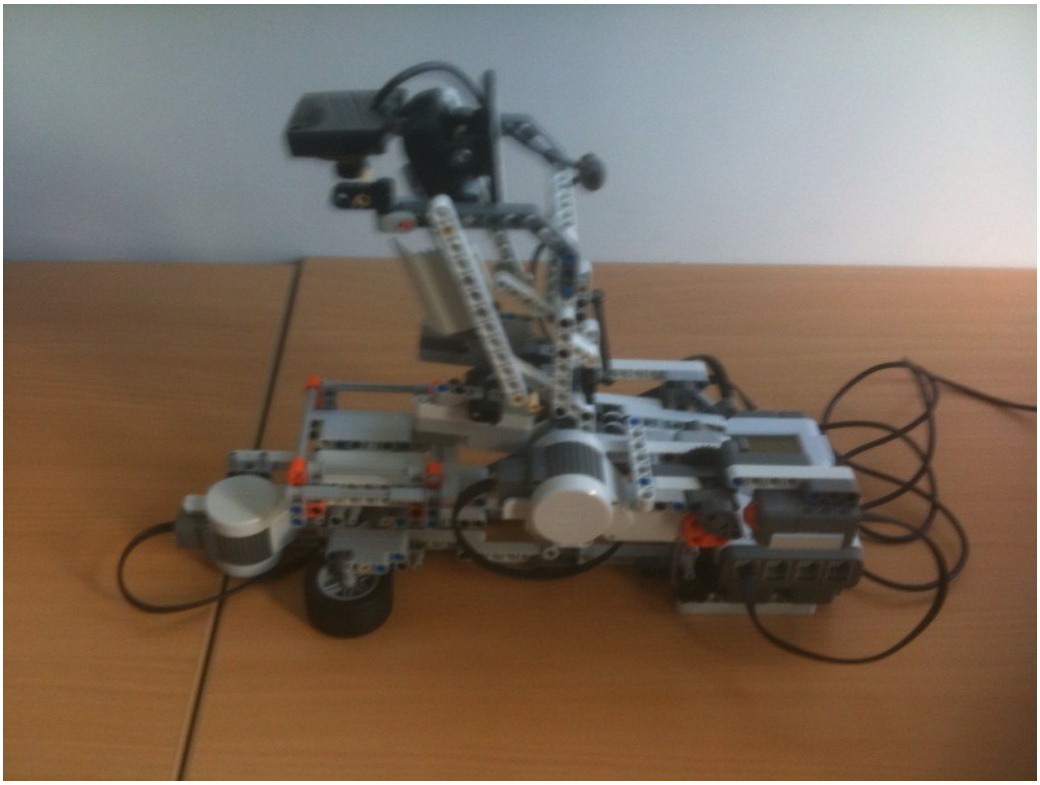
UNIVERSITÉ PARIS DESCARTES
UFR DE MATHÉMATIQUES ET INFORMATIQUE



Robot Rubik's Cube
Rapport Final

Bakiri Farhat Jaafar Jabeur Nabid Rami

Tuteuré par : M. Pellier Damien

Remerciements

Nous tenons à remercier tous ceux qui été à nos cotés dans notre projet et qui nous ont soutenu jusqu'à la fin du semestre.

Nous remercions aussi notre tuteur M. PELLIER Damien pour son aide et les idées qu'il nous a apportées tout au long de ce projet.

Un grand merci a tous les tuteurs et au bureau de secrétariat qui nous ont supporté et attendu plusieurs fois tard pour ouvrir et fermer la salle des Robots (notre lieu de travail).

Préambule

Cadre de développement

Notre projet « Robot Rubik's Cube » est un des projets proposés pour l'unité d'enseignement obligatoire « Projet Tutoré » en 3ème année de Licence Mathématique et Informatique, parcours Informatique de l'université Paris Descartes. Cette UE a pour objectif de nous initier au développement d'une application dans un environnement donné. Le développement d'application se fait en groupe en imaginant un client, ce qui nous permet d'avoir une idée sur la vie pratique. C'est un projet sur lequel nous devons s'organiser dans le temps, se répartir les tâches et surtout respecter les spécifications avant de coder et de programmer.

Les auteurs

BAKIRI Farhat, JAAFAR Jabeur et NABID Rami sont tous des étudiants en Licence 3ème année Mathématique et Informatique et Applications à l'université René Descartes en parcours Informatique.

Le sujet du Projet

Notre projet, comme l'indique son nom, consiste à construire un Robot capable de résoudre un Rubik's cube de n'importe quel état initial. Nous verrons dans la suite les détails de ce projet passionnant.

Table des matières

Chapitre 1

Préambule

1.1 Le Rubik's Cube

Vous avez certainement manipulé au moins une fois un Rubik's cube, nous allons maintenant vous présenter brièvement et sous des aspects moins communs ce casse tête.

1.1.1 Description de l'appareil

Le Rubik's cube, comme l'indique son nom, est un cube. Il est formé d'un assemblage de 26 petits cubes appelés ***cubics.*** Le Rubik's cube est un cube 3x3x3 (c'est le Rubik's cube standard, celui que nous avons utilisé pour notre Robot), chaque face de ce cube est composée de 9 cubics. Mais il existe plusieurs tailles et formes : le Pocket cube (2x2x2), Rubik's Revenge (4x4x4), Professor's Cube (5x5x5), mais aussi d'autres formes géométriques (tétraèdre, octaèdre...).

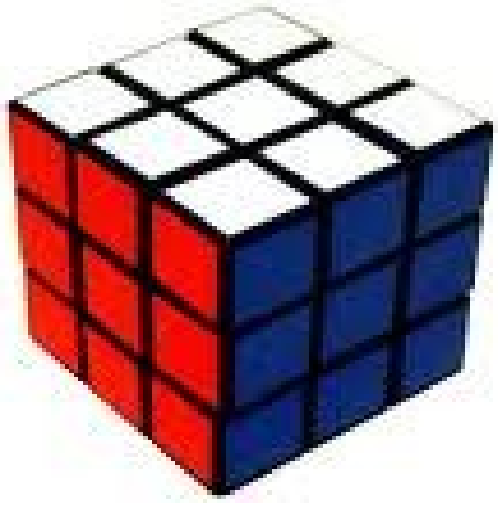


Fig 1.1 Rubik's cube résolu

Chaque face pivote indépendamment des autres grâce à un mécanisme interne assez simple mais complexe composé notamment d'un axe central reliant tous

les cubics centraux de toutes les faces.

Fig 1.2 Rubik's Cube désassemblé en cubics

Dans le cas d'un Rubik's cube résolu, chaque face est d'une seule couleur. Le Rubik's cube est composé ainsi de 6 couleurs différentes : blanc, jaune, rouge, orange, bleu et vert. Grâce au rotation des faces les couleurs se mélangent, le but du casse-têtes est de réunifier la couleur de chaque faces.

