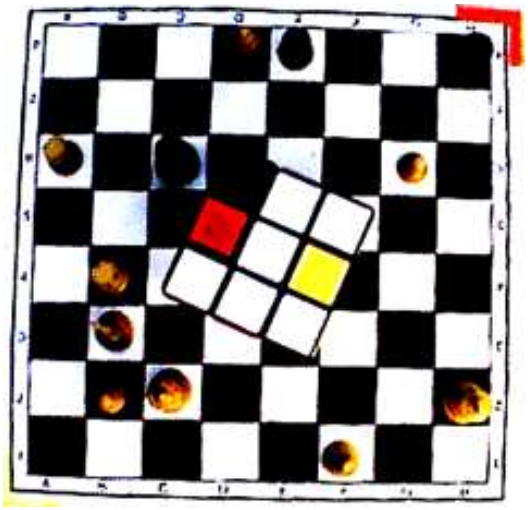
UNIVERSITÉ de Liège

Échecs & maths

Rapport du travail pratique réalisé dans le cadre du cours « Traitement numérique des images » du Professeur M. Van Droogenbroeck



Année Académique 2004-2005

Freylinger Michaël 2ELEN
Pierlot Vincent 2ELEN
Stevens Alexandre 2LINFO
Verdin Jean-François 2ELEN

1. Introduction

Ce rapport représente deux mois de travail dans le cadre du cours « *traitement numérique des images* » du Professeur M. Van Droogenbroeck. Ce cours étant optionnel, c'est un réel intérêt pour ce type de réalisations qui nous a poussé à nous inscrire à ce dernier.

Au début, tout était toutefois assez vaste et nous n'avions que peu d'idées de la façon de réaliser les trois objectifs de ce travail à savoir :

1. Décrire la position de toutes les pièces présentes sur un échiquier tournant sur un plateau à vitesse constante et filmé par 8 caméras Axis (fig 1).
2. A partir d'une caméra, marquer (en temps réel) et suivre une pièce blanche (le fou blanc sur la case blanche dans notre cas) et une pièce noire (le roi noir sur une case noire).
3. Sur l'image capturée par la caméra surplombant l'échiquier, remplacer (en temps réel) les couleurs de la face du Rubik's Cube (placé au centre de l'échiquier) par les couleurs de la face opposée.

Ce projet représentait une masse de travail énorme à réaliser sur seulement quelques semaines. C'est pourquoi, il a fallu une bonne répartition des tâches et une communication efficace dans le groupe. Grâce à cela, nous avons pu obtenir les résultats escomptés.

fig 1: Exemple d'images fournies par les caméras Axis : vue de profil à gauche et vue du dessus à droite.

2. Description de la démarche de résolution

2.1. Préliminaires

Avant de nous plonger dans le cœur même du projet, il nous a fallu nous familiariser avec l'environnement de travail, la programmation en C sous Linux. Ceci a été fait à partir du programme d'exemple disponible, qui permet facilement d'afficher correctement les images fournies par les caméras Axis sur le réseau intranet et internet de l'Université.

2.2. Résolution de la première partie

2.2.1. Détection des cases occupées par une pièce

Pour compléter cet objectif, il nous semble naturel d'utiliser la vue du dessus (caméra port 8007). C'est en effet la seule vue où les cases non recouvertes par le Rubik's Cube sont visibles en permanence.

Dans cette vue, il nous faut connaître avec précision la position de l'échiquier. À cette fin, deux gommettes de couleur ont été placées sur deux coins opposés de l'échiquier. Nous les localisons. Ceci nous permet de connaître la position de toutes les cases de l'échiquier.

Ensuite, pour chaque case, nous déterminons, s'il y a une pièce et si cette pièce est blanche ou noire, étant entendu qu'aucune case occupée par le Rubik's Cube ne peut contenir de pièce.

2.2.2. Détermination du type de chaque pièce

La vue du dessus délivrant peu d'informations sur les différentes pièces, il nous semble logique de travailler sur la caméra qui nous est attribuée pour l'objectif 2 (caméra port 8011).

Grâce aux matrices projectives en coordonnées homogènes, nous pouvons automatiquement passer d'un pixel dans la vue du dessus à un pixel dans la vue de profil. Partant d'un pixel du centre d'une case contenant une pièce (dans la vue du dessus), nous pouvons donc définir une région d'intérêt dans la vue de profil.

Ceci n'est pas réalisé pour chaque image. Nous déterminons les moments où le champ de vision est libre. Ensuite, nous effectuons une segmentation dans cette région d'intérêt afin d'obtenir un objet (fig 2).

Comment caractériser correctement une pièce à partir de cet objet ? Ici réside tout le

problème. Nous utilisons une méthode de corrélation entre signatures.

Nous projetons tout d'abord l'objet sur l'axe vertical pour obtenir une signature. Ensuite, nous la corrélons successivement avec les signatures des différents modèles de pièces. La corrélation est basée sur le coefficient de corrélation linéaire.

