
PILOTAGE DE ROBOT

COMPTE RENDU DE PROJET

Lien GitHub : https://github.com/hakamir/LD_Sambot

TABLE DES MATIERES

Introduction	3
Spécifications générales	4
Matériel.....	4
Structure.....	5
Exigences systèmes du logiciel	5
Exigences architecturales du logiciel	6
Exigences détaillées du logiciel	7
Spécifications détaillées	10
Connectique du robot - Commandes et informations.....	10
MSP430G2553.....	10
MSP430G2231.....	11
Algorithme de fonctionnement.....	12
Description du fonctionnement du robot	12
Schéma du fonctionnement du robot.....	13
Quelques précisions sur le fonctionnement du robot	15
Récupération de la distance en millimètre	16
Modules.....	18
Fonctions	19
Tests unitaires (boîtes noires)	23
Test du module « movement.c ».....	23
Test du module « measure.c »	27
Test du module « UART.c »	29
Test du module « SPIM.c ».....	33
Test du module « SPIS.c »	34
Test du module « servomotor.c ».....	35
Tests d'intégration	39
Test de la communication spi.....	39
Test de la communication UART avec le robot	39
Conclusion	41

INTRODUCTION

Durant notre de notre enseignement de 2^e année, nous avons pour but de concevoir un robot pilotable et autonome et ce en introduisant des notions des cours de bus de communication et de qualité logiciel. De ce fait, la structure du robot comporte deux cartes launchpad établissant une communication *Serial Peripheral Interface* (SPI), ainsi qu'une carte Bluetooth permettant le pilotage du robot à distance communiquant par bus *Universal Asynchronous Receiver Transmitter* (UART).

Ce projet a eu l'avantage de nous confronter à de nombreux obstacles pouvant survenir dans le domaine des systèmes embarqués et est donc un excellent moyen pédagogique pour nous préparer répondre aux différentes problématiques mise en avant dans la suite de ce rapport.

SPECIFICATIONS GENERALES

MATERIEL

Afin de pouvoir respecter le cahier des charges, le robot devra disposer d'un matériel adapté. Nous disposons de :

- 2 cartes Launchpad comportant chacune un microcontrôleur différent (MSP430G2553 et MSP430G2231)



- 1 module bluetooth RN-42



- 1 servomoteur HS-422



- 1 capteur infrarouge

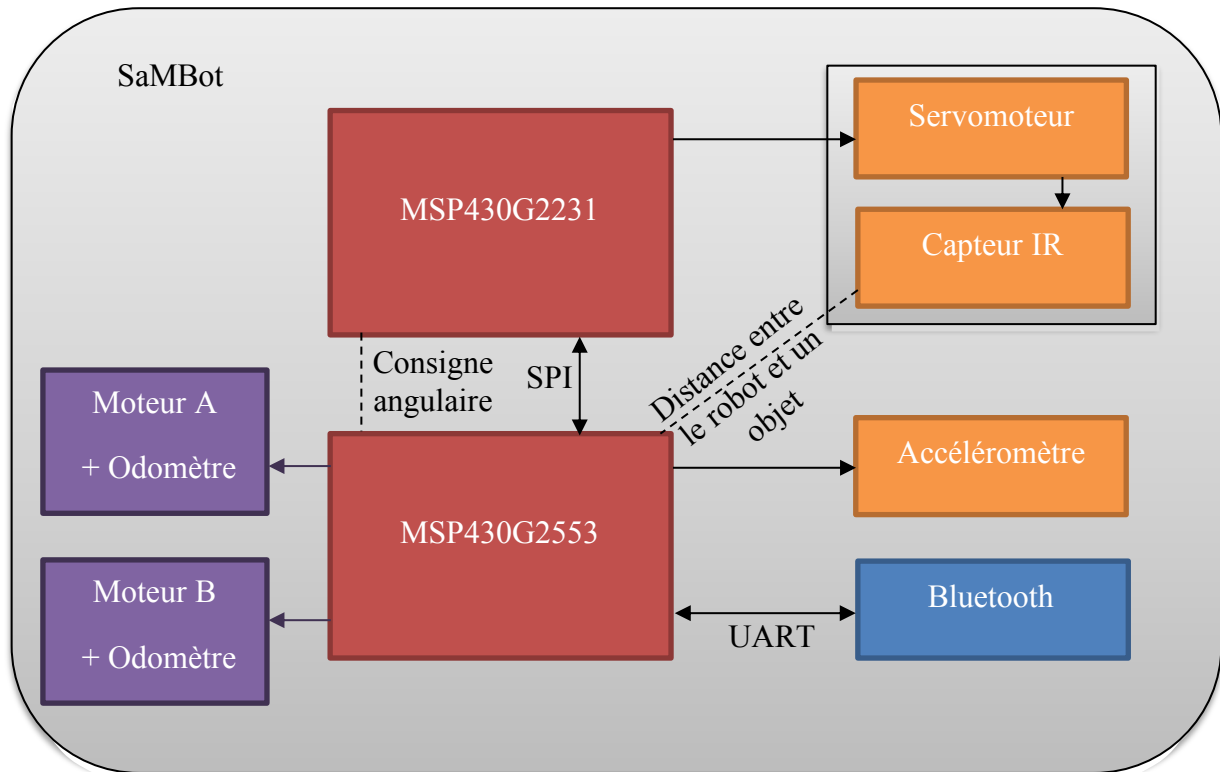


- 1 module SAMBot comportant une structure en plastique équipé de deux moteurs entrainants chacun une roue munie chacune d'un odomètre.



STRUCTURE

Grâce à ce matériel, nous avons défini la structure du robot qui suivra la configuration suivante :



EXIGENCES SYSTEMES DU LOGICIEL

SYS_0001

Nom: Déplacement

Texte: Le robot se déplacera.

SYS_0002

Nom: Communication UART

Texte: Le robot communiquera en bluetooth avec d'autres appareils.

SYS_0003

Nom: Communication SPI

Texte: Le robot aura deux microcontrôleurs qui communiqueront entre eux.

SYS_0004

Nom: Détection

Texte: Le robot effectuera le balayage d'un capteur infrarouge à partir d'un servomoteur.

EXIGENCES ARCHITECTURALES DU LOGICIEL

M_0001

Nom: Mouvement

Texte: Le robot devra effectuer des actions de déplacement élémentaires.

Couverture: **SYS_0001**

Module: **movement**

M_0002

Nom: Communication UART

Texte: Le robot devra être capable de recevoir et d'envoyer des informations à un appareil connecté via bluetooth grâce à une communication UART.

Couverture: **SYS_0002**

Module: **UART**

M_0003

Nom: Communication SPI

Texte: Le robot permettra à ses deux microcontrôleurs de s'envoyer et recevoir des informations.

Couverture: **SYS_0003**

Module: **SPIM, SPIS**

M_0004

Nom: Capteur infrarouge

Texte: Le robot sera capable de détecter un objet devant lui dans un champ de 180° grâce à un capteur infrarouge.

Couverture: **SYS_0004**

Module: **measure**

M_0005

Nom: Servomoteur

Texte: Le robot sera capable d'effectuer un balayage devant lui.

Couverture: **SYS_0004**

Module: **servomotor**

EXIGENCES DETAILLEES DU LOGICIEL

F_0001

Nom: Mouvement

Texte: Le robot devra pouvoir se déplacer, c'est-à-dire, avancer, reculer, tourner à différentes vitesses.

Couverture: **M_0001**

Fonction: **move**

F_0002

Nom: Arrêt

Texte: Le robot devra pouvoir s'arrêter à tout moment et instantanément.

Couverture: **M_0001**

Fonction: **stop**

F_0003

Nom: Initialisation UART

Texte: Le robot devra pouvoir initialiser les ports permettant la transmission de données par les bus UART.

Couverture: **M_0002**

Fonction: **UART_init**

F_0004

Nom: Transmission UART

Texte: Le robot transmettra des données via la communication UART.

Couverture: **M_0002**

Fonction: **UART_Tx**

F_0005

Nom: Réception UART

Texte: Le robot recevra des données via la communication UART.

Couverture: **M_0002**

Fonction: **UART_Rx**

F_0006

Nom: Initialisation SPI

Texte: Le robot devra pouvoir initialiser les ports permettant la transmission de données par les bus SPI sur le microcontrôleur MSP430G2553.

Couverture: **M_0003**

Fonction: **SPIM_init**

F_0007

Nom: Initialisation SPI

Texte: Le robot devra pouvoir initialiser les ports permettant la transmission de données par les bus SPI sur le microcontrôleur MSP430G2231.

Couverture: **M_0003**

Fonction: **SPIS_init**

F_0008

Nom: Transmission SPI

Texte: Le robot transmettra des données via la communication SPI sur le microcontrôleur MSP430G2553.

Couverture: **M_0003**

Fonction: **SPIM_Tx**

F_0009

Nom: Transmission SPI

Texte: Le robot transmettra des données via la communication SPI sur le microcontrôleur MSP430G2231.

Couverture: **M_0003**

Fonction: **SPIS_Tx**

F_00010

Nom: Réception SPI

Texte: Le robot recevra des données via la communication SPI sur le microcontrôleur MSP430G2553.