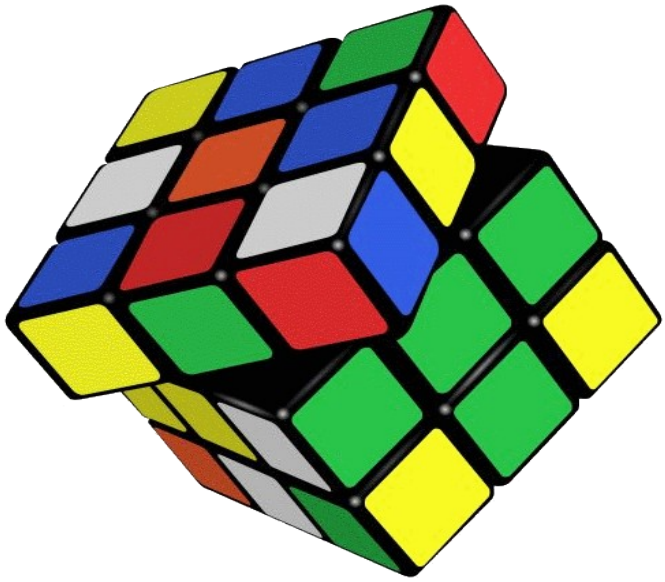


Fig 1.3 Rubik's cube en rotation

Il y a 3 types de cubics : les cubics centraux, les cubics d'arêtes et les cubics de coins. Les cubics centraux sont tous d'une couleur différente et ils sont

invariant les uns des autres : le blanc en face du jaune, le rouge en face du orange et le bleu en face du vert. De ce fait ils forment un repère invariant.
Les cubics d'arêtes ont deux couleurs chacune sur leurs deux faces apparentes, il y en a 12, ils sont situés à coté des cubics centraux.
Les cubics de coins sont coloriés de trois couleurs sur les 3 faces apparentes et il y en a 8 et ils se trouvent bien évidement sur les coins.

1.1.2 Un peu d'histoire

Le Rubik's cube a été inventé par le professeur d'architecture et sculpteur Hongrois *Erno Rubik* en 1974.

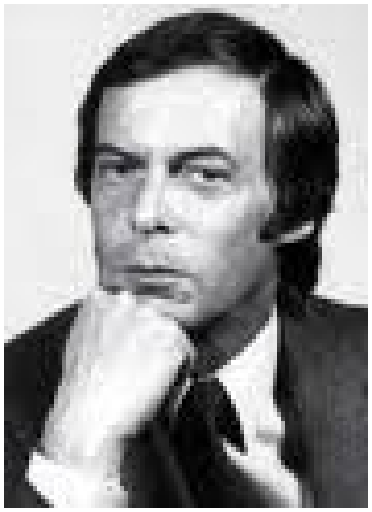


Fig 1.4 Erno Rubik

Le produit a été testé en 1977 et peu après les premiers Rubik's cube se sont vendus dans les magasins de Budapest.
En Hongrie, le cube gagne en popularité et lui permet d'être vendu dans le monde entier en 1980. Plus de 100 millions de cubes sont vendus entre 1980 et 1982.
Vu la popularité de ce jeu, plusieurs concours de vitesse ont vu le jour. La pratique du Rubik's cube en vitesse est appelé *Speedcubing.* Vu la complexité du casse-têtes, finir un Rubik's cube en moins d'une minute paraît d'une grande performance, pourtant le record mondial de ce jeu est de 7.08 seconds réaliser par ***Erik Akkersdijk*** .

1.1.3 Les Standards

Il existe quelque désignations du Rubik's cube pour mieux comprendre sa résolution. Ces standard servent à représenter les faces et les mouvements que nous pouvons appliquer.

Des noms ont été choisis pour désigner chaque face, chaque face est representé par la couleur de son ***cubic*** central. On a choisi cette convention pour notre application :

- la face bleue est appelée face UP, abréviation U ;
- la face rouge est appelée face FRONT, abréviation F ;
- la face jaune est appelée face RIGHT, abréviation R ;
- la face verte est appelée face DOWN, abréviation D ;
- la face orange est appelée face BACK, abréviation B ;
- la face blanche est appelée face LEFT, abréviation L.

Grâce à ces désignation on peut facilement représenter les mouvements. En effet, le standard veut que lorsque l'on appelle un mouvement R, cela veut dire tourner la face RIGHT d'un quart de tour dans le sens horaire. Le quart de tour est le mouvement de base du Rubik's Cube. On peut aussi décrire le mouvement inverse, noté alors R', et qui veut dire tourner la face RIGHT d'un quart de tour dans le sens anti-horaire , aussi on peut décrire le mouvement R2 qui veut dire tourner la face RIGHT d'un demi tour.

1.2 Algorithme de résolution

Il existe plusieurs algorithmes informatique de résolution du Rubik's cube qui réalise une recherche (A*, IDA*...) en parcourant un arbre de recherche, dont chaque nœud représente un état du Rubik's Cube et chaque arrête un mouvement.
Voici une petite explications des algorithmes les plus évolués à ce jour :

- L'algorithme THISTLETHWAITE :

L'idée de cet algorithme est de deviser le problème en sous problèmes. Il les devise en restreignant le type de mouvements que vous pouvez exécuter.
En particulier, cet algorithme divise le groupe cube dans la chaine des sous groupes suivants :

$G_0 = <L,R,F,B,U,D>$
$G_1 = <L,R,F,B,U2,D2>$
$G_2 = <L,R,F2,B2,U2,D2>$
$G_3 = <L,R,F2,B2,U2,D2>$
$G_4 = \{I\}$

Ensuite, il prépare des tableau pour chacun des coûts de $G_{[i+1]}/G_{[i]}$. Pour chaque élément, il trouve une séquence de coup qu'il prend pour le prochain groupe plus petit.
Le nombre de coup de cet algorithme est la somme des plus grandes processus à

chaque étapes et c'est 52 mouvements.

- L'algorithme de KOCIEMBA :

Cet algorithme est une amélioration de l'algorithme THISTLETHWAITE. En effet il réduit le nombre de sous groupe à 3 comme suit :

$G_0 = <L,R,F,B,U,D>$
$G_1 = <L,R,F2,B2,U2,D2>$
$G_2 = \{I\}$
Le nombre de mouvements maximum pour résoudre le Rubik's cube avec cet algorithme est de 21.

- L'algorithme de KORPF :

En 1997, Richard KORPF a annoncé un algorithme avec le quel il a résolu de manière optimale des instances aléatoires du cube. Sur les 10 cubes qu'il a résolu aucun n'a demander plus de 18 mouvements.
L'algorithme utilisé est nommé IDA*. IDA* est un algorithme de recherche en profondeur d'abord qui cherche une solution de plus en plus longues dans une série d'itérations, utilisant une heuristique limite inférieur pour élaguer les branches une fois leurs limites dépasse celle-ci.
Il fonctionne à peu près comme suit. D'abord, il a identifié un nombre de sous-problèmes qui sont suffisamment petits pour être résolu de façon optimale. Il a utilisé :

- Le cube est limité seulement à ces coins sans regarder les arêtes.
- Le cube est limité seulement à 6 arêtes, on ne regarde pas les coins ni les autres arêtes.
- Le cube est limité aux 6 autres arêtes.