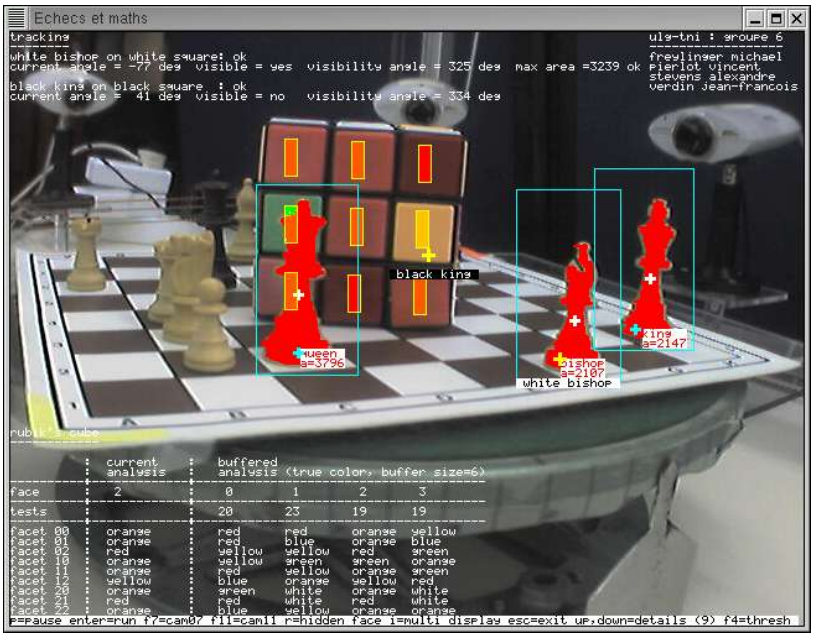


fig 2: Segmentation, reconnaissance des modèles.

Pour les pièces noires, il n'a pas été nécessaire de recourir à cette technique. Dans la vue du dessus, on peut déterminer la pièce d'aire minimale (la tour noire) , les deux autres pièces étant fixées pour l'objectif 2 : la dame noire, sur une case blanche et le roi noir, sur une case noire.

2.3. Résolution de la deuxième partie

En nous appuyant sur l'objectif 1, nous connaissons la position des pièces à suivre. Par conséquent, nous pouvons effectuer un « tracking » en marquant les pièces dans la vue de profil (fig 3).

De la même façon que pour le premier objectif, nous pouvons déterminer les moments où la pièce suivie est obstruée par le Rubik's Cube dans la vue de profil. Ainsi, nous calculons l'angle de rotation pendant lequel la pièce est visible.

Pour l'aire maximum de la pièce, nous retenons l'aire des pièces après la segmentation dans la vue de profil.

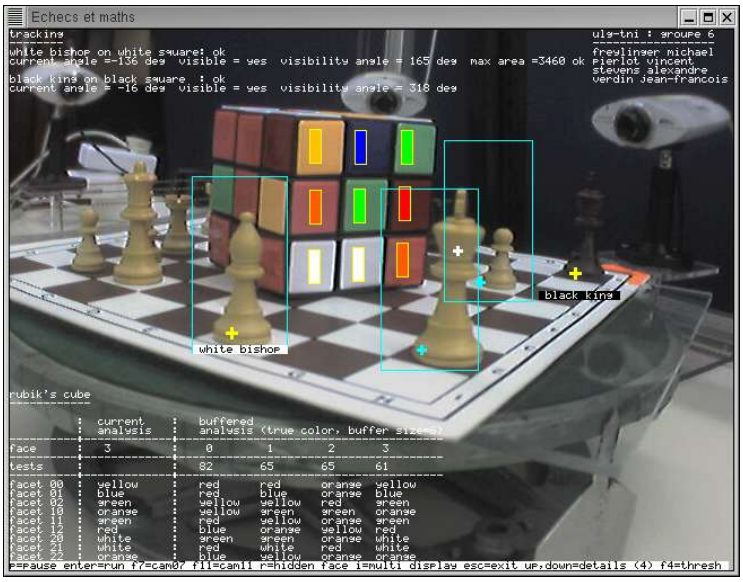


fig 3: Tracking du roi noir et du fou blanc.

2.4. Résolution de la troisième partie

Partant également de l'objectif 1, nous connaissons la position du Rubik's Cube dans la vue du dessus et donc la position des facettes. De même (encore grâce aux matrices projectives), nous pouvons déduire la position des facettes dans la vue de profil.

Nous définissons des régions d'intérêt sur les facettes et déterminons la couleur majoritaire pour chaque facette par seuillage par une segmentation simple et robuste(fig 4).

Une fois la couleur de toutes les facettes déterminée, nous pouvons déduire, grâce à un algorithme, les couleurs de la face cachée. Enfin, nous remplissons par ces couleurs les facettes du Rubik's Cube dans la vue du dessus.

