



ECOLE
POLYTECHNIQUE
DE BRUXELLES

TRAN-H201 - Création d'un robot ramasseur-trieuse d'objets

Rapport final

BENDALI-BRAHAM Nadyme

MATAGNE Milo-Matéo

KEMPTER Tristan

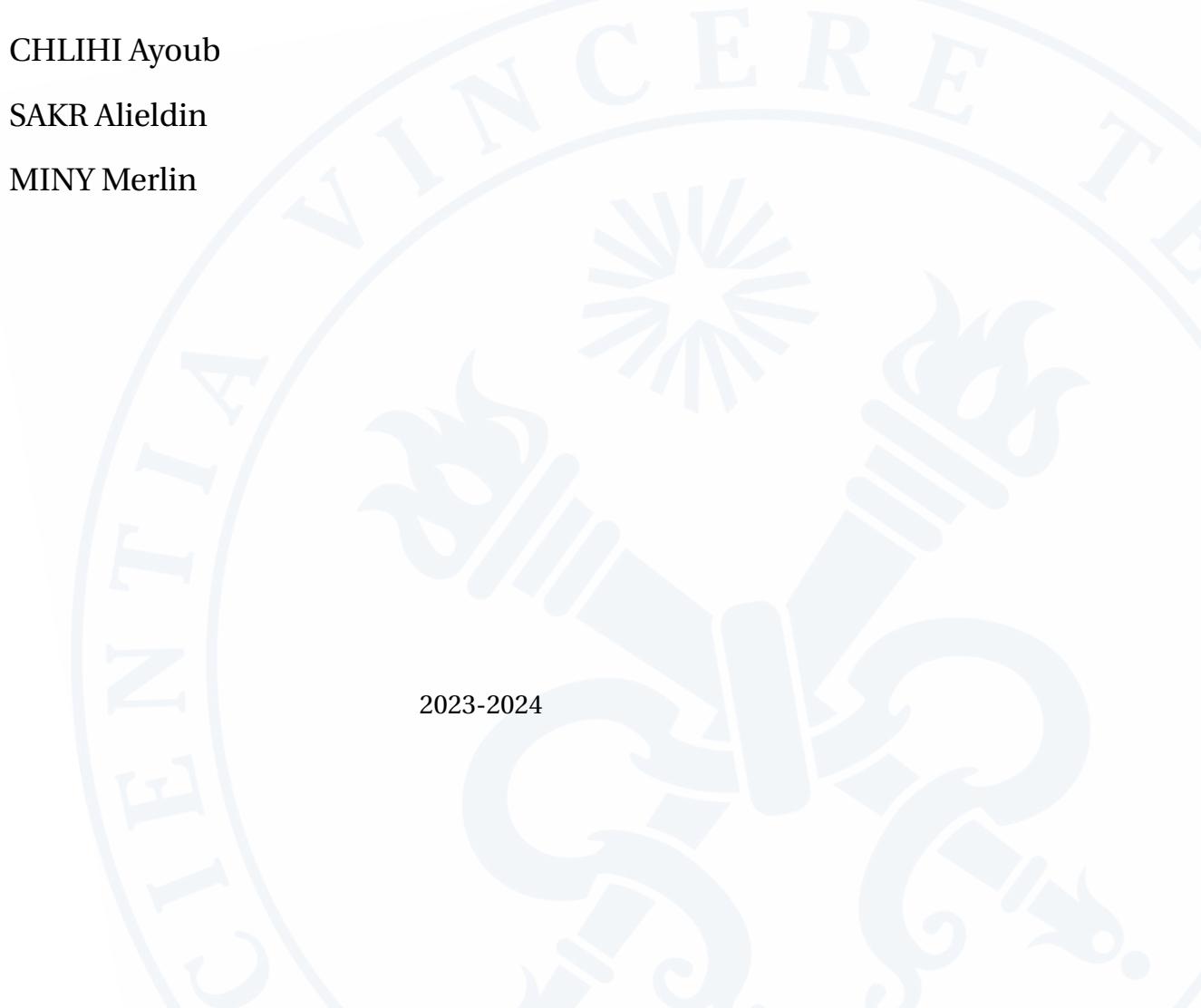
MAT Matthew

Professeur : Jérémie ROLAND

CHLIHI Ayoub

SAKR Alieldin

MINY Merlin

A large, faint watermark of the ULB seal is visible in the background. The seal is circular with the text "CLEMENTIA VIN CERET" around the top edge and "SCIENTIA" around the bottom edge. In the center is a stylized torch or flame.

2023-2024

Abstract

Dans le cadre du projet d'électromécanique de BA2 à l'Ecole Polytechnique de Bruxelles lors de l'année 2023-2024, il a été demandé aux étudiants de réaliser un robot ramasseur-trieuse d'objets. Celui-ci doit évoluer le long d'un tracé connu à l'avance sur lequel il doit se repérer de façon autonome. A certaines extrémités du tracé se trouvent des objets que le robot doit reconnaître et déplacer vers un autre emplacement pré-déterminé, en fonction de si l'objet se trouve au bon endroit ou pas. Ce rapport présente la démarche menée et les résultats obtenus par un groupe de sept étudiants accompagnés par un tuteur.

Pour commencer, plusieurs composants électroniques ont été comparés dans le but d'optimiser le prototype final. A l'aide de Python et de C++, un algorithme de déplacement et de logique a été codé. Une régulation PID a été créée pour maintenir le robot sur la ligne et améliorer sa précision. Ensuite, un prototype 3D a été réalisé à l'aide des logiciels Fusion360 et Solidworks grâce aux imprimantes 3D disponibles au FabLab.

Pour terminer, les différents tests ont mis en lumière le travail effectué par le groupe, dans le but de concevoir un robot autonome et fonctionnel.

Mots-clés : robot, arduino, capteur, moteur, régulation, prototype.

Nombre de mots dans l'abstract : 200

Table des matières

1	Introduction	3
2	Description du projet	4
3	Prototype	6
3.1	Introduction	6
3.2	Exigences de conception	6
3.2.1	Suivi du tracé	6
3.2.2	Reconnaissance d'objets	8
3.3	Modélisation 3D	9
3.3.1	Châssis	9
3.3.2	Bras mécaniques	12
3.4	Composants électroniques	16
3.4.1	Description générale du robot	16
3.4.2	Arduino	17
3.4.3	Shield Arduino Nano Every	18
3.4.4	Servomoteurs	19
3.4.5	Micro switch	19
3.4.6	Capteur de distance à ultrason	20
3.4.7	Capteur infrarouge	21
3.4.8	Moteur à courant continu	22
3.5	Implémentation des moteurs	23
4	Modélisation mathématique	24
4.1	Équations mécaniques	24

4.2 Régulation PID	25
4.2.1 Principes de l'automatisation	25
4.2.2 Proportionnel Intégral Dérivée	25
5 Programmation	28
5.1 Algorithme	28
5.1.1 Introduction	28
5.1.2 Code python	28
5.1.3 Optimisations	30
5.1.4 Code Arduino	32
6 Gestion	35
6.1 Gestion d'équipe	35
6.1.1 SWOT	36
6.2 Budget et dépenses	37
7 Conclusion	38
Annexes	1
A Fiche technique shield Arduino nano every	1
A.1 De Python à C++	2
A.1.1 Traduction du code Python	2
A.1.2 Fusion des deux codes	3
A.1.3 Code C++	3

1

Introduction

Dans le cadre de la formation d'ingénieur civil à l'Ecole Polytechnique de Bruxelles, tout au long de leur seconde année de bachelier, les étudiants sont amenés à réaliser un projet multidisciplinaire par équipe d'environ 6 étudiants. Ils sont accompagnés tout au long de leur projet par un tuteur ou une tutrice. Celui de l'année académique 2023-2024 en électromécanique consiste en la conception et la réalisation d'un robot, qui doit être capable de se mouvoir le long d'un tracé pré-déterminé, de reconnaître des objets dont les dimensions sont connues et de les déplacer au besoin. Le robot est jugé sur sa capacité à se repérer sur le tracé, et à déplacer efficacement les différents objets ainsi que sur la qualité d'optimisation de ses déplacements le long du tracé. Les sections suivantes vont traiter le projet plus en détail. L'une d'entre elles sera centrée sur la réalisation du prototype, et présentera la modélisation de celui-ci ainsi que les différents composants utilisés. Ensuite, la suivante sera consacrée à la modélisation mathématique où la régulation PID y sera notamment expliquée. Un autre chapitre expliquera le raisonnement du code, qui joue un rôle crucial dans ce projet. Le subséquent décrira comment le robot a été assemblé et construit. Enfin, l'ultime section traitera de la gestion d'équipe, et présentera un SWOT.

2

Description du projet

Le cahier des charges stipule que le robot doit être capable de se déplacer de façon autonome sur le tracé (2.1). Celui-ci est composé d'une bande de ruban adhésif noir déposée sur une plaque en bois peinte en blanc. Comme indiqué sur le schéma, le tracé est composé de sept branches aux extrémités desquelles se trouvent 6 stations et un départ. Chacune des stations possède un label A, B ou C; ceux-ci indiquent la place correcte pour les objets correspondants.

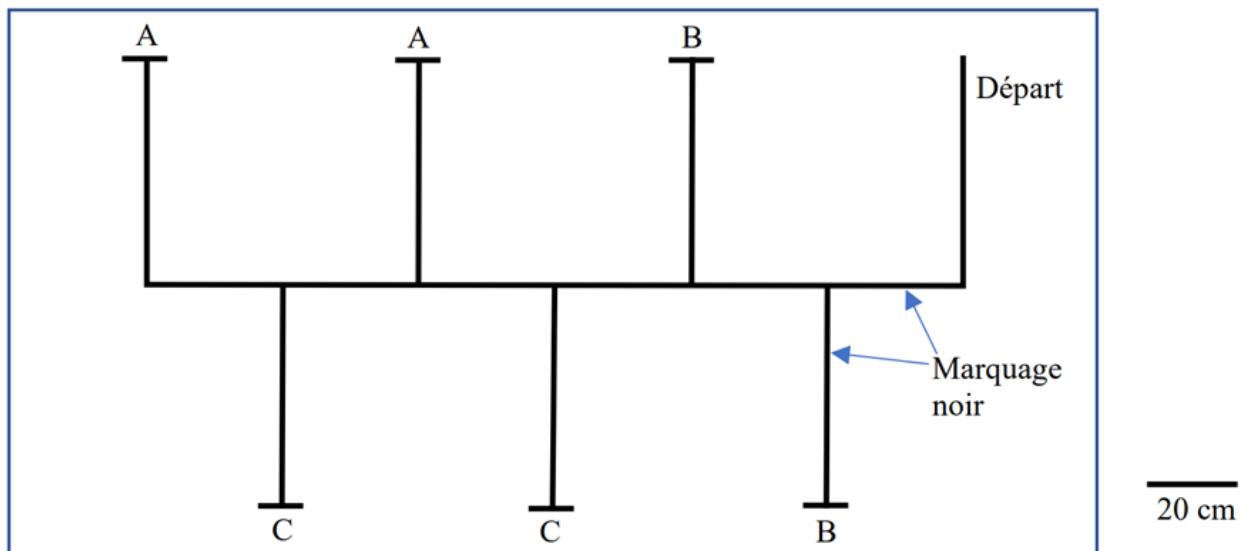


FIGURE 2.1 – Schéma du tracé du robot fourni dans le guide du projet.

Il y a 3 types d'objets (2.2), chaque station ne peut contenir qu'au plus un objet, et au plus 4 objets sont répartis entre les six stations (on sait donc qu'il n'y a jamais plus de deux objets du même type).

	Type A, haltère (8 cm de haut, diamètre 4 cm / 2 cm)
	Type B, parallélépipède rectangle à base carrée (8 cm de haut, 4 cm de côté)
	Type C, cylindre (8 cm de haut, 4 cm de diamètre)

FIGURE 2.2 – Tableau fourni dans le guide de projet comprenant les différents types d'objets et leurs dimensions respectives.

La tâche du robot est de faire en sorte que tous les objets soient dans une station de leur type. Pour cela il doit être capable de se repérer sur le tracé (savoir quand il arrive à une station et à quel type de station il se trouve), différencier les types d'objets et être capable de déplacer les objets le long du tracé.

Certaines limitations sont également imposées par le cahier des charges. Le robot doit être complètement autonome une fois posé sur le tracé, aucun repère ou objet extérieur au tracé et aux objets ne peuvent interférer ou aider le robot à se situer, à être piloté, ou même à être alimenté. L'objet sélectionné par le robot doit absolument quitter le sol, il faut donc que le robot soit capable de soulever chaque type d'objet.

3.1 Introduction

Cette section aborde le processus de conception du prototype. Ce dernier doit être capable de supporter tous les composants nécessaires au fonctionnement du robot tout en gardant des dimensions lui permettant de se déplacer correctement sur le circuit. Dans cette section, seront aussi présentés les différents composants placés sur le robot ainsi que leur rôle.

3.2 Exigences de conception

3.2.1 Suivi du tracé

Choix du type de composant

Comme expliqué plus haut à la section (2.1) le robot évolue sur une plaque blanche sur laquelle un tracé connu est marqué à l'aide d'un ruban adhésif noir. Vu la taille du tracé, la piste du robot mobile est immédiatement adoptée et l'on commence à se questionner sur les différents types de capteurs pouvant tirer profit d'une ligne noire contrastant fortement avec le sol blanc, l'alternative étant un système d'odométrie qui n'utilisera pas du tout cette ligne pour se déplacer.

Il est alors possible d'envisager deux types de composants : une caméra, à laquelle serait ajoutée un algorithme de reconnaissance d'images permettant de détecter la position et la direction de la ligne par rapport au robot ou un capteur infrarouge capable de faire la différence entre le ruban adhésif et la surface environnante.

Compte tenu du budget relativement contraignant et surtout de la complexité ajoutée par l'ajout d'un système de reconnaissance d'images, le groupe a préféré dans un premier temps opter pour la solution la plus efficace vu la simplicité du tracé (un seul type de jonction et des bandes toujours droites et supposées exemptes d'irrégularités).

Deux types de détecteurs infrarouges qui étaient fournis dans le kit de démarrage ont pu être testés, ainsi que beaucoup d'autres composants qui vont s'avérer être utiles dans la suite du projet.

Implémentation

Pour pouvoir faire des ajustements au fur et à mesure que le robot avance (c'est-à-dire le maintenir le long de la ligne en corrigeant sa trajectoire) et pour pouvoir détecter les virages à prendre, des capteurs infrarouges ont été implémentés à l'avant du robot. Pour pouvoir suivre la ligne, le robot était équipé d'une plaque de 3 capteurs infrarouges permettant de suivre la ligne. Un central qui permet de s'assurer que le robot est toujours sur la ligne noire, et 2 capteurs latéraux qui permettent d'estimer l'erreur de la position par rapport à la ligne, comme expliqué en (4.2). Chaque capteur renvoie une valeur analogique qui représente le ratio noir/blanc qu'il capte. Une grande valeur est associée à plus de noir et une petite valeur est associée à plus de blanc. Par exemple, si le capteur gauche est à côté de la ligne noire, il va quand même renvoyer une valeur non-nulle car il capte moins de blanc que s'il n'était pas du tout à côté de la ligne noire. Il y a donc besoin des deux capteurs sur les côtés pour pouvoir calculer l'erreur, qui est donc la différence entre les deux valeurs renvoyées par lesdits capteurs. Le robot a également besoin de deux capteurs externes pour pouvoir reconnaître les intersections, pour pouvoir faire la différence entre les intersections avec un chemin allant vers la droite et les intersections avec un chemin allant vers la gauche. Il y a donc un total de 5 capteurs infrarouges qui sont strictement nécessaires au bon fonctionnement du robot. La distance à laquelle sont placés les capteurs initialement n'est pas vraiment importante car il sera essentiel de calibrer ces derniers avant de lancer la procédure.

