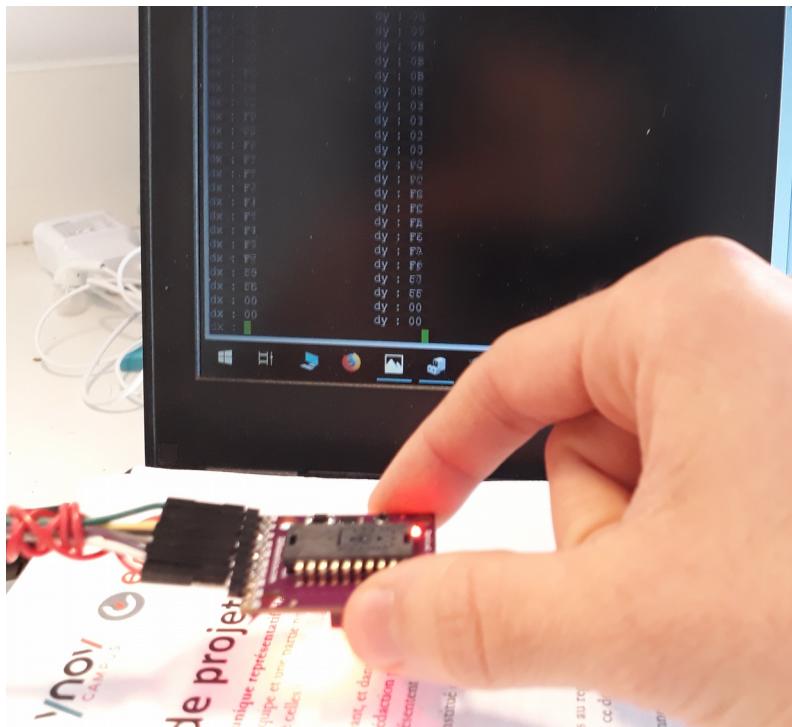


Illustration 15: Démonstration d'un capteur optique

Illustration 15: Démonstration d'un capteur optique

Objectif : Transformer les valeurs des capteurs en données sur la même échelle que celles produites la manette

Nous recevons donc une valeur sur 8 bits signés. Comme expliqué dans l'étude de registre un '400'(décimal) reçu correspond à un pied, soit 2,54cm. On converti donc facilement la donnée

12 cf. page 18

13 Signed short : Format de stockage de donnée sur 16 bits qui prend en compte les valeur négatives, ce format permet d'exprimer des valeurs dans l'intervalle [-32 767, +32 767]

en centimètre puis en mètre. De plus on connaît la fréquence échantillonnage grâce au timer mis en place, en divisant notre valeur en mètre par notre temps échantillonnage en seconde. Enfin on multiplie cette valeur par $\frac{1}{T_{\text{éch}}}$. Nous obtenons ainsi la vitesse de notre robot. La dernière étape

est de remettre cette valeur de vitesse sur un signed short à l'échelle de la manette. Comme expliqué dans la partie communications, la valeur 32 767 (valeur max) correspond à une vitesse de 10m/s.

$$\begin{array}{ccc} 32\,767 & x \\ \text{Rapide produit en croix :} & & \text{on obtient donc notre valeur à comparer pour} \\ 10\,m/s & V_{\text{calculé}} & \\ \end{array}$$

notre PID en multipliant notre vitesse par 32767/10.

Dans le code définitif¹⁴ il faut condenser tous les calculs précédents composés uniquement de variable que l'on connaît (hors la valeur d'entrée évidemment) en un seul facteur dans un objectif de rapidité. On note x la valeur provenant du capteur et y la valeur de sortie de la fonction de conversion :

$$y = \frac{x * 0,0254}{\frac{400}{0,006} * 3276,7} = \frac{x * 0,20807045}{0,006} = x * 34,678$$

Illustration 16: Calcul de simplification de la conversion

E. Carte connecteur

L'utilisation des capteurs doit se faire aussi dans le cadre matériel, on sait que les deux capteurs seront connectés à la carte du micro-contrôleur mais ils doivent être en place sur le sol du robot. De plus les capteurs optiques au besoin de lumière pour fonctionner, on trouve dans la fiche technique du capteur la bande de fréquence à laquelle le capteur est sensible.

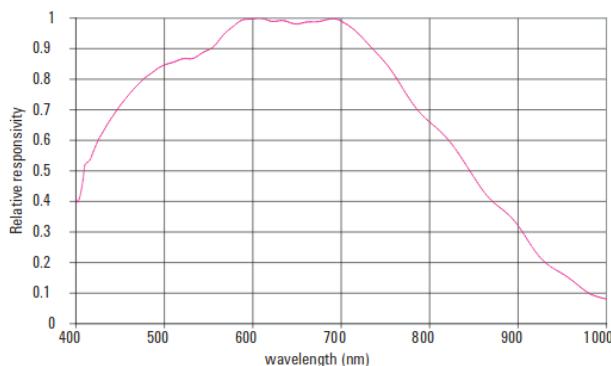


Illustration 17: Graphique de la sensibilité du capteur

14 Voir page 49 code PID

Nous allons donc utiliser une LED rouge par capteur pour éclairer le sol où l'on mesure les déplacements. C'est dans cette optique que nous avons développé une petite carte de connexion pour relier notre capteur optique à la carte centrale à l'aide d'une nappe ainsi qu'un branchement pour une LED¹⁵. La carte ainsi effectuée ressemble à cela :

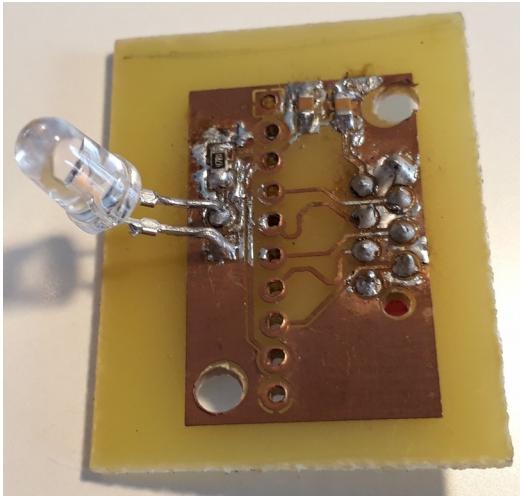


Illustration 18: face avant de la carte de connexion

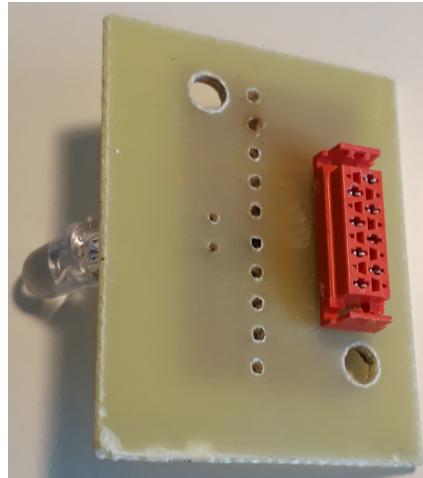


Illustration 19: face arrière de la carte de connexion

L'utilisation de cette carte est comme celle-ci assez basique, le connecteur rouge sur la face arrière doit être relié à la carte centrale. On branche le module de capteur optique à l'aide d'un header droit depuis la face arrière en mettant le capteur du côté de la LED. Nous avons également récupéré des lentilles de souris optique qui pourrait servir à remplacer l'objectif déjà installé qui semble plus adapté à de longue distance de captation. Il aurait fallu pouvoir tester directement sur un robot fini mais nous n'en avons pas eu l'occasion, de plus j'aurais aimé fournir une image avec le capteur en place sur la carte mais nous n'avons jamais reçu nos capteurs.