Clairement, le nombre de mouvements requis pour résoudre n'importe les quels de ces sous problèmes est une limite inférieur pour le nombre de coups que vous aurez besoin pour résoudre le Rubik's cube.
Cet algorithme est bien décrit en détail dans son document : «*Finding Optimal Solutions to Rubik's Cube Using Pattern Databases*».

1.3 Mathématique et Rubik's Cube

Le Rubik's cube est un casse-têtes mathématique, sa structure et les mouvements qui lui sont associé sont en corrélation avec un objet mathématique fondamental : ***les groupes***

1.3.1 Rubik's cube et théorie des groupes

Le Rubik's Cube est associé à la théorie des groupes car l'ensemble des manœuvres de celui-ci forme un groupe.
Soit *M* l'ensemble des manœuvres du Rubik's Cube, et soit suivi de une opération.
Soit *m,n* et *o € M*.
* *m* suivi de *n € M* (une manœuvre suivie d'une autre manœuvre est encore une manœuvre).
Par conséquent, *M* est stable par l'opération suivi de .
* (*m* suivi de *n*) suivi de *o* <=> *m* suivi de (*n* suivi de *o*).
Par conséquent, suivi de est une opération associative.
* Soit ɜ la manœuvre qui consiste à ne rien faire.
m suivi de ɜ <=>*m* (une manœuvre suivie de ne rien faire est toujours la même manœuvre).
Ainsi, la manœuvre qui consiste à ne pas bouger est l'élément neutre.
Chaque manœuvre peut être effectuée à l'envers. Ceci est donc une manœuvre inverse.
Voila donc pourquoi l'ensemble des manœuvres du Rubik's Cube constitue un groupe.

1.3.2 Nombre de position différente

Le nombre de positions différentes est de 8! × 37 × 12! × 210 = 11 × 72 × 53 × 314 × 227 = 43 252 003 274 489 856 000 (c'est-à-dire plus de 43 milliards de milliards de combinaisons), dont 1 seule correspond au cube fini. Pour donner une idée du nombre de combinaisons, en passant en revue 1 milliard de combinaisons différentes par seconde, cela prendrait plus de 1 200 ans pour toutes les parcourir .

Cela se calcule comme suit:

1. Chaque arête peut prendre deux orientations possibles. Étant donné qu'on ne peut pas changer l'orientation d'une arête seule, l'orientation de toutes les arêtes fixe l'orientation de la dernière. Cela nous donne 211 possibilités d'orientation des arêtes.
2. Chaque coin a trois orientations possibles. De même, on ne peut pas retourner un coin seul, l'orientation du dernier coin est donc fixée par les autres. Cela nous donne 37 possibilités d'orientation de coins.
3. Les arêtes peuvent s'interchanger entre elles, ce qui nous donne 12! possibilités de positionnements pour les arêtes.
4. Les coins peuvent s'interchanger entre eux. Cela fait 8! possibilités.
5. Mais il existe un problème dit de parité : on ne peut échanger juste deux coins ou deux arêtes (mais on peut interchanger deux coins ET deux arêtes). La

position des arêtes et des premiers coins fixe donc la position des 2 derniers coins et il faut donc diviser le résultat par deux.

Ce qui donne bien :8! × 37 × 12! × 210 = 43 252 003 274 489 856 000

Les centres ne sont pas considérés dans ce calcul, car ce sont eux qui nous servent de points de repère.

1.4 Les Robots Lego Mindstorms

1.4.1 Description de l'appareil

Lego Mindstorms NXT est un jeu de construction et de robotique présenté par Lego en 2006. Il succède à la gamme Lego Mindstorms. En 2010, Mindstorms NXT en est à sa version 2.0.

Fig 1.5 Boite lego mindstorm

Les principales caractéristiques de la nouvelle gamme LEGO MINDSTORM NXT sont :

- Brique intelligente programmable NXT 4 ports d'entrée et 3 ports de sortie

Fig 1.6 La brique NXT

- Connexions USB et Bluetooth
- 3 servo moteurs interactifs
- 10 capteurs dont (4 fournis dans la boite principale): ultrason, son, lumière, contact et couleur, autodirecteur infrarouge, gyroscopique, infrarouge, accéléromètre et boussole.

Fig 1.7 Capteur de contact

Fig 1.8 Capteur ultrason

Fig 1.9 Capteur de lumière

- Logiciel de programmation intuitif avec une version Lego de Labview
- Compatible avec Windows et Mac
- Accessoires supplémentaires (non fournis dans la boite principale): clé Bluetooth USB et batterie rechargeable.

1.4.2 Utilisation possible

Au moyen de sa brique intelligente NXT, Lego Mindstorm offre plusieurs idée pour construire des Robots originaux et intelligents.

Plusieurs plans de construction sont fournis par Lego Mindstorm, on peut les retrouver sur le site : http://mindstorms.lego.com/.

Nous avons pu constater la présence de plusieurs Robot Lego Mindstorm sur le net, ces Robots on été conçus par des inconnus. Nous remarquerons que l'utilisation du robot peut être très variée, du joueur de puissance 4 au robot tireur de chasse de WC, en passant par des robots gyropode .
On peut visiter ce lien pour en voir quelque un :
http://www.youtube.com/results?search_query=robot+lego+mindstorm&aq=f

Le Robot Rubik's Cube, qui est notre projet, est un Robot capable de résoudre un Rubik's cube de n'importe quel état initial, nous nous sommes procuré les plans de ce Robot dans livre de ***Daniele Benedettelli*** « *LEGO MINDSTORMS NXT THINKING ROBOTS* ».

Chapitre 2

Analyse de l'existant

2.1 Les robots Rubik's cube

2.1.1 Les projets

Avec un peu de recherche sur le net, on peut voir un certain nombre de vidéos de Robot Rubik's cube. La plupart de ces robots sont des robots Lego Mindstorm nxt 2.0 comme celui de ***Daniele Benedettelli***, mais aussi des robots Lego Mindstorm nxt 1.0. Ces projets n'utilisent pas tous un ordinateur pour la capture et le calcul de solution (voir http://tiltedtwister.com/)
Il existe d'autres robots capable de résoudre un Rubik's cube comme le robot « Robot II » qui a une forme humaine et le robot « CubeStormer » qui est un robot Lego Mindstorm très développé par rapport aux autres, on remarque surtout que ce robot peut tenir le cube de 4 faces différentes en même temps.

2.1.2 Les records

Comme on a vu ci-dessus le record détenu du Speedcubing est de 7.08s, cependant pour les robots Rubik's cube c'est différent. La plupart des robots sont capables de résoudre le Rubik's cube en minimum 50s ce qui n'est pas négligeable, par contre le robot « CubeStormer » a réussi à résoudre ce casse-têtes en moins de 12s ce qui déclenche une rivalité entre l'homme et la machine.

