



Projet Tutoré

***ROBOT D'EXPLORATION EN MILIEU
HOSTILE***

***BERLAND BASTIEN, DAVILLARS GREGORY,
THERON JOSHUA***

Enseignant réfèrent : Joaquin Jorge Rodriguez

Table des matières

Table des matières.....	2
Table des illustrations.....	3
Contexte.....	4
Qu'est-ce que la robotique ?	4
Quels sont les différents types de robot ?	4
Préambule du projet.....	5
Cahier des charges	6
Contexte du projet.....	6
Périmètre du projet	6
Aspects fonctionnels et techniques et analyse des besoins	6
Ressources	7
Histoire.....	7
Chaîne fonctionnelle et diagrammes SysML	8
Gantt	13
Choix des composants	14
Nomenclature.....	15
WIFIBOT	16
Description SUB-D9 mâle :	17
Commande du WifiBot avec ROS.....	20
Modélisation 3D	22
GPS.....	26
Cartographique	29
Partie Retour vidéo.....	31
Montage électronique.....	32
Partie programmation vision.....	33
Mise en commun et assemblage.....	37
Quelles optimisations possibles ?	38
Difficultés :	39

Avis personnel.....	40
Remerciements	41
Sources	42
Annexes	43
Annexe 2 : Fichier URDF	51
Annexe 3 : Schéma Wifibot.....	52
Annexe 4 : Programme Python	53

Table des illustrations

Figure 1: chaine d'information	9
Figure 2 : Diagramme des exigences.....	10
Figure 3 : Diagrammes des cas d'utilisation	11
Figure 4 : Diagramme de définition de blocs	12
Figure 5 : Gantt	13
Figure 6 : Wifibot	16
Figure 7 : RS232	17
Figure 8 : Description SUB-D9	17
Figure 9 : Simulation RVIZ	21
Figure 10 : Support v1	22
Figure 11 : Support v2.....	22
Figure 12 : Capot v2.....	22
Figure 13 : Capot v1	22
Figure 14 : Wifibot	23
Figure 15 : Limite	25
Figure 16 : GPS	26
Figure 17 : Raspberry pin.....	27
Figure 18 : Informations GPS.....	27
Figure 19 : Lidar	29
Figure 20 : SLAM	30
Figure 21 : Caméra raspberry	31
Figure 22 : Montage électronique.....	32
Figure 23 : Wifibot	37

Contexte

Qu'est-ce que la robotique ?

La robotique est un domaine d'activité recouvrant l'étude, la conception et la fabrication de robots ou machines automatisées. Elle implique des compétences en informatique, en électronique et en mécanique. De ce fait, pour être considérée comme un robot, une machine doit être pourvue de capteurs qui analysent l'environnement, mais également d'un système de traitement logique et d'actionneurs.

A partir des informations obtenues au sein de son environnement, le robot peut agir de manière autonome et intelligente en se basant sur des modèles d'apprentissage embarqués ou du Machine Learning.

Quels sont les différents types de robot ?

La robotique englobe un champ d'activité très large et diversifié. Dont voici les principaux et leurs champs d'application avec quelques exemples existants :

- **INDUSTRIE** : robot de gestion de chaîne d'assemblage
- **ARMEE** : drone, robot-espion, robot-mule
- **SECURITE** : vidéosurveillance
- **SANTE** : échographie, chirurgie assistée
- **AEROSPATIAL** : robot explorateur de la NASA
- **TRANSPORT** : voiture autonome
- **USAGE DOMESTIQUE** : robot aspirateur, robot tondeuse
- **ACCOMPAGNEMENT** : jouet automatisé, robot humanoïde
- **INFORMATIQUE** : chatbot, assistant vocal

Préambule du projet

Notre groupe de trois étudiants en 5ème année a cette année choisi pour projet d'étude la réalisation d'un robot d'exploration en milieu hostile. Un sujet complet qui nous permettra de renforcer nos compétences en mécanique, électronique et programmation.

Nous avons souhaité nous orienter sur un robot destiné à être utile dans de nombreux domaines. Notre robot d'exploration en milieu hostile pourrait être destiné à l'industrie, à l'armée ou à la sécurité, s'adaptant au besoin en changeant le préhenseur de son bras (robot manipulateur, robot démineur, etc.). Leur précision permet de limiter les risques.



Cahier des charges

Contexte du projet

Aujourd'hui, un nombre significatif d'interventions sont sujettes à de gros risques où une intervention humaine est nécessaire (guerre, accidents nucléaire, fuite de produit chimique, zones accidentées, etc.). C'est pourquoi, dans le cadre de notre projet robotique tutoré, nous avons souhaité nous orienter sur un robot destiné à être utile à de multiples fins. Notre robot d'exploration en milieu hostile pourrait être utile dans de nombreux domaines tels que l'industrie, l'armée ou la sécurité, s'adaptant au besoin en changeant le préhenseur de son bras (robot manipulateur, robot démineur, etc.). Sa précision et son contrôle à distance permet de limiter au maximum les risques.

Les objectifs du projet sont les suivants :

- ✓ Exploration de milieu à risques
- ✓ Limiter les dégâts matériels
- ✓ Remplacer l'humain pour des explorations à risques

Périmètre du projet

- Le robot sera limité à l'exploration terrestre non humide, l'isolation à des dégâts liquides n'est ici pas prise en compte
- Le projet sera limité au matériel disponible au plateau robotique et à l'iut ainsi que la possibilité d'imprimer des éléments en 3D
- Il devra être réalisé avant la date de rendu le 15 décembre

Aspects fonctionnels et techniques et analyse des besoins

Dans le but d'explorer des milieux à risques, le robot aura besoin de pouvoir se déplacer sur plusieurs axes. Son déplacement devra être adapté à plusieurs terrains (rocaillieux, plat, sableux...), que nous limiterons ici à humide. Afin de pouvoir visualiser à distance ce que filme le robot, il sera équipé d'une caméra intégrée avec vision nocturne pour un renvoi d'image en direct, ainsi que d'un capteur LiDAR permettant la création d'une cartographie. Cette caméra devra être capable de détecter l'objet souhaité à

distance. Il aura aussi besoin d'un bras robotisé pour interagir avec l'extérieur commandable et automatisé.

Nos besoins matériels sont les suivant :

- Un châssis avec sa batterie
- Un moteur brushless pour se déplacer
- Une caméra
- Un bras robotisé

Ressources

Ressources techniques

- Les ressources mise à notre disposition sont l'équipement robotique du plateau, L'accessibilité aux imprimantes 3D, un fer à souder ainsi que tout l'outillage de base.

Ressources intellectuelles

- Nous remercions tout le corps enseignant ainsi que les chercheurs du plateau qui ont bien voulu répondre à nos questions.

Délais

- Le projet devra être réaliser entre les dates du 4 septembre 2023 et du 15 décembre 2024.

Histoire

L'origine de ce projet remonte au sud de la France, où un groupe d'amis s'adonne depuis plusieurs années à l'exploration des nombreux affluents des rivières environnantes. Un jour, l'un d'entre eux découvrit l'entrée d'une caverne, suscitant ainsi la curiosité des jeunes aventuriers. Malheureusement, la grotte se révéla bien trop dangereuse pour une exploration humaine approfondie. Face à cette réalité, l'idée d'utiliser un robot télécommandé pour explorer ce tunnel accidenté émergea. Cependant, étant donné le coût élevé de ces robots sur le marché, la construction apparut comme la seule option viable. C'est ainsi que naquit l'idée de concevoir un robot d'exploration dans le but de parcourir des cavernes et d'autres lieux ensevelis.

Chaîne fonctionnelle et diagrammes SysML

➤ Chaîne fonctionnelle :

La chaîne fonctionnelle est utilisée dans la conception et surtout dans l'amélioration d'un système.

➤ La chaîne d'information :

La chaîne d'information pilote la chaîne d'énergie et comprend les trois fonctions techniques suivantes : acquérir, traiter et communiquer.

Les informations de nature logique ou analogique, circulent à travers le système afin d'obtenir des ordres d'exécution et des comptes rendus à destination des utilisateurs ou d'autres systèmes.

➤ La chaîne d'énergie :

La chaîne d'énergie comprend les quatre fonctions techniques suivantes : alimenter/stocker, distribuer, convertir et transmettre. Elle reçoit des ordres venant de la chaîne d'information.

L'énergie peut prendre plusieurs formes (mécanique, électrique, thermique, chimique...). Elle circule à travers le système afin de réaliser des actions sur la matière d'œuvre pour apporter la valeur ajoutée.

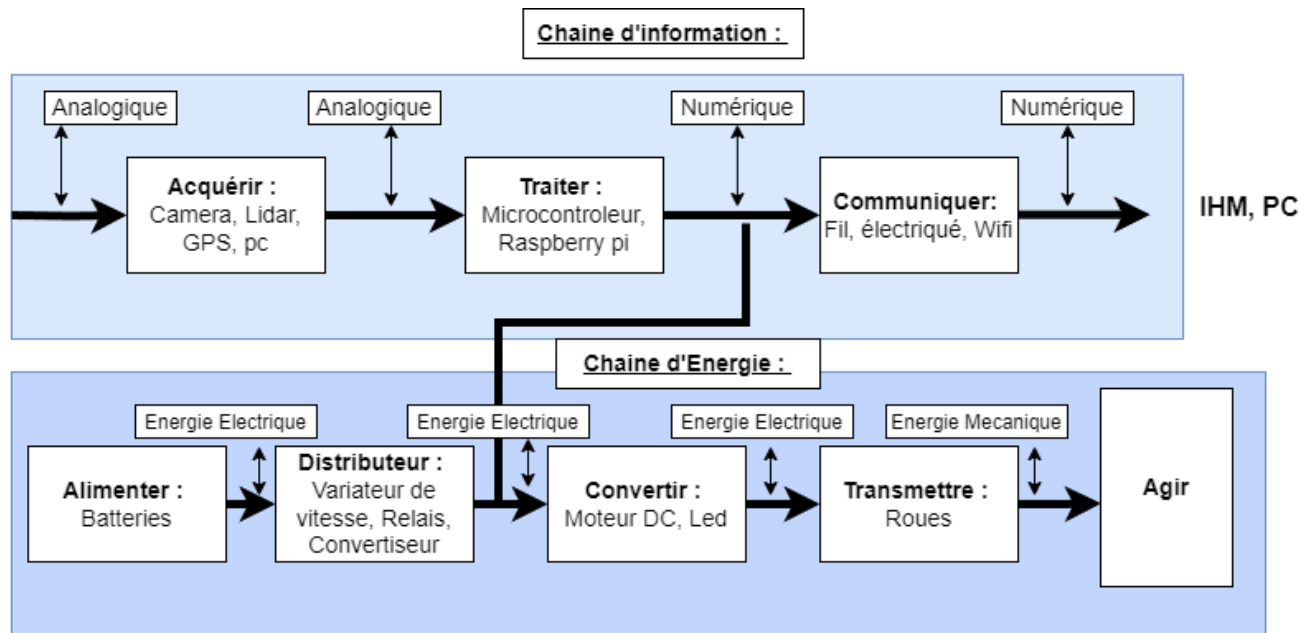


Figure 1: chaine d'information

Diagramme des exigences :

C'est un diagramme fonctionnel. Il décrit les exigences du cahier des charges fonctionnel. Une exigence exprime une capacité ou une contrainte à satisfaire par un système. Elle peut exprimer une fonction que devra réaliser le système ou une condition de performance technique, physique, de sécurité, de fiabilité, d'ergonomie ou d'esthétisme.

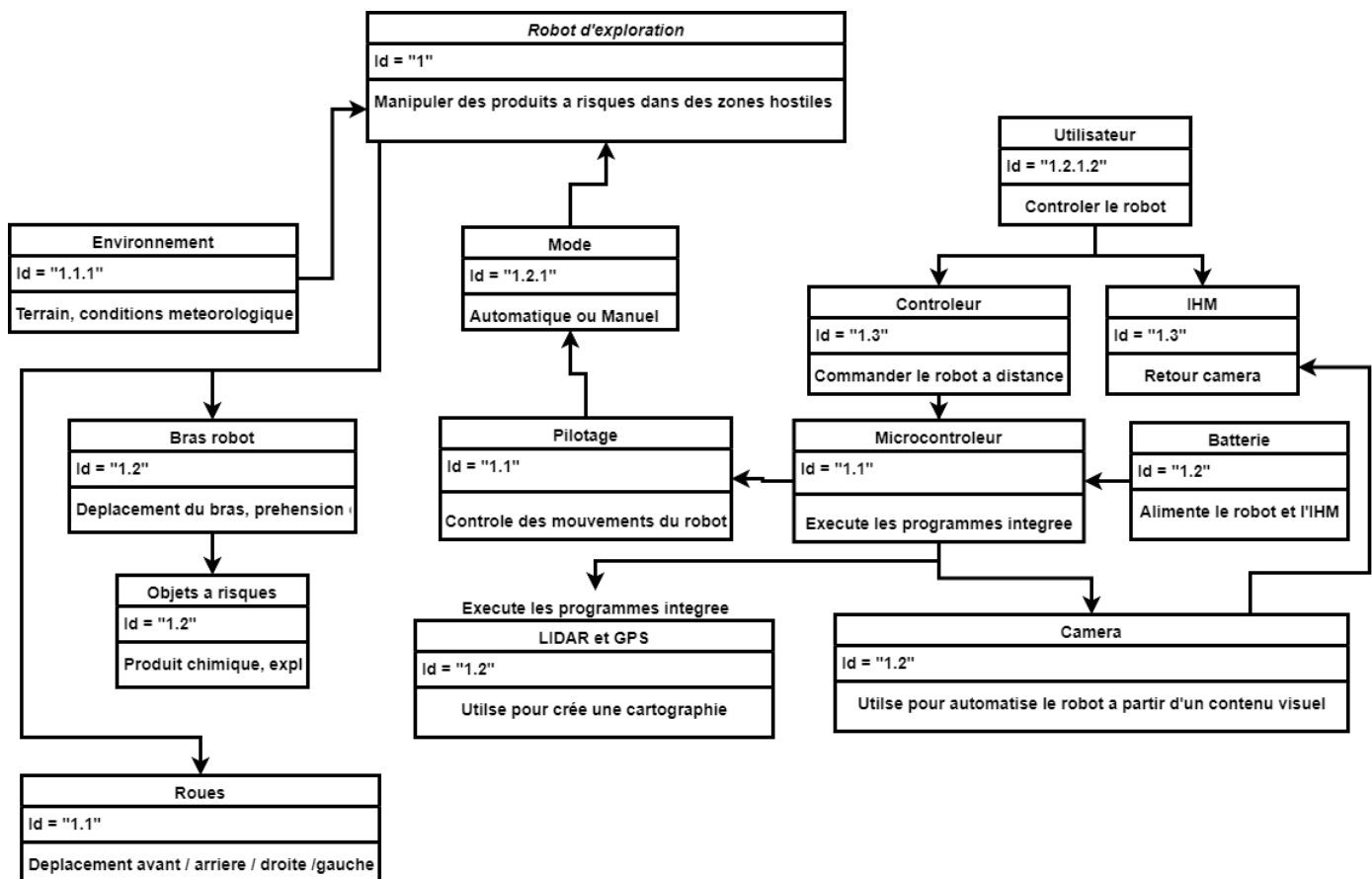


Figure 2 : Diagramme des exigences

Diagramme des cas d'utilisation :

