Michel Deriaz
Frédéric Ehrler

08.07.2002

Rapport de Cybernétique

Concours Mindstorms

1. Introduction :

Que c'est capricieux, un robot ! Initialement on réfléchi à toutes les astuces possibles pour gagner quelques centièmes de secondes par-ci et par-là. Puis, au fur et à mesure qu'on expérimente et qu'on se rend compte de nombreuses imprécisions de la caméra ou du robot, moins on devient exigeant et l'on se dit : « Faisons quelque chose qui marche pas trop mal, et puis on verra après pour les optimisations. » Et finalement : « Pourvu que le robot ne tombe pas de la table… ».

Et bien oui, c'est de loin pas une tâche élémentaire que d'aller chercher des cylindres colorés avec des pièces Lego !

Nous avons choisis le langage Java pour réaliser notre travail. Pour le programme principal, ce fut certainement un bon choix. En effet, la clarté et la puissance de ce langage ont permis de développer rapidement le simulateur ainsi que le programme d'IA, chargé de trouver le chemin du robot. Pour la partie caméra, utiliser Java était plus audacieux. En effet, les nombreuses librairies disponibles en C ne le sont pas en Java… Ce n'était donc pas forcément le plus efficace, mais l'intérêt de faire soi-même le code compense largement ce désagrément. Par contre, utiliser Java pour le robot lui-même, c'est un peu jouer aux aventuriers. Nous avons utilisé LeJos, la version Java pour le RCX. Malheureusement ce langage est encore jeune et de loin pas stable. Le principal problème que nous avons rencontré est que les moteurs adoptent un comportement aléatoire dès que la taille du code dépasse 3.8 ko, alors que nous somme censé pouvoir disposer de 12 ko. Ensuite certaines bizarreries n'ont toujours pas trouvé de réponse. Par exemple, le fait d'avoir trois moteurs connectés sur le RCX n'était pas non plus très apprécié (moteurs tournant extrêmement lentement, et de sens contraire).

Voyons maintenant un peu plus précisément les différentes parties du projet.

2. Le Robot :

La spécification initiale était très optimiste. Le robot devait pouvoir :

- Trouver et capturer les cylindres sans la caméra, lorsque le robot était suffisamment près, en émettant un signal infrarouge par le RCX et en écoutant la réponse avec un capteur lumineux.
- Déposer les cylindres de manière précise tout seul, en se servant d'un capteur lumineux orienté vers le sol, pouvant ainsi « voir » les goals.
- Prendre plusieurs cylindres à la fois.

Le dernier point a été rendu impossible par une clause du règlement. Par contre les deux autres clauses ont été rendues impossibles du fait que le code sur le RCX était tant limité !

Nous avons finalement construit et fait une batterie de tests sur deux robots différents, afin de savoir lequel sera l'heureux élu pour le concours. Par contre, nous avons cherché pour les deux à les faire le plus mince que possible (nous avons même omis les pneus), afin de naviguer plus facilement, sans risquer de heurter un cylindre ou le bord d'une porte.

Robot 1 : Le DiffBot

Diff pour différentiel, car le robot en utilise un pour soulever les cylindres et Bot comme la fin de « robot ». La difficulté était de trouver comment soulever les cylindres en utilisant qu'un seul moteur, vu que notre slogan « robot le plus petit que possible » était contradictoire avec l'utilisation d'un deuxième RCX. La solution a donc été de se servir d'une pince pour tenir les cylindres et d'une roue pour les monter et les descendre. Cette roue est en fait connectée aux roues faisant avancer le robot. Ainsi, si le robot avance, la roue tourne en arrière et donc le cylindre monte, si le robot recule, la roue tourne en avant et le cylindre descend, et finalement si le robot pivote (donc une roue tournant en avant, l'autre en arrière), le différentiel annule le mouvement et la roue déplaçant le cylindre ne bouge pas.