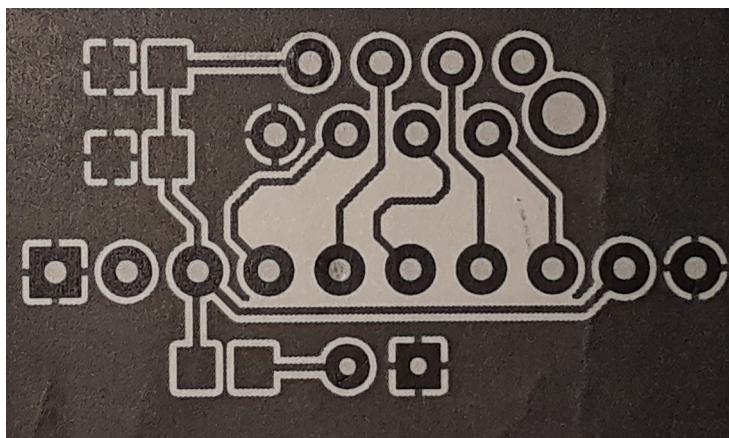


Illustration 20: Schémas de la carte de connexion (une seul face)

15 Voir schémas de la carte ci-dessus

V. CENTRALE INERTIELLE

A. Choix

En plus des capteurs optiques, il nous été demandé de mettre en place une centrale inertie, ce genre de dispositif sont utilisés pour récupérer des informations d'accélération et de vitesse angulaire. Le choix du dispositif que nous avons acheté fût assez rapide : nous avons choisi le MPU-6050, les raisons qui ont motivé ce choix sont multiple : c'est un module très populaire dans le monde de l'informatique embarquée, on retrouve de très nombreux exemple d'utilisation et de codes qui pourraient nous être utile en cas de problème, il est assez petit et prêt à l'emploi, en effet il nous suffira de souder le 'header' à notre futur carte. La communication se fait par I2C, qui est un protocole que nous avons étudié, de plus, nous tenions à éviter un nouvel appareil communiquant en SPI de peur d'être à court de canal SPI sur le micro-contrôleur. Enfin, lorsque l'on s'intéresse à la fiche technique il semble être simple d'utilisation pour un coût réduit.



Illustration 21: Photo du capteur choisi

B. Présentation du module

Cette centrale inertie possède six axes de mesures, trois axes de gyroscope qui permettent d'obtenir une vitesse de rotation sur chaque axe et trois axes accélémètre qui permettent d'obtenir une accélération (en g) sur chaque axe. Tous ces axes de mesure peuvent être configurés de manière différente : de la mesure la plus précise à la mesure sur la plus grande gamme de valeur, jusqu'à 2000°/s pour les gyroscopes et jusqu'à 16g pour les accélémètres. Toutes ces données sont communiquées par I2C à une vitesse de 400kHz. Aussi le capteur possède un capteur de température qui ne nous sera pas utile.

Le branchement du module est assez basique : les broches Vcc et GND seront logiquement reliés respectivement à une tension 3V et à la masse. Les broches pour la communication I2C seront reliées directement aux broches analogues du micro-contrôleur. La broche INT ne nous sera pas utile, bien que cette broche peut fournir des informations qui pourraient nous intéresser comme le fait qu'une donnée est prête, il paraît plus simple de ne pas en tenir compte. Les broches XDA et XCL sont prévues pour la gestion d'un oscillateur externe ce que nous n'utilisons pas. Enfin la broche AD0 permet de changer l'adresse du module lorsque qu'il est en mode esclave (le mode que l'on utilise) plus précisément, relier cette broche à la masse donnera l'adresse suivante au module :

b1101000 alors que relier AD0 à une tentions 3V donnera cette adresse ci : b1101001, par soucis de commodité nous avons choisi la première option.

C. Étude de registre

registres 0x0D à 0x10 : Registre de tests

Ces registres sont utilisés en lecture pour s'assurer du fonctionnement mécanique des différents capteurs du module.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]		DLPF_CFG[2:0]			

Registre 0x1A : configurationEXT_SYNC_SET : 0 → On déactive la fonctionnalité car elle nécessite l'utilisation d'une pin que l'on ne retrouve pas sur le module que l'on a acheté.

DLPF_CFG : Ces bits permettent le réglage de filtres qui vont s'appliquer sur les accéléromètres et gyroscopes. Changer ses bites permet de régler les paramètres de bande passante et de délai de réponse de la manière suivante :

DLPF_CFG	Accelerometer (Fs = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Illustration 22: Utilisation des bits DLPF_CFG

Registre 0x18 : configuration des gyroscope s

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Les bits n° 7, 6 et 5 permettent l'activation des procédures d'auto-test respectivement pour chaque gyroscope.

FS_SEL : Permet de régler la précision ainsi que l'étendu des mesures pour les gyroscopes : plus la mesure est précise et moins on est capable de mesurer de grandes valeurs. Le réglage suit la règle suivante :

FS_SEL	Full Scale Range
0	± 250 °s
1	± 500 °s
2	± 1000 °s
3	± 2000 °s

Tableau 9: Signification des différentes valeurs des bits FS_SEL

Registre 0x1C – Configuration des accéléromètres

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]			-	

Les bits n° 7, 6 et 5 possèdent la même utilité que précédemment

AFS_SEL : utilise le même principe que pour les gyroscopes sauf que cette fois ils s'agit de mesure en g : plus on souhaite mesurer de grandes valeurs, moins elles seront précises.

AFS_SEL	Full Scale Range
0	$\pm 2g$
1	$\pm 4g$
2	$\pm 8g$
3	$\pm 16g$

Tableau 10: Signification des différentes valeur des bits AFS_SEL

En effet, vu que dans tous les cas les valeurs sont stockées dans un registre sur 16 bits, une variation de un dans ce registre lorsque AFS_SEL est réglé sur 0 correspondra donc à une accélération de $6 \times 10^{-5}g$ alors que la même variation avec AFS_SEL réglé sur 3 correspond à une accélération de $5 \times 10^{-4}g$.

Registre 0x3B à 0x40 – Mesures d'accéléromètre

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59								ACCEL_XOUT[15:8]
3C	60								ACCEL_XOUT[7:0]
3D	61								ACCEL_YOUT[15:8]
3E	62								ACCEL_YOUT[7:0]
3F	63								ACCEL_ZOUT[15:8]
40	64								ACCEL_ZOUT[7:0]

Ces registres permettent d'accéder aux valeurs mesurées par les accéléromètres, sur 16 bits

Registres 0x43 à 0x48 – Mesures de gyroscope

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67								GYRO_XOUT[15:8]
44	68								GYRO_XOUT[7:0]
45	69								GYRO_YOUT[15:8]
46	70								GYRO_YOUT[7:0]
47	71								GYRO_ZOUT[15:8]
48	72								GYRO_ZOUT[7:0]

Ces registres permettent d'accéder aux valeurs mesurées par les gyroscopes, sur 16 bits

Outre ces registres beaucoup d'autres existent sur le module mais ne seront pas utilisés, en effet le module peut être utiliser comme maître pour une communication I2C avec d'autre capteur/module. Dans notre cas il ne s'agit évidemment pas de notre utilisation.

D. Mise en place

La mise en fonctionnement du module se fait par une courte phase d'initialisation où nous appliquons ce que nous avons vu dans l'étude de registre. Nous avons eu du mal à vraiment intégrer de façon pertinente les valeurs en provenance de la centrale inertie. En effet toutes les données que l'on obtient sont redondante par rapport aux informations fournis par les capteurs optiques. Bien-sûr la redondance de donnée n'est pas en soit une mauvaise chose et permettrait de limiter les marges d'erreurs, néanmoins l'avancé tardive du projet ne permet pas de savoir si de tel informations seront utiles tant que l'on ne sait pas si les capteurs optiques en condition réel sont précis et efficaces.

