HEH.be
Sciences
et technologies

Avenue V. Maistriau 8a
B-7000 Mons
+32 (0)65 33 81 54
scitech-mons@heh.be

WWW.HEH.BE

Gestion de projet

Réalisation d'une voiture avec Raspberry PI

Année Académique 2020-2021
Groupe N°8

Mathieu Bawin
Antoine Dhainaut
Damien Ghys
Quentin Mettens
Clément Raulier

WALLONIE-BRUXELLES ENSEIGNEMENT
pôle hainuyer

HEH.be Sciences et technologies

Avenue V. Maistriau 8a
B-7000 Mons
+32 (0)65 33 81 54
scitech-mons@heh.be

WWW.HEH.BE

Gestion de projet

Réalisation d'une voiture avec Raspberry PI

Année Académique 2020-2021
Groupe N°8

Mathieu Bawin
Antoine Dhainaut
Damien Ghys
Quentin Mettens
Clément Raulier

WALLONIE-BRUXELLES ENSEIGNEMENT
pôle hainuyer

Table des matières

1. Introduction

Tesla Autopilot. Voici le nom d'une technologie parmi les plus avancées en termes de conduite autonome du monde actuel. Mais comme pour tout projet d'une telle envergure, il a bien fallu commencer quelque part.
En avril dernier, nous avons été confrontés à cette réalité lors d'une semaine entière de projet. Son objectif : rendre autonome une voiture miniature, construite autour d'un célèbre micro-ordinateur, dans le but de la faire participer à une course sur circuit.

Afin de comprendre le résultat de cette semaine de projet intense, il paraît important de commencer la rédaction de ce rapport en présentant et expliquant le fonctionnement des différents éléments de notre voiture miniature. Ensuite, nous présenterons la méthodologie qui nous a permis de gérer ce projet : Scrum.
Ainsi, il sera possible de comprendre l'enchaînement des différents événements, les problèmes qui se sont posés face à nous, mais aussi de proposer des pistes pour de potentielles solutions qui permettraient d'améliorer notre prototype de véhicule autonome.

2. Présentation détaillée des composants

2.1. Raspberry PI

Avant de commencer à présenter les différents composants que nous allons utiliser, nous devons d'abord présenter le micro-ordinateur Raspberry PI.

Lors de cette semaine de projet, nous utiliserons le Raspberry 3B+ qui appartient à la quatrième génération de Raspberry. Celui-ci a été équipé d'un processeur quadricœur puissant de 4x1,4GHz et prend désormais en charge le Bluetooth 4.2.

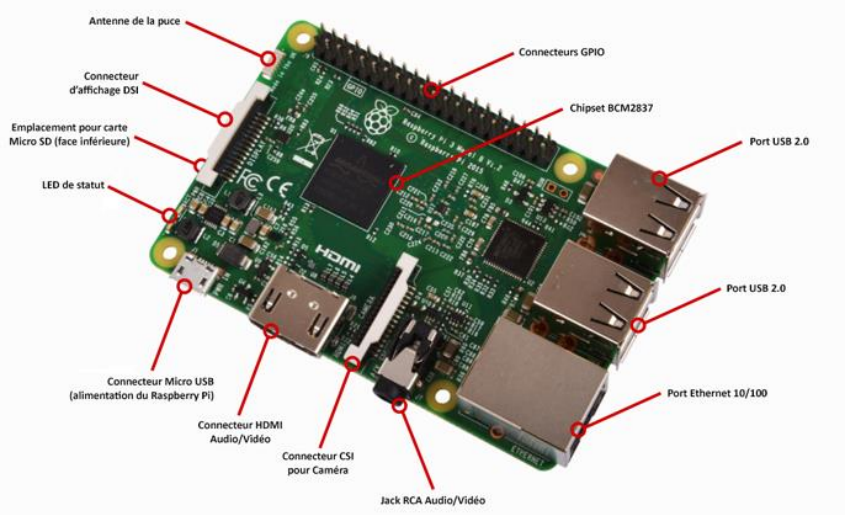


Figure 1: Schéma du Raspberry PI

Lorsque nous le recevons pour la première fois, celui-ci est complètement nu et ne possède pas de système d'exploitation permettant d'interagir avec les différents composants qui peuvent être connectés sur les broches GPIO.

Pour cela, nous devons installer le système d'exploitation Raspbian qui est une distribution optimisée pour le Raspberry (basée sur Debian).

2.2. Les capteurs ultrason

Le capteur ultrason ou sonar est une technologie utilisée depuis déjà plusieurs dizaines d'années. Il est notamment utilisé pour la cartographie des navires en mer ou tout simplement pour la détection des pièces cassées, problèmes mécaniques, etc.

Lors de ce projet, nous ne l'utiliserons pas pour détecter des navires mais pour repérer et avertir notre robot (voiture) des différents obstacles qui se trouvent devant lui ou sur ses côtés.

Figure 2: Capteur ultrason

2.2.1. Branchement

Voici un schéma permettant de démontrer le branchement d'un capteur HC-RS04 :

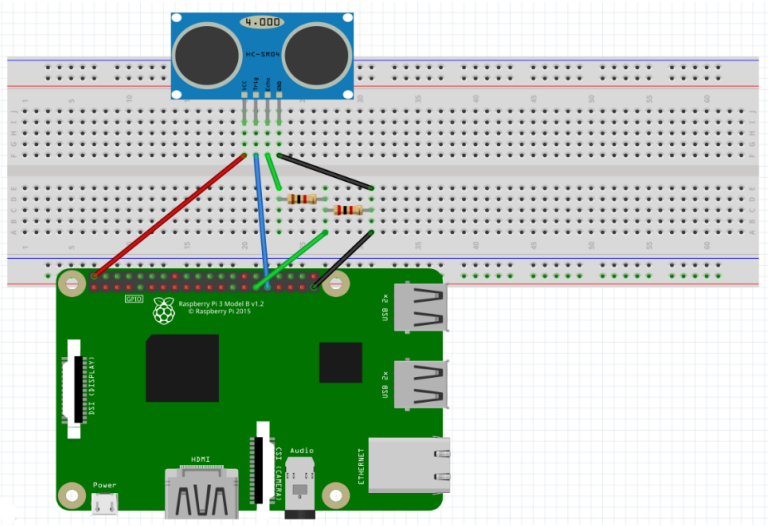


Figure 3: Branchement du HC-SR04

Pour le branchement des différentes broches du capteur ultrason :

- Une broche VCC (GPIO40) : +5V
- Une broche Trig (GPIO6) : Entrée
- Une broche Echo (GPIO5) : Sortie
- Une broche GND

Pour éviter tout éventuel problème, deux résistances (1k et 2k) ont été rajoutées permettant ainsi de réduire la tension du signal Echo de 5V à 3,3V.

2.2.2. Fonctionnement

Pour déclencher la mesure d'une onde à ultrason (2 à 400cm), il faut émettre une tension de 5V (HIGH) d'au minimum 10 µs sur la broche d'entrée Trig. Le composant va donc émettre huit impulsions de 40Khz. Ces impulsions se propagent dans l'air à la vitesse du son et lorsqu'elles rencontreront un obstacle, elles se réfléchiront et reviendront sous forme d'écho au capteur. Lorsque le capteur l'aura détecté, il enverra un signal HIGH sur la sortie Echo.

Pour calculer la distance parcourue, il faudra récupérer le temps écoulé entre l'émission de l'onde et la réception de l'écho que nous multiplierons ensuite par la vitesse du son qui équivaut à 340 m/s. Ensuite, nous devons diviser en deux cette valeur puisque le son fait un aller-retour.

L'entrée Trig ainsi que la sortie Echo sont toutes les deux des petites cellules piézo-électriques qui vont soit vibrer lorsqu'une tension est appliquée (émetteur), soit produire une tension lorsqu'un écho est reçu (récepteur).

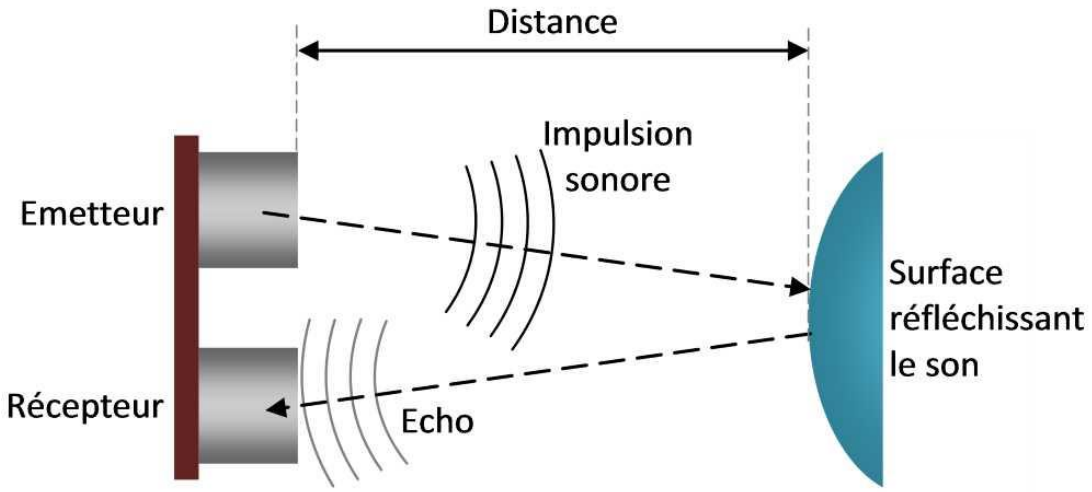


Figure 4: Fonctionnement du HC-SR04

2.3. Le capteur de ligne

Le capteur infra-rouge qui sera présent en dessous du capteur ultrason de devant permettra au robot de compter le nombre de fois où celui-ci passera la ligne d'arrivée.

Lors du dernier tour du circuit, la voiture devra automatiquement s'arrêter lorsqu'elle détectera la ligne d'arrivée.