Couverture: **M_0003**

Fonction: **SPIM_Rx**

F_00011

Nom: Réception SPI

Texte: Le robot recevra des données via la communication SPI sur le microcontrôleur MSP430G2231.

Couverture: **M_0003**

Fonction: **SPIS_Rx**

F_00012

Nom: Initialisation capteur infrarouge

Texte: Le robot initialisera le capteur infrarouge.

Couverture: **M_0004**

Fonction: **measure_init**

F_00013

Nom: Mesure capteur infrarouge

Texte: Le robot pourra mesurer la distance entre lui-même et un objet.

Couverture: **M_0004**

Fonction: **measure**

F_00014

Nom: Conversion de mesure

Texte: Le robot pourra convertir une mesure du capteur infrarouge en centimètre.

Couverture: **M_0004**

Fonction: **convert_measure**

F_00015

Nom: Initialisation servomoteur

Texte: Le robot devra pouvoir initialiser le servomoteur.

Couverture: **M_0005**

Fonction: **servomotor_init**

F_00016

Nom: Initialisation PWM

Texte: Le robot initialisera la PWM nécessaire au servomoteur.

Couverture: **M_0005**

Fonction: **servomotor_PWM_init**

F_00017

Nom: Arrêt servomoteur

Texte: Le robot pourra arrêter le servomoteur instantanément.

Couverture: **M_0005**

Fonction: **servomotor_stop**

F_00018

Nom: Rotation servomoteur

Texte: Le robot permettra au servomoteur de tourner.

Couverture: **M_0005**

Fonction: **servomotor_set_deg**

F_00019

Nom: Balayage servomoteur

Texte: Le robot permettra au servomoteur de faire un balayage

Couverture: **M_0005**

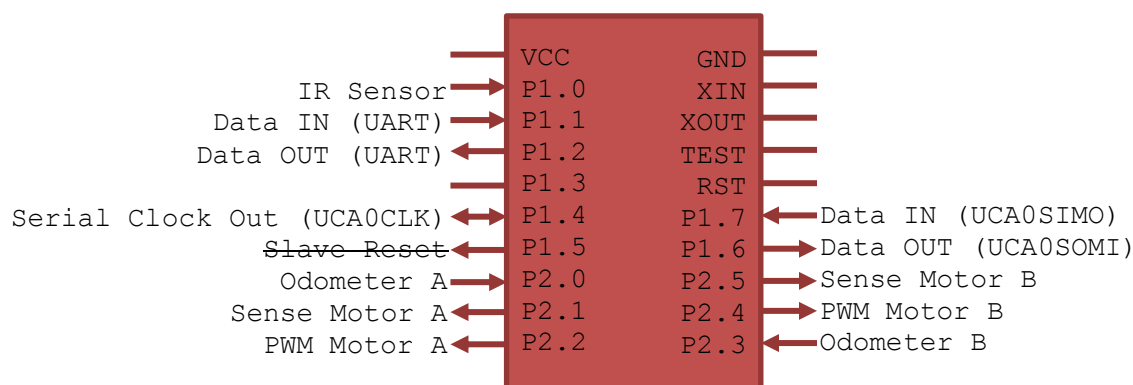
Fonction: **servomotor_sweeping**

SPÉCIFICATIONS DÉTAILLÉES

CONNECTIQUE DU ROBOT - COMMANDES ET INFORMATIONS

Afin de pouvoir clarifier les différents branchements des deux microcontrôleurs MSP 430, nous avons jugé utile de définir deux schémas représentant les entrées et sorties de chaque pin.

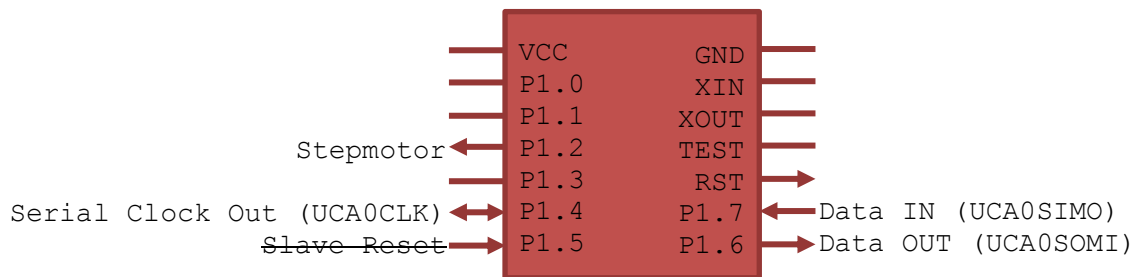
MSP430G2553



Comme nous pouvons le voir dans le précédent schéma :

- Notre communication UART est effectuée sur les ports 1.1 et 1.2
- Notre communication SPI est effectué en maître avec :
 - Émission au port 1.7
 - Réception au port 1.6
 - Horloge au port 1.4
 - ~~Sélecteur d'esclave au port 1.5~~
- Notre moteur A est piloté sur les pins 2.1 pour le sens de rotation et 2.2 pour l'activation
- Notre odomètre A envoie les informations sur le port 2.0. (*Non utilisé*)
- Notre moteur B est piloté sur les pins 2.5 pour le sens de rotation et 2.4 pour l'activation
- Notre odomètre B envoie les informations sur le port 2.3. (*Non utilisé*)
- Notre capteur infrarouge transmet son signal sur le port 1.0

MSP430G2231



Comme nous pouvons le voir dans le précédent schéma :

- Notre communication SPI est effectué comme esclave avec :
 - Émission au port 1.6
 - Réception au port 1.7
 - Horloge au port 1.4
 - ~~Sélecteur d'esclave au port 1.5~~
- Notre moteur pas à pas est piloté sur le port 1.2

ALGORITHME DE FONCTIONNEMENT

DESCRIPTION DU FONCTIONNEMENT DU ROBOT

Le robot que nous avons créé peut se déplacer dans un environnement inconnu de manière autonome ou en étant dirigé. Le code que nous avons écrit contient plusieurs étapes.

Tout d'abord toutes les fonctions utiles au robot sont initialisées (le timer A1, les communications UART et SPI, les ports des moteurs, du capteur infrarouge, du servomoteur). Après initialisation, le robot est arrêté, en mode manuel et le servomoteur effectue un balayage devant lui pour détecter des objets grâce au capteur infrarouge. Le balayage s'effectue de la manière suivante : le MSP 2553 envoie une consigne angulaire au MSP 2231 qui effectue la rotation du servomoteur pour le placer au bon angle. Ce procédé se répète infiniment en envoyant les consignes angulaires suivantes : 0°, 45°, 90°, 135°, 180°, 135°, 90°, 45°. Cela permet au servomoteur d'effectuer un balayage du capteur infrarouge.

Après initialisation, l'utilisateur décide alors ce que fera le robot. Plusieurs options s'offrent à lui :

- h : Aide,
- 8 : Faire avancer le robot,
- 2 : Faire reculer le robot,
- 4 : Faire tourner le robot à gauche
- 6 : Faire tourner le robot à droite,
- 5 : Arrêter le robot,
- 0 : Robot en mode manuel,
- 1 : Robot en mode automatique.

Si l'utilisateur choisit h, l'aide s'affiche sur son application de contrôle.

Si l'utilisateur choisit 8, le robot avance en ligne droite à vitesse maximale.

Si l'utilisateur choisit 2, le robot recule en ligne à vitesse maximale.

Si l'utilisateur choisit 4, le robot tourne à gauche de 45°.

Si l'utilisateur choisit 6, le robot tourne à droite de 45°.

Si l'utilisateur choisit 5, le robot s'arrête.

Si l'utilisateur choisit 0, le robot se met en mode manuel.

Si l'utilisateur choisit 1, le robot se met en mode automatique.

Dans le cas du mode manuel, si le robot détecte un objet, il informe l'utilisateur qu'il faut l'éviter en lui envoyant un message sur son interface : « Évitez l'objet ! ».

Dans le cas du mode automatique, si le robot détecte un objet, plusieurs cas sont possibles :

Si le robot détecte un objet à 90° degrés sur sa gauche, il tourne de 45° sur sa droite.

Si le robot détecte un objet à 45° degrés sur sa gauche, il tourne de 90° sur sa droite.

Si le robot détecte un objet devant, il fait demi-tour.