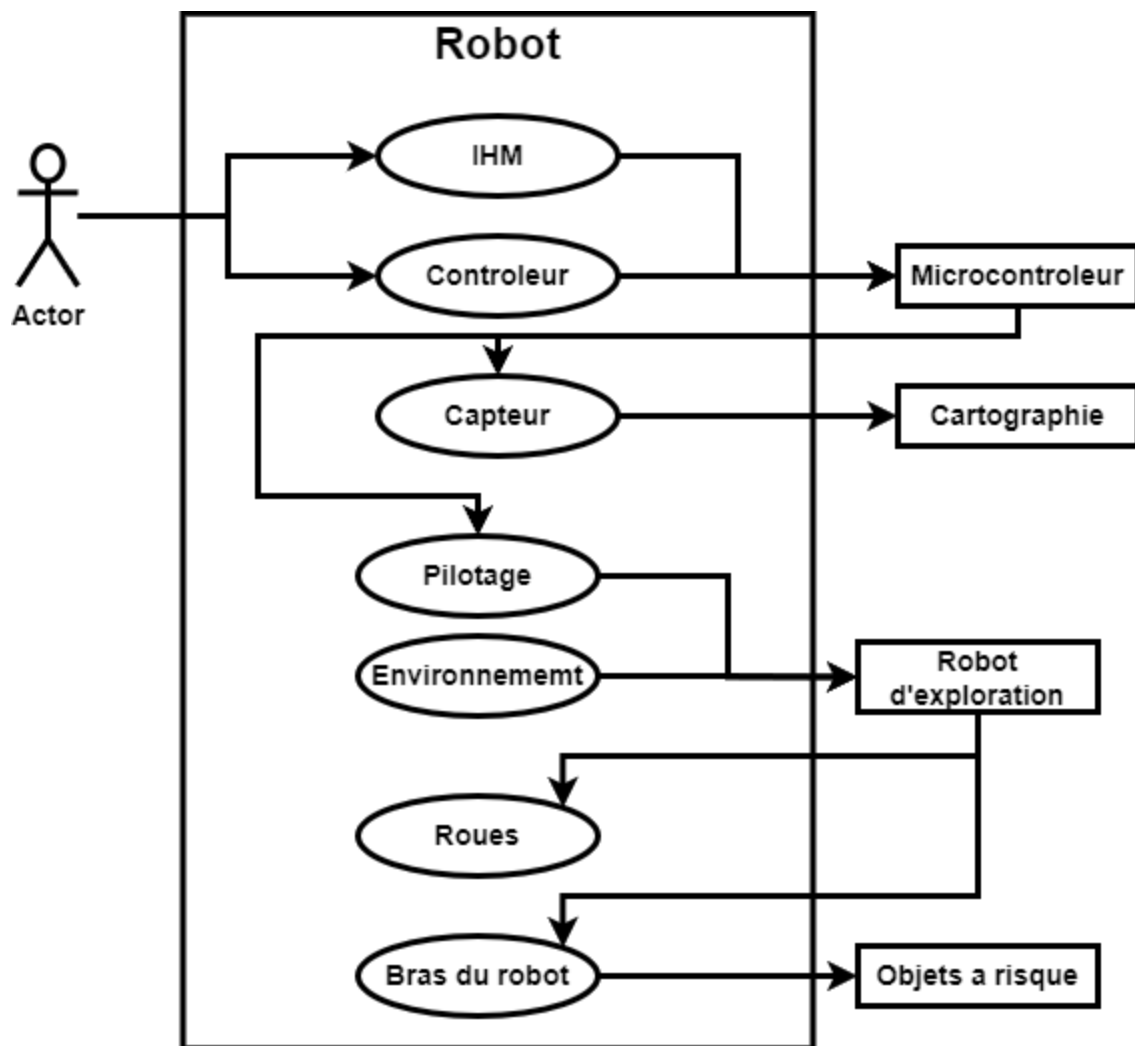


Figure 3 : Diagrammes des cas d'utilisation

C'est un diagramme fonctionnel. Il montre les interactions fonctionnelles des acteurs et du système d'étude. Il délimite précisément le système, décrit ce qu'il fera sans spécifier comment (et non ce que fera l'utilisateur). Il exprime les services offerts par le système aux utilisateurs (Actor).

Diagramme de définition de blocs :

C'est un diagramme statique. Il montre les briques statiques : blocs, composition, associations... Il est utilisé pour décrire l'architecture matérielle du système.

Un bloc est une entité bien délimitée qui encapsule principalement des attributs (variables d'état), des opérations (procédures comportementales), des contraintes, des ports (échange de flux avec l'extérieur) et des parts (sous-blocs internes).

Un bloc peut modéliser tout le système, un élément matériel ou logiciel.

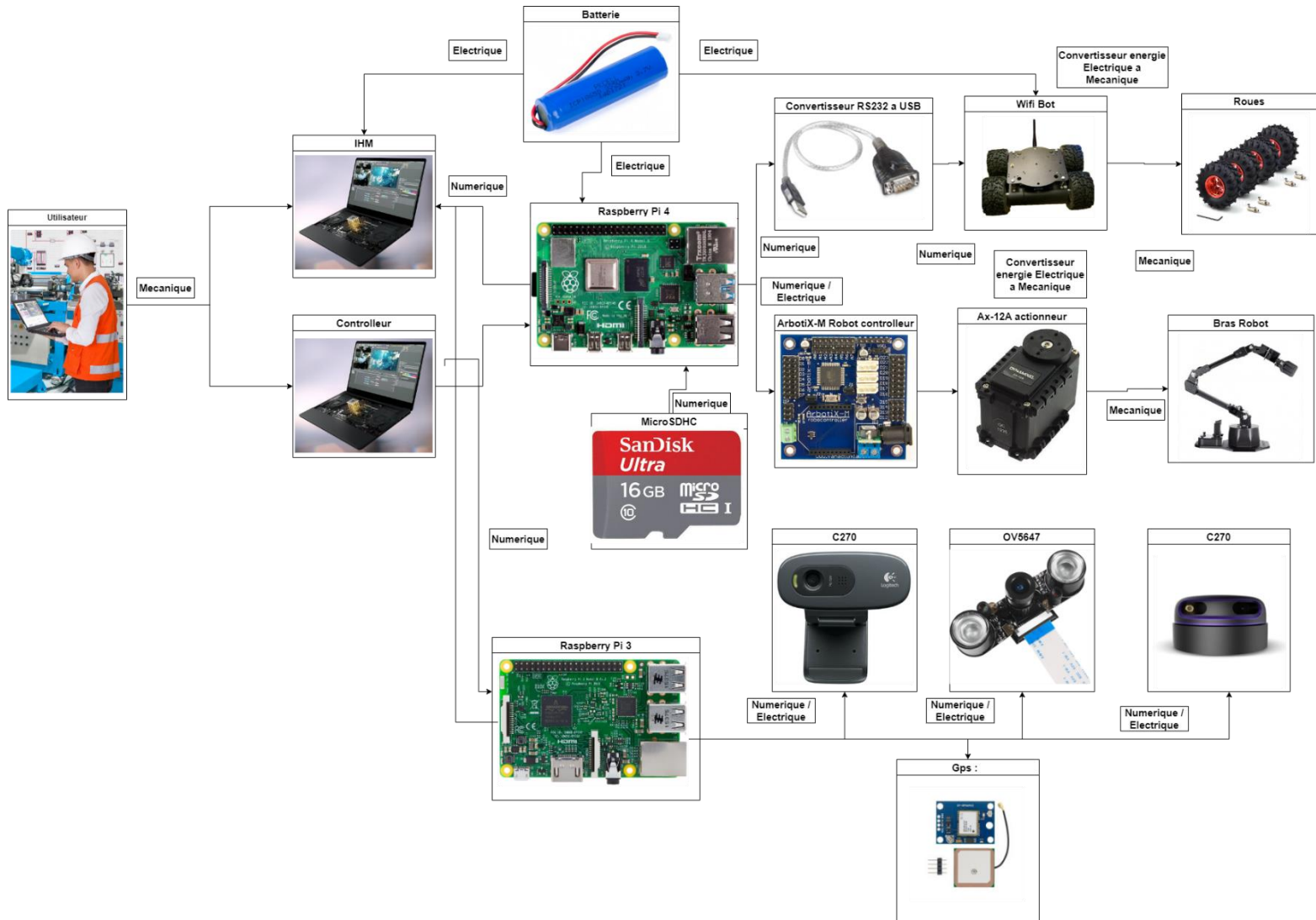


Figure 4 : Diagramme de définition de blocs

Gantt

C'est l'outil de planification par excellence. Il permet de planifier notre projet de la meilleure façon possible et de visualiser rapidement l'avancement des différentes tâches à réaliser. C'est aussi un bon outil de communication puisqu'il a permis à tous les membres de l'équipe de suivre le planning établi et de connaître l'avancée du projet en temps réel.

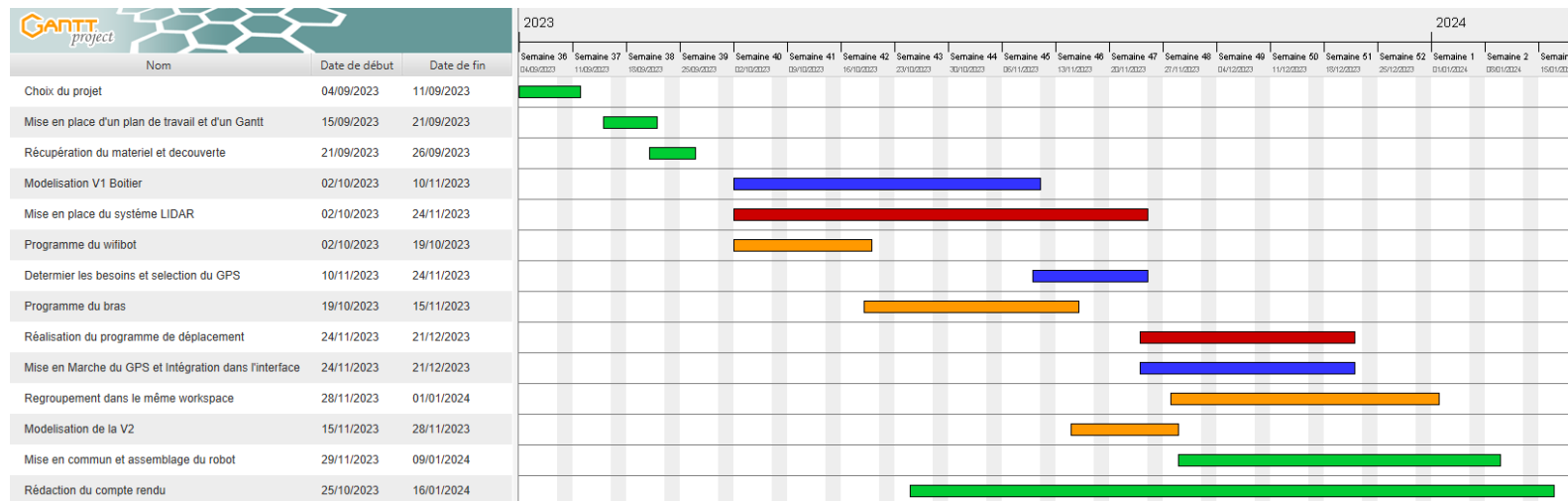


Figure 5 : Gantt

Choix des composants

Le robot présenté est une plateforme robotique avancée et polyvalente, intégrant divers composants et technologies pour une performance optimale dans des environnements variés.

1. **Raspberry Pi 4 et 3** : Ces deux cartes Raspberry Pi agissent comme des cartes mères embarquées, centralisant la commande des autres composants. Leur connectivité USB permet de gérer la communication et le retour d'informations entre l'utilisateur, la webcam, et le WifiBot.
2. **Réseau Wifi** : Un réseau wifi est mis en place pour permettre la commande à distance du robot.
3. **Stockage et Système d'Exploitation** : Le robot est équipé d'une carte SD de 16 Go et utilise l'OS Raspbian, bien que d'autres systèmes d'exploitation soient également compatibles.
4. **WifiBot Robotics** : Cette plateforme robotique Open Source est composée d'un châssis en acier, d'une batterie, de quatre moteurs avec contrôleurs Brushless, et d'une carte électronique. Elle fournit des informations sur la batterie, la communication, et les retours des contrôleurs.
5. **ArbotiX-M Robot Controller** : Ce contrôleur de robot polyvalent est lié au bras robotique et compatible avec les actionneurs Bioloid et Dynamixel. Il bénéficie d'une large communauté open source offrant de nombreuses bibliothèques et exemples.
6. **Bras Robotique** : Équipé de servomoteurs AX-12A, le bras peut suivre sa vitesse, température, tension, et charge. Un algorithme de contrôle ajuste la position de chaque servo pour maintenir la position de l'arbre.
7. **Système Visuel** : La webcam Logitech C270 HD offre une résolution de 720p et 30 ips pour une vision claire. Une caméra Night Vision OV5647 avec lumières infrarouges est utilisée pour les déplacements nocturnes.
8. **Géolocalisation et Cartographie** : Un module GPS Gy-NEO6MV2 assure la géolocalisation. Pour la cartographie, le robot utilise le capteur LiDAR RPLIDAR A3.
9. **Alimentation** : Un convertisseur DC-DC HiLetgo est utilisé pour adapter la tension de sortie de la batterie pour alimenter la Raspberry Pi et le bras articulé.

Ce système intégré permet au robot de naviguer et de fonctionner efficacement dans diverses conditions, tout en offrant une plateforme flexible pour des applications variées.

Nomenclature

<u>NATURE-NOM</u>	<u>NBR</u>	<u>DIMENSION</u> <u>LXWXH</u> <u>(MM)</u>	<u>VALEUR</u> <u>NOMINALE DE</u> <u>RÉFÉRENCE</u>	<u>TENSIONS</u>	<u>MARQUES</u>	<u>PRIX</u> <u>(€)</u>
RASPBERRY-PI 4	1	85 x 56 x 17	64-Bit	5.5V et 3A	Fondation Raspberry Pi	60
RASPBERRY-PI 3	1	85 x 56 x 17	64-Bit	5.5V et 3A	Fondation Raspberry Pi	35
WIFIBOT	1	320 x 370 x 150	Brushless powerful motors	4V x 12V	Nexter Robotics	2000
AX-12A ACTUATOR	5	32 x 50 x 40	Dynamixel servomotors	2V x 1.5A	Robotis	59.92
ARBOTIX-M ROBOCONTROLLER	1	61 x 61 x 10	10 more general purpose I/O pins	12V	Dynamixel	40
C270	1	72x31x66	HD Webcam	5V x 150 mA	Logitech	20
MICROSDHC	1	9x8	16 Giga	x	SanDisk	10
CONVERTISEUR	3	10.6 x 8.6 x 3.5	dc -dc	1.5 V à 35 V	Hiletgo	13
RPLIDAR A3	1	80 x 80 x 45	10 - 25 m	5v x a3	Slamectec	800
OV5647	1	25 x 24 x 3.5	5 MP	5v x a3	WAVESHARE	25
GY-NEO6MV2	1	25x35	9600bauds/s	5v	ICQUANZX	10

WIFIBOT

Le wifi bot est un robot d'exploration open source pour la recherche ou le développement. Il est fabriqué par l'entreprise de robotique française Nexter Robotics qui le propose à un prix qui varie en fonction des options de 2000 à 3000 euros.

