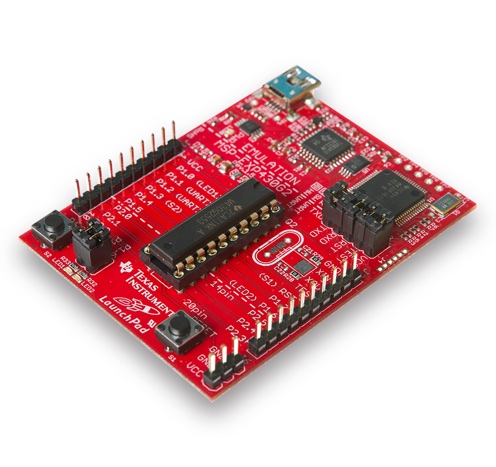
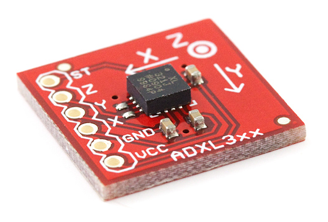
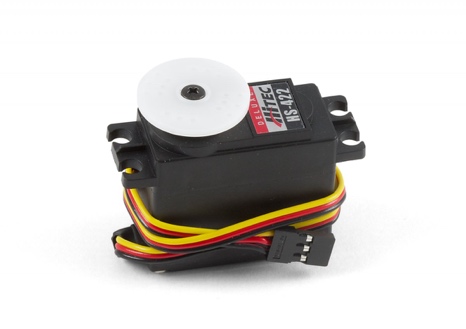
Spécifications générales

Afin de pouvoir respecter le cahier des charges, le robot devra respecter les consignes appliquées au cahier des charges fourni.

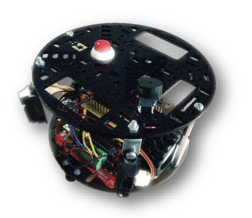
Tout d’abord, il nous faut du matériel pour réaliser le robot. Nous disposons de :

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



* 1 module bluetooth RN-42
* 1 accéléromètre
* 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.



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

**SaMBot**

MSP430G2553

MSP430G2231

Bluetooth

Accéléromètre

Capteur IR

Servomoteur

SPI

UART

Consigne angulaire

## 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 :
  + Déplacement **move()**
  + Arrêt **stop()**
* 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()**
* Un module **SPIM.c** accompagné du header **SPIM.h** comportant les fonctions :
  + Initialisation du dispositif **SPIM\_init()**
  + Transmission de données **SPIM\_Tx()**
  + Réception de données **SPIM\_Rx()**
* 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()**
* Un module **accelerometer.c** accompagné du header **accelerometer.h** comportant les fonctions :
  + Initialisation de l’accéléromètre **acc\_init()**
  + Saisie de la position initiale du robot **pos\_init()**
  + Saisie de la position actuelle du robot **pos\_current()**

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()**
  + Balayage du servomoteur **servomotor\_sweeping()**
* Un module **SPIS.c** accompagné du header **SPIS.h** comportant les fonctions :
  + Initialisation du dispositif **SPIS\_init()**
  + Transmission de données **SPIS\_Tx()**
  + Réception de données **SPIS\_Rx()**
* …

## Fonctions

Voici tout d’abord les modules utilisées par le microcontrôleur MSP430G2553 :

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

### movement.c

* *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.

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

### 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 vi 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).

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.

* *RECEIPT* = **SPIM\_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.

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 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 ave la fonction **measure()** en centimètre. Elle prend en entrée un entier *MES* et renvoie en sortir un entier *MES\_CM*.

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 utilisées 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’initialisé 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*.

* *VOID* = **servomotor\_sweeping(***VOID***)** : cette fonction permet au servomoteur d’effectuer en continue un balayage devant lui. Elle ne prend aucune valeur en entrée et ne renvoie rien en sortie.

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.
* *VOID* = **SPIS\_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.

* *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.

**De plus, le robot devra pouvoir se déplacer en mode manuel. Pour ce faire, l’utilisateur du robot pourra le guider grâce à une application qui communiquera en Bluetooth.**

### 

Tests unitaires

# Test du module « movement.c »

## Fonction : move()

**Situation : PAS ENCORE TESTE !**

### Résultats attendus

Listes des tests à effectuer :

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

### Résultats obtenus

…

## Fonction : stop()

### Résultats attendus

Listes des tests à effectuer :

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

### Résultats obtenus

…

# Test du module « measure.c »

## Fonction : measure\_init()

**Situation : PAS ENCORE TESTE !**

### Résultats attendus

Listes des tests à effectuer :

1. measure\_init() : Les registres sont bien initialisé ainsi que l’ADC.

### Résultats obtenus

…

## Fonction : measure()

### Résultats attendus

Listes des tests à effectuer (supposition d’une linéarité du capteur) :

1. Distance <10 mm : La valeur obtenue en sortie de la fonction vaut 1023.
2. Distance 40 mm : La valeur obtenue en sortie de la fonction vaut 1023.
3. Distance 100 mm : La valeur obtenue en sortie de la fonction vaut environ 786.
4. Distance 150 mm : La valeur obtenue en sortie de la fonction vaut environ 590.
5. Distance 200 mm : La valeur obtenue en sortie de la fonction vaut environ 393.
6. Distance 250 mm : La valeur obtenue en sortie de la fonction vaut environ 197.
7. Distance 300 mm : La valeur obtenue en sortie de la fonction vaut environ 0.
8. Distance >300 mm : La valeur obtenue en sortie de la fonction vaut 0.

### Résultats obtenus

…