pilotage de robot

# Compte rendu de projet

*Lien GitHub :* [*https://github.com/hakamir/LD\_Sambot*](https://github.com/hakamir/LD_Sambot)

Table des matières

[**Introduction** 3](#_Toc509773598)

[**Spécifications générales** 4](#_Toc509773599)

[Matériel 4](#_Toc509773600)

[Structure 5](#_Toc509773601)

[Exigences systèmes du logiciel 5](#_Toc509773602)

[Exigences architecturales du logiciel 6](#_Toc509773603)

[Exigences détaillées du logiciel 7](#_Toc509773604)

[**Spécifications détaillées** 10](#_Toc509773605)

[Connectique du robot - Commandes et informations 10](#_Toc509773606)

[MSP430G2553 10](#_Toc509773607)

[MSP430G2231 11](#_Toc509773608)

[Algorithme de fonctionnement 12](#_Toc509773609)

[Description du fonctionnement du robot 12](#_Toc509773610)

[Schéma du fonctionnement du robot 13](#_Toc509773611)

[Quelques précisions sur le fonctionnement du robot 15](#_Toc509773612)

[Récupération de la distance en millimètre 16](#_Toc509773613)

[Modules 18](#_Toc509773614)

[Fonctions 19](#_Toc509773615)

[**Tests unitaires (boîtes noires)** 23](#_Toc509773616)

[Test du module « movement.c » 23](#_Toc509773617)

[Test du module « measure.c » 27](#_Toc509773618)

[Test du module « UART.c » 29](#_Toc509773619)

[Test du module « SPIM.c » 33](#_Toc509773620)

[Test du module « SPIS.c » 34](#_Toc509773621)

[Test du module « servomotor.c » 35](#_Toc509773622)

[**Tests d’intégration** 39](#_Toc509773623)

[Test de la communication spi 39](#_Toc509773624)

[Test de la communication UART avec le robot 39](#_Toc509773625)

[**Conclusion** 41](#_Toc509773626)

# Introduction

Durant notre de notre enseignement de 2e année, nous avions 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 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 répondre aux différentes problématiques mise en avant dans la suite de ce rapport.

# Spécifications générales

## Matériel

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 définit la structure du robot qui suivra la configuration suivante :

SaMBot

MSP430G2553

MSP430G2231

Bluetooth

Accéléromètre

Capteur IR

Servomoteur

SPI

UART

Distance entre le robot et un objet

Moteur A

+ Odomètre

Moteur B

+ Odomètre

Consigne angulaire

## Exigences systèmes 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 détaillées 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 un 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

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 :
  + 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.