Il est divisé en deux parties distinctes : la plateforme mobile et la carte de commande.

La carte de commande peut varier en fonction des modèles mais les plus courantes sont les Raspberry Pi 3 et 4, le module Jetson Nano/TX2 ou bien encore Xavier AGX/NX. Néanmoins, cette carte de commande est permissive et adaptable car elle permet l'ajout de toutes sortes de modules extérieurs, comme des caméras, des micros, une antenne wifi ou bien une grande variété de capteurs.

De son côté, la partie plateforme mobile est-elle composée :

- D'un châssis en acier avec un dimension de L : 32cm, l : 37 cm, : 15 cm pour un poids de 3.8 kilos
- D'une batterie de 12.8V LIFEP04 avec une capacité de 10AH.
- De quatre moteurs Brushless de 12v, un train épicycloïdal de 26:1 et 156 rpm
- D'un encodeur de 336 signaux par tour.
- D'une carte électronique permettant la gestion de la batterie, des capteurs, des contrôleurs et des moteurs.
- D'un convertisseur USB/RS232 équipé sur la carte électronique.

1. Carte électronique

2. Batterie

3. Moteur

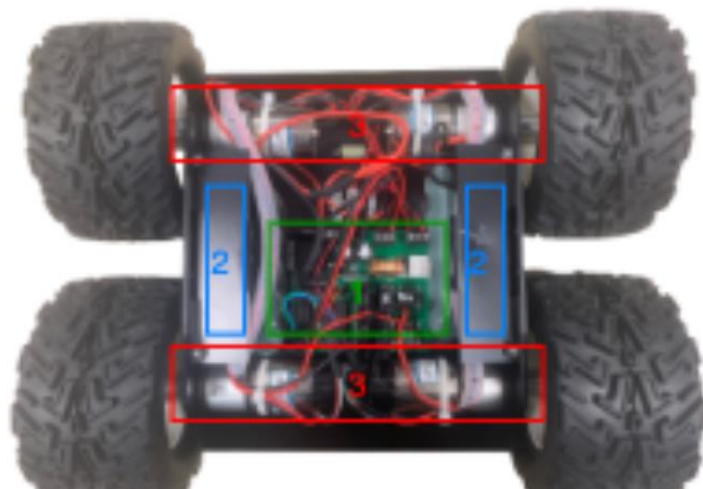


Figure 6 : Wifibot

Le protocole RS232 est un protocole de communication standardisé permettant la communication entre deux appareils informatiques. Plus souvent appelé port série ou Port Com. Daté de 1962, standardisé par l'EIA (Electronic Industries Alliance), est un protocole en voie de disparition depuis l'apparition de l'Usb. Le connecteur le plus connu est le SUB-D 9, qui communique entre 75 à 128000 Baudes (bits par seconde). Dans notre cas, le baudrate idéal est situé à 19200, ce protocole nous est donc largement suffisant.

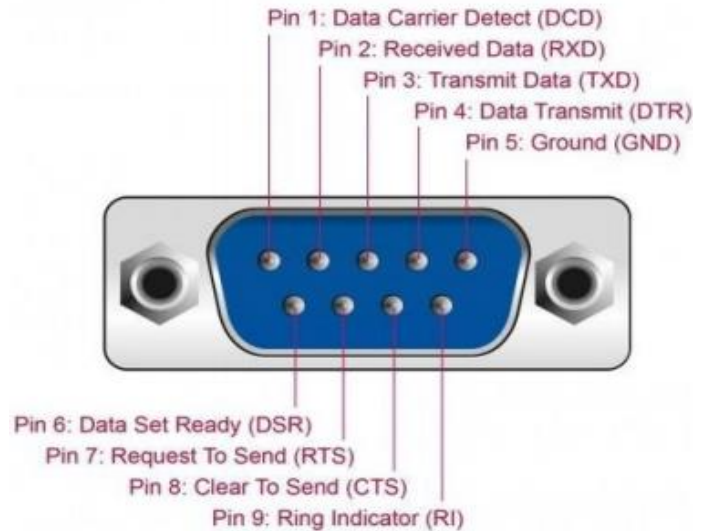


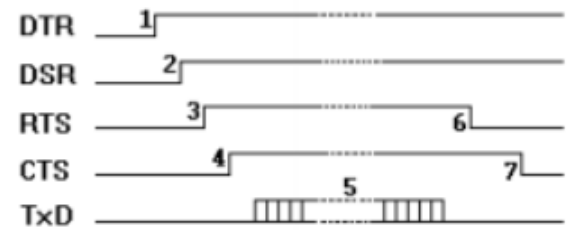
Figure 7 : RS232

Description SUB-D9 mâle :

N° broche	Fonction et orientations des signaux. (Cible, Envoyeur)		
1	DCD	<----	Détecte le départ de la transmission
2	RxD (ou RD)	<----	Récupéré les données envoyer par la cible
3	TxD (ou TD)	---->	Envoie des données par l'envoyeur
4	DTR/	---->	Confirmation de l'état du terminal/ordinateur
5	SG, GND	----	La masse (GND)
6	DSR/	<----	Données prêtes à l'envoi
7	RTS/	---->	Demande de l'envoi
8	CTS/	<----	Prêt à l'envoi
9	RI	<----	Sonnerie

Figure 8 : Description SUB-D9

- 1 DTR L'ordinateur d'envoi indique qu'il peut envoyer
- 2 DSR Il indique que les données sont prêtes à être envoyé
- 3 RTS Il va demander l'autorisation à la cible si il peut émettre
- 4 CTS La cible informe qu'il est prêt
- 5 TxD L'envoi des données per l'envoyeur
- 6 RTS L'envoyeur annule sa demande démission
- 7 CTS L'envoyeur repasse en attente



Basic Communication :

Pour avancer le robot nécessite un envoi d'un signal qui est une suite de caractères codés en char.

Le premier char est composé du nombre : « 255 ». Le deuxième char est la taille, dans le cas de Wifibot, le plus souvent c'est 0x07 pour la décimal 7.

Pour le troisième et quatrième, c'est la vitesse de rotation de gauche que l'on fait varier de 0 -> 240 « tics » qui sont une unité définie dans la doc.

Pour le cinquième et sixième, c'est la vitesse de rotation de droite que l'on fait varier de 0 -> 240 « tics ».

Le septième est l'orientation de rotation des roues et le ON/OFF des roues.

Le char est décomposé de la manière suivante (1 bytes char -> 8 bits)

(128) Bit 7 Contrôle de vitesse en boucle fermée côté gauche : 1 -> ON / 0 -> OFF

(64) Bit 6 Signal Indicateur de vitesse avant/arrière côté gauche : 1 -> avant / 0 -> arrière (32) Bit 5

Contrôle de vitesse en boucle fermée côté droite : 1 -> ON / 0 -> OFF

(16) Bit 6 Signal Indicateur de vitesse avant/arrière côté droite : 1 -> avant / 0 -> arrière

(8) Bit 3 Relais 4 On/Off (D'SUB 15 POWER Pin 13 et 14)

(4) Bit 2 Relais 3 On/Off (D'SUB 15 POWER Pin 11 et 12)

(2) Bit 1 Relais 2 On/Off (D'SUB 15 POWER Pin 4 et 5)

(1) Bit 0 Relais 1 pour capteur. On/Off : 0 est OFF 1 est ON (DSUB15 POWER Pin 3)

Les huitième et neuvième Char sont le CRC 16 bits.

Par exemple, pour contrôler le côté gauche et le côté droit (Gauche & Droite 2 moteurs, 2 encodeurs) : La vitesse est comprise entre 0-240. Si nous voulons que le côté gauche et le côté droit se déplacent à la vitesse 120 vers l'avant sans contrôle moteur, nous envoyons :

Char 1 : 255

Char 2 : 0x07

Char 3 : 120

Char 4 : 00

Char 5 : 120

Char 6 : 0

Char 7 : 80 (0 + 64 + 0 + 16)

Char 8 – Char 9 : CRC16(data)

Commande du WifiBot avec ROS

Introduction

Ce rapport présente le développement de la commande d'un rover à quatre roues, nommé Wifibot, et d'un bras robotique Phantom Xpincher, réalisé à l'aide du système d'exploitation robotique (ROS). L'objectif était de mettre en place des commandes efficaces et adaptables pour ces dispositifs robotiques.

Commande du Wifibot

La commande du Wifibot s'est basée sur l'adaptation d'un code existant. En exploitant le travail initial réalisé par Joaquim, des modifications ont été apportées à certains fichiers pour les ajuster aux spécificités du projet. Ces modifications ont permis d'obtenir un contrôle optimal du rover, garantissant ainsi sa performance et sa réactivité.

Développement du Package ROS pour le Bras Robotique

Dans le cadre du développement du bras robotique Phantom Xpincher, la création d'un package ROS spécifique a été une étape cruciale. Ce package a été enrichi par l'intégration d'une modélisation 3D du projet, permettant ainsi une simulation précise et réaliste dans les environnements RVIZ et Gazebo.

Une part importante de ce travail a impliqué la conception et la création de fichiers URDF (Unified Robot Description Format) (Annexe 2). Ces fichiers sont essentiels pour définir la structure physique et les propriétés mécaniques du bras robotique. Ils contiennent des informations détaillées sur les différentes pièces du robot, telles que les liens et les articulations, ainsi que leurs relations et leurs contraintes de mouvement.

La création des fichiers URDF a nécessité une compréhension approfondie à la fois de la structure physique du bras robotique et de la manière dont ces éléments interagissent dans un espace tridimensionnel. En utilisant URDF, une représentation fidèle et fonctionnelle du bras a été construite, ce qui a permis des simulations précises et un contrôle efficace dans les environnements de simulation RVIZ et Gazebo.

Grâce à l'emploi de ces fichiers URDF, la simulation et le contrôle du bras robotique sous ROS ont atteint un niveau de précision et de fiabilité élevé. Ceci a été crucial pour tester divers scénarios et mouvements avant la mise en œuvre réelle, réduisant ainsi les risques d'erreurs ou de dysfonctionnements dans l'environnement physique.

Intégration avec MoveIt

L'intégration de MoveIt, un outil de planification de mouvement pour ROS, a été un élément clé pour le contrôle du bras robotique. L'utilisation de MoveIt a rendu possible la mise en place de commandes précises pour le bras, assurant ainsi une manipulation sûre et précise, essentielle pour les opérations prévues.

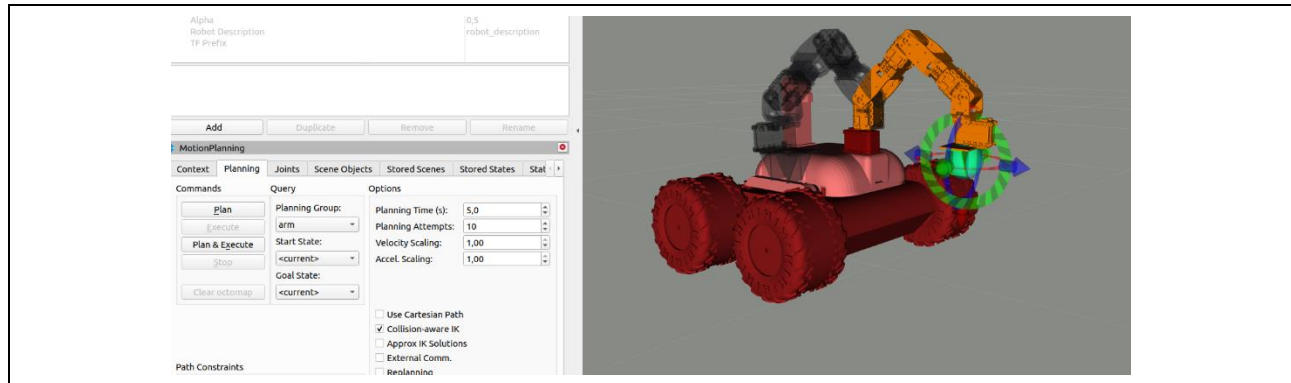


Figure 9 : Simulation RVIZ

Mise en route

Pour la mise en route, un script Bash est configuré pour s'exécuter automatiquement au démarrage de la Raspberry Pi. Ce script est responsable du lancement de ROS ainsi que des pilotes nécessaires pour le WifiBot et le bras robotique. Cette configuration assure un démarrage fluide et une initialisation automatique des composants essentiels du robot.

Conclusion

La mise en œuvre de la commande du Wifibot et du bras robotique Phantom Xpincher, intégrant l'utilisation avancée de ROS et la création minutieuse de fichiers URDF, démontre une application efficace de la robotique avancée. Cette approche a non seulement permis de résoudre des défis techniques complexes, mais a également ouvert la voie à une meilleure compréhension des possibilités offertes par la robotique moderne et ses applications dans des projets multidisciplinaires.

Modélisation 3D

Afin d'optimiser la réalisation du robot, de nombreuses parties ont été conçues en 3D avant la phase de conception physique. Trois versions ont été développées pour chaque composant : la première en tant que prototype, la deuxième en tant que version fonctionnelle permettant des tests et du prototypage, et enfin la version finale, à la fois esthétique et fonctionnelle.

La première composante est le socle, qui se fixera sur le WifiBot, abritant les emplacements pour les cartes électroniques et renforçant certaines parties du bras robotique. Ci-dessous, vous trouverez la version 2 du support de carte ainsi que la nouvelle version de ce dernier :

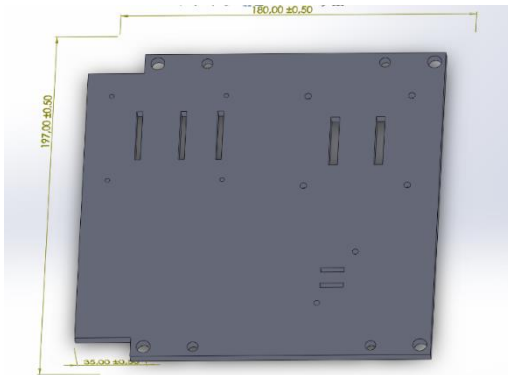


Figure 10 : Support v1

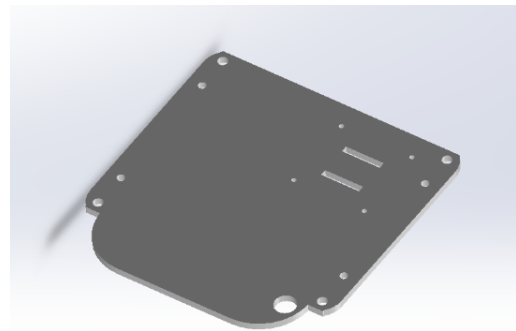


Figure 11 : Support v2

La seconde pièce est le capot du robot. La V2 se caractérise par sa simplicité et sa forme carrée et haute, conçue pour accueillir tous les composants à l'intérieur afin de les protéger de l'environnement extérieur. En revanche, la V3 est plus basse, plus arrondie et ergonomique, offrant une esthétique améliorée. Voici les deux versions des capots :

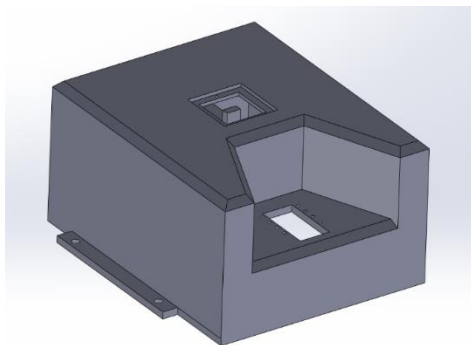


Figure 13 : Capot v1

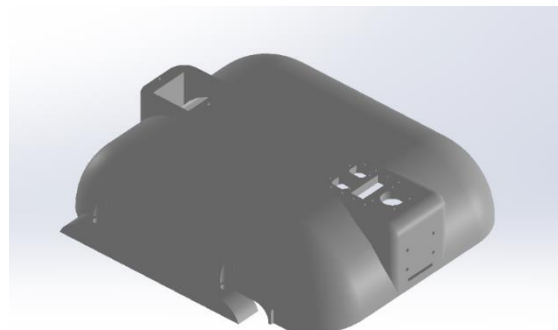


Figure 12 : Capot v2

Enfin, la dernière partie est la tour destinée à accueillir la caméra et le capteur Lidar, permettant la cartographie de l'environnement où se trouve le robot.