2.2 Les programmes Rubik's cube

Il existe sur le net de nombreux programmes permettant de manipuler un Rubik's cube. La plupart de ces programmes sont des Applets java qui permettent de manipuler le cube et de le résoudre sans un vrai calcul de solution c'est à dire sans algorithme de résolution, ils consistent seulement à appliquer l'inverse des mouvements que nous avons appliqués.
Néanmoins il existe quelques programmes qui sortent du lot et qui nous offre une vrai résolution du Rubik's cube. Dans ce cadre on peut citer le programme « Cube Explorer 4.0» réalisé par M. Kociemba, « Cube Twister » qui offre la possibilité de

manipuler d'autre forme de cube et qui permet une véritable résolution à l'aide de l'algorithme de Kociemba.
Enfin, nous pouvons citer « CubeSolver » de ***Daniele Benedettelli*** qui offre les même fonctionnalités principales que notre projet (programmé en C) mais ne propose pas d'affichage 3D du Rubik's Cube.

2.3 Acquisition d'images

Après de nombreuses recherches, il s'avère qu'il n'y a pas beaucoup de solutions pour l'acquisition de l'image par webcam qui va nous permettre de numériser le Rubik's cube.
Parmi ces quelques solutions, on peut citer les API QTJava et Java Media Framwork (JMF) qui nous offrent une possibilité d'acquisition d'images. La première API est déprécié, et la second n'offre pas la portabilité que propose le langage JAVA.

2.4 Analyse d'image

Pour l'analyse de l'image, les solutions sont multiple et indépendante du langage de programmation choisis.

On pourra associer des solutions comme la détection de contour pour récupérer le cube sur notre image à l'aide de l'application des matrices comme Sobel ou Prewitt puis la transformé de Hough pour récupérer les lignes de notre Rubik's cube.

Aussi on peut récupérer la couleur de chacune des cases qui devrait être chacune une facette. La récupération de couleur se fait par l'intermédiaire des valeurs RGB ou encore LAB.

Chapitre 3

Introduction au projet

3.1 But

Le but de notre projet est donc d'implémenter un logiciel permettant au Robot de manipuler un Rubik's cube 3x3x3 et de pouvoir le résoudre au moyen d'un algorithme de résolution optimale. Donc, on doit permettre à l'utilisateur l'acquisition du Rubik's cube par le moyen d'une webcam et sa modélisation en 3D.
Tout ça via une interface graphique ergonomique et efficace ce qui nous distinguera des autres logiciels ou la capture par webcam est peu présente.

3.2 Justification set priorités

Notre priorité est de mener à bien ce projet et de réussir ces fonctionnalités obligatoires : le robot doit pouvoir capturer et résoudre un Rubik's cube dans une état aléatoire.

Après avoir vu et tester plusieurs programmes permettant la manipulation du Rubik's cube, il serai intéressant pour nous d'avoir une application qui rapporte plus que les autres, nous visons donc à améliorer certaines fonctionnalités déjà existantes tout en incluant une possibilité rarement présente : l'acquisition par webcam, la vue en 3D ainsi que la manipulation du Rubik's Cube, la résolution par un Robot (l'application des mouvements calculés pour résoudre le Rubik's cube).

3.3 Schéma de structure

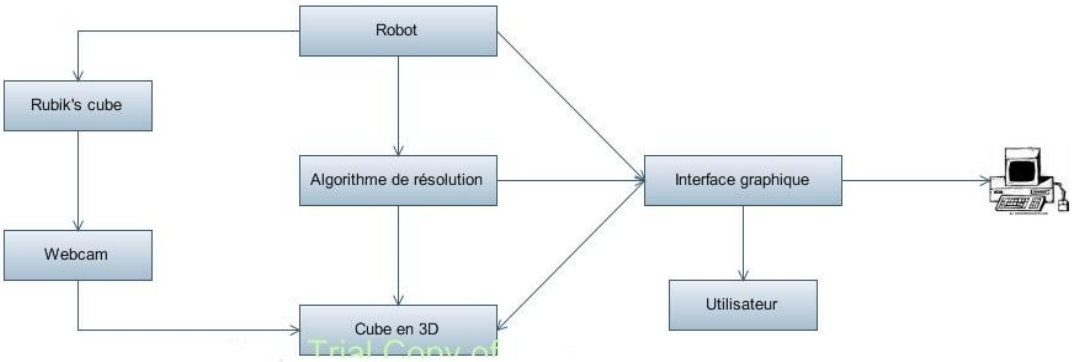


Fig 3.1 schéma de structure

3.4 Schéma de fonctionnement

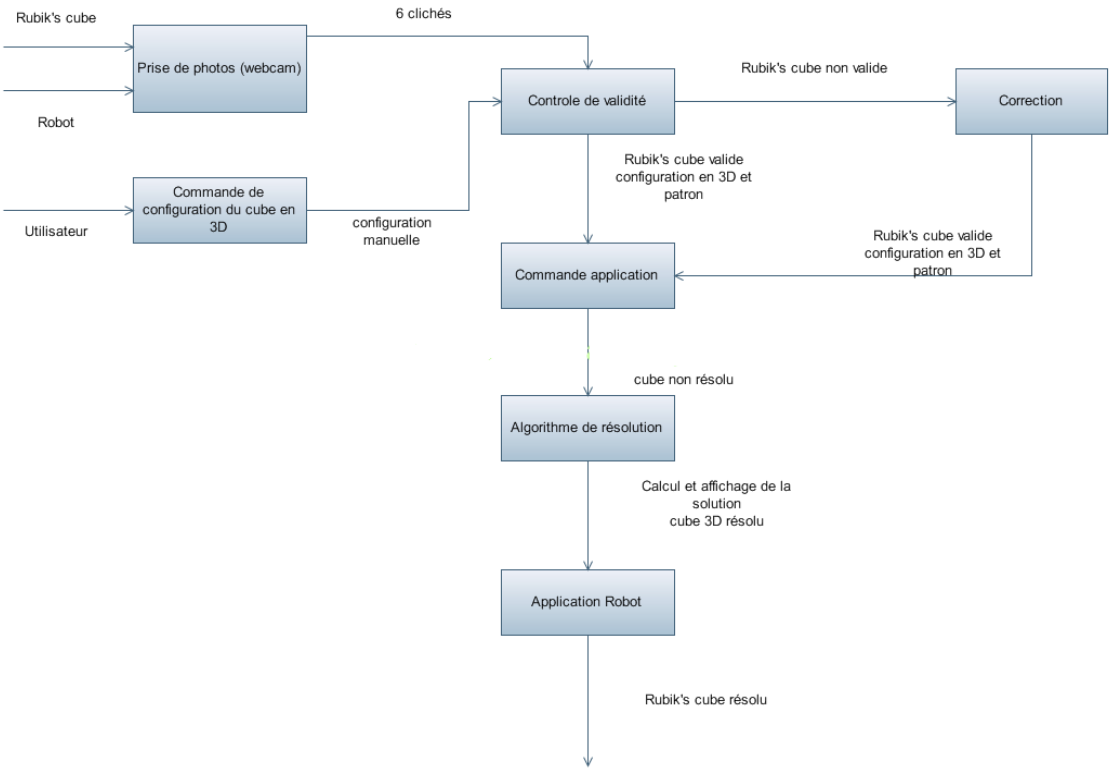


Fig 3.2 schéma de fonctionnement

Chapitre 4

Besoins fonctionnels

Nous allons décrire ici les besoins fonctionnels de l'application principale et celle se trouvant sur le robot. Ces besoins reflètent les objectifs indispensable à la réalisation du projet. Ce chapitre va présenter ces fonctionnalités sur lesquels nous nous sommes engagés sous un aspect moins exhaustif que sur le cahier des charges.

