pilotage de robot

# Compte rendu de projet

Table des matières

[Introduction 3](#_Toc509419222)

[Spécifications générales 4](#_Toc509419223)

[Structure 4](#_Toc509419224)

[Spécifications détaillées 5](#_Toc509419225)

[Connectique du robot - Commandes et informations 5](#_Toc509419226)

[MSP430G2553 5](#_Toc509419227)

[MSP430G2231 6](#_Toc509419228)

[Modules 7](#_Toc509419229)

[Fonctions 8](#_Toc509419230)

[Algorithme de fonctionnement 12](#_Toc509419231)

[Description du fonctionnement du robot 12](#_Toc509419232)

[Conclusion 15](#_Toc509419233)

# Introduction

Durant notre enseignement de deuxième année, nous avions pour but de concevoir un robot pilotable et autonome et ce en introduisant des notions de 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 communicant 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 à répondre aux différentes problématiques mises en avant dans la suite de ce rapport.

# Spécifications générales

## Structure

Grâce au matériel fourni, nous avons définit la structure du robot qui suivra la configuration suivante :

SaMBot

MSP430G2553

MSP430G2231

Bluetooth

Accéléromètre

Capteur IR

Servomoteur

SPI

UART

Moteur A

+ Odomètre

Moteur B

+ Odomètre

Distance entre le robot et un objet

Consigne angulaire

# 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

IR Sensor

Data IN (UART)

Data OUT (UART)

Serial Clock Out (UCA0CLK)

~~Slave Reset~~

Odometer A

Sense Motor A

PWM Motor A

Data IN (UCA0SIMO)

Data OUT (UCA0SOMI)

Sense Motor B

PWM Motor B

Odometer B

VCC

P1.0

P1.1

P1.2

P1.3

P1.4

P1.5

P2.0

P2.1

P2.2

GND

XIN

XOUT

TEST

RST

P1.7

P1.6

P2.5

P2.4

P2.3

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

Stepmotor

Serial Clock Out (UCA0CLK)

~~Slave Reset~~

Data IN (UCA0SIMO)

Data OUT (UCA0SOMI)

VCC

P1.0

P1.1

P1.2

P1.3

P1.4

P1.5

GND

XIN

XOUT

TEST

RST

P1.7

P1.6

Comme nous pouvons le voir dans le précédent schéma :

* Notre communication SPI est effectué comme esclave avec :
  + Emission 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.