Une fois assemblée, la structure du robot présente l'aspect suivant :

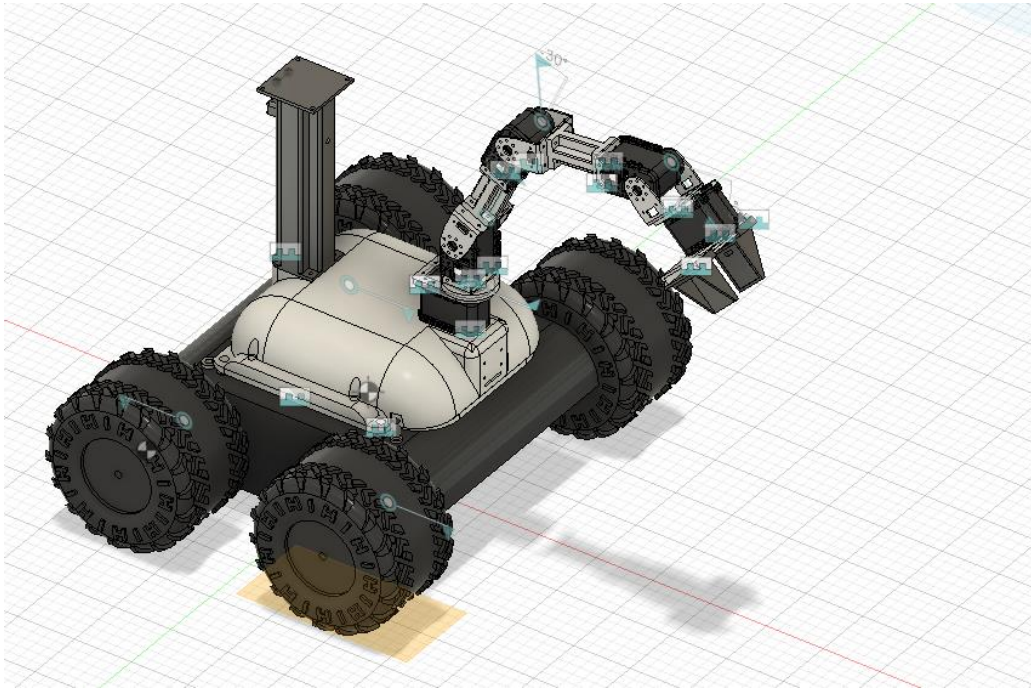
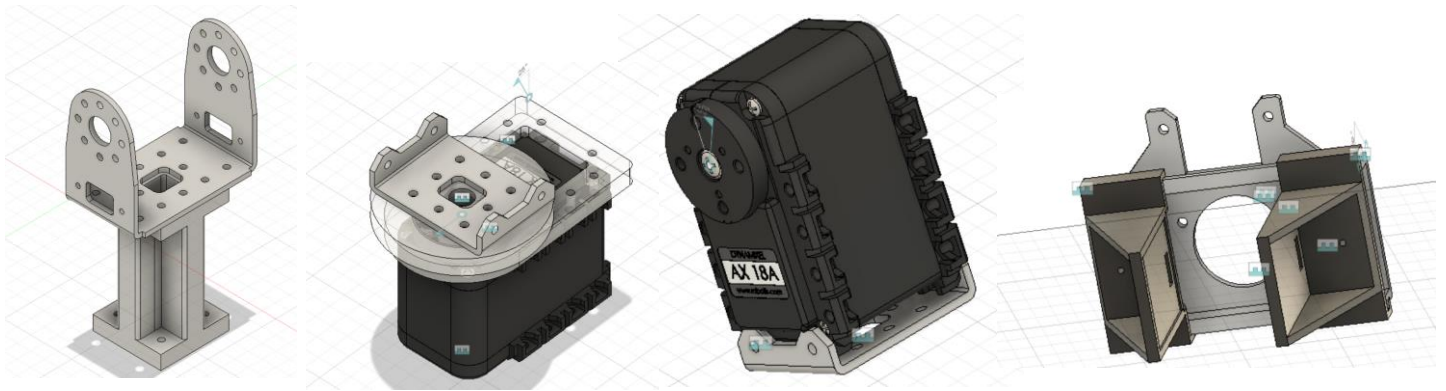


Figure 14 : WifiBot

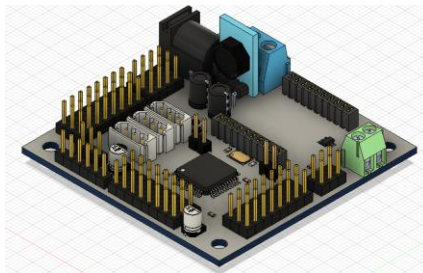
Pour mener à bien le projet et accomplir les tâches qui lui sont assignées, il était impératif de concevoir un bras robotique adaptable au WifiBot.

La première étape a consisté à déterminer le type de manipulateur pouvant être utilisé pour le projet. Celui-ci devait simplement être capable de soulever une petite charge, compte tenu de son stade de prototype.

Il était donc essentiel de s'inspirer de ce qui était réalisé dans le monde industriel, en particulier en ce qui concerne les bras manipulateurs utilisés sur les "rovers" professionnels. La conclusion a été qu'un manipulateur à 5 axes serait suffisant pour le projet.



Ainsi, le manipulateur à 5 axes a été conçu sur Fusion 360, incluant la modélisation des segments, des servomoteurs, de leurs supports, ainsi que du préhenseur permettant de saisir les pièces.



Pour obtenir une vision plus précise du robot dans le monde réel, avec l'emplacement des composants et les contraintes de mouvement du manipulateur, tous les éléments ont été inclus dans notre modélisation. Cette approche nous a permis de définir les limites angulaires de notre manipulateur.

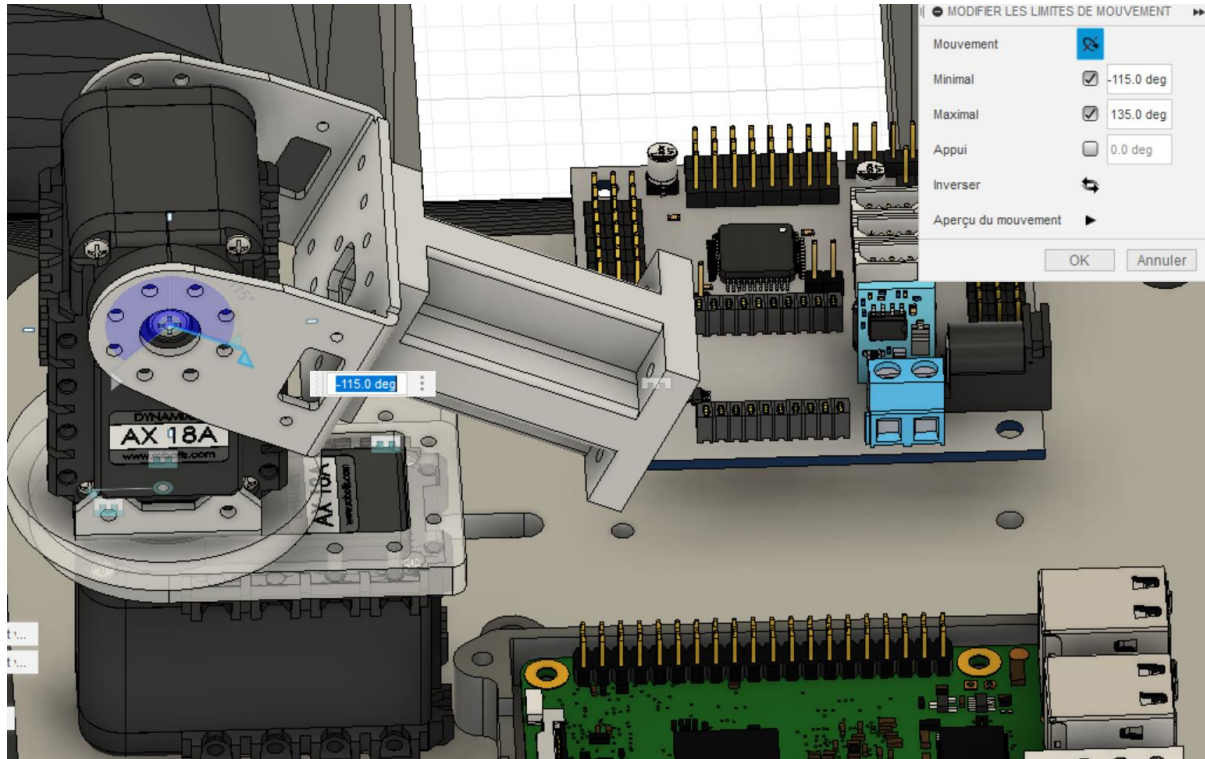


Figure 15 : Limite

GPS

Dans le cadre de ce projet visant à naviguer en environnement hostile et à être contrôlé à distance, il est probable que la visibilité directe soit compromise. Pour pallier cette difficulté, un module GPS est proposé comme solution.

Principe de Fonctionnement : Le Global Positioning System (GPS), ou « Système mondial de positionnement » en français, est un système de localisation par satellites. Il fonctionne en calculant la distance entre un récepteur GPS et plusieurs satellites. Les satellites transmettent régulièrement les informations nécessaires pour déterminer leur position, permettant ainsi au récepteur de connaître ses coordonnées géographiques précises. Cette détermination s'effectue par le biais de la trilatération, utilisant les informations reçues pour calculer la position exacte de l'utilisateur.

Concernant l'illustration des satellites, les points noirs représentent ceux actifs dans la localisation du GPS, tandis que les points rouges indiquent des satellites inactifs pour ce système.

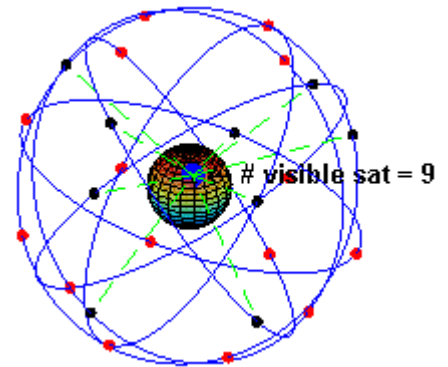


Figure 16 : GPS

Après nos recherches, nous avons sélectionné le module GPS GY-NEO6MV2 pour la géolocalisation. Il est compatible avec les cartes Raspberry Pi et est proposé à un coût abordable de 10 euros.

Quelques spécifications techniques :

- Taille Module : 25mm x 25mm
- Taille Antenne : 36mm x 26mm
- Alimentation : 3.3V DC - 5V DC
- Liaison série : 9600bps
- Sensibilité antenne : -161 dBm

Branchement :

VCC à la broche 1, qui est 3.3v

RX à la broche 10, qui est RX (GPIO15)

TX à la broche 8, qui est TX (GPIO14)

Gnd à la broche 6, qui est Gnd.

Une fois les connexions effectuées voici comment lancer le GPS et l'utiliser. Par défaut, le Raspberry Pi utilise l'UART comme console série. Il faut désactiver cette fonctionnalité afin de pouvoir utiliser l'UART pour notre propre application. Ouvrez une session de terminal sur le Raspberry Pi. La première chose à faire est de sauvegarder le fichier `cmdline.txt` avant de le modifier.

`sudo cp /boot/cmdline.txt /boot/cmdline_backup.txt` et appuyez sur Entrée.

Ensuite, éditez `cmdline.txt` et supprimez l'interface série.

Tapez `sudo nano /boot/cmdline.txt` et appuyez sur Entrée.

Supprimez `console=ttyAMA0,115200` et sauvegardez le fichier en appuyant sur `Ctrl X`, `Y` et `Enter`.

Tapez maintenant `sudo nano /etc/inittab` et appuyez sur Entrée.

Trouvez `ttyAMA0` en appuyant sur `Ctrl W` et en tapant `ttyAMA0` sur la ligne de recherche.

Lorsqu'il trouve cette ligne, appuyez sur `home`, insérez un symbole `#` pour commenter cette ligne, et appuyez sur `Ctrl X`, `Y`, `Enter` pour sauvegarder.

Tapez `sudo reboot` et appuyez sur Entrée pour redémarrer le Pi.

Raspberry Pi B+ J8 Header





















Pin#	NAME		NAME	Pin#	
01	3.3v DC Power		DC Power 5v	02	
03	GPIO02 (SDA1, I2C)		DC Power 5v	04	
05	GPIO03 (SCL1, I2C)		Ground	06	
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08	
09	Ground		(RXD0) GPIO15	10	
11	GPIO17 (GPIO_GEN0)		BITCLOCK GEN1 GPIO18	12	
13	GPIO27 (GPIO_GEN2)		Ground	14	
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16	
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18	
19	GPIO10 (SPI_MOSI)		Ground	20	
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22	
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24	
25	Ground		(SPI_CE1_N) GPIO07	26	
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28	
29	GPIO05		Ground	30	
31	GPIO06		GPIO12	32	
33	GPIO13		Ground	34	
35	GPIO19 LRCLOCK		GPIO16	36	
37	GPIO26		DATA IN	GPIO20	38
39	Ground		DATA OUT	GPIO21	40

Figure 17 : Raspberry pin

Time:	2015-07-26T16:14:01.000Z	PRN:	Elev:	Azim:	SNR:	Used:
Latitude:	33.458175 N	1	41	146	26	Y
Longitude:	79.566635 W	4	47	108	28	Y
Altitude:	-41.0 ft	7	86	228	08	Y
Speed:	0.5 mph	8	13	047	32	Y
Heading:	0.0 deg (true)	9	17	205	21	Y
Climb:	0.0 ft/min	11	63	138	21	Y
Status:	3D FIX (537 secs)	13	07	321	00	N
Longitude Err:	+/- 65 ft	16	00	089	00	N
Latitude Err:	+/- 53 ft	17	08	231	18	N
Altitude Err:	+/- 108 ft	19	44	040	21	N
Course Err:	n/a	27	13	046	29	N
Speed Err:	+/- 89 mph	28	29	296	11	N
Time offset:	0.575	30	55	314	00	N
Grid Square:	FM03fk					

Figure 18 : Informations GPS

Tester le GPS en utilisant quelques programmes standard. Ouvrez une session de terminal et tapez `sudo apt-get install gpsd gpsd-clients` et appuyez sur Entrée. Post l'installation, il faut démaré sur le port série :

Tapez `stty -F /dev/ttyAMA0 9600` et appuyez sur Entrée.