Deux types d'information nous proviennent de la centrale :

- Les accéléromètres vont nous permettre d'obtenir les vitesses actuelles sur les axes x et y. Ceci se fait en deux étapes, une étape de conversion depuis les valeurs du capteur vers les unités du système international. Ensuite il faut convertir la valeur obtenue en g vers une vitesse en m/s qui pourra ensuite servir pour l'asservissement en vitesse.
- Les gyroscopes fournissent quant à eux une vitesse angulaire, plus familièrement il donne la vitesse de rotation du module sur lui-même. Il nécessiterait des maths un peu plus compliquées pour convertir des °/s en vitesse de rotation de roue que l'on rapportait ensuite sur notre base 16bits pour l'utilisation dans notre PID. Aussi, et cette fonctionnalité à été mise en place, en intégrant les valeurs de gyroscope sur le temps on peut obtenir l'angle absolu du robot.



Illustration 23: Démonstration pour la centrale

Finalement peu de fonctionnalité envisageable ont été mise en place, dans tous les cas nous avons montré notre capacité à obtenir des informations en provenance du capteur, à partir de là il ne manque qu'un pas pour les exploiter dans un futur projet.

VI. PID

A. Principe

Dans les systèmes où l'on souhaite contrôler un mécanisme physique à partir d'un environnement électronique se pose un problème important : Les systèmes physiques possède de nombreuses variables (tel que l'inertie) qui vont empêcher le système d'atteindre les caractéristiques souhaitées instantanément et de manière précises. C'est à cette problématique que répond le PID.

Le PID (proportionnel, intégrale, dérivé) est un système de contrôle prenant en compte la mesure des effets des commandes sur le monde réel et qui utilise cette donnée pour ajuster la commande donnée. Pour cela trois procédés sont utilisés, dans un premier temps doit être calculé l'erreur qui correspond à la différence entre la commande (ce que l'on veut) et le retour du (des capteur(s)) (ce qui se passe réellement). Cette différence est notée $e(t)$. On applique donc à cette erreur ces trois calculs :

- P(proportionnel) : On multiplie $e(t)$ par une valeur fixe, l'objectif est d'appliquer un facteur (toujours le même) sur $e(t)$ pour faire croître (ou décroître) la commande.
- I (integral) : Ce paramètre va enregistrer toutes les valeurs d'erreurs précédemment enregistrées et de les additionner. Ce calcul permet de supprimer les erreurs résiduelles¹⁶. En effet, si au bout d'un grand nombre d'itérations la commande en sortie du PID est toujours la même mais légèrement fausse, le paramètre 'I' va compenser ce décalage. Ce paramètre est prévu pour faire effet vers la fin du processus de stabilisation quand la somme a eu le temps de grandir
- D (dérivé) : Le paramètre D va permettre de prévoir les futures évolutions de l'erreur, en effet, le calcul de le dérivé permet d'obtenir « la pente » de la courbe de l'évolution de l'erreur en fonction du temps. L'objectif de ce paramètre est d'aplatir la courbe d'erreur et de la stabiliser sur la commande souhaitée

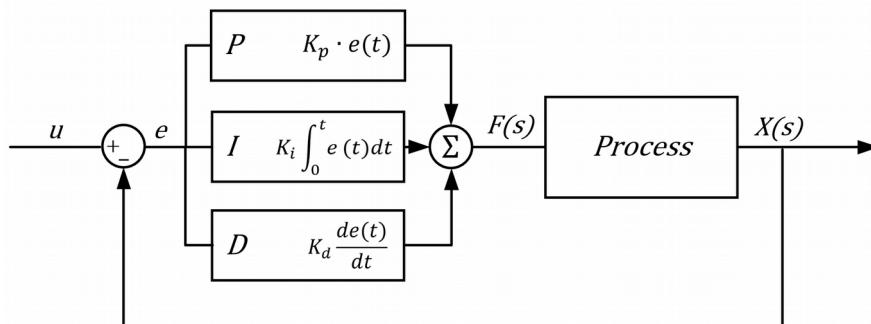


Illustration 24: Principe du PID

Il va être appliqué à chaque paramètre un coefficient on obtient ainsi la forme mathématique suivante :

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

¹⁶ Ex erreur résiduelle : Un potentiomètre (commande) dont les graduations sont décalées va générer une erreur résiduelle égale au décalage

Avec :

$u(t)$: valeur de la nouvelle commande

K_p, K_i, K_d : Les coefficients pour chaque paramètre

Variable pour intégrer, représente toutes les valeurs précédentes de t

T

B. Mise en œuvre

Une fois le principe du PID compris, la mise en œuvre logiciel de ce dernier est assez simple. Nous avons besoin de plusieurs PID au sein de notre robot :

- Le plus évident, mais aussi le plus important : Un PID pour la vitesse de chaque moteur individuel. Le calcul de ce PID est fait sur le micro-contrôleur relié aux ESC, la commande provient du micro-contrôleur centrale et est assimilable à une vitesse en tour par seconde. C'est l'encodeur présent sur le moteur qui permet de récupérer la vitesse réelle du moteur. Cette partie est détaillée dans le rapport sur les moteurs.
- Il va également falloir mettre en place un PID pour les déplacements réels du robot, la commande provient de la manette sous forme de deux coordonnées (dx, dy), le retour du déplacement réel du robot est fourni par le capteur optique central
- Enfin, de la même manière que pour les déplacements il faut mettre en place un PID pour la rotation du robot, le retour étant cette fois fourni par la différence de déplacement vertical entre les deux capteurs optiques

Contrôleur

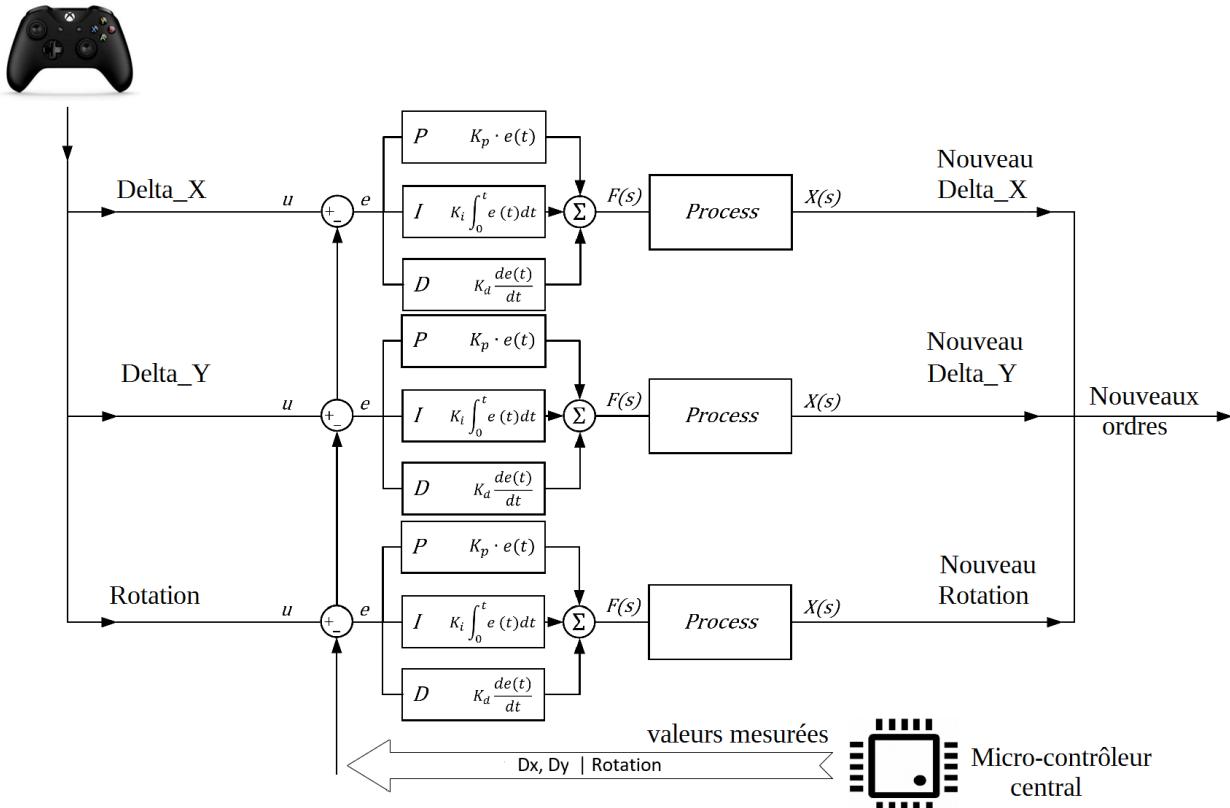


Illustration 25: Représentation du chemin des données au travers des calculs de PID

Il est bon de noter que beaucoup de système n'utilise que les paramètres P et I, en effet, en plus de ne pas être indispensable le paramètre D est sensible aux erreurs de mesure et au bruit qui peuvent faire varier la commande finale de manière importante.

