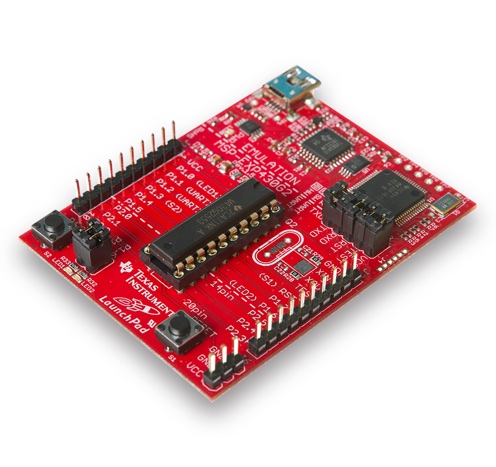
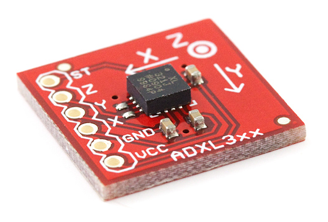
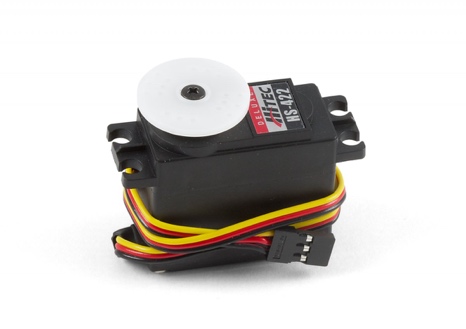
Spécifications générales

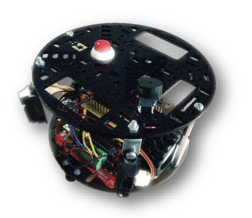
Afin de pouvoir respecter le cahier des charges, le robot devra respecter les consignes appliquées au cahier des charges fourni.

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 **accelerometer.c** accompagné du header **accelerometer.h** comportant les fonctions :
  + Initialisation de l’accéléromètre **acc\_init()**
  + Saisie de la position initiale du robot **init\_pos()**
  + Saisie de la position actuelle du robot **current**\_**pos()**

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

* Un module **scanner.c** accompagné du header **scanner.h** comportant les fonctions :
  + Mesure de distance **IR\_sensor()**
  + Mesure d’angle **servomotor()**
* 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

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

### 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 ne renvoie aucune valeur. Cette fonction est active sans délais.
* La variable *DIRECTION* spécifie la direction que le robot doit prendre. Elle pourra prendre quatre valeurs différentes :
  + *FORWARD*: Les moteurs A et B sont activés et tourne dans la direction avant.
  + *BACKWARD*: Les moteurs A et B sont activés et tourne dans la direction arrière.
  + *LEFT*: Les moteurs A et B sont activés et tourne dans la direction gauche.
  + *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 *FORWARD*.

* *VOID* = **stop()**: cette fonction provoque l’arrêt du robot instantanément, ne prend pas de valeur en entrée et ne renvoie aucune valeur.

### scanner.c

Ensuite, le robot devra pouvoir éviter tout objet se trouvant sur son passage. Pour cela, un capteur infrarouge fixé sur un servomoteur permettra de détecter tout objet se trouvant devant le robot dans un rayon de 180 degrés. Les différentes fonctions utiles sont présentées ci-dessous :

* *DISTANCE* = **IR\_sensor()**: cette fonction permet de calculer la distance entre le robot et un objet en renvoyant la variable *DISTANCE* et ne prend pas de valeur en entrée.

La variable *DISTANCE* correspond à la distance entre un objet et le robot en millimètres. D’après le capteur infrarouge utilisé, cette valeur sera comprise entre 40 et 300mm. Dans le cas où la valeur de *DISTANCE* serait hors de cet intervalle, la valeur renvoyée serait *NULL*.

* *ANGLE* = **servomotor()** : cette fonction permet de balayer l’espace grâce au capteur infrarouge devant le robot en renvoyant la variable *ANGLE* et ne prend pas de valeur en entrée.

La variable *ANGLE* correspond à la direction du capteur infrarouge en degrés. La mesure des valeurs s’effectuera dans le sens trigonométrique et seront comprises entre 0 et 180 degrés. (0 à droite, 90 devant, 180 à gauche). Si la valeur de *ANGLE* est en dehors de cet intervalle, la valeur renvoyée sera *NULL*.

**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.**

### UART.c

* VOID = **UART\_init()**: 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 *RECEIVE* **(DE TYPE A DEFINIR)** au dispositif Bluetooth (i.e à l’appareil connecté au robot).
* *RECEIPT* = **UART\_Rx()** : cette fonction permet de recevoir une donnée *RECEIPT* **(DE TYPE A DEFINIR)** provenant du dispositif bluetooth (i.e à l’appareil connecté au robot).

### 

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

…