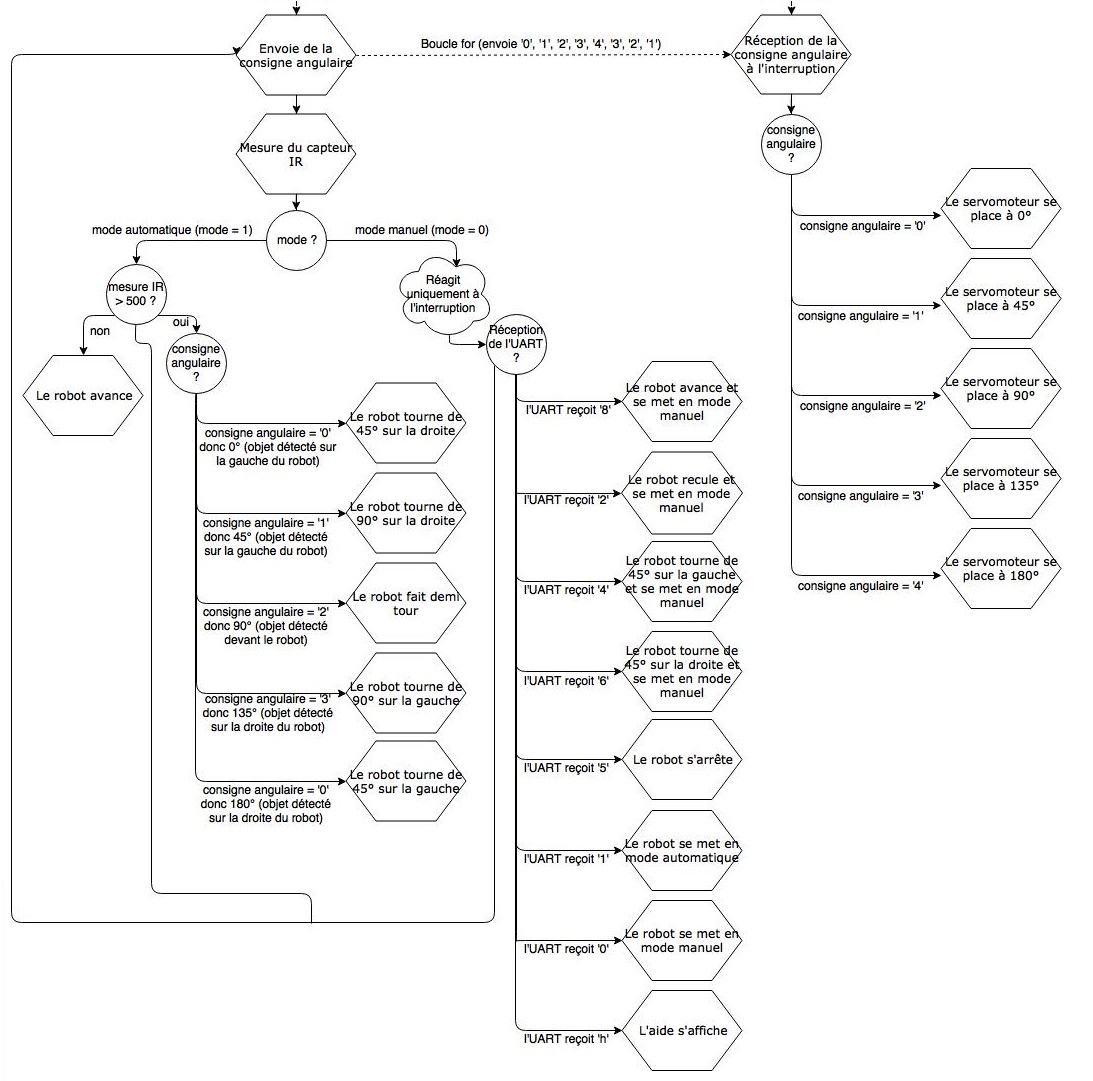
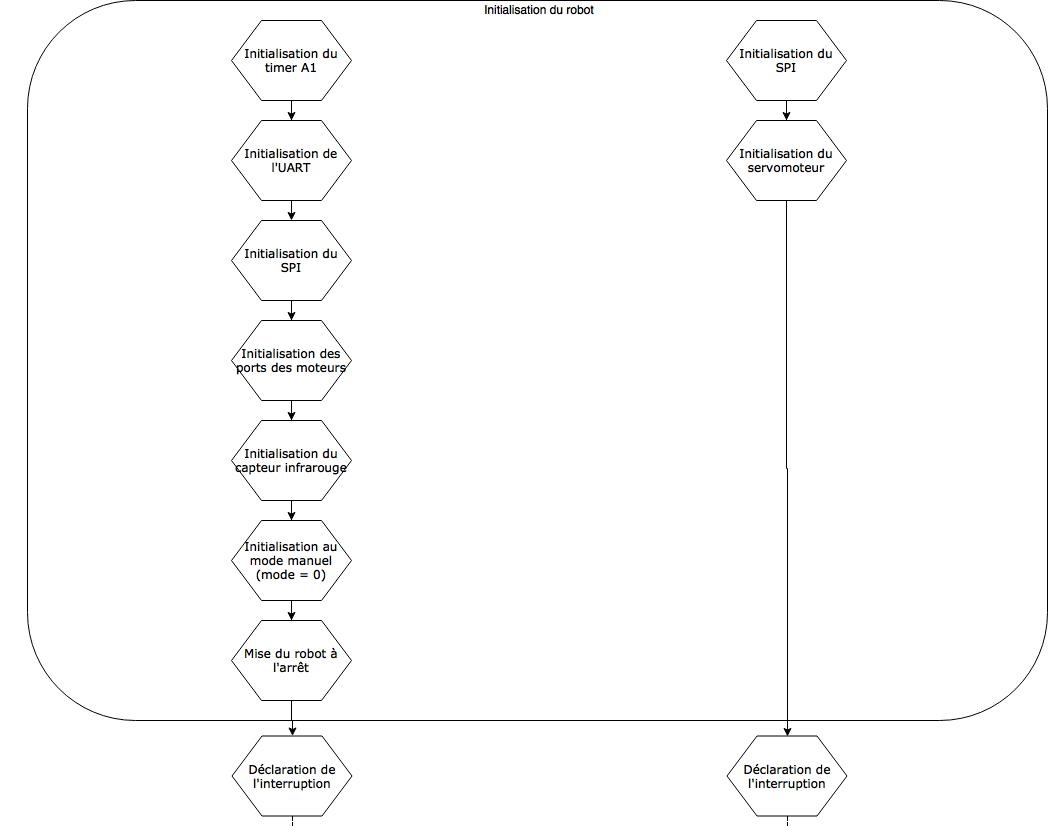
Si l’utilisateur choisit h, l’aide s’affiche sur son application de contrôle.  
Si l’utilisateur choisit 8, le robot avance en ligne droite à vitesse maximale.  
Si l’utilisateur choisit 2, le robot recule en ligne à vitesse maximale.  
Si l’utilisateur choisit 4, le robot tourne à gauche de 45°.  
Si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

