2012-2013

PROJET SI : RUBIK'Solver

Conception, modélisation et réalisation d'une machine qui résout le Rubik's Cube

PROJET SI : RUBIK'Solver

Conception, modélisation et réalisation d'une machine qui résout le Rubik's Cube

I) Présentation du projet

Notre monde est envahi de robots. Ces derniers ont pour beaucoup le devoir de réaliser des tâches de plus en plus complexes, et cela de façon la plus autonome possible. Ainsi est née l'idée d'une machine capable de résoudre le Rubik's Cube. Automatisation, complexité, électronique, mécanique, informatique, tout y est. Notre conception initiale s'est effectuée dans cet état d'esprit général et a finalement abouti à l'ensemble des caractérisations suivantes :

FONCTIONS	CRITERES	NIVEAU	FLEXIBILITE
FP1 : Résoudre le Rubik's Cube automatiquement	C1 : Automatisation de la résolution C2 : Pouvoir le résoudre à partir de n'importe quel mélange	Complète 100%	F0 F0
FC1 : Contrôler la machine par ordinateur	C1 : Interface utilisateur simple	100% intuitive	F1
FC2 : S'adapter à la taille standard d'un cube	C1 : Cube de production officielle	5.7cm ± 1mm	F0
FC3 : Réduire le temps de résolution	C1 : Temps moyen de résolution	2 min ± 30 sec	F2
FC4 : Respecter les normes de sécurité	C1 : Normes en vigueur	100%	F0
FC5 : Ne pas polluer et limiter les coûts	C1 : Utiliser des matériaux de récupération C2 : Utiliser des matériaux non polluants	60% ± 10% 60% ±10%	F2 F2
FC6 : Etre facilement transportable	C1 : Dimensions extérieures C2 : Masse maximale	L = l = h = 60 (mm) ± 10% 10kg	F2 F2
FC7 : Communiquer avec l'électronique	C1 : Liaison matérielle	Câble USB	F1
FC8 : Alimenter en énergie	C2 : Réseau EDF	220V, 50Hz	F0

(Les parties en bleu sont celles qui ont fait l'objet d'une étude personnelle.)

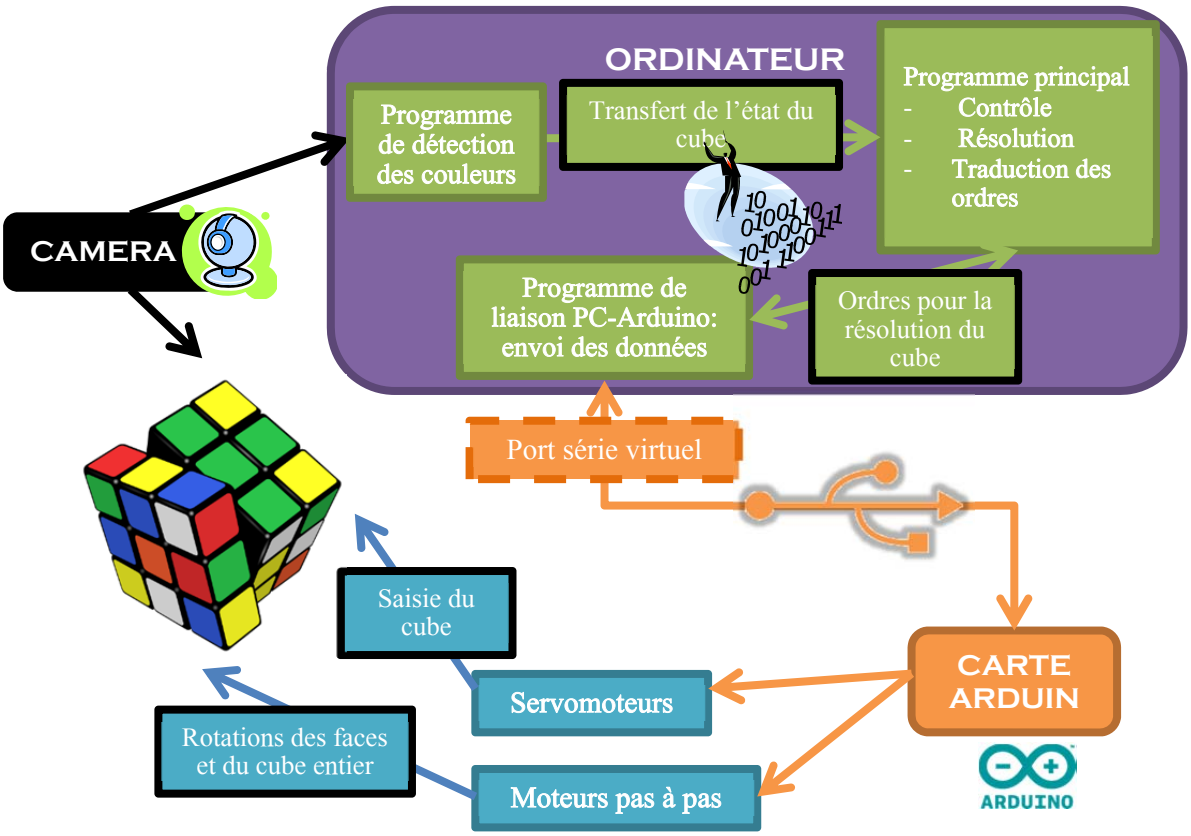
Etant responsable principalement du programme informatique et de tout ce qui a de près ou de loin attrait à la programmation logicielle sur ordinateur, il a obligatoirement fallu remplir ces fonctions :