Cependant, en effectuant les tests, la méthode avec les 5 capteurs infrarouges ne s'est pas avérée performante dû à une distance entre les 3 capteurs de la plaque trop importante. Par conséquent, le triplet de capteurs sur le bas du robot a été remplacé par une plaque de 8 capteurs et les deux capteurs permettant de tourner ont été supprimés. La position de la ligne de capteurs sur le bas du robot a aussi été changée. Le triplet était auparavant placé au milieu du robot (entre les 2 roues) mais l'octet de capteurs est maintenant placé à l'avant du robot ce qui permet au robot d'être plus précis sur les virages. En effet, pour un même angle de déviation, l'arc de cercle sera plus grand si le rayon de déviation est plus grand (comme illustré en 3.2). La valeur étant plus grande pour une petite déviation, le robot se rendra plus vite compte de son erreur.

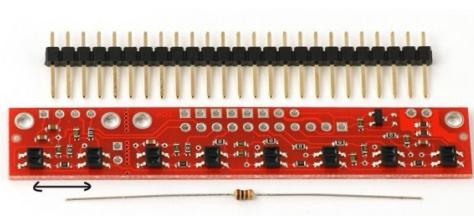


FIGURE 3.1 – Photo de la ligne de 8 capteurs et représentation de la distance entre deux capteurs négligeable par rapport à la longueur de la huit capteurs.

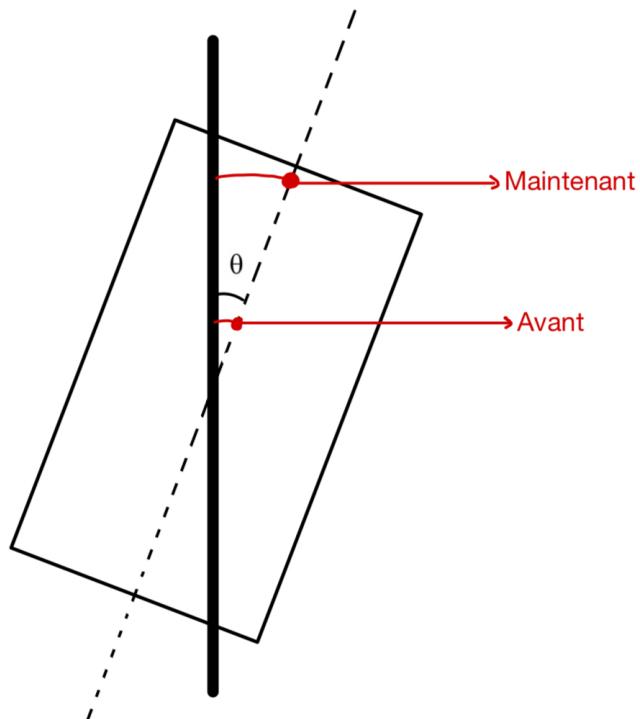


FIGURE 3.2 – Illustration de la justification du changement de position des capteurs (purement illustratif et sert à visualiser la logique de la justification)

3.2.2 Reconnaissance d'objets

Le besoin d'une phase de "reconnaissance" apparut très tôt lors de la recherche d'une stratégie efficace de manipulation des objets. Ladite phase consiste simplement en un premier aller d'un bout à l'autre du tracé du robot pour établir une première carte de présence des colis à chacune des stations afin de décider du trajet optimal. Pour ce faire, un capteur adéquat doit être installé

sur le robot de manière à ce qu'il ait un accès clair à ce qui se trouve devant lui.

Dans le matériel mis à disposition par le Beams se trouvent des capteurs à ultrasons permettant de calculer des distances. Ces capteurs sont utilisés pour détecter la présence ou non d'un objet à un emplacement lors de la phase de repérage. En plus de n'être que très peu onéreux et facile à implémenter, ils retournent des valeurs précises pour la plage de distance à laquelle le robot sera confronté. Il est donc judicieux de se servir de ceux qu'offre le Beams. Ces capteurs à ultrasons ont été testés en les plaçant à une distance de 60 cm de l'objet pour voir s'ils arrivaient bien à capter l'objet, ce qui était le cas.

Dans un second temps, le bras en prenant l'objet doit pouvoir différencier une hâtère d'un parallélépipède et d'un cylindre. Ceci est accompli en palpant l'objet à l'aide du bras mécanique qui est équipé de deux micro switchs, le capteur à ultrason intervient également dans la confirmation de la présence d'un objet. Leur fonctionnement est résumé ci-dessous :

Statut		
Capteur de distance	1	Micro switch côté?
	1	1 -> Parallélépipède
	0	0 Micro switch bas?
	0	1 -> Cylindre 0 -> Hâtère
	0	-> Pas d'objet

Le capteur de distance a également pour rôle de confirmer qu'un objet a bien été soulevé puisque l'objet disparaît de son champ de détection quand il est soulevé.

3.3 Modélisation 3D

Le groupe a utilisé le logiciel Fusion360 afin de créer les différents éléments du prototype. Une modélisation 3D offre plusieurs avantages dans la conception d'un robot. Elle donne une visualisation claire du projet, elle permet aussi de détecter les possibles problèmes et d'optimiser le robot avant la fabrication. Dans le cadre de ce projet, la modélisation 3D permet d'imprimer divers éléments avec une imprimante 3D, notamment celles disponibles au FabLab.

3.3.1 Châssis

Tous les éléments du robot ont un point en commun, le châssis. Il constitue la base sur laquelle tous les éléments vont être attachés. Une modélisation 3D du châssis a été créée afin d'optimiser l'implémentation de ces éléments. À l'aide d'une imprimante 3D, le châssis a été fabriqué en PLA.

Dimensionnement du premier châssis

Le design du robot a toute son importance. Le robot doit être suffisamment stable pour éviter de se retourner ou de faire tomber le colis qu'il transporte. Le dimensionnement du robot doit optimiser l'utilisation de ses ressources. Le positionnement des différents capteurs doit se faire le plus efficacement possible afin de minimiser les cas limites.

Afin de dimensionner correctement le robot, il faut d'abord choisir la bonne disposition des roues. Le modèle qui a été retenu est composé de 2 roues placées à l'arrière et une roue libre à l'avant. Cette structure composée d'une partie motorisée placée à l'arrière a un défaut : le robot peut se retourner s'il accélère trop vite. Fort heureusement et ce parce que le robot doit être capable de récupérer des objets, le robot est équipé d'un bras mécanique positionné à l'avant de celui-ci. Ce bras sert donc de contrepoids et le problème est réglé.

Les dimensions du robot sont celles qui permettent surtout d'incorporer tous les composants. La longueur de celui-ci est quant à elle très importante. Les capteurs infrarouges doivent être à une certaine distance de l'axe des roues afin de bien contrôler ses mouvements. En effet, durant le premier quadrimestre, le robot était équipé de capteurs infrarouges dans l'axe des roues. Il était impossible de correctement contrôler le robot de cette manière. Le premier châssis a eu pour but primaire de pouvoir servir de plateforme de développement pour tous les autres sous-systèmes du robot :

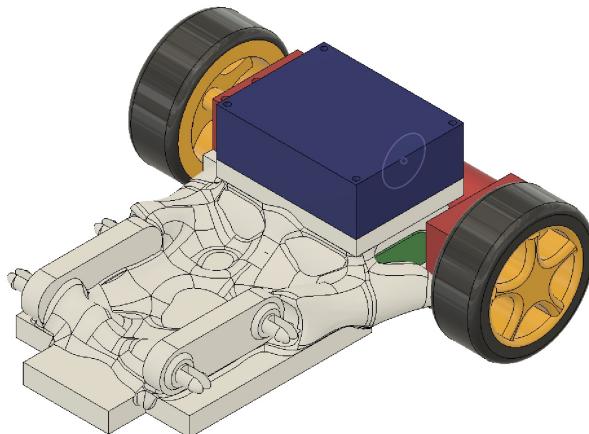


FIGURE 3.3 – Modélisation 3D du premier prototype

- Positionnement des capteurs infrarouges sous le châssis : d'où la présence d'une grande plaque permettant de rechercher une position efficace pour la rangée de capteurs de ligne.
- Modularité du bras mécanique : des clips déformables (non motorisés) permettent d'accro-

cher et décrocher différentes longueurs de bras et de travailler sur la pince de façon complètement indépendante du châssis pour travailler sur la détection d'objets sans accaparer le prototype.

- preuve de concept de la méthode de modélisation automatique.

Second châssis

Le second châssis reprend les éléments du premier en y améliorant certains des points faibles :

- L'écart entre les moteurs (minimal) est conservé.
- Les axes pour le bras mécanique sont augmentés d'un emplacement pour servomoteur.
- La roue libre est reculée pour pouvoir placer les capteurs infrarouges le plus en avant possible par rapport au moteur (la roue libre passe derrière les capteurs).
- Une protection est ajoutée autour pour protéger les capteurs infrarouges des perturbations lumineuses et des potentiels chocs.
- La plaque destinée à tester divers emplacements de capteurs est enlevée.
- La hauteur du support de roue libre est ajustée pour assurer que le robot soit bien modélisé parallèle au sol.
- Des trous passe-câbles sont ajoutés pour réduire la quantité de fils risquant de se coincer dans les parties en mouvement.
- D'autres problèmes mineurs de tolérance sont réglés pour assurer une bonne amplitude de mouvement et la solidité des vis dans le plastique.

Le châssis est alors régénéré à partir de ces nouvelles contraintes à l'aide de l'outil de modélisation automatique de Fusion 360.

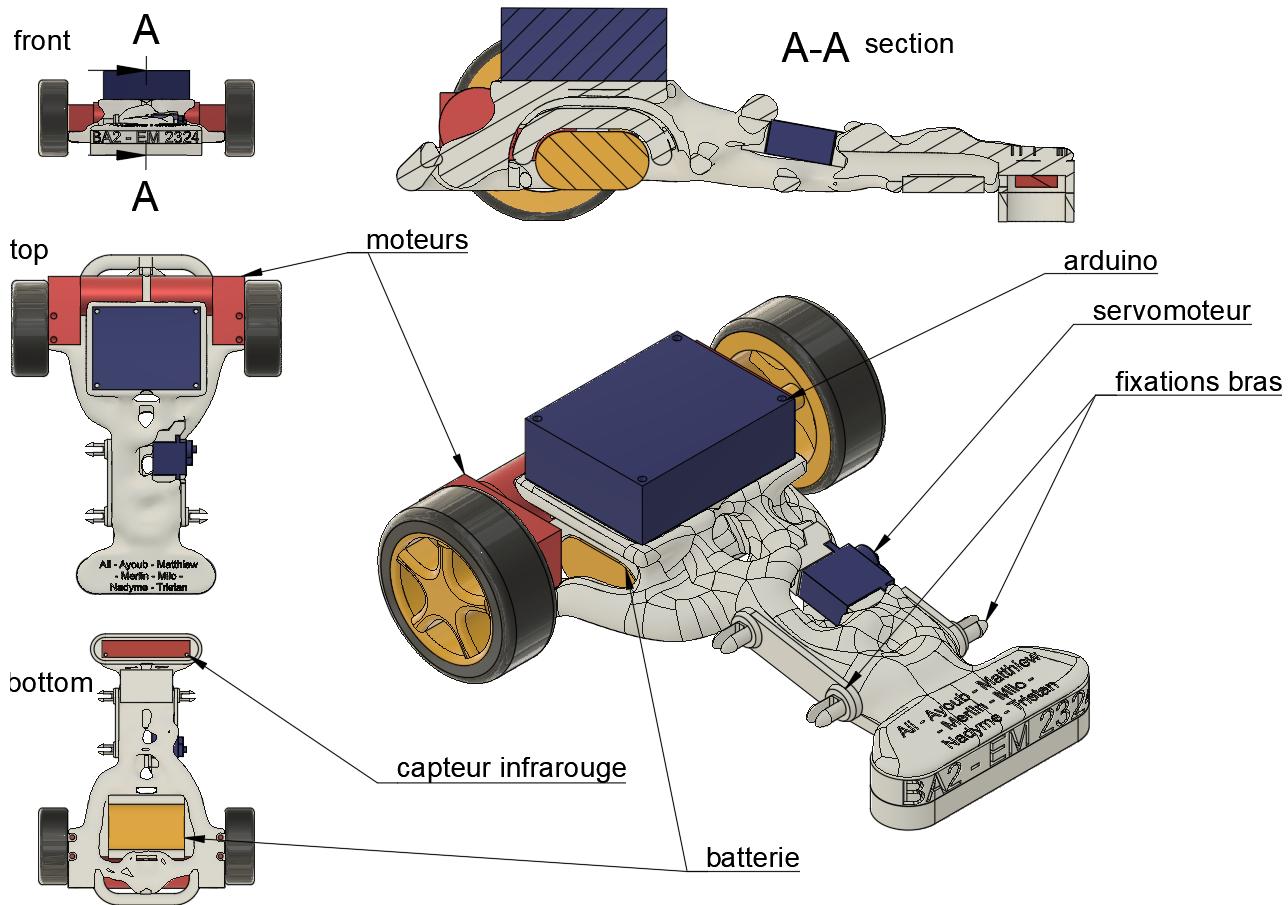


FIGURE 3.4 – Modèle 3D du second châssis avec les composants principaux.

3.3.2 Bras mécaniques

Les bras mécaniques ont été réalisés en 3D de la même manière que le châssis. Grâce à la modélisation 3D, un système d'engrenages a pu être conçu afin d'améliorer la précision du système. Les bras mécaniques sont pilotés par un servomoteur. Afin de reconnaître les objets, chaque bras est équipé de plusieurs micro switchs. Le bras est composé de plusieurs éléments :

- Un support qui sert de connexion entre le châssis et le bras, il s'agit de la première partie mobile sur laquelle sont accrochés les bras ainsi que le servomoteur qui les pilote.
- Un bras motorisé, qui par un jeu d'engrenages fait bouger les bras servant à détecter les objets. Les bras sont aussi les pièces sur lesquelles sont attachées les micro switchs et les "grips".
- Les "grips", des pièces imprimées en plastique plus souple destinées à se déformer au contact de l'objet et à activer les micro switchs afin de reconnaître les objets, ceux-ci sont modulaires et sont passés par plusieurs formes avant d'obtenir des résultats satisfaisants

3.3.2.

Bras

Dans un premier temps, les bras ont été designés avec un simple axe de rotation 3.5. Le mouvement lorsque les bras se refermaient décrivait une courbe. Cela pouvait causer des problèmes pour la reconnaissance des différents objets. En effet, les bras, en suivant ce mouvement, peuvent pousser les objets ou les attraper d'une manière qui empêcherait le système de reconnaissance de fonctionner correctement.

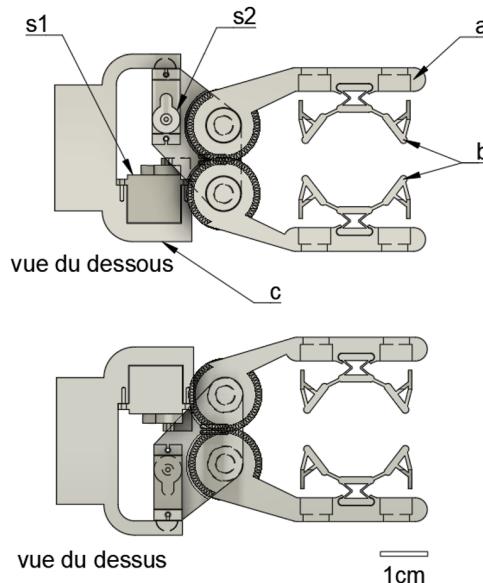


FIGURE 3.5 – Modèle 2D des bras mécaniques, avec :

- a : bras rigide en PLA
- b : pince souple en TPU
- c : support de la pince relié au châssis
- s : servomoteurs

Il a alors été décidé de revoir le système des bras mécaniques afin de l'optimiser. Cette deuxième version permet aux bras de fonctionner en parallèle. De cette manière, il est assuré que les objets soient correctement attrapés et permet également d'éviter des erreurs lors de la reconnaissance d'objets.

Le choix du PLA pour les bras se justifie par la rigidité du matériau, et du TPU par sa souplesse et son adhérence.