Figure 5: Capteur de ligne

2.3.1. Branchement

Voici un schéma permettant de démontrer le branchement d'un capteur de ligne :

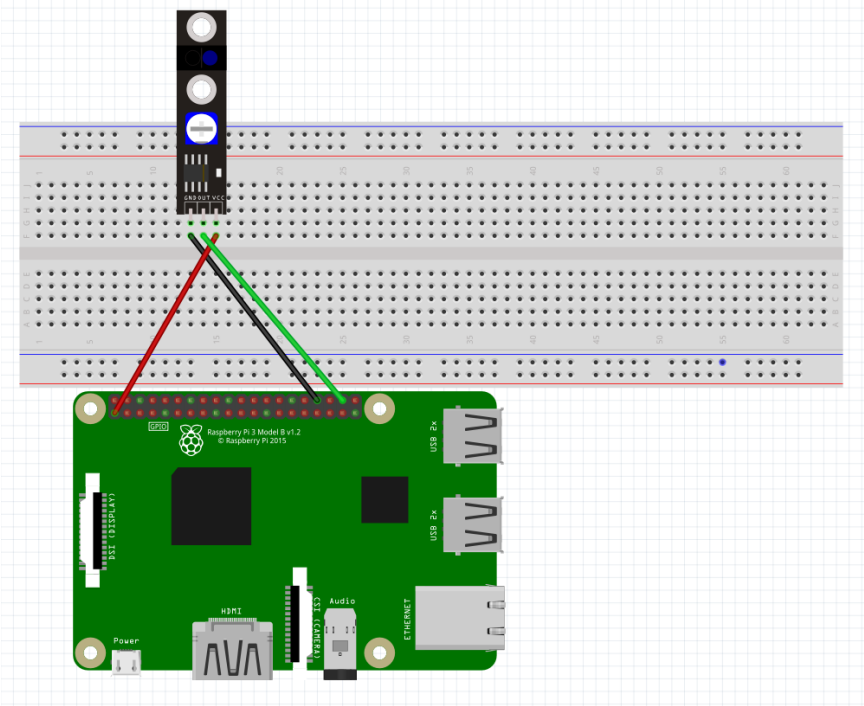


Figure 6: Branchement du capteur de ligne

Pour le branchement des différentes broches du capteur :

- Une broche VCC (GPIO1) : +3.3V
- Une broche VOUT (GPIO20) : Sortie numérique
- Une broche GND

2.3.2. Fonctionnement

Le capteur d'intensité lumineuse ou capteur de ligne permettra de savoir le taux de lumière qui est reflété en dessous du capteur.

La diode LED (émetteur) qui se trouve sur le capteur va émettre de la lumière, celle-ci va ensuite être reflétée en fonction de la surface qui se trouve en dessous du composant. En fonction du taux de lumière que le composant recevra, le capteur saura sur quel type de surface il se trouve.

Si l'émetteur émettait de la lumière sur une surface noire ou très foncée, il n'y aurait peu voire quasiment pas de lumière qui sera récupérée par le récepteur qui se trouve sur le capteur. À l'inverse, si cette surface est de couleur blanche, le taux de lumière que le récepteur recevra, sera beaucoup plus élevé.

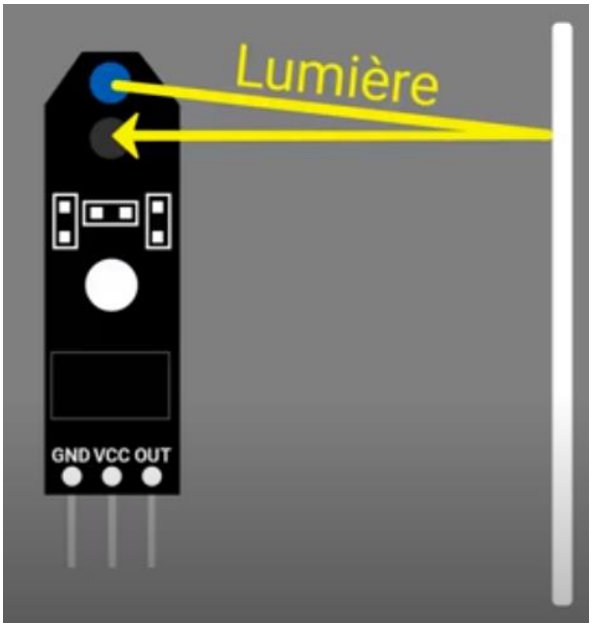


Figure 7 : Fonctionnement du capteur de ligne

2.4. Le servomoteur

Un servomoteur est un mécanisme de petite taille et très économique, grâce à son faible encombrement et à son faible poids (+/- 9g), ce sont des composants utilisés dans de nombreux domaines notamment en électronique.

Contrairement au moteur pas à pas, qui est souvent utilisé pour faire un mouvement continue, les servomoteurs ne savent pas faire plus d'un tour. Ils sont donc utilisés pour réaliser des plus petits mouvements de grande précision.

En d'autres termes, notre servomoteur permettra que notre robot puisse tourner dans un sens ou dans l'autre.

Figure 8: Servomoteur

2.4.1. Branchement

Voici un schéma permettant de démontrer le branchement d'un servomoteur :

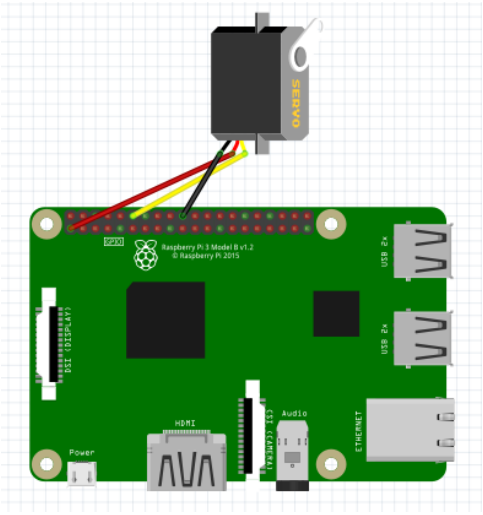


Figure 9 : Branchement du servomoteur

Pour le branchement des différentes broches du servomoteur :

- Une broche VCC (GPIO1) : +3.3V
- Une broche permettant d'utiliser le PWM(GPIO18)
- Une broche GND

2.4.2. Fonctionnement

Pour fonctionner, le servomoteur a besoin du PWM (Pulse Width Modulation), cette technologie permettra de faire varier l'intensité de la puissance du servomoteur.
Son principe consiste à alterner rapidement deux états et en manipulant leur durée, nous pouvons obtenir un état intermédiaire.

Les servomoteurs sont dirigés par un signal de 50Hz (période de +/- 20ms), ce qui veut dire que la durée d'impulsion est comprise entre 1ms et 2ms. Normalement, les servos ne peuvent tourner que de 0 à 180 degrés. Nous pouvons donc ajuster par exemple cette durée d'impulsion pour que le servomoteur donne un angle voulu assez précis.

Par exemple, si nous décidons d'émettre une durée d'impulsion de 1,5ms le servomoteur donnera donc un angle de 90 degrés.

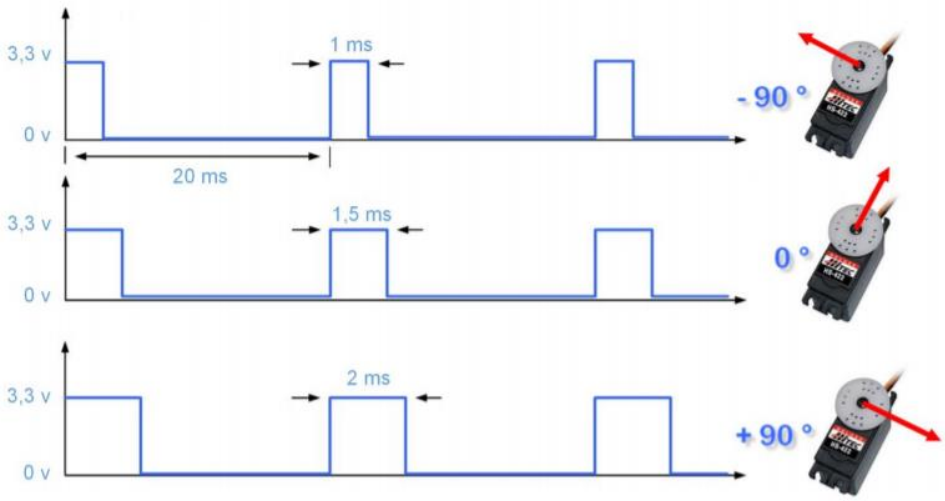


Figure 10 : Fonctionnement du servomoteur

2.5. Moteur à courant continu

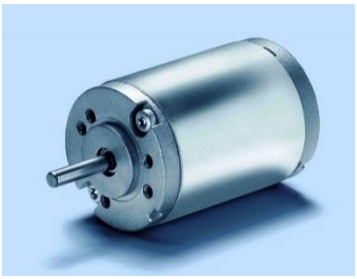


Figure 11 : Moteur à CC

Le moteur à courant continu permettra à notre voiture (robot) de pouvoir avancer, reculer. Lors du projet, nous n'en n'utiliserons que deux.
Les deux moteurs se situeront à l'arrière, du coup, à l'avant nous aurons deux roues qui suivront le mouvement.

2.5.1. Branchement

Voici un schéma permettant de démontrer le branchement d'un moteur CC :

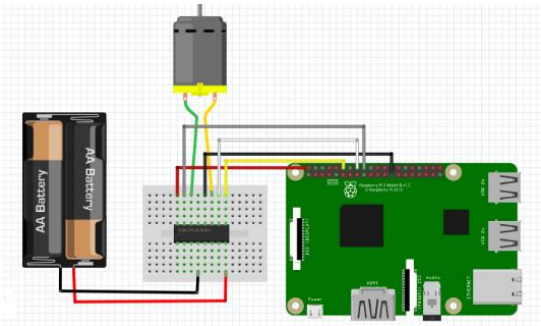


Figure 12 : Branchement du moteur

Pour le branchement permettant de faire fonctionner le moteur :

- Nous aurons besoin du composant SN754410NE ou L293N

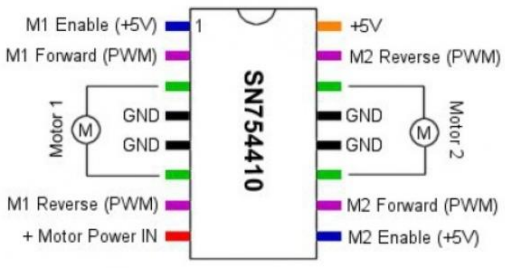


Figure 13 : Schéma du SN754410NE

- Une broche VCC (GPIO1) : +5V
- La broche M1 Enable et M2 Enable permettent d'activer les ponts
- La broche M1/M2 Forward et M1/M2 Reverse permettent d'envoyer une tension afin de le diriger dans une direction : PWM
- Motor 1 et Motor 2, sont reliées aux bornes des moteurs.
- Une broche GND

2.5.2. Fonctionnement

Pour faire correctement fonctionner notre voiture, il faut que celle-ci puisse avancer mais aussi reculer. Pour cela, nous aurons besoin d'un pont en H (L293N ou SN754410NE).

Le pont en H est un composant qui permettra à notre robot de pouvoir avancer et reculer sans devoir inverser le branchement des bornes pendant qu'il roule. Ce composant est capable de faire passer le courant d'un moteur dans un sens et puis dans l'autre en basculant des interrupteurs (transistors).

