POLYTECH
ANGERS

Rapport de Projet - PEIP2
Robot solveur de Rubik's Cube
2023

Rémi BROTTES
Bilal KHANOUS
Mélissa JOLIVET

Encadré par :
Sébastien Lagrange

université angers

Remerciements

Nous tenions tout particulièrement à remercier M. Sébastien Lagrange, pour l’aide qu’il nous a apportée en tant que tuteur, ainsi que M. Franck Mercier et M. Boris Rayer, pour leur assistance lors de l'utilisation des imprimantes 3D. Enfin, nous voulions aussi remercier M. Nicolas Delanoue pour ses explications, qui nous ont permis de trouver d'autres méthodes de résolutions du cube.

Sommaire

Table des figures

1- Introduction

1.1 Contexte du projet

Étant actuellement en seconde année d'école d'ingénieur préparatoire à Polytech Angers, on nous a demandé de réaliser un projet par groupe de 2 à 4 personnes sur une durée de 100h. Nous avons eu le choix entre plusieurs projets variés. L'objectif était d'utiliser l'ensemble des compétences acquises au cours de nos deux années. Nous avons donc choisi de réaliser un solveur de Rubik's Cube.

Il fallait pour cela être capable de créer un système de motorisation afin de permettre au solveur d'effectuer les rotations du cube nécessaire à sa résolution.

Nous avons commencé le projet le 25 janvier 2023 et avons fini celui-ci le 1 juin avec la remise du rapport et du blog. Ce projet est supervisé par notre tuteur, Mr.Lagrange, un enseignant de Polytech Angers.

Nous avons choisi ce projet car il mélange le développement informatique et la modélisation. Nous voulions travailler sur quelque chose de concret et avoir des retours visuels sur notre avancement.

1.2 Présentation du projet/cahier des charges

Nous avons réalisé un solveur de Rubik's Cube capable de mélanger le cube et de le résoudre. Pour cela, il a été nécessaire de créer un programme informatique capable de résoudre un Rubik's Cube ainsi qu'une structure motorisée capable de recevoir et suivre ses instructions.

Durant nos premiers cours d'organisation industrielle, nous avons pu nous poser des questions sur le cahier des charges de ce produit comme une entreprise l'aurait fait. Plusieurs diagrammes ont ensuite été réalisés dans le cadre de ce cours afin de mieux représenter notre projet.

<u>Analyse fonctionnelle</u>

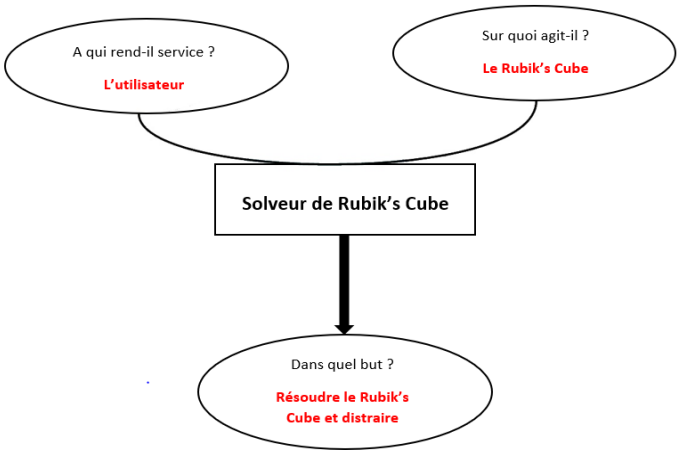


Figure 1 : Diagramme de la bête à corne

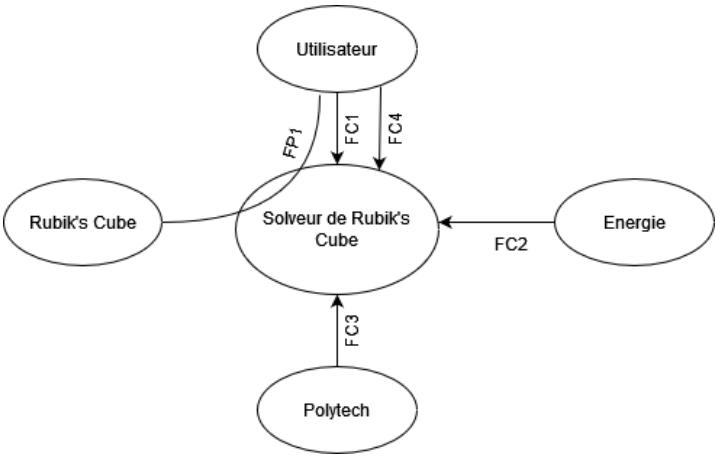


Figure 2 : Diagramme Pieuvre

FP1	Divertir l'utilisateur avec un Rubik's Cube
FC1	Esthétique
FC2	Energie
FC3	Coût
FC4	Vitesse

Nous n'avions pas de contraintes spécifiques pour notre projet. Le but n'étant pas d'atteindre des performances de compétition, notamment en termes de vitesse.
Ainsi, nous avons choisi les fonctionnalités suivantes :

- une interface graphique permettant de donner des instructions au robot.
- l'utilisation de la méthode de résolution classique pour le solveur
- la possibilité pour le robot de se mélanger tout seul.
- le fait de ne pas utiliser de capteurs pour connaître l'état du cube, l'état est entré par l'utilisateur au lancement du programme.