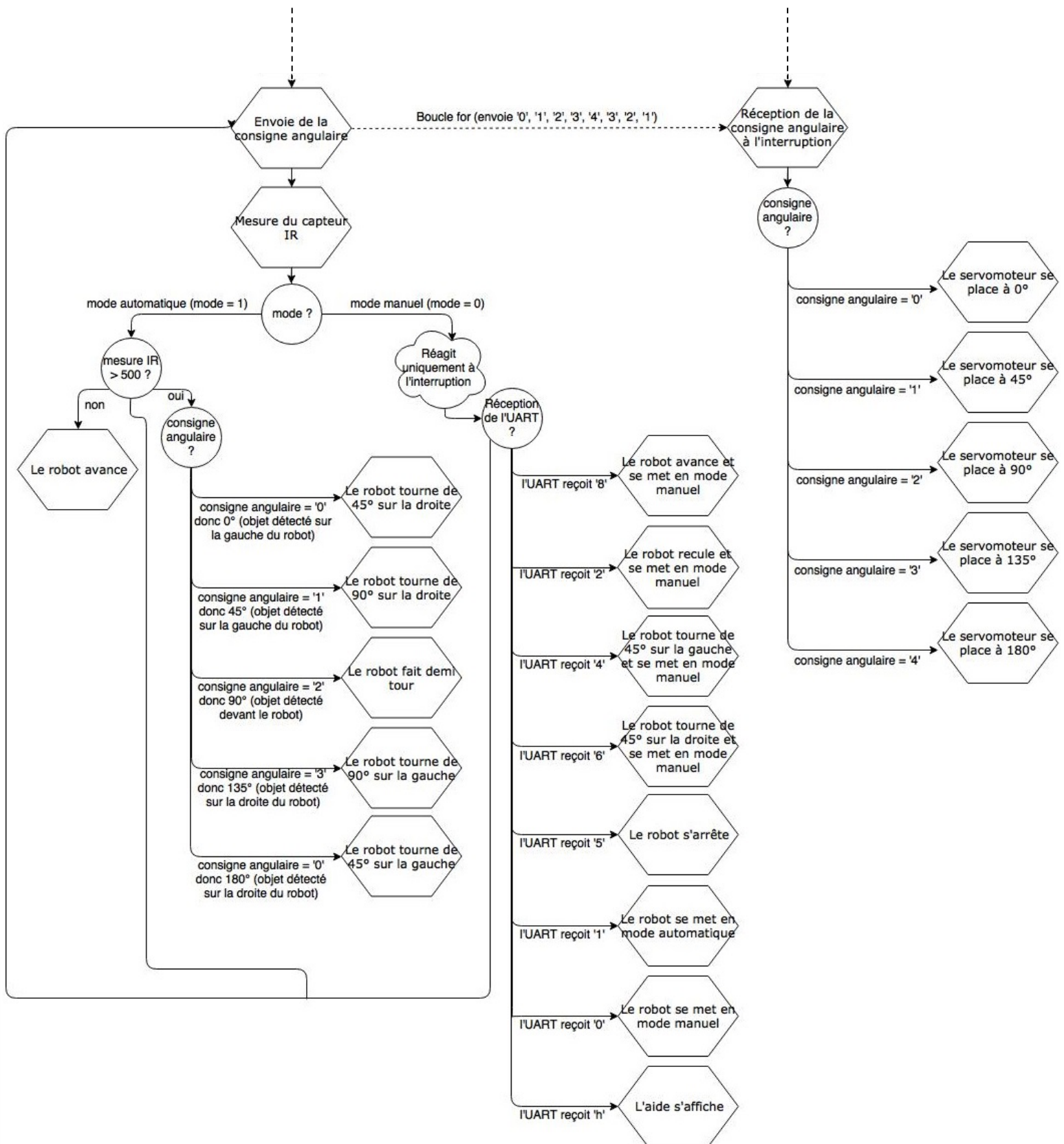
Si le robot détecte un objet à 45° degrés sur sa droite, il tourne de 90° sur sa gauche.

Si le robot détecte un objet à 90° degrés sur sa droite, il tourne de 45° sur sa gauche.

Lorsque le robot est en mode automatique, l'utilisateur peut reprendre le contrôle à tout moment en appuyant sur une touche disponible autre que 1.

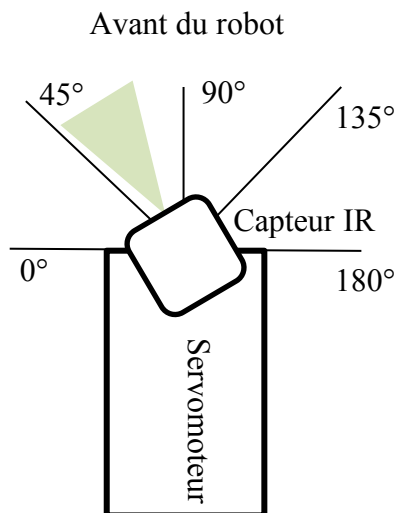
SCHEMA DU FONCTIONNEMENT DU ROBOT





QUELQUES PRECISIONS SUR LE FONCTIONNEMENT DU ROBOT

Précédemment, pour le balayage du servomoteur, nous avons défini différents angles où se place le servomoteur. Le schéma ci-dessous montre ces angles :



Pour le placement de ce servomoteur aux bons angles, le MSP 2553 envoie des caractères non signés en continu au MSP 2231 via une communication SPI (boucle for). Cependant, ne pouvant pas envoyer deux caractères d'un seul coup, nous avons défini le code suivant :

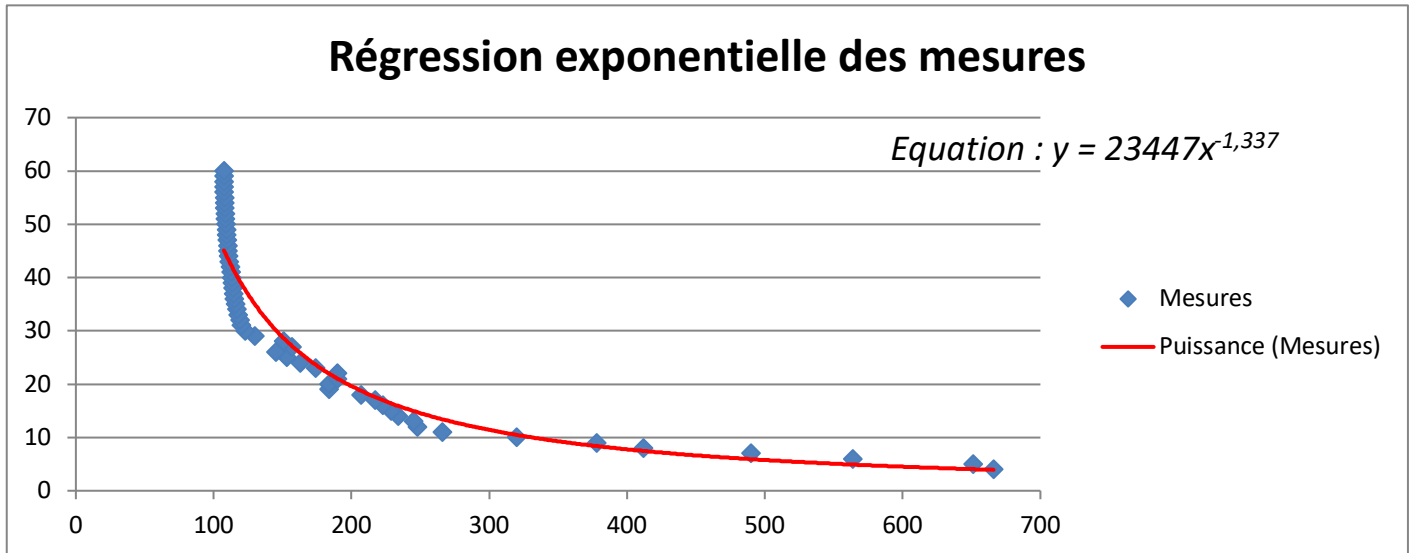
Caractère envoyé	Consigne angulaire correspondante
'0'	0°
'1'	45°
'2'	90°
'3'	135°
'4'	180°

A la réception du caractère, le MSP2231 place le servomoteur sur l'angle correspondant. Ce procédé étant répétant tout au long du fonctionnement du robot, cela permet un balayage permanent permettant de détecter les objets.

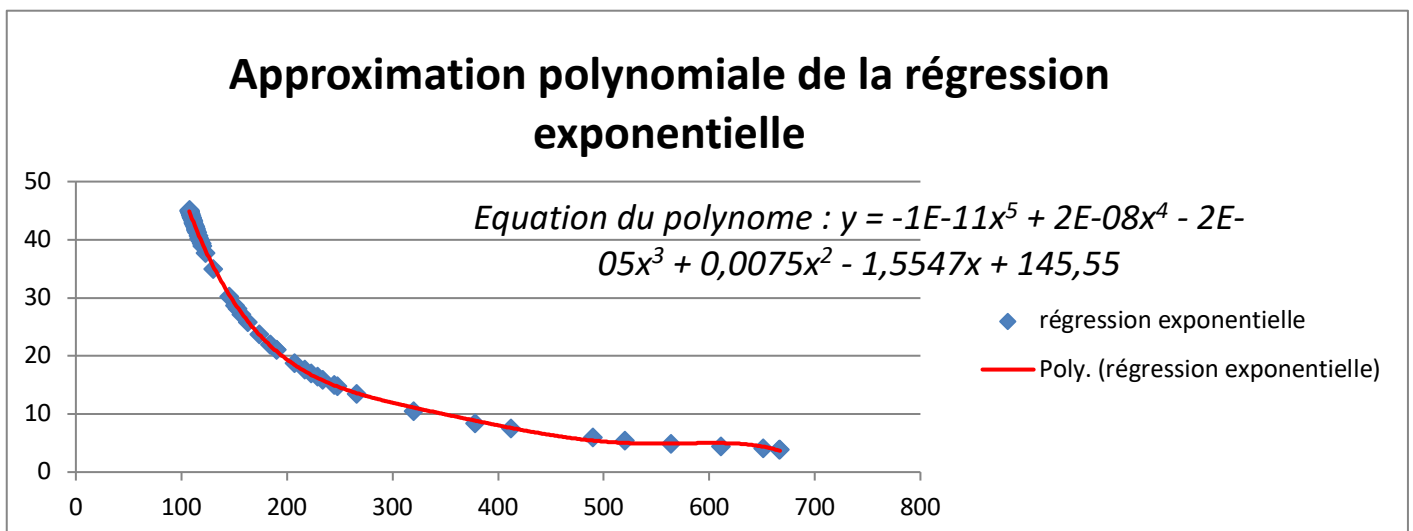
RECUPERATION DE LA DISTANCE EN MILLIMETRE

Afin de pouvoir afficher la distance d'un obstacle par rapport au capteur IR, il était nécessaire d'adapter la valeur décimale (comprise en 0 et 1023) fournie par le convertisseur analogique/numérique du microcontrôleur.