Démarrez maintenant GPSD :

Tapez `sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock` et appuyez sur Entrée.

Affichez maintenant en tapant `cgps -s` et appuyez sur Entrée.

Cartographie

Le RPLIDAR est un capteur LIDAR peu coûteux adapté aux applications de SLAM robotique en intérieur. Il offre un champ de balayage à 360 degrés, une fréquence de rotation de 5,5 Hz/10 Hz avec une distance garantie de 25 mètres pour le modèle A3.

Grâce au moteur de traitement d'image à haute vitesse conçu par RoboPeak, les coûts globaux sont considérablement réduits, faisant du RPLIDAR le capteur idéal dans des domaines sensibles aux coûts tels que les robots grand public et les amateurs de matériel.

Le RPLIDAR A3 effectue une mesure de distance à grande vitesse avec plus de 16 000 échantillons par seconde. Pour un balayage nécessitant 360 échantillons par rotation, une fréquence de balayage de 10 Hz peut être atteinte. Les utilisateurs peuvent personnaliser la fréquence de balayage de 2 Hz à 10 Hz librement en contrôlant la vitesse du moteur de balayage. Le RPLIDAR s'auto-adapte à la vitesse de balayage actuelle

Le RPLIDAR A3 effectue une mesure de distance à grande vitesse avec plus de 16 000 échantillons par seconde. Le RPLIDAR s'auto-adapte à la vitesse de balayage actuelle.

La création d'une carte à l'aide d'Hector SLAM sous ROS 2 avec un capteur LIDAR A3 offre une solution robuste et efficace pour la cartographie en temps réel dans des environnements robotiques. ROS 2, successeur de ROS (Robot Operating System), introduit des améliorations significatives en termes de modularité, de performance et de prise en charge des nouvelles fonctionnalités matérielles. Hector SLAM, quant à lui, est un algorithme SLAM (Simultaneous Localization and Mapping) qui permet au robot de se localiser et de cartographier son environnement simultanément.



Figure 19 : Lidar

Pour démarrer le programme : Exécutez le système en utilisant la commande de lancement appropriée. Assurez-vous que le LIDAR A3 est alimenté, et le robot est libre de se déplacer dans l'environnement que vous souhaitez cartographier.

“ros2 launch hector_slam_launch mapping_launch.py”

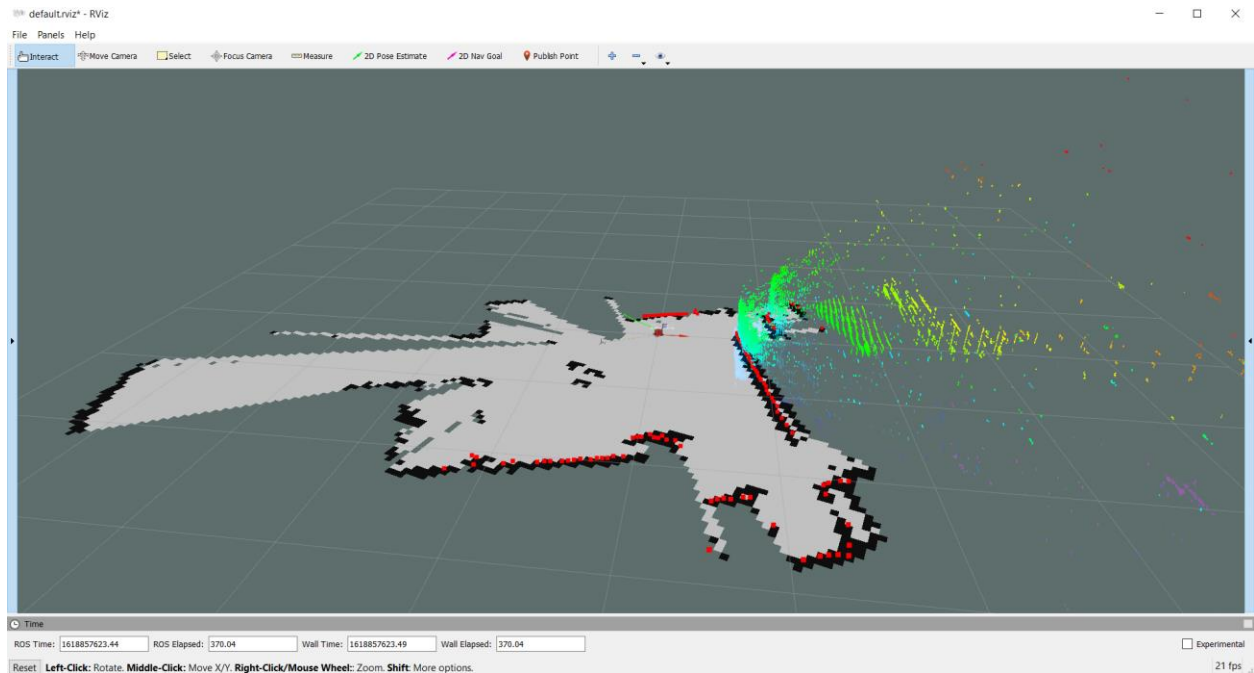


Figure 20 : SLAM

Pendant que le robot se déplace, le LIDAR A3 collecte en continu des données de distance. Hector SLAM utilise ces données pour estimer la pose du robot et créer une carte en temps réel. La carte résultante est mise à jour à mesure que le robot explore de nouveaux secteurs de l'environnement.

Des outils de visualisation ROS 2 tel que Rviz sont utilisés pour visualiser la carte en temps réel.

Le programme est un mixte du programme python permettant de faire tourner le Capteur lidar ainsi que du programme GitHub Hector Slam.

Partie Retour vidéo

L'utilisation d'une caméra Raspberry Pi avec vision nocturne offre des fonctionnalités étendues pour la capture d'images dans des conditions de faible luminosité. En outre, l'intégration d'une deuxième caméra permettant de générer deux flux vidéo en temps réel pour une diffusion sur une page web offre une solution polyvalente pour la surveillance, la sécurité ou d'autres applications nécessitant une surveillance à distance.

La caméra Raspberry Pi équipée de vision nocturne utilise généralement des capteurs infrarouges pour capturer des images même dans l'obscurité.

Ces capteurs activent automatiquement des LED infrarouges pour éclairer la scène, créant ainsi des images claires et détaillées même dans des conditions de faible éclairage. Cette capacité s'avère précieuse dans divers contextes, tels que la surveillance nocturne ou la capture d'images dans des endroits peu éclairés.

Pour créer un système avec deux flux vidéo en direct, vous pouvez utiliser une deuxième caméra Raspberry Pi standard. Ces caméras peuvent être configurées pour générer des flux vidéo distincts et être connectées à la Raspberry Pi via les ports dédiés. Il a été utilisé le logiciel tel que Motion qui permet gérer la diffusion des flux vidéo vers une page web.

Une première caméra Raspberry Pi à un port USB et la deuxième utilise le port dédié pour une caméra CSI (Camera Serial Interface) sur la Raspberry Pi.

Une fois connecté à la Raspberry Pi, les deux flux des caméras sont récupérés sur Motion et transférés dans une page web.

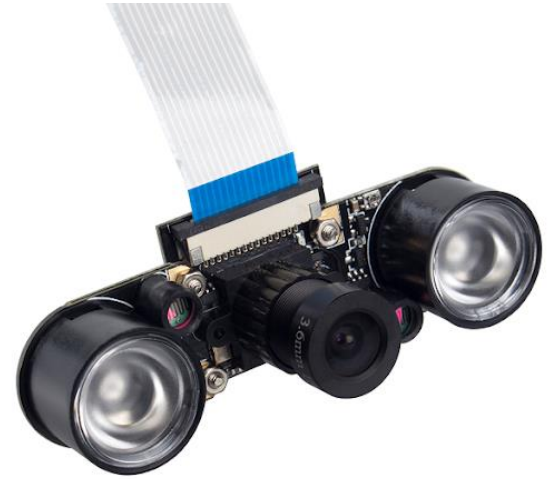


Figure 21 : Caméra raspberry

Montage électronique

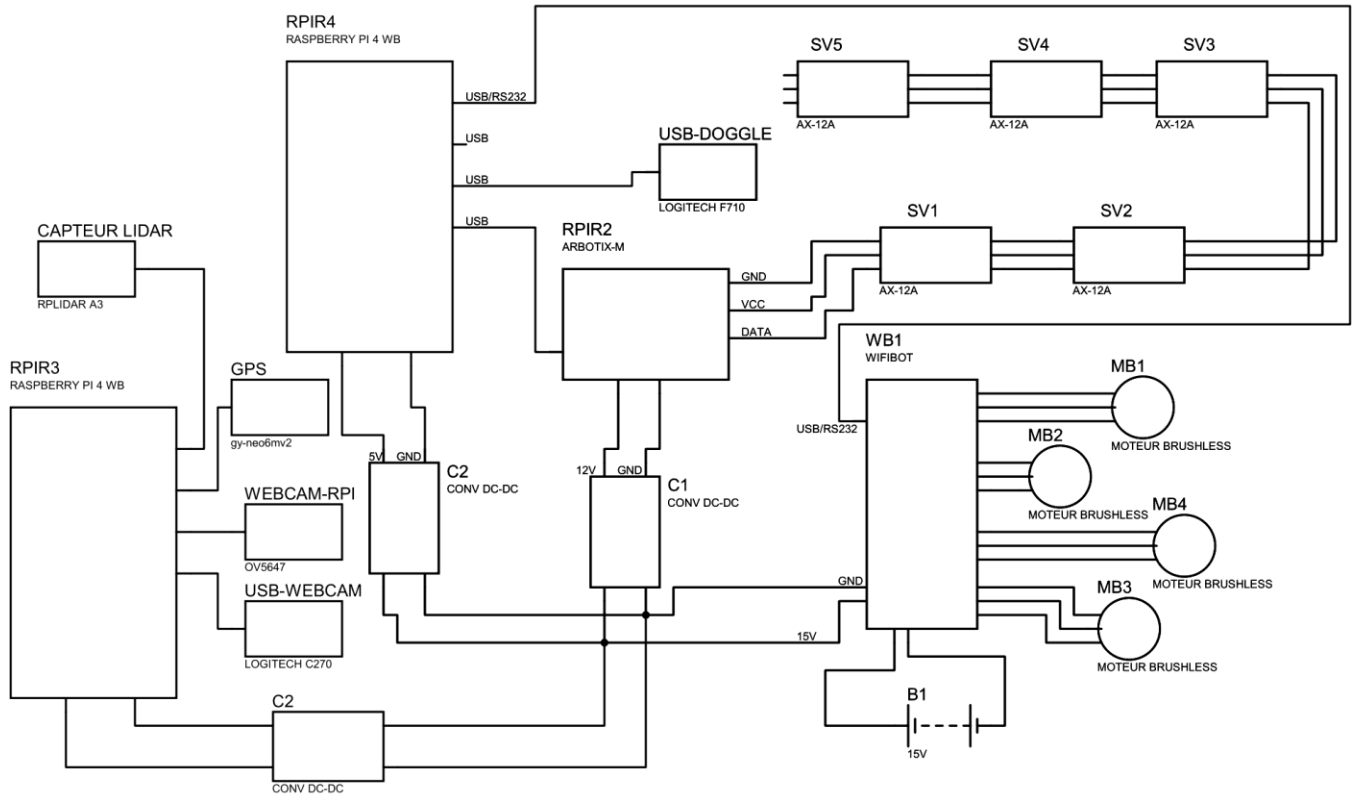


Figure 22 : Montage électronique

Partie programmation vision

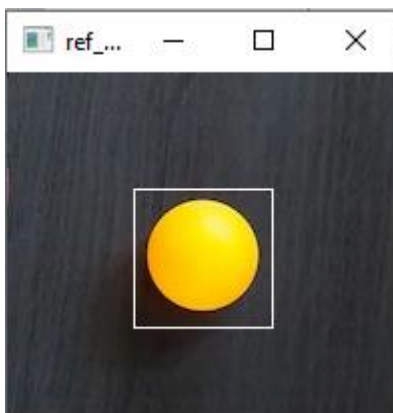
La possible utilisation d'une carte graphique comme carte embarquée pour une meilleure performance pourrait être envisageable, mais nous avons finalement choisi de garder la Raspberry car plus compatible avec les autres modules du robot, le programme doit donc être le moins gourmand possible pour toujours assurer la bonne vision et un frame rate suffisant.

Les différentes étapes de la programmation ont été les suivantes :

- Recherches YOLOv5 (trop gourmand en ressources pour une Raps) / utilisation d'open CV ou cv2 possible / créer un programme léger pour tourner efficacement car il devra combiner la vue en temps réelle du robot ainsi que la détection et l'interprétation de l'image observée



- Détection sur une photo d'objet



- Détection de bords de l'objet

```
frame = imutils.resize(frame, width=600)
blurred = cv2.GaussianBlur(frame, (11, 11), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

mask = cv2.inRange(hsv, ColorLower, ColorUpper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

cnts = imutils.grab_contours(cnts)
```

- Création d'un cercle afin de s'en servir ensuite pour trouver la distance caméra/objet

```
if len(cnts) > 0:
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

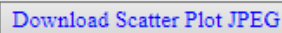
    if radius > 15 :
        cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 3)
        cv2.circle(frame, center, 5, (0, 0, 255), -1)

        if radius != 0 :
            cv2.putText(frame, f"Largeur observee = {round(radius,2)} unit", (50, 50), fonts, 1, (WHITE), 2)

pts.appendleft(center)
```

- | | B | C | D | E | F | G | H | I |
|------|---|------|----|------|------|----|------|---|
| elle | | 21 | 36 | 40,5 | 45,5 | 29 | 52,5 | |
| | | 68,5 | 41 | 36,5 | 31 | 51 | 26 | |

argeur observee = 60.69



```
f"Distance = {round(72.24*0.98**int(radius),2)} cm".
```

- Mise en place du module sur python pour le détection vidéo (en temps réel)

```
if not args.get("video", False):  
    vs = VideoStream(src=0).start() #set up usb port  
  
else:  
    vs = cv2.VideoCapture(args["video"])  
  
time.sleep(2.0)
```

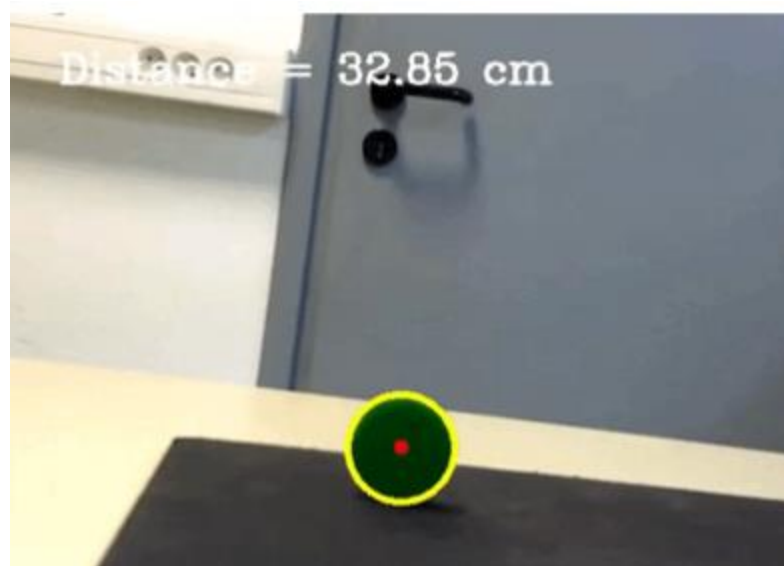
- Comparaison sur cv2 pour un fichier xml (grosse base de données)

```
ball_detector = cv2.CascadeClassifier("C:/Users/33778/Documents/Cours_Esirem/4A/Projet_4A/CodesPythonXml/rawballdetection.xml")
```

- Adaptation du programme pour sa légèreté avec simplement détection finale d'une balle de couleur.

```
ColorLower = (29, 86, 6)    #red#(160,100,20)  
ColorUpper = (64, 255, 255) #red#(179,255,255)
```

Version finale du programme transférée et adaptée à la Raspberry, où le frame rate est largement suffisant pour notre utilisation et où la valeur distance-objet peut être communiquée directement au bras robot.