1.3 Organisation du temps de travail/répartition des tâches

Après avoir fait l'étude du projet en groupe, nous avons décidé de faire la programmation et la conception CAO en parallèle pour mieux se répartir les tâches.
Ces petits groupes nous ont donc permis d'avancer plus efficacement et plus rapidement.

Travail réalisé	Participants	Temps passé (h)
Etude	Bilal, Mélissa, Rémi	10
Programmation	Bilal, Rémi	70
Modélisation	Mélissa, Rémi	50
Fabrication	Bilal, Mélissa, Rémi	15
Rédaction	Bilal, Mélissa, Rémi	25

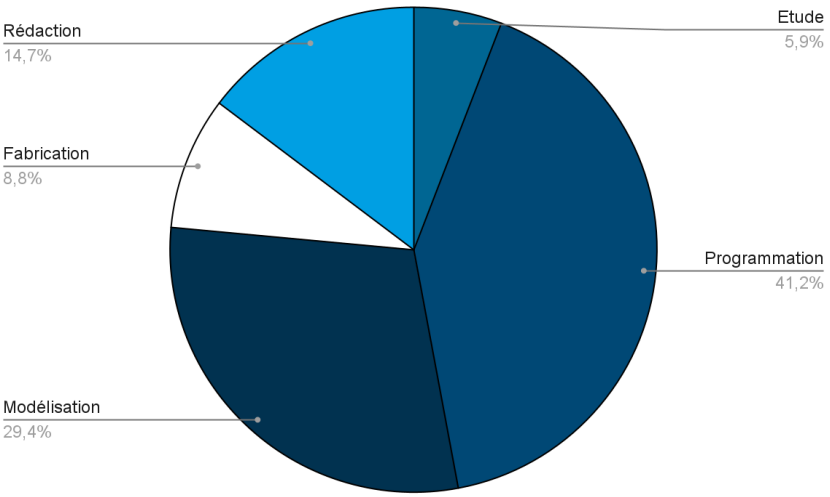


Figure 3 : Tableau et Camembert représentant la répartition des tâches

3- Description du travail réalisé

3.1 Conception

3.1.1 Etude

Pour commencer le projet nous avons réfléchi à la structure du solveur, à la façon de faire tenir le Rubik's Cube et aux moteurs à utiliser (nombre, nature).
Dans un premier temps, pour la structure nous nous sommes dirigés vers les *Makerbeam* à la suite des conseils de M.Lagrange, que nous avons mesurés et étudiés. Ces mesures nous ont permis de les modéliser et de créer la structure sur *SolidWorks*.

Dans un second temps, nous avons cherché plusieurs façons de faire tenir le Rubik's Cube :

- changer le centre des faces du cube en remplaçant le centre de chaque face par une pièce modélisée qui sera raccrochée au moteur.
- via des aimants en utilisant les aimants déjà présents dans le cube et d'autres aimants à l'extérieur pour faire bouger le cube.
- par pression en utilisant des plaques appuyées contre chaque face afin d'utiliser la friction générée pour tourner les faces.
- en utilisant 4 tiges ou 4 lames autour du centre de chaque face avec une pièce modélisée faisant bouger la face et en s'insérant dans les espaces autour du centre.

Nous avons choisi d'utiliser la méthode des quatres tiges, car elle nous a semblé être la plus simple à réaliser, comparée aux autres qui ne semblaient parfois même pas vraiment réalisables.

Ensuite, pour le cube à utiliser nous avons choisi le cube de *GAN* connecté car il est prévu pour le solveur de GAN. Ainsi, il possède plus d'accroche et a une meilleure fluidité de rotation que d'autres Rubik's Cube.

Enfin, nous avons décidé d'utiliser un moteur sur chaque face, c'est-à-dire six, afin de pouvoir utiliser la méthode de résolution classique du Rubik's Cube. Nous nous sommes donc intéressés au choix de ces moteurs en en étudiant différents types:

Il devait être capable de :

- tourner à au moins 270°, ou en rotation continue
- utiliser une position cible au lieu d'une vitesse

C'est pour ces raisons que nous avons décidé d'utiliser des moteurs *Ax-12A* de la marque *Dynamixel*, que M.Lagrange nous a conseillés, puisqu'ils correspondent parfaitement à nos critères.
Une fois l'ensemble de l'étude réalisée nous avons donc pu commencer la modélisation.

3.1.2 Modélisation

Pour la réalisation du solveur, il nous fallait une structure capable de soutenir les moteurs. Nous avons donc décidé de faire l'assemblage avec Solidworks afin de s'assurer qu'il sera réalisable.
Tout d'abord, séparément, nous avons créé les pièces que nous allions utiliser, c'est-à-dire le Rubik's Cube, les Makerbeams (profilé, équerre, connexions des sommets) ainsi que la pièce que nous allions imprimer pour lier le moteur et le cube.

Puis, nous avons réfléchi à la façon d'obtenir l'assemblage du moteur sans avoir à le modéliser, car il était trop complexe à réaliser. Après une recherche, nous avons trouvé une modélisation du dynamixel AX-12A (voir Figure 4) sur internet, ce qui nous a permis de compléter notre maquette virtuelle.