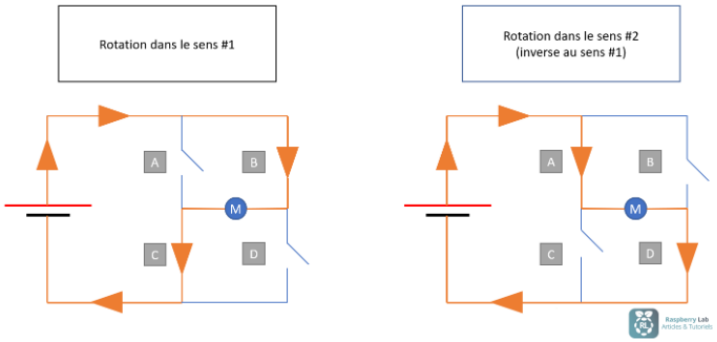


Figure 14 : Fonctionnement du pont H

Ensuite, nous devons pouvoir gérer la vitesse de rotation du robot à l'aide de la technologie PWM. Pour cela, nous n'allons pas envoyer une tension continue au moteur mais plutôt une tension entre +VCC et 0V.
Le moteur va d'abord être alimenté, il va donc commencer à tourner, puis, la tension va être coupée. Le moteur n'aura donc pas le temps de s'arrêter et va donc rouler moins vite.

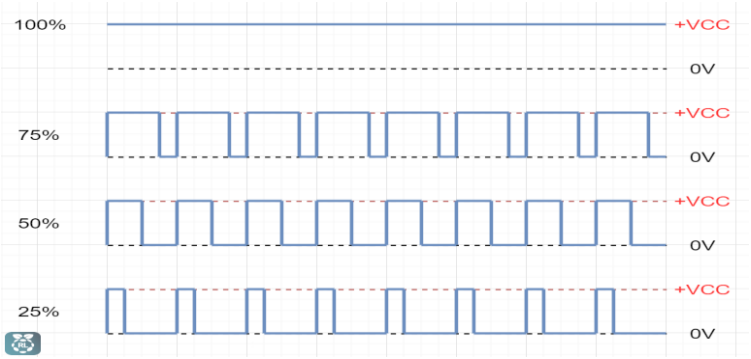


Figure 15 : Régulation de la vitesse du moteur

3. Présentation de la méthodologie SCRUM

3.1. Qu'est-ce que la méthode SCRUM ?

Tout d'abord, le terme « scrum » signifie « mêlé » et fait référence à la mêlée du rugby. Cette méthode est basée sur la méthode Agile et est dédiée à la « gestion de projet ». Il permet notamment de modifier ou de délivrer un produit très rapidement, ce qui n'était pas le cas des méthodes traditionnelles.

Pour fonctionner, cette méthode se base sur trois piliers :

- Communiquer
- Faire le point
- S'adapter

Ces piliers sont gérés respectivement par le Scrum Master, l'équipe et enfin le Product Owner.

Le Scrum Master doit s'assurer que les différents principes de la méthode Scrum soient respectés, ainsi que la bonne communication entre les différents membres de l'équipe afin de pouvoir améliorer la productivité.

L'équipe s'organise entre eux afin de pouvoir atteindre leur objectif dans le temps imparti. Celle-ci est composée de cinq à dix personnes (développeurs, testeurs). Le Scrum Master fait également partie de l'équipe.

Le Product Owner collabore activement avec le client et va notamment endosser son rôle auprès de l'équipe pour définir les spécificités fonctionnelles du produit que veut le client. Il se charge notamment de les prioriser et de valider les fonctionnalités développées par l'équipe.

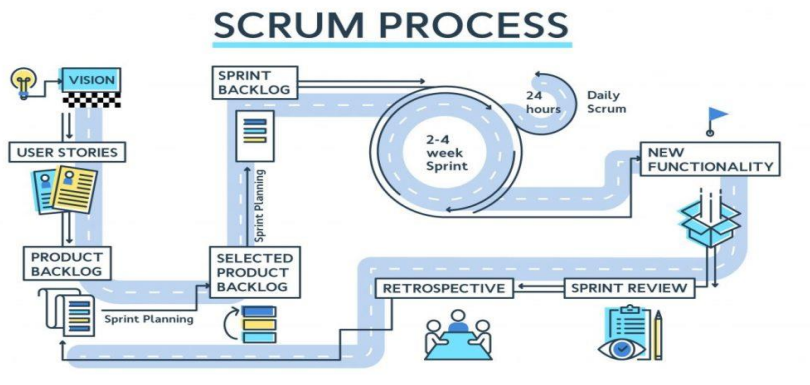


Figure 16 : Méthodologie SCRUM

3.2. Mise en pratique de la méthode SCRUM ?

La méthode Scrum commence par la rédaction des différentes **User Story**, c'est-à-dire ce que veut concrètement le client ou tout simplement, quelle fonctionnalité son produit aura-t-il besoin. Lors de la semaine de projet, les user story nous avait déjà été communiqué par le product owner.

Par exemple : *En tant qu'utilisateur du module je veux pouvoir faire avancer et reculer ma voiture dans le but de faire des courses.*

De ces différentes story, va émaner des exigences, elles seront hiérarchisées dans le **Product Backlog**. Ce product backlog pourra continuellement évoluer pour refléter les besoins.

Par exemple, pour compléter la story défini précédemment :

- *Réalisation des scripts permettant de faire avancer et reculer la voiture.*
- *Réalisation des tests du module.*

Ensuite, c'est à ce moment-là que nous pouvons réellement commencer le projet. Celui-ci sera découpé en plusieurs itérations que l'on nomme **Sprint** (3). L'ensemble des sprints est appelé le **Sprint Backlog.** En tant normal, un sprint à tendance à durer entre deux et quatre semaine. Malheureusement, le projet ne dure qu'une seule semaine, c'est pourquoi nous avons donné une limite de **1,5 jours** par sprint.

En chaque début de sprint, le scrum master organisera un **Sprint Planning Meeting** qui durera maximum **30 minutes** et qui permettra de décider quels seront les users story qui seront développées pendant ce sprint, ainsi que la répartition des différentes tâches pour réaliser les story.

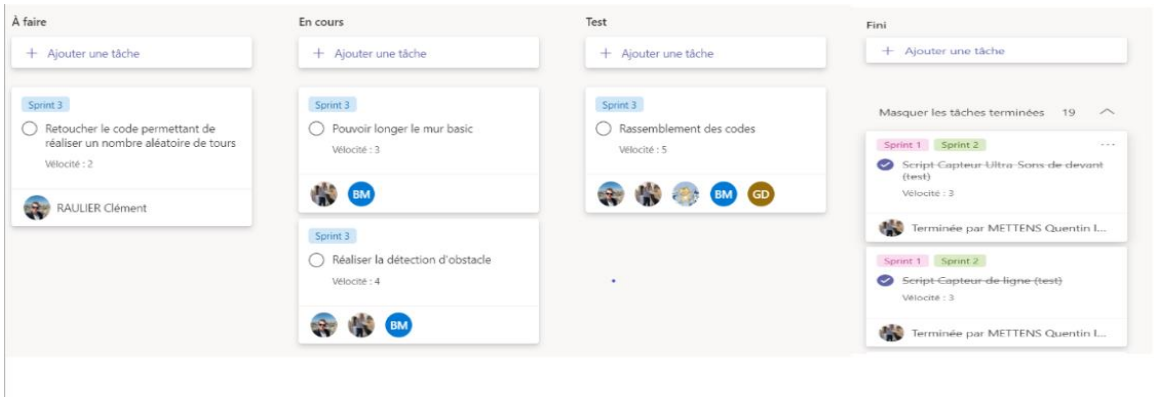


Figure 17 : Scrum Board (sprint 3)

Ces notamment à ce moment que l'équipe utilisera la technique du **T-Shirt Sizing** (1 à 5) pour définir un degré de vélocité pour les différentes tâches. Lorsque l'équipe avait des difficultés à définir la vélocité des tâches, nous utilisions la technique du **Planning Poker** qui permettait de noter les différentes tâches à l'aide de carte.

Figure 18 : Planning Poker

Ensuite, pour continuer de respecter cette méthodologie scrum, nous avons réalisé chaque matin un **Daily Scrum** (15 minutes). Le daily scrum est une sorte de mêlée qui se déroule débout où chaque membre de l'équipe pourra discuter de ce qu'il a fait la **veille**, de ces **problèmes rencontrés** ainsi que de ce qu'il comptait faire **aujourd'hui**.

Figure 19 : Daily Scrum

Le scrum master tient un Burndown Chart (Diagramme de Gant[1]), c'est un graphique permettant de voir l'état d'avancement du projet.

Celui-ci permettra d'avoir une vision claire sur l'aspect général du projet, ainsi que des tâches en cours et de celles à venir dans les différents sprints. Cela nous permettra donc de nous rendre compte de la faisabilité du projet.

[1] Voir annexe

À la fin de chaque daily scrum, le scrum master mettra à jour son diagramme de Gant et déléguera ensuite les différentes tâches qui posent problèmes aux différents membres de l'équipe.

À la fin du sprint, un rendez-vous est pris avec les clients, celui-ci s'appelle le **Sprint Review** et consiste à présenter les fonctionnalités qui ont été réalisées lors du sprint sous forme de démonstration afin d'avoir un retour du client.

Les éventuelles améliorations suggérées ou les problèmes rencontrés seront ensuite rajoutés dans le product backlog et priorisés pour le sprint suivant.

À la fin du sprint review, l'équipe calcule sa vélocité en additionnant les points d'estimation associés aux différentes tâches acceptées. Une tâche non terminée ou partiellement terminée ne rapportera aucun point car celle-ci n'est pas utilisable. Lorsque la vélocité a été calculé, le scrum master pourra réaliser le diagramme de vélocité[2] qui se base sur la vélocité théorique (c'est-à-dire le nombre de points que nous avons attribués aux tâches lors du sprint planning meeting) mais aussi sur la vélocité réelle (ces la vélocité que nous venons de calculer).

L'ensemble de l'équipe doit se réunir pour organiser un **Sprint Retrospective**. Ce rendez-vous a comme objectif de faire réfléchir l'ensemble de l'équipe sur la manière dont il vient de travailler pendant le sprint. On va notamment parler des différentes choses **négatives**, ce que l'on devrait **garder** ou encore ce que l'on devrait **commencer** à faire pour le prochain sprint. En d'autres termes, elle consiste à améliorer petit à petit son mode d'organisation et son mode de fonctionnement afin de faire évoluer l'efficacité et la productivité de l'équipe lors du prochain sprint.

