AMMAR Selim  
CARRÉ Amaury  
FRANÇOIS-CHARLOT Axel  
CLÉMENT Joris  
FALGAYRETTES Julien  
GARCIA Elliot

4 ETI, Equipe D1

Projet Tronc Commun

Robot de manutention de mise en rayon

# Chapitre 1 - Présentation de l’équipe

## Répartition des tâches

L’animateur du projet et responsable Planificateur : Elliot Garcia  
Le responsable – Gestionnaire des rendus : Julien Falgayrettes  
Le responsable Communication : Joris Clément  
Le responsable « Respects des Consignes » : Selim Ammar  
Le responsable Qualité – Tests – Aspects contrôle technique : Axel François  
Le responsable Logistique : Amaury Carré

## Outils mis en place

Communication : Groupe de discussion Messenger  
Compilation et versionning du code : [GitHub](https://github.com/AxFrancois/PTC-Robotic)Suivis de l’état du projet et planning : Petite réunion d’équipe à chaque fin de séance pour informer des tâches accomplies et de ce qu’il reste à faire. Réalisation d’une TODO List lors de l’étude du projet.

# Chapitre 2 – Présentation du projet

## Contexte et définition du problème

Dans le cadre du programme de l’année 4 ETI, nous devons réaliser un robot avec un cahier des charges ouvert ayant pour but de démontrer les capacités d’innovation et de mise en œuvre de l’équipe. Suite à la première séance de brainstorming, nous avons convenu d’une liste d'idées de projet, pour finalement sélectionner celle du robot de manutention.

## Objectifs du projet

L’objectif de ce projet sera de concevoir un robot de transport de marchandises dans des rayons de supermarchés. Le robot devra pouvoir transporter des articles et les emmener à leur emplacement sans aide extérieure.

## Périmètre du projet

Le matériel sera fourni par CPE concernant le châssis du robot et les cartes contrôleur. En cas de besoin, il sera possible de commander des pièces via l’équipe encadrante.

## Description fonctionnelle des besoins

* Déplacement

Le robot doit se déplacer le long d’un circuit pré établi suivant les rayons. Il doit être capable de se diriger seul d’un point A à un point B et de réagir en cas d’obstacle.

* Transport

Le robot doit pouvoir transporter des articles **entre X et X kilogramme** de **type Y**. Aussi, le robot devra être capable de prendre un article posé dans la pièce de stockage, se déplacer sans le perdre et enfin le déposer à son emplacement prévu.