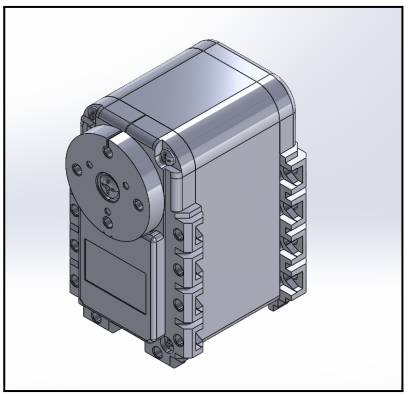
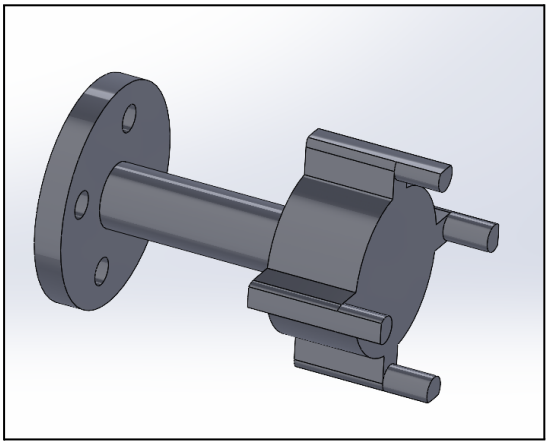


Figure 4 : Dynamixel AX-12A

Afin de pouvoir tenir le Rubik's Cube avec les moteurs, nous avons créé deux pièces: un axe que l'on a nommé "axle" qui sera relié au moteur, et une poignée que l'on a nommé "handle" qui sera en contact avec le cube (voir Figure 5). Pour la poignée, nous avons eu l'idée de nous inspirer des maintiens des solveurs produits par Gan.



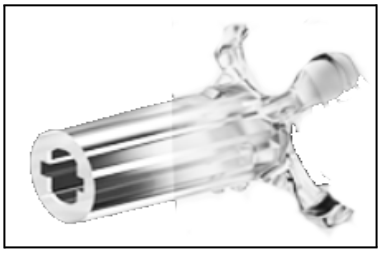


Figure 5 : Axle + handle en Solidwork (gauche),handle original de Gan (droite)

Une fois les pièces terminées, nous avons réalisé plusieurs sous-assemblage afin de pouvoir faire un assemblage final (voir Figure 6) plus facilement.

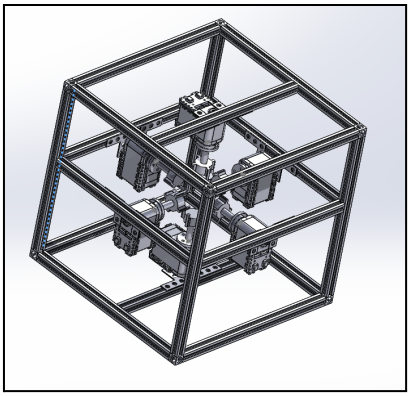


Figure 6 : Assemblage du solveur en SolidWorks

3.1.3 Construction

Afin de commencer la construction, il nous fallait commander les moteurs rapidement car les MakerBeams étaient déjà présents à Polytech. La recherche des moteurs était donc l'une des priorités dès le début du projet. Une fois les moteurs dynamixel choisis, nous avons décidé d'utiliser un starter pack afin de relier le moteur à l'ordinateur. Cela nous permettait d'alimenter les moteurs et d'avoir un bon adaptateur entre moteur et ordinateur.

Une fois la modélisation terminée, l'ensemble des pièces imprimées, les moteurs et leurs accessoires arrivés, nous avons pu commencer la construction finale du solveur. Nous avons tout d'abord assemblé les MakerBeams, puis nous avons connecté les moteurs entre eux en série et avec la carte. Nous avons pu étudier le fonctionnement du programme de résolution du Rubik's Cube ainsi que les deux pièces reliant le moteur et le cube.

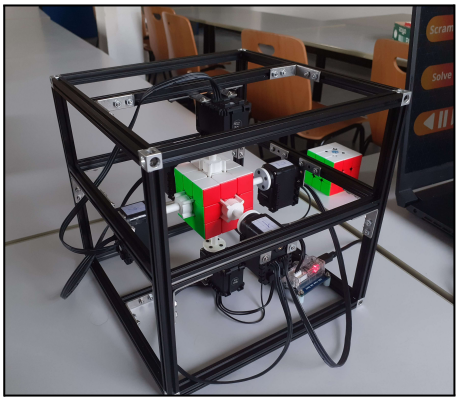


Figure 7 : Solveur de Rubik's Cube final

3.1.4 Problèmes rencontrés

Lors des différentes parties de la conception nous avons rencontré plusieurs problèmes :

Dans un premier temps, lors de la modélisation, il a été difficile de créer le Rubik's Cube exactement comme dans la réalité car les trous centralisés sont d'une forme unique. Nous avons donc essayé de le représenter de la manière la plus réaliste possible. Cependant, nous avons pu remarquer que dans l'assemblage Solidworks le handle ne rentrait pas parfaitement dans les trous en raison de cette approximation. Nous n'avons donc pas pu l'assembler dans l'assemblage final.
De plus, en réalisant la modélisation finale de la structure nous avons pu voir que les trous du moteur n'avaient pas les mêmes dimensions que les trous des équerres du MakerBeam, et qu'une partie de la structure du moteur gênait et empêchait l'assemblage. Pour résoudre ces deux problèmes, nous nous sommes dirigés vers l'utilisation de rondelles pour pallier au défaut de dimensions, puis avons positionné le moteur d'une certaine façon afin qu'il ne se bloque pas avec le profilé et les équerres.