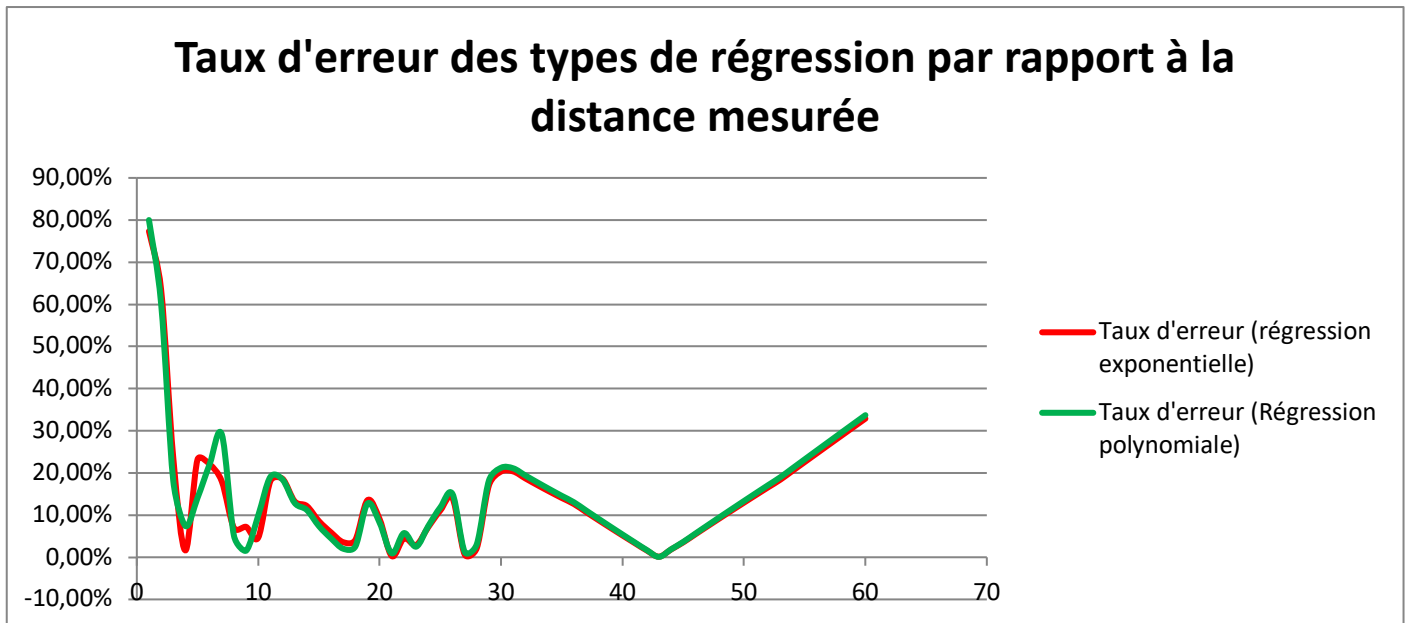
Pour se faire, une série de mesure a été prise. Un morceau de carton blanc situé en face du capteur nous a permis de saisir les valeurs retournées selon la distance. Nous avons trouvé le comportement suivant :



Les mesures montrent que le capteur suit une loi exponentielle (courbe rouge) dont l'équation est fournie dans le tableau ci-dessus. Cependant, nous savons, après expérience, que le MSP430 ne reconnaît pas la fonction `pow()` fourni par l'outil `math.h`. Nous avons alors dû chercher à effectuer une régression polynomiale de nos mesures. Afin d'éviter des ondulations trop importantes vers les valeurs comprises entre 300 et 700, notre régression s'effectuera selon la courbe de tendance exponentielle trouvée. Le graphe suivant montre l'approximation polynomiale (en rouge) :



Nous avons donc trouvé une fonction polynomiale pouvant théoriquement contourner le problème de disponibilité de la fonction `pow()`. Voici d'ailleurs ci-dessus un graphique montrant le taux d'erreur des régressions exponentielles et polynomiales par rapport aux valeurs mesurées :



Les deux courbes montrent que l'utilisation de la régression polynomiale sur l'approximation exponentielle influe peu sur le taux d'erreur d'origine.

Cependant, après expérience, nous nous sommes rendu compte que notre volonté de précision dans les mesures n'était pas adéquate quant aux capacités du matériel. En effet, le MSP430 n'est pas assez puissant pour fournir une valeur respectant l'équation. Nous avons donc dû abandonner l'idée de transmettre la valeur de distance calculée dans le MSP430. Cependant, la valeur post-ADC peut, elle, être transmise et devra être interprétée dans le logiciel Android. Nous pourrions par ailleurs passer outre l'approximation polynomiale.

MODULES

Les différents codes du projet seront séparés en différents modules dont la liste utilisée par le microcontrôleur MSP430G2553 est donnée ci-dessous :

- Un module **movement.c** accompagné du header **movement.h** comportant les fonctions :
 - Initialisation du timer gérant la vitesse des roues **init_timer_A1()**
 - Déplacement **move()**
 - Arrêt **stop()**
 - Le mode de déplacement automatique du robot **automode()**

- Un module **UART.c** accompagné du header **UART.h** comportant les fonctions :
 - Initialisation du dispositif **UART_init()**
 - Transmission de données **UART_Tx()**
 - Réception de données **UART_Rx()**
 - L'affiche de texte sur la console **envoi_msg_UART()**

- Un module **SPIM.c** accompagné du header **SPIM.h** comportant les fonctions :
 - Initialisation du dispositif **SPIM_init()**
 - Transmission de données **SPIM_Tx()**

- Un module **measure.c** accompagné du header **measure.h** comportant la fonction :
 - Initialisation de l'outil mesure **measure_init()**
 - Mesure de distance **measure()**
 - Conversion de la mesure en centimetre **convert_measure()**

Et ci-dessous les codes utilisés par le microcontrôleur MSP430G2231 :

- Un module **servomotor.c** accompagné du header **servomotor.h** comportant la fonction :
 - Initialisation du servomoteur **servomotor_init()**
 - Initialisation de la PWM du moteur **servomotor_PWM_init()**
 - Arrêt du servomoteur **servomotor_stop()**
 - Rotation du servomoteur **servomotor_set_deg()**

- Un module **SPIS.c** accompagné du header **SPIS.h** comportant les fonctions :
 - Initialisation du dispositif **SPIS_init()**
 - Réception de données **SPIS_Rx()**

FONCTIONS

Voici tout d'abord les modules utilisés par le microcontrôleur MSP430G2553 :

Dans un premier temps, le robot devra effectuer des actions de déplacement élémentaires, clarifiées ci-dessous :

MOVEMENT.C

- **VOID = init_timer_A1(VOID)** : Cette fonction initialise le timer A1 du μ C permettant de gérer la vitesse des roues gauches et droites du robot. La période vaut 100 μ s, ce qui permet de régler l'état haut de la PWM de chaque moteur (P2.2 et P2.4) directement en pourcentage.
- **VOID = move(DIRECTION, SPEED_L, SPEED_R)** : cette fonction permet au robot de se déplacer en prenant en entrée les variables *DIRECTION*, *SPEED_L* et *SPEED_R* et en ne renvoyant aucune valeur. Cette fonction est active sans délais.

La variable *DIRECTION* est un entier et spécifie la direction que le robot doit prendre. Elle pourra prendre quatre valeurs différentes :

- 1 correspond à la valeur *FORWARD* : Les moteurs A et B sont activés et tourne dans la direction avant.
- 2 correspond à la valeur *BACKWARD* : Les moteurs A et B sont activés et tourne dans la direction arrière.
- 3 correspond à la valeur *LEFT* : Les moteurs A et B sont activés et tourne dans la direction gauche.
- 4 correspond à la valeur *RIGHT* : Les moteurs A et B sont activés et tourne dans la direction droite.