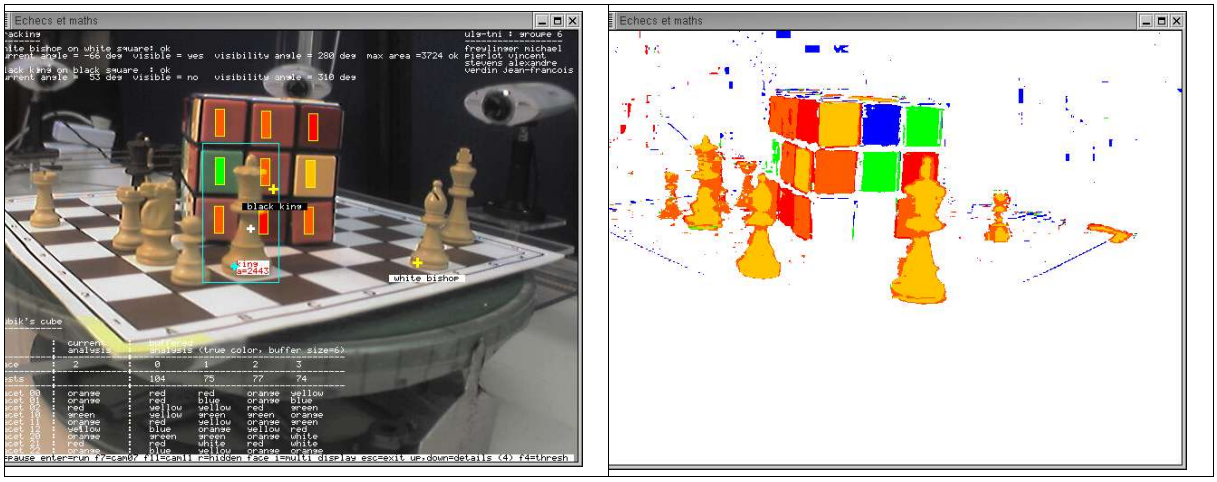
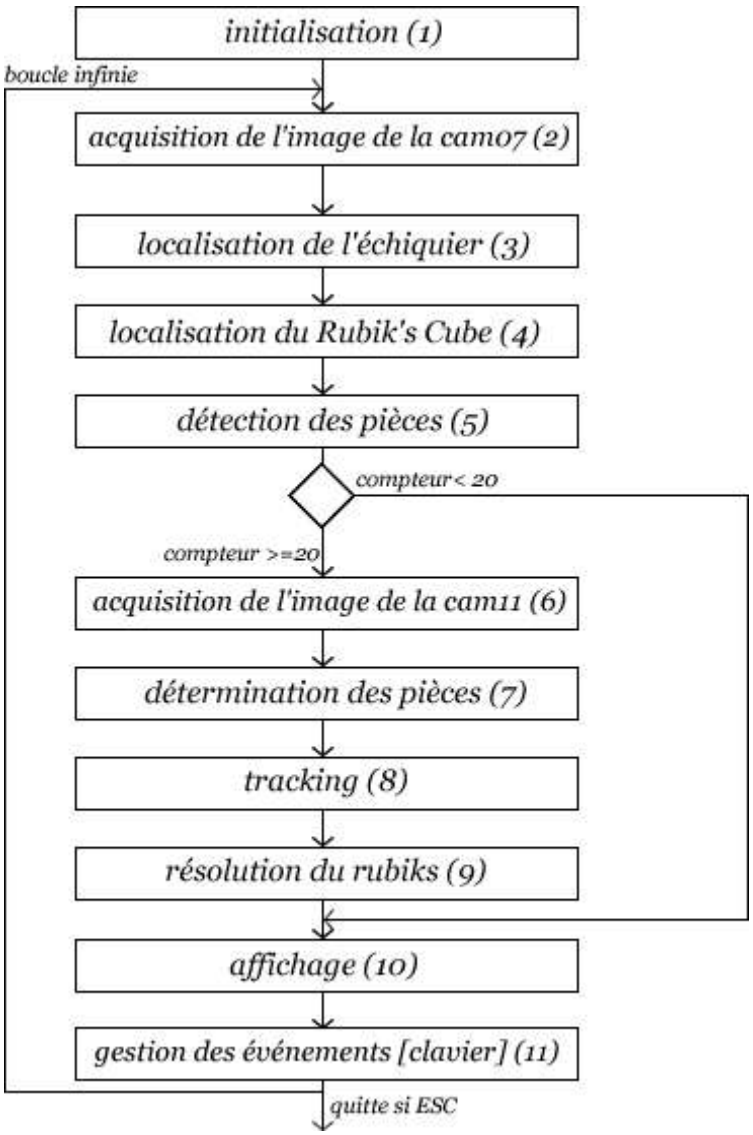


fig 4: Segmentation pour déterminer les couleurs :à droite sur une image entière et à gauche sur des parties de facettes.

3. Schéma-bloc



4. Détail des parties du programme

4.1. Acquisition de l’image

Le programme d’exemple possédait déjà des fonctions d’accès au réseau : connexion, acquisition d’image, … Cependant, nous les avons réécrites complètement en simplifiant au maximum le code.

En effet, le code déjà présent ne permettait pas d’accéder une deuxième fois à une même caméra après être passé sur une autre. Ce code nous a paru très compliqué à modifier il nous a semblé moins long et plus instructif de recréer un code d’accès au réseau. Celui-ci est repris dans le fichier « http.c » (avec son fichier d’en-tête « http.h »). Le passage par fichier lors de l’acquisition de l’image a également été remplacé par un passage par mémoire afin d’augmenter la vitesse d’exécution.