4.1 Relatifs à l'interface et à la modélisation

L'application principale doit permettre la modélisation d'un cube en mémoire. L'application doit aussi avoir une interface graphique permettant d'interagir avec le cube et de le visualiser. Une vue patron doit être implémentée.
Les couleurs des faces doivent être les couleurs officiels.

4.2 Relatifs à la manipulation

L'application principale doit permettre la manipulation du Rubik's Cube se trouvant en mémoire a travers l'application des différents mouvements possibles du Rubik's Cube.
L'application robot doit permettre de manipuler le vrai cube grâce aux moteurs et capteurs disponibles.

4.3 Relatifs à l'acquisition et à l'initialisation

L'initialisation du cube doit être possible à l'aide d'une webcam.
L'interface doit permettre à l'utilisateur de vérifier que les couleurs détectés par la webcam soit les bonnes.

4.4 Relatifs à la résolution

Le calcul de la solution pour revenir à l'état initial du Rubik's Cube doit être performante. La résolution par le robot doit être faite en un minimum de coup . Des algorithmes efficaces existent, il faudra que l'application en ai au moins un

d'implémenté .

4.5 Relatifs à l'application par le robot

Le logiciel robot doit communiquer avec l'application principale pour connaitre la solution à appliquer au Rubik's Cube. Le robot possède 2 moyens de communication l'USB et le Bluetooth . Une liaison utilisant un de ces deux moyens doit être implémenté.

Chapitre 5

Fonctionnalités implémentées et exemples de fonctionnement

Nous verrons ici, en opposition avec le chapitre précédent, ce qui a été réellement réalisé.

5.1 Interface et visualisation

L'interface graphique se décompose en 3 vues :

- une vue permettant la capture ;
- une vue éclatée du cube de type patron ;
- une vue du cube en trois dimensions.