- **Récupérer les couleurs du Rubik's Cube mélangé,**
- **Résoudre ce Rubik's Cube, trouver la série de mouvements correspondants,**
- **Traduire ces ordres en série d'actions matérielles des moteurs,**
- **Gérer le protocole entre l'électronique et l'informatique.**

Une attention toute particulière doit être donnée à la liaison entre ces parties. C'est principalement cette liaison qui sera le cœur du fonctionnement global du système. Par exemple, le protocole entre l'informatique et l'électronique a fait l'objet de tests préliminaires.

II) Conception

La conception de la machine s'est réalisée en plusieurs temps : d'abord une phase de discussion, de recherches, puis une phase de choix précis, de dimensionnements, de choix de matériaux, de librairies logicielles, de moteurs, etc. de façon plus ou moins collégiale. L'étude poussée fut, pour elle, personnelle. En est finalement ressorti l'ensemble des éléments suivants aboutissant au



fonctionnement :

Ce schéma d'organisation explique la globalité du fonctionnement de la machine. Nous nous intéresserons dans ce dossier précisément à la partie informatique. Ajoutons tout de suite, qu'accessoirement, le programme peut posséder plusieurs caractéristiques supplémentaires pour améliorer la fonction contrainte **C1 : Interface utilisateur simple**.

- Posséder une interface graphique utilisable par tout le monde,
- Posséder des graphiques clairs et agréables,
- Posséder un visuel en temps réel du Rubik's Cube, en 2D voire 3D.

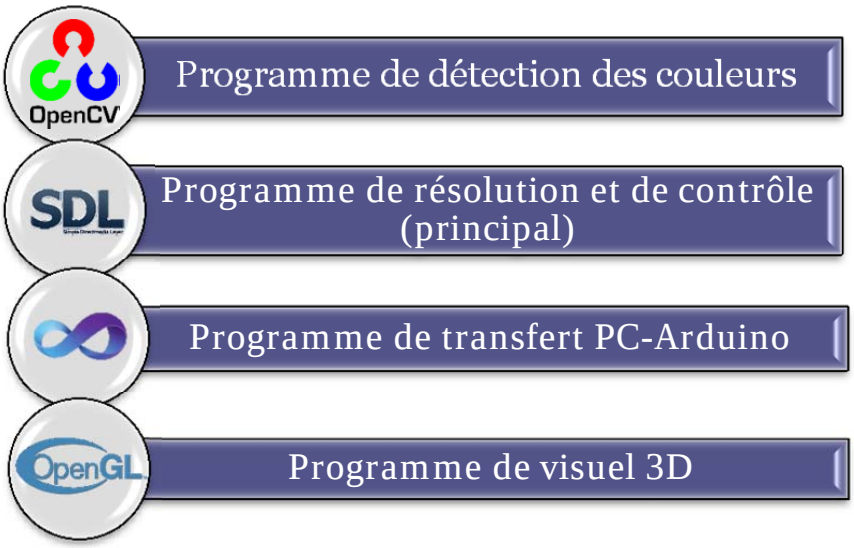
S'y ajoutent une multitude de petites fonctionnalités annexes, peu importantes fonctionnellement, mais utiles, comme un chronomètre, l'affichage des mouvements effectués, des enregistrements fonctionnels pour comprendre ce qui n'a éventuellement pas marché lors d'une résolution, etc.