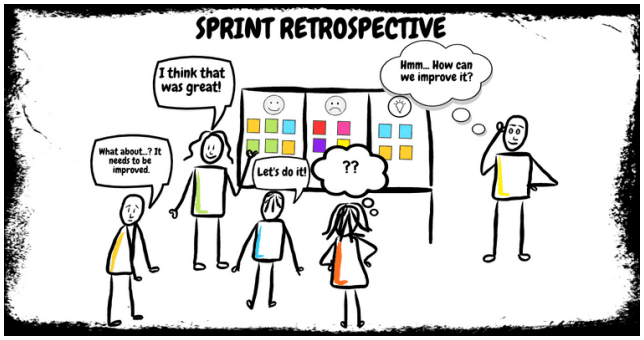


Figure 20: Sprint Retrospective

[2] Voir annexe

4. Déroulement du projet

4.1. Installation et configuration de l'OS

Pour débuter le projet, nous devons préparer une carte micro SD afin d'y installer le système d'exploitation du raspberry qui servira de disque dur pour celui-ci.
Dans un premier temps, il est nécessaire de la formater. Il existe plusieurs moyens, mais le plus simple reste de l'insérer dans un ordinateur (sans oublier de l'ouvrir en écriture) et de la formater depuis l'explorateur de fichier.

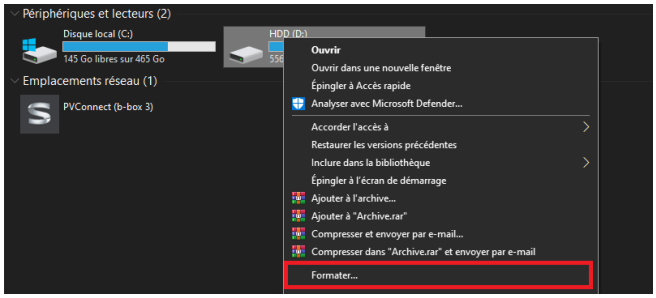


Figure 21: Exemple de formatage

Dans un second temps, il faut installer le système d'exploitation, à savoir Raspbian sur la carte SD via Win32 Disk Imager. Il suffit d'installer la dernière version de l'OS, d'insérer la carte SD dans notre PC et d'utiliser Win32 Disk Imager qui s'occupera de tout.

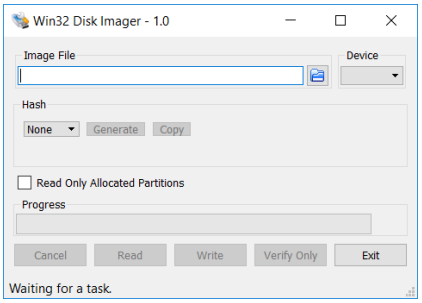


Figure 22: Win32 Disk Imager

Lorsque l'OS est installé sur la carte SD, on peut directement se connecter au raspberry PI en le connectant à un écran, et en y branchant souris et clavier. Le nom d'utilisateur et le mot de passe par défaut sont : pi et raspberry. Maintenant que nous sommes connectés au raspberry, on arrive sur l'interface de configuration. Le plus important de cette configuration étant l'activation du SSH. Disponible depuis la partie « Advanced Options » ➔ « SSH » ➔ « Enable ».

Il est ensuite nécessaire de vérifier que le système est bien à jour. Pour se faire, on ouvre un terminal afin de pouvoir lancer la commande suivante :

```
sudo aptitude update -y && sudo aptitude upgrade -y && sudo reboot
```

Maintenant que le raspberry est prêt, il va nous falloir se connecter en SSH. Celui-ci étant déjà activé, il faut que le raspberry soit connecté au même réseau que nous. Notre première idée a été de le connecter directement au réseau de l'école, mais le raspberry ne semblait pas le supporter. On a alors envisagé de le connecter directement en Ethernet en lui donnant une adresse IP fixe. Cette solution nous a permis de travailler les premières heures, mais n'était pas pratique sur le long terme.

4.1.1. Création d'un hotspot Wi-Fi

Pour pallier au problème de connexion SSH, nous avons opté pour la création d'un hotspot wi-fi directement sur le raspberry afin de pouvoir se connecter à son réseau et donc de pouvoir se connecter en SSH par wi-fi.
Pour se faire, nous avons besoin d'accéder au terminal. Afin d'éviter tout problème, on travaille sur une copie du fichier de configuration permettant la connexion à une box.

```
sudo cp
/etc/wpa_supplicant/wpa_supplicant.conf/etc/wpa_supplicant/wpa_supplicant.c
onf.sav
sudo cp /dev/null /etc/wpa_supplicant/wpa_supplicant.conf
```

On édit donc le fichier suivant : `/etc/wpa_supplicant/wpa_supplicant.conf`

En y rajoutant les lignes suivantes :

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
```

Maintenant que l'interface Wi-Fi est disponible, nous pouvons installer le logiciel (RaspAP) qui va nous permettre de fabriquer un point d'accès depuis la Raspberry PI.
Nous avons donc qu'à insérer une commande qui va installer ce logiciel, ainsi qu'une interface graphique qui nous permet la gestion et la configuration de cet hotspot.
(Lien GitHub du logiciel : https://github.com/RaspAP/raspap-webgui)

```
wget -q https://git.io/voEUQ -O /tmp/raspap && bash /tmp/raspap
```

Dès lors que le hotspot est installé, on peut s'y connecter en Wi-Fi et donc en SSH via l'adresse IP du Raspberry (l'username et le mot de passe sont toujours les mêmes : pi et raspberry). Par facilité, nous avons donné une IP fixe à celui-ci.
Enfin, pour transférer nos fichiers depuis un poste de travail, nous avons opté pour Filezilla qui utilise la connexion SSH pour transférer nos scripts et programmes.

4.1.2. Installation de Python3

Raspbian possédant python2 de base, nous avons installé python3 afin de pouvoir lancer nos scripts, eux aussi, en python3. Pour se faire, une seule commande suffit depuis un terminal :