Robot 2 : Le MiniBot

Les mauvaises langues l'appelleront le « RidiculeBot », mais c'est le fruit d'une longue étude et de nombreuses heures de réflexion ! L'inconvénient est évidemment qu'il ne soulève pas les cylindres mais qu'il se contente de les pousser. Ce qui nous aura certainement valu une pénalité durant le concours. De plus, un petit inconvénient mineur vient s'ajouter : il faut deux sets de calibrations, car le robot n'avance ou ne tourne pas de la même façon avec ou sans cylindre. Mais il possède par contre un avantage de taille : il est petit, très petit. Et donc il navigue avec plus de facilité.

Le choix :

Compte tenu des avantages et des inconvénients de chacun de ces robots, nous avons finalement opté pour MiniBot, en espérant que la pénalité qui nos sera infligée (vu qu'il faudrait soulever les cylindres) sera compensée par les avantages apportés.

3. Le code du Robot :

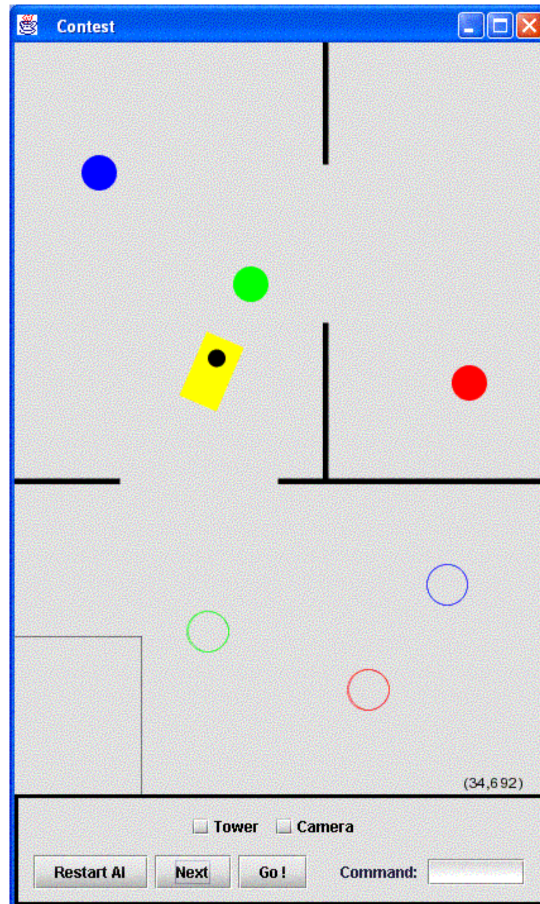
Après avoir chargé le firmware sur le robot, il nous reste théoriquement 12 ko, mais, comme déjà mentionné dans l'introduction, le robot adopte un comportement bizarre dès que la taille du code approche 3.8 ko. Nous avons essayé avec deux versions différentes de Java, la 1.3.0 et la 1.4.0 ainsi que l'utilisation de l'option "target" dans le compilateur "lejosc", mais les limitations restèrent similaires. La modification de la structure du code (comme faire plus de classes avec moins de code, ou modifier les structures de données) permettait parfois de repousser très légèrement les limites, mais sans nous donner réelle satisfaction. Le code dans le robot est donc minimal, toute "l'intelligence" se trouve donc sur le PC.

Nous avons constaté que le temps de transmission entre la tour et le robot n'était pas négligeable. Nous avons donc écrit un programme permettant de mesurer ces temps et nous avons obtenu:

- Communication de 50 short (2 octets) PC -> RCX: 32.26 sec
- Communication de 100 short (2 octets) PC -> RCX: 63.55 sec
- Communication de 50 int (4 octets) PC -> RCX: 63.15 sec
- Communication de 100 int (4 octets) PC -> RCX: 126.72 sec

C'est pourquoi nous avons cherché à réduire au maximum les communications. Toutes les instructions sont transmises sous forme d'une paire de short: Le premier donne la rotation à effectuer, le second donne le distance à parcourir. Les commandes spéciales, comme ouvrir la pince ou tester la calibration, sont donnés par des codes supérieurs à 500 (comme premier short). Ensuite, une fois terminé, le RCX envoie lui aussi un short pour dire qu'il est à nouveau disposé à recevoir une nouvelle instruction.