- Les encodeurs que nous utilisons sont d'une grande précision cela ne devrait pas être problématique
- Il est assez difficile de connaître par avance la précision des capteurs optiques
- La mise en place logiciel du PID va donc se faire paramètre par paramètre : après avoir calculé l'erreur on va d'abord calculer le paramètre P grâce à une simple multiplication.
- Le paramètre I est également assez simple : une variable va être utilisée pour stocker la somme de toutes les valeurs d'erreurs multipliées par la fréquence d'échantillonnage, il est important de choisir une variable assez grande pour contenir toutes les valeurs. Mais sans tests préalable il est difficile de prévoir la taille réellement nécessaire vu qu'elle dépend du temps (et donc du nombre d'échantillons) utile pour atteindre la commande.
- Enfin on effectue un calcul simplifié du paramètre D en soustrayant la valeur actuelle de l'erreur par la valeur n-1 avant de diviser le tout par notre fréquence d'échantillonnage, certes le résultat est approximatif mais il reste fiable d'autant qu'il s'agit ici de calculs qui font être effectués environ 150 fois par seconde d'où l'intérêt de les garder relativement simples.

A l'heure de la rédaction de ces lignes nous n'avons toujours pas de châssis et les ESC qui permettent le fonctionnement des moteurs ne sont toujours pas opérationnels. Il paraît difficile de mettre en condition nos capteurs en termes de lumière est de position par rapport au sol. Dans ces conditions il est compliqué de savoir si le paramètre D posera problème bien que les fiches techniques de nos capteurs (encodeur et capteur optique) soit encourageante.

C. Calibration

La mise en place théorique d'un PID n'est pas réellement complexe contrairement au choix des trois facteurs qui sont la clé de voûte du concept. Chaque paramètre possède des caractéristiques propres qu'il faut équilibrer. Essayer de trouver le facteur manuellement sans méthode n'est pas envisageable tant le nombre de combinaisons possibles est important. Néanmoins nous laisseront de côté les méthodes de calibration calculatoire sont souvent très « générique » est plus réservé à un environnement où l'on peut pas effectuer de nombreux tests.

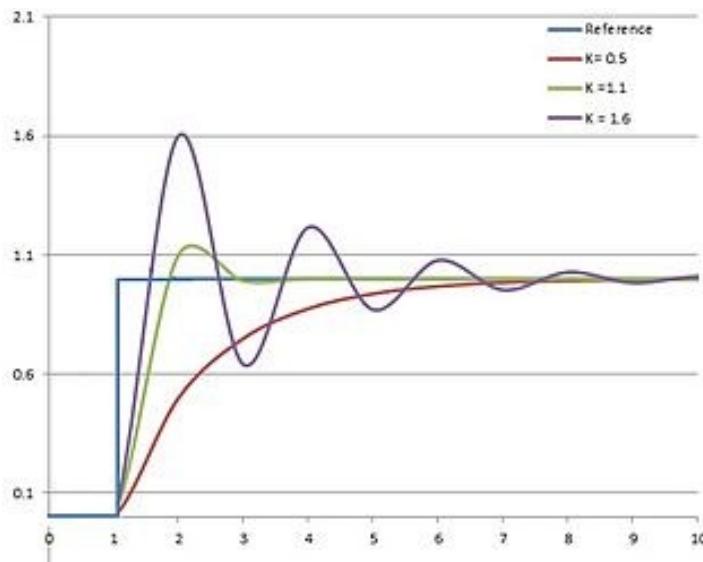


Illustration 26: Exemple des variations de $e(t)$ en fonction de K_p (facteur du paramètre proportionnel)

Méthode de mise en place des facteurs du PID

Pour pouvoir trouver les facteurs adéquats il faut passer par une phase de test avec le matériel tel qu'il sera lors de l'utilisation finale. Il faut ensuite pouvoir observer la courbe $e(t)$ (erreur en fonction du temps) de notre processus chaque test. Dans cet optique il faut mettre en place un banc de calibration avec en entré une commande fixe et en sortie, qui serait communiqué par le module radio par exemple ou par liaison série, la valeur de $e(t)$. Cette étape est facilement réalisable d'autant que le module radio est fonctionnel.

Dans un premier temps les facteurs pour I et D sont mis à 0 (désactivés) et on incrémente la valeur du facteur pour P jusqu'à ce que la pente initiale soit suffisante et que la sortie finisse par osciller. Vient ensuite le réglage du facteur pour I, nous allons l'incrémenter jusqu'à éliminer l'« offset » pour que notre sortie converge vers notre commande souhaitée. Enfin nous allons incrémenter le facteur pour D pour « lisser » notre courbe. Ce facteur diminue à la fois les dépassements de commande mais aussi le temps nécessaire pour atteindre la commande.

VII. CALCULS POUR LES DÉPLACEMENTS

A. Calculs théoriques

Comme nous le savons notre robot est dit holonomes, grâce à ses roues spéciales le robot peut se déplacer dans n'importe quelle direction sans avoir à effectuer de rotation. L'objectif va être de réussir à calculer la vitesse de rotation de chaque moteur (et donc de chaque roue) pour que le robot se dirige dans la bonne direction.

Il faut dans un premier temps recueillir une information capitale : l'angle de chaque roue par rapport à une droite horizontale, on peut ensuite facilement représenter les vecteurs unitaires de chaque moteur. La décision de l'agencement des roues a été assez vite décidée ce qui nous donne le schéma suivant :

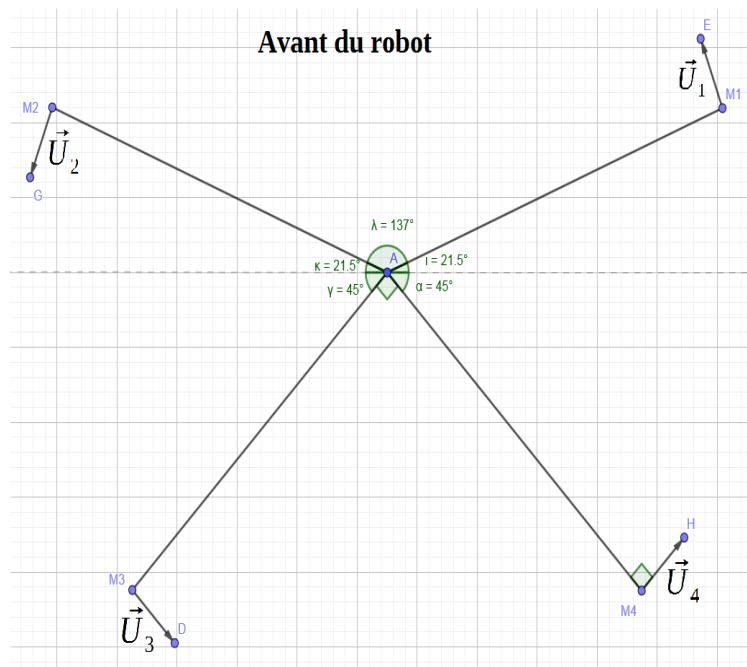


Illustration 27: Représentation des vecteurs unitaires des moteurs

Cette représentation permet également de choisir un identifiant pour chaque moteur que l'on gardera sur tout le projet. Les vecteurs unitaires sont des vecteurs de norme 1 qui représentent premièrement la direction¹⁷ de la force appliquée par les moteurs et deuxièmement le sens¹⁸ positif de rotation du moteur (et donc d'application de la force). On note \vec{U}_n un vecteur unitaire : avec n le numéro du moteur correspondant.