III) Choix du langage et des librairies

Connaissant les langages C et C++, la recherche s'est orientée vers ces outils. Pour l'analyse d'images, la librairie OpenCV fut adoptée, qui permet la récupération de flux vidéo. Quant à la résolution du Rubik's Cube, elle découle plus de la logique que de fonctions complexes. Elle ne

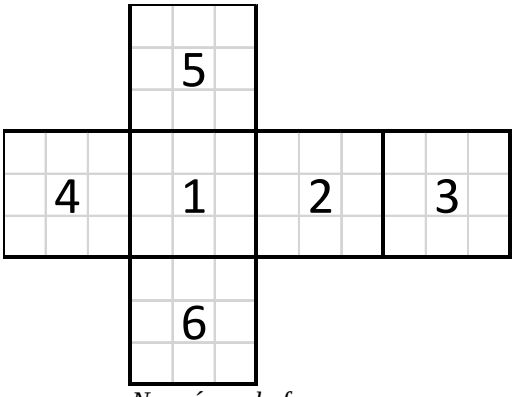
nécessite que de la manipulation de variables, de tableaux, de chaînes, etc. Bref, des fonctions simples incluses dans les librairies de base du langage C.

Pour le visuel, la manipulation d'images et l'organisation d'une fenêtre visuellement agréable, le choix s'est orienté vers la SDL. Pour le transfert série, nous avons finalement choisi le C++, où la classe IO :: Ports :: Serial donnait des fonctions d'ouverture, de lecture et d'écriture très confortables pour la programmation. Côté carte électronique, elle se programme à l'aide d'un langage très proche du C, et contrôle grâce à certaines librairies incluses dans le logiciel de l'IDE les moteurs décrits dans la première partie. Enfin, le module de visuel 3D a finalement nécessité un moteur 3D. Notre sélection finale, après discussion, fut faite en faveur d'OpenGL, librairie très puissante et ouverte. Soit le résumé :

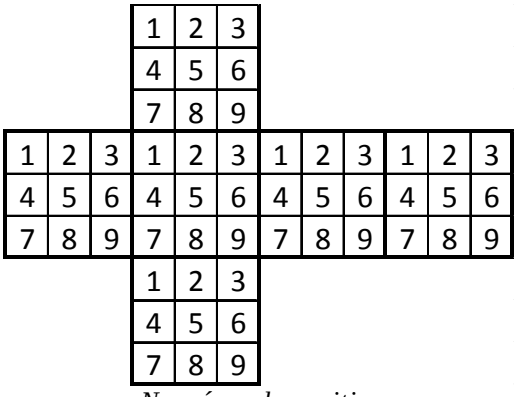


IV) Logique des programmes sur ordinateur

J'ai d'abord cherché à modéliser qualitativement le Rubik's Cube dans un code. La solution choisie a été un tableau à deux dimensions, de la forme *tableau*[*numéro face*][*numéro position*], suivant un patron étalé du cube, comme suit :



Numéros de face



Numéros de position

Grâce à ce tableau, il est possible de travailler sur le Rubik's Cube spatialement. On note que le travail sur le cube ne se fait pas dans n'importe quel sens. La résolution s'effectue toujours spatialement dans le même référentiel de face. Ainsi, la face 1 est le devant, la 5 le dessus, et la 2 la droite. Le sens positif de rotation est le sens horaire, le sens négatif le sens trigonométrique. Cette modélisation permet un travail de logique sur le Rubik's Cube. (Voir image PROGRAMME PRINCIPAL en annexe)

La seconde étude s'est portée sur l'analyse des couleurs. La librairie OpenCV utilise un système de récupération de flux en tant que successions d'images en temps réel. Le programme doit pouvoir récupérer l'état du Rubik's Cube tel qu'il est défini dans un tableau-type expliqué. Il doit donc, pour faire simple, donner le tableau initial sur lequel l'algorithme principal va travailler. En prévoyant l'orientation du cube grâce aux mouvements de bras, on peut ainsi définir où se situera chaque plage sur l'écran, et en fonction de la face à quel endroit cela correspond.