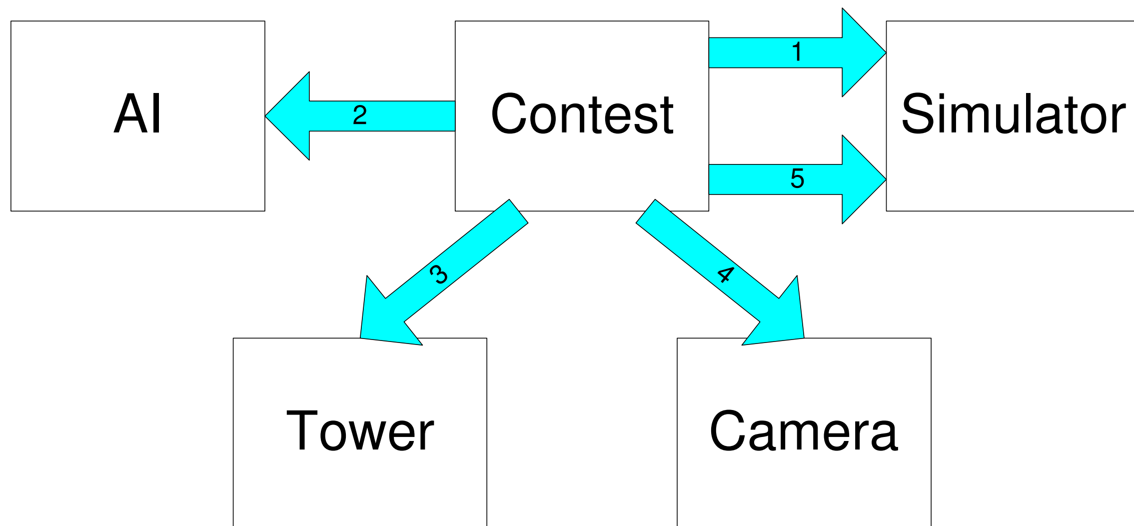
4. Le programme :



Le programme principal affiche le simulateur ainsi que le tableau de commandes. Le mode d'emploi peut se résumer en quelques points:

- Cochez "Tower" si les instructions doivent être également transmises à la tour (donc au robot), sinon elle seront transmises uniquement au simulateur.
- Cochez "Camera" si le simulateur doit être mis à jour en fonction de la caméra, sinon il sera mis à jour en fonction de la commande effectuée.
- Déplacez les différents objets de la scène en utilisant le menu contextuel.
- Pressez "Restart AI" pour initialiser le programme calculant les déplacements du robot.
- Pressez "Next" pour envoyer commande par commande les déplacements calculés.
- Pressez "Go !" pour enchaîner les déplacements.
- Dans la zone "Command: " taper "x y" avec x l'angle en degré, et y le déplacement en mm, ou une commande spéciale (les codes se trouvent dans le fichier GC.java).

Dans les conditions du concours, avec les cases "Tower" et "Camera" cochées, les classes principales sont sollicitées de la manière suivante:



1) La classe principale, Contest, demande les positions de différents objets du terrain au simulateur.
2) Contest demande la prochaine commande à exécuter à la classe AI, en lui transmettant la situation actuelle sur le terrain.
3) Contest transmet la commande à la classe "Tower", donc indirectement au robot, puis attend que ce dernier ait terminé.
4) Contest demande les positions des différents objets de la scène à la caméra.
5) Les informations fournies par la caméra sont transmises au simulateur afin qu'il puisse se mettre à jour. Et on recommence.

L'avantage des cases à cocher est conséquent. Ainsi tout le programme d'AI (responsable de calculer les déplacements) a pu être développé uniquement avec le simulateur (cases "Tower" et "Camera" décochées).

5. La classe AI :

La classe AI (Artificiel Intelligence) est la partie "intelligente" du programme. Elle est composée que d'une seule méthode publique, "public String next(Positions positions)". On lui donne en paramètre l'ensemble des positions des différents objets du terrain (ce qu'on obtient par le simulateur) et elle nous retourne sous forme d'un String la commande à envoyer au robot.