### Schéma du 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

0°

45°

90°

135°

180°

Avant du robot

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 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 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 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 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 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 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.

# Tests unitaires (boîtes 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.**

Résultats attendus

Liste des tests à effectuer :

1. init\_timer\_A1() : Les ports du timer sont initialisés et la période vaut 100 µs.

Résultats obtenus

1. init\_timer\_A1() : **validé.**

Fonction : Move\_init()

**Situation : Testée.**

Résultats attendus

Liste des tests à effectuer :

1. move\_init() : Les ports des moteurs sont initialisés.

Résultats obtenus

1. move\_init () : **validé.**

Fonction : move()

**Situation : Testée.**

Résultats 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

Résultats 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.**

Résultats attendus

Liste des tests à effectuer :

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

Résultats obtenus

1. stop() : **validé.**

Fonction : automode()

**Situation : Testée.**

Résultats attendus

Liste des tests à effectuer :

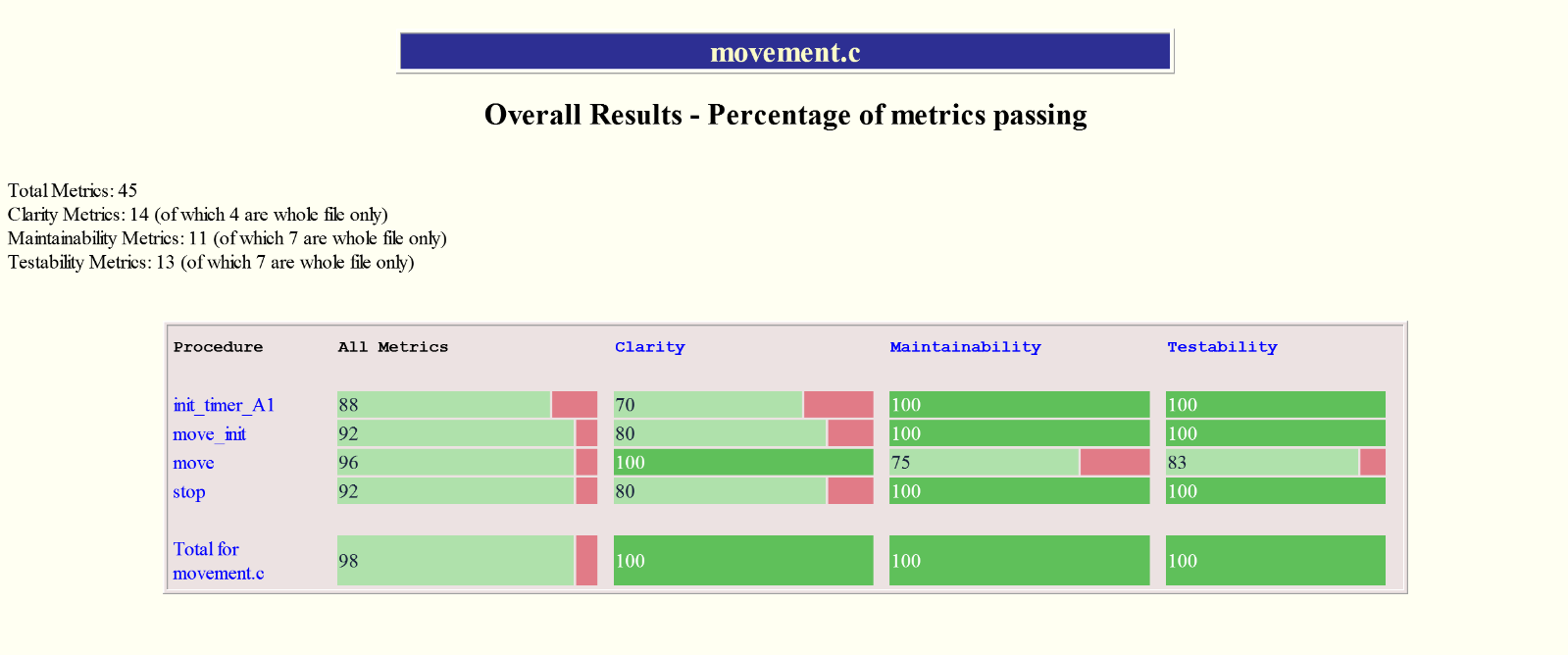
1. automode(150,’0’) : Le servomoteur est à la position 0° et un obstacle se situe à la valeur 150/667.
2. automode(130,’1’) : Le servomoteur est à la position 45° et un obstacle se situe à la valeur 130/667.
3. automode(10,’2’): Le servomoteur est à la position 90° et un obstacle se situe à la valeur 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.