¹⁷ La direction est une information portée par une droite

¹⁸ Une droite possède deux sens, le sens est représenté par une flèche

Une fois ces bases posées nous allons devoir mettre en place une méthode pour déplacer notre robot. On pose le vecteur de déplacement souhaité du robot, on utilisera les

$$\mathbf{V} = \begin{pmatrix} X \\ Y \end{pmatrix}$$

représentations cartésiennes des vecteurs pour les calculs à l'aide du langage C de manière à éviter d'avoir à utiliser la trigonométrie dans notre futur code.

Pour obtenir la norme du vecteur de chaque moteur qui permettra d'obtenir notre déplacement nous allons devoir réaliser le produit scalaire du vecteur V avec chacun de nos vecteurs unitaires. Le produit scalaire de deux vecteurs est un nombre issu de la longueur de chaque vecteur mais principalement dépendant de l'angle qu'ils forment.

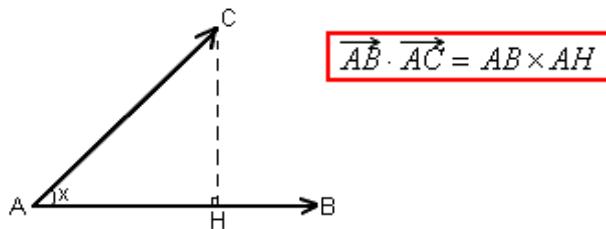


Illustration 28: Bref définition du produit scalaire

Pourquoi effectuer le produit scalaire entre V et Un ? Pour mieux comprendre le principe de notre calcul, intéressons-nous au calcul du produit scalaire à l'aide d'un cosinus¹⁹.

$$x = \text{angle}(CAB)$$

$$\overrightarrow{AB} \cdot \overrightarrow{AC} = AB * AC * \cos(x)$$

Illustration 29: Formule du produit scalaire

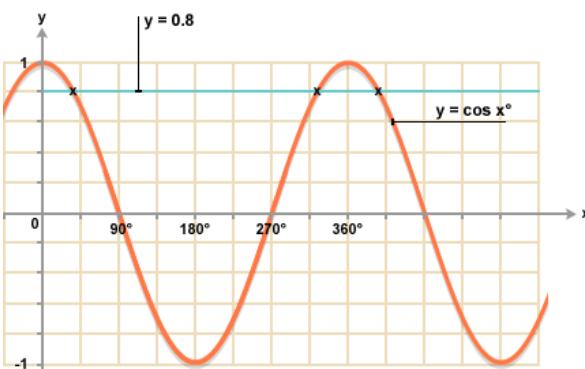


Illustration 30: Représentation fonction cosinus

Intéressons-nous à deux points remarquables de la fonction cosinus :

¹⁹ Bien-sûr, peu importe la formule utilisée le résultat reste le même

- Lorsque $x = 0^\circ$, alors $\cos(x)=1$, dans notre cas cela signifie que lorsque les vecteurs V et U_n seront colinéaires²⁰ (angle nul) alors le produit scalaire sera égal au produit des normes de nos vecteurs. De plus U_n étant un vecteur unitaire sa norme est égale à 1. On obtient donc :

avec $x=0$

$$\vec{V} \cdot \vec{U}_n = V * U_n * 1$$

$$\vec{V} \cdot \vec{U}_n = V$$

→ Dans le cas où le vecteur de direction est colinéaire au vecteur unitaire de la roue, alors le vecteur de vitesse de la roue possède une valeur maximum.

- Lorsque $x = 90^\circ$ ou 180° , alors $\cos(x)=0$, cette fois ci les vecteurs V et U_n sont orthogonaux, on obtient de la même manière :

avec $x=90^\circ$ ou $x=180^\circ$

$$\vec{V} \cdot \vec{U}_n = V * U_n * 0$$

$$\vec{V} \cdot \vec{U}_n = 0$$

→ Dans ce cas le vecteur de vitesse de la roue sera nul, en effet, le robot se dirige de tel sorte que le moteur ne peut pas être utile car il est orthogonal au vecteur vitesse.

- Tous les autres cas se situent entre ces deux extrêmes ils sont bien-sûr supportés par notre calcul.

Exemple :

On note \vec{M}_n le vecteur colinéaire à \vec{U}_n et de même origine, représentant la vitesse du

moteur pour que le robot se dirige dans la bonne direction. Comme nous l'avons expliqué, la norme de ce vecteur est égale au produit scalaire entre V et U_n .

²⁰ Deux vecteurs sont dit colinéaires lorsque les droites qui les portent sont parallèles.

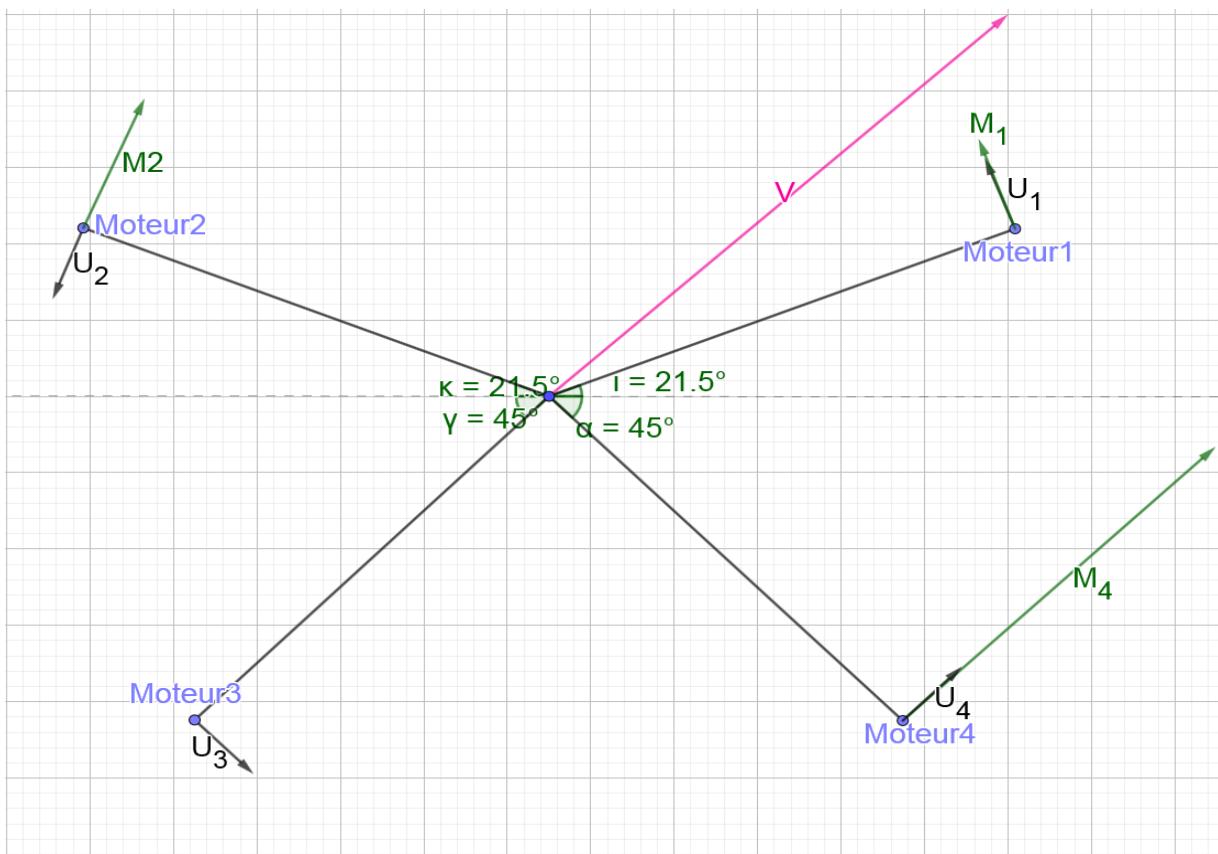


Illustration 31: Représentations des vecteurs présentés

Déterminer les composantes de \vec{U}_n

Avant de pouvoir effectuer le moindre calcul nous aurions besoins de calculer l'angle entre chaque vecteur unitaire et une droite horizontale (par exemple) afin de reporter ces valeurs sur les vecteurs vitesses pour obtenir l'angle x . Comme expliqué précédemment nous allons plutôt utiliser les composantes x et y de nos vecteurs unitaires pour accélérer les calculs.