Les variables *SPEED_L* et *SPEED_R* permettent respectivement de spécifier la vitesse de rotation des moteurs gauche et droit en pourcentage. Les valeurs saisies devront donc être des entiers positifs compris entre 0 et 100.

Dans le cas où les variables *DIRECTION* et *SPEED* sont hors de leurs champs de sélection, la valeur de *SPEED* sera considérée comme étant 0 et *DIRECTION* prendra la valeur 1 : *FORWARD*.

- **VOID = stop(VOID)** : cette fonction provoque l'arrêt du robot instantanément, ne prend pas de valeur en entrée et ne renvoie aucune valeur.
- **VOID = automode(MES, DIRECTION)** : Il s'agit du programme permettant le déplacement automatique du robot. Cette fonction prend en entrée un entier *MES* correspondant à la valeur décimal (entre 0 et 667, 667 étant la valeur maximale) de la mesure de distance fournit par le capteur infrarouge, ainsi que la *DIRECTION* de l'obstacle par rapport au robot, correspondant à la position angulaire du servomoteur.

Ensuite, le robot devra être capable de recevoir et d'envoyer des informations à un appareil connecté via le dispositif Bluetooth :

UART.C

- **VOID = UART_init(VOID)** : cette fonction sert à initialiser les conditions d'utilisation de l'UART spécifique au MSP430G2553.
- **VOID = UART_Tx (RECEIVE)** : cette fonction permet de transmettre une donnée à un appareil connecté via le dispositif Bluetooth.

La variable *RECEIVE* est un caractère non signé qui est transmis via le dispositif Bluetooth (i.e à l'appareil connecté au robot).

- **RECEIPT = UART_Rx(VOID)** : cette fonction permet de recevoir une donnée d'un appareil connecté via dispositif Bluetooth.

La variable *RECEIPT* est un caractère non signé provenant du dispositif Bluetooth (i.e à l'appareil connecté au robot).

- **VOID = envoi_msg_UART(*msg)** : Cette fonction permet d'envoi un texte à la console utilisateur permettant d'informer de la situation du robot.

La variable **msg* est un tableau de caractère non signé de taille non contraint qui est transmis via le dispositif Bluetooth (i.e à l'appareil connecté au robot).

De plus, le microcontrôleur MSP430G2553 devra pouvoir transmettre et recevoir des données au microcontrôleur MSP430G2231 :

SPIM.C

- **VOID = SPIM_init(VOID)** : cette fonction sert à initialiser les conditions d'utilisation de la communication via SPI. Elle ne prend et ne renvoie aucune valeur en entrée et en sortie.
- **VOID = SPIM_Tx(RECEIVE)** : cette fonction permet de transmettre une donnée via la communication SPI au second microcontrôleur.

La variable *RECEIVE* est un caractère non signé qui est transmis via la communication SPI.

Ensuite, le robot devra pouvoir mesurer la distance entre lui-même et un obstacle grâce à un capteur infrarouge :

MEASURE.C

- *VOID* = **measure_init** (*VOID*) : cette fonction permet d'initialiser la mesure du capteur infrarouge. Elle ne prend et ne renvoie aucune valeur en entrée et en sortie.
- *MES* = **measure**(*VOID*) : cette fonction permet de calculer la distance entre le robot et un objet en renvoyant un entier qui est la variable *MES* et ne prend pas de valeur en entrée. La particularité de cette fonction est qu'elle effectue dix mesures à la suite et effectue une moyenne de ces valeurs, ce qui améliore la précision de la valeur retournée.

La variable *MES* correspond à la distance entre un objet et le robot. Cette valeur sera comprise entre 0 et 1023.

- *MES_CM* = **convert_measure**(*MES*) : cette fonction permet de convertir la valeur obtenue avec la fonction **measure()** en centimètre. Elle prend en entrée un entier *MES* et renvoie en sortie un entier *MES_CM*. Cette valeur est directement dépendante des caractéristiques du capteur.

La variable *MES* correspond à la distance entre un objet et le robot. Cette valeur sera comprise entre 0 et 1023.

La variable *MES_CM* correspond à la distance entre un objet et le robot. Cette valeur sera comprise entre 40 et 300 centimètres.

A présent, voici les modules utilisés par le microcontrôleur MSP430G2231 :

Dans un premier temps, le servomoteur devra pouvoir effectuer un balayage devant le robot :

SERVOMOTOR.C

- **VOID = servomotor_init(VOID)** : cette fonction permet d'initialiser le servomoteur. Elle ne prend et ne renvoie aucune valeur en entrée et en sortie.
- **VOID = servomotor_PWM_init(VOID)** : cette fonction permet d'initialiser la PWM du servomoteur. Elle ne prend aucune valeur en entrée et ne renvoie en sortie aucune valeur. Lors de l'initialisation, cette fonction permet d'initialiser les TACCR0 (TACCR0 et TACCR1).
- **VOID = servomotor_stop(VOID)** : cette fonction permet de stopper le servomoteur instantanément. Elle ne prend aucune valeur en entrée et ne renvoie aucune valeur en sortie.
- **TACCR = servomotor_set_deg(DEG)** : cette fonction permet au servomoteur de faire une rotation. Elle prend en entrée la variable *DEG* et renvoie la valeur *TACCR*.

La variable *DEG* prend une valeur en degré qui est un entier compris entre 0 et 180.

La variable *TACCR* est un entier compris entre 500 et 2500 dans notre cas. Elle correspond à la valeur de TACCR1 pour la PWM du servomoteur et est calculée à partir de la valeur de *DEG*.

Ensuite, le microcontrôleur MSP430G2231 devra pouvoir transmettre et recevoir des données au microcontrôleur MSP430G2553 :

SPIS.C

- **VOID = SPIS_init(VOID)** : cette fonction sert à initialiser les conditions d'utilisation de la communication via SPI. Elle ne prend et ne renvoie aucune valeur en entrée et en sortie.
- **RECEIPT = SPIS_Rx(VOID)** : cette fonction permet de recevoir une donnée via la communication SPI du second microcontrôleur.

La variable *RECEIPT* est un caractère non signé provenant de la communication SPI.

TESTS UNITAIRES (BOITES NOIRES)

Nous avons réalisé tous les tests au fur et à mesure de la création de chaque fonction. Nous avons voulu rééditer ces tests pour apporter des images et donc plus de preuves, mais nous avons eu un problème de connectique avec la carte launchpad 2553 après un seul test. Nous n'avons donc pas pu ajouter d'images pour prouver nos résultats.

TEST DU MODULE « MOVEMENT.C »

FONCTION : INIT_TIMER_A1()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. init_timer_A1() : Les ports du timer sont initialisés et la période vaut 100 μ s.

RESULTATS OBTENUS

1. init_timer_A1() : **validé.**

FONCTION : MOVE_INIT()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

2. move_init() : Les ports des moteurs sont initialisés.

RESULTATS OBTENUS

2. move_init () : **validé.**

FONCTION : MOVE()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. move(*FORWARD*, 20, 80) : Les moteurs vont en marche avant, vitesse 20 / 80.
2. move(*BACKWARD*, 80, 80) : Les moteurs vont en marche arrière, vitesse 80 / 80.
3. move(*LEFT*, 40, 40) : Les moteurs permettent une rotation à gauche, vitesse 40 / 40.
4. move(*RIGHT*, 20, 20) : Les moteurs permettent une rotation à droite, vitesse 20 / 20.
5. move(10,120,-4) : Les moteurs vont en marche avant, vitesse 0/0

RESULTATS OBTENUS

1. move(*FORWARD*, 20, 80) : **Validé.**
2. move(*BACKWARD*, 80, 80) : **Validé.**
3. move(*LEFT*, 40, 40) : **Validé.**
4. move(*RIGHT*, 20, 20) : **Validé.**
5. move(10,120,-4) : **Validé.**

FONCTION : STOP()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. stop() : Les moteurs s'arrêtent. Les LED indiquant le sens de rotation ne changent pas.

RESULTATS OBTENUS

1. stop() : **validé.**

FONCTION : AUTOMODE()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. automode(150,'0') : Le servomoteur est à la position 0° et un obstacle se situe à la valeur 150/667.
2. automode(130,'1') : Le servomoteur est à la position 45° et un obstacle se situe à la valeur 130/667.
3. automode(10,'2') : Le servomoteur est à la position 90° et un obstacle se situe à la valeur 10/667.
4. automode(500,'0') : Le servomoteur est à la position 0° et un obstacle se situe à la valeur 500/667.
5. automode(500,'1') : Le servomoteur est à la position 45° et un obstacle se situe à la valeur 500/667.
6. automode(500,'2') : Le servomoteur est à la position 90° et un obstacle se situe à la valeur 500/667.
7. automode(500,'3') : Le servomoteur est à la position 135° et un obstacle se situe à la valeur 500/667.
8. automode(500,'4') : Le servomoteur est à la position 180° et un obstacle se situe à la valeur 500/667.