Son fonctionnement est très simple. Pour aller d'un point A à un point B, il y a un certain nombre de passages obligés. Par exemple, pour s'assurer que le robot passe de manière perpendiculaire une porte, il est forcé de passer par un point se trouvant avant la porte ainsi qu'un point se trouvant après. Le programme détermine alors le premier passage obligé, sous forme du point associé à une tolérance maximale qu'on peut se permettre. Et tant que le robot ne sera pas dans la zone de tolérance, il aura pour mission de rejoindre ce même passage obligé.

6. L'analyse d'image :

Le processus de mise en place :

Les objectifs à résoudre grâce a l'analyse d'image étaient les suivant :

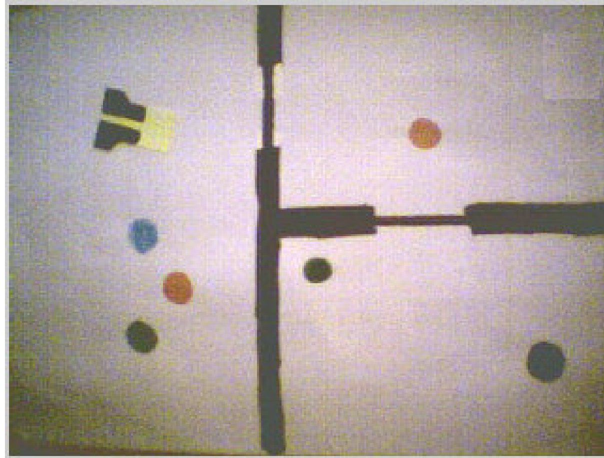
- Repérer les murs et les portes.
- Repérer les positions des cylindres.
- Repérer la position du robot et son angle.

En effet le robot que nous avons conçu, ne possédant aucun capteur pour se repérer, doit faire appel à la camera pour faire l'acquisition de son environnement et connaître sa position, afin de pouvoir planifier les actions à accomplir.
Le robot n'a pas la possibilité d'une autonomie total dans le déplacement, car malheureusement malgré une bonne calibration, il va petit a petit accumuler des erreur de déplacement et n'aura au final plus aucune précision pour les tache qu'il a accomplir, c'est pourquoi il faut une camera pour le guider.

L'approche choisie a été de faire un programme d'analyse d'image entièrement en java. De l'acquisition d'image par la camera en utilisant JMF jusqu'au prétraitement d'image en utilisant JAI. De plus une interface imposante a également été programmée, elle nous permet de pouvoir changer rapidement les paramètres d'analyse d'image et ainsi pouvoir s'adapter rapidement a de nouvel environnement lumineux, ou encore a des changements de couleur de cylindre. Dans la même optique, une grande quantité d'image traitée différemment sont accessible, ainsi, si un élément n'est pas visible sur une image, on a toujours la possibilité de le rechercher sur une autre image.

La partie de prétraitement d'image :

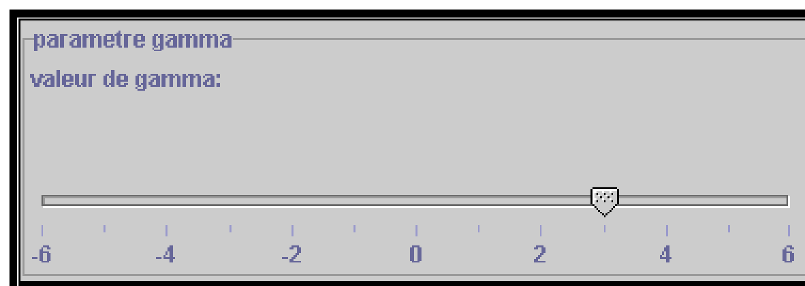
Cette partie a été réalisée en partie avec JAI. JAI est une librairie java (Java Advanced Imaging) qui permet un traitement plus aisé des images, par exemple pour les opérations de convolution, ou de lookup table.