Nous avions pensé à implémenter des « threads » pour augmenter la vitesse du logiciel : un pour l’acquisition, un pour la décompression, un pour le traitement et un pour l’affichage. Cependant, ceci n’était pas une priorité et nous ne l’avons pas fait, faute de temps.

Nous effectuons l’acquisition d’une image dans la vue du dessus (caméra port 8007). Nous stockons l’heure et le temps d’acquisition, ce qui nous permet de mesurer la fréquence de traitement.

Nous avons également modifié le calcul de cette fréquence. La fréquence moyenne était obtenue auparavant à partir du nombre total d'images acquises depuis le lancement du programme (« *running mean* »). Au fil du temps, la moyenne voyait donc son "inertie" augmenter et la valeur calculée et affichée devenait de moins en moins significative du nombre d’images traitées par seconde. Nous avons remplacé cette valeur par une moyenne mobile.

Les images sont acquises au format JPEG et décompressées en format RGB. Or, nos traitements nécessitent également de manipuler les images dans l’espace HSV. La conversion RGB->HSV ne nécessite pas un calcul important en soi, mais nous devons l’effectuer sur des images entières. Aussi, pour optimiser la vitesse de traitement, nous créons des « look-up tables » en teinte et saturation lors de l'initialisation du programme. Cependant, ceci prenant un temps important, nous avons créé des fichiers contenant ces « look-up tables » (« slut.dat » et « hlut.dat » : 16MB chacun). Nous obtenons ainsi un démarrage plus rapide (très utile lors des tests). Si le programme ne trouve pas ces fichiers, il les crée en RAM.

Nous pouvons afficher au besoin l'image acquise en teinte et saturation après cette conversion (fig 5).

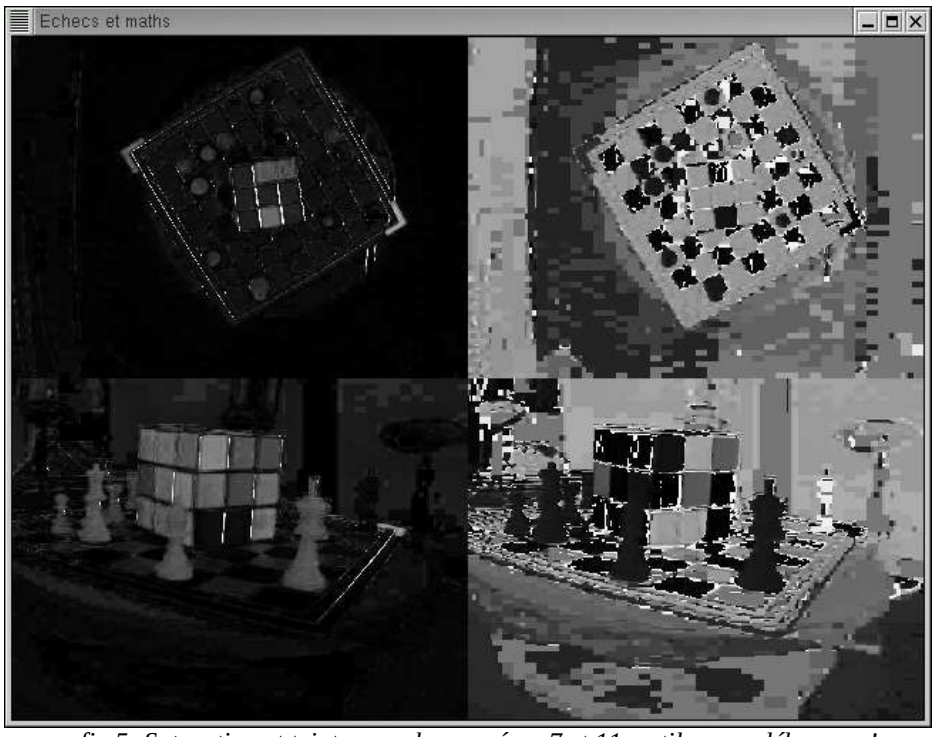


fig 5: Saturation et teinte pour les caméras 7 et 11 : utile pour déboguer !

4.2. Localisation de l’échiquier

4.2.1. Localisation des gommettes

Pour chaque gommette, nous procédons de la même manière. Seuillage de teinte et seuillage de saturation de l’image acquise. ET logique des deux images obtenues. Érosion du résultat pour éliminer les parasites. Recherche du barycentre de l’image obtenue qui correspondra au centre d’une gommette.

Cette détection de gommettes s’effectue telle quelle lors de la première acquisition. Ensuite, nous déterminons par prédiction (connaissant la vitesse de rotation de l'échiquier) une région d'intérêt dans laquelle nous réalisons la détection de gommettes . Cela améliore le temps d'exécution. De plus, en travaillant sur des régions d'intérêt, nous évitons une détection erronée des gommettes engendrée par l'éclairage du spot.

Ainsi, nous obtenons les coins de l’échiquier.

4.2.2. Détermination de tout l'échiquier

Les gommettes sont trop éloignées, aussi nous appliquons une correction aux coins de l’échiquier pour que ceux-ci correspondent exactement aux bords des cases. Nous connaissons donc une diagonale de l’échiquier. Ensuite, nous calculons les deux autres coins

de l’échiquier par la méthode de la médiatrice.