RESULTATS OBTENUS

1. automode(150,'0') : **validé**. Le robot ne réagit pas à la présence l'obstacle situé à sa gauche.
2. automode(130,'1') : **validé**. Le robot ne réagit pas à la présence l'obstacle situé à son avant-gauche.
3. automode(10,'2') : **validé**. Le robot ne réagit pas à la présence l'obstacle situé devant lui.
4. automode(500,'0') : **validé**. Le robot esquive l'obstacle situé à sa gauche en tournant de 45° à droite.
5. automode(500,'1') : **validé**. Le robot esquive l'obstacle situé à sa gauche en tournant de 90° à droite.
6. automode(500,'2') : **validé**. Le robot esquive l'obstacle situé à sa devant lui en faisant demi-tour.
7. automode(500,'3') : **validé**. Le robot esquive l'obstacle situé à sa droite en tournant de 90° à gauche.
8. automode(500,'4') : **validé**. Le robot esquive l'obstacle situé à sa droite en tournant de 45° à gauche.

Conclusion :

Les fonctions sont opérationnelles et prêtes à être utilisées pour diriger le robot.

Rapport LDRA du module movement.c :

movement.c

Overall Results - Percentage of metrics passing

Total Metrics: 45

Clarity Metrics: 14 (of which 4 are whole file only)

Maintainability Metrics: 11 (of which 7 are whole file only)

Testability Metrics: 13 (of which 7 are whole file only)

Procedure	All Metrics	Clarity	Maintainability	Testability
init_timer_A1	88	70	100	100
move_init	92	80	100	100
move	96	100	75	83
stop	92	80	100	100
Total for movement.c	98	100	100	100

TEST DU MODULE « MEASURE.C »

FONCTION : MEASURE_INIT()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. measure_init() : Les registres sont bien initialisés ainsi que l'ADC.

RESULTATS OBTENUS

1. measure_init() : **Validé.**

FONCTION : MEASURE()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer (supposition d'une linéarité du capteur) :

1. Distance <40 mm : La valeur obtenue en sortie de la fonction est difficilement prédictible.
2. Distance 40 mm : La valeur obtenue en sortie de la fonction vaut 667.
3. Distance entre 40 et 300 mm : La valeur obtenue en sortie de la fonction est entre 0 et 667.
4. Distance 300 mm : La valeur obtenue en sortie de la fonction vaut environ 0.
5. Distance $\rightarrow \infty$: La valeur obtenue en sortie de la fonction vaut 0.

RESULTATS OBTENUS

1. Distance <10 mm : **Validé.** La valeur obtenue en sortie de la fonction varie entre 0 et 667.
2. Distance 40 mm : **Validé.** La valeur obtenue en sortie de la fonction vaut 667.
3. Distance 40 et 300 mm : **Validé.** La valeur obtenue en sortie de la fonction varie entre 0 et 667.
4. Distance 300 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut environ 130.
5. Distance $\rightarrow \infty$: **NON VALIDE :** La valeur obtenue en sortie de la fonction peine à descendre en dessous de 110.

FONCTION : CONVERT_MEASURE()

Situation : Non utilisable avec la méthode de régression utilisée. Le microcontrôleur n'est pas assez puissant pour calculer la distance avec la précision demandée. (Voir page 16-17).

RESULTATS ATTENDUS

Liste des tests à effectuer (supposition d'une linéarité du capteur) :

6. Distance <10 mm : La valeur obtenue en sortie de la fonction vaut 1023.
7. Distance 40 mm : La valeur obtenue en sortie de la fonction vaut 1023.
8. Distance 100 mm : La valeur obtenue en sortie de la fonction vaut environ 786.
9. Distance 150 mm : La valeur obtenue en sortie de la fonction vaut environ 590.
10. Distance 200 mm : La valeur obtenue en sortie de la fonction vaut environ 393.
11. Distance 250 mm : La valeur obtenue en sortie de la fonction vaut environ 197.
12. Distance 300 mm : La valeur obtenue en sortie de la fonction vaut environ 0.
13. Distance >300 mm : La valeur obtenue en sortie de la fonction vaut 0.

RESULTATS OBTENUS

6. Distance <10 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut 660.
7. Distance 40 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut 660.
8. Distance 100 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut environ 320.
9. Distance 150 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut environ 251.
10. Distance 200 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut environ 243.
11. Distance 250 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut environ 125.
12. Distance 300 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut environ 155.
13. Distance >300 mm : **NON VALIDE** : La valeur obtenue en sortie de la fonction vaut 108 (60 cm).

Conclusion :

Il est nécessaire de trouver une régression approchant le comportement du capteur. Nous avons trouvé que cette régression est sous la forme exponentielle :

$$f(x) = Kx^{-a} \text{ avec } K == 23447 \text{ et } a = -1,337. \text{ (pour un résultat en cm)}$$

Cependant, la fonction **pow()** n'est pas compatible avec notre dispositif MSP430. Il est nécessaire de trouver une solution alternative. Une adaptation pour une régression polynomiale a été utilisée mais les contraintes de conception du matériel ne permettent pas d'obtenir la précision nécessaire pour fournir une valeur en mm. Le polynôme utilisé fût le suivant :

$$P(x) = -10^{-11}x^5 + 2 \cdot 10^{-8}x^4 - 2 \cdot 10^{-5}x^3 + 0.0075x^2 - 1.5575x + 145.55$$

TEST DU MODULE « UART.C »

FONCTION : UART_INIT()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. UART_init() : Les ports du l'UART sont tous initialisés et opérationnels pour effectuer une transmission de données.

RESULTATS OBTENUS

1. UART_init() : **Validé.**

FONCTION : UART_TX()

Situation : Testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. UART_Tx('1') : L'information voulant être transmise se retrouve bien sur le buffer d'émission.
2. UART_Tx('5') : L'information voulant être transmise se retrouve bien sur le buffer d'émission.
3. UART_Tx('9') : L'information voulant être transmise se retrouve bien sur le buffer d'émission.

RESULTATS OBTENUS

1. UART_Tx('1') : Validé.

workspace - test_2553/test_UART.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

Debug

test_2553 [Code Composer Studio - Device Debugging]

TI MSP430 USB1/MSP430 (Suspended)

main() at test_UART.c:27 0xC026

0x0000 (no symbols are defined)

Name	Value	Description
UCA0CTL1	0x80	USCI A0 Control Register 1 [Memory Mapped]
UCA0BR0	0x68	USCI A0 Baud Rate 0 [Memory Mapped]
UCA0BR1	0x00	USCI A0 Baud Rate 1 [Memory Mapped]
UCA0MCTL	0x00	USCI A0 Modulation Control [Memory Mapped]
UCA0STAT	0x00	USCI A0 Status Register [Memory Mapped]
UCA0RXBUF	0x00	USCI A0 Receive Buffer [Memory Mapped]
UCA0TXBUF	0x31	USCI A0 Transmit Buffer [Memory Mapped]
UCA0ABCTL	0x00	USCI A0 LIN Control [Memory Mapped]

```

19 P1OUT &= ~BIT0;
20
21
22 /* Test fonction : UART_init() */
23 UART_init();
24
25 /* Test fonction : UART_Tx() */
26 UART_Tx('1');
27 delay_cycles(3000000);
28 UART_Tx('5');
29 delay_cycles(3000000);
30 UART_Tx('9');
31
32 /* Test fonction : UART_Rx() */
33 //__enable_interrupt();
  
```

test_UART.c measure.c

test_2553

MSP430: Flash/FRAM usage is 290 bytes. RAM usage is 80 bytes.

Runtime Object View

Viewable Modules

Runtime Object View

Connect Target

Executable path

2. UART_Tx('5') : Validé.

workspace - test_2553/test_UART.c - Code Composer Studio

File Edit View Project Tools Run Scripts Window Help

Debug

test_2553 [Code Composer Studio - Device Debugging]

TI MSP430 USB1/MSP430 (Suspended)

main() at test_UART.c:29 0xC04E

0x0000 (no symbols are defined)

Name	Value	Description
UCA0CTL1	0x80	USCI A0 Control Register 1 [Memory Mapped]
UCA0BR0	0x68	USCI A0 Baud Rate 0 [Memory Mapped]
UCA0BR1	0x00	USCI A0 Baud Rate 1 [Memory Mapped]
UCA0MCTL	0x00	USCI A0 Modulation Control [Memory Mapped]
UCA0STAT	0x00	USCI A0 Status Register [Memory Mapped]
UCA0RXBUF	0x00	USCI A0 Receive Buffer [Memory Mapped]
UCA0TXBUF	0x35	USCI A0 Transmit Buffer [Memory Mapped]
UCA0ABCTL	0x00	USCI A0 LIN Control [Memory Mapped]

```

19 P1OUT &= ~BIT0;
20
21
22 /* Test fonction : UART_init() */
23 UART_init();
24
25 /* Test fonction : UART_Tx() */
26 UART_Tx('1');
27 delay_cycles(3000000);
28 UART_Tx('5');
29 delay_cycles(3000000);
30 UART_Tx('9');
31
32 /* Test fonction : UART_Rx() */
33 //__enable_interrupt();
  
```

test_UART.c measure.c

test_2553

MSP430: Flash/FRAM usage is 290 bytes. RAM usage is 80 bytes.