Mise en commun et assemblage

Une fois assemblé, voici l'apparence finale de notre robot :



Figure 23 : Wifibot



Figure 24 : Wifibot

Quelles optimisations possibles ?

Les optimisations possibles du robot pourront être :

1. La mise en place d'un système de commande sans fil performant afin de commander le robot sur une très grande distance
2. L'ajout d'un deuxième bras robotisé afin de permettre la manipulation d'objets de plus grande taille.
3. La création d'un logiciel pour le déplacement autonome pour le repenser en robot mobile.
4. Zone de stockage sur le robot pour récupérer plus de déchets.

Difficultés :

La première et principale difficulté a été la connexion avec le WifiBot. En effet, le code fourni dans la documentation technique était adapté à la carte mère vendue avec le robot. Malheureusement dans notre cas, il a fallu utiliser un Raspberry Pi et nous orienter plus vers un système tournant sous Ros.

Le second problème est que le Raspberry Pi a du mal à faire tourner l'intégralité du système correctement.

Avis personnel

Joshua Theron : Ce projet a été un réel plaisir à suivre, c'était une idée que je voulais réaliser de longue date. Il m'a personnellement beaucoup apporté car j'ai pu découvrir le domaine des drones terrestres et leur fonctionnement. La partie où j'ai le plus appris a été la programmation du Lidar qui m'a apporté de nouvelles compétences.

Gregory Davillars : Ayant déjà eu l'opportunité de travailler avec Raspberry et ROS, ce projet a été une expérience valorisante, me permettant d'approfondir mes connaissances dans ces domaines. Il a également renforcé mes compétences en Linux et en Bash, grâce à l'usage de ssh pour la connexion entre le WifiBot et mon PC, ainsi que le développement d'un script de contrôle en Bash.

Bastien Berland : Projet enrichissant, une vraie envie de le mettre en œuvre et de le concrétiser, réflexion approfondie lors de l'établissement du cahier des charges et des diagrammes. Sujet rempli de difficultés techniques qui apportent de nombreuses solutions intéressantes.

Remerciements

Nous tenons tout d'abord à remercier Joaquim Rodriguez qui nous a été d'une grande aide lors de la mise en route du WifiBot, mais aussi Raphaël Duverne pour le prêt matériel. Nous souhaitons également adresser nos remerciements les plus sincères aux équipes pédagogiques et administratives de l'ESIREM ainsi que de l'IUT qui ont facilité nos démarches et ont assuré un suivi quand nous le désirions tout au long de notre projet.

Conclusion :

Après de longues semaines de recherches et de mise en place, nous avons enfin pu créer une maquette fonctionnelle. Le Robot d'exploration a été un projet des plus intéressants, qui nous a donné du fil à retordre. Il nous a permis de découvrir beaucoup de choses au niveau électrique, mécanique et informatique et a pu renforcer notre cohésion et travail d'équipe.

Sources

<https://www.wifibot.com/>

<https://github.com/search?l=C%2B%2B&q=wifibot&type=Repositories>

<https://www.freva.com/assign-fixed-usb-port-names-to-your-raspberry-pi/>

<https://arduino103.blogspot.com/2017/10/raspberry-pi-detection-du-materiel-et.html>

<https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/#overview>

https://github.com/radosz99/tennis-ball-detector/blob/main/classifiers/cascade_12stages_24dim_0_25far.xml

<https://pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/>

https://github.com/kipr/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

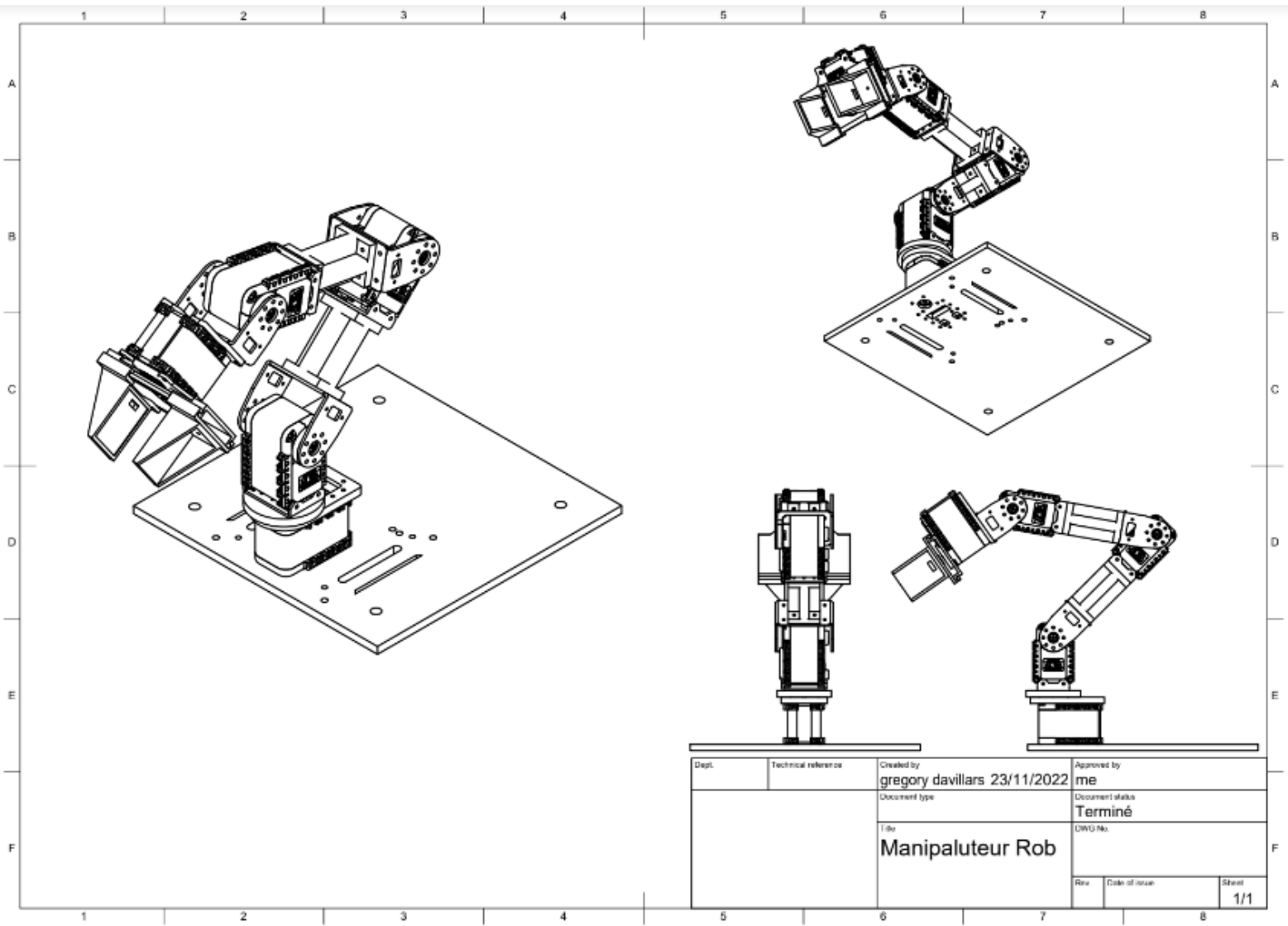
<https://bitbucket.org/Joako1991/code-wifibot/src/master/>

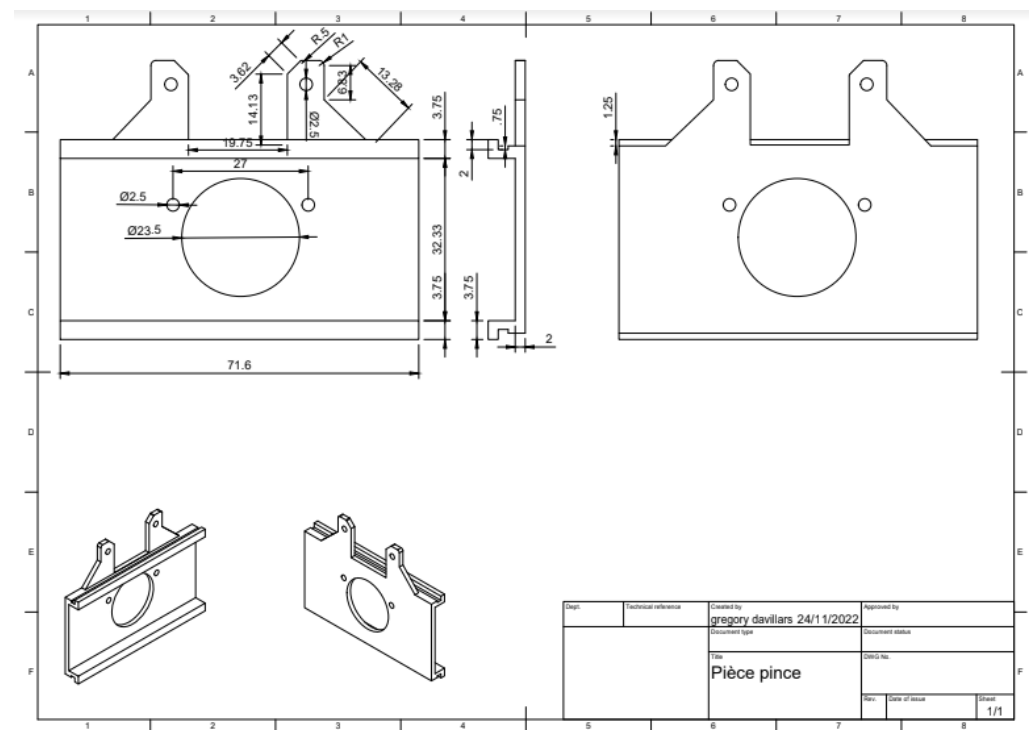
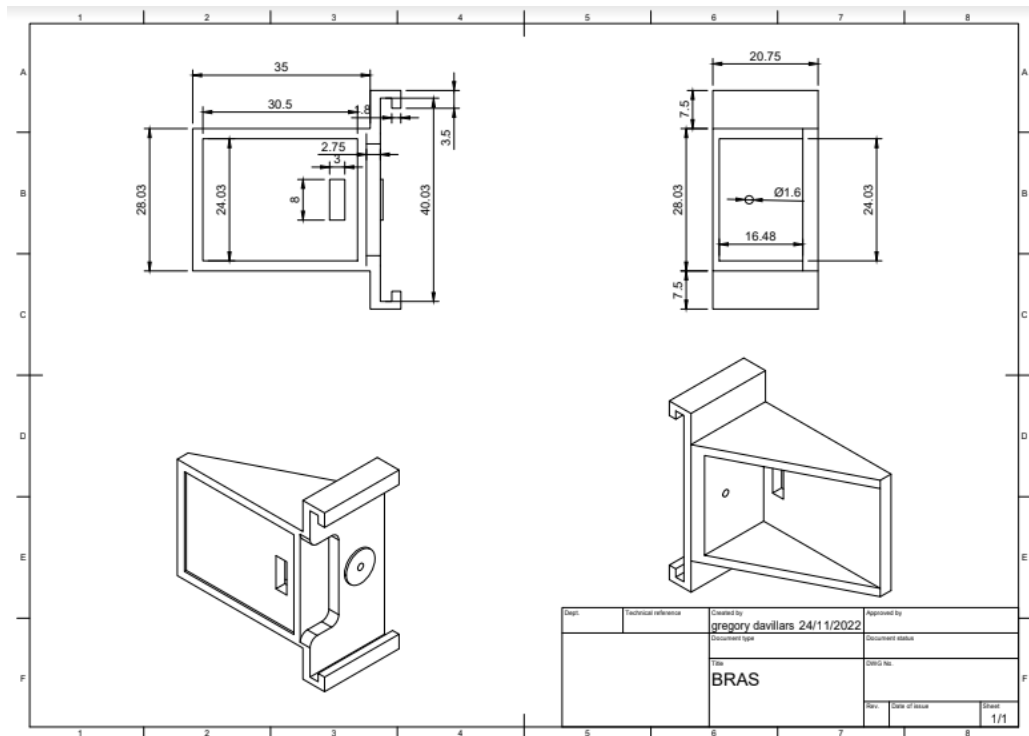
Voici le lien vers notre espace de travail GitHub, qui contient notre projet ROS :