A partir des quatre coins, nous calculons les centres de toutes les cases par des opérations vectorielles, intersections de droites et calculs de barycentres.

Nous connaissons donc à ce stade une information sur la disposition de l'échiquier (fig 6) sans laquelle il serait difficile d'atteindre les objectifs.

4.3. Localisation du Rubik’s Cube

Tout d’abord, nous effectuons un seuillage en saturation sur l’image acquise. Ceci nous permet d’obtenir les facettes rouges, orange et jaunes. Ensuite, nous effectuons des seuillages RGB différents pour les facettes bleues, vertes et blanches. Ces seuillages s’adaptent aux variations d’éclairage dues au spot. Nous réalisons un OU logique de toutes ces détections. Nous identifions ensuite les différents objets et leur barycentres. Ces objets correspondent aux différentes facettes délimitées par les interstices. Nous calculons les distances entre les objets pris deux à deux. Nous retenons la distance maximale qui correspond à la diagonale du Rubik’s Cube. Ici encore, nous appliquons une correction de la diagonale pour passer aux véritables coins du Rubik’s Cube.

Si nous ne trouvons pas de diagonale, nous effectuons une interpolation des coins du Rubik's Cube à partir des coins obtenus précédemment.

La localisation du Rubik’s Cube (9) nous servira pour les objectifs 2 et 3, mais également pour éviter des recherches inutiles lors de la détection de pièces.

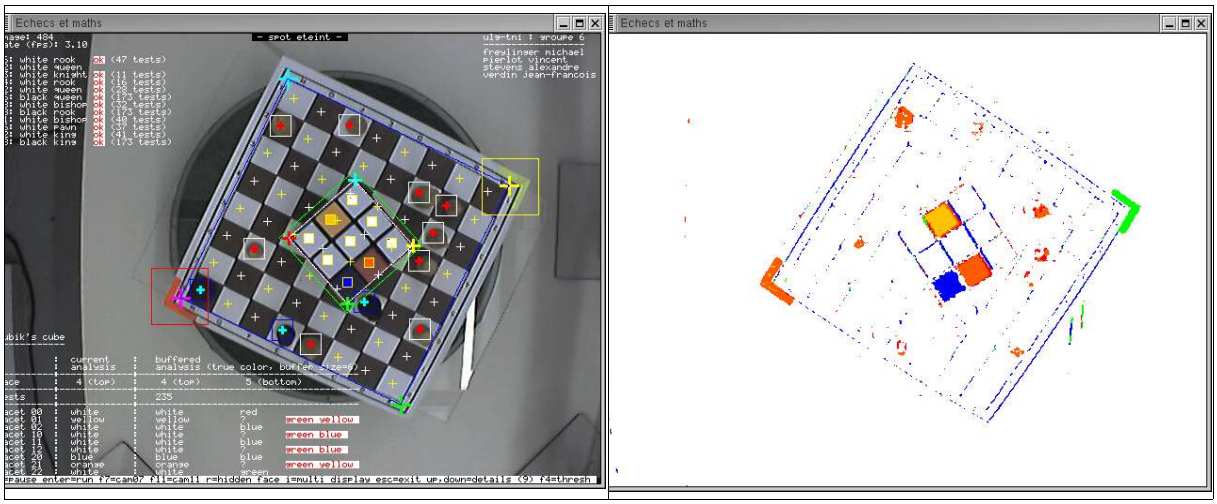


fig 6: A gauche : détection de l'échiquier, du cube, et des pièces sur la vue du dessus. A droite : test de notre seuillage de saturation pour détecter certaines facettes du cube.

4.4. Détection des pièces

Pour ce faire, nous opérons un balayage sur les cases. Premièrement, nous évitons le Rubik’s Cube dont nous connaissons les coordonnées. Nous acquérons une sous-image de la case courante, puis nous différencions la détection des pièces blanches de la détection des pièces noires.

Pour les pièces blanches, nous avons recours à un seuillage de saturation-teinte sur la sous-image. Ensuite, nous comptons le nombre de pixels segmentés. Si ce nombre est supérieur à un seuil, nous considérons qu'il y a une pièce blanche sur la case.

Pour les pièces noires, nous ne pouvons nous servir de la saturation ni de la teinte. Nous calculons la moyenne et l’écart type dans la sous-image. Nous conditionnons par rapport à ces deux valeurs pou déterminer la présence d’une pièce noire. Ce conditionnement est différent suivant que la case est noire ou blanche. Un seuillage en luminance est ajouté pour les cases noires.

Cependant, les variations de luminosité et le fait que les pièces peuvent changer de position rendent cette détection instantanée fragile. Une erreur peut survenir de manière ponctuelle. Aussi, nous avons mis en place un système de moyenne mobile. Nous stockons les résultats des dernières détections de pièces pour chaque case, et nous assurons la présence d’une pièce sur une case si la détection se réalise sur un majorité d'image.

4.5. Détermination des pièces

La détermination du type des pièces présentes sur l'échiquier est probablement l'une des tâches les plus complexes à implémenter. Pour les pièces blanches, une détermination à partir d'une caméra latérale nous a paru prometteuse. Si l'on peut assez facilement localiser les pièces par de simples seuillages dans cette vue, il est par contre très difficile de les faire correspondre à une case de l'échiquier.