Dans un second temps, pendant la première impression de la pièce reliant le moteur au Rubik's Cube, nous nous sommes rendu compte qu'elle n'était pas imprimable en 3D due à sa forme. En effet, les différences de couches requises n'étaient pas réalisables pour l'imprimante 3D que nous avions à notre disposition. Il a donc fallu la séparer en deux plus petites pièces pouvant s'imbriquer entre elles, un axe (axle) et une poignée (handle). De plus, lors de l'impression d'un des prototypes de notre handle, la pièce se détachait parce qu'elle n'avait pas assez de surface au sol, et que l'imprimante n'était pas bien calibrée. Afin de pallier ce problème, nous avons décidé de changer d'imprimante.

Dans un troisième temps, durant la construction du solveur, nous nous sommes aperçu de certaines erreurs de dimensionnement sur les différentes pièces. Par exemple, l'axe n'avait pas le bon écart entre les trous réservés aux vis. Nous avons donc dû modifier les fichiers CAO en redimensionnant la pièce. De plus, plusieurs prototypes de handle n'avaient pas la bonne forme permettant la fluidité de la rotation des faces et se cassaient en raison des forces exercées. Par conséquent, nous avons réalisé différentes versions (voir Figure 8) afin de trouver celle qui permettait la meilleure fluidité de rotation, et la meilleure solidité.

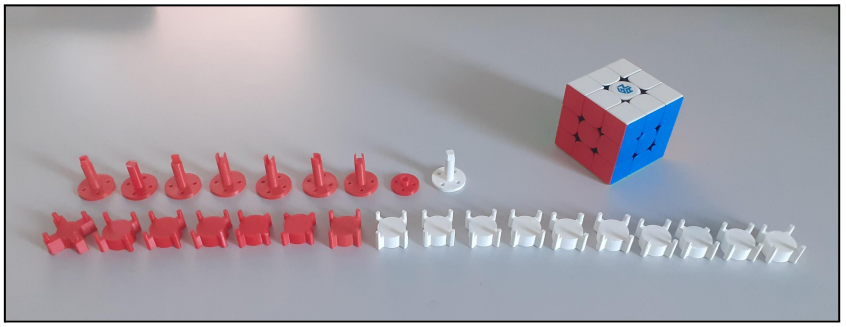


Figure 8 : Image de l'évolution des différents prototypes de handle et axle

3.2 Programmation

En ce qui concerne la partie programme du solveur, nous nous sommes dirigés vers le choix de la ***programmation orientée objet***. Cela nous permet de gérer les différentes fonctionnalités que nous voulions intégrer. Nous avons donc choisi ***Python*** comme langage de programmation.
Pour chacune d'elle, nous avons tout d'abord considéré quel type de structure utiliser. Puis nous nous sommes lancés dans l'implémentation de cette fonctionnalité. Enfin, nous avons effectué nos vérifications à l'aide du débogage afin de garantir au maximum l'absence d'erreur.

Un schéma représentant la structure finale du programme est disponible en annexe. (voir annexe 1)

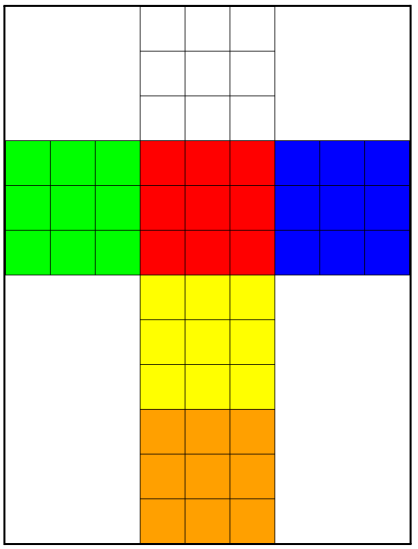


Figure 9 : Patron du cube

3.2.1 Étapes de la phase de programmation

Nous avons commencé par chercher un moyen de reproduire le fonctionnement d'un Rubik's Cube à partir d'un programme Python. Pour cela, nous avons fait le choix d'utiliser une matrice de 9 cellules pour représenter chacune des 6 faces du cube. Nous nous sommes ensuite intéressés au module ***pygame*** pour l'interface visuelle du cube. Pour cela, nous avons utilisé un programme d'affichage qu'un membre de l'équipe (Rémi) avait déjà conçu pour ses projets personnels.
Comme représentation, nous avons choisi le patron du cube (voir Figure 8), avec la face Rouge (Front) au centre et la face Blanche (Up) en haut.

Ensuite, pour permettre les rotations des faces, nous avons choisi d'enregistrer dans un fichier la façon dont les rotations d'une face affectent les faces adjacentes.

Pour représenter les mouvements, nous avons commencé par utiliser la nomenclature officielle : une rotation horaire est représentée par une lettre parmi R U F B L D (Right Up Front Back Left Down) correspondant à une face, et une rotation anti-horaire par une lettre suivie d'une apostrophe comme R' U' F' B' L' D'.
Cette méthode est efficace, mais nous avons préféré utiliser un seul caractère par mouvement afin de les représenter plus facilement sous forme de chaîne. Nous avons donc opté pour prendre une lettre minuscule pour le sens horaire et majuscule pour le sens anti-horaire.
Les algorithmes de résolution auront donc pour but de générer une chaîne qui résulte en un cube résolu. Ces algorithmes seront exécutés grâce au ***threading*** pour que les calculs soient faits sans ralentir le programme.