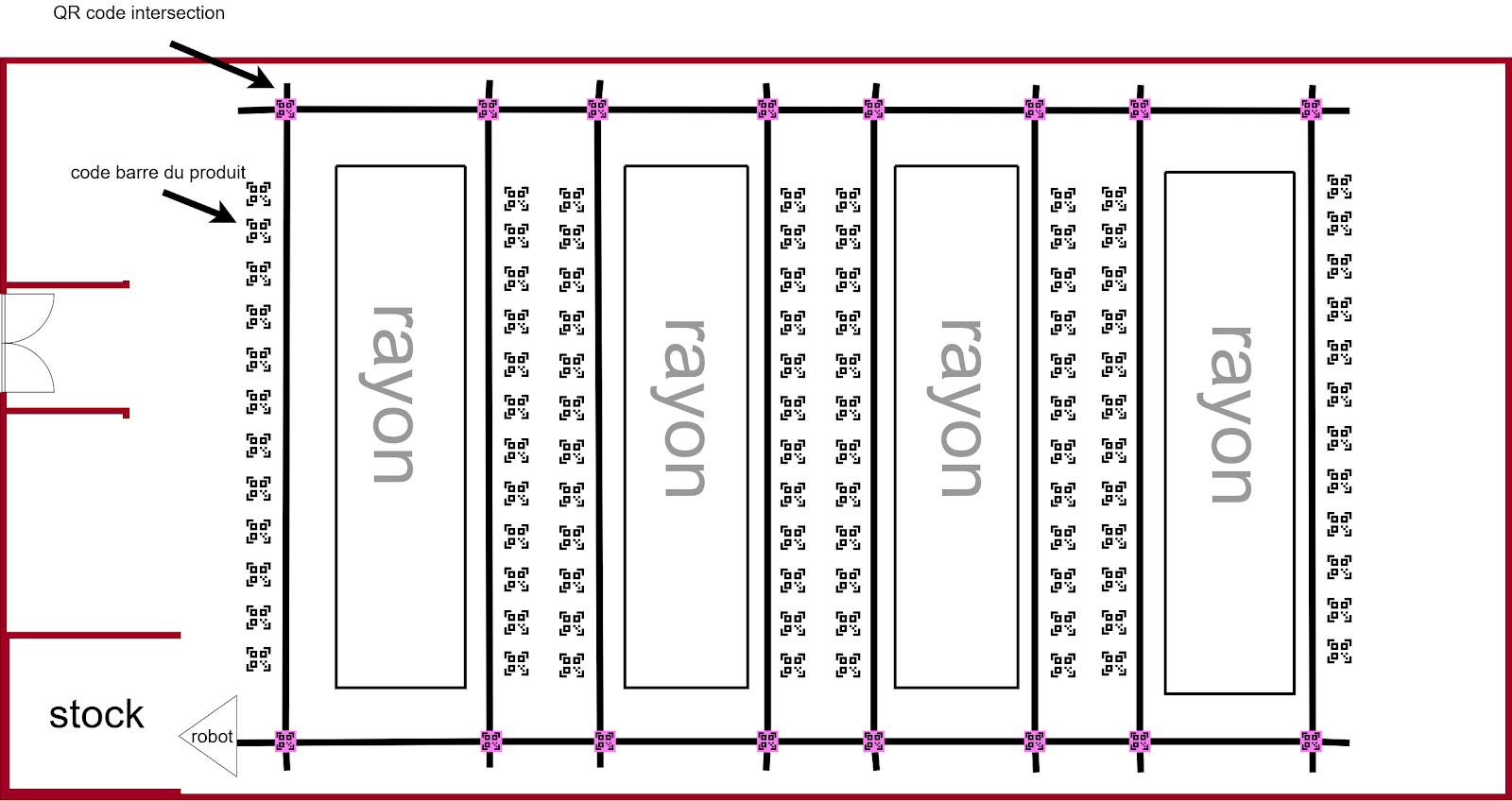
* Administration

Le robot doit pouvoir savoir où déposer un article et ne pas constituer un danger pour les personnes autour et lui-même. Il devra pouvoir recevoir un trajet à prendre, le comprendre et le suivre. De plus, il devra transférer des informations sur sa consommation et s’arrêter de lui-même en cas d’anomalie.

* Connectivité

Nous souhaitons implémenter la possibilité de suivre la position du robot en temps réel, transmettre les informations enregistrées par le robot à l’utilisateur et de le gérer à distance.

## Représentation graphique du fonctionnement



## Répartition des tâches techniques jusqu’au Jalon1

* Nom de Etudiant 1 : Falgayrettes Julien

Responsabilité : ***Contrôle moteur des roues avec stm32+ contrôle servomoteur + Communication entre stm32 et raspberry pi***

* Nom de Etudiant 2 : Amaury Carré

Responsabilité : ***QR code reader, suiveur de ligne, UART raspi et stm32***

* Nom de Etudiant 3 : Elliot Garcia

Responsabilité : ***IHM, base de données***

* Nom de Etudiant 4 : Joris Clément

Responsabilité : ***Schématisation et rédaction des CR de fin de séance, télémètres à ultrasons, communication stm32 et Serializer***

* Nom de Etudiant 5 : Axel François

Responsabilité : ***API IHM-Robot, Communications IHM-NodeJS-Robot, Conteneurisation***