La solution que nous avons adoptée est de faire correspondre à chaque pixel d'une image vue du dessus (par exemple les cases de l'échiquier), un autre pixel dans l'image vue de côté. Ceci est possible en utilisant un système de coordonnées homogènes et une matrice de transformation projective (3x3). Nous avons cependant choisi de travailler avec une matrice de dimension plus grande (3x4) qui permet également de transformer des points à trois coordonnées (avec la profondeur) (cf. [2] p. 448 pour plus de détails). Cela nous a été extrêmement utile pour l'analyse des facettes latérales du Rubik's Cube.

L'inconvénient majeur de cette technique est que la matrice n'est valable que pour une seule position relative des deux caméras. Une légère modification de cette position oblige de recalculer la matrice de transformation. Afin de rendre notre programme indépendant de cette matrice, nous avons créé des fichiers ASCII contenant les coefficients pour chaque caméra. Ces fichiers sont lus par notre programme lors de son exécution. Ces fichiers sont générés par un script MATLAB. Il est ainsi très facile et très rapide d'obtenir les nouvelles matrices au cas où les caméras changeraient de position et cela sans même recompiler le programme.

Pour la méthode de détermination des pièces, elle a été choisie de manière à être simple, rapide et efficace dans notre cas particulier. En effet, après de nombreuses réflexions, nous avons remarqué que les pièces avaient chacune un profil vertical propre. De plus, tous ces profils sont invariant par rotation d'axe vertical sauf pour le cavalier pour lequel nous déterminons deux profils (vue de face et vue de côté) . L'idée est de comparer le profil d'une pièce de l'image avec le profil de modèles chargés en mémoire (à partir de fichiers binaires).

Dans notre cas, le modèle d'une pièce est son effondrement sur l'axe vertical. Nous avons donc une représentation unidimensionnelle et unique de chaque pièce appelée sa signature. Pratiquement, ces signatures ont été créées par notre programme de manière interactive.

La détermination du type de chaque pièce blanche est obtenue en maximisant le coefficient de corrélation linéaire entre les signatures des modèles et la signature de l'objet à reconnaître. La longueur de la signature d'un objet étant dépendante de la distance à laquelle se trouve l'objet, il est indispensable d'adapter la taille des signatures avant de procéder à la corrélation. La signature de l'objet est obtenue après segmentation de l'objet qui se trouve dans une région d'intérêt calculée par transformation projective à partir de la vue du dessus. La segmentation consiste en un seuillage teinte-saturation complété par une identification de régions en connexité huit.

Pour les pièces noires, un procédé similaire n'a pas fourni de résultats probants. Nous avons alors décidé de tirer profit des contraintes imposées par le cahier des charges ; à savoir qu'il n'y aurait pas de pièces supplémentaires sur l'échiquier, et donc toujours trois pièces noires :

- la reine noire sur la case blanche (indispensable pour le tracking);
- le roi noir sur la case noire (indispensable pour le tracking);
- une autre pièce : la tour noire.

Il se trouve que le roi et la dame sont les pièces les plus grandes du jeu. A cause de la perspective, ces deux pièces on la plus grande aire vues du dessus. Il suffit donc de déterminer la pièce noire qui possède la surface la plus faible afin de savoir que ce n'est ni une dame ni un roi. Pour ces deux dernières, la couleur de la case indique le type de la pièce. La surface des pièces est calculée à partir d'un simple seuillage en luminance sur les cases d'intérêt.

4.6. Tracking

La seule contrainte imposée pour le tracking étant son affichage sur l'image d'une caméra qui nous est imposée, nous avons profité du fait que nous possédions toutes les informations nécessaires dans la vue du dessus . Nous obtenons alors les coordonnées des pièces à suivre par simple transformation projective. L'aire maximale des pièces suivies est obtenue grâce à la segmentation de l'objet effectuée préalablement pour la reconnaissance des pièces. L'angle de visibilité est calculé à partir d'un angle absolu pour lequel la pièce passe derrière le Rubik's Cube et un autre angle lorsque la pièce redevient visible. Nous savons calculer facilement si la pièce se trouve derrière le cube ou non grâce aux transformations projectives. Nous connaissons en effet les coordonnées des coins du cube et celles des pièces dans la vue du dessus.

4.7. Résolution du Rubik's Cube

Nous localisons premièrement le cube dans la vue du dessus comme expliqué dans l'objectif 1. Nous déterminons ensuite des régions d'intérêt dans les facettes pour y analyser la couleur. L'étape suivante est de trouver une manière de connaître la position des facettes du cube dans la vue de profil. Comme nous l'avons expliqué, nous utilisons une matrice de transformation projective de dimension 3x4. Ceci nous permet de transformer un point défini par ses trois coordonnées dans la vue du dessus en un autre point défini par deux coordonnées dans une vue latérale (fig 7).