Le premier algorithme que nous avons fait s'appelle “scramble”. Il sert à mélanger le cube. Il a pu servir de test pour vérifier le bon fonctionnement des autres algorithmes.
Nous avons aussi prévu un moyen de colorier le patron du cube pour redéfinir sa position initiale, ainsi qu'un moyen de reset le cube.

Nous nous sommes ensuite penchés sur la résolution du cube, qui sera la partie la plus importante du programme. Nous avons implémenté l’algorithme classique de résolution du cube étage par étage. Premièrement, nous avons développé plusieurs petites fonctions qui nous seront utiles, comme la recherche de la position d’une arête ou d’un coin du Rubik's Cube.

La programmation de la résolution a aussi été la plus longue partie du programme. En effet, il existe beaucoup de cas particuliers dans les configurations du cube. Nous avons ainsi rencontré plusieurs problèmes en cours de route, notamment pour les deux premières étapes: la croix blanche et la première couronne.
Pour les étapes suivantes, comme la deuxième couronne ou la croix jaune, l'implémentation à été plus simple grâce à l'utilisation de techniques préexistantes comme la méthode du belge ou de la chaise.

Finalement, nous simplifions la chaîne obtenue pour retirer les mouvements inutiles, comme trois rotations dans un sens au lieu d’une dans l’autre, ou deux rotations qui se compensent.

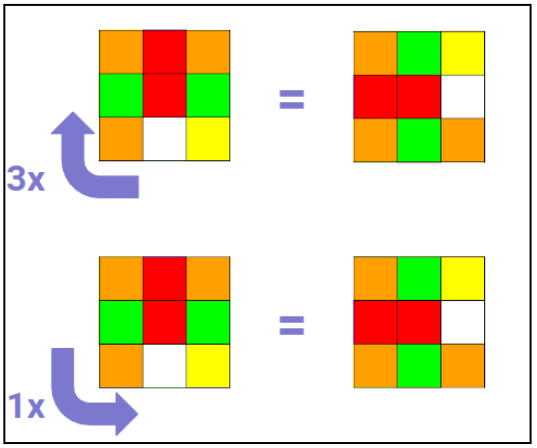


Figure 10 : Optimisation des mouvements

En parallèle de la résolution du cube, nous avons aussi développé le programme pour contrôler les moteurs. Tout d’abord nous avons récupéré en ligne la bibliothèque “dynamixel_sdk” distribuée par Dynamixel et avons essayé les programmes exemples fournis avec. Cela nous a permis de comprendre comment connecter les moteurs, leur envoyer les instructions et contrôler plusieurs moteurs en même temps.

Chaque moteur est désigné par un identifiant et connaît sa position actuelle et sa position cible. Un moteur tourne lorsque sa position actuelle et sa position cible sont différentes. Pour faire tourner un moteur il faut donc modifier la valeur de la cible stockée à l’adresse “goal_position” de sa mémoire.
Ces adresses sont données dans la documentation des moteurs et diffèrent avec chaque modèle.

Nous avons créé une classe Dynamixel qui *encapsule* cette bibliothèque pour faciliter son utilisation et ne garder que ce qui nous est utile.
Cette classe contient donc des méthodes pour donner une position cible à un moteur, récupérer la position actuelle, et gère entièrement la communication avec les moteurs.

Nous avons aussi créé une classe fille représentant notre moteur: AX12a, qui contient ces valeurs d'adresses et les paramètres de communication du modèle que nous possédons. Ce programme peut ainsi facilement être modifié pour fonctionner avec d'autres moteurs de la gamme Dynamixel.
Ce programme est disponible en open source dans les annexes. (voir annexe 3)

Une fois ce programme terminé, nous avons fait une classe qui sert d'interface entre le programme principal et la commande du moteur que nous avons nommé "Controller". Ce programme reçoit la chaîne de mouvement à exécuter et la convertit en instruction pour les moteurs.
Le programme principal envoie une chaîne de mouvements non optimisés. Le controller l'optimise pour retirer les mouvements inutiles et les sépare en instructions simples pour les moteurs.

Malheureusement, sur le produit final, les moteurs échouent parfois à tourner une face, pour palier à ce problème, nous avons ajouté une fonctionnalité pour revenir en arrière et réessayer un mouvement qui a échoué.

Finalement, nous avons ajouté une jolie interface pour permettre aux utilisateurs d'utiliser le programme et conclure le projet.