Nous allons devoir donc calculer une fois pour toutes les valeurs de : $\vec{U}_n = \begin{pmatrix} dx \\ dy \end{pmatrix}$. Obtenir ces valeurs fait appel à des notions simples de trigonométrie.

On peut voir ici notre vecteur \vec{U}_1 , on trouve l'angle ρ en manipulant un peu les angles, nous avons $21,5^\circ$ entre l'horizontale (axe x) et le moteur 1. Il suffit de retirer $21,5$ à l'angle droit formé par le vecteur porté par le moteur, ainsi on obtient les $68,5^\circ$ affiché.

On sait que :



Illustration 31: Représentation du vecteur \vec{U}_1 et de ses composantes.

« Zoom » sur la représentation précédente

De plus **la norme de** \vec{U}_1 étant égale à 1 on obtient donc facilement dx et dy, qui sont égaux dans ce cas à :

$$dx = 0,366 \quad \text{et} \quad dy = 0,930$$

Pour être sûr de nos résultats qui sont cruciaux quant à la réussite des déplacements nous pouvons les vérifier à l'aide d'un logiciel de géométrie²¹. En appliquant la même méthode nous allons pouvoir obtenir nos huit valeurs de composantes. Nous obtenons donc les valeurs suivantes :

Numéro moteur	\vec{U}_n dx	\vec{U}_n dy
1	0,366	0,930
2	-0,366	-0,366
3	0,707	-0,707
4	0,707	0,707

²¹ J'utilise pour ma part GeoGebra pour vérifier mes calculs

B. Application

On pose : $\vec{V} = \begin{pmatrix} 10 \\ 10 \end{pmatrix}$, on cherche à obtenir la norme de chaque vecteur \vec{M}_n que l'on notera M_n . On effectue donc le calcul suivant pour chaque moteur : $M_n = Un_{dx} * V_{dx} + Un_{dy} * V_{dy}$ qui représente la formule du produit scalaire en utilisant les valeurs cartésiennes et non les angles. On obtient donc les valeurs de M_n . On peut donc, maintenant que l'on a calculé leur norme tracer les vecteur \vec{M}_n qui correspondent (à un facteur prêt) à la puissance de chaque moteur²².

C. Mise en place

Une fois les calculs théoriques défini précisément il ne reste plus qu'à mettre en place le code qui permettra l'exécution des calculs. Dans un premier temps, les coordonnées cartésiennes des vecteurs unitaires sont entrés en dur dans le fichier .h. On sait que notre programme devra prendre en entré trois variables : delta_X, delta_Y, rotation, en provenance de la manette et toutes de type signed short. Notre tâche va être de transformé ces paramètres en quatre valeurs de puissance de moteur de type signed char²³.

Comme nous allons effectuer des opérations sur des variables qui peuvent avoir déjà atteint leur valeur maximum, on utilise une variable provisoire de grande taille pour être sur d'éviter tout dépassement. On y applique donc le calcul précédemment expliqué.

A ce stade il n'y a aucune raison pour que les valeurs de puissance obtenues soit comprises dans l'intervalle [-128,+128], nous notons cette valeur intermédiaire Mn_{temp} . Pour parvenir à remettre cette valeur dans notre échelle il nous faut observer la valeur maximum qui peut être attribuée à un moteur.

On obtient cette valeur maximale lorsque le vecteur vitesse voulue est de norme maximum et qui est colinéaire au vecteur unitaire d'un moteur. Le cas où $\vec{V} = \begin{pmatrix} 32767 \\ 32767 \end{pmatrix}$ remplit ce cahier des charges, effet ce vecteur est colinéaire au vecteur $\vec{U}_4 = \begin{pmatrix} 0,707 \\ 0,707 \end{pmatrix}$, on peut donc obtenir la valeur maximum de M_n en appliquant notre calcul. On obtient donc 46332 (sans la virgule qui est troquée en c). Ce que cela signifie est simple, on veut qu'une valeur de 46332 donne une puissance de sortie de 128, on applique un rapide produit en croix : $\frac{46332}{128} = \frac{Mn_{temp}}{P_{out}}$, on déduit donc que l'on doit notre valeur de sortie doit être égale à :

$$P_{out} = Mn_{temp} * \frac{128}{46332} \text{ soit : } P_{out} = Mn_{temp} / 357$$

Viens ensuite la rotation que nous n'avons pas abordé dans la partie théorique car elle se résout de manière assez intuitive : au vu du positionnement des vecteurs vitesse de nos moteurs, il

22 Voir illustration 31

23 Format de stockage de donné sur huit bits, acceptant des valeur dans l'intervalle [-128,+128]

suffit d'ajouter la même valeur à la norme de tous les vecteurs pour faire tourner le robot sur lui-même. L'addition d'une rotation négative fera tout simplement tourner le robot dans le sens inverse.

On remarque alors un problème, si le vecteur vitesse est de taille important (voir de taille maximum) il peut être orienté de telle sorte que les valeurs de vitesse pour le moteur dépassent les 127 ou -128 imposé par note type de variable (notons que même avec un type de variable plus grand le problème serait le même).

La solution part d'un postulat : nous avons choisi plus tôt 10 m/s comme valeur vitesse maximum de référence, mais l'on sait que cette valeur risque d'être difficile à atteindre réellement pour le robot, ainsi dans la pratique le robot sera probablement moins rapide.

La solution choisie est donc, lorsque la rotation rend des valeurs trop grandes, de diviser toutes nos valeurs par un facteur commun (4 par exemple) ainsi on obtient de valeurs de libre pour ajouter la rotation.

Cette solution va finalement aller directement modifier les ordres donnés par l'utilisateur mais nous gardons l'essentiel de cette dernière : une rotation comme voulue ainsi qu'un déplacement dans la bonne direction. Modifier la commande partait capitale lorsqu'elle n'est pas applicable.

CONCLUSION

A la fin des trois mois de projet plusieurs fonctions sont en place : la lecture de donné avec les trois capteurs, deux capteurs optiques et la centrale inertie, fonctionne très bien et l'on peut lire les données reçues. La partie calcul pour les déplacements du robot semble également fonctionnelle bien que je n'ai pas était en mesure de la tester les valeurs que l'on obtient sont cohérentes.

Tout ce qui concerne le PID à bien été mise en place mais la phase de calibration est de loin la plus compliquée et chronophage, nous n'avons pas pu l'effectuer en l'absence d'un robot fonctionnel il reste donc beaucoup de travail à effectuer de ce côté. De manière générale toutes les fonctions n'ont pas (ou très peu) été testées, peut-être aurions-nous dû mettre en place des procédures de test ne nécessitant pas de robot notamment pour les capteurs optiques afin d'en affiner notre usage avant la fin du projet.

Le travail effectué n'est néanmoins pas vain, nous sommes assez proche d'un robot fonctionnel et peu de temps de travail supplémentaire serait nécessaire pour obtenir un premier prototype répondant à nos attentes.

De notre côté le partie gestion de projet fût assez erratique, bien que le résultat soit correct améliorer cette partie aurait peut-être permis de finaliser le projet dans les temps.

Remerciement

Merci à Steve N'Guyen pour son engagement sans faille pour notre projet.