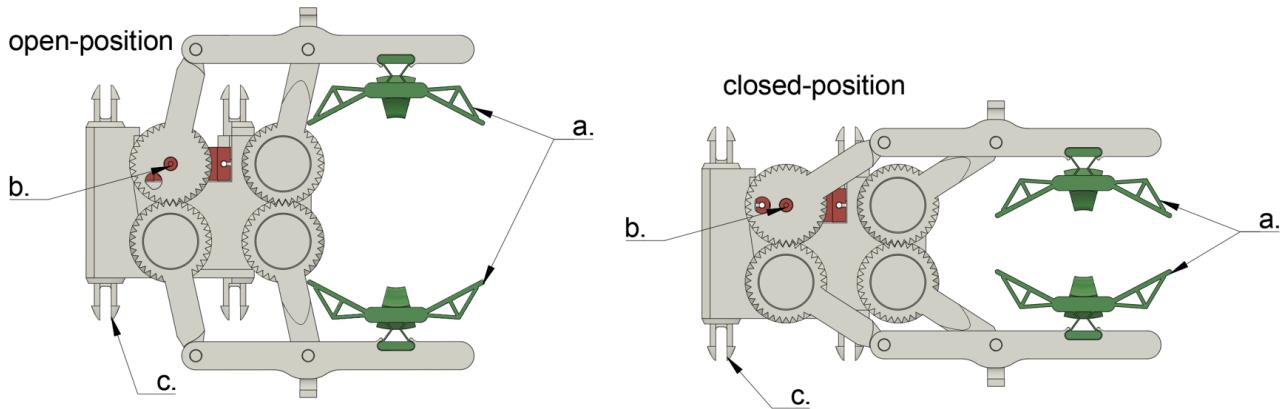


FIGURE 3.6 – Modèle 3D de la deuxième version des bras mécaniques, avec :
a : pinces souples (version 2 comportant un actionneur pour l'haltère)
b : bras motorisé par un servomoteur
c : support du bras

"Grips" ou mors doux

Les grips sont des pièces imprimées en 3D destinées à remplir plusieurs rôles très importants pour le bon fonctionnement du robot, ce sont effectivement eux qui sont l'interface entre le robot et les objets :

- Ne pas abîmer l'objet ou le robot lors de la manipulation, une pièce en matériau trop dur ou abrasive pourrait abîmer les objets manipulés, tandis que s'ils sont trop durs ils pourraient faire forcer les servo et les abîmer, ils doivent donc présenter une certaine déformabilité.
- Accommoder de potentielles erreurs dans le placement du robot, en effet le placement du robot ne peut pas être parfait, une marge d'erreur potentielle arbitraire d'un centimètre est considérée dès le départ pour assurer la résilience du système de détection (c'est à dire appuyer sur les bons micro switchs même si l'objet n'est pas idéalement placé).
- Assurer une prise assez ferme pour ne pas que les objets tombent durant leur déplacement le long du tri.
- Ne pas perturber le dépôt des objets s'ils venaient à devoir être déplacés à nouveau durant le processus de tri.

L'impression 3D de type FDM¹ implique une très grande anisotropie des propriétés de l'objet imprimé, typiquement l'orientation de la surface de dépôt et la structure interne de l'objet jouent

1. Fused Deposition Modelling ou dépôt de filament fondu. Procédé d'impression 3D selon lequel un filament de matière première est liquéfié au moyen d'un corps de chauffe, déposé par une buse d'extrusion suivant le tracé prévu par un logiciel de découpe, et solidifié par refroidissement. (dictionnaire terminologique)

un rôle extrêmement important dans les propriétés de l'objet imprimé, apporter une solution de simulation afin d'obtenir une forme idéale sort largement du domaine du projet. Différentes solutions sont donc apportées au fur et à mesure afin d'obtenir une géométrie satisfaisante. Le groupe est passé par environ une dizaine de variantes différentes dont les différents aspects seront résumés.

Celles-ci présentent quelques points communs :

- Localisation des zones de déformation, les essais présentant des zones de déformation moins bien définies se sont avérées significativement plus sensibles au positionnement initial des objets et avaient en général une mauvaise reproductibilité, le positionnement des objets influant alors plus sur la localisation des déformations les plus grandes ce qui amène à de faux positifs et négatifs.
- Une forme en "V", à des fins de reproductibilité, il est très important que si un objet cylindrique vient à être excentré par rapport à sa position idéale, la fermeture de la pince devrait pouvoir le placer au mieux pour éviter un faux positif. opter pour des grips en "V" est apparu être la solution la plus efficace. L'angle de ce "V" ne doit pour autant pas restreindre l'ouverture de la pince.
- Minimiser les forces d'activation des boutons, un des enjeux les plus cruciaux a été de jouer entre la faible force de fermeture du servomoteur et celle d'activation des micro switchs.

Les évolutions importantes et leurs particularités sont reprises ci-dessous :

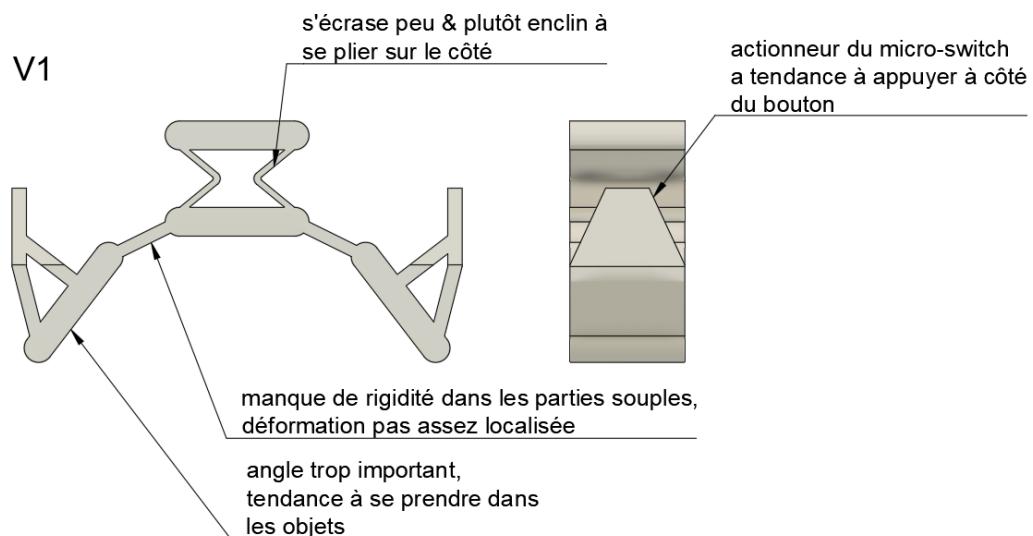


FIGURE 3.7 – Schéma récapitulatif V1

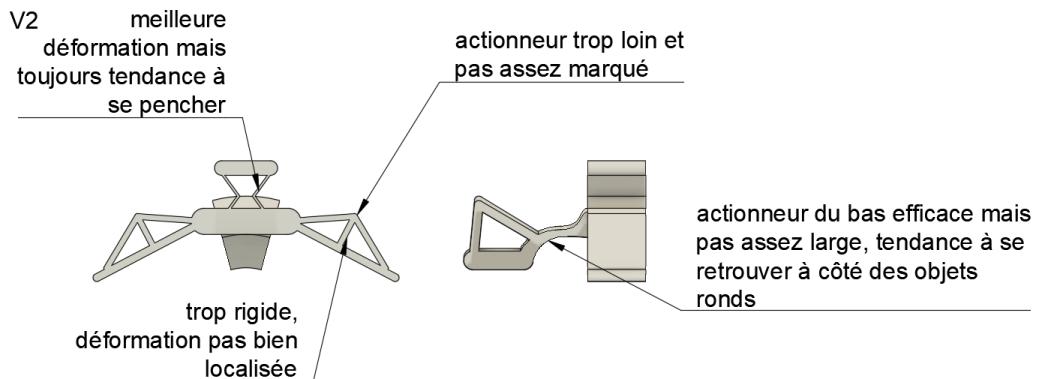


FIGURE 3.8 – Schéma récapitulatif V2

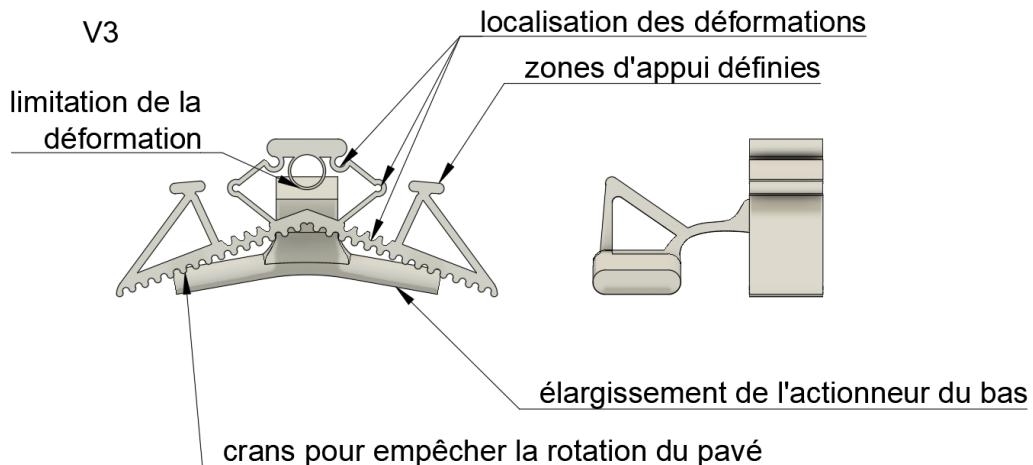


FIGURE 3.9 – Schéma récapitulatif V3

3.4 Composants électroniques

Cette section explique l'utilisation et le fonctionnement des divers composants électroniques utilisés, ainsi que la justification de leur choix.

3.4.1 Description générale du robot

Avant de citer tous les différents composants, il est important de d'abord visualiser l'image générale du robot. Celui-ci est composé d'un châssis sur lequel la plaque Arduino est attachée. Tous les composants sont connectés à cette plaque étant donné qu'elle est elle-même connectée à la batterie qui fournit l'énergie électrique et que l'Arduino comporte le code logique, qui va traiter les informations reçues des capteurs et envoyer les commandes aux moteurs. Un octet de capteurs est disposé sur le bas du robot servant à capter la ligne pour que ce dernier la suive. Les capteurs

ultrasons placés à l'avant permettent au robot de déterminer si un objet se trouve devant lui. Les bras du robot se trouvent aussi à l'avant et portent des micros switchs qui permettent de déterminer quel objet pince le robot. Tous ces capteurs en plus des moteurs (pour faire tourner les roues et faire bouger la pince) et la batterie sont connectés au shield Arduino au moyen de fils électriques.

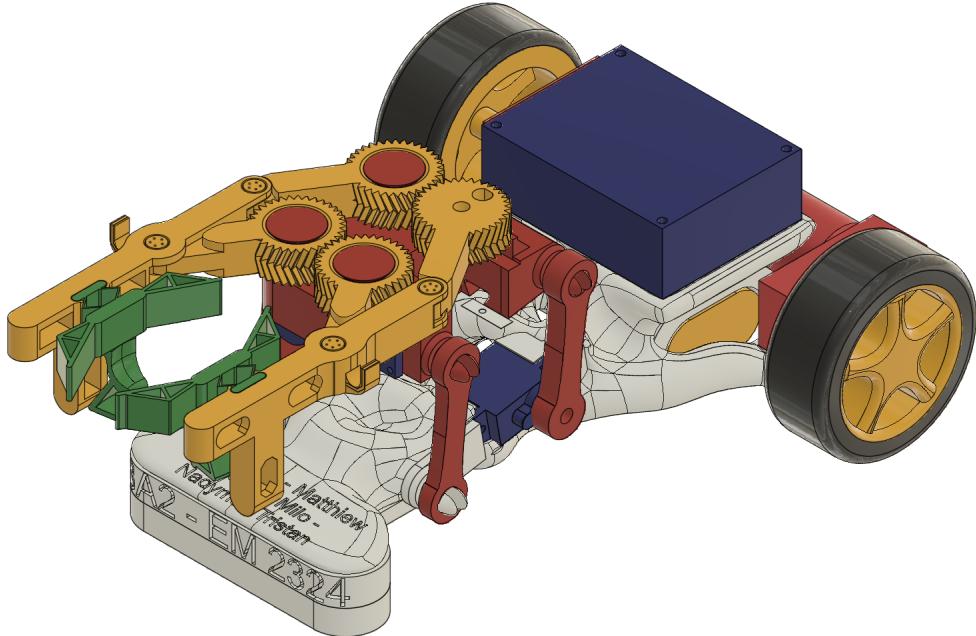


FIGURE 3.10 – Modèle 3D de l'ensemble du robot

3.4.2 Arduino

L'Arduino ([ARD], s.d.) est le composant clé du robot, il s'agit d'un circuit imprimé équipé d'un microcontrôleur qui va stocker le code afin d'envoyer les informations aux différents composants connectés à celui-ci. Il existe plusieurs sortes de cartes Arduino, l'Arduino Nano Every sera utilisé dans le cadre du projet, car il est assez puissant pour effectuer le code et contient assez de ports pour y connecter tous les composants.

Fonctionnement

L'Arduino va récolter des informations via ses capteurs et réagir en envoyant des signaux aux différents moteurs. Les cartes Arduino ont des broches d'entrée/sortie qui permettent de se connecter à des capteurs, des actionneurs et d'autres composants électroniques. ([RMC2], s.d.)

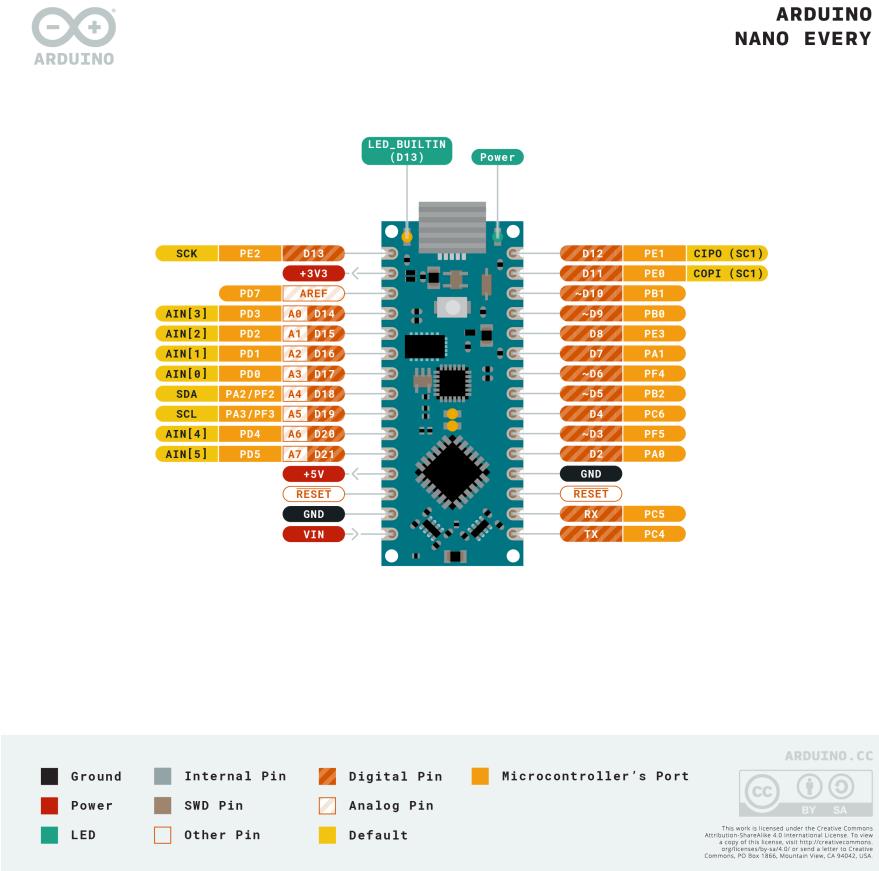


FIGURE 3.11 – Schéma d'un Arduino Nano Every

3.4.3 Shield Arduino Nano Every

Un shield sur lequel l'Arduino sera posé ainsi que le driver permettant le contrôle des moteurs est également fourni. Le shield permet de relier l'Arduino, le driver, ainsi que les différentes bornes sur un circuit imprimé. Il permet une grande aisance et facilité quant à la connexion des câbles, capteurs et autres composants.