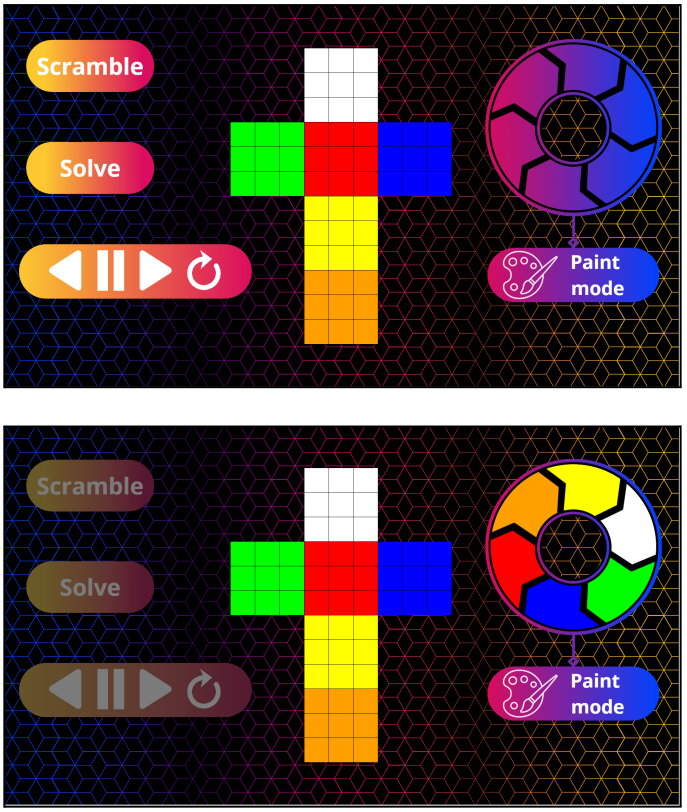


Figure 11 : Interface pour le programme

3.2.2 Problèmes rencontrés

Avant de commencer la programmation, nous avons dû décider de comment exécuter les rotations des faces du cube.

Lors de la rotation d'une face, toutes les cellules sur la face concernée sont tournées d'un quart de tour, mais la rotation affecte aussi les faces adjacentes.

Lorsqu'il a fallu reproduire ce fonctionnement dans notre programme, nous n'avons pas eu de problème pour tourner les cellules sur la face. Cependant, tourner les cellules des face adjacentes s'est avéré plus compliqué.
Pour cela, nous avons enregistré dans un fichier les indications nécessaires au programme pour tourner chaque face en donnant les déplacements des cellules (voir Figure 11: la ligne 3 de la face blanche devient la colonne 1 de la face bleue).

Cependant, bien que les valeurs de la ligne blanche se déplacent ensuite à la même position sur la colonne bleue, on peut voir par exemple que la position 1 de la colonne verte se déplace quant à elle la position 3 de la ligne blanche. Il y a donc parfois des cas inversés lorsque l'on applique le mouvement à la face.

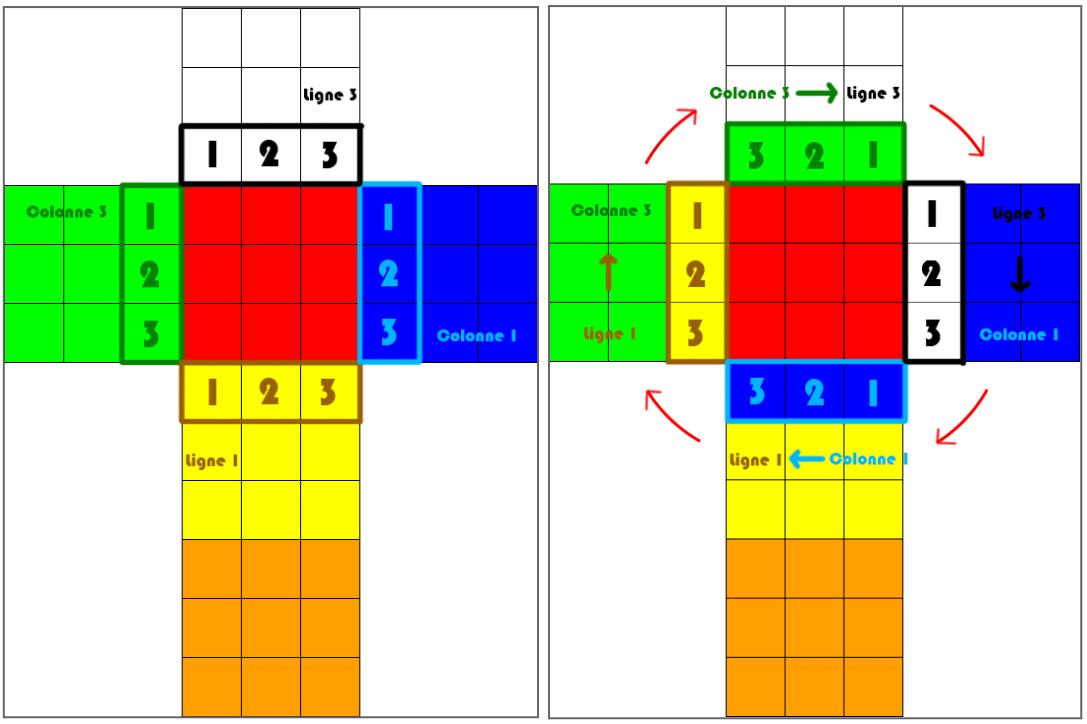


Figure 12 : Schéma du mouvement d'une face

De plus, nous avons aussi rencontré un problème d'orientation du cube. En effet, sur le patron, la face orange (Back) est inversée par rapport aux autres, ce qui a obligé l'implémentation de corrections. Par exemple, la méthode dite "de la chaise" qui est utilisée à la fin de la résolution a besoin d'une face de référence. Lorsqu'il faut appliquer cette méthode sur la face orange, il faut inverser tous les mouvements qui utilisent cette face orange.