```
sudo apt-get install python3
```

4.2. Avancer et tourner

Au niveau du raspberry, on peut retrouver des roues motorisées à l'arrière qui lui permettent soit d'avancer, soit de reculer dans notre cas.
Pour ce faire, il nous fallait faire en sorte de démarrer les deux moteurs en même temps afin de permettre à la voiture de bouger en avant ou en arrière.
De plus, la vitesse de ces moteurs est modifiable, on peut la régler comme on le souhaite.

Il aurait été possible de faire tourner la voiture grâce à ces roues arrière en faisant reculer l'une et avancer l'autre...
Cependant, il était plus efficace et plus simple d'utiliser les deux roues avant montées sur un essieu et contrôlées avec un servo-moteur.
De ce fait pour tourner la voiture à droite ou à gauche, il nous fallait seulement donner un angle au servo afin qu'il oriente les roues dans le sens voulu.

4.3. Demi-tour

Nous déclarons tout d'abord les pins correspondants pour l'utilisation des moteurs ainsi qu'un objet pour le servo-moteur (la direction). Nous créons une fonction *setup* afin de définir l'état des pins. Dans la fonction *forward* et *backward,* Nous faisons tourner les moteurs durant la variable *tps* qui vaut 2.5 secondes. Nous finissons la fonction par éteindre les moteurs. Dans les fonctions suivantes (*tournerG, tournerD, tournerGA, tournerDA*), nous définissons un angle pour le servo-moteur qui permet de tourner l'essieu à gauche ou à droite selon la fonction. Nous faisons ensuite appel à la fonction de *forward* ou *backward* pour la faire avancer pendant que les roues sont tournées. Nous terminons par remettre les roues droites. Dans la main nous appelons les fonctions afin d'effectuer tous les demi-tours possibles.

4.4. Longer le mur

Le capteur ultrason est l'un des composants principaux du projet. Il permet non seulement de longer les murs mais aussi de détecter et d'esquiver par la gauche les obstacles se trouvant devant lui. Il permet notamment d'éviter les collisions avec les véhicules qui se trouvent sur ses côtés.

Tout d'abord, pour commencer, nous allons placer le robot (voiture) plus ou moins à quelques dizaines de centimètres du mur. Ensuite, la broche va émettre une impulsion via l'émetteur du capteur de gauche. Lorsque l'onde aura détecté un obstacle, elle va revenir en écho sur la broche réceptrice du capteur. Le but du programme, est de s'éloigner et de se rapprocher du mur en effectuant des gauches droites pour que le robot ne percute pas le mur.

Pour cela, lorsque la distance entre le mur et le capteur de gauche est de plus de 70 cm, la voiture va se rapprocher du mur en tournant les roues à un angle de 85 ° à l'aide du servomoteur. Au fur et à mesure qu'on avance vers le mur, la voiture va de plus en plus ralentir jusqu'à ce que celle-ci soit à une distance entre 32 et 34 centimètres du mur. Etant donné que le Raspberry met un certain temps pour calculer les informations et les envoyer au moteur à courant continue ou au servomoteur, il se peut que la distance se réduise. Pour cela, lorsque la distance entre le mur et le capteur est de moins de 30 centimètres, la voiture va accélérer et tourner vers la droite à un angle de 130°.
Malheureusement, lors du dernier sprint review, ce programme fût un échec. Lorsque la voiture était dans le circuit, elle se mettait directement à 130° et la faisait percuter le mur droit.

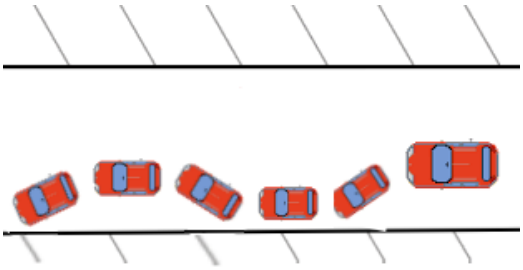


Figure 23 : Longer le mur (théorie)

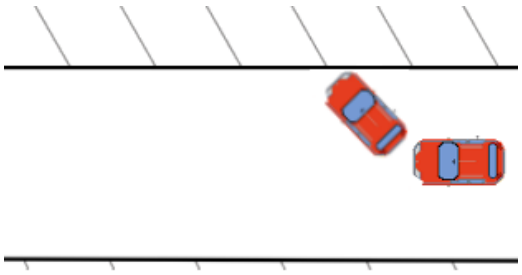


Figure 24: Longer le mur (réelle)

4.5. Détection d'obstacle

Ensuite, nous avions dû implémenter la détection d'obstacle à l'aide notamment du capteur de devant.

À partir du moment où la distance entre le capteur et l'obstacle est comprise entre 51 et 58 cm, la vitesse du robot va être divisée par deux et va permettre ainsi à la machine d'avoir le temps de freiner. Le temps que la voiture freine celle-ci se rapprochera de l'obstacle, à ce moment-là, la distance qui sépare le capteur et l'obstacle sera comprise entre 16 et 48 cm.

Dès lors, la voiture commencera par tourner de 40 ° vers la gauche, attendra deux secondes, permettant ainsi à la voiture de lui laisser le temps pour esquiver l'obstacle. Pour au final tourner de 160 ° vers la droite afin de reprendre sa trajectoire initiale.

Malheureusement, lors du dernier sprint review, ce programme fût un échec. La voiture n'avait pas assez de temps entre le moment où elle commence par tourner à gauche et lorsqu'elle reprend sa position initiale. En d'autres termes, elle finissait par percuter l'obstacle par la droite.

Figure 25: Evitemment d'obstacle

4.6. Détection de la ligne d'arrivée

Nous déclarons une classe ainsi que son constructeur. Par la suite la fonction. La variable *stp* sert à retourner un état afin de savoir si la voiture a fait son nombre de tour voulu. La variable *i* sert de compteur. Nous demandons à l'utilisateur de rentrer un nombre de tour à effectuer. Cette valeur est stockée afin de la réutilisé plus tard.

Au niveau de la boucle : la condition est que si l'état actuel et précédent sont différent, nous comptons un tour, sinon il ne se passe rien. Si le compteur est inférieur à 1, nous ne faisons rien car la voiture démarre derrière la ligne d'arrivée. Elle va donc détecter la ligne mais elle ne doit pas compter. Si le compteur est supérieur, elle affiche à l'écran qu'on a passé la ligne. Quand le compteur – 1 est égal au nombre entré, l'état de la variable *stp* sera modifiée, ce qui permettra d'arrêter le véhicule.

5. Problèmes rencontrés

Lors de la semaine de projet, tout ne sait pas passer comme prévu, notamment avec l'utilisation de la méthode scrum mais aussi lors de la programmation des différents composants.
Heureusement à la fin de chaque sprint, nous avions la possibilité de réunir l'ensemble de l'équipe pour pouvoir mettre sur la table les différents problèmes rencontrés (sprint retrospective). Ce qui nous a permis de nous améliorer tant que collectivement qu'individuellement.

Lors de la première rétrospective, nous avions pu remarquer que l'entraide était un des points forts de l'équipe. Le seul problème, c'est que la plupart d'entre nous prenaient pas mal de temps à aider les autres pour finalement ne pas avancer sur les différentes tâches qui avaient été attribués.
Le deuxième problème qui avait été répertorié, c'est que nous avons chacun commencé à programmer en prenant une version différente de Python (2-3), ce qui avait causé quelque retard.
Mais l'un des plus gros problèmes, est que nous n'avions pas eu la bonne idée d'effectuer des tests en milieu extérieur, notamment pour tester l'hotspot Wi-Fi sur le Raspberry. Du coup, nous n'avions rien su présenter de fonctionnel aux différents clients lors du sprint review.

Lors de la seconde rétrospective, nous avions été fortement perturbé et ralenti puisque certains avaient commencé à programmer en orienter objet alors que d'autres codaient de manière séquentielle. Du coup, cela posait quelques problèmes lorsque nous avions dû faire avancer et tourner le robot en même temps.
De plus, nous n'avions pas fait très attention au niveau de durabilité des piles lors du sprint review. Nous avions dû enlever et remettre des nouvelles piles pour faire fonctionner le robot, ce qui ne faisait pas très professionnel devant les clients.
Enfin, nous avons découvert que certains câbles n'étaient pas branchés au bon emplacement. Afin de ne pas s'emmêler dans les connexions des câbles, nous avons décidé de tout retirer et de tout recâbler. Plus tard, nous avons été aider pour un problème qu'on pensait provenir du code, mais en réalité se situait au niveau du câblage. Sur le schéma, les connexions à l'I2C devaient être permutées.

Lors du dernier sprint, ce qui nous a posé le plus gros problème, est le manque de test des différentes fonctionnalités du robot sur le circuit, notamment la fonctionnalité longer le mur ou encore l'évitement d'obstacle. Nous aurions pu valider ces user story, si nous avions pris plus de temps pour les tester.

6. Améliorations et conclusions

Pour conclure ce projet, nous pouvons assumer cette morale : *Le chemin de l'autonomie est long et semé d'obstacles que nos capteurs ne pourront pas toujours capter, ce qui rend leur évitement plus compliqué*. Cette morale est d'autant plus vraie que lorsque l'on essaie de créer un véhicule autonome.

Néanmoins, cette semaine nous aura appris de nombreuses choses, telles que le travail en équipe, assisté d'une méthodologie dont les avantages ne sont plus à démontrer, ou encore la programmation orientée objet à l'aide d'un des langages de programmation parmi les plus utilisés (Python), le tout sur un Raspberry Pi.

Certes, tout ne s'est pas passé comme nous l'espérions, nous aurions pu améliorer la fluidité du code en utilisant les différents capteurs ultrasons en simultané, éviter de passer plusieurs heures sur un câblage mal fait, ou encore essayer de passer beaucoup plus de temps sur le circuit pour tester et ainsi apercevoir les différents problèmes que nous aurions pu éviter lors du sprint review.

Enfin, ce projet nous aura appris à confronter nos avis et à trouver des solutions ensemble à certains problèmes, mais aussi et surtout il nous aura permis d'affirmer nos choix d'orientation pour le futur.

7. Bibliographie

7.1. Syllabus

- Michiels M., *Initiation nano-ordinateurs*, Bachelier en informatique et système finalité réseaux et télécommunications, Année académique 2020-2021
- Naizy G., *Gestion de projet informatique* : SCRUM, Bachelier en informatique et système finalité réseaux et télécommunications, Année académique 2020-2021

7.2. Source électronique

7.2.1. Raspberry

- D-clics Numérique, 2017, *Introduction : Fonctionnement du Raspberry PI 3*, [en ligne] https://d-clics.laligue24.org/tutoriel/raspberry/51-introduction-fonctionnement-du-raspberry-pi-3
- Conrad, 2018, *Raspberry PI RP-3B+ 1G sans système d'exploitation*, [en ligne] https://www.conrad.fr/p/raspberry-pi-rp-3b-1-gb-sans-systeme-dexploitation-1668026
- Léa-Linux, 2013, *Présentation du Raspberry PI*, [en ligne] https://lea-linux.org/documentations/Pr%C3%A9sentation_du_Raspberry_Pi
- Grafikart, 2015, *Qu'est-ce qu'un Raspberry PI*, [en ligne] https://grafikart.fr/blog/raspberry-pi-utilisation

7.2.2. Capteur de Ligne

- Sensorkit,2017, *KY-033 module suiveur de ligne*, [en ligne] https://sensorkit.fr.joy-it.net/index.php?title=KY-033_Module_suiveur_de_ligne
- Mocq F., 2016, *Capteur de distance et suiveur de ligne pour un robot*, [en ligne] https://www.framboise314.fr/deux-capteurs-pour-un-robot/#Capteur_de_suivi_de_ligne
- Zelf Lab (Youtube), 2019, *Raspberry Pi : Robot télécommandé (EP 5 : Emetteur/Récepteur de lumière)*, [en ligne] https://www.youtube.com/watch?v=5DTSjL-F-V8&t=508s&ab_channel=ZEFLAB

7.2.3. Capteur ultrason

- SparkFun Electronics, 2019, *SparkFun HC-SR04 Ultrasonic Distance Sensor*, [en ligne] https://www.mouser.be/new/sparkfun/sparkfun-hcsr04-distance-sensor/
- Freva.com, 2019, *Capteur ultrason HC-SR04 sur Raspberry Pi*, [en ligne] https://www.freva.com/fr/2019/05/20/capteur-ultrason-hc-sr04-sur-raspberry-pi/
- Eskimon (Zeste de savoir), 2018, *Réaliser un télémètre à ultrason*, [en ligne] https://zestedesavoir.com/tutoriels/343/realiser-un-telemetre-a-ultrasons/
- Bachelard L., 2015, *HC-SR04 Module de détection aux ultrasons – Utilisation avec Picaxe*, [en ligne], https://www.gotronic.fr/pj2-hc-sr04-utilisation-avec-picaxe-1343.pdf
- CFaury, 2016, *Capteur de distance à Ultrasons*, [en ligne] https://arduino.blaisepascal.fr/capteur-de-distance-a-ultrasons/

7.2.4. Moteur à CC

- Maulny J., 2017, *Contrôler un robot à deux roues avec Raspberry*, [en ligne] https://alcalyn.github.io/control-robot-two-engines/
- Raspberry Lab, 2019, *Tutoriel Raspberry PI L298N : Contrôle de moteur à cc*, [en ligne] https://raspberry-lab.fr/Composants/Module-L298N-controleur-moteur-Raspberry-Francais/
- Aranacorp, 2018, *Piloter un moteur cc avec Raspberry*, [en ligne] https://www.aranacorp.com/fr/pilotez-un-moteur-cc-avec-raspberrypi/
- Hellopro,, *Moteur à cc m36x13*, [en ligne] https://www.hellopro.fr/moteur-courant-continu-m36x13-2016234-257431-produit.html

7.2.5. Servomoteur

- Raspberry Pi Tutorial,, *Contrôle du servomoteur de Raspberry PI*, [en ligne] https://raspberrypi-tutorials.fr/controle-du-servo-moteur-de-raspberry-pi/
- Raspberry Pi Fr, 2020, *Comment contrôler un servomoteur avec la Raspberry PI*, [en ligne] https://raspberry-pi.fr/servomoteur-raspberry-pi/
- Framboise314, 2017, *Commande d'un servomoteur avec le Raspberry PI*, [en ligne] https://www.framboise314.fr/wp-content/uploads/2017/11/05_01_servomoteur.pdf
- Aranacorp, 2017, *Pilotez un servo avec Arduino*, [en ligne] https://www.aranacorp.com/fr/pilotez-un-servo-avec-arduino/

7.2.6. Méthode Scrum

- Canal Etudiants IAE La Rochelle (Youtube), 2019, *La méthode SCRUM*, [en ligne] https://www.youtube.com/watch?v=I4acqK_Jr70&ab_channel=CanalEtudiantsIAELaRochelle
- La minute Agile (Youtube), 2019, *Origines de l'agile ? La minute Scrum #45*, [en ligne] https://www.youtube.com/watch?v=0057PRCZJZM&ab_channel=LaMinuteAgile
- StrategeMarketing (Youtube), 2019, *SCRUM la méthode agile en 10 minutes (Projet Agile)*, [en ligne] https://www.youtube.com/watch?v=anZcEIQlpoY&ab_channel=StrategeMarketing
- Les agilitateurs (Youtube), 2016, *Comprendre le framework SCRUM en 7 minutes*, [en ligne] https://www.youtube.com/watch?v=_hVw1_4xtT8&ab_channel=LesAgilitateurs
- Wemanity, 2021, *La méthode scrum : définition et mode d'emploi*, [en ligne] https://weblog.wemanity.com/fr/la-methode-scrum-definition-et-mode-demploi/
- Anuraj S.L, 2018, *8 Bad Habits that ruin Daily Scrum Meeting*, [en ligne] https://medium.com/agile-coaching-playbook/8-bad-habits-that-ruin-daily-scrum-meetings-d305134a3638
- Mohammed M. (Youtube), *Scrum pour les nuls*, [en ligne] https://www.youtube.com/watch?v=kZTLlWkxFN4&ab_channel=MoullouzeMohamed
- Let's Scrum It, 2020, *Scrum Events #5 Sprint Retrospective*, [en ligne] https://letsscrumit.com/scrum-events-5-sprint-retrospective/

7.3. Illustrations

8. Annexes

8.1. Programme complet

- Programme moteur

Motor.py