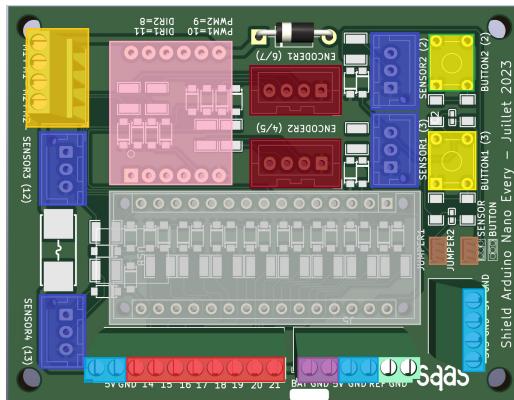


FIGURE 3.12 – Schéma du shield pour Arduino Nano Every distribué dans les ressources du projet([shield] 2023)

L'Arduino Nano Every est une carte de développement compacte offrant une polyvalence grâce à sa petite taille, ses nombreuses broches d'entrée/sortie, et une interface USB intégrée pour une programmation pratique.

16 pins de la plaque Arduino sont utilisés sur les 26 présents.

3.4.4 Servomoteurs

Un servomoteur se compose d'un moteur associé à un circuit de contrôle qui régule la vitesse, la direction et la position du moteur. Deux servomoteurs permettent l'utilisation de la pince servant à capter les objets. La sollicitation de servomoteurs (à la place de moteurs) se justifie par leur précision et leur capacité à effectuer une rotation avec un couple de forces permettant de soulever l'objet. En effet, les bras de la pince ont besoin de tourner avec un angle bien précis pour pouvoir reconnaître et attraper les objets et ce sont les servomoteurs qui permettent de remplir ce rôle au mieux. ([SM-S] 2021) ([Kumar] 2020) ([serv1], s.d.)



FIGURE 3.13 – Image de servomoteur provenant de Amazon

3.4.5 Micro switch

Lors des recherches sur les manières dont le robot pourrait reconnaître les objets, il a été trouvé judicieux de choisir des micros switchs (3.14), que ce soit pour le coût ou la simplicité d'implémentation. Dans le cadre de ce projet, les micros switchs étaient de parfaits candidats.

Une lame métallique se retrouve en position de repos contre un bouton, lorsqu'une pression est exercée contre la lame, la lame vient appuyer sur le bouton ce qui envoie une information. Par un jeu de positionnement des micros switchs sur les bras mécaniques, il est possible de déterminer le type d'objets que le robot tient dans ses bras.



FIGURE 3.14 – Micro switch

Farnell

3.4.6 Capteur de distance à ultrason

Le capteur de distance à ultrason sera utilisé pour détecter les différents objets que ce robot devra intercepter. La portée de ce capteur est de 2 à 400 cm. ([URM], s.d.) Ce capteur est facile d'utilisation, il n'est pas cher et il est relativement précis compte tenu de la fonction qui lui est destinée. Ces différentes qualités sont les raisons pour lesquelles le groupe a opté pour ce capteur. ([S2D120], s.d.)



FIGURE 3.15 – Capteur de distance à ultrason HC-SR04

HC-SR04

3.4.7 Capteur infrarouge

En prenant comme source des robots suiveurs de ligne déjà réalisés ([GTAad] 2021), l'utilisation de capteurs infrarouges a été directement priorisée. Comme capteurs infrarouges servant à déterminer la bande noire, une plaque de série de huit capteurs a été utilisée après en avoir utilisé une de trois capteurs accompagnée de deux autres capteurs infrarouges permettant de tourner. Le changement de capteurs a déjà été expliqué précédemment. La plaque de huit capteurs en question a été fournie par le Fablab. Ce type de capteur a une portée de 3 à 6 mm.

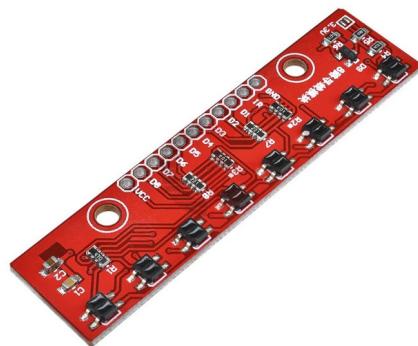


FIGURE 3.16 – Plaque contenant 8 capteurs infrarouges analogiques

8LineIR

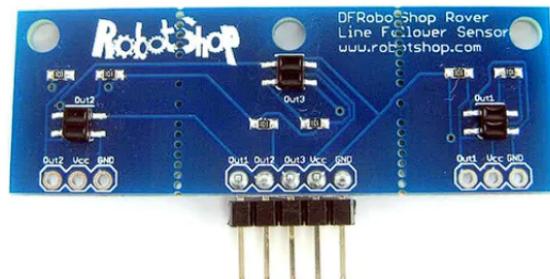


FIGURE 3.17 – Plaque de 3 capteurs infrarouges utilisée pour suivre la ligne dans la première version du robot

. CapteurIR3



FIGURE 3.18 – Un des deux capteurs infrarouges permettant de tourner dans la première version du robot

CapteurIR

3.4.8 Moteur à courant continu

Choix du type de moteur

Les moteurs utilisés pour le déplacement du robot sont des moteurs à courant continu (DC). Ces moteurs ont un meilleur contrôle de la vitesse, ce qui est idéal en robotique, par rapport aux moteurs à courant alternatif. Ils ne sont également pas limités dans leur rotation comme c'est le cas pour les servomoteurs. Les moteurs choisis ont un faible coût et un couple suffisant pour faire déplacer le robot. Ceux-ci pouvant supporter une tension de 3 à 6 volts, ils sont idéaux à utiliser dans ce projet. En effet, la tension nécessaire pour les alimenter n'étant pas trop élevée, il sera facile de trouver des batteries adaptées.([easytech], s.d.)

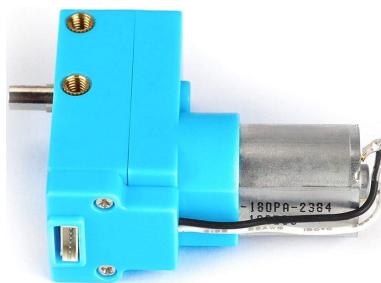


FIGURE 3.19 – Moteur à courant continu

MotorE

Ce moteur a été choisi car la puissance fournie par celui-ci (3,7 Watts) est amplement suffisante pour déplacer ce robot à la vitesse voulue. En effet, le couple nominal (0,800 kg.cm) fourni par ce moteur est capable de faire tourner les roues sous la contrainte du poids du robot et de l'éventuel objet récupéré. La vitesse de rotation de 14.000 tours par minutes subvient à la gamme de vitesse de déplacement du robot désirée.

3.5 Implémentation des moteurs

Deux moteurs ont été installés, un à chaque roue car le robot a besoin de deux roues pouvant tourner indépendamment l'une de l'autre pour pouvoir effectuer des virages. Le choix des moteurs quant à lui est justifié par le fait qu'ils ont été disponibles dès le commencement du projet et qu'ils répondent aux besoins du robot. Ces moteurs sont connectés à l'Arduino par l'intermédiaire du driver moteur, qui transmet les informations nécessaires pour déterminer la direction dans laquelle ils doivent tourner. Cela dépend de l'instruction donnée, que ce soit avancer, reculer, tourner ou s'arrêter. Lorsqu'une rotation sur place est souhaitée, il suffit d'activer les deux moteurs à une vitesse identique mais dans des directions opposées, en fonction du côté vers lequel il faut orienter le robot.

4

Modélisation mathématique

4.1 Équations mécaniques

Dans cette section, l'attention sera portée sur le mouvement mécanique qu'effectuera le robot ainsi que les équations qui le décrivent. À son départ, le robot commence sa course en MRUA. L'équation du MRUA est donnée par :

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2 \quad (4.1)$$

Où :

$x(t)$ est la position à l'instant t ,

x_0 est la position initiale,

v_0 est la vitesse initiale,

a est l'accélération,

t est le temps.

Une fois la vitesse désirée atteinte, le robot avance suivant un MRU. L'équation du MRU est donnée par :

$$x(t) = x_0 + v t \quad (4.2)$$

Où :

- $x(t)$ est la position à l'instant t ,
- x_0 est la position initiale,
- v est la vitesse constante,
- t est le temps.

4.2 Régulation PID

4.2.1 Principes de l'automatisation

L'automatisation permet à un système d'accomplir des tâches manuelles par lui-même et de décider comment réguler la tâche à faire suivant une certaine démarche. Ainsi, en utilisant des capteurs pour évaluer la situation présente, les données sont ensuite transmises à un régulateur qui prendra la décision d'agir afin de corriger l'erreur et ramener le système à l'état souhaité. Cette action est donc commandée à des actionneurs qui vont effectuer le travail nécessaire. Cette démarche doit permettre au système de revenir à la situation voulue malgré des perturbations prévisibles.

Initialement, un système qui corrige les erreurs en se basant seulement sur la position réelle du robot par rapport à la ligne était utilisé. 3 capteurs infrarouges noir/blanc ont été placés entre les 2 roues du robot. Ceux-ci permettent de garder le capteur central sur la ligne noire. En effet, lorsque le capteur de gauche touche la ligne noire, le robot est en train de dévier vers la droite, le système de correction exécute donc un tournage vers la gauche en accélérant le moteur de droite. Ce système agit tant que l'autre capteur de correction n'a pas été activé, ce qui engendre un mouvement de zigzag inévitable. Ce système fonctionne bien pendant un certain temps, mais déviant de la ligne sans cesse, il augmente de plus en plus l'erreur et finit par complètement quitter la ligne. En plus du fait que le robot n'a pas la capacité de faire la différence entre une grande et une petite déviation, le système de correction applique la même accélération à toute déviation, ce qui cause le dérapage complet par rapport à la ligne.([PID], s.d.)

4.2.2 Proportionnel Intégral Dérivée

Une démarche intéressante serait maintenant de se questionner sur la façon de corriger l'erreur du régulateur.

Une première approche pour déterminer une stratégie est celle du régulateur en tout ou rien. Elle consiste à demander à l'actionneur de soit marcher soit s'arrêter. Le souci avec cette approche est qu'elle ne permet pas à l'actionneur de doser. Cette stratégie pourrait fonctionner si une hystérèse était employée, c'est-à-dire faire marcher l'actionneur plus qu'il ne faut, puis l'éteindre et attendre jusqu'à ce qu'il faille le rallumer. Cependant l'hystérèse n'est pas utile dans le cas de ce robot.

Une deuxième approche consisterait à calculer l'erreur et à la corriger en fonction de sa valeur, c'est-à-dire plus l'erreur est grande plus elle est corrigée. Cette technique permet à l'actionneur de fonctionner de manière continue. Cependant, le résultat recherché ne sera jamais obtenu car si l'erreur vaut 0, l'actionneur va s'arrêter étant donné que c'est une proportion. Cette approche pourrait être utile dans ce projet puisqu'elle permettrait au robot de ne pas trop s'écartez de la ligne.

À présent, la régulation reste à améliorer. Le moyen qui va être utilisé pour atteindre la valeur voulue (que l'erreur vaille 0) est l'action par intégration. Celle-ci permettra de minimiser l'erreur et de la faire tendre vers 0. Le souci est que lorsqu'elle atteint la valeur voulue, elle la dépasse légèrement avant de reprendre cette valeur et la garder.

Pour éviter ce dépassement, une dérivée va être utilisée, qui va permettre de prédire le comportement de la fonction (la fonction d'erreur ici n'est pas connue à l'avance, elle se construit durant le parcours du robot).

Ainsi est obtenue une formule comprenant 3 termes (une proportion, une intégrale et une dérivée) qui réguleront l'erreur du robot.

$$U = K_e e + K_l \int_0^t e(t) dt + K_d \frac{de}{dt}$$

La méthode PID consiste à avoir un système de correction qui se base sur l'erreur de la position, de l'intégrale de cette erreur et de sa dérivée. La dépendance en position est évidente, c'est ce qui a déjà été mis en œuvre. La dépendance en l'intégrale de l'erreur permet de doser l'accélération en fonction de l'amplitude de l'erreur. Enfin, la dépendance en la dérivée permet de prédire l'erreur et de commencer l'accélération corrective avant même la détection de l'erreur. Pour mettre cette méthode en place il faut avoir une courbe qui représente l'erreur. Pour ce faire il faut poser une courbe de vitesse idéale des moteurs et prendre des mesures de position des moteurs durant leur fonctionnement pour pouvoir obtenir une courbe d'erreur qui sera la différence entre l'intégrale de la courbe de vitesse idéale et la courbe de position réelle. Cependant, par manque de temps, les mesures sur les moteurs n'ont pas pu être prises.

La régulation PID, dans le cadre de ce projet, va permettre au robot de ne pas dévier de la ligne.

Les variables de la régulation seront les valeurs renvoyées par les capteurs infrarouges analogiques suiveurs de ligne. L'erreur (notée e dans l'équation) représente la déviation du robot qui sera renseignée par les capteurs.

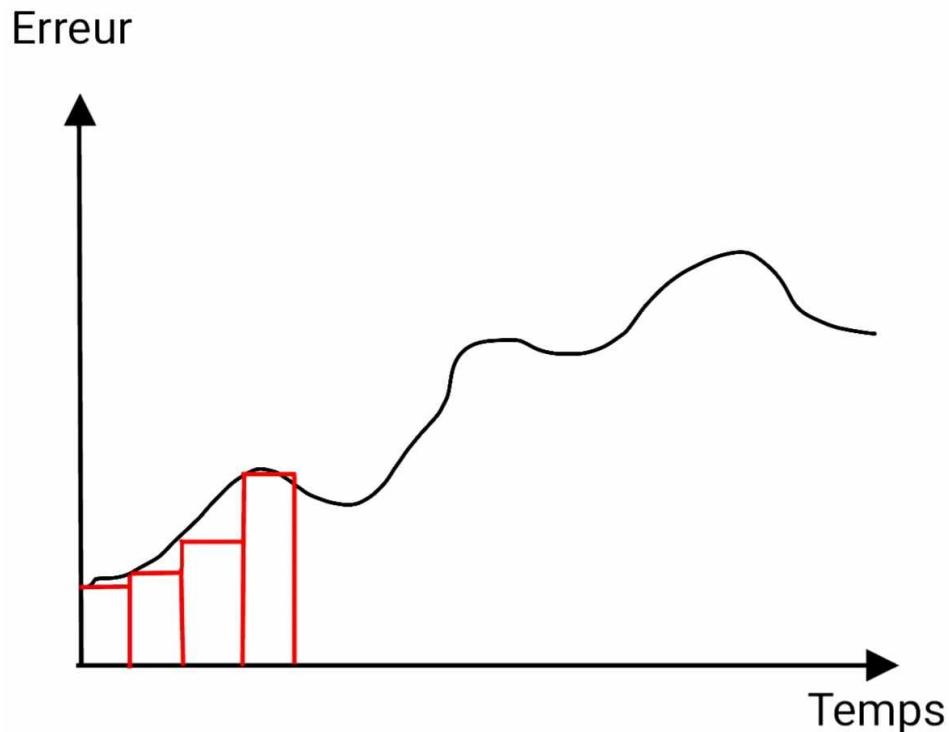


FIGURE 4.1 – Exemple de graphique de l'erreur en fonction du temps, les rectangles rouges représentant l'intégrale approchée de l'erreur

La plupart des informations concernant la régulation PID proviennent du séminaire Modélisation-Régulation donné par Laurent Catoire.

5.1 Algorithme

5.1.1 Introduction

Le code constitue une partie centrale de ce projet. En effet, le robot doit être complètement autonome. Il doit donc être capable de se déplacer en suivant la ligne noire, savoir où il se trouve, où il se rend, où sont placés les objets, où il faut les placer correctement, et dans quel ordre. Le code a été séparé en plusieurs parties : la simulation, incluant toute la partie logique, et une partie déplacement, incluant surtout les fonctions motrices du code, ainsi que la détection de lignes.

5.1.2 Code python

Une simulation du déplacement du robot sur la carte a d'abord été effectuée en Python, afin d'obtenir une première structure du code final qui sera lui en C++. Pour simuler la situation réelle, une nouvelle carte (l'emplacement des objets à trier) sera générée à chaque lancement du code. Ceci a été fait très facilement grâce au module random déjà présent dans les bibliothèques Pycharm.