Résultats 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 :

## Test du module « measure.c »

Fonction : measure\_init()

**Situation : Testée.**

Résultats attendus

Liste des tests à effectuer :

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

Résultats obtenus

1. measure\_init() : **Validé.**

Fonction : measure()

**Situation : Testée.**

Résultats 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 🡪∞ : La valeur obtenue en sortie de la fonction vaut 0.

Résultats 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 🡪∞ : **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).**

Résultats attendus

Liste 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

1. Distance <10 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut 660.
2. Distance 40 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut 660.
3. Distance 100 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut environ 320.
4. Distance 150 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut environ 251.
5. Distance 200 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut environ 243.
6. Distance 250 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut environ 125.
7. Distance 300 mm : **NON VALIDE :** La valeur obtenue en sortie de la fonction vaut environ 155.
8. 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 :

avec et . (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 :

## Test du module « UART.c »

Fonction : UART\_init()

**Situation : Testée.**

Résultats 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.

Résultats obtenus

1. UART\_init() : **Validé.**

Fonction : UART\_Tx()

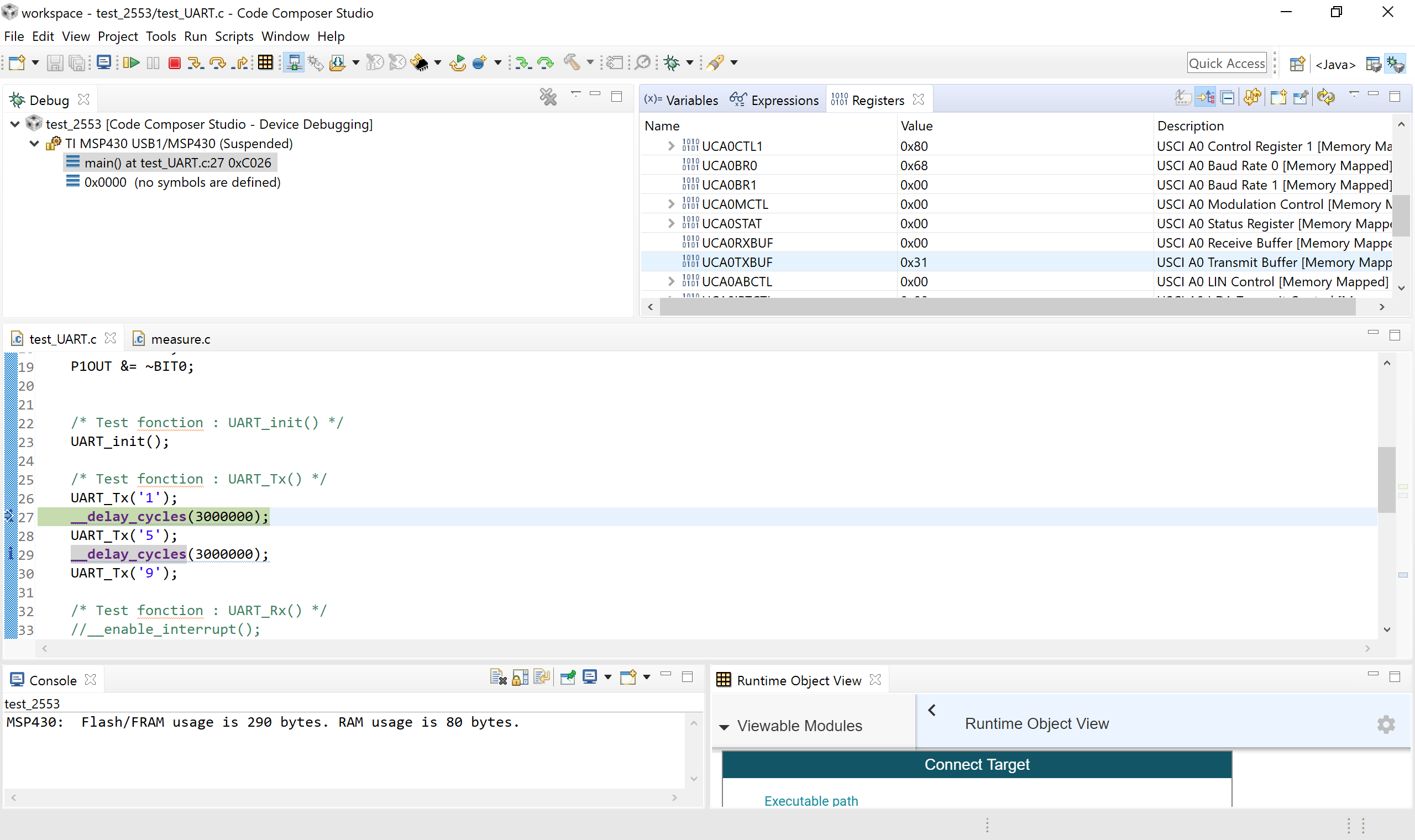
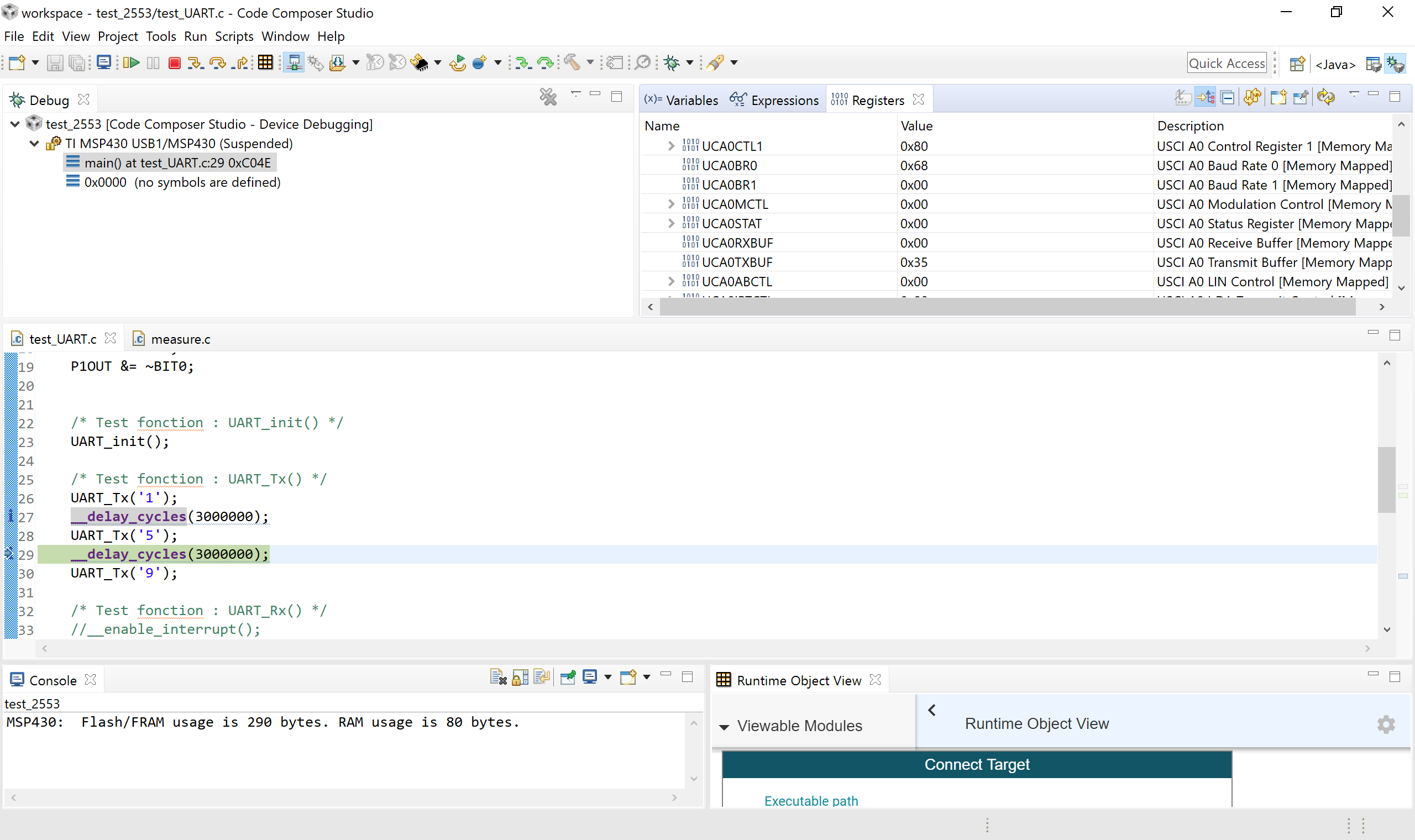
**Situation : Testée.**