BIBLIOGRAPHIE

System Management Interface **System Management Bus Specification** *smbus.org*
http://www.smbus.org/specs/SMBus_3_0_20141220.pdf#page=81 le 20/12/2014

Philips Semiconductors **AN10216-01** I2C MANUAL *nxp.com*
<https://www.nxp.com/docs/en/application-note/AN10216.pdf> le 24/03/2003

Total Phase, Inc. **7-bit, 8-bit, and 10-bit I2C Slave Addressing** *totalphase.com*
<https://www.totalphase.com/support/articles/200349176>

freescale **MC9S08QE8 MC9S08QE4 Reference Manual** *freescale.com*
http://cache.freescale.com/files/microcontrollers/doc/ref_manual/MC9S08QE8RM.pdf
le 01/02/2011

freescale **MC9S08QE128 MC9S08QE96 MC9S08QE64 Reference Manual** *freescale.com*
<https://www.nxp.com/docs/en/reference-manual/MC9S08QE128RM.pdf>
le 01/06/2007

Mike Grusin **Serial Peripheral Interface** *sparkfun.com*
<https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>

Ligo George **SPI – Serial Peripheral Interface** *electrosome.com*
<https://electrosome.com/spi/> le 30/04/2017

Agilent Technologie **Agilent ADNS-3080 High-performance Optical Mouse Sensor Data Sheet** *alldatasheet.com*
<http://pdf1.alldatasheet.com/datasheet-pdf/view/193358/HP/ADNS-3080.html> 2005

InvenSense **MPU -6000 and MPU-6050Product Specification** *invensense.com*
<https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>
le 08/19/2013

Dynamics and control **Proportional Integral Derivative (PID) Control** *apmonitor.com*
<http://apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative> le 12 sept. 2014

Brian Douglas **PID Control - A brief introduction** *youtube.com*
<https://www.youtube.com/watch?v=UR0hOmjaHp0> le 13 déc. 2012

David Kohanbash **PID Control (with code), Verification, and Scheduling**

robotsforroboticists.com

<http://robotsforroboticists.com/pid-control/>

le 22/09/2014

Brett beauregard **Improving the Beginner's PID – Introduction** *brettbeauregard.com*

<http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/> le 15/04/2011

ANNEXES

Tables des Annexes

ANNEXE A : Lien git-hub.....	47
ANNEXE B.1 : calculs.h.....	47
ANNEXE B.2 : calculs.c.....	52
ANNEXE C : I2C.h.....	53
ANNEXE D.1 : SPI_Capteurs.h.....	54
ANNEXE D.2 : SPI_Capteurs.c.....	55
ANNEXE E.1 : centrale.h.....	56
ANNEXE E.2 : centrale.c.....	57
ANNEXE F.1 : optical.h.....	58
ANNEXE F.2 : optical.c.....	59

ANNEXE A : Lien git-hub
https://github.com/LavergneC/Projet_B3
ANNEXE B.1 : calculs.h

```

1  #include <stdio.h>
2  #include <math.h>
3
4  #define PERIODE_ECH (double)0.006 //6µs
5
6  #define M1DX (double)-0.36650
7  #define M1DY (double)0.930417
8  #define M2DX (double)-0.36650
9  #define M2DY (double)-0.930417
10 #define M3DX (double)0.707106
11 #define M3DY (double)-0.707106
12 #define M4DX (double)0.707106
13 #define M4DY (double)0.707106
14
15 #define Tech_centrale 0.024 //Temps entre chaque recuperation de donne sur le MPU-6050
16 #define Tech_capteur 0.003 //Temps entre chaque recuperation de donne sur les capteurs
17
18
19 typedef struct {
20     signed short errors[100];
21     double kp;
22     double ki;
23     double kd;
24     unsigned char i;
25 }VariablesPID;
26
27 /*
28 *initPID
29 * La fonction permet d'initialiser la variable suite à sa création en initialisant toutes les variables de la structure
55 */
56
57 * conversionCapteurToPID
58 * Cette fonction permet de convertir une mesure d'un capteur optique afin de l'injecter dans un calcul de PID
59 * @param : signed char data : la donnée en provenance d'un capteur
60 * @return signed short : la valeur convertie
61 */
62 signed short conversionCapteurToPID(signed char data);
63
64 /*
65 * getvitesse
66 * permet d'obtenir la vitesse du robot grâce aux valeurs des capteurs
67 * @param : signed char dx : mesure de déplacement horizontal
68 *          signed char dy : mesure de déplacement vertical
69 * @return : double : vitesse en m/s
70 */
71 double getVitesse(signed char dx, signed char dy);
72
73 /*
74 * convAngleAbsolu
75 * Permet d'obtenir l'angle absolu du robot par rapport à sa position de départ ou depuis le dernier reset
76 * @param : signed short vitesse : valeur mesurée par la centrale inertielle (GYRO_Z)
77 *          unsigned char reset : si reset !=0 la position actuelle devient la position de référence
78 *          signed short * angle : valeur absolue de l'angle du robot
79 */
80 void convAngleAbsolu(signed short vitesse, unsigned char reset, signed short * angle);

```

ANNEXE B.2 : calculs.c

```

11 void getVMoteurs(signed short dx, signed short dy, signed short rot, signed char VMoteurs[4]){
12     unsigned char i,j;
13     signed long temp[4]={0,0,0,0};
14     signed short deplacement =0;
15
16     rot = (signed char)(rot/254);
17     temp[0] = dx * M1DX + dy * M1DY;
18     temp[1] = dx * M2DX + dy * M2DY;
19     temp[2] = dx * M3DX + dy * M3DY;
20     temp[3] = dx * M4DX + dy * M4DY;
21
22     for(i=0; i<4;i++){
23         temp[i]/=364; //On remet temp à l'échelle [-128,+127]
24         if((temp[i] + rot ) < -128 || (temp[i] + rot )>127){ // Si la rotation fait sortir de l'intervalle
25             for(j=0; j<4; j++) /*on diminue toutes les valeurs d'un quart*/
26                 temp[j] -= temp[j]/4;
27         }
28     }
29     for(i=0; i<4;i++){
30         if((temp[i] + rot ) < -128 || (temp[i] + rot )>127){
31             deplacement = fabs(temp[i]+rot) - 127;
32             if(temp[i]>0)
33                 rot -= deplacement;
34             else
35                 rot += deplacement;
36         }
37     }
38
39     for(i=0; i<4; i++){
40         temp[i] += rot;
41         VMoteurs[i] = temp[i];
42     }
43 }
44
45 /*
46
51 void initPID(VariablesPID *vars, double kp, double ki, double kd){
52     unsigned char i;
53
54     for(i=0; i<100; i++)
55         vars->errors[i] = 0;
56     vars->kp = kp;
57     vars->ki = ki;
58     vars->kd = kd;
59     vars->i = 0;
60 }

```

```

70 signed short calculPID(signed short commande, signed short retourCapteur, VariablesPID *variablesPID){
71     unsigned char j;
72     double P, I, D;
73     signed short somme;
74     signed char error = commande - retourCapteur;
75
76     I=0;
77     variablesPID->i = (variablesPID->i+1)%100;
78     variablesPID->errors[variablesPID->i] = error;
79
80     P = variablesPID->kp * error;
81
82     for(j=0; j<100; j++)
83         I+= variablesPID->errors[j] * PERIODE_ECH;
84     I *= variablesPID->ki;
85
86     D = (error - variablesPID->errors[variablesPID->i-1]) / PERIODE_ECH;
87     D *= variablesPID->kd;
88
89
90     if (P+I+D>32767)
91         somme = 32767;
92     else if (P+I+D <-32767)
93         somme = -32767;
94     else
95         somme = (signed short) P+I+D;
96
97     return somme;
98 }
106 signed short conversionCapteurToPID(signed char data){
107     double tempData = data;
108
109     tempData *=34.6784; /* x0.0254/400*3276.7/0.006 */
110
111     return (signed short)tempData;
112 }
140 double getVitesse(signed char dx, signed char dy){
141     signed short normeV = sqrt(dx*dx + dy*dy);
142     double distance = (normeV * 2.54)/400; //une valeur de 400 correspond à un pied (2.25 cm)
143     return distance/Tech_capteur;
144 }