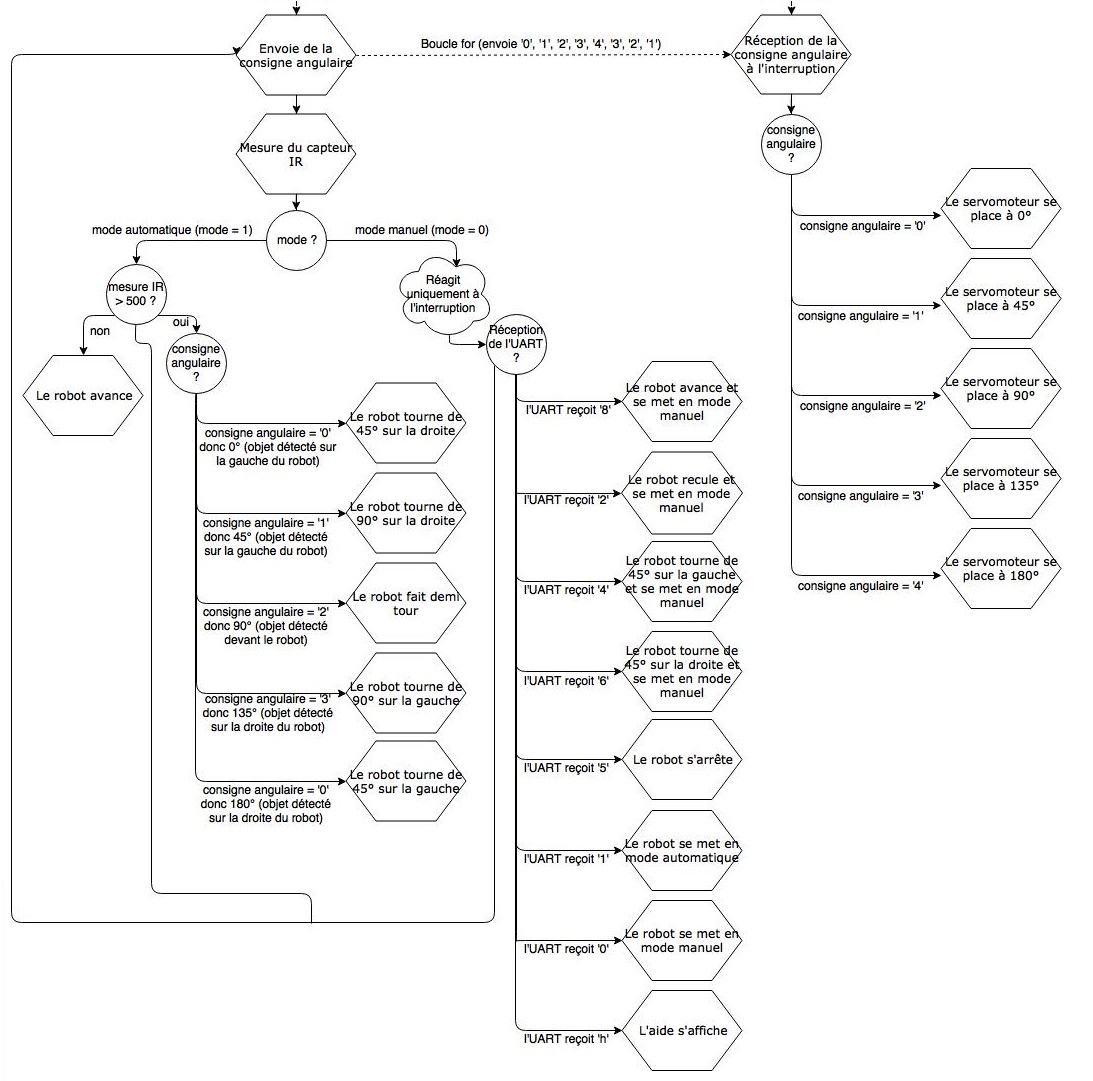
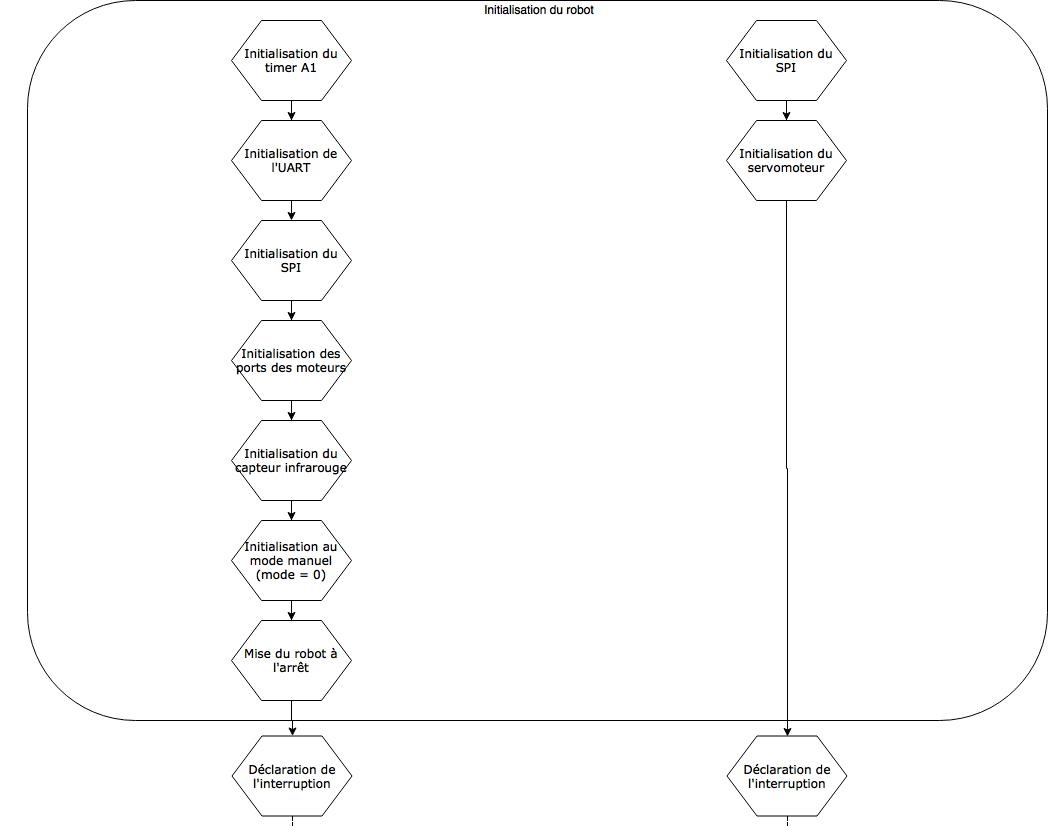
Si l’utilisateur choisit h, l’aide s’affiche sur son application de contrôle.  
Si l’utilisateur choisit 8, le robot avance en ligne droite à vitesse maximale.  
Si l’utilisateur choisit 2, le robot recule en ligne à vitesse maximale.  
Si l’utilisateur choisit 4, le robot tourne à gauche de 45°.  
Si 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.

### Schéma de fonctionnement du robot



### Quelques précisions 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 :

Servomoteur

Capteur IR

45°

90°

135°

180°

0°

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 permanant permettant de détecter les objets.

## Récupération de la distance en millimètre

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 après transformation de la valeur de tension grâce à 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 reconnait 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 une 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 pourrons 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 µC permettant de gérer la vitesse des roues gauches et droites du robot. La période vaut 100 µs, 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 ne fournit pas 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 sortir 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’initialisée 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.

# Conclusion

Durant cet enseignement, nous avons pu apprendre à maîtriser différents types de communication série. La communication *Serial Peripheral Interface* (SPI), la communication *Universal Asynchronous Receiver Transmitter* (UART) et la communication Bluetooth.

L’établissement de la communication UART entre notre robot et un dispositif externe s’est effectué sans problème. Cependant la communication SPI a été abordée avec plus de difficultés. En effet, nous avons tenté d’effectuer la communication en vain car notre microcontrôleur 2231 était grillé. Cela nous a fait perdre un temps précieux avant de comprendre l’origine du disfonctionnement. Après avoir changé de microcontrôleur, nous avons réussi à établir la communication SPI grâce au code de monsieur DUCHEMIN. De plus, une fois n’est pas coutume, notre servomoteur ne fonctionnait pas. Nous avons testé plusieurs codes avant de détecter le problème. Après changement de matériel, nous avons réalisé rapidement une fonction de balayage. Après la réalisation de toutes les fonctions nécessaires, nous les avons toutes assemblées sans difficulté.

Durant ce projet, nous avons pu voir qu’il est difficile d’identifier une erreur. En effet, elle peut venir du logiciel ou du matériel.