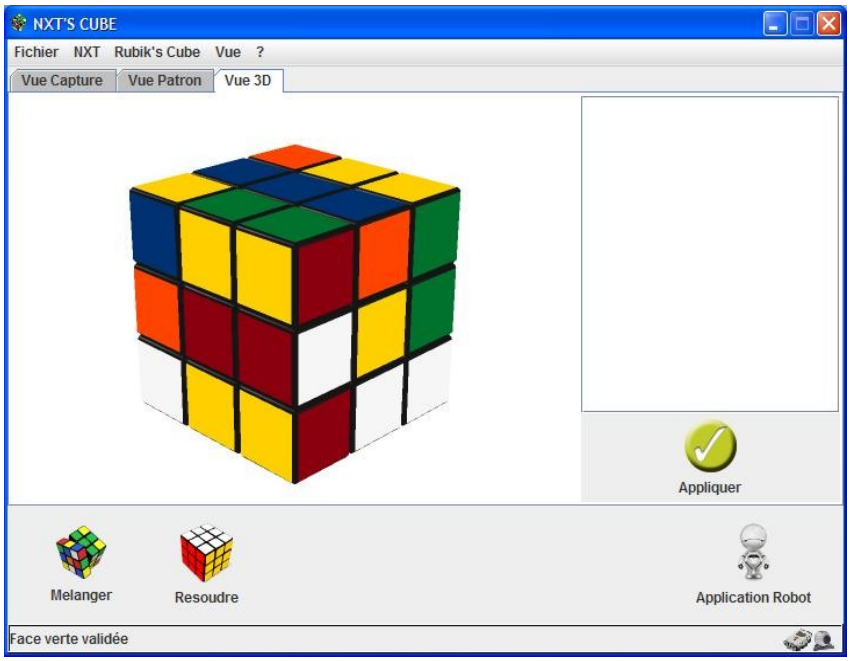


Fig 5.1 Vu en 3D

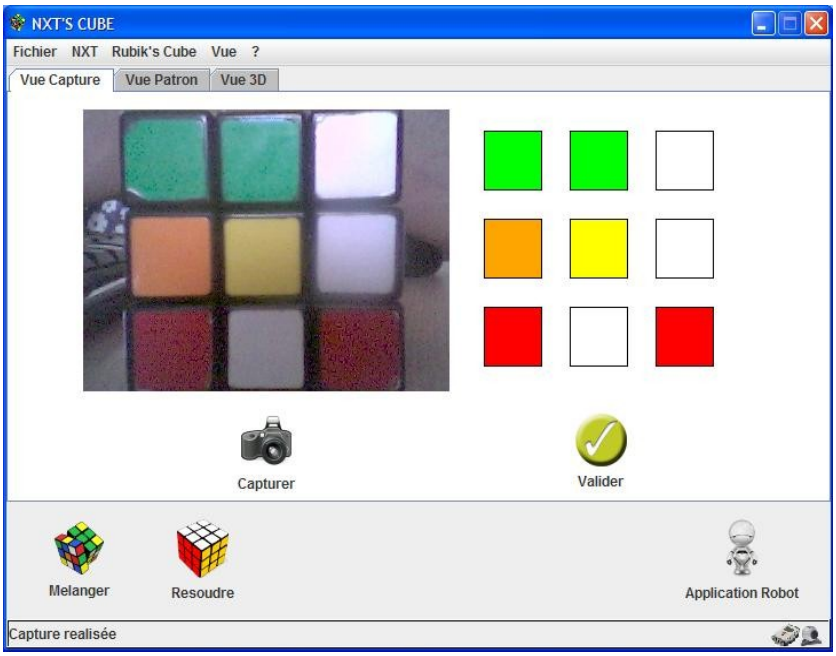


Fig 5.2 Vu Capture

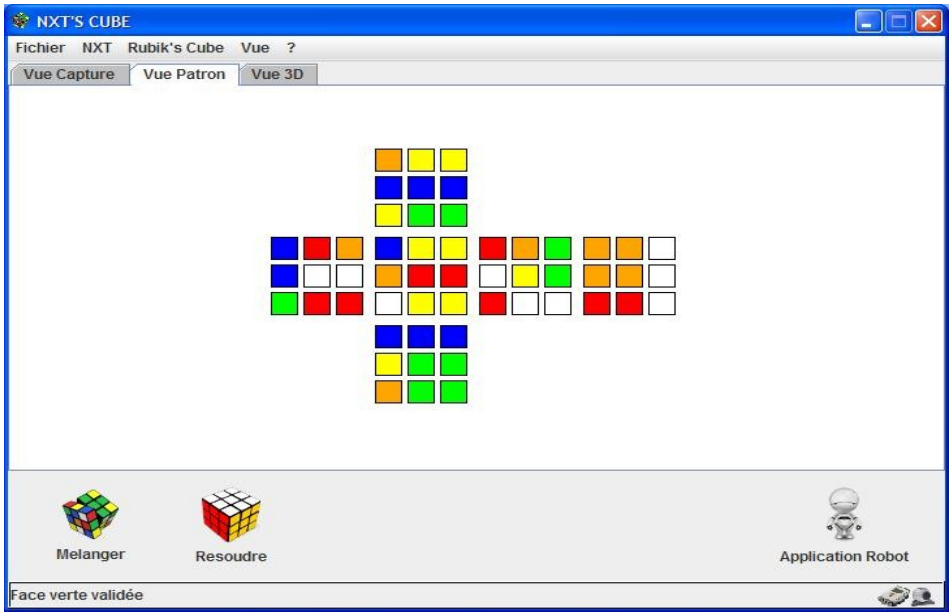


Fig 5.3 Vue Patron

L'initialisation d'un cube peut se faire par webcam ou directement sur le patron en cliquant sur les différentes facettes du Rubik's Cube, cela aura pour effet de les colorier.
Au niveau de la vue permettant la capture on a accès direct au flux de la webcam ainsi qu'un panel graphique permettant de corriger les erreurs de détection.
On peut aussi générer un cube dans un état aléatoire à l'aide du bouton dédié.

5.2 Interface et visualisation

La vue en 3 Dimension permet la manipulation complète du cube. Faire pivoter le cube selon tous les axes est possible a l'aide de la souris, de même que l'application de n'importe quel mouvement.
Le robot lui aussi peut manipuler le cube . Tous les mouvements ont été codé pour que les moteurs et capteurs permettent de les appliquer.
Une zone de texte a été ajoutée afin de pouvoir appliquer au cube une série de mouvements. Il suffira d'inscrire des mouvements à effectuer (en respectant la notation imposée) dans cette zone de texte pour les appliquer par la suite, soit au cube en 3D, soit au robot.

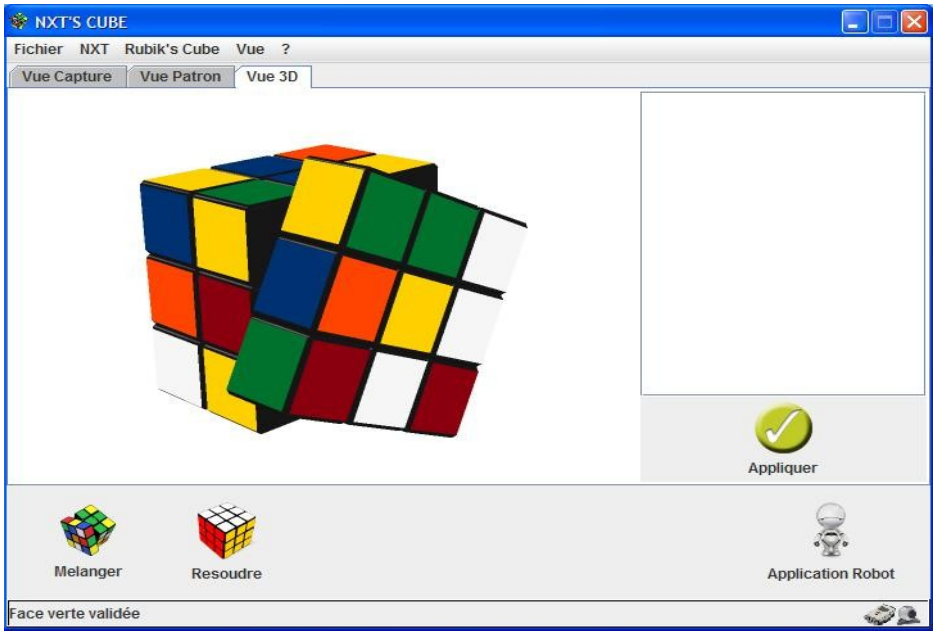


Fig 5.4 Vue 3D en résolution

5.3 Capture et contrôle de validité

La capture et le contrôle de validité à été établi dans la vue Capture.
Dans cette vue nous avons en direct le flux de la webcam ainsi qu'une représentation graphique de la face détectée, en effet lors de la prise d'une photo l'application va colorier la représentation graphique avec les couleurs détectées. L'utilisateur avant de valider la face vérifiera que l'application ne s'est pas trompé et pourra éventuellement corriger d'un simple clique sur la facette erronée.
L'utilisateur peut aussi corriger sa modélisation directement sur le patron si une face erronée a été validée.
La barre d'état de l'application signal si une modélisation est valide ou non.

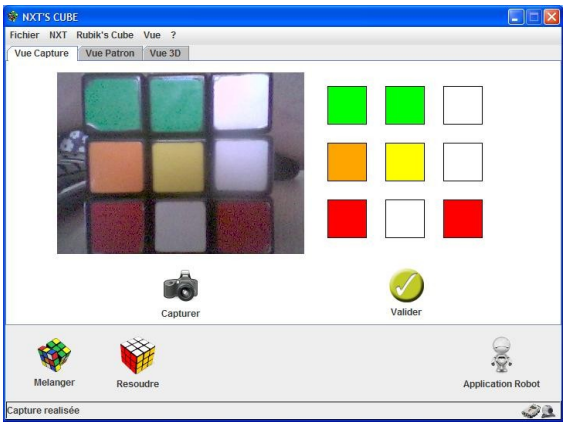


Fig 5.5 Vue Capture(2)

5.4 Résolution et application

La résolution du Rubik's Cube se fait à l'aide de l'algorithme de Kociemba implémenté par l'API CubeTwister. Cet algorithme permet de trouver une solution pour n'importe quel état valide en 22 coups au maximum. Cette résolution permet une application rapide par le robot. En effet pour un coup possible le robot peut effectuer 1 à 4 mouvements pour positionner le cube comme il le faut. Il nous était indispensable d'avoir un algorithme opérationnel et efficace .

L'application de la solution par le robot peut être faite que s'il y a connexion entre l'application principale et l'application robot . Cette connexion est possible en USB ou en Bluetooth. Un petit langage de communication a été codé pour permettre cette liaison .

Chapitre 6

Comparaison : Prévisions / Réalisations

Nous allons dans ce chapitre mettre en évidence les différences entre les prévisions faites et le travail réalisé .

6.1 Comparaison : Cahier des charges / Travail réalisé

6.1.1 Fonctionnalités obligatoires

Toutes les fonctionnalités obligatoires ont été implémentées et sont fonctionnelles. Au-delà de ca nous avons permis à l'application principale de pouvoir être utilisée sans webcam et/ou robot.
L'application robot communique avec l'application principale en USB ou en Bluetooth.
L'api CubeTwister qui sera présenté un peu plus bas nous a permis d'utiliser un très beau moteur graphique 3D interactif déjà implémenté, pour un rendu parfait.