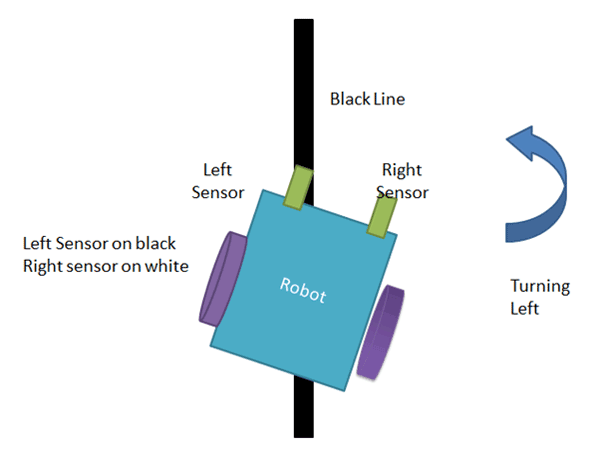
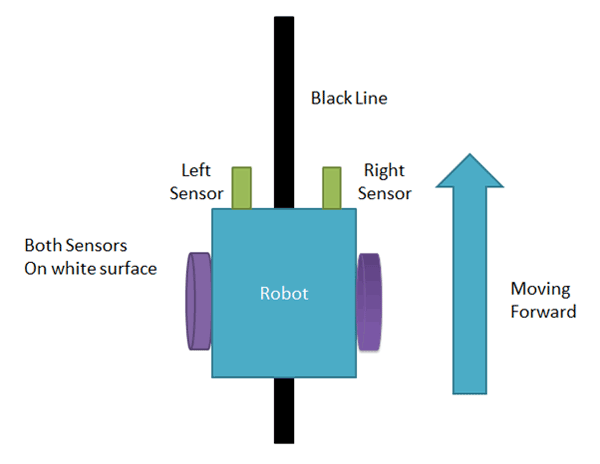
* Nom de Etudiant 6 : Selim Ammar

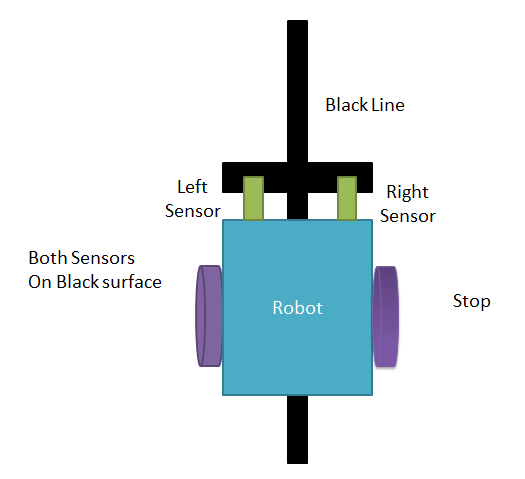
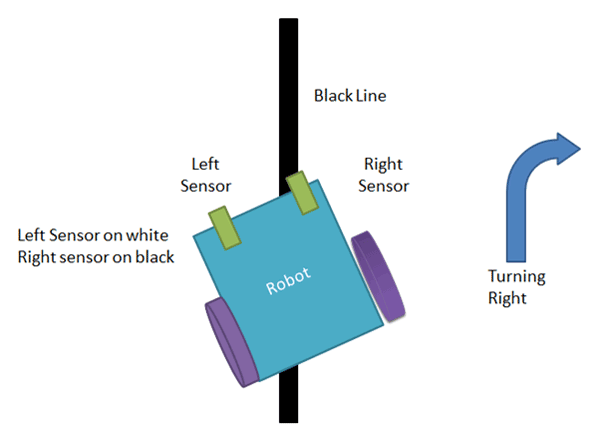
Responsabilité : ***QR code reader***

:

# Chapitre 3 : Raspberry pi3B

## 3.1 : Détection de bandes au sol et suivi de ligne





Codes et Algorithmie :

* Cas 1) Les deux détecteurs infrarouges détectent le sol (couleur claire) et retourne un 1, alors le robot avance tout droit.
* Cas 2) Un des deux détecteurs capte du noir à droite (couleur de la bande), alors le robot tourne à droite : moteur droit à l’arrêt et moteur gauche tourne.
* Cas 3) Un des deux détecteurs capte du noir à gauche (couleur de la bande), alors le robot tourne à gauche : moteur droit tourne et moteur gauche à l’arrêt.
* Cas 4) Les deux détecteurs captent du noir et retournent un 0. Le robot s'arrête.