Runtime Object View

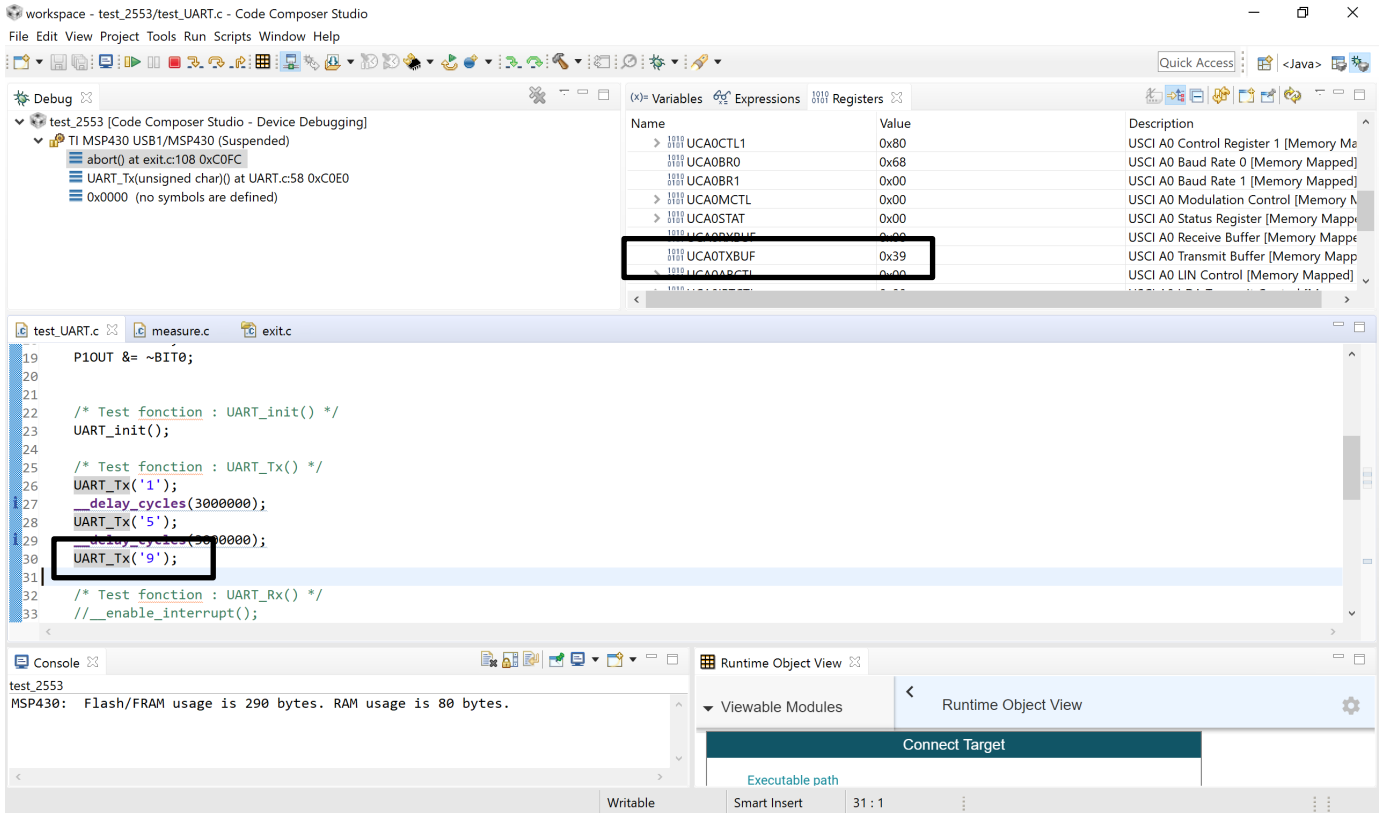
Viewable Modules

Runtime Object View

Connect Target

Executable path

3. UART_Tx('9') : Validé.



FONCTION : UART_RX()

Situation :

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. UART_Rx() : L'information que le μ C doit recevoir se retrouve bien sur le buffer de réception.
2. UART_Rx() : La led 1.0 s'allume si la valeur reçue est 0x31.
3. UART_Rx() : La led 1.0 s'éteint si la valeur reçue est 0x30.

RESULTATS OBTENUS

1. UART_Rx() : Validé.
2. UART_Rx() : Validé.
3. UART_Rx() : Validé.

FONCTION : ENVOI_MSG_UART()

Situation :**RESULTATS ATTENDUS**

Liste des tests à effectuer :

1. envoi_msg_UART('Bonjour') : Le message « Bonjour » doit être transmis et affiché sur l'application.
2. envoi_msg_UART('Test UART') : Le message « Test UART » doit être transmis et affiché sur l'application.

RESULTATS OBTENUS

1. envoi_msg_UART('Bonjour') : **Validé.**
2. envoi_msg_UART('Test UART') : **Validé.**

Conclusion :

La communication entre un appareil externe et le dispositif Bluetooth est fonctionnelle.

TEST DU MODULE « SPIM.C »

FONCTION : SPIM_INIT()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. SPIM_init() : Les ports du SPI sont tous initialisés et opérationnels pour effectuer une transmission de données.

RESULTATS OBTENUS

1. SPIM_init() : **Validé.**

FONCTION : SPIM_TX()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. SPIM_Tx(0x30) : L'information voulant être transmise se retrouve bien sur le buffer d'émission.
2. SPIM_Tx(0x31) : L'information voulant être transmise se retrouve bien sur le buffer d'émission.

RESULTATS OBTENUS

1. SPIM_Tx(0x30) : **Validé.**
2. SPIM_Tx(0x31) : **Validé.**

Conclusion :

L'émission d'instruction du master est établie et fonctionnelle.

TEST DU MODULE « SPIS.C »

FONCTION : SPIS_INIT()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. SPIS_init() : Les ports du SPI sont tous initialisés et opérationnels pour effectuer une transmission de données.

RESULTATS OBTENUS

1. SPIS_init() : **Validé.**

USI		
> USICTL0	0xF6	USI Control Register 0 [Memory Mapped]
> USICTL1	0x10	USI Control Register 1 [Memory Mapped]
> USICKCTL	0x00	USI Clock Control Register [Memory Mapped]
> USICNT	0x08	USI Bit Counter Register [Memory Mapped]
USISRL	0x23	USI Low Byte Shift Register [Memory Mapped]
USISRH	0x00	USI High Byte Shift Register [Memory Mapped]
USICTL	0x10F6	USI Control Register [Memory Mapped]
USICCTL	0x0800	USI Clock and Counter Control Register [Memory Mapped]
USISR	0x0023	USI Shift Register [Memory Mapped]

FONCTION : SPIS_RX()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. SPIS_Rx() : L'information transmise par le maître se retrouve bien sur le buffer USISRL. Si l'information reçue est 0x30, la led du pin 1.0 s'éteint.
2. SPIS_Rx() : L'information transmise par le maître se retrouve bien sur le buffer USISRL. Si l'information reçue est 0x31, la led du pin 1.0 s'éteint.

RESULTATS OBTENUS

1. SPIS_Rx() : L'information reçue est 0x30, la led du pin 1.0 s'éteint. **Validé.**
2. SPIS_Rx() : L'information reçue est 0x31, la led du pin 1.0 s'allume. **Validé.**

Conclusion :

La communication entre le master et le slave est établie et fonctionnelle.

TEST DU MODULE « SERVOMOTOR.C »

FONCTION : SERVOMOTOR_INIT()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. servomotor_init() : Le port du servomoteur est initialisé et opérationnel pour recevoir la PWM.

RESULTATS OBTENUS

1. servomotor_init() : **Validé.**

▼ Port_1_2			
> 1010 0101 P1IN	0x02	<div style="border: 1px solid black; padding: 5px; display: inline-block;"> P1DIR = 0000 0100 P1SEL = 0000 0100 </div>	Port 1 Input [Memory Mapped]
> 1010 0101 P1OUT	0xFF		Port 1 Output [Memory Mapped]
> 1010 0101 P1DIR	0x04		Port 1 Direction [Memory Mapped]
> 1010 0101 P1IFG	0x00		Port 1 Interrupt Flag [Memory Mapped]
> 1010 0101 P1IES	0xFF		Port 1 Interrupt Edge Select [Memory Mapped]
> 1010 0101 P1IE	0x08		Port 1 Interrupt Enable [Memory Mapped]
> 1010 0101 P1SEL	0x04		Port 1 Selection [Memory Mapped]
> 1010 0101 P1REN	0x00		Port 1 Resistor Enable [Memory Mapped]

FONCTION : SERVOMOTOR_PWM_INIT()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. servomotor_PWM_init() : Les registres nécessaires pour effectuer la PWM sont bien initialisés de manière à avoir une période de 20 ms. De plus, la PWM est réglée pour que le servomoteur se place à la position 0°.