6.1.2 Fonctionnalités optionnelles

En option nous aurions aimé permettre à l'utilisateur d'obtenir d'un simple clique un état remarquable du Rubik's cube (CF : annexe Pattern's). Cependant une liste de patterns est disponible dans la documentation de l'application, l'utilisateur pourra écrire ces mouvements dans la zone de saisie de mouvements de l'application et ainsi reproduire ces patterns.
Nous avons également instauré un apprentissage de la webcam. A chaque utilisation de la webcam par l'utilisateur un histogramme de couleurs est complété. Vu que l'utilisateur corrige la détection des couleurs on a pensé utile d'enregistrer ces corrections pour ne plus refaire les erreurs. Du coup après quelques utilisation dans un environnement comparable (surtout en luminosité) la détection des couleurs faite par l'application gagne énormément en précision.

6.2 Comparaison Planning prévisionnel/Planning

Le planning qui a été établi en début de semestre était limité à 12h par semaine d'utilisation de la salle robot. Le planning modèle établi comme guide sur le site de M. Pellier à été suivis rigoureusement. Les vacances scolaires n'étant pas comptées dans ce planning, elles nous auront permis de faire face aux différents imprévus survenu (ex : reconstruction du robot) et elles nous auront surtout permis de pouvoir utiliser la salle robotique sur une plage horaire plus importante.

Chapitre 7

Choix et librairies utilisés

7.1 Choix du langage

Le choix du langage est imposé avec le sujet . Cependant comme nous aavons choisi le sujet nous avons aussi choisi en fonction du langage .
Le langage de programmation qui a été utilisé dans se projet est JAVA .
Comme java represente le langage que nous maitrisons le plus nous avons trouvé interessant de finaliser notre formation (en licence) Java par un projet conséquent afin de maitriser parfaitement tous les derniers acquis de l'UE PROG5.

7.2 Choix des Libraires

7.2.1 *API CubeTwister*

L'api CubeTwister est une référence. Il semble être le plus populaire. Ce programme est libre et Open Source. Il permet de traiter un cube 3 par 3 , mais aussi des Cubes très étonnant de taille et forme variable. Un algorithme de résolution du cube 3 par 3 est déjà implémenté. Cet algorithme est celui de kociemba. Nous avons réutilisé les objets graphiques (le cube 3 par 3) de cette API ainsi que son algorithme de résolution. Nous voulions apporter une évolution a cette application en permettant la capture par webcam et la résolution par un robot plutôt que d'implémenter une version moins performante.

Nous avions pensé à intégrer dans notre application une extension de son algorithme de résolution pour résoudre les cube de type 4 par 4, fonctionnalités absente de CubeTwister mais cela représenté une charge de travail supplémentaire que nous ne pouvions assumer.

Sur le site de officiel de cube twister la javadoc n'est pas mise a jour, nous avons eu

quelques difficultés à l'installation. Nous avons enfin réussi en recompilant les fichiers sources et en régénérant une documentation JavaDoc.

7.2.2 *API Java Media Framework*

L'api Java Media Framework permet d'incorporer des données de type audio ou video dans un programme java. En effet cet API fournit un support permettant de capturer et stocker des données de type audio ou vidéo. L'installation de cette api n'est pas evidente non plus, en effet pour le support de la webcam une configuration précise est a réaliser sur l'API avant qu'elle ne soit utilisable, afin de configurer la webcam.

De plus cette API ne fonctionne correctement que sur windows. Beaucoup de problème ont été recensé sur les autres plateforme, surement du à la reconnaissance des peripheriques USB et des drivers sur linux ou Mac.
Cet Api efficace une fois bien paramétré sous Windows fonctionne parfaitement mais fait perdre la portabilité de notre application.

7.2.3 *API Lejos*

L'api Lejos est complète facile a installer (les variables d'environnement s'ajoutent dorénavant automatiquement à l'installation). La documentation est claire et beaucoup d'exemple de code sont fourni afin de bien se familiariser avec les classes de cette API. Les moteurs et capteurs peuvent être manipulés directement par l'ordinateur.

7.2.4 *API Jxl*

L'api Jxl permet de manipuler des fichier de type tableurs . Nous utilisons cette api pour la gestion de notre histogramme de couleurs .
L'installation se fait facilement, il y a juste à copier le fichier jar source dans le dossier d'éclipse pour une utilisation immédiate .
Cet api est très connu et permet de générer des rapports ou autre sous la forme d'un tableau et de les manipuler dans un programme java.

Chapitre 8

Description des différents modules de l'application

8.1Capture

8.1.1 Description du module

Ce module représente tous les éléments et classes nécessaires à la capture de l'état du Rubik's Cube à l'aide de la webcam

8.1.2 Points forts, points faibles

Sous Windows notre application détecte toutes les webcams. Évidement il faut que la webcam soit installée et fonctionnel sous Windows. L'application arrive à récupérer la liste des webcams installées sur l'ordinateur (en générale y en a qu'une seule et les test une à une .

En revanche l'utilisation de l'api JMF nous permettant de faire cela est bien multiplateforme mais pose de gros problème pour la capture du flux vidéo sur les plateformes autres que Windows.

8.1.3 Problèmes rencontrés

Ce projet fut semé de problèmes en tout genre auxquels nous avons du faire face.

Mécanique :

Des le début, un robot déjà monté nous a été proposé, mais celui-ci n'était pas correctement monté (le bras n'attrapait pas le Rubik's Cube) et nous aura fait perdre de précieuses semaines à tenter de lui trouver solution. Finalement les plans d'un

autre robot ont été achetés ce qui nous aura permis d'avoir un robot bien plus stable et fonctionnel.

Capture :

La détection de couleurs fut un mini challenge dans notre projet. Nous somme passé par plusieurs étapes, les couleurs traitées était au début au format RGB puis pour une plus grande précision nous les avons convertit au format LAB ce qui nous aura permis de mieux géré la lumière. Malgré ce changement la qualité de la détection n'était toujours pas satisfaisante. Nous avons donc établit un histogramme de couleurs pour permettre à l'application l'apprentissage et donc l'amélioration des détections faites pour une reconnaissance accru en fonction de la webcam et de l'environnement.

8.2 Modélisation

8.2.1Description du module

Ce module à pour bute de modéliser le Rubik's cube en mémoire et de permettre sa manipulation.

8.2.2 Points forts, points faibles

Nous avons modélisé nous même le patron du Rubik's Cube mais pour un rendu spectaculaire l'utilisation de l'API Cube Twister nous a apporté une visualisation en 3D et une interactivité avec le cube.
La modélisation est simple à l'aide de cette api , la manipulation du cube également.

8.2.3 Problèmes rencontrés

La prise en main de l'api ne fut pas aisé. L'installation nous a posé quelques petits problème au début mais une fois les sources recompilées et la javadoc régénéré nous avons pu nous familiariser avec cette API.
Cette api est très étoffé et repérer les différents objets qui peuvent nous être utile ne fut pas évident. Cependant avec du temps et un peu de persévérance nous avons réussi a manipuler certaine fonctionnalités de cette api.