Fonctions complémentaires :

* En cas d’arrêt lié à la détection d’une intersection, lecture de QR code.
  + Si QR code bon, rotation
  + Sinon, le robot continue tout droit
  + Si aucun QR code ou code-barre n’est détecté, alors le robot reste sur place.
* Si détection d’obstacle, contournement
* Communication UART entre le Raspberry PI et le STM32 qui contrôle le robot.

## 3.2 : Communication UART entre Raspi et STM32

Pour pouvoir utiliser une liaison UART sur le raspi 3 il y a un problème car sur cette version le véritable UART est utilisé par le Bluetooth. Pour pouvoir le rediriger vers notre fonctionnalité, il faut désactiver le Bluetooth sur la console et réactiver l’UART0. Une fois cela réalisé, il n’y a plus de grandes difficultés car du côté STM32 il suffit de s’assurer que l’UART soit bien configuré (même baud rate…) et que les liaisons filaires soient correctes.

## 3.3 : Lecture de QR codes et codes-barres

Lors de nos recherches la bibliothèque pyzbar sur python, capable de lire les différents codes, nous a fait choisir d’utiliser le Raspberry pi pour la lecture de QR code. Idée qui s’est renforcée lorsque nous avons reçu une webcam compatible Raspi.

Sur le Raspi, notre programme utilise pyzbar et cv2 (pour la webcam). Ce programme sera appelé à chaque intersection, pour retrouver le chemin à prendre.

# Chapitre 4 : Détection des obstacles télémètre à ultrasons

# Insertion de l’image...

Voici notre module télémètre à ultrason, le HC-SR04, qui va nous permettre de détecter les objets. Son fonctionnement est assez simple, il possède 4 ports Trig, Echo, Vcc et Gnd.

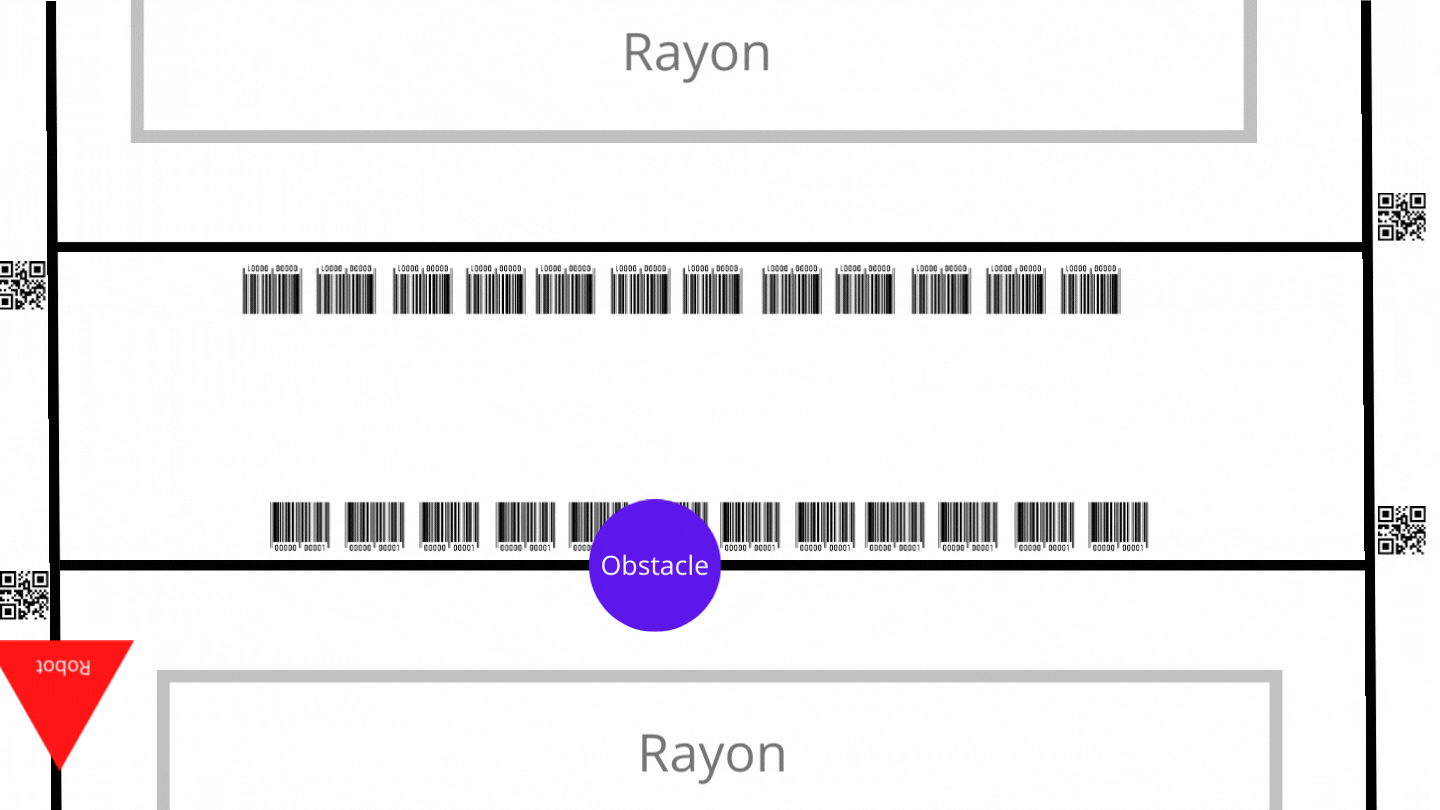
* Vcc et Gnd sert à l'alimentation.
* Trig sert à déclencher une mesure lorsque nous envoyons un signal sur cette broche.
* Echo sert à réceptionner l’onde émise.