Le fonctionnement basique du déplacement du robot se décompose en plusieurs phases dont la première est la suivante : celui-ci va d'abord faire un premier aller en passant par toutes les intersections. Cette première étape vise à vérifier s'il y a un objet ou pas à chaque croisement grâce aux capteurs de distance, sans néanmoins s'engager dans les différentes allées pour découvrir de quel objet il s'agit. Le premier chemin suivi par le robot est représenté ci-dessous en rouge.

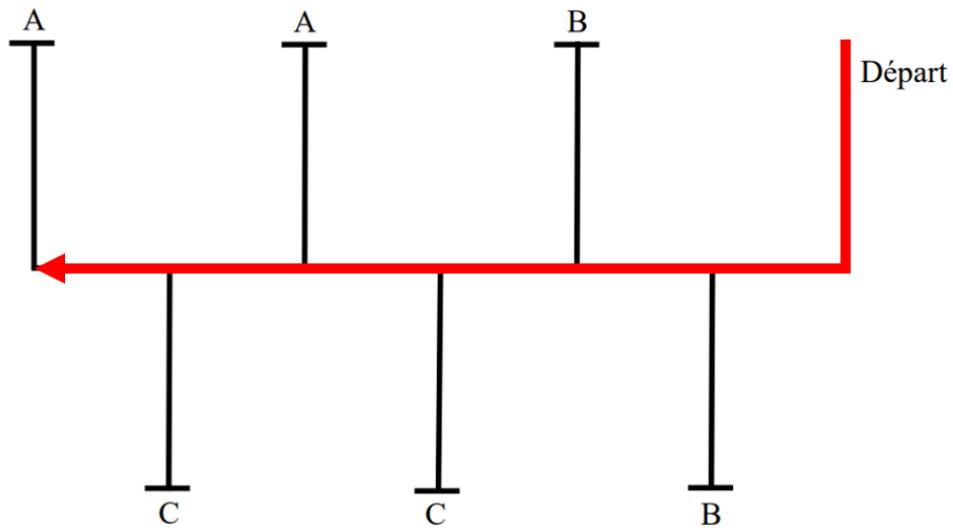


FIGURE 5.1 – Trajectoire du robot lors du premier aller

Comme mentionné précédemment, ceci est possible grâce aux capteurs de distance ultrasoniques, qui sont décrits dans la section dédiée 3.4.6. Une fois que le robot connaît l'état de chaque emplacement (vide ou rempli), il commence à trier les objets. Il se rend dans chaque allée qu'il sait remplie, et vérifie le type d'objet qui s'y trouve (grâce aux capteurs sur les bras du robot). Il décide ensuite s'il faut déplacer l'objet en main, ou le reposer (s'il est au bon endroit ou si les 2 emplacements possibles pour cet objet sont déjà occupés). Tous les objets sont stockés dans une liste, qui se met à jour au fur et à mesure que les colis sont déplacés. Ce code se répète en boucle jusqu'à ce que tous les objets soient bien placés.

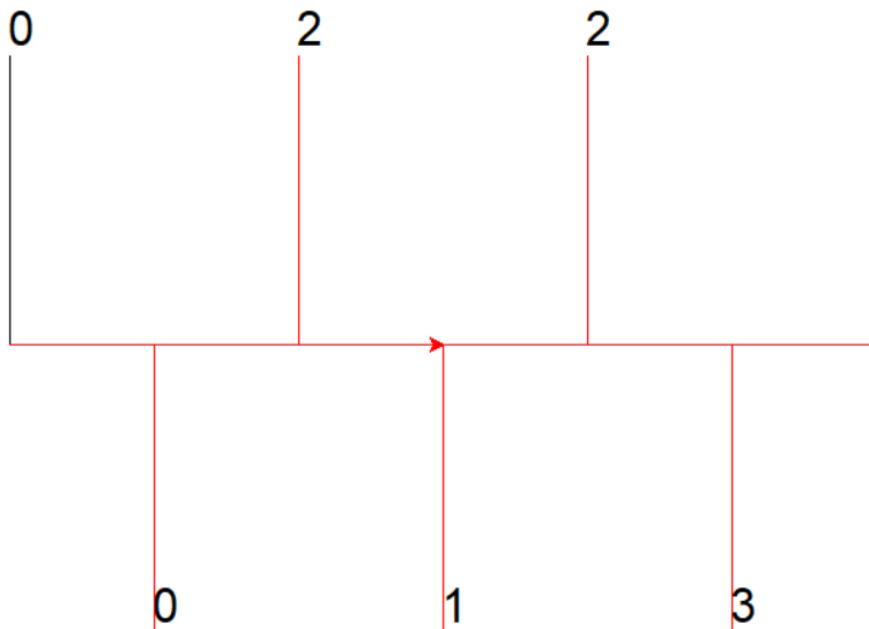


FIGURE 5.2 – Exemple de carte parcourue par le robot

Une des grandes difficultés de ce code est d'optimiser les déplacements du robot. Celui-ci doit trier les objets en le moins de temps possible. Pour cela, le code doit minimiser le chemin parcouru par le robot, par exemple en évitant de faire des allers-retours inutiles, ou en ayant les mains vides. Afin de visualiser les déplacements du robot sur la carte, le module Turtle a été utilisé dans un premier temps. Le code a par la suite dû être légèrement modifié afin de pouvoir l'intégrer dans l'Arduino. Il a fallu notamment enlever la génération aléatoire des objets (étant donné que le robot ignorera tout de la carte avant de commencer à se déplacer), ainsi que la partie animation utilisant le module Turtle.

Un code a également été écrit afin d'identifier et de régler les cas limites (edge cases). Ce code consistait simplement à créer toutes les permutations d'objets possibles respectant le cahier des charges, et de tester l'algorithme de logique effectué précédemment, afin de voir si celui-ci fonctionne de manière optimale pour tous les cas possibles. La distribution du nombre d'étapes (incluant le déplacement d'intersection en intersection, les virages, le ramassage et la dépose) pour effectuer le tri en entier est représentée dans le graphique ci-dessous, qui s'apparente à une loi normale de moyenne 72. L'axe vertical représente donc le nombre de combinaisons d'objets possible associés à chaque nombre d'étapes pour résoudre ces dernières, qui est indiqué sur l'axe des abscisses.

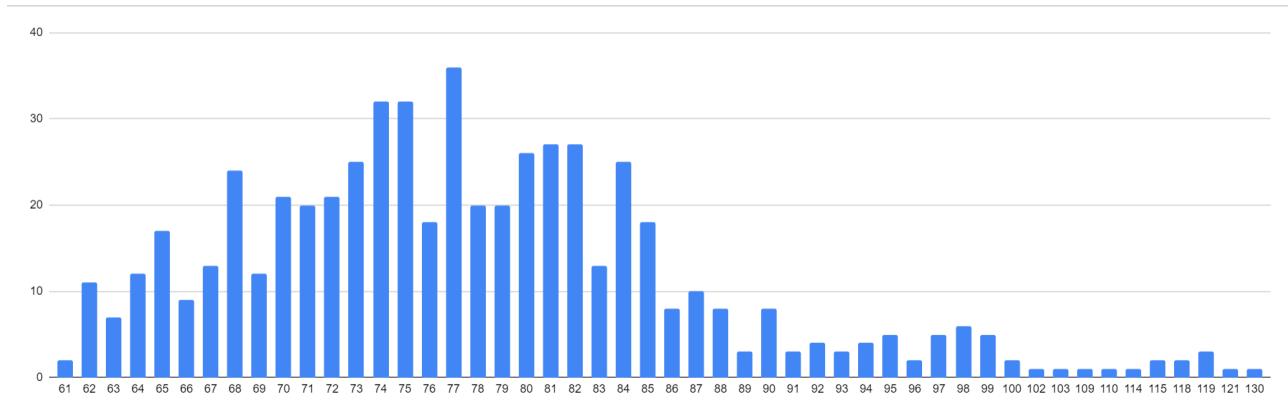


FIGURE 5.3 – Distribution nombre d'étapes pour atteindre l'état rangé

5.1.3 Optimisations

Afin d'obtenir le code optimal, une première version du code a été modifiée, en l'optimisant peu à peu, en ajoutant des nouvelles fonctionnalités en analysant les edge cases. Cependant, il existe plusieurs moyens de comparer les différentes mises à jour de l'algorithme. En effet, il est possible de confronter les moyennes du nombre d'étapes, ou les moyennes du temps écoulé pour trier tous les objets. Pour faire le lien entre les deux, des coefficients ont été utilisés pour simuler le temps

mis pour effectuer certaines étapes. Par exemple, un virage de 90 degrés ne prend pas autant de temps qu'un déplacement rectiligne de 40 cm, ou qu'un ramassage d'objet. Ces coefficients ont été mesurés expérimentalement, en faisant faire au prototype les différentes actions.

Tout d'abord, en utilisant les contraintes du cahier des charges, une première optimisation a paru évidente. Si le robot a déjà reconnu 3 objets, de type AAB (objets quelconques mais deux du même type) il sait que le dernier objet se trouvant sur la carte est le dernier type d'objet C. Ceci évite au robot d'aller vérifier de quel type d'objet il s'agit, s'il ne peut pas directement le livrer au bon emplacement. Cette optimisation a diminué de 2 étapes la moyenne de toutes les permutations, la faisant passer à 70 étapes.

Ensuite, en passant en revue les permutations qui posaient problème, il a été observé que le robot faisait parfois certaines étapes inutiles lors de la phase de repérage. Le robot se rendait aux dernières intersections afin de déterminer quels objets s'y trouvaient, alors qu'il avait trouvé 4 objets sur les premiers emplacements qu'il avait scannés. Une condition a donc été rajoutée pour qu'il n'aille pas jusqu'au bout de la carte, simplement en utilisant le fait qu'il y a toujours 4 objets sur le circuit. Cette deuxième optimisation a fait passer la moyenne de 70 à 68 étapes.

Après ces optimisations, le cas limite en terme d'étapes est la permutation "230101", indiquée sur le schéma ci-dessous. Le nombre d'étapes pour trier cet arrangement d'objets est de 104.

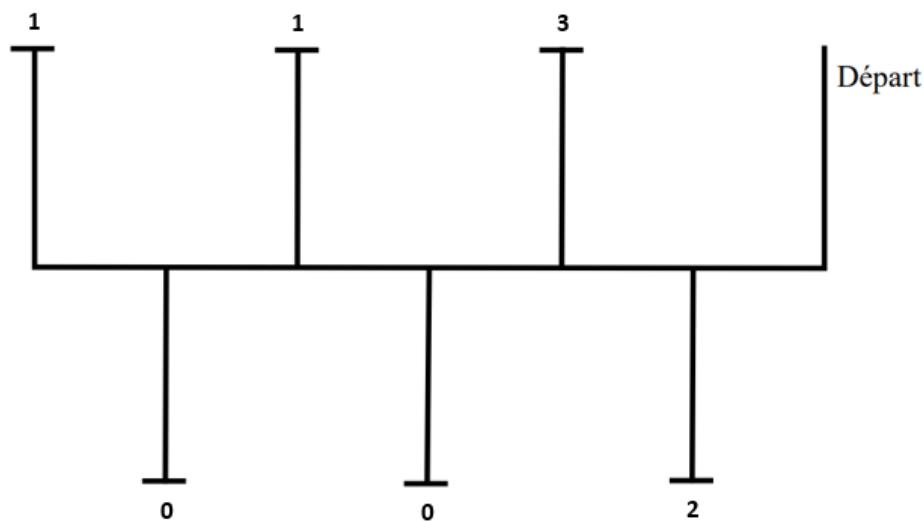


FIGURE 5.4 – Edge Case (230101) après la deuxième optimisation

Pour ce qui est de l'optimisation temporelle du code pour toutes les permutations possibles, la moyenne passe de 257 secondes à 215 secondes, et sa distribution est fortement réduite. Ceci est illustré grâce au graphique ci-dessous, avec en rouge la distribution du deuxième code optimisé. L'axe des ordonnées représente le nombre de permutations résolvant le parcours en un même

temps, représenté sur l'axe horizontal. Les coefficients ont été déterminés expérimentalement afin d'avoir une simulation proche du réel.

Pickup = Drop = 2 secondes 90 degrés = 3 secondes Avancer 10 cm = 1 seconde

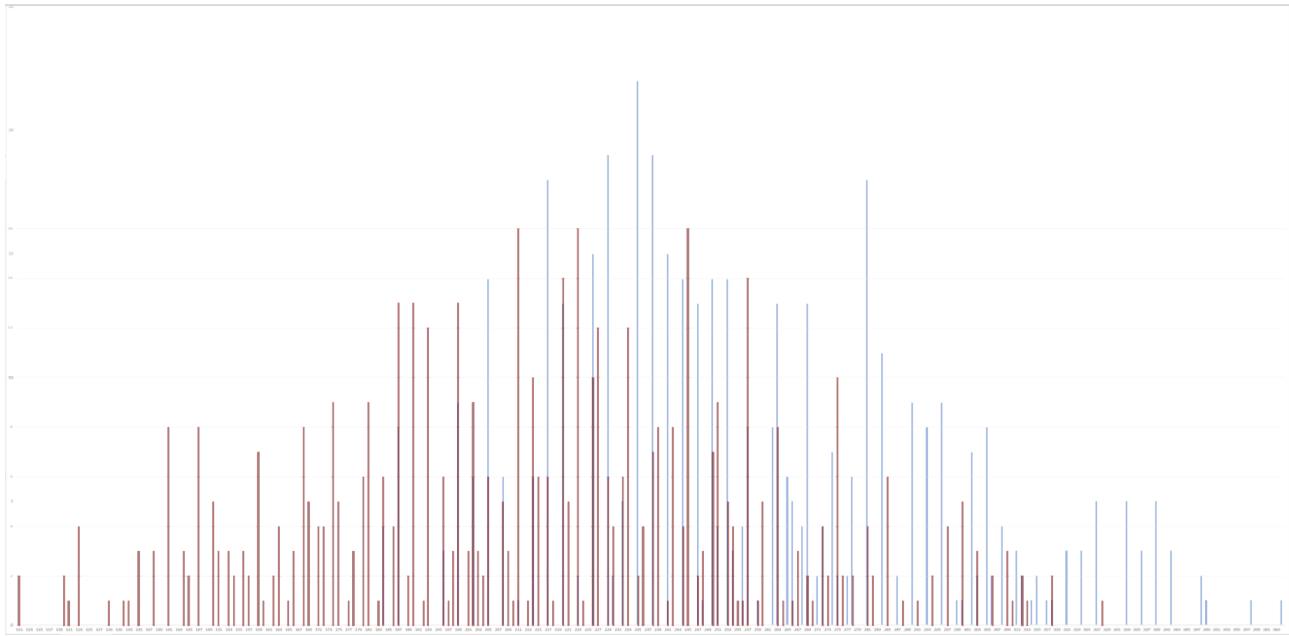


FIGURE 5.5 – Distribution de probabilité du temps mis pour résoudre les parcours, avec :
 Rouge : Optimisé, moyenne = 215, minimum = 111, maximum = 328
 Bleu : Non-optimisé, moyenne = 257, minimum = 184, maximum = 364

Ces données représentées graphiquement à la figure 5.5 proviennent directement d'un code qui a testé toutes les permutations possibles et mesuré le temps mis par le robot pour les résoudre, en utilisant les coefficients mentionnés précédemment pour simuler le temps d'exécution réel. La version optimisée en rouge est clairement translatée vers la gauche comparée à la version non optimisée en bleu. Une amélioration très conséquente a également eu lieu pour ce qui est du temps mis pour résoudre le cas limite. En effet, ce temps est réduit d'environ 36 secondes entre les deux versions du code.

5.1.4 Code Arduino

Le code mécanique du robot est effectué à l'aide du logiciel IDE Arduino, l'environnement de développement de la carte Arduino. Il s'agit d'un logiciel qui permet d'écrire, de compiler et de télécharger le code sur la carte Arduino. Il est basé sur le langage de programmation C/C++.

La première étape du code a été donc de permettre au robot à suivre la bande noire. Pour effectuer cela les huit capteurs infrarouges analogiques du centre captent le signal (INPUT) pour

l'envoyer à l'Arduino qui va activer les moteurs (OUTPUT).