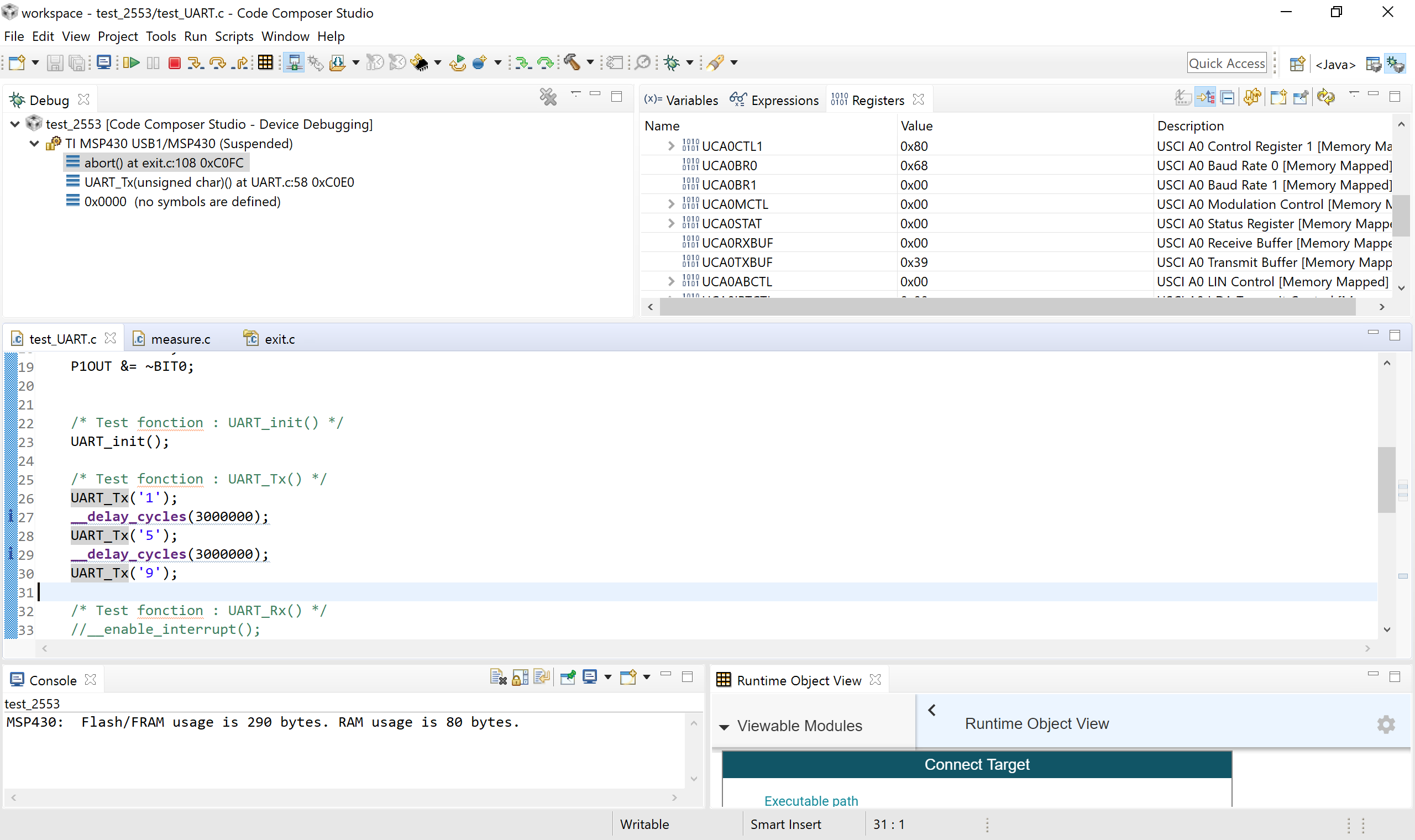
Résultats attendus

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.

Résultats obtenus

1. UART\_Tx(‘1’) : **Validé.**
2. UART\_Tx(‘5’) : **Validé.**



1. UART\_Tx(‘9’) : **Validé.**

Fonction : UART\_Rx()

**Situation :**

Résultats attendus

Liste des tests à effectuer :

1. UART\_Rx() : L’information que le µC 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.

Résultats obtenus

1. UART\_Rx() : **Validé.**
2. UART\_Rx() : **Validé.**
3. UART\_Rx() : **Validé.**

Fonction : envoi\_msg\_UART()

**Situation :**

Résultats attendus

Liste des tests à effectuer :

1. envoi\_msg\_UART(‘Bonjour’) : Le message « Bonjour » doit être transmit et affiché sur l’application.
2. envoi\_msg\_UART(‘Test UART’) : Le message « Test UART» doit être transmit et affiché sur l’application.

Résultats 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.**

Résultats attendus

Liste des tests à effectuer :

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

Résultats obtenus

1. SPIM\_init() : **Validé.**

Fonction : SPIM\_Tx()