La détection d'une couleur se fera par moyenne de l'ensemble des composantes, puis par comparaison par rapport aux couleurs du centre : on crée un coefficient de proximité de couleur. Plus le coefficient est petit, plus la couleur est proche. Pour calculer ce coefficient, on fait la somme des différences des moyennes des composantes de pixels d'un échantillon par rapport à celles des centres, spatialement fixes. Cet algorithme va ensuite transmettre au programme principal, par fichier, l'état initial du cube. (Voir image VISUEL CAMERA en annexe)

Le troisième programme de ma partie était le programme de liaison PC-Arduino. Ce dernier ne réalise que quelques tâches simples. Il récupère la série de mouvements traduits pour la machine, et gère le protocole de transfert. Il réalise cette boucle (voir à droite) jusqu'à aboutissement des mouvements. Il gère aussi la gestion de la pause et de l'actualisation des différentes représentations du cube. (Voir image PROGRAMME PC-ARDUINO en annexe)

Autre et dernier programme PC, l'affichage du Rubik's Cube en 3D. Ce dernier a pour tâche de montrer l'état du cube en temps réel, et de surcroît de montrer les mouvements effectués. Pour cela, nous avons d'abord défini spatialement des points, un par un, auxquels nous avons attribué des surfaces colorées. Le programme va d'abord récupérer l'état initial du cube. Par une série de transformations, les mêmes que celles du programme principal, on va actualiser la position du cube, puis attendre le mouvement suivant. (Voir image CUBE 3D en annexe)

V) Evolutions possibles

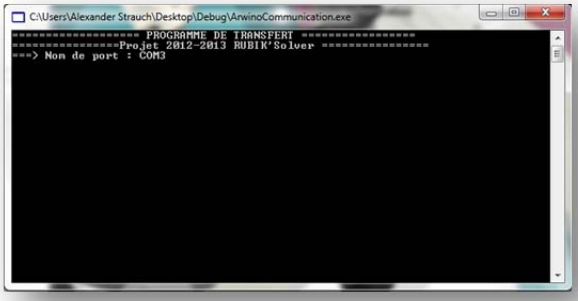
Tous les programmes souffrent plus ou moins de difficultés diverses. L'affichage en temps réel du Rubik's Cube, parfois s'arrête par erreur de lecture de fichiers. Toutes les procédures classiques de programmation ont pourtant été appliquées. Ce mode de communication n'étant pas très adapté, une évolution possible serait vers **la mise en place de sockets,** plus complexes cependant à prendre en main, et qui impliqueraient une refonte des protocoles de communication. Cette amélioration sera menée dans le futur. Le programme de résolution fonctionne à tous les coups (10000 essais menés en boucle concluants permettent de l'affirmer avec quasi-certitude), cependant, si on lui envoie un cube insolvable, il ne sera pas toujours capable de le « voir », et ce malgré la fonction de renvoi d'erreur. A côté de cela, la SDL étant peu optimisée, il est toujours possible de **transiter vers GTK+**, autre librairie bien plus puissante. Le problème le plus important reste cependant les erreurs de détection de couleur. Ces dernières arrivent notamment au niveau des couleurs proches, qui à cause de la piètre qualité de notre caméra et de la calibration automatique Windows rendent les différences de composantes trop proches les unes des autres. **Un système de calibration préalable et d'échantillonnage est encore à l'étude.**

VI) Conclusion

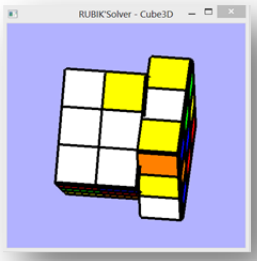
Nous avons atteint une très bonne partie de nos objectifs initiaux. La machine est capable de résoudre un Rubik's Cube. Nous n'avons pas atteint les 2 minutes 30 initialement prévues, mais nous sommes seulement deux fois plus lents, ce qui, pour une tâche si complexe, reste acceptable. Pour tout de même terminer sur une note positive, voici le résultat final de tous les programmes, en page suivante.