```python
import board
import busio
import adafruit_pca9685
import RPi.GPIO as GPIO
from time import sleep

class Motor(object):

    i2c = busio.I2C(board.SCL, board.SDA)
    hat = adafruit_pca9685.PCA9685(i2c)
    hat.frequency = 50

    M0_En = 23   # pin4
    M0_A  = 17  # pin11
    M0_B  = 18  # pin12

    M1_En = 24   # pin4
    M1_A  = 27  # pin13
    M1_B  = 22  # pin15

    def __init__(self):

        GPIO.setmode(GPIO.BCM)  # GPIO Numbering
        GPIO.setup(self.M0_En, GPIO.OUT)
        GPIO.setup(self.M0_A,  GPIO.OUT)
        GPIO.setup(self.M0_B,  GPIO.OUT)
        GPIO.setup(self.M1_En, GPIO.OUT)
        GPIO.setup(self.M1_A,  GPIO.OUT)
        GPIO.setup(self.M1_B,  GPIO.OUT)
        self.motorL = GPIO.PWM(self.M0_En, 100)
        self.motorR = GPIO.PWM(self.M1_En, 100)

    """ Goes Forwards for 'x' seconds at speed 'speed' """
    def forwards(self, speed = 100):
        GPIO.output(self.M0_A,  GPIO.HIGH)
        GPIO.output(self.M0_B,  GPIO.LOW)
        GPIO.output(self.M0_En, GPIO.HIGH)
        GPIO.output(self.M1_A,  GPIO.HIGH)
        GPIO.output(self.M1_B,  GPIO.LOW)
        GPIO.output(self.M1_En, GPIO.HIGH)
        self.motorL.start(speed)    # TODO : Tester si 0 < speed <= 100 ?
        self.motorR.start(speed)
        print("Go !")
        #sleep(t)
        #self.stop()
        print("stop !")

    """ Goes Backwards for 'time' seconds at speed 'speed' """
    def backwards(self, speed = 100):
        GPIO.output(self.M0_A,  GPIO.LOW)
        GPIO.output(self.M0_B,  GPIO.HIGH)
        GPIO.output(self.M0_En, GPIO.HIGH)
        GPIO.output(self.M1_A,  GPIO.LOW)
        GPIO.output(self.M1_B,  GPIO.HIGH)
        GPIO.output(self.M1_En, GPIO.HIGH)
        self.motorL.start(speed)
        self.motorR.start(speed)
        print("Go !")
        #sleep(t)
        print("before stop !")
        #self.stop()
        print("stop !")


    """ Stops the car """
    def stop(self):
        print("stop")
        GPIO.output(self.M0_En, GPIO.LOW)
        GPIO.output(self.M1_En, GPIO.LOW)

if __name__ == '__main__': # Program starts here

    motorExec = Motor()
    try:
        print("test")
        motorExec.forwards(80)
        sleep(1)
        motorExec.forwards(50)
        sleep(1)
        motorExec.forwards(30)
        sleep(1)
        motorExec.forwards(10)
        sleep(1)
        motorExec.stop()
        print("Nice")
    except:
        print("Rate")
```

- Programme tourner

Servo.py

```python
from adafruit_servokit import ServoKit
import RPi.GPIO as GPIO
kit = ServoKit(channels=16)

class Servo:
    def __init__(self):
        self.angle = 90

    def turn(self, x):
        self.angle = x
        kit.servo[0].angle = self.angle

    def straight(self):
        self.angle = 92
        kit.servo[0].angle = self.angle
```

- Programme demi-tour

demitour.py

```python
import time
import RPi.GPIO as GPIO
from adafruit_servokit import ServoKit

#Pin assiciation for motor
M0_En = 23  # pin4
M0_A  = 17  # pin11
M0_B  = 18  # pin12

M1_En = 24  # pin4
M1_A  = 27  # pin13
M1_B  = 22  # pin15

kit = ServoKit(channels=16)

tps=2.5

def setup():
    GPIO.setmode(GPIO.BCM)  # GPIO Numbering
    GPIO.setup(M0_En, GPIO.OUT)
    GPIO.setup(M0_A,  GPIO.OUT)
    GPIO.setup(M0_B,  GPIO.OUT)
    GPIO.setup(M1_En, GPIO.OUT)
    GPIO.setup(M1_A,  GPIO.OUT)
    GPIO.setup(M1_B,  GPIO.OUT)

def forward(tps):
    # Going Forwards
    GPIO.output(M0_A,  GPIO.HIGH)
    GPIO.output(M0_B,  GPIO.LOW)
    GPIO.output(M0_En, GPIO.HIGH)
    GPIO.output(M1_A,  GPIO.HIGH)
    GPIO.output(M1_B,  GPIO.LOW)
    GPIO.output(M1_En, GPIO.HIGH)

    time.sleep(tps)

    # Stop
    GPIO.output(M0_En, GPIO.LOW)
    GPIO.output(M1_En, GPIO.LOW)

def backward(tps):
    # Going Backwards
    GPIO.output(M0_A,  GPIO.LOW)
    GPIO.output(M0_B,  GPIO.HIGH)
    GPIO.output(M0_En, GPIO.HIGH)
    GPIO.output(M1_A,  GPIO.LOW)
    GPIO.output(M1_B,  GPIO.HIGH)
    GPIO.output(M1_En, GPIO.HIGH)

    time.sleep(tps)

    # Stop
    GPIO.output(M0_En, GPIO.LOW)
    GPIO.output(M1_En, GPIO.LOW)

def tournerG():
    kit.servo[0].angle =0
    forward()
    kit.servo[0].angle = 103

def tournerD():
    kit.servo[0].angle =180
    forward()
    kit.servo[0].angle = 97

def tournerGA():
    kit.servo[0].angle =0
    backward()
    kit.servo[0].angle = 103

def tournerDA():
    kit.servo[0].angle =180
    backward()
    kit.servo[0].angle = 97

def tournerGMur(x):
    x=0
    kit.servo[0].angle = x
    forward(0.5)
    kit.servo[0].angle = 103

def tournerDMur(x):
    x=180
    kit.servo[0].angle = x
    forward(0.5)
    kit.servo[0].angle = 97

def toutdroit():
    kit.servo[0].angle =100
    forward(0.5)

if __name__ == '__main__': # Program starts here
    setup()
    try:
        tournerD()
        tournerG()
        tournerDA()
        tournerGA()
    except KeyboardInterrupt:
        destroy()
```

- Programme ultrason

CapteurUltraSons.py

```python
import RPi.GPIO as GPIO
import time
import demitour
import threading
from Motor import Motor
from Sovor import Servo


GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)



class CapteurUltraSons(threading.Thread):
    avancer = Motor()
    def __init__(self, sonar, recept, distanceObstacle):
        threading.Thread.__init__(self)
        self.pinSonar = sonar
        self.pinRecept = recept
        self.distanceObstacle = distanceObstacle

    def cleanup(self):
        GPIO.cleanup()

    def getDistance(self):
        while True :

            """Sortie"""
            GPIO.setup(self.pinRecept, GPIO.OUT)

            """Envoie une impulsion de 10 microseconde"""
            GPIO.output(self.pinRecept, True)
            time.sleep(0.00001)
            GPIO.output(self.pinRecept, False)

            start = time.time()
            count = time.time()

            """Entree"""
            GPIO.setup(self.pinSonar,GPIO.IN)

            """Sauvegarde le temps de depart de l'impulsion qui a ete emise """
            while GPIO.input(self.pinSonar)==0 and time.time()-count<0.1:
                start = time.time()

            count=time.time()
            stop=count

            while GPIO.input(self.pinSonar)==1 and time.time()-count<0.1:
                """Sauvegarde le temps d'arrivee de l'impulsion """
                stop = time.time()
```

CapteurUltraSons.py

```python
            """Calcul la difference de temps entre l'emission de l'impulsion et de la reception"""
            elapsed = stop-start

            """Multiplication du temps par la vitesse du son"""
            distance = elapsed * 34000
            distance = distance / 2

            """Permet d'arrondir la variable distance a deux decimales"""
            distance = round(distance,2)

            if distance <= 60 and self.pinSonar == 5 and self.pinRecept == 6:
                print ("Distance:",distance - 0.5,"cm", "Obstacle detecte-devant")
                time.sleep(0.1)


            elif distance > 60 and self.pinSonar == 5 and self.pinRecept == 6:
                print ("Aucun soucis-devant")
                time.sleep(0.5)




            if distance < self.distanceObstacle and self.pinSonar == 19 and self.pinRecept == 26:
                print ("Distance:",distance - 0.5,"cm", "Obstacle detecte-Droite")
                time.sleep(0.5)

            elif distance > self.distanceObstacle and self.pinSonar == 19 and self.pinRecept == 26:
                print ("Aucun soucis-Droite")
                time.sleep(0.5)



            if distance < 10 and self.pinSonar == 9 and self.pinRecept == 11:
                print ("Ecarte Mur Gauche")



            if distance > 10 and self.pinSonar == 9 and self.pinRecept == 11:
                print("Fonce vers le mur Gauche")


            if distance > 70 and distance < 200 and self.pinSonar == 9 and self.pinRecept == 11:
                print ("vers Mur Gauche")
                time.sleep(0.11)
                print (distance)
                moteur.forwards(40)
                tourner.turn(85)
```

CapteurUltraSons.py