Nous ouvons donc calculer le temps que sépare l’émission et la réception de l’onde et donc déterminer la distance d=c\*t   
Avec d : la distance, c : la vitesse du son dans l’air et t : le temps écoulé entre l'émission et la réception

Nous pourrons donc créer une fonction afin que le robot s'arrête s'il détecte un objet à partir d’une certaine distance et qui contourne l’obstacle après un certain temps.

L’algorithme développé sera le suivant :

1. Si le robot capte un obstacle à moins de 100 cm devant lui, il s’immobilise et émet un son. Dans le cas où l’obstacle est un humain, il sera averti.
2. Si après quelques secondes l’obstacle est toujours présent, le robot oriente son télémètre ultrason à droite puis à gauche afin de déterminer la direction dans laquelle il va contourner l’obstacle (dans le cas pratique il y aura un rayon d’un côté, il choisira donc l’autre côté). Puis avance de 60 cm (choix arbitraire à adapter à la largeur du rayon).
3. Le robot effectue une rotation de 90° dans son sens initial de circulation. Puis vérifie s'il n’y a pas d’obstacle devant lui, si oui, il avance tant qu’il n’a pas dépassé l’obstacle ou rencontré une ligne noire. Sinon il s’immobilise et s'arrête.
4. Le robot effectue une nouvelle rotation à 90° dans la direction de la ligne noire et avance jusqu'à l'avoir rejoint.
5. Fin de l’algorithme de déplacement, reprise du fonctionnement normal.

Animation du fonctionnement :  


# Chapitre 5 : Déplacement du robot

Pour contrôler les moteurs des roues du robot, nous utilisons la Serializer. Cela va permettre de contrôler les moteurs grâce à des chaînes de caractères transmis sur le port Rx de la carte en fonction du mouvement que nous souhaitons.

Nous utiliserons principalement 3 commandes qui sont :

* mogo motorId:vel [motorId:vel]
* digo id:distance:vel [id:distance:vel]
* stop

Pour faire fonctionner les moteurs, il faut envoyer ces chaînes de caractères sur la liaison UART de la Serializer et principalement sur le Rx. Pour cela, nous utilisons la STM32 pour transmettre les données.