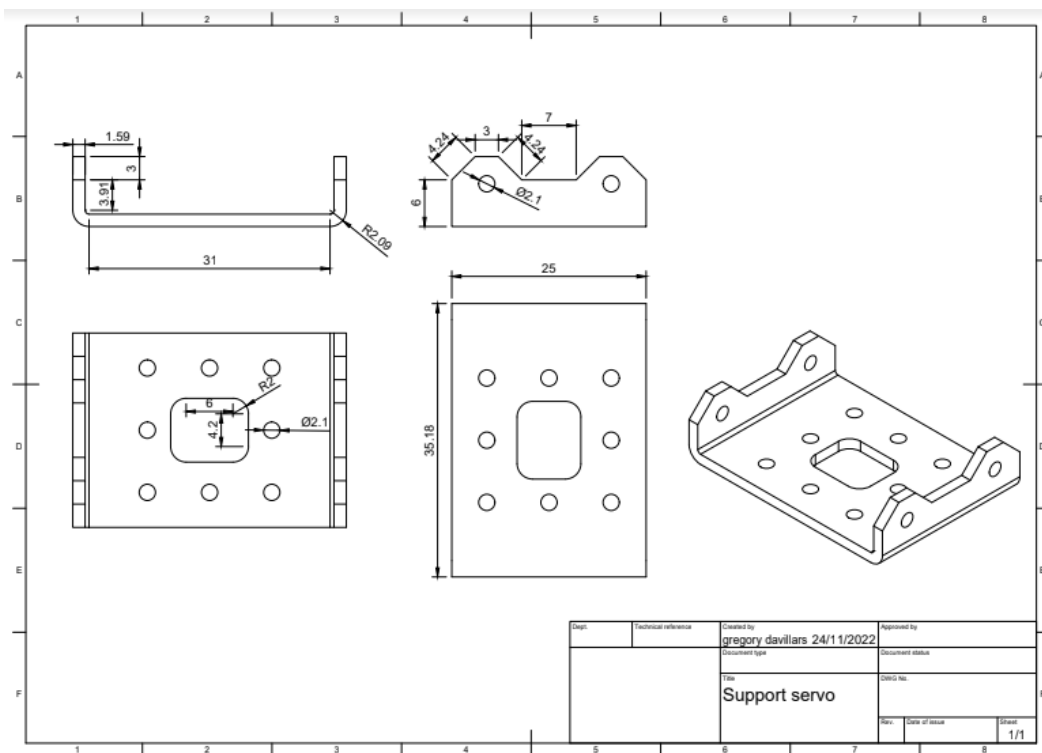
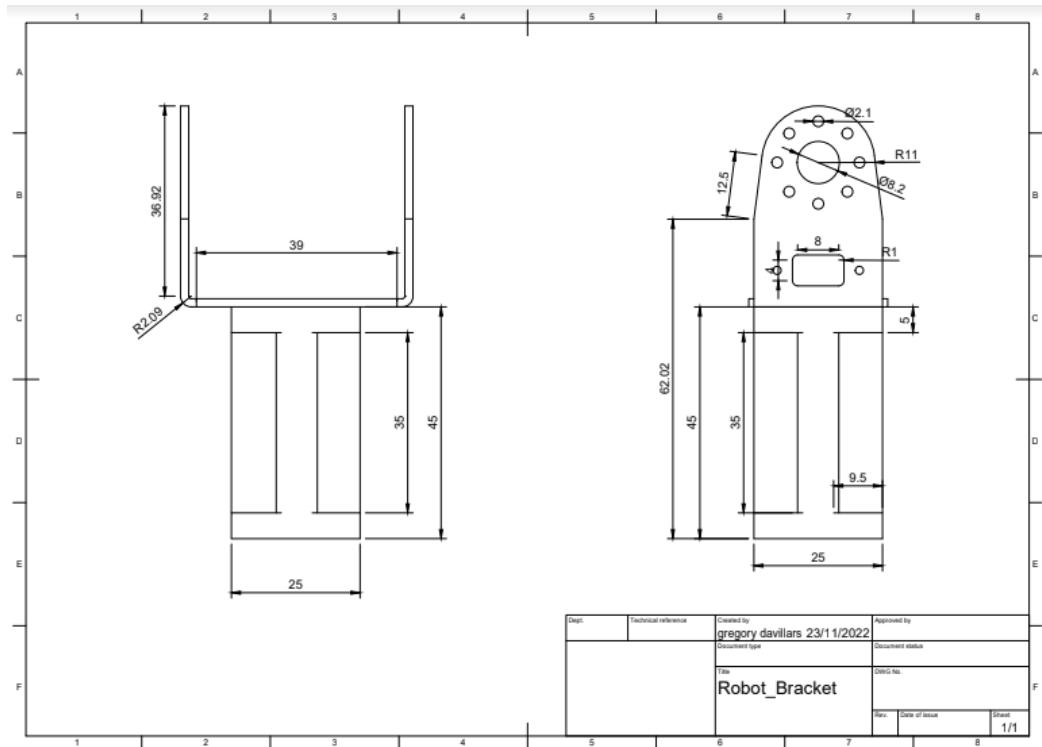
https://github.com/Greg7900/Wifibot_5A_Project