RESULTATS OBTENUS

1. servomotor_PWN_init() : **Validé.**

▼ 1010 0101 Timer_A2		
1010 0101 TAIV	0x0000	Timer A Interrupt Vector Word [Memory Mapped]
> 1010 0101 TACTL	0x0211	Timer A Control [Memory Mapped]
> 1010 0101 TACCTL0	0x0409	Timer A Capture/Compare Control 0 [Memory Mapped]
▼ 1010 0101 TACCTL1	0x00E1	Timer A Capture/Compare Control 1 [Memory Mapped]
1010 0101 CM	00 - CM_0	Capture mode 1
1010 0101 CCIS	00 - CCIS_0	Capture input select 1
1010 0101 SCS	0	Capture synchronize
1010 0101 SCCI	0	Latched capture signal (read)
1010 0101 CAP	0	Capture mode: 1 / Compare mode: 0
1010 0101 OUTMOD	111 - OUTMOD_7	Output mode 2
1010 0101 CCIE	0	Capture/compare interrupt enable
1010 0101 CCI	0	Capture input signal (read)
1010 0101 OUT	0	PWM Output signal if output mode 0
1010 0101 COV	0	Capture/compare overflow flag
1010 0101 CCIFG	1	Capture/compare interrupt flag
1010 0101 TAR	0x3E98	Timer A Counter Register [Memory Mapped]
1010 0101 TACCR0	0x4E20	Timer A Capture/Compare 0 [Memory Mapped]
1010 0101 TACCR1	0x01F4	Timer A Capture/Compare 1 [Memory Mapped]

FONCTION : SERVOMOTOR_STOP()

Situation : Non testée.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. servomotor_stop() : le servomoteur s'arrête de tourner instantanément.

RESULTATS OBTENUS

1. servomotor_stop() : **Validé**. (Appui sur le bouton P1.3, arrêt du moteur)

> 1010 0101 P1DIR	0x00	Port 1 Direction [Memory Mapped]
-------------------	------	----------------------------------

FONCTION : SERVOMOTOR_SET_DEG()

Situation : Fonctionnelle.

RESULTATS ATTENDUS


Liste des tests à effectuer :

1. servomotor_set_deg(0) : Le registre TACCR1 prend la valeur 500 et le servomoteur se place à la position 0°.
2. servomotor_set_deg(45) : Le registre TACCR1 prend la valeur 1000 et le servomoteur se place à la position 45°.
3. servomotor_set_deg(90) : Le registre TACCR1 prend la valeur 1500 et le servomoteur se place à la position 90°.
4. servomotor_set_deg(135) : Le registre TACCR1 prend la valeur 2000 et le servomoteur se place à la position 135°.

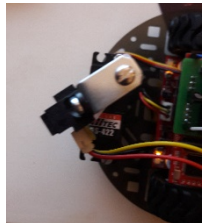
5. servomotor_set_deg(180) : Le registre TACCR1 prend la valeur 2500 et le servomoteur se place à la position 180°.
6. servomotor_set_deg(53) : Le registre TACCR1 prend la valeur 1500 et le servomoteur se place à la position 90°.
7. Pour une valeur en entrée non comprise entre 0 et 180 inclus, le registre TACCR1 prend la valeur 0 et le servomoteur s'arrête.

RESULTATS OBTENUS

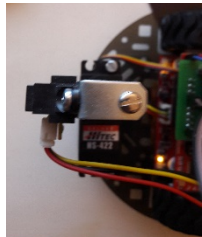
1. servomotor_set_deg (0) : **Validé.**

1010 0101 TACCR1	0x01F4	Timer A Capture/Compare 1 [Memory Mapped]
		

2. servomotor_set_deg (45) : **Validé.**

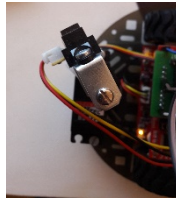
1010 0101 TACCR1	0x03E8	Timer A Capture/Compare 1 [Memory Mapped]
		

3. servomotor_set_deg (90) : **Validé.**

1010 0101 TACCR1	0x05DC	Timer A Capture/Compare 1 [Memory Mapped]
		

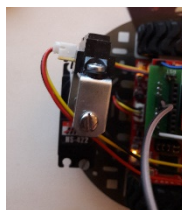
4. servomotor_set_deg 135) : **Validé.**

1010 0101 TACCR1	0x07D0	Timer A Capture/Compare 1 [Memory Mapped]
---------------------	--------	---



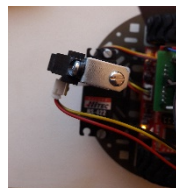
5. servomotor_set_deg (180) : **Validé.**

1010 0101 TACCR1	0x09C4	Timer A Capture/Compare 1 [Memory Mapped]
---------------------	--------	---



6. servomotor_set_deg(53) : **Validé.**

1010 0101 TACCR1	0x05DC	Timer A Capture/Compare 1 [Memory Mapped]
---------------------	--------	---



Conclusion :

Le servomoteur est fonctionnel et prêt à être utilisé.

TESTS D'INTEGRATION

TEST DE LA COMMUNICATION SPI

FONCTIONS : SPIM_TX() ET SPIS_RX()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. SPIM_Tx(0x30) et SPIM_Rx() : L'information voulant être transmise se retrouve bien sur le buffer d'émission et l'information transmise par le maître se retrouve bien sur le buffer USISRL et la led correspondant au pin 1.0 s'éteint.
2. SPIM_Tx(0x31) et SPIM_Rx() : L'information voulant être transmise se retrouve bien sur le buffer d'émission et l'information transmise par le maître se retrouve bien sur le buffer USISRL et la led correspondant au pin 1.0 s'allume.

RESULTATS OBTENUS

1. SPIM_Tx(0x30) et SPIM_Rx() : **Validé.**
2. SPIM_Tx(0x31) et SPIM_Rx() : **Validé.**

TEST DE LA COMMUNICATION UART AVEC LE ROBOT

FONCTIONS : UART_RX(), MOVE() ET STOP()

Situation : Fonctionnelle.

RESULTATS ATTENDUS

Liste des tests à effectuer :

1. Envoie de '8' avec l'application. L'information que le μ C doit recevoir se retrouve bien sur le buffer de réception et le robot doit avancer avec la fonction move(FORWARD,100,100).
2. Envoie de '2' avec l'application. L'information que le μ C doit recevoir se retrouve bien sur le buffer de réception et le robot doit reculer avec la fonction move(BACKWARD,100,100).
3. Envoie de '4' avec l'application. L'information que le μ C doit recevoir se retrouve bien sur le buffer de réception et le robot doit tourner à gauche avec la fonction move(LEFT,100,100).
4. Envoie de '5' avec l'application. L'information que le μ C doit recevoir se retrouve bien sur le buffer de réception et le robot s'arrêter avec la fonction stop().

RESULTATS OBTENUS

1. Après envoi de '8' avec l'application, l'information se trouve dans le buffer de réception et le robot avance. **Validé.**
2. Après envoi de '2' avec l'application, l'information se trouve dans le buffer de réception et le robot recule. **Validé.**
3. Après envoi de '4' avec l'application, l'information se trouve dans le buffer de réception et le robot tourne à gauche. **Validé.**
4. Après envoi de '5' avec l'application, l'information se trouve dans le buffer de réception et le robot s'arrête. **Validé.**

CONCLUSION

Durant ce projet, nous avons pu apprendre à maîtriser différents bus de communications séries (SPI et UART par un interfaçage Bluetooth) ainsi qu'à travailler sur la qualité du code en respectant les normes en vigueur et le principe du cycle en V.

Nous avons eu affaire à plus de soucis matériels que logiciels durant ce projet. Une grande partie du temps attribué à la programmation a été gâché à cause d'un servomoteur défectueux ainsi qu'une poignée de cartes MSP430G2553 et MSP430G2231 qui présentaient des problèmes internes. Persuadés que nos erreurs provenaient de nos codes car étant prosélytes en la matière, ce n'est qu'à une semaine de l'issue du projet que nous avons pu, avec un matériel fonctionnel, constater que l'intégralité de nos codes semblaient respecter les critères du cahier des charges.

Outre les problèmes matériels, LDRA a présenté quelques soucis de fonctionnements inexpliqués qui ont fait perdre un temps précieux durant la phase de test. Les analyses des métriques et du respect de la norme MISRA 2012 n'ont pu être finalisés dans leur totalité et les rapports enregistrés. Tout ce que nous pouvons dire est que le principal problème concernant la maintenabilité du code était une forte quantité de nœuds dans les fonctions usant de fonctions switch. Pour tout le reste, nos modules respectaient les critères de clartés et de testabilité au minimum à 80%.

Dans l'ensemble, les principales difficultés concernant le code se rapportaient au bus de communication *Serial Peripheral Interface* (SPI) où chacun des codes que nous avons tenté d'écrire ne fonctionnait pas. Certains que nous avons testé fonctionnaient sur des cartes différentes des nôtres (ce qui nous a mis la puce à l'oreille concernant les potentiels défauts techniques que nous pouvions avoir), d'autres étaient instables. La solution finale fût d'utiliser un code fourni par M. Duchemin qui était parfaitement adapté pour éviter tout problème de décrochage de la communication.