Un premier code se basant sur ([GTAad] 2021) a été utilisé. Pour ce faire une plaque de trois capteurs infrarouge a été utilisée proche de l'axe des roues au centre de rotation du robot, ainsi que deux capteurs extrêmes pour détecter les intersections. Cet emplacement ainsi que la méthode numérique des capteurs (1 ou 0) n'était pas adéquat.

Ce tableau reprend la logique du code en fonction des valeurs retournées par les capteurs. En considérant que 0 représente une détection de blanc et 1 une détection de noir.

Capteur gauche	Capteur droit	Résultat
1	1	S'arrête
0	0	Avance tout droit
1	0	Tourne à gauche en actionnant la roue de droite
0	1	Tourne à droite en actionnant la roue de gauche

Capteur extrême gauche	Capteur extrême droit	Résultat
1	1	S'arrête
1	0	Virage sur place de 90 degrés vers la gauche
0	1	Virage sur place de 90 degrés vers la droite

Ce code simpliste permettant de suivre simplement une ligne droite n'était pas fonctionnel de par l'emplacement des capteurs et d'une autre part à cause de l'écart trop important entre les capteurs de la plaque.

Ainsi l'utilisation de la rangée de huit capteurs a été décidée, l'avantage de celle-ci étant que les capteurs sont plus nombreux et plus resserrés, ce qui permet de détecter plus rapidement si le robot commence à quitter la ligne. Ces capteurs sont placés tout à l'avant du robot (expliqué à 3.2) et la logique du code a donc dû être modifiée en optant pour la méthode PID inspirée de ([PID], s.d.). Cette méthode va permettre un suivi de ligne beaucoup plus fiable et plus fluide.

Application de la méthode PID 4.2

Afin d'appliquer la méthode PID, il a fallu changer les capteurs à des capteurs analogiques dans le but d'avoir un continuum de valeurs pour la régulation PID. Parmi les huit capteurs, les quatre capteurs centraux s'occupent du suivi de ligne, et les deux extrêmes de la détection de l'intersection, les deux restants sont inutilisés. La logique du code est la suivante : Le robot commence par

effectuer un tour sur lui-même afin de calibrer ses capteurs analogiques en mesurant un panel de valeur parmi la ligne noire et son environnement dont le but est d'obtenir la valeur maximale et minimale des valeurs captées. Avec cela le robot est aisément capable de savoir s'il se trouve sur une ligne noire, celle-ci ayant une valeur proche de la maximale. Après cette brève phase de calibrage, le robot commence le suivi de ligne. Afin dès lors de ne pas permettre une déviation du robot de la ligne une erreur est mesurée par la différence de valeur de la mesure des deux capteurs gauches et droites. Cette erreur est alors distribuée à 3 variables (P,I,D) comme suit :

$$P = \text{erreur}$$

$$I = I + \text{erreur}$$

$$D = \text{erreur} - \text{erreurPrecedente}$$

Ces 3 variables : P, I, D, représente respectivement les termes Proportionnel, Intégrale et Dérivée. Ces termes seront chacune multiplié par une constante. Ces constantes ont été trouvées expérimentalement en observant le comportement du robot. Ainsi la somme de l'ensemble de ces termes constitue la valeur du PID, c'est cette valeur que l'on va retirer à la valeur de la vitesse d'un moteur et la rajouter à celle de l'autre moteur.

$$\text{PIDvalue} = (Kp * P) + (Ki * I) + (Kd * D)$$

Finalement si un des deux capteurs extrêmes détecte une intersection, le robot va soit la passer, soit prendre l'intersection en avançant le robot jusqu'à son centre de rotation par un délai pour ensuite effectuer un virage sur place de 90 degrés.

6.1 Gestion d'équipe

Dans le but de gérer l'équipe au mieux lors de ce projet, certains membres de l'équipe ont pu endosser le rôle de chef de projet. Son rôle a plusieurs facettes qui seront détaillées lors de ce chapitre. Afin de maintenir une bonne organisation tout au long du projet, plusieurs outils ont été utilisés.

Pour rester au courant de l'avancée des tâches, et de la répartition des rôles, le logiciel Gantt a été utilisé, permettant au chef de projet d'organiser et de tenir informé le reste du groupe de l'évolution des tâches, et des différents jalons. Généralement, les tâches étaient partagées par 2 voire 3 personnes maximum (afin de ne pas être trop à avoir la même mission), selon l'urgence de celle-ci, ou la charge de travail nécessaire pour l'accomplir.

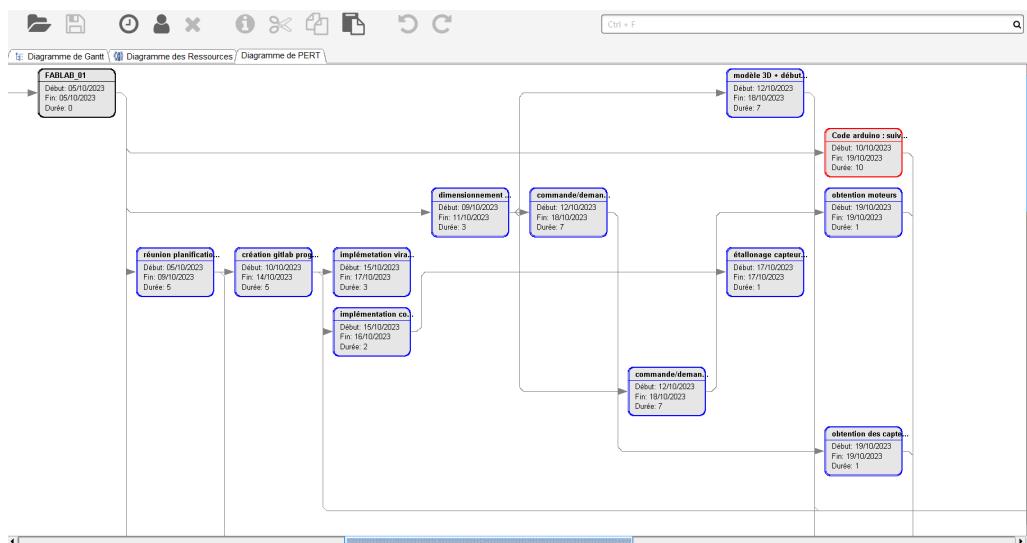


FIGURE 6.1 – Partie du Gantt affichant visuellement les liens entre les tâches

Afin de garder une trace écrite de l'avancée du projet au fil des semaines, des Status Report ont été effectués à chaque réunion, dans le but de renseigner un superviseur de projet, ou un membre

de l'équipe n'ayant pas pu venir. Ceux-ci permettent également de se rendre compte rétrospectivement au fil du projet si les différentes deadlines étaient respectées, afin de s'adapter pour les prochaines échéances. Un autre instrument qui a été très utile pour le bon fonctionnement de ce projet était GitLab. Ce site Internet permet à tout le groupe de se partager des documents de manière très ergonomique, et sécurisée dans le sens où l'on ne risque pas de perdre des documents en les écrasant involontairement, grâce au système collaboratif de "commit".

6.1.1 SWOT

Un tableau SWOT, (Strengths, Weaknesses, Opportunities, Threats), ou en français : Forces, Faiblesses, Opportunités, et Menaces, a également été effectué. Celui-ci a été réalisé par le groupe dans son entièreté afin de rendre compte des points forts et faibles de l'équipe, et comment y remédier au plus vite pour assurer le bon déroulement du projet. Ce dernier a évolué et a été mis à jour tout au long de l'année.

Q1	Strengths	Weaknesses	Opportunities	Threats
	Modélisation 3D	Communication	Formation Impression 3D	Etudes chronophages
	Python	Manque d'investissement	Formation PID	Deadlines
	Travaux manuels	Ponctualité	Formation découpe Laser	Bugs informatiques
Q2	Strengths	Weaknesses	Opportunities	Threats
	Bonne entente	Organisation	Matériel chez soi	Problèmes d'imprimantes
	C++	Partage de connaissances	Echanges avec autres groupes	Conflits horaires

Les plus grosses forces de ce groupe qui ont été identifiées sont la modélisation 3D, ainsi que le code, tant en Python qu'en C++, compatible avec Arduino.

Pour ce qui est des faiblesses, celles qui ont été les plus impactantes sont le manque de communication, de ponctualité, et plus généralement d'investissement de la part de certains membres du groupe.

Les opportunités les plus importantes qui ont pu être saisies par le groupe ont notamment été les diverses formations disponibles au Fablab, ainsi que les échanges (tant d'idées que de matériaux) avec d'autres groupes.

Enfin, certains éléments limitants qui ont ralenti le bon déroulement du projet sont notamment des études très demandantes en terme de temps, et énormément de problèmes d'impressions, qui n'étaient pas du ressort du groupe.

6.2 Budget et dépenses

Voici le tableau récapitulatif des dépenses effectuées pour la conception du prototype actuel.

Catégorie	Quantité	Prix/Unité	Prix total
Servo SG90	2	3,60 €	7,20 €
Moteur	2	14,40 €	28,80 €
Arduino Nano Every	1	12,50 €	12,50 €
PLA	0,6	19,80 €	11,88 €
Micro switch	3	0,22 €	0,66 €
Batterie 18650	2	1,60 €	3,20 €
Contrôleur moteur	1	5,50 €	5,50 €
			69,74 €

Nonobstant que le budget de 100 euros communiqué dans le cahier des charges lors du lancement du projet soit largement respecté, les dépenses ne sont néanmoins pas exemptes de quelques remarques.

Les moteurs ayant été empruntés au laboratoire responsable du projet en électromécanique, leur prix n'a pas été pris en compte lors de la conception. S'il était question de reproduire ce robot dans un autre contexte, il ne serait pas nécessaire de se procurer des moteurs à encodeur optique, un simple moteur à courant continu avec une réduction similaire remplirait la tâche de façon similaire.

Il est à noter que le prix du Shield Arduino n'apparaît pas ici puisqu'il s'agit d'un composant purement d'interface présent à des fins pédagogiques. Son prix est d'autant plus difficile à estimer qu'il s'agit d'une carte faite en petites quantités uniquement à l'intention des étudiants.

Le prix des câbles, des soudures ou des connecteurs n'ont pas été non plus pris en compte puisque non nécessaires ou trop faibles pour justifier leur présence.

Conclusion

Somme toute, ce projet a permis à tout le groupe d'apprendre énormément de nouvelles compétences à ajouter à l'arsenal de chacun qui seront de grande utilité dans la suite de ce bachelier en études d'ingénieur civil.

D'abord il a fallu construire un prototype fonctionnel, capable de se déplacer en suivant une ligne noire, et de reconnaître et soulever des objets. Pour cela, il a été indispensable de d'abord modéliser celui-ci en 3D. Ceci passe par la conception des bras, du châssis, leur implémentation, ainsi que celle des composants électroniques nécessaires au bon fonctionnement du robot autonome. Pour ces derniers, il a été obligatoire de justifier l'utilisation de chacun, que ce soit par leurs propriétés, leurs prix, ou leurs disponibilités.

Pour obtenir un robot qui suit correctement la ligne noire, sachant qu'il y a toujours des imprécisions dues au sol, à la précision des moteurs, ou à des légers frottements, il a fallu passer par la régulation PID. En effet, pour un problème qui paraît de prime abord assez basique, il a fallu recourir à une méthode quelque peu complexe, nécessitant simulations et manipulations d'équations mathématiques. Cette logique utilise une mesure de l'erreur sur la position du robot (comparée à sa position idéale), et permet de la réduire au maximum, en utilisant un terme Proportionnel, un terme Dérivé, et un terme Intégré.

Ensuite, une partie majeure de ce projet a été la programmation. En effet, en ayant un prototype et une méthode de suivi de ligne fonctionnelle, il faut désormais faire le lien entre les deux. Ceci, et rajouter l'algorithme de logique est possible grâce à la programmation sur Arduino. Celle-ci a dans un premier temps été effectuée sur Python, afin d'obtenir une simulation en 2 dimensions assez réaliste du temps pris au robot afin de trier tous les objets. Ladite simulation permet d'optimiser les déplacements en étudiant les cas limites correspondant aux permutations d'objets les plus

difficiles à trier. Ensuite, cette modélisation a été traduite en C++, afin de l'assigner directement au prototype et d'effectuer des tests, jusqu'au fonctionnement parfait du robot.

Bibliographie

[algos]. *A Survey of Path Planning Algorithms for Mobile Robots* [en eng]. 2021.

Ce document a servi à la planification du déplacement du robot dans le code logique.

[ANTON] (Anton, D M, S E Szabo, C A Mociar et A F Iuhas). 2021. « Line follower mobile robots, prototypes of robots and functional of mobile robots ». *IOP Conference Series : Materials Science and Engineering* 1169, n° 1 (août) : 012043. <https://doi.org/10.1088/1757-899X/1169/1/012043>. <https://dx.doi.org/10.1088/1757-899X/1169/1/012043>.

Cette étude a été choisie pour son intérêt à analyser le fonctionnement des robots suiveurs de ligne ainsi que leur programmation en C++.

[Apb23] (ed). 2021. « Arduino push button - complete tutorial », 23 août 2021. Visité le 7 octobre 2023. <https://roboticsbackend.com/arduino-push-button-tutorial/>.

Cet article traitant le fonctionnement d'une plaque Arduino a permis la compréhension et le bon usage de celui-ci.

[ARD]. « Arduino Every datasheet ». s.d. Visité le 18 octobre 2023. <https://docs.arduino.cc/resources/datasheets/ABX00028-datasheet.pdf>.

Ce document explique le fonctionnement d'une plaque Arduino ce qui est utile pour ce projet étant donné que le robot en est équipé afin d'y implémenter le code logique.

[ARDbat] (Clyde, Cox). 2020. « How to Power an Arduino With a Battery », 13 juin 2020. Visité le 12 octobre 2023. <https://www.circuitbasics.com/how-to-choose-the-right-battery-to-power-up-your-arduino/>.

Cet article a aidé le groupe à fabriquer sa propre batterie.

[ardrob] (Warren, John-David., Josh. Adams et Harald. Molle). 2011. *Arduino Robotics* [en eng]. 1st ed. 2011. Technology in action Arduino robotics. Berkeley, CA : Apress. ISBN : 1-4302-3184-X.

Ce livre traite plusieurs concepts importants à la réalisation d'un robot similaire à celui du projet. C'est pourquoi il a servi de grande source d'inspiration.

[ARDsens] (Cicolani, Jeff.). 2018. *Beginning Robotics with Raspberry Pi and Arduino Using Python and OpenCV* [en eng]. 1st ed. 2018. Berkeley, CA : Apress. ISBN : 1-4842-3462-6.

Cet article a servi à comprendre les bases de l'utilisation de Raspberry Pi et d'Arduino avec Python.

[astar] (Thaddeus Abiy, Hannah Pang, Beakal Tiliksew, and 2 others contributed). *A* Search*. [Online; accessed 8-October-2023]. [https://brilliant.org/wiki/a-star-search/#:~:text=A%20\(pronounced%20as%20%22A,to%20B%20can%20be%20difficult..](https://brilliant.org/wiki/a-star-search/#:~:text=A%20(pronounced%20as%20%22A,to%20B%20can%20be%20difficult..)

Ce document a été utile pour établir le code logique permettant de faire déplacer le robot.

[AVOrob] (Mothe, Rajesh, S. Tharun Reddy, G. Sunil et Chintoju Sidhardha). 2020. « An IoT Based Obstacle Avoidance Robot Using Ultrasonic Sensor and Arduino » [en eng]. *IOP Conference Series : Materials Science and Engineering* (Bristol) 981 (4) : 42002-. ISSN : 1757-8981.

Cet article aborde la conception d'un robot capable d'éviter des obstacles. Il a permis au groupe de comprendre l'utilisation du capteur ultrasonique.

[correl] (Correll, Nikolaus). 2020. *Introduction to Autonomous Robots*. Boulder, Colorado : Nikolaus Correll. ISBN : 978-0-692-70087-7.

Ce livre explique plusieurs concepts nécessaires à la construction d'un robot ce qui est très utile dans le cadre de ce projet.