```
            elif distance > 50 and distance < 69 and self.pinSonar == 9 and self.pinRecept == 11:
                print ("vers Mur Gauche")
                time.sleep(0.11)
                avancer.forwards(30)
                tourner.turn(85)
            elif distance > 36 and distance < 49 and self.pinSonar == 9 and self.pinRecept == 11:
                print ("vers Mur Gauche")
                time.sleep(0.11)
                print (distance)
                avancer.forwards(20)
                tourner.turn(88)
            elif distance > 31 and distance < 35 and self.pinSonar == 9 and self.pinRecept == 11:
                print ("vers Mur Gauche")
                avancer.forwards(20)
            elif distance < 30 and self.pinSonar == 9 and self.pinRecept == 11:
                print ("Ecarte Mur Gauche")
                time.sleep(0.11)
                print (distance)
                avancer.forwards(40)
                tourner.turn(130)
            else:
                avancer.forwards(40)
                tourner.straight()




    def run (self):
        print ("Run - ", self.pinSonar, self.pinRecept)
        self.getDistance()
        self.cleanup()

"""
def main():
        thread1 = CapteurUltraSons(5,6,60)
        thread1.start()

        thread2 = CapteurUltraSons(19,26,60)
        thread2.start()

        thread3 = CapteurUltraSons(9,11,60)
        thread3.start()
"""
```

- Programme longer le mur

Longermur.py

```
import adafruit_pca9685
from adafruit_servokit import ServoKit
import RPi.GPIO as GPIO
import time
from Servo import Servo
from Motor import Motor
class CapteurUltraSons():

    def __init__(self, sonar, recept, distanceObstacle):
        self.pinSonar = sonar
        self.pinRecept = recept
        self.distanceObstacle = distanceObstacle


    def cleanup(self):
        GPIO.cleanup()

    def getDistance(self):

        while True :

            """Sortie"""
            GPIO.setup(self.pinRecept, GPIO.OUT)

            """Envoie une impulsion de 10 microseconde"""
            GPIO.output(self.pinRecept, True)
            time.sleep(0.00001)
            GPIO.output(self.pinRecept, False)

            start = time.time()
            count = time.time()

            """Entree"""
            GPIO.setup(self.pinSonar,GPIO.IN)

            """Sauvegarde le temps de depart de l'impulsion qui a ete emise """
            while GPIO.input(self.pinSonar)==0 and time.time()-count<0.1:
                start = time.time()

            count=time.time()
            stop=count

            while GPIO.input(self.pinSonar)==1 and time.time()-count<0.1:
                """Sauvegarde le temps d'arrivee de l'impulsion """
                stop = time.time()

            """Calcul la difference de temps entre l'emission de l'impulsion et de la reception"""
            elapsed = stop-start

            """Multiplication du temps par la vitesse du son"""
            distance = elapsed * 34000
            distance = distance / 2
```

```python
            """Permet d'arrondir la variable distance a deux decimales"""
            distance = round(distance,2)
            return distance



if __name__ == '__main__': # Program starts here

    capteurGauche = CapteurUltraSons(9,11,100)
    avancer = Motor()
    tourner = Servo()

    while True:
            distancegauche = capteurGauche.getDistance()
            time.sleep(0.1)
            print(distancegauche)
            if distancegauche > 49:
                print ("vers Mur Gauche")
                time.sleep(0.11)
                avancer.forwards(40)
                tourner.turn(70)
            elif distancegauche > 35 and distancegauche < 49:
                print ("vers Mur Gauche")
                time.sleep(0.11)
                avancer.forwards(30)
                tourner.turn(78)
            elif distancegauche > 20 and distancegauche < 35:
                print ("vers Mur Gauche")
                time.sleep(0.11)
                avancer.forwards(20)
                tourner.turn(80)
            elif distancegauche > 10 and distancegauche < 20:
                time.sleep(0.11)
                avancer.forwards(50)
                tourner.turn(110)
            elif distancegauche < 10:
                print ("Ecarte Mur Gauche")
                time.sleep(0.11)
                avancer.forwards(50)
                tourner.turn(120)
            else:
                avancer.forwards(40)
                tourner.straight()
```

- Programme compteur de ligne

Infrarouge.py

```python
import RPi.GPIO as GPIO                         #Import GPIO library
import time                                     #Import time library

class InfraredSensor (object):

    def __init__(self):                             #Super Constructor

        GPIO.setmode(GPIO.BCM)                        #Set GPIO BCM numbering
        self.IN_SENSOR = 20                           #Associate pin 38 to TRIG
        GPIO.setup(self.IN_SENSOR,GPIO.IN)            #Set pin as GPIO in

    def caption(self):

        stp=False                                    #set the return to stop the car
        i=0                                         #Line counter
        print("Entrer un nombre de tour:")
        cpt=input()                                  #Store the number of loop to do
        cpt=int(cpt)
        etatprecedent=False                         #Boolean to don't count the same line
        while True:                                  #infinte loop

            B=GPIO.input(self.IN_SENSOR)                    #Safe the state of the IS
            if(B==True and etatprecedent!=B):               #condition
                i+=1                                        #Counter increment
                if(i>1):
                    print("détection de la ligne d'arrivée")    #Print line detection
                    print(i-1)                                  #Print counter

            if(i-1==cpt):
                stp=True
                break

            etatprecedent=B                             #Safe the previously state
            time.sleep(0.1)                            #Delay of 1 second
        return stp                                  #Stop the car at the number set by the user

infrarouge= InfraredSensor()                        #New objet
infrarouge.caption()                                #Run the function
```

- Programme « faire un tour du circuit » (story12POO)

Story12POO.py

```python
import board
import busio
import adafruit_pca9685
from adafruit_servokit import ServoKit
import RPi.GPIO as GPIO
import time
import threading
from Servo import Servo
from Motor import Motor

class CapteurUltraSons(threading.Thread):

    def __init__(self, sonar, recept, distanceObstacle):
        threading.Thread.__init__(self)
        self.pinSonar = sonar
        self.pinRecept = recept
        self.distanceObstacle = distanceObstacle


    def cleanup(self):
        GPIO.cleanup()

    def getDistance(self):

        while True :

            """Sortie"""
            GPIO.setup(self.pinRecept, GPIO.OUT)

            """Envoie une impulsion de 10 microseconde"""
            GPIO.output(self.pinRecept, True)
            time.sleep(0.00001)
            GPIO.output(self.pinRecept, False)

            start = time.time()
            count = time.time()

            """Entree"""
            GPIO.setup(self.pinSonar,GPIO.IN)

            """Sauvegarde le temps de depart de l'impulsion qui a ete emise """
            while GPIO.input(self.pinSonar)==0 and time.time()-count<0.1:
                start = time.time()

            count=time.time()
            stop=count

            while GPIO.input(self.pinSonar)==1 and time.time()-count<0.1:
                """Sauvegarde le temps d'arrivee de l'impulsion """
                stop = time.time()
```

Story12POO.py

```python
            """Calcul la difference de temps entre l'emission de l'impulsion et de la reception"""
            elapsed = stop-start

            """Multiplication du temps par la vitesse du son"""
            distance = elapsed * 34000
            distance = distance / 2

            """Permet d'arrondir la variable distance a deux decimales"""
            distance = round(distance,2)
            return distance

```

Story12POO.py

```
64
65
66  if __name__ == '__main__': # Program starts here
67
68      capteurGauche = CapteurUltraSons(9,11,100)
69      capteurDevant = CapteurUltraSons(5,6,60)
70      avancer = Motor()
71      tourner = Servo()
72      while True:
73              distancegauche = capteurGauche.getDistance()
74              time.sleep(0.1)
75              print(distancegauche)
76              distancedevant = capteurDevant.getDistance()
77              time.sleep(0.1)
78              print(distancedevant)
79              if distancedevant < 15 and distancegauche < 15:
80                  print ("PJIDFGPSIDFJGBSDGJDOPFG")
81                  time.sleep(0.11)
82                  avancer.forwards(40)
83                  tourner.turn(170)
84              if distancegauche > 49:
85                  print ("vers Mur Gauche")
86                  time.sleep(0.11)
87                  avancer.forwards(40)
88                  tourner.turn(45)
89              elif distancegauche > 35:
90                  print ("vers Mur Gauche")
91                  time.sleep(0.11)
92                  avancer.forwards(30)
93                  tourner.turn(78)
94              elif distancegauche > 20:
95                  print ("vers Mur Gauche")
96                  time.sleep(0.11)
97                  avancer.forwards(30)
98                  tourner.turn(80)
99              elif distancegauche > 10:
100                 time.sleep(0.11)
101                 avancer.forwards(20)
102                 tourner.turn(130)
103             elif distancegauche < 10:
104                 print ("Ecarte Mur Gauche")
105                 time.sleep(0.11)
106                 avancer.forwards(50)
107                 tourner.turn(140)
108             else:
109                 avancer.forwards(40)
110                 tourner.straight()
111
112
113
```

- Programme Main

main.py

```python
from Servo import Servo
import threading
import RPi.GPIO as GPIO
import time
import adafruit_pca9685
import board
import busio


class CapteurUltraSons(threading.Thread):

    def __init__(self, sonar, recept, distanceObstacle):
        threading.Thread.__init__(self)
        self.pinSonar = sonar
        self.pinRecept = recept
        self.distanceObstacle = distanceObstacle

    def cleanup(self):
        GPIO.cleanup()

    def getDistance(self):
        GPIO.setmode(GPIO.BCM)
        while True :

            """Sortie"""
            GPIO.setup(self.pinRecept, GPIO.OUT)

            """Envoie une impulsion de 10 microseconde"""
            GPIO.output(self.pinRecept, True)
            time.sleep(0.00001)
            GPIO.output(self.pinRecept, False)

            start = time.time()
            count = time.time()

            """Entree"""
            GPIO.setup(self.pinSonar,GPIO.IN)

            """Sauvegarde le temps de depart de l'impulsion qui a ete emise """
            while GPIO.input(self.pinSonar)==0 and time.time()-count<0.1:
                start = time.time()

            count=time.time()
            stop=count

            while GPIO.input(self.pinSonar)==1 and time.time()-count<0.1:
                """Sauvegarde le temps d'arrivee de l'impulsion """
                stop = time.time()

            """Calcul la difference de temps entre l'emission de l'impulsion et de la reception"""
            elapsed = stop-start
```

main.py

```python
            """Calcul la difference de temps entre l'emission de l'impulsion et de la reception"""
            elapsed = stop-start

            """Multiplication du temps par la vitesse du son"""
            distance = elapsed * 34000
            distance = distance / 2