Nous définissons donc la liaison UART4 de la STM32 avec un baudrate de 19200. En ajoutant une résistance Pull Up sur le signal transmis par la carte, la Serializer va donc bien comprendre ce message. Grâce à cela nous avons donc pu contrôler le robot en fonction des données que nous recevons sur le Rx de la Discovery et en fonction de ce qu’elle reçoit, elle va transmettre un ordre de déplacement pour les moteurs.

De plus, pour communiquer avec le Raspberry PI 3 B nous définissons une autre liaison UART5 avec le même baudrate que celle-ci, nous sommes donc maintenant capables de faire réagir les roues du robot en fonction que ce que renvoie le suiveur de ligne du Raspberry ou la caméra qui capte les QR code. Nous n’avons pas développé le code pour ça, mais la liaison entre les deux cartes fonctionne.

Et pour contrôler avec le module XBee que l’IHM utilise, nous allons utiliser l’UART4 car le baudrate est identique avec la Serializer.

# Chapitre 6 : IHM

Nous avons choisi de développer une IHM pour notre robot. En effet, cet outil est indispensable pour la distribution du produit afin de le rendre accessible au plus néophyte (et cliquer sur un bouton pour envoyer une instruction est bien plus simple que de devoir taper la commande entière). L’IHM nous servira aussi à effectuer les tests pour le développement du robot, surtout sa partie déplacement autonome.

Il est d’ailleurs à noter que dans le contexte de la vente d’un produit à une entreprise tierce, le choix d’un IHM web est intéressant car ayant une bonne maintenabilité dans le temps à condition de la développer avec des outils structurels compétitifs comme cité ci-après.

Nous avons donc développé une IHM en PHP grâce à Apache. Nous utilisons Smarty pour réaliser des templates de page web, et le site est doublé d’une base de données PostgreSQL qui servira à stocker les commandes de l’API et les informations pour établir le trajet du robot pour le déplacement autonome. Le choix du PostgreSQL est dû au fait qu’il utilise les requêtes communes du SQL et est à licence libre. Ainsi le PostgreSQL peut s’adapter à un très grand nombre de structure de données et représente un choix économique plus qu’intéressant.

# Chapitre 7 : Protocole de communication

Pour accompagner la mise en place d‘IHM, il est nécessaire de définir une API permettant de communiquer d’un point de vue logicielle entre l’IHM et la STM32 embarquée. Pour cela, nous avons défini des mots clés permettant de transmettre des informations au robot depuis l’IHM, et vice-versa. La liste actuelle des commandes prévues est renseignée sur le GitHub du projet, à [ce lien](https://github.com/AxFrancois/PTC-Robotic/blob/main/XBEE/Documentation%20commandes.md). Cette liste est bien sûr exhaustive, et susceptible à des évolutions au cours du développement.

Le rôle de cette API est double. Elle a pour but de nous permettre de tester simplement les fonctionnalités du robot individuellement grâce à la console de l’IHM, et de permettre la remontée d’information pour les utilisateurs.

Nous avons également réalisé un petit programme en python permettant de générer automatiquement le squelette du code pour le traitement, à la fois en C pour la STM32 et en JS pour l’IHM. Cela nous permettra de faire évoluer notre API sans risquer de casser nos programmes précédemment réalisés, ce qui est à la fois un gain de temps et d'efficacité.

# Chapitre 8 : Interface IHM - Port série

L’IHM, réalisé sous forme de serveur Apache (PHP), utilise les langages classiques du développement web, à savoir l’HTML, le CSS, le JavaScript et le PHP. Néanmoins, un problème majeur se pose : aucun de ces langages n’est capable de s’interfacer avec un port série directement. Il nous a donc fallu utiliser un autre programme pour faire le pont entre ces deux composantes. Plusieurs choix s’offraient à nous, et nous avons finalement choisi d’utiliser NodeJS, une plateforme logicielle en JavaScript, car ce langage proposait une solution simple et performante pour la communication avec le port série.