[DCmtr] (ArduinoGetStarted). s.d. « Arduino - DC Motor | Arduino Tutorial ». Arduino Getting Started. Visité le 12 octobre 2023. <https://arduinogetstarted.com/tutorials/arduino-dc-motor>.

Cet article a permis au groupe de comprendre comment contrôler un moteur à courant continu avec Arduino.

[djik] (Wikipedia contributors). 2023. *Dijkstra's algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-October-2023]. https://en.wikipedia.org/w/index.php?title=Dijkstra%27s_algorithm&oldid=1178891955.

Cet article explique une stratégie d'algorithme permettant de trouver le chemin le plus court entre des noeuds ce qui a été utile dans la conception du code logique qui a pour but de faire déplacer le robot de sorte à accomplir sa fonction le plus vite possible.

[DunPar] (MAYE, Pierre.). 2006. *Moteurs électriques pour la robotique*. 2ème édition. Paris : Dunod. ISBN : 978-2-10-070036-3.

Cette ouvrage a aidé au groupe à choisir le moteur adéquat et à utiliser ses capacités au mieux.

[easytech] (Arie, Benjamin). s.d. « What Are the Different Types of Arduino® Motors? » <https://www.easytechjunkie.com/what-are-the-different-types-of-arduino-motors.htm>.

Cet article informe sur les moteurs à utiliser pour Arduino ce qui est pertinent dans le cadre du projet car le robot est équipé d'une plaque Arduino.

[grid] (Balch, Tucker). s.d. « Gridbased Navigation for Mobile Robots ». 29 : 1-9.

Ce document a été utile pour établir le code logique permettant de faire déplacer le robot.

[GTAad] (GTadmin). 2021. « Réaliser un Robot suiveur de ligne avec une carte Arduino UNO ». Gootrio, 17 octobre 2021. Visité le 1^{er} octobre 2023. <https://gootrio.com/realiser-un-robot-suiveur-de-ligne-avec-une-carte-arduino-uno/>.

Cet article explique le fonctionnement d'un robot suiveur de ligne ce qui a été utile dans la réalisation du robot.

[hobby] (Hobby, SriTu). 2021. « How to make a line follower robot using a 3-way IR infrared sensor module ». SriTu Hobby, 1^{er} avril 2021. Visité le 4 octobre 2023. <https://srituhobby.com/how-to-make-a-line-follower-robot-using-a-3-way-ir-infrared-sensor-module/>.

Cet article propose une stratégie pour avoir un robot suiveur de ligne en utilisant une plaque de 3 capteurs infrarouges. Cette stratégie a été répliquée dans le cadre du projet au Q1 mais a ensuite été abandonné au Q2 car le robot suivait mal la ligne.

[Kumar] (Kumar, Vineet, Veena Sharma, O. P. Rahi et Utsav Kumar). 2020. « Robust Control of Position and Speed for a DC Servomotor System Using Various Control Techniques ». In *Advances in Electrical and Computer Technologies*, sous la direction de Thangaprakash Sengodan, M. Murugappan et Sanjay Misra, 1101-1107. Lecture Notes in Electrical Engineering. Singapore : Springer. ISBN : 9789811555589. https://doi.org/10.1007/978-981-15-5558-9_93.

Cette article donne des informations importantes pour le groupe quant aux stratégies de contrôle utilisées pour le contrôle de position et de vitesse d'un servomoteur à courant continu.

[LegoR] (Koch, Grady). 2020. *The LEGO Arduino cookbook : expanding the realm of MINDSTORMS EV3 invention* [en eng]. 1st ed. 2020. Place of publication not identified : Apress. ISBN : 1-4842-6303-0.

Cet article a servi à comprendre comment le groupe pouvait utiliser MINDSTORMS EV3 avec Arduino.

[lightF]. « Light sensing with the Flying-Fish series from MH ». s.d. <https://www.hackster.io/ingo-lohs/light-sensing-with-the-flying-fish-series-from-mh-0e51ab>.

Cet article a permis de comprendre comment utiliser les capteurs infrarouges Flying Fish-series de chez MH.

[LineFol] (Engineer, Crazy). 2021. « Line Follower Robot : Arduino NANO - QTR-8RC - PID Line Follower Robot V1 ». Arnab Kumar Das, 4 septembre 2021. Visité le 11 octobre 2023. <https://www.arnabkumardas.com/line-follower-robot/arduino-nano-qtr-8rc-line-follower-robot-v1/>.

Cet article a aidé le groupe à comprendre le fonctionnement de la régulation PID.

[PID] (robotjunkies). s.d. « How to code your Line follower robot with PID control and working code !, 2022. » <https://www.youtube.com/watch?v=8Lj5ycrT9Fw..>

Cette vidéo a aidé le groupe à coder une régulation PID pour un robot suiveur de ligne.

[PP01] (Gasparetto, Alessandro, Paolo Boscariol, Albano Lanzutti et Renato Vidoni). 2015. « Path Planning and Trajectory Planning Algorithms : A General Overview ». *Mechanisms and Machine Science* 29 (mars) : 3-27. https://doi.org/10.1007/978-3-319-14705-5_1.

Cet article a été utile pour établir le code logique permettant de suivre la ligne.

[proARD] (Anderson, Rick. et Dan. Cervo). 2013. *Pro Arduino* [en eng]. 1st ed. 2013. Berkeley, CA : Apress. ISBN : 1-4302-3940-9.

Cet article a permis d'approfondir l'utilisation d'Arduino.

[RMC2]. « Robot Modeling and Control, 2nd Edition | Wiley ». Wiley.com. s.d. Visité le 23 septembre 2023. <https://www.wiley.com/en-us/Robot+Modeling+and+Control%2C+2nd+Edition-p-9781119524045>.

Ce document traite plusieurs concepts nécessaires à la réalisation d'un robot similaire à celui du projet, ainsi il a été une grande source d'inspiration.

[Rmotion] (García, Edgar A. Martínez). 2022. *Motion Planning* [en eng]. London : IntechOpen. ISBN : 1-83969-775-X.

Ce document a servi à la planification du déplacement du robot dans le code logique.

[S2D120]. « Data sheet Sharp 2D120X ». s.d. <https://www.pololu.com/file/0J157/GP2D120-DATASHEET.pdf>.

Ce pdf est la datasheet du Sharp 2D120X.

[SERV] (ArduinoGetStarted). s.d. « Arduino - Servo Motor | Arduino Tutorial ». Arduino Getting Started. Visité le 12 octobre 2023. <https://arduinogetstarted.com/tutorials/arduino-servo-motor>.

Cet article a permis au groupe de comprendre comment contrôler un servomoteur avec Arduino.

[serv1]. « Comment choisir un servomoteur | Kollmorgen ». s.d. Visité le 10 octobre 2023. <https://www.kollmorgen.com/fr-fr/blogs/comment-choisir-un-servomoteur>.

Ce site a servi à choisir le servomoteur, il donne des informations sur certains aspects techniques à prendre en compte.

[shield]. « ULB-Polytech / BA2-Projects / EMELEC / Arduino_Shield · GitLab ». GitLab. 2023, 5 octobre 2023. Visité le 14 décembre 2023. https://gitlab.com/ulb-polytech/ba2-projects/emelec/Arduino_Shield.

Ce document a servi à savoir comment utiliser la plaque Arduino.

[SM-S]. « 'Speed Control of the Direct Current Servomotor and the Stepper Motor with Arduino UNO Platform'. » 2021.

Cet article a servi à utiliser un servomoteur de type SM-S2309S à l'aide d'Arduino.

[T-Des] (T., Abdulmuttalib, Fatima R. et Osama T.). 2018. « Design and Construction Objects Store System using Line Follower Robot ». *International Journal of Computer Applications* 181, n° 15 (17 septembre 2018) : 27-35. ISSN : 09758887, visité le 11 octobre 2023. <https://doi.org/10.5120/ijca2018917773>. <http://www.ijcaonline.org/archives/volume181/number15/rashid-2018-ijca-917773.pdf>.

Cet article a été sélectionné car il propose des idées intéressantes pour créer un robot autonome avec Arduino, en utilisant des méthodes nouvelles pour se déplacer et stocker des objets.

[URM]. « how to chose the best URM ultrasonic sensor ». s.d. Webpage. <https://www.dfrobot.com/blog-13411.html#>.

Cet article présente plusieurs capteurs à ultrasons ce qui a permis de choisir le mieux adapté dans le cadre du projet, c'est-à-dire qu'il puisse détecter l'objet à la distance voulue.

[VEX]. « VEX Lift & Claw Design Guide - EG1004 Lab Manual ». s.d. Visité le 23 septembre 2023. https://manual.eg.poly.edu/index.php/VEX_Lift_%26_Claw_Design_Guide.

Cette source décrit une pince permettant de soulever un objet ce qui est pertinent dans le cadre de ce projet car le robot doit être capable de cette fonction.

[visib] (Wang, Paul Keng-Chieh.). 2015. *Visibility-based Optimal Path and Motion Planning* [en eng]. 1st ed. 2015. Studies in Computational Intelligence, 568. Cham : Springer International Publishing. ISBN : 3-319-09779-2.

Ce document a servi à la planification du déplacement du robot dans le code logique.

[WP-ARD] (Louis, Leo). 2016. « Working Principle of Arduino and Using it as a Tool for Study and Research ». *International Journal of Control, Automation, Communication and Systems* 1, n° 2 (30 avril 2016) : 21-29. ISSN : 24557889, visité le 11 octobre 2023. <https://doi.org/10.5121/ijcacs.2016.1203>. <https://airccse.com/ijcacs/papers/1216ijcacs03.pdf>.

Cet article a servi à comprendre le fonctionnement d'Arduino et a aidé le groupe à comprendre son utilisation.

Annexes



Fiche technique shield Arduino nano every

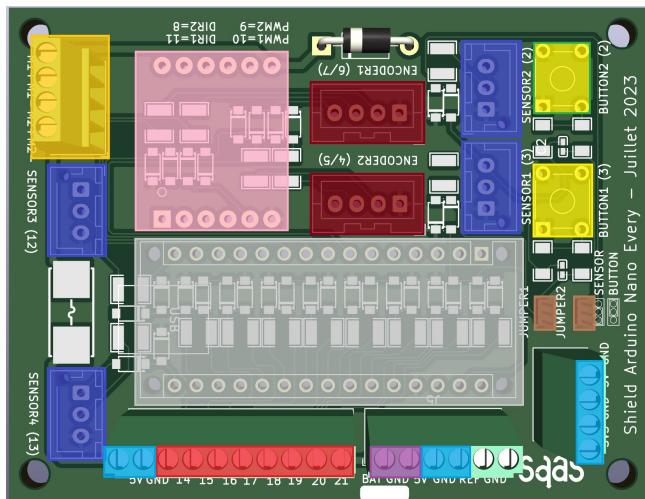


FIGURE A.1 – Schéma d'un shield pour arduino nano every

- En gris, l'emplacement de l'Arduino Nano Every
- En violet, les bornes permettant d'alimenter la carte
- En bleu clair, les bornes permettant d'accéder au 5V ou 3,3V
- En bleu foncé, les connecteurs à 3 broches pour des capteurs binaires
- En jaune, les boutons-poussoirs
- En brun, les jumpers permettant de choisir entre bouton-poussoir et capteur
- En rouge, les bornes permettant d'accéder aux pattes analogiques de l'Arduino
- En orange, le bornier pour les moteurs
- En rose, l'emplacement du driver moteur DRI0044
- En bordeaux, les connecteurs pour les encodeurs moteurs
- En bleu clair, les bornes permettant de fixer la tension de référence de la conversion analogique numérique (CAN)

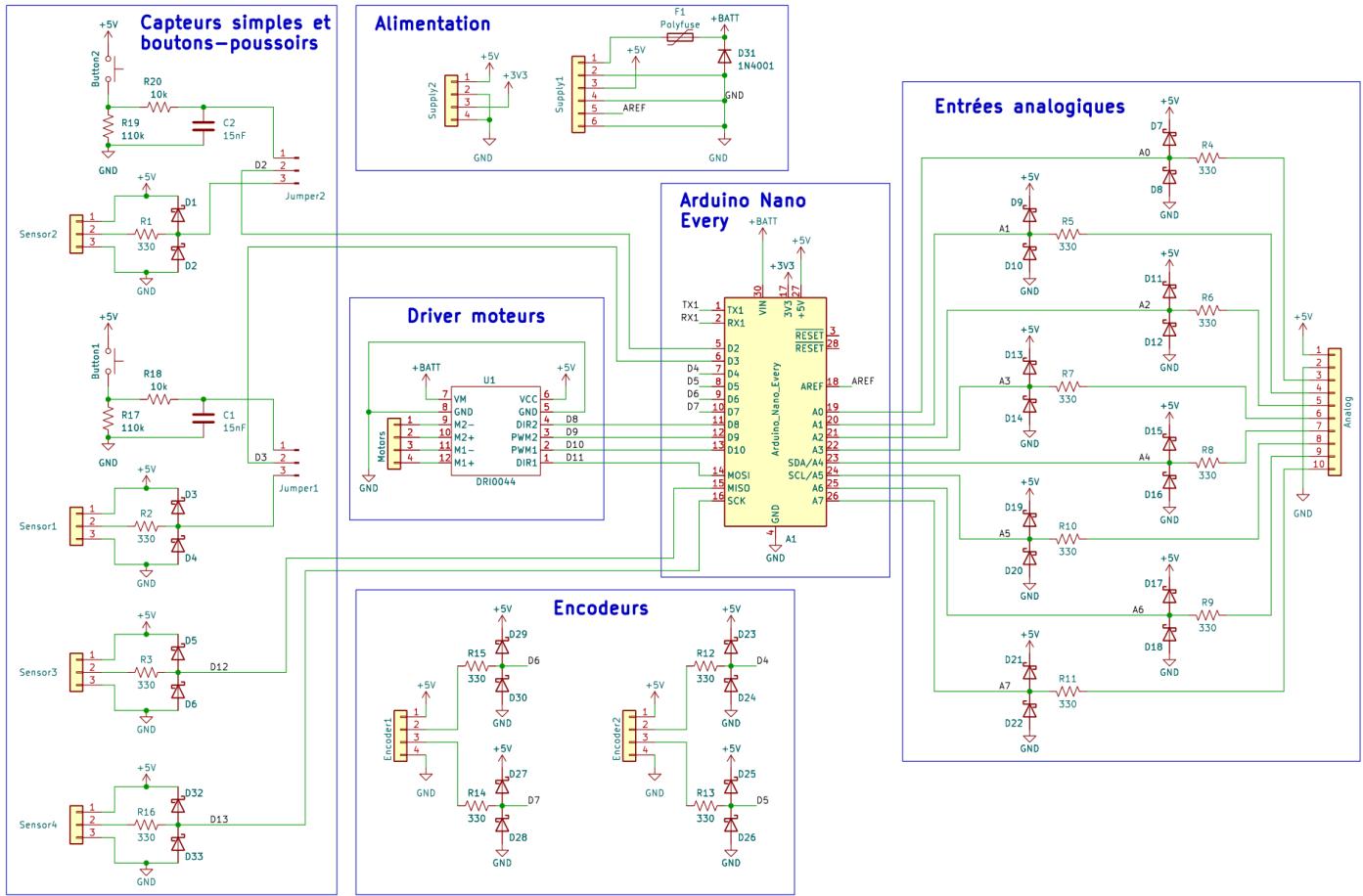


FIGURE A.2 – Circuit électrique de l'ensemble du shield provenant de gitlab.

A.1 De Python à C++

A.1.1 Traduction du code Python

Afin de mettre les 2 algorithmes en commun pour obtenir un code de robot fonctionnel, il a fallu les mettre dans le même langage. La technologie Arduino étant utilisée dans ce projet, il a paru évident de traduire l'algorithme de logique en C++. Pour cela, une intelligence artificielle du nom de CodeConvert.AI a été utilisée. Cette dernière a traduit l'algorithme dans les grandes lignes, mais a laissé passer quelques subtilités entre les mailles du filet. En effet, Python est un langage interprété, alors qu'à l'inverse, C++ est un langage compilé. Cette différence est cruciale lors de la traduction de code, car elle signifie qu'il faut être beaucoup plus précis lorsqu'on code en C++, car Python est un langage qui effectue certaines fonctions implicitement. Ceci inclut par exemple la définition de variables, dont il faut définir le type en même temps que la valeur, ou la définition de fonctions, dont il faut définir le type d'information renournée spécifiquement. Une autre difficulté causée par le fait que le C++ est un langage compilé est que le code ne se lance pas s'il y a une erreur

(même sans importance) dans le code. Ceci a néanmoins permis de corriger toutes les erreurs ou imperfections minimes du code (qui était nonobstant fonctionnel).