**Situation : Fonctionnelle.**

Résultats 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.

Résultats 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.**

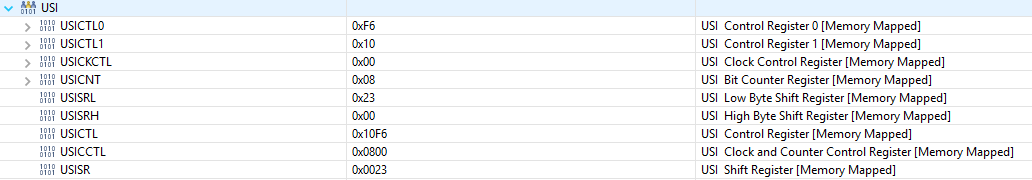
Résultats attendus

Liste des tests à effectuer :

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

Résultats obtenus

1. SPIS\_init() : **Validé.**



Fonction : SPIS\_Rx()

**Situation : Fonctionnelle.**

Résultats 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.

Résultats 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.**

Résultats attendus

Liste des tests à effectuer :

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

Résultats obtenus

1. servomotor\_init() : **Validé.**



P1DIR = 0000 0**1**00

P1SEL = 0000 0**1**00

Fonction : servomotor\_PWM\_init()

**Situation : Fonctionnelle.**

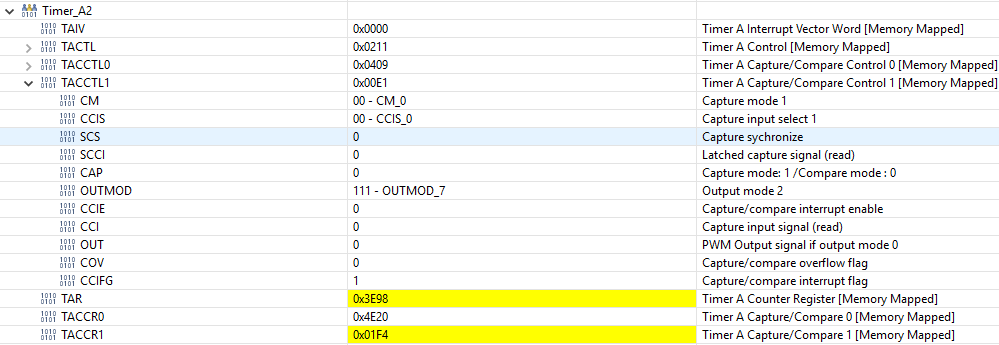
Résultats attendus

Liste des tests à effectuer :

1. servomotor\_PWM\_init() : Les registres nécessaires pour effectuer la PWM sont bien initialisé de manière à avoir une période de 20 ms. De plus, la PWM est réglé pour que le servomoteur se place à la position 0°.