Le site web et le NodeJS utilisent bien tous les deux le JavaScript comme langage de programmation, mais les deux environnements d’exécutions sont radicalement différents et incompatibles. Il nous a donc fallu mettre en place une communication entre l’IHM (front-end) et le NodeJS (back-end). Pour cela nous avons utilisé Websocket, une API très populaire et simple d’implémentation.

Une fois toutes ces étapes réalisées, nous pouvons enfin envoyer des informations au robot, et lui aussi !

# Chapitre 9 : Conteneurisation de la solution

La réalisation de la partie IHM et NodeJS s’est effectuée sur les ordinateurs de 2 personnes différentes, et lors de la mise en commun nous avons rencontré plusieurs problèmes bloquants, mais ces derniers n'était pas constant d’un ordinateur à l’autre, ce qui les rendait d’autant plus compliqué à diagnostiquer et corriger ! Nous avons donc conjointement décidé de modifier notre solution pour l’intégrer dans des conteneurs.

La technologie conteneurs, dont l’exemple le plus connu et celui que nous avons utilisé est Docker, permet de créer un environnement virtuel directement connecté avec le noyau de l’ordinateur dont tous les paramètres sont déterminés. Celà permet de pouvoir travailler dans le même environnement quelle que soit la machine sans risquer des problèmes d’incompatibilité et de librairies manquantes !

Une fois ce système mis en place, ce qui à malheureusement pris pas mal de temps car cette technologie est nouvelle pour nous et apprise sur le tas, nous pouvions lancer toutes nos applications d’un seul clic !

Un hic nous a cependant bien embêtés ensuite, c’est que nous avons eu des problèmes pour configurer les volumes des conteneurs. Les volumes sont des fichiers spéciaux qui sont modifiables à la fois par le conteneur et le système d'exploitation, ce qui permet de faire du développement directement dans le conteneur. Après beaucoup d’essais infructueux nous avons finalement réussi à mettre ce système en place.

## Chapitre 10 : Versioning et automatisation des tests

Nous avons mis en place un repository GitHub pour travailler efficacement sur la partie logicielle du projet. Cet outil permet de versionner notre développement et de revenir à une version précédente en cas de problème.

Nous avons également mis en place des tests automatiques qui vérifient la bonne santé de notre code, notamment grâce à GitHub Action qui vérifie que nos conteneurs réussissent à se lancer avec succès, et avec Sonarcloud qui vérifie que notre code n’a pas de bug et est correctement réalisé.