ANNEXE

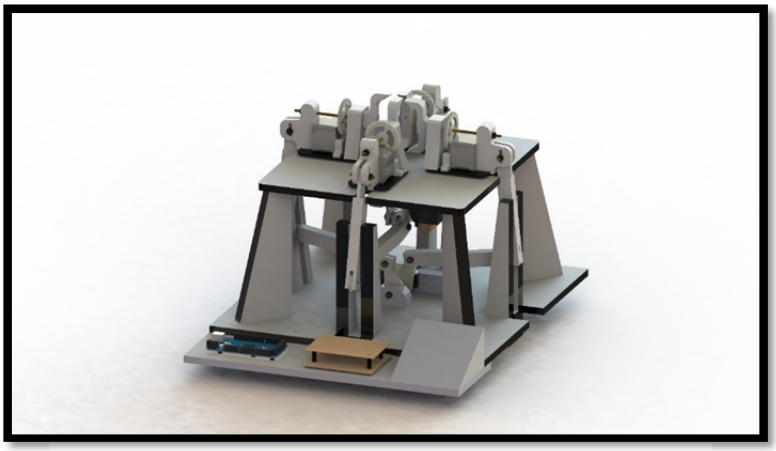
VISUEL CAMERA



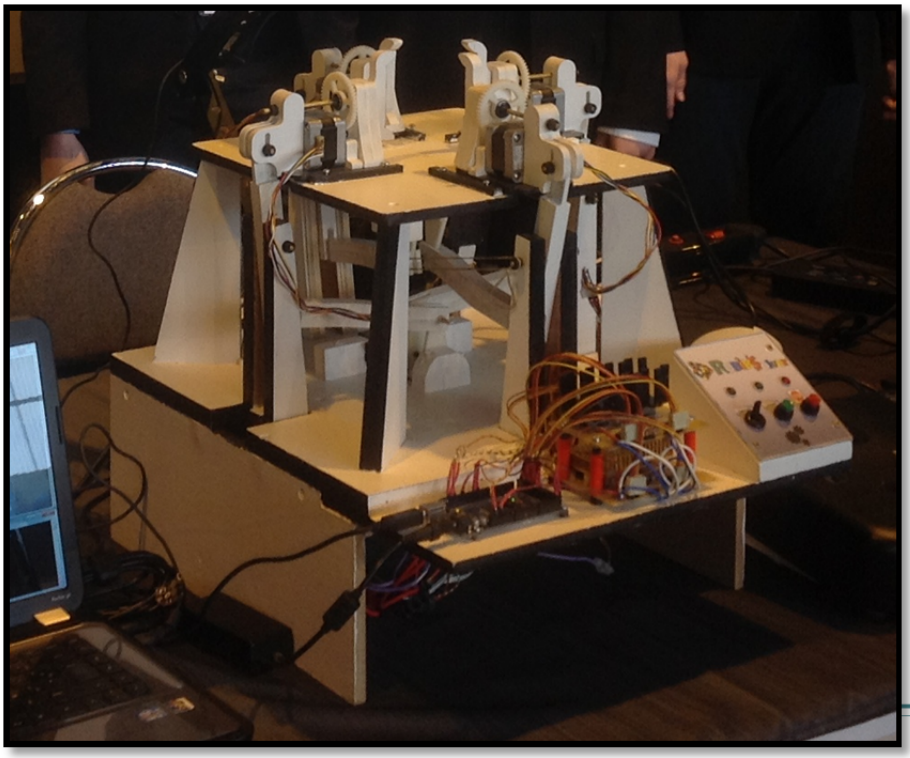
PROGRAMME PC-ARDUINO



CUBE 3D



MODELISATION DE LA MACHINE



PROGRAMME PRINCIPAL