Pour finir, nous avons eu des difficultés pour implémenter la résolution de la face blanche. Bien que cette étape soit souvent la plus facile pour la plupart du monde, il n'y a pas de méthode détaillée pour permettre au robot de la faire. Nous avons donc dû en créer une nous même, ce qui fut long et fastidieux. En effet, il y avait un grand nombre de cas particuliers, et après chaque étape implémentée, nous devions faire énormément de tests pour tous les trouver.

4- Conclusion

4.1 Critique des résultats obtenus

Nous avons réussi à réaliser un solveur capable de résoudre et mélanger un Rubik's Cube. Pour cela nous avons utilisé Solidwork pour la modélisation 3D et python pour la programmation. Cependant pour améliorer notre solveur il faudrait trouver une solution afin d'enlever le Rubik's Cube du solveur. De plus, il serait nécessaire d' ajouter une caméra qui sera capable de détecter les différentes parties du Rubik's Cube sans devoir le faire sur l'interface. Enfin, il faudrait améliorer ou changer la pièce reliant le moteur et le Rubik's Cube afin qu'il ne se bloque pas.

Une autre alternative serait d'utiliser la fonction bluetooth du Rubik's Cube que nous avons. En effet ce Rubik's Cube possède un gyroscope et pourrait être utilisé par le programme pour connaître la configuration actuelle.

4.2 Conclusion personnelles

Bilal Khanous

Au final, je dirais que ce projet était non seulement enrichissant et divertissant à réaliser, mais aussi particulièrement intéressant pour moi. Par exemple, je ne savais pas comment résoudre un Rubik's Cube auparavant, donc pour pouvoir programmer un algorithme de résolution, il a fallu que j'apprenne comment faire.

De plus, puisque j'aimerais continuer mes études dans le domaine de l'Informatique, cette expérience concrète est quelque chose qui me sera utile pour mes prochaines années à Polytech. Par exemple, grâce aux nouvelles méthodes et techniques de codage acquises que je pourrais maintenant utiliser dans d'autres projets.

Rémi Brottes

Ce projet a été une très bonne expérience, j'ai apprécié implémenter la résolution des Rubik's Cube et voir l'évolution concrète de notre assemblage. Ce projet m'a également montré que ce que j'avais appris lors de mon stage de fin de première année sur les communications entre appareils pouvait m'être utile pour interagir avec les moteurs.

Enfin, ce projet aura aussi montré l'importance de la répartition des tâches qui nous a permis de travailler plus efficacement.

Mélissa Jolivet

J'ai trouvé ce projet très intéressant car il m'a permis de me questionner sur la façon de trouver une solution à un problème posé.

De plus, j’ai pu perfectionner mes compétences en Solidwork et j’ai appris à utiliser l’imprimante 3D. Je l’ai apprécié car il est concret ce qui m’a permis de voir l’avancement du solveur. En effet, voir le projet avancé m'a motivée pour continuer et aller plus loin.

Enfin, il m’a permis de renforcer le travail d’équipe et l’autonomie, des qualités nécessaires pour un ingénieur.

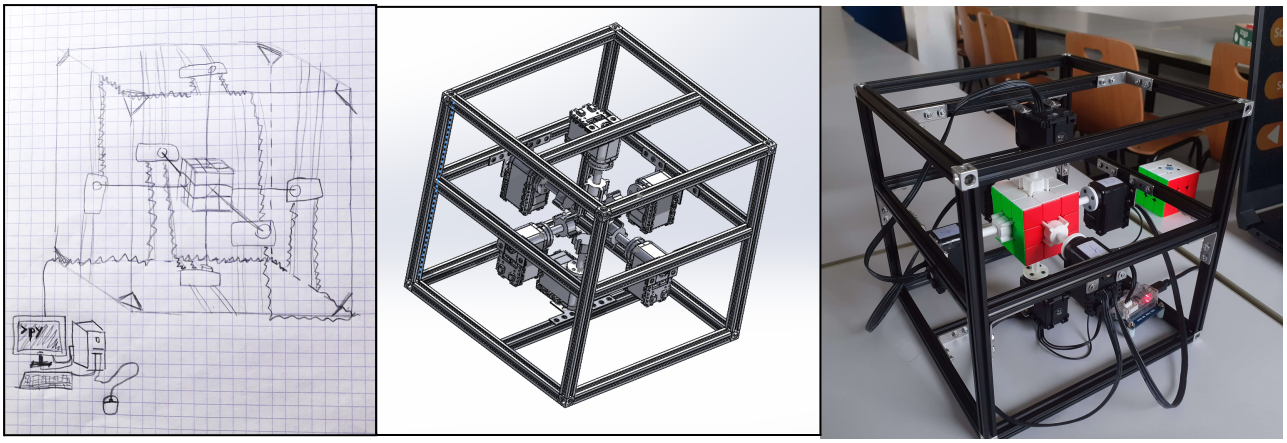


Figure 13 : Evolution du projet

5- Remarques

On a pu remarquer pendant nos recherches qu’il existait d’autres moyens techniques pour résoudre un Rubik's Cube. On aurait pu créer un bras mécanique pour le résoudre à une main, un solveur avec 4 moteurs, un Rubik's Cube avec le mécanisme de résolution dans le Rubik's Cube lui-même ou bien un solveur avec 2 moteurs.