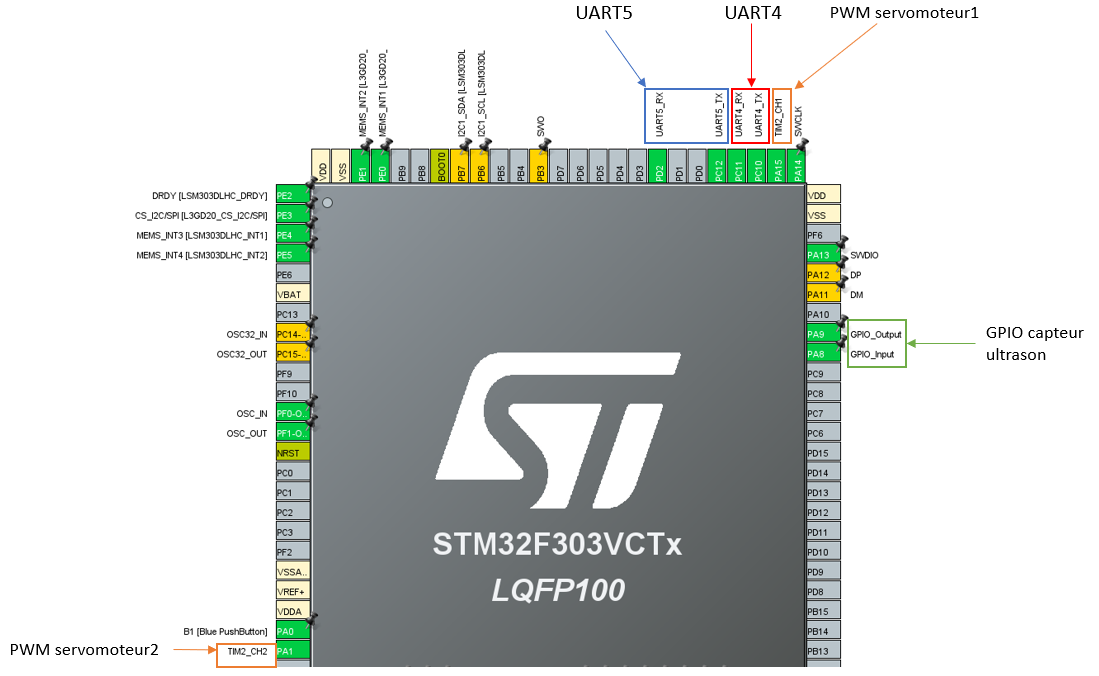
# Chapitre 11 : Contrôle servomoteur

Nous utilisons les servomoteurs pour permettre aux capteurs à ultrasons de capter les obstacles sur 180°, cela va nous être utile lors d’un contournement d’obstacle pour choisir le côté où le robot doit aller pour esquiver convenablement l’obstacle.

Pour cela nous définissons un Timer2 qui utilise la Clock interne du microcontrôleur qui est à 48MHz dans ma Clock Configuration, ensuite nous choisissons un prescaler de 48 pour obtenir une fréquence de 1MHz et nous définissons ensuite une « Counter period » égale à 20 000 pour ensuite obtenir une Clock de fréquence 50Hz. Ensuite grâce à l’IDE nous contrôlons le Duty cycle de mon Timer pour contrôler l’angle du servomoteur en fonction du temps à l’état haut. Pour ce servomoteur un temps à l’état haut de 0.5 ms correspond à 0°, un temps de 1.5 ms à l’état haut correspond à 90 ° et donc un temps correspondant à 2,5 ° correspond à un angle de 180°. Nous pouvons donc diriger de manière précise le servomoteur.

# Chapitre 12 : Configuration de la STM32

Nous avons donc configuré les ports de la STM32 pour les deux liaisons UART, Le PWM pour contrôler les servomoteurs et les broches pour le capteur de distance qui correspond à ECHO et TRIG.



Chapitre 13 : Intégration des sous-ensembles

Méthode d’intégration des sous-ensembles :

Une fois les différentes partie réalisé, la communication se fera principalement via des liaisons UART qui vont venir transmettre des messages à la STM32 qui va pouvoir en fonction du message reçu mettre en marche ces actionneurs.

Nous allons connecter les ports de l’UART de la raspberry pi sur un port receveur de la STM32 pour recevoir les données envoyées de la raspberry pi, nous faisons de même pour l’IHM nous branchons le module radio Xbee sur un port Receiver de la STM32 pour recevoir les données.

Grâce à la fonction de CallBackRX nous allons pouvoir activer nos actionneurs dès qu’un message est reçu pour réagir directement, cette méthode est très bien car le programme tourne en continue et c’est seulement lors d’une réception de donnée que le programme rentre dans la fonction de CallBack pour réagir.

Test des sous-ensembles :

Nous avons déjà testé l’envoie de données entre la raspberry pi et la stm32 et cela fonctionne, nous allons donc tester les ensembles un par un et ensuite nous allons faire le branchement final pour que tout fonctionne correctement ensemble.

Pour cela nous décidons d’actionner les roues du robot dès qu’un message est reçu pour bien vérifier le fonctionnement de l’IHM et de la raspberry pi connecté à la STM32.

Pour le télémètre à ultrasons nous mettons en mouvement les roues du robot et à partir d’une certaine distance nous envoyons un message d’arrêt si celui-ci arrête bien les roues dès que la distance dépasse une limite la partie capteur fonctionne.