8.3 Résolution

8.3.1 Description du module

Ce module a pour but la résolution du Rubik's cube à partir de n'importe quel état.

8.3.2Points forts, points faibles

Notre résolution se base sur l'algorithme de Kociemba implémenté dans l'api CubeTwister. Cet Algorithme à été mise au point par Herbert Kociemba en 1992 en améliorant un algorithme existant celui de Morwen B. Thistlethwaite. Cet algorithme ne peut être utilisé que par un ordinateur car il se sert des tables de mouvements pour calculer une solution.

La résolution est calculé très rapidement (2 secondes en moyenne) et la solution n'excède jamais 22 mouvements.

8.3.3 Problèmes rencontrés

Nous n'avons pas eu de réel problème dans ce module. Une fois l'api CubeTwister prise en main nous avons su facilement utiliser la résolution qu'offrait l'API.

8.4 Application au robot

8.4.1 Description du module

Ce module va permettre au robot de recevoir la solution de l'application principale et de l'appliquer sur le Rubik's Cube.

8.4.2 Points forts, points faibles

L'application robot permet de commander les capteurs et moteurs pour que le robot puisse exécuter l'ensemble des mouvements possibles, et de permettre à l'application de se connecter afin qu'il puisse recevoir la solution. Le robot doit aussi mémoriser la position du cube tout au long de l'application.
La liaison entre les deux applications peut se faire en Bluetooth ou USB.

Les limites de ce module sont atteintes par la capacité mécanique du matériel. En effet les capteurs et moteurs ne sont pas d'une grande précision. Si la solution est trop longue a appliquer des problèmes mécaniques peuvent apparaitre (baisse de l'intensité de la batterie ramollit le bras qui entraine un échec de la rotation verticale du cube , la pince du bras se desserre d'elle-même après une longue utilisation cela peut provoquer un échec de la rotation horizontale du cube) . Ces problème sont connus du monde des Legos Mindstorms. L'auteur du livre qui nous a permis la construction du robot prévient qu'il faut jouer avec l'écart de la pince pour une résolution sans faute. A-t- il oublié de précisé qu'il faut de préférence faire le plein de la batterie avant une résolution?

8.4.3 Problèmes rencontré

Les problèmes rencontrés ont été nombreux mais principalement liés à la construction du robot. En effet nous avons en début de semestre perdu pas mal de temps avec une version d'un robot qui n'aurai jamais pu résoudre le Rubik's Cube, car il a été construit avec les plan de la version 1.0 du NXT alors que le robot fourni était la version 2.0 du constructeur. Sur internet nous avons trouvé beaucoup de vidéos de petits robots de type legos qui résolvait le Rubik's cube mais aucun plan de construction. Nous avons résolut notre problème à la fnac avec l'achat d'un livre (Build A Rubik's Cube Solver de Daniele Benedettelli)

Chapitre 9

Extensions et améliorations possibles

Dans ce chapitre on a voulu proposer des extensions et améliorations possible à apporter au projet. Certaines sont difficilement réalisable cependant d'autres améliorations sont facilement implémentables.

9.1 Extensions

9.1.1 Acquisition

Une possibilité d'extension au niveau de l'acquisition par webcam est possible en permettant la capture d'un Rubik's Cube 4 par 4.
Cette extension demandera une petite modification de la capture afin d'augmenter le nombre de carré (un par facette) et la modélisation du cube peut être faite à l'aide de cube twister.

9.1.2 Robot

Dans la même optique mais avec une difficulté complètement différente une autre extension du projet serai de faire résoudre un Rubik's Cube 4 par 4 par le robot. Cette extension nécessite de reconcevoir complètement le robot. Et d'implémenter un

9.2 Améliorations

9.2.1 Acquisition

La détection des couleurs n'étant toujours pas parfaite, une amélioration pourrait etre réaliser sur cette fonctionnalité.
Nous aurions pu essayer de prendre plusieurs photos par face dans different sens

possibles (seulement 2). En effet la lumière joue un rôle important lors d'une erreur dans la détection. Prendre plusieurs photos permettrai à l'application de les comparer et de déduire plus facilement les différentes couleurs de la face du cube.

Enfin, il aurait été intéressant de résoudre le problème de portabilité avec JMF.

9.2.2 Robot

Lors de l'application de la solution par le robot , le robot exécute tour à tour tous les mouvements nécessaire à la résolution du Rubik's Cube. Cependant le bras mécanique n'est pas infaillible et lors d'une longue résolution il peut rater un mouvement à cause des pièces legos qui se desserrent au fur et à mesure. Une amélioration intéressante, avec une détection des couleurs améliorée, serai de pouvoir vérifier à chaque rotation du cube que la face visible sur la webcam est bien celle attendu (c'est à dire que le robot à bien appliquer le mouvement). Comme cela en cas d'erreur de mouvements du robot il recalculera la solution et se corrigera de lui-même .

9.2.3 Interface graphique

L'interface graphique offre déjà une bonne ergonomie d'utilistion.
Cependant une amélioration possible serai de synchroniser parfaitement les threads de résolution sur l'application et sur le robot.
Étant donné que le cube 3D est vraiment manipulable selon tous les axes on pourrai synchroniser le cube 3D avec le Rubik's Cube que manipule le robot (rotation de face, pivotage du Rubik's Cube...).

9.2.4 Divers

Le robot permet de jouer des sons. La seule contrainte pour faire parler le robot est que la brique NXT possède une mémoire très limité.
Cependant lui faire dire quelques mots en fin de résolution ainsi qu'a la connexion aurai été sympa.

Annexes

A- Documents Wikipédia :

Vous pouvez visiter ces pages internet de Wikipédia pour avoir plus d'idée.

- Lego Mindstorms :
  http://fr.wikipedia.org/wiki/Lego_Mindstorms_NXT

- Rubik's cube :
  http://fr.wikipedia.org/wiki/Rubik%27s_Cube

- Algorithme de résolution :
  http://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube

B- Glossaire :

Cubic : On appellera cubic, les petits cubes formant le Rubik's cube. Un Rubik's Cube se compose de 26 cubics (en effet il n'y a pas de cube au centre).

Cubiste : Amateur de la pratique du Rubik's Cube.

Face : Afin d'identifier les différentes faces du Rubik's Cube nous utiliserons le standard anglais, ainsi la face de devant sera appeler F(ront), la face de derrière B(ack), puis U(p), D(own), R(ight) et enfin L(eft).

Rotation : Pour une question de simplification on appellera les différentes rotations des faces par le nom de la face elle même. Ainsi « L » représentera une rotation d'un quart de tour de la face Left dans le sens horaire, « L' » dans le sens anti-horaire, enfin « L^2 » représentera une rotation d'un demi tour de la face Left.

C- Plan de construction du Robot :

Voir sur le site du projet plan en .pdf et en vidéo.