A.1.2 Fusion des deux codes

Après avoir correctement traduit le code Python en C++, il a fallu implémenter l'un dans l'autre. Pour cela, certaines modifications ont dû être apportées, notamment le fonctionnement de la localisation du robot. En effet, dans la simulation en Python, l'odométrie avait été utilisée, afin de se concentrer uniquement sur l'aspect logique du code du robot, alors que pratiquement, la localisation du robot est simplement basée sur le suiveur de lignes et la mémoire du robot. En sachant combien d'intersections il a croisé, celui-ci peut savoir facilement entre quels deux points de la carte il se trouve en tout instant. Ainsi, en suivant la ligne dans une direction ou dans l'autre, il se retrouve à l'intersection suivante, et effectue ainsi toutes les tâches de tri qu'il a à faire.

A.1.3 Code C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>

using namespace std;

class Emplacement{
private:
    short type{10};

protected:
    pair<int, int> pos;
public:
    Emplacement(pair<int, int> p){
        pos = p;
    }
    short get_type(){
        return type;
    }
    pair<int, int> get_pos(){
        return pos;
    }
    void set_type(short t){
        type = t;
    }
};

class Hand : public Emplacement{
public:
    void set_pos(pair<int, int> new_pos){
        pos = new_pos;
    }
};
```

```

vector<int> n_etape={}, temps_execution={};
short bon = 0;
const Emplacement emplacement1 = Emplacement({-10, -20}), emplacement2 =
Emplacement({-20, 20}), emplacement3 = Emplacement({-30, -20}), emplacement4 =
Emplacement({-40, 20}), emplacement5 = Emplacement({-50, -20}), emplacement6 =
Emplacement({-60, 20});
vector<Emplacement> liste_emplacement = {emplacement1, emplacement2, emplacement3,
emplacement4, emplacement5, emplacement6};
short T_PICKUP = 2, T_DROP = 2, T_TOURNER = 3, T_AVANCER = 2;

class Robot{
private:
    pair<int, int> r_pos{0, -20};
    int etapes{0}, orientation{270}, temps = 0;
    vector<pair<pair<int, int>, string>> trajet{{{0, 20}, "a"}};
    Hand in_hand{Emplacement{{0, 0}}};
    string direction{"sud"};
    bool arret{false};
public:
    void init(){
        temps = 0;
        in_hand.set_type(0);
        in_hand.set_pos({0, 20});
        r_pos = {0, 20};
        orientation = 270;
        arret = false;
        direction= "sud";
        etapes=0;
        trajet={{{0, 20}, "a"}};
    }
    pair<int, int> get_r_pos(){
        return r_pos;
    }
    vector<pair<pair<int, int>, string>> get_trajet(){
        return trajet;
    }
    string get_direction(){
        return direction;
    }
    void f_trajet(){
        trajet.back().second+='f';
    }
    int etape(){
        return etapes;
    }
    void pick_up(Emplacement obj){
        ++etapes;
        temps += T_PICKUP;
        trajet.back().second+="p";
        int ind = r_pos.first/-10-1;
        liste_emplacement[ind].set_type(0);
        in_hand.set_type(obj.get_type());
        cout<<"en_main_j 'ai_l 'objet_"<<in_hand.get_type()<<endl;
    }
    void drop(){
        etapes++;
        temps += T_DROP;
        trajet.back().second+="d";
        int ind = r_pos.first/-10-1;
        liste_emplacement[ind].set_type(in_hand.get_type());
        cout<<"j 'ai_drop_l 'objet_"<<in_hand.get_type()<<"_a_la_place_"<<r_pos.first<<':
            '<<r_pos.second<<endl;
        in_hand.set_type(0);
        if(bon!=3)tourne(-180);
    }
}

```

```

    }

void move_to(pair<int, int> new_pos) {
    temps+= T_AVANCER*pow((pow(r_pos.first -new_pos.first , 2)+pow(r_pos.second-
        new_pos.second , 2)), 1/2);
    etapes++;
    in_hand.set_pos(new_pos);
    trajet.emplace_back(new_pos, " ");
    r_pos = new_pos;
    cout<<"position_actuelle;"<<r_pos.first<<': '<<r_pos.second<<"_"<<
        liste_emplacement[0].get_type()<<liste_emplacement[1].get_type()<<
        liste_emplacement[2].get_type()<<liste_emplacement[3].get_type()<<
        liste_emplacement[4].get_type()<<liste_emplacement[5].get_type()<<"_en_
        main;"<<in_hand.get_type()<<endl;
}

void tourne(int angle) {
    etapes++;
    trajet.back().second+="t ";
    temps+=T_TOURNER*abs(angle)/90;
    orientation += angle;
    if(angle<0){
        cout<<"je_tourne_a_droite"<<endl;
    }
    else{
        cout<<"je_tourne_a_gauche"<<endl;
    }
    if(0 > orientation or 360 <= orientation){
        while (0 > orientation) orientation += 360;
        while (360 <= orientation) orientation -= 360;
    }
    if(orientation == 0)direction = "est";
    else if(orientation == 90)direction = "nord";
    else if(orientation == 270)direction = "sud";
    else direction = "ouest";
    cout<<"direction;"<<direction<<endl;
}

void stop() {
    n_etape.push_back(etapes);
    temps_execution.push_back(temps);
    cout<<"je_m'arrete\n" temps_approximatif;"<<temps/60<<"_min_"<<temps%60<<"_sec"
        <<endl;
    arret = true;
}

bool get_arret() {
    return arret;
}

int get_in_hand() {
    return in_hand.get_type();
}
};

Robot robot = Robot();
vector<short> map_debut;
const vector<short> map_correcte = {1, 1, 2, 3, 2, 3};
int nombre_cas = 540;

short devine_le_dernier_objet(const vector<short>& liste) {
    short n1=0, n2=0, n3=0, res;
    for (short i: liste){
        if(i==1)n1++;
        else if(i==2)n2++;
        else if(i==3)n3++;
    }
    if((n1==2 && n2==1) || (n2==2 && n1==1))res=3;
}

```

```

else if((n1==2 && n3==1) || (n3==2 && n1==1))res=2;
else if((n2==2 && n3==1) || (n3==2 && n2==1))res=1;
else res = 5;
cout<<"j'ai devine le dernier objet : "<<res<<endl;
robot.f_trajet();
return res;
}

vector<pair<int, int>> comptage(const vector<int>& liste) {
    vector<pair<int, int>> res = {};
    bool cond;
    for(int i : liste){
        cond= true;
        for(auto & re : res){
            if(i==re.first){
                cond= false;
                re.second++;
                break;
            }
        }
        if(cond)res.emplace_back(i, 1);
    }
    return res;
}

short detecte_objet(short ind){
    short res;
    if(map_debut[ind] == 0) {
        res=0;
        liste_emplacement[ind].set_type(0);
        cout<<"detection d'objet type = 0"<<endl;
    }
    else {
        res=1;
        liste_emplacement[ind].set_type(5);
        cout<<"detection d'objet type = 5"<<endl;
    }
    return res;
}

void recognise(){
    short ind = robot.get_r_pos().first/-10-1, n5=0;
    short ind_e;
    vector<short> map = {};
    liste_emplacement[ind].set_type(map_debut[ind]);
    cout<<"reconnaissance de l'objet : "<<map_debut[ind]<<endl;
    for(short i=0; i<6; ++i) {
        if (liste_emplacement[i].get_type() == 5) {
            ++n5;
            ind_e = i;
        }
        map.push_back(liste_emplacement[i].get_type());
    }
    if(n5==1)liste_emplacement[ind_e].set_type(devine_le_dernier_objet(map));
}

void premiel_aller(){
    robot.move_to({0, 0});
    robot.tourne(-90);
    short sign, n5=0;
    for(short i=0; i<6; i++){
        if(i%2==0) sign = 1;
        else sign = -1;
        if(n5!=4) {
}

```

```

        robot.move_to({-(i+1) * 10, 0});
        robot.tourne(90 * sign);
        n5+=detecte_objet(i);
        if (i != 5 && n5!=4) robot.tourne(-90 * sign);
    }
    else {
        detecte_objet(i);
        cout<<"devine_le_zero_a_"<<liste_emplacement[i].get_pos().first<<': '<<
            liste_emplacement[i].get_pos().second<<endl;
    }
}
if(liste_emplacement[5].get_type()!=5&&n5!=4) robot.tourne(90*sign);
}

void go_to(pair<int, int> pos_cible){
    int sign, sign2;
    if(robot.get_r_pos().first <= pos_cible.first) sign = 1;
    else sign = -1;
    if((robot.get_r_pos().first/-10)%2==0) sign2 = 1;
    else sign2 = -1;
    if(robot.get_r_pos().second != 0) {
        robot.move_to({robot.get_r_pos().first, 0});
        robot.tourne(90*sign*sign2);
    }
    while (robot.get_r_pos().first != pos_cible.first){
        robot.move_to({robot.get_r_pos().first+10*sign, 0});
    }
    if(robot.get_r_pos().second != pos_cible.second){
        if((robot.get_r_pos().first/-10)%2==0) sign2 = 1;
        else sign2 = -1;
        if(robot.get_direction() == "est" || robot.get_direction() == "ouest") robot.tourne
            (90*sign*sign2);
        robot.move_to(pos_cible);
    }
}

vector<pair<int, int>> ver_spot(int type_emplacement){
    vector<pair<int, int>> res = {{0, 0}, {0, 0}};
    bool prem = true;
    for(short i = 0; i<6; i++){
        if(map_correcte[i] == type_emplacement){
            if(prem and liste_emplacement[i].get_type() == 0) {
                res[0] = liste_emplacement[i].get_pos();
                prem = false;
            }
            else if(liste_emplacement[i].get_type() == 0) res[1] = liste_emplacement[i]
                .get_pos();
        }
    }
    return res;
}

pair<int, int> obj_proche(){
    int min = 70, test;
    pair<int, int> res = {};
    for(int i = 0; i<6; i++){
        test = abs(liste_emplacement[i].get_pos().first - robot.get_r_pos().first);
        if((liste_emplacement[i].get_type() == 5 && test < min) || (test < min &&
            liste_emplacement[i].get_type() != 0 and liste_emplacement[i].get_type() !=
            map_correcte[i] && ver_spot(liste_emplacement[i].get_type())[0].first != 0) ){
            min = test;
            res = liste_emplacement[i].get_pos();
        }
    }
}

```

```

}
cout<<"objet_proche:"<<res.first<<': '<<res.second<<endl;
return res;
}

void check() {
    bon = 0;
    for (short i=0; i<6; ++i) if (liste_emplacement[i].get_type() != 0 && liste_emplacement[i].get_type() == map_correcte[i]) ++bon;
    if (bon == 4) robot.stop();
    cout<<"bien_place:"<<bon<<endl;
}

void livrer(vector<pair<int, int>> spot_libre) {
    short ind = robot.get_r_pos().first/-10-1;
    if (robot.get_in_hand() == map_correcte[ind] or (spot_libre[0] == make_pair(0, 0)
        and spot_libre[1] == make_pair(0, 0))) robot.drop();
    else if (spot_libre[0] != make_pair(0, 0)) {
        robot.tourne(180);
        go_to(spot_libre[0]);
        robot.drop();
    }
    else {
        robot.tourne(180);
        go_to(spot_libre[1]);
        robot.drop();
    }
}

void operation() {
    premiel_aller();
    vector<pair<int, int>> spot_libre;
    while (!robot.get_arret()) {
        check();
        if (!robot.get_arret()) {
            pair<int, int> objet_proche = obj_proche();
            go_to(objet_proche);
            recognise();
            Emplacement Emplacement_sur_place = liste_emplacement[robot.get_r_pos().first / -10 - 1];
            robot.pick_up(Emplacement_sur_place);
            spot_libre = ver_spot(Emplacement_sur_place.get_type());
            cout << "spot_libre:" << spot_libre[0].first << ':' << spot_libre[0].second << "_et_" << spot_libre[1].first << ':' << spot_libre[1].second << endl;
            livrer(spot_libre);
        }
    }
}

bool condition_pour_respecter_le_cahier_des_charges(const vector<short>& liste, short element) {
    short n0 = 0, n1 = 0, n2 = 0, n3 = 0;
    bool res = true;
    for (short i : liste) {
        if (i==0)n0++;
        else if (i==1)n1++;
        else if (i==2)n2++;
        else n3++;
    }
    if((n1 == 2 && ((n2 == 1 && element==2) or (n3 == 1 && element==3) or element == 1)
        ) || (n2 == 2 && ((n1 == 1 && element==1) or (n3 == 1 && element==3) or element == 2)) || (n3 == 2 && ((n2 == 1 && element==2) or (n1 == 1 && element==1) or element==3)) || (n0<2 && liste.size()==5 && element != 0) || (n0 == 0 && liste.
}
```

```

        size ()==4 && element != 0 ) || (n0 == 2 && element == 0))res = false;
return res;
}

void boucle() {
    vector<vector<short>> deja_vu = {};
    constexpr short min = 0, max = 3;
    bool cond;
    random_device rd;
    default_random_engine eng(rd());
    uniform_int_distribution<short> distr(min, max);
    while(deja_vu.size ()!= nombre_cas) {
        cond=true;
        vector<short> l = {};
        for (int i = 0; i < 6; i++) {
            short h = distr(eng);
            while (!condition_pour_respecter_le_cahier_des_charges(l, h)) h = distr(eng
                );
            l.push_back(h);
        }
        for(const auto & i : deja_vu) {
            if(i==l) {
                cout<<"progression:"<<deja_vu.size ()*100/nombre_cas<<%<<endl;
                cond = false;
                break;
            }
        }
        if(cond) deja_vu.push_back(l);
    }
    for(const auto & i : deja_vu) {
        for (Emplacement o: liste_emplacement)o.set_type(10);
        map_debut = i;
        robot.init();
        operation();
        cout<<"liste_finale:"<< liste_emplacement[0].get_type()<<liste_emplacement
            [1].get_type()<<liste_emplacement[2].get_type()<<liste_emplacement[3].
            get_type()<<liste_emplacement[4].get_type()<<liste_emplacement[5].get_type
            ()<<" liste_debut:"<<map_debut[0]<<map_debut[1]<<map_debut[2]<<map_debut
            [3]<<map_debut[4]<<map_debut[5]<<endl;
        cout<<"nombre_d'etapes:"<<robot.etape()<<endl;
        vector<pair<pair<int, intint, intfor(pos = trajet.begin(); pos != trajet.end(); ++pos) cout<<'('<<pos->first.
            first<<': '<<pos->first.second<<") "<<pos->second<<"_>_";
        cout<<'('<<robot.get_r_pos().first<<': '<<robot.get_r_pos().second<<") s_fin"<<
            endl;
    }
}

int main() {
    boucle();
    float moy_e = 0;
    int moy_t = 0;
    for(int i : n_etape) {
        moy_e += i;
        cout << i << ",";
    }
    cout<<endl;
    for(int i : temps_execution) {
        moy_t += i;
        cout <<i<<",";
    }
    cout<<endl;
}

```

```
moy_e/= nombre_cas;
moy_t/= nombre_cas;
cout<<"moyenne_d'etapes : " <<moy_e << " et de temps " <<moy_t/60 << " min " <<moy_t%60 << "
sec " << endl;
vector<pair<int, int>> compte_e = comptage(n_etape);
vector<pair<int, int>> compte_t = comptage(temps_execution);
cout<<'[ ';
for(const auto e : compte_e) cout<<'[ '<<e.first << ", " <<e.second << ']' ;
cout<<' ] ' << endl;
cout<<'[ ';
for(const auto t : compte_t) cout<<'[ '<<t.first << ", " <<t.second << " ] , " ;
cout<<' ] ' ;
return 0;
}
```