```

ANNEXE C : I2C.h²⁴

```

20 #define TIME_OUT_I2C 10
21 #define MPU_ADR (0x68<<1)
22
23 typedef enum{I2C_OK=0, I2C_FAILED}I2C_Status;
24
25
26 /*
27 * I2C_init
28 * Permet l'initialisation du module I2C du micro-contrôleur pour notre usage
29 */
30 void I2C_init(void);
31
32 /*
33 * Fonctions permettant la génération de commandes I2C de base
34 */
35 void I2C_start();
36 void I2C_stop();
37 void I2C_restart();
38 /*-----*/
39
40 /*
41 * Fonctions I2C pour la lecture/écriture d'un octet
42 */
43 void I2C_write_oct(unsigned char oct, I2C_Status *status);
44 void I2C_read_oct(unsigned char * oct, I2C_Status *status,unsigned char NACK);
45 /*-----*/
46
47 /*
48 * Fonctions I2C pour la lecture/écriture d'une donnée dans un registre
49 */
50 void I2C_write(unsigned char registre, unsigned char data, I2C_Status *status);
51 void I2C_read(unsigned char registre, unsigned char *data, I2C_Status *status);
52 /*-----*/

```

²⁴ I2C.c non mis en annexe du fait de la longueur de fichier et du manque de pertinence (disponible sur le git)

ANNEXE D.1 : SPI_Capteurs.h

```
13  typedef enum{MODULE1=0,MODULE2=1}Module;
14
15  #define MOTION 0x02
16  #define DX 0x03
17  #define DY 0x04
18  #define CS_1 PTAD_PTAD7
19  #define CS_2 PTAD_PTAD0
20
21  /*
22  * SPI_init
23  * Permet l'initialisation du module SPI du micro-contrôleur pour notre usage
24  */
25  void SPI_init(void);
26
27  /*
28  * SPI_w_r
29  * Fonction pour écrire et lire par SPI
30  * @param : unsigned char write[2] : Tableau contenant les deux valeurs qui seront écrites sur le bus (mettre le second à 0x00 pour une lecture)
31  *          unsigned char *read : valeur lue sur le bus lors de la seconde écriture (ignorer en cas d'écriture)
32  */
33  void SPI_w_r(unsigned char write[2], unsigned char *read, Module module);
--
```

ANNEXE D.2 : SPI_Capteurs.c

```

14 void SPI_init(void){
15     PTADD_PTADD0=1;
16     PTADD_PTADD7=1;
17     SPIBR=0x03;
18     SPIC1_MSTR=1;
19     SPIC1_CPOL=1;
20     SPIC1_CPHA = 1;
21     SPIC1_SPE=1;
22     CS_1 = 1;
23     CS_2 = 1;
24
25     return;
26 }

34 void SPI_w_r(unsigned char write[2], unsigned char *read, Module module){
35     unsigned char i;
36
37     if(module == MODULE1)
38         CS_1 = 0;
39
40     else
41         CS_2 =0;
42
43     for(i=0; i<2; i++){
44         if(SPIS_SPTEF){
45             SPID = write[i];
46         }
47
48         delai(83);
49         if(SPIS_SPRF){
50             *read = SPID;
51         }
52         //delai(83);
53     }
54
55     if(module == MODULE1)
56         CS_1 = 1;
57     else
58         CS_2 =1;
59
60     return;
61 }

```

ANNEXE E.1 : centrale.h

```

12  #define PWR_MGMT_1 0x6B
13  #define INT_PIN_CRG 0x37
14  #define ACCEL_CONFIG 0x1C
15  #define GYRO_CONFIG 0x1B
16  #define CONFIG 0x1A
17
18  #define ACCEL_X 0x3B
19  #define ACCEL_Y 0x3D
20  #define GIRO_Z 0x47
21
22  #define CENTRALE_TIMEOUT 10

24  /*
25  * initCentrale
26  * fonction pour initialiser les registres de configuraiton du module
27  * @param : I2C_Status *status : Variable permettant de s'assurer que les communicatin I2C ont fonctionés
28  */
29  void initCentrale(I2C_Status *status);
30
31  /*
32  * getData
33  * Permet de lire une donnée stockée sur deux registres se suivant;
34  * @param : unsigned char registre : adresse du premier registre à lire
35  *          unsigned char *data1 : valeur de retour pour le premier registre lu à l'adresse 'regsitre'
36  *          unsigned char *data2 : valeur de retour pour le second registre lu à l'adresse 'regsitre'+1
37  */
38  void getData(unsigned char registre, unsigned char *data1, unsigned char *data2);
39
40  /*
41  * visuAcce
42  * Routine permettant de tester les communication avec la centrale en affichant par liaison série ACCEL_X, ACCLE_Y et GYRO_Z
43  */
44  void visuAcce(void);

```

ANNEXE E.2 : centrale.c²⁵

```

17 void initCentrale(I2C_Status *status){
18     I2C_write(PWR_MGMT_1, 0x09, status);
19     if(*status == I2C_FAILED)
20         return;
21
22     I2C_write(INT_PIN_CRG, 0x00, status);
23     if(*status == I2C_FAILED)
24         return;
25
26     I2C_write(ACCEL_CONFIG, 0x08, status)
27     if(*status == I2C_FAILED)
28         return;
29
30     I2C_write(GYRO_CONFIG, 0x08, status);
31     if(*status == I2C_FAILED)
32         return;
33
34     I2C_write(CONFIG, 0x01, status);
35     if(*status == I2C_FAILED)
36         return;
37
38     *status = I2C_OK;
39     return;
40 }

49 void getData(unsigned char registre, unsigned char *data1, unsigned char *data2){
50     unsigned char cpt=0;
51     I2C_Status status;
52
53     do{
54         I2C_read(registre, data1, &status);
55     }while(status==I2C_FAILED && cpt <CENTRALE_TIMEOUT);
56     cpt=0;
57     do{
58         I2C_read(registre+1, data2, &status);
59     }while(status==I2C_FAILED && cpt <CENTRALE_TIMEOUT);
60
61     return;
62 }

```

25 Je n'est pas mis le code de la fonction vissuAcce car il ne s'agit que de tests

ANNEXE F.1 : *optical.h*

```
13  /*
14  * sensors_init
15  * Permet d'initialiser le registre de configuration des deux capteurs optiques
16  */
17 void sensors_init(void);
18
19 /*
20 * getValues
21 * Fonction pour récupérer les valeurs de déplacement d'un capteur optique
22 * @param : signed char dxdy[2] : tableau où seront stocké les valeurs lues
23 *          Module module : Permet de sélectionner le module que l'on veut interroger
24 */
25 void getValues(signed char dxdy[2], Module module);
```

ANNEXE F.2 : *optical.c*

```

11  /*
12   * sensors_init
13   * Permet d'initialiser le registre de configuration des deux capteurs optiques
14   */
15 void sensors_init(){
16     unsigned char writeConf[2] = {0x8A, 0x50};
17     unsigned char data = 0x00;
18
19     SPI_w_r(writeConf, &data, MODULE1);
20     SPI_w_r(writeConf, &data, MODULE2);
21
22     return;
23 }
24
25 /*
26 * getValues
27 * Fonction pour récupérer les valeurs de déplacement d'un capteur optique
28 * @param : signed char dxdy[2] : tableau où seront stocké les valeurs lues
29         Module module : Permet de sélectionner le module que l'on veut interroger
30 */
31 void getValues(signed char dxdy[2], Module module){
32     unsigned char data;
33     unsigned char read[2]={MOTION,0x00};
34
35     SPI_w_r(read, &data, module);
36
37     if(data & 0x80){
38         read[0]=DX;
39         SPI_w_r(read, &data, module);
40         dxdy[0] = (signed char) data;
41         read[0]= DY;
42         SPI_w_r(read, &data, module);
43         dxdy[1] = (signed char) data;
44     }
45     else {
46         dxdy[0] = 0;
47         dxdy[1] = 0;
48     }
49     return;
50 }
```