Résultats obtenus

1. servomotor\_PWN\_init() : **Validé.**



Fonction : servomotor\_stop()

**Situation : Non testée.**

Résultats attendus

Liste des tests à effectuer :

1. servomotor\_stop() : le servomoteur s’arrête de tourner instantanément.

Résultats obtenus

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

C:\Users\Rodolphe\AppData\Local\Microsoft\Windows\INetCache\Content.Word\stop.png

Fonction : servomotor\_set\_deg()

**Situation : Fonctionnelle.**

Résultats attendus

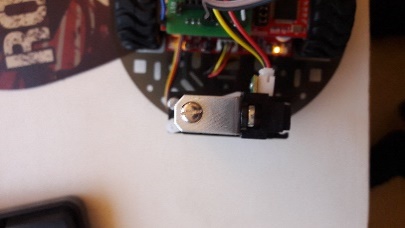
Liste des tests à effectuer :

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.

Résultats obtenus

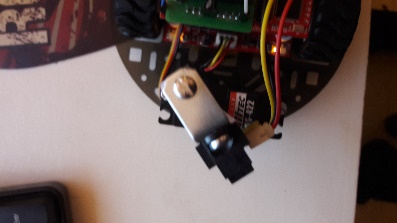
1. servomotor\_set\_deg (0) : **Validé.**

deg0



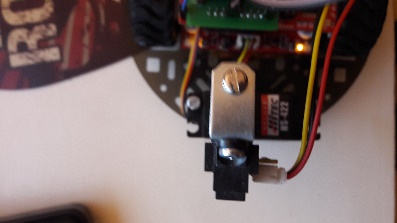
1. servomotor\_set\_deg (45) : **Validé.**

deg45



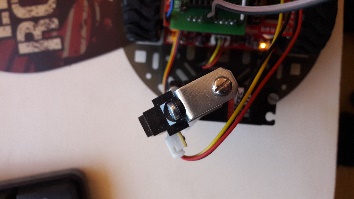
1. servomotor\_set\_deg (90) : **Validé.**

deg90



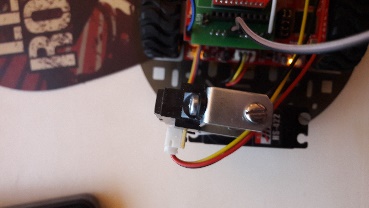
1. servomotor\_set\_deg 135) : **Validé.**

deg135



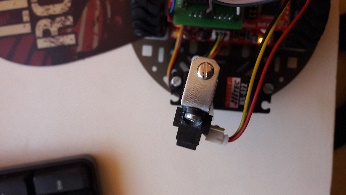
1. servomotor\_set\_deg (180) : **Validé.**

deg180



1. servomotor\_set\_deg(53) : **Validé.**

deg53



**Conclusion :**

Le servomoteur est fonctionnel et prêt à être utilisé.

# Tests d’intégration

## Test de la communication spi

Fonctions : spim\_tx() et SPIS\_rx()

**Situation : Fonctionnelle.**

Résultats 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.

Résultats 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.**

Résultats attendus

Liste des tests à effectuer :

1. Envoie de ‘8’ avec l’application. L’information que le µC 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 µC 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 µC 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 µC doit recevoir se retrouve bien sur le buffer de réception et le robot s’arrêter avec la fonction stop().

Résultats obtenus

1. Après envoie de ‘8’ avec l’application, l’information se trouve dans le buffer de réception et le robot avance. **Validé.**
2. Après envoie de ‘2’ avec l’application, l’information se trouve dans le buffer de réception et le robot recule. **Validé.**
3. Après envoie de ‘4’ avec l’application, l’information se trouve dans le buffer de réception et le robot tourne à gauche. **Validé.**
4. Après envoie 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 poigné 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 avions tenté d’écrire ne fonctionnait pas. Certains que nous avions 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.

*Lien GitHub :* [*https://github.com/hakamir/LD\_Sambot*](https://github.com/hakamir/LD_Sambot)