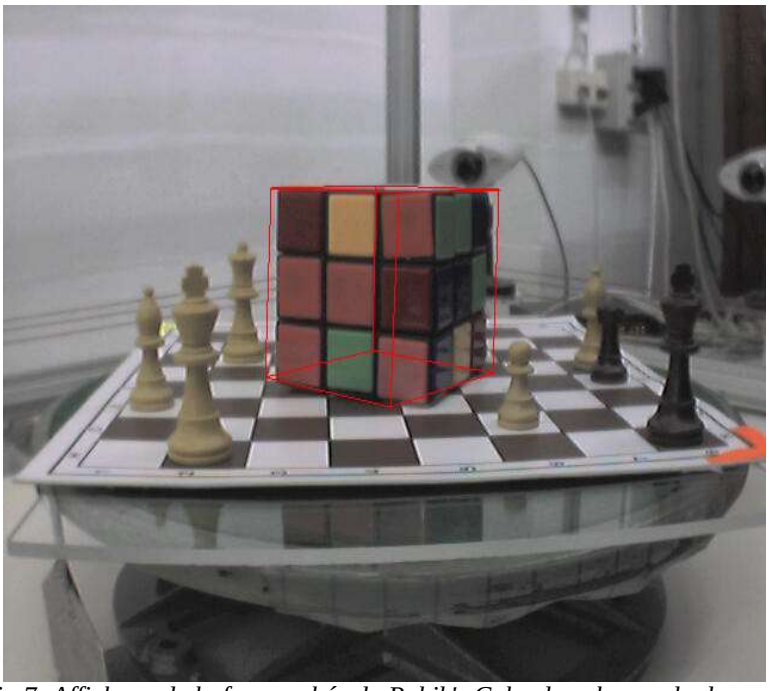


fig 7: Affichage de la face cachée du Rubik's Cube dans la vue du dessus.

Nous utilisons ensuite une fonction qui détermine la couleur d'une région d'intérêt et nous stockons les couleurs obtenues dans des tableaux.

La fonction se base sur un seuillage assez simple et robuste. Sachant que lui passer en argument une région d'intérêt d'une des 6 couleurs du cube, il n'y a que 6 cas à gérer.

On détecte d'abord le blanc par seuillage sur la saturation. Si ce dernier n'est pas concluant, il reste 5 cas. Il est aisé de déterminer si la couleur recherchée est le vert ou le bleu. Dans ces deux cas, les composantes respectives sont les composantes maximales. Le jaune est déterminé par seuillage, une fois les trois autres possibilités éliminées, en tenant du compte du fait qu'il y a plus de vert que de bleu dans cette couleur secondaire alors que dans le cas de l'orange et du rouge, ces valeurs sont plus ou moins équivalentes. La couleur orange est différentiée du rouge, lors qu'il ne reste que deux possibilités, par un seuillage sur la moyenne des trois composantes.

Dans chaque région d'intérêt (correspondant à une facette) , on détermine la couleur présente majoritairement pixel par pixel. Ceci limite les nuisances qu'apportent les pièces qui passent devant une facette.

A partir des couleurs de toutes les facettes visibles, le but maintenant est de déterminer la face cachée du Rubik's Cube.

Données :

- Le centre jaune du cube est toujours opposé au centre vert.
- Le centre blanc est toujours opposé au centre bleu.
- Le centre rouge est toujours opposé au centre orange.

Dès lors, il est aisé de déterminer le centre de la face cachée. Les coins de cette dernière sont trouvés facilement, car chaque coin possède des couleurs uniques. A partir des deux couleurs visibles composant le coin, on trouve la 3ème couleur. Déterminer les couleurs composants les « arrêtes cachées » est moins simple. En effet, en tout, il y a 12 facettes dont 8 sont visibles. On se retrouve donc avec 4 couleurs isolées dont il faut trouver la correspondance parmi quatre couples de couleurs représentant les arrêtes non visibles. On cherche ensuite à faire les associations. Dans le cas où plusieurs possibilités existent, on les affiche sur l'image en rouge sur fond blanc et on colorie la facette non déterminée en gris ().