            """Permet d'arrondir la variable distance a deux decimales"""
            distance = round(distance,2)
            return distance
    """
    def run (self):
        print ("Run - ", self.pinSonar, self.pinRecept)
        self.getDistance()
        self.cleanup()
    """
class InfraredSensor (object):

    def __init__(self):                             #Super Constructor

        GPIO.setmode(GPIO.BCM)                    #Set GPIO BCM numbering
        self.IN_SENSOR = 20                         #Associate pin 38 to TRIG
        GPIO.setup(self.IN_SENSOR,GPIO.IN)          #Set pin as GPIO in

    def caption(self):

        stp=False                                  #set the return to stop the car
        i=0                                       #Line counter
        print("Entrer un nombre de tour:")
        cpt=input()                                #Store the number of loop to do
        cpt=int(cpt)
        etatprecedent=False                       #Boolean to don't count the same line
        while True:                                #infinte loop

            B=GPIO.input(self.IN_SENSOR)                    #Safe the state of the IS
            if(B==True and etatprecedent!=B):               #condition
                i+=1                                        #Counter increment
                if(i>1):
                    print("détection de la ligne d'arrivée")    #Print line detection
                    print(i-1)                                  #Print counter

            if(i-1==cpt):
                stp=True
                avancer.stop()
                break

            etatprecedent=B                             #Safe the previously state
            time.sleep(0.1)                            #Delay of 1 second
        return stp                                    #Stop the car at the number set by the user
```

main.py

```python
class Motor(object):

    i2c = busio.I2C(board.SCL, board.SDA)
    hat = adafruit_pca9685.PCA9685(i2c)
    hat.frequency = 50

    M0_En = 23   # pin4
    M0_A  = 17  # pin11
    M0_B  = 18  # pin12

    M1_En = 24   # pin4
    M1_A  = 27  # pin13
    M1_B  = 22  # pin15

    def __init__(self):

        GPIO.setmode(GPIO.BCM) # GPIO Numbering
        GPIO.setup(self.M0_En, GPIO.OUT)
        GPIO.setup(self.M0_A,  GPIO.OUT)
        GPIO.setup(self.M0_B,  GPIO.OUT)
        GPIO.setup(self.M1_En, GPIO.OUT)
        GPIO.setup(self.M1_A,  GPIO.OUT)
        GPIO.setup(self.M1_B,  GPIO.OUT)
        self.motorL = GPIO.PWM(self.M0_En, 100)
        self.motorR = GPIO.PWM(self.M1_En, 100)

    """ Goes Forwards for 'x' seconds at speed 'speed' """
    def forwards(self, speed = 100):
        GPIO.output(self.M0_A,  GPIO.HIGH)
        GPIO.output(self.M0_B,  GPIO.LOW)
        GPIO.output(self.M0_En, GPIO.HIGH)
        GPIO.output(self.M1_A,  GPIO.HIGH)
        GPIO.output(self.M1_B,  GPIO.LOW)
        GPIO.output(self.M1_En, GPIO.HIGH)
        self.motorL.start(speed)    # TODO : Tester si 0 < speed <= 100 ?
        self.motorR.start(speed)
        print("Go !")
        #sleep(t)
        #self.stop()
        print("stop !")

    """ Goes Backwards for 'time' seconds at speed 'speed' """
    def backwards(self, speed = 100):
        GPIO.output(self.M0_A,  GPIO.LOW)
        GPIO.output(self.M0_B,  GPIO.HIGH)
        GPIO.output(self.M0_En, GPIO.HIGH)
        GPIO.output(self.M1_A,  GPIO.LOW)
        GPIO.output(self.M1_B,  GPIO.HIGH)
        GPIO.output(self.M1_En, GPIO.HIGH)
        self.motorL.start(speed)
        self.motorR.start(speed)
        print("Go !")
        #sleep(t)
        print("before stop !")
        #self.stop()
        print("stop !")


    """ Stops the car """
    def stop(self):
        print("stop")
        GPIO.output(self.M0_En, GPIO.LOW)
        GPIO.output(self.M1_En, GPIO.LOW)
```

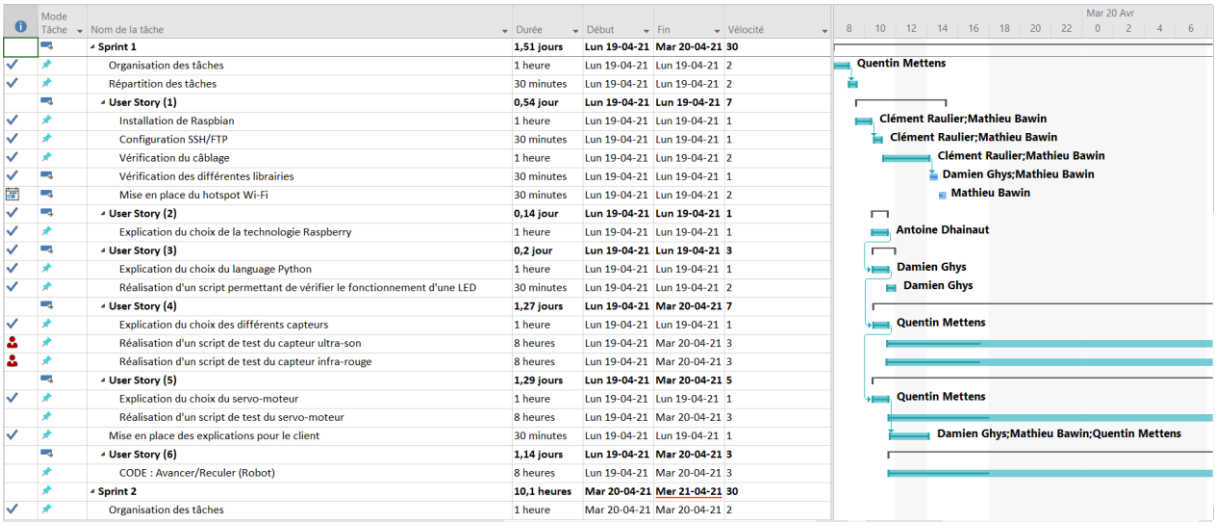
main.py

```
#
if __name__ == '__main__': # Program starts here

    avancer = Motor()                                   #objet moteur
    tourner = Servo()                                   #objet servo
    avancer.forwards(100)
    capteurinfra = InfraredSensor()                       #capteur infrarouge
    capteurGauche = CapteurUltraSons(9,11,100)
    capteurDevant = CapteurUltraSons(5,6,60)
    while True:
        distancegauche = capteurGauche.getDistance()
        distancedevant = capteurDevant.getDistance()
        capteurDevant.cleanup()
        capteurGauche.cleanup()
        #capteurinfra.caption()


        if distancedevant < 15:                  #capteur devant
            print ("faut tourner à droite")
            if distancegauche < 15:             #capteur gauche
                print ("Ecarte Mur Gauche")
                time.sleep(0.11)
                avancer.forwards(40)
                tourner.turn(130)

        if distancegauche > 70:
            print ("vers Mur Gauche")
            time.sleep(0.11)
            print(distancegauche)
            avancer.forwards(40)
            tourner.turn(85)
        elif distancegauche > 50:
            print ("vers Mur Gauche")
            time.sleep(0.11)
            avancer.forwards(30)
            tourner.turn(85)
        elif distancegauche > 36:
            print ("vers Mur Gauche")
            time.sleep(0.11)
            print (distancegauche)
            avancer.forwards(20)
            tourner.turn(88)
        elif distancegauche > 31:
            print ("vers Mur Gauche")
            avancer.forwards(20)
        elif distancegauche < 30:
            print ("Ecarte Mur Gauche")
            time.sleep(0.11)
            print (distancegauche)
            avancer.forwards(40)
            tourner.turn(130)
```

```
        else:
            avancer.forwards(40)
            tourner.straight()
```

8.2. Gestion du projet

- Diagramme de Gant



Mode Tâche	Nom de la tâche	Durée	Début	Fin	Vélocité
	Sprint 1	**1,51 jours**	**Lun 19-04-21**	**Mar 20-04-21**	**30**
✓	Organisation des tâches	1 heure	Lun 19-04-21	Lun 19-04-21	2
✓	Répartition des tâches	30 minutes	Lun 19-04-21	Lun 19-04-21	2
	User Story (1)	**0,54 jour**	**Lun 19-04-21**	**Lun 19-04-21**	**7**
✓	Installation de Raspbian	1 heure	Lun 19-04-21	Lun 19-04-21	1
✓	Configuration SSH/FTP	30 minutes	Lun 19-04-21	Lun 19-04-21	1
✓	Vérification du câblage	1 heure	Lun 19-04-21	Lun 19-04-21	2
✓	Vérification des différentes librairies	30 minutes	Lun 19-04-21	Lun 19-04-21	1
	Mise en place du hotspot Wi-Fi	30 minutes	Lun 19-04-21	Lun 19-04-21	2
✓	**User Story (2)**	**0,14 jour**	**Lun 19-04-21**	**Lun 19-04-21**	**1**
✓	Explication du choix de la technologie Raspberry	1 heure	Lun 19-04-21	Lun 19-04-21	1
✓	**User Story (3)**	**0,2 jour**	**Lun 19-04-21**	**Lun 19-04-21**	**3**
✓	Explication du choix du language Python	1 heure	Lun 19-04-21	Lun 19-04-21	1
✓	Réalisation d'un script permettant de vérifier le fonctionnement d'une LED	30 minutes	Lun 19-04-21	Lun 19-04-21	2
	User Story (4)	**1,27 jours**	**Lun 19-04-21**	**Mar 20-04-21**	**7**
✓	Explication du choix des différents capteurs	1 heure	Lun 19-04-21	Lun 19-04-21	1
	Réalisation d'un script de test du capteur ultra-son	8 heures	Lun 19-04-21	Mar 20-04-21	3
	Réalisation d'un script de test du capteur infra-rouge	8 heures	Lun 19-04-21	Mar 20-04-21	3
	User Story (5)	**1,29 jours**	**Lun 19-04-21**	**Mar 20-04-21**	**5**
✓	Explication du choix du servo-moteur	1 heure	Lun 19-04-21	Lun 19-04-21	1
	Réalisation d'un script de test du servo-moteur	8 heures	Lun 19-04-21	Mar 20-04-21	3
✓	Mise en place des explications pour le client	30 minutes	Lun 19-04-21	Lun 19-04-21	1
	User Story (6)	**1,14 jours**	**Lun 19-04-21**	**Mar 20-04-21**	**3**
	CODE : Avancer/Reculer (Robot)	8 heures	Lun 19-04-21	Mar 20-04-21	3
	Sprint 2	**10,1 heures**	**Mar 20-04-21**	**Mer 21-04-21**	**30**
✓	Organisation des tâches	1 heure	Mar 20-04-21	Mar 20-04-21	2

- Diagramme de vélocité

x/y	Vélocité th	Vélocté Ré
Sprint 1	30	24
Sprint 2	30	26
Sprint 3	33	18