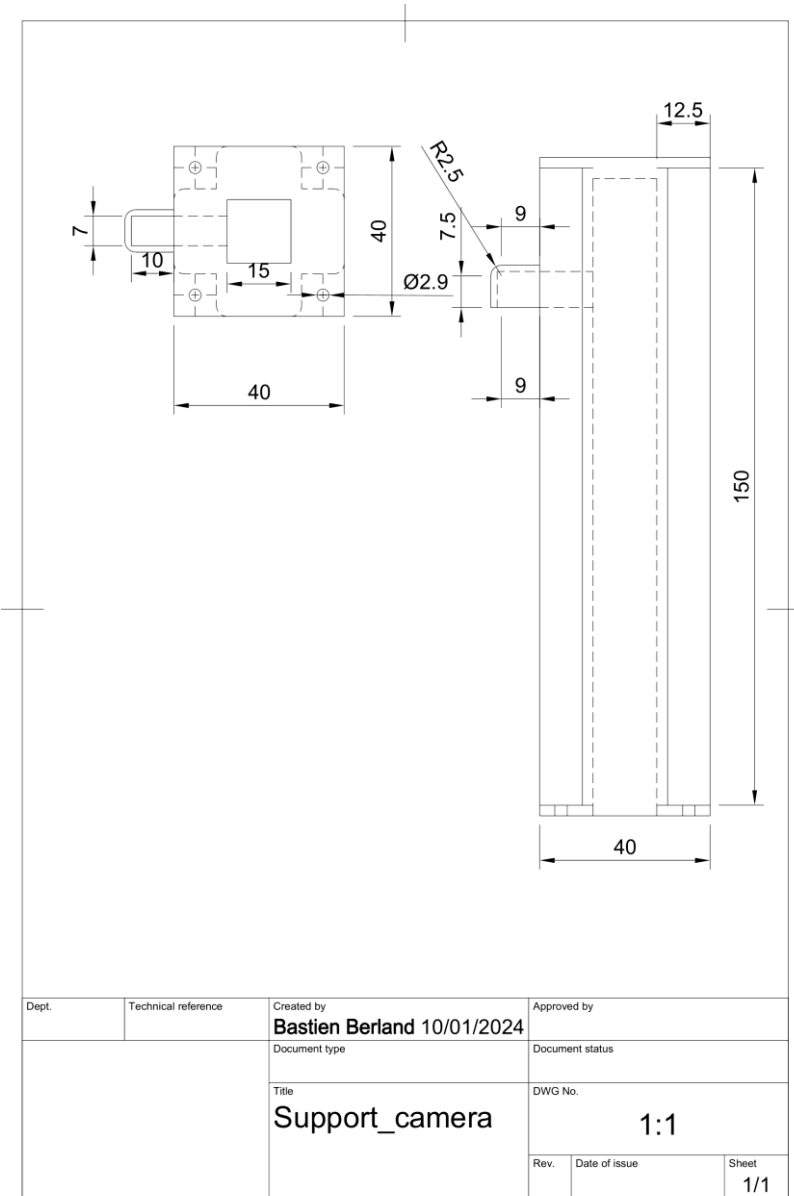
Annexes

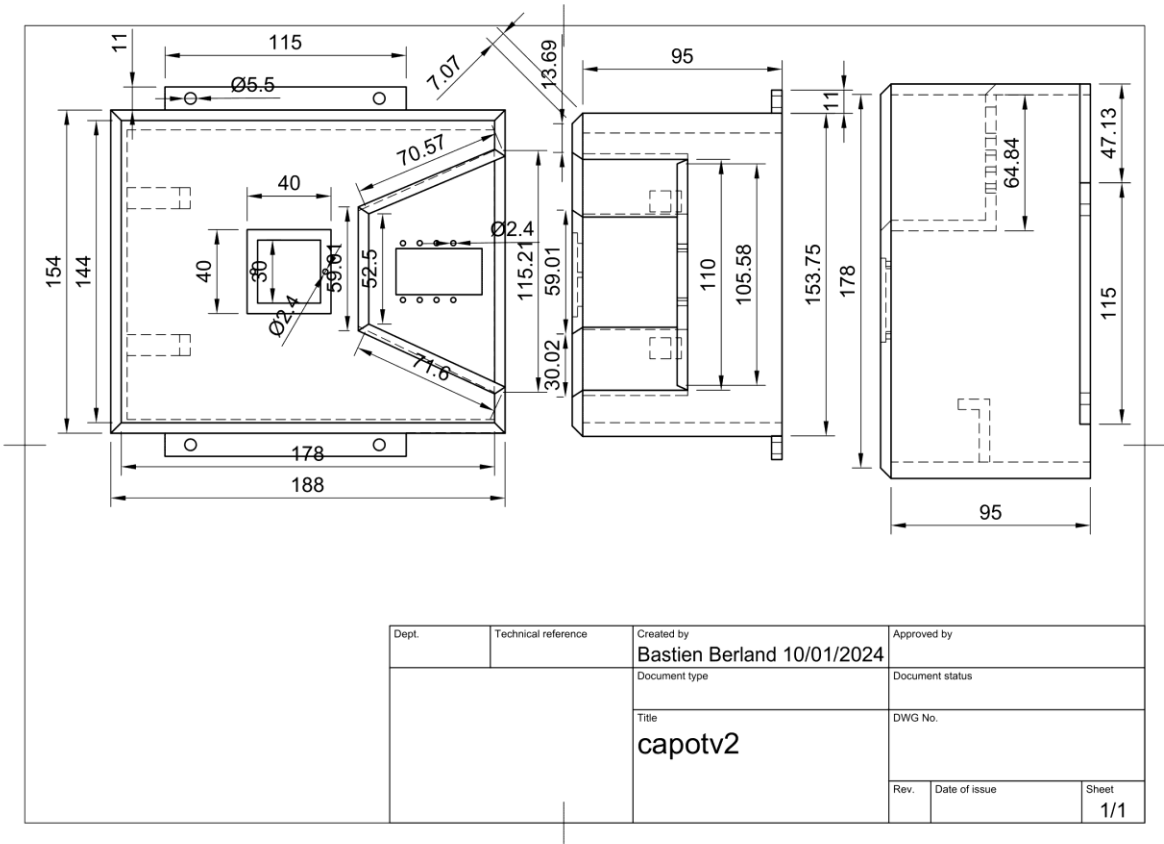
Annexe 1

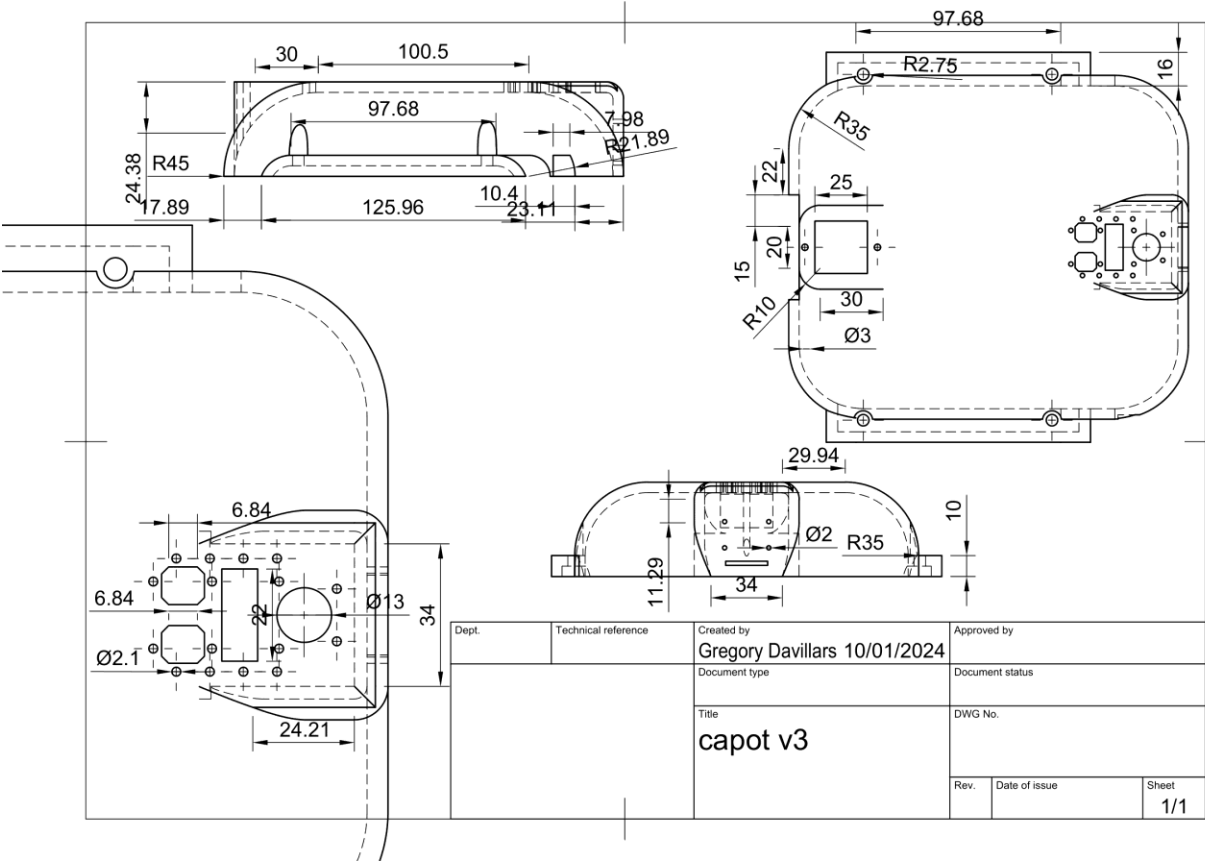


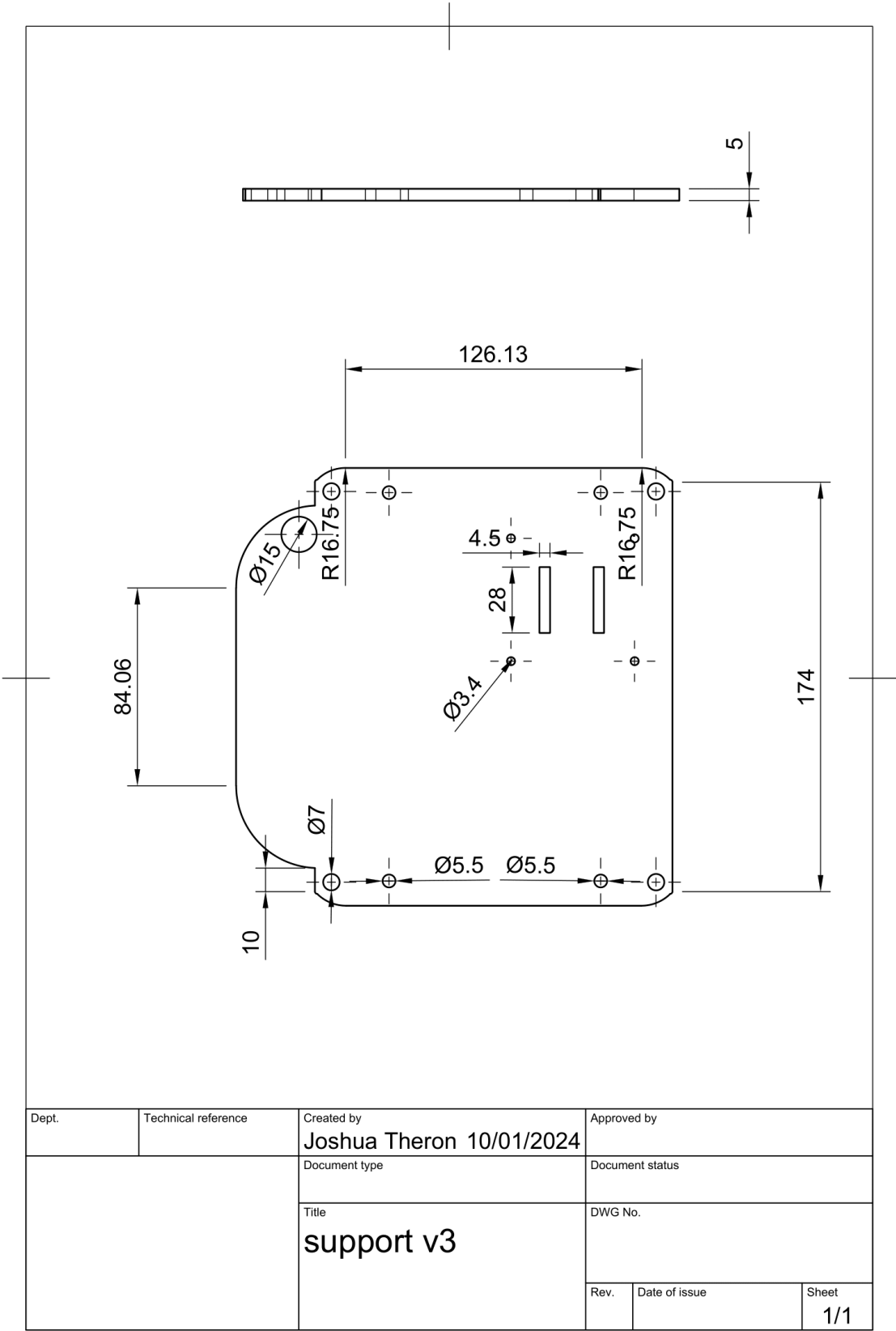


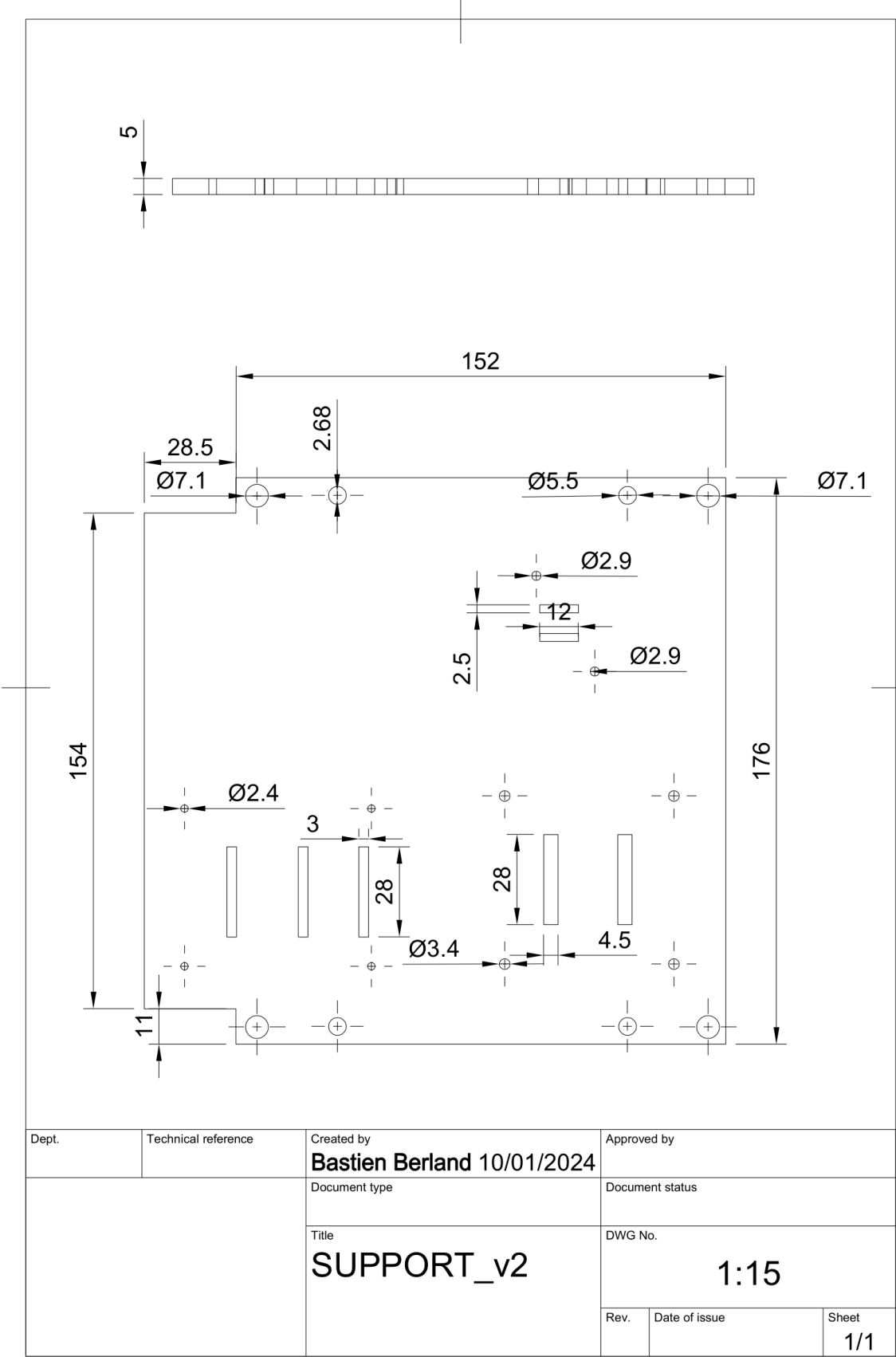








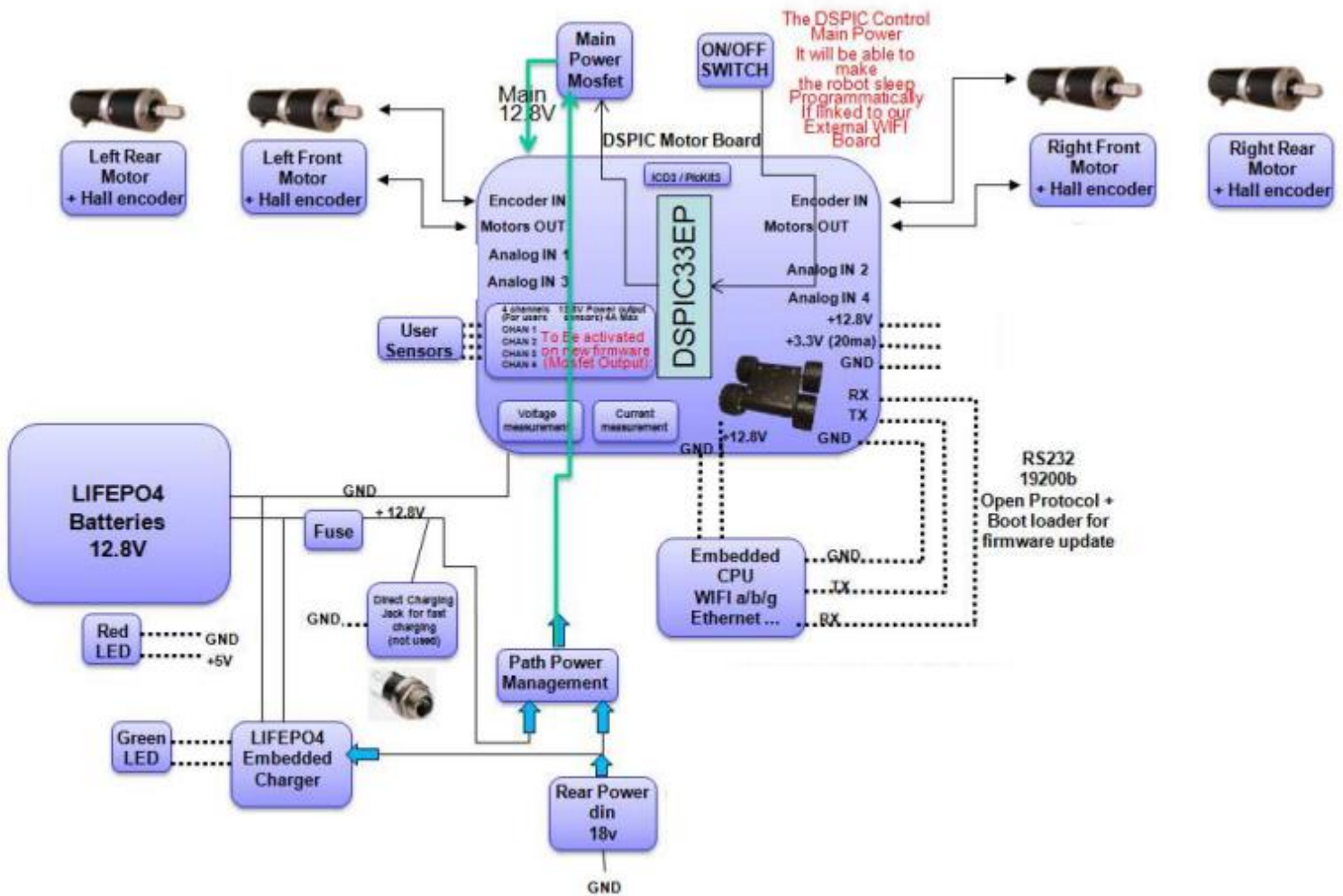




Annexe 2 : Fichier URDF

```
Wifibot.xacro x settings.json
C: > Users > grego > OneDrive > Bureau > wifibot_urdf > Wifibot_description > urdf > Wifibot.xacro
1  <?xml version="1.0" ?>
2  <robot name="Wifibot" xmlns:xacro="http://www.ros.org/wiki/xacro">
3
4  <xacro:include filename="$(find Wifibot_description)/urdf/materials.xacro" />
5  <xacro:include filename="$(find Wifibot_description)/urdf/Wifibot.trans" />
6  <xacro:include filename="$(find Wifibot_description)/urdf/Wifibot.gazebo" />
7  <link name="base_link">
8  <inertial>
9      <origin xyz="0.0008538308347129231 -0.00188802983375572 0.06669854261691827" rpy="0 0 0"/>
10     <mass value="30.263664321753744"/>
11     <inertia ixx="0.124124" iyy="0.154149" izz="0.248504" ixy="-2e-05" iyz="-3e-06" ixz="-2.4e-05"/>
12 </inertial>
13 <visual>
14     <origin xyz="0 0 0" rpy="0 0 0"/>
15     <geometry>
16         <mesh filename="package://Wifibot_description/meshes/base_link.stl" scale="0.001 0.001 0.001"/>
17     </geometry>
18     <material name="silver"/>
19 </visual>
20 <collision>
21     <origin xyz="0 0 0" rpy="0 0 0"/>
22     <geometry>
23         <mesh filename="package://Wifibot_description/meshes/base_link.stl" scale="0.001 0.001 0.001"/>
24     </geometry>
25 </collision>
26 </link>
27
```

Annexe 3 : Schéma Wifibot



WifibOT LAB V4

Annexe 4 : Programme Python

```
1  from collections import deque
2  from imutils.video import VideoStream
3  import argparse
4  import cv2
5  import imutils
6  import time
7
8  fonts = cv2.FONT_HERSHEY_COMPLEX
9  WHITE = (255, 255, 255)
10
11 # construct the argument parse and parse the arguments
12
13 ap = argparse.ArgumentParser()
14 ap.add_argument("-v", "--video",
15                 help="path to the (optional) video file")
16 ap.add_argument("-b", "--buffer", type=int, default=64,
17                 help="max buffer size")
18 args = vars(ap.parse_args())
19
20 ColorLower = (29, 86, 6)      #red#(160,100,20)
21 ColorUpper = (64, 255, 255)   #red#(179,255,255)
22 pts = deque(maxlen=args["buffer"])
23
24 if not args.get("video", False):
25     vs = VideoStream(src=0).start() #set up usb port
26
27 else:
28     vs = cv2.VideoCapture(args["video"])
29
30 time.sleep(2.0)
```

```

31
32
33 while True:
34     # grab the current frame
35     frame = vs.read()
36     frame = frame[1] if args.get("video", False) else frame
37
38     if frame is None:
39         break
40
41
42     frame = imutils.resize(frame, width=600)
43     blurred = cv2.GaussianBlur(frame, (11, 11), 0)
44     hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
45
46
47     mask = cv2.inRange(hsv, ColorLower, ColorUpper)
48     mask = cv2.erode(mask, None, iterations=2)
49     mask = cv2.dilate(mask, None, iterations=2)
50     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
51
52     cnts = imutils.grab_contours(cnts)
53
54     center = None
55
56     # only proceed if the ball is found
57
58     if len(cnts) > 0:
59
60         c = max(cnts, key=cv2.contourArea)
61         ((x, y), radius) = cv2.minEnclosingCircle(c)
62         M = cv2.moments(c)
63         center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
64
65         if radius > 15 :
66
67             cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 3)
68             cv2.circle(frame, center, 5, (0, 0, 255), -1)
69

```

```

70
71     if radius != 0 :
72         cv2.putText(frame, f"Distance = {round(72.24*0.98**int(radius),2)} cm", (50, 50), fonts, 1, (WHITE), 2)
73
74     pts.appendleft(center)
75
76
77     # show the frame to our screen
78     cv2.imshow("Frame", frame)
79     key = cv2.waitKey(1) & 0xFF
80     # if the 'q' key is pressed, stop the loop
81
82     if key == ord("q"):
83         break
84     # if we are not using a video file, stop the camera video stream
85 if not args.get("video", False):
86     vs.stop()
87     # otherwise, release the camera
88 else:
89     vs.release()
90 # close all windows
91 cv2.destroyAllWindows()

```

```

1 <?xml version="1.0" ?>
2 <robot name="Wifibot" xmlns:xacro="http://www.ros.org/wiki/xacro">
3
4 <xacro:include filename="$(find wifibot_description)/urdf/materials.xacro" />
5 <xacro:include filename="$(find wifibot_description)/urdf/Wifibot.trans" />
6 <xacro:include filename="$(find wifibot_description)/urdf/Wifibot.gazebo" />
7
8
9
10 <link name="world">
11 </link>
12
13 <xacro:macro name="wifibot_v2">
14     <joint name="chassis_joint" type="fixed">
15         <parent link="world"/>
16         <child link="chassis_link"/>
17         <origin xyz="-0.1 0 0" rpy="0 0 0"/>
18     </joint>
19     <link name="chassis_link">
20         <inertial>
21             <origin xyz="0.0008538308347129231 -0.00188802983375572 0.06669854261691827" rpy="0 0 0"/>
22             <mass value="30.263664321753744"/>
23             <inertia ixx="0.124124" iyy="0.154149" izz="0.248504" ixy="-2e-05" iyz="-3e-06" ixz="-2.4e-05"/>
24         </inertial>
25         <visual>
26             <origin xyz="0 0 0" rpy="0 0 0"/>
27             <geometry>
28                 <mesh filename="package://wifibot_description/meshes/base_link.stl" scale="0.01 0.01 0.01"/>
29             </geometry>
30             <material name="gray"/>
31         </visual>
32         <collision>
33             <origin xyz="0 0 0" rpy="0 0 0"/>
34             <geometry>
35                 <mesh filename="package://wifibot_description/meshes/base_link.stl" scale="0.01 0.01 0.01"/>
36             </geometry>
37         </collision>
38     </link>

```