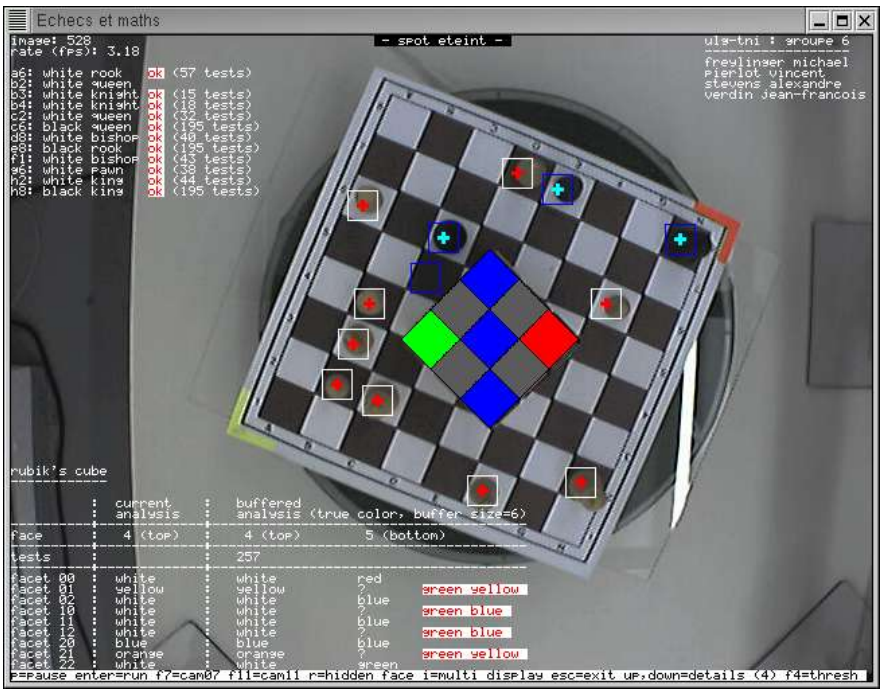


fig 8: Une fois les 5 faces déterminées, l'algorithme engendre la 6ème face. Le gris reflète une indétermination. Les différentes possibilités sont affichées en rouge sur fond blanc en surimpression de l'image. ***Affichage de la face cachée du Rubik's Cube dans la vue du dessus.***

4.8. Affichage

La librairie SDL nous permet d'afficher des images. Nous avons hésité à utiliser la librairie GTK. N'ayant aucune expérience en programmation d'interface graphique, nous avons préféré améliorer une librairie simple comme SDL. Nous avons ajouté de nombreuses fonctions d'affichage. Traçage de lignes, de rectangles, de croix, remplissage de formes, affichage de sous-images, affichage multiple (4 images en une), création d'une police personnalisée pour affichage de texte en surimpression, etc.

Nous allouons de la mémoire dynamiquement pour les images (et une seule fois). Nous utilisons les structures « image » de la librairie Xlim. De plus, nous avons créé une structure « img » personnalisée (voir « type.h »). « img » nous permet de manipuler des images tridimensionnelles.

5. Description des performances en fonction des critères de robustesse

Objectif 1 :

Toutes les pièces sont détectées et déterminées. La robustesse de ces deux opérations est obtenue par un système de vote majoritaire sur les buffers circulaires.

La détermination des pièces blanches par corrélation entre signatures donne des très bons résultats. Toutefois, il existe des limitations à notre programme. En effet, pour effectuer ces corrélations nous attendons le moment propice où le champ de vision dans la caméra de profil est dégagé : la pièce n'est masquée ni par le Rubik's Cube, ni par une autre pièce. Ceci peut entraîner des cas où certaines pièces ne sont pas déterminées.

Pour les pièces noires, étant donné que nous nous basons sur le fait que seules trois pièces sont présentes sur l'échiquier, nous ne gérons pas les autres cas, par exemple le remplacement de la tour par une autre pièce noire, où le cas très improbable où le programme ne détecterait pas trois pièces noires.

Objectif 2 :

Le tracking du fou blanc et du roi noir est réalisé ainsi que l'affichage de l'aire pour la pièce blanche. Nous n'avons pas calculé l'aire de la pièce noire parce que nous n'avons pas pu effectuer de segmentation des pièces noires dans la vue de profil. L'angle de rotation pendant lequel les pièces sont visibles est également calculé.

Objectif 3 :

Les couleurs de toutes les facettes sont déterminées correctement. Grâce à l'algorithme,

la face cachée peut être déterminée et affichée dans la vue du dessus. La détection du Rubik's Cube dans la vue du dessus pourrait cependant être améliorée, car la détection des facettes blanches n'y pas assez robuste et l'interpolation de la position du cube est imparfaite.

Critères de robustesse :

- Les pièces peuvent être déplacées avant le démarrage du programme. Notre programme, non content de remplir cet objectif, permet également un déplacement des pièces *pendant* le programme (avec un temps d'adaptation lié à la longueur au buffer circulaire).
- La position des caméras ne peut pas être modifiée avant le démarrage du programme, car les matrices de transformation utilisées nécessitent une calibration très précise. Cependant, comme expliqué plutôt, une mise à jour via un script Matlab peut être réalisée facilement.
- La position du cube ainsi que les couleurs des facettes peuvent varier avant le démarrage étant donné que le programme acquière les informations nécessaire à la réalisation de l'objectif 3 en temps réel. Ici encore, les facettes du cube pourraient changer pendant l'exécution du programme.
- L'intensité lumineuse n'est pas uniforme et il peut y avoir un brusque changement dû au spot. Tout ceci est géré par la robustesse de notre système de buffer circulaire. En effet, ces variations de luminance entraînant des problèmes de détection, il nous était impossible de nous baser sur des valeurs instantanées. De plus, notre programme est capable détecter le basculement du spot et adapter certains seuils en conséquence.
- Lors du fonctionnement, le cube occulte les pièces à suivre une partie du temps. Rappelons encore ici l'utilité du buffer circulaire.

6. Références

1. Marc Van Droogenbroek. *Traitement numérique des images.* Centrale des cours de l’AEES, Université de Liège, septembre 2003.
2. M. Sonka, V. Hlavac, R. Boyle. *Image Processing, Analysis, and Machine Vision.* PWS Publishing, 1999.
3. L. G. Shapiro, G. C. Stockman. *Computer Vision.* Prentice Hall, 2001.