6- Résumé

6.1 Français

Étant actuellement en seconde année d'école d'ingénieur préparatoire à Polytech Angers, nous avions comme projet de réaliser un Solveur de Rubik's Cube. Notre objectif était de le faire fonctionner en trouvant des solutions aux problèmes rencontrés. Pour cela nous l'avons séparé en plusieurs parties : la Modélisation, la Programmation et le Montage.

Ainsi, pour la partie modélisation nous avons réalisé le modèle complet du solveur sur Solidworks à partir des pièces que nous avions modélisé au préalable. Dans un même temps, nous avons programmé sur Python la méthode classique de résolution d'un Rubik's Cube à l'aide d'une simulation du cube. Pour finir, nous avons monté le solveur à l'aide de 6 moteurs, d'un Rubik's Cube, de MakerBeams, ainsi que les pièces que nous avions réalisées en 3D afin de relier les moteurs au cube.

Finalement, notre solveur est capable de mélanger et de résoudre un Rubik's Cube, en donnant la possibilité de rentrée une configuration initiale du cube.

6.2 English

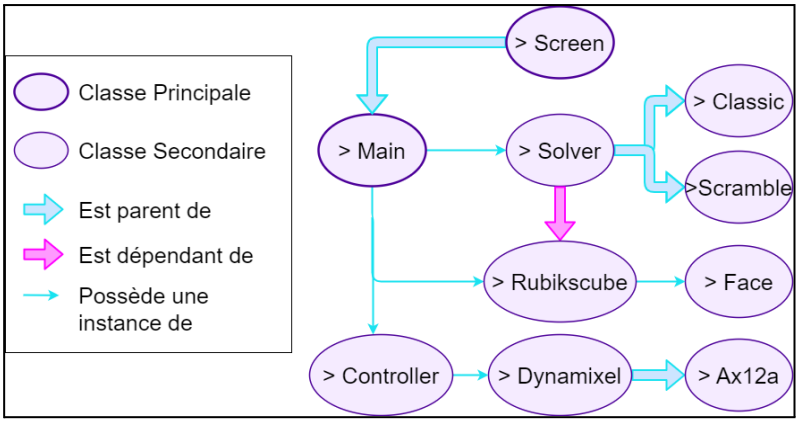
Currently in the second year of preparatory engineering school at Polytech Angers, our project consists of a Rubik's Cube Solver. Our objective was to make it work by finding solutions to all encountered problems.To do this, we split the tasks into multiple parts: Modeling, Programmation and Assembly.

So, for the modelisation, we recreated the complete model of the solver on Solidworks from the pieces we had made beforehand. At the same time, we have coded in Python the classical method to solve a Rubik's Cube using a simulation of the cube. Then, we assembled the solver by putting together the 6 motors, the Rubik's Cube, the MakerBeams, and the pieces we 3D printed in order to connect the motors to the cube.

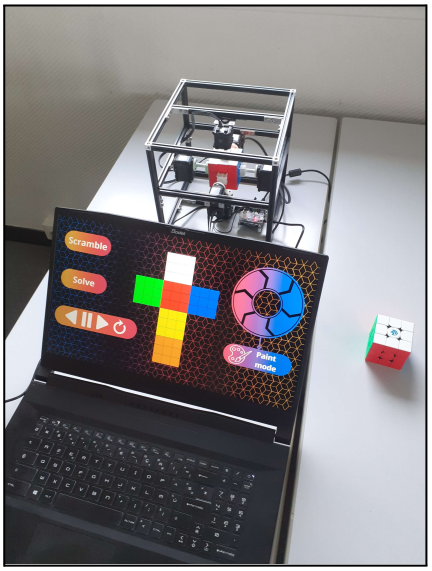
Finally, our solver is capable of scrambling and solving a Rubick' Cube, while giving us the possibility of choosing an initial state for the cube.

7- Annexes

7.1 Images



Annexe 1: Graphe de la structure du programme



Annexe 2 : Photo du solveur avec son interface

7.2 Liens

Annexe 3: module du contrôle des moteurs en open source
https://gitlab.com/Rexmine/dynamixel-wrapper.git

Annexe 4: Vidéo de notre solveur de Rubik's Cube en fonctionnement
https://youtu.be/ca8RG-YmusM

8- Glossaire

SolidWorks: Logiciel permettant de créer en modélisation 3D des pièces, des assemblages.

MakerBeam: Profilé en aluminium permettant la réalisation d'une structure.

GANCube: Entreprise spécialisée dans les Rubik's Cube de compétition, elle distribue aussi un robot Solveur de Rubik's Cube dont nous nous sommes inspirés.

Dynamixel AX-12a: Moteur distribué par Dynamixel, ce modèle est le plus bas de gamme mais est généralement suffisant pour les applications ne demandant pas de grand couple.

Dynamixel: Entreprise qui vend des moteurs.

Programmation Orientée Objet: (Abrégé POO) Paradigme de programmation qui consiste à structurer un programme grâce à des classes.

Python: Langage de programmation

Pygame: Bibliothèque graphique et Moteur de jeux vidéo 2D en python

Threading: Concept de programmation qui consiste à effectuer plusieurs tâches en parallèle.

Encapsulation: Concept de programmation qui consiste à enfermer un programme compliqué dans une "boîte noire" pour en faciliter l'utilisation depuis l'extérieur.