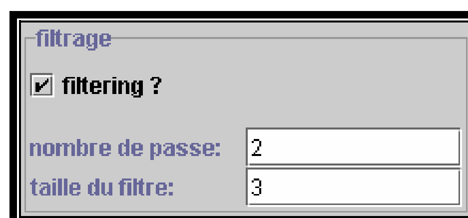
Comme on le voit ici l'image des base est de relativement mauvaise qualité, il est donc très important d'effectuer un prétraitement afin d'avoir une image exploitable.

Les différents prétraitements communs à toutes les images :

- Une correction gamma, qui diminue fortement les zones d'ombre sans toute fois trop dégrader les zones illuminées. Cette valeur de gamma est réglable à travers l'interface.



- L'application d'un filtre moyenneur, il permettra d'avoir des zones d'image bien plus homogène et ainsi éviter les amas de pixels épars qui bruitent de façon très désagréable l'image lors de seuillage ou d'autre traitement. On peut décider de la taille du filtre ainsi que du nombre de passe à faire sur l'image.

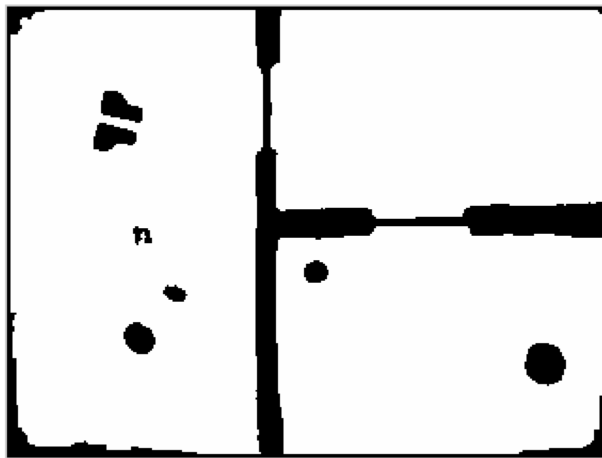


Une fois ces prétraitements effectués, on peut travailler à partir d’une image de base de meilleure qualité.

Les images accessibles pour la détection des différents objets :

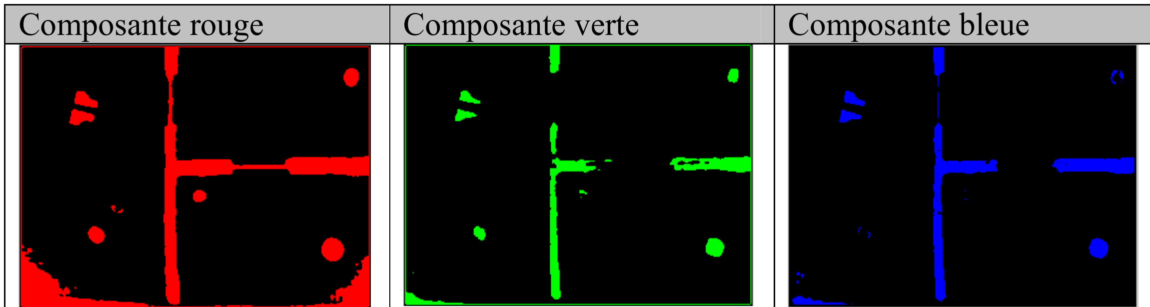
Comme je l'ai dit plus haut, l'approche a été d'avoir plusieurs images disponibles afin de pouvoir sélectionner les meilleurs pour les recherches. Parmi toutes ces images on a entre autre.

- L'image noir et blanc : Pour obtenir cette image, on somme les trois composante RGB et on divise par 3. Le résultat ainsi calculé sera copié sur toutes les bandes de l'image, on aura ainsi une image noir et blanc. On effectue ensuite une binarisation, qui est en faite équivalent a un seuillage binaire. Le seuil sur lequel s'effectue la binarisation peut bien entendu être changé dynamiquement à travers l'interface.
  Cette image est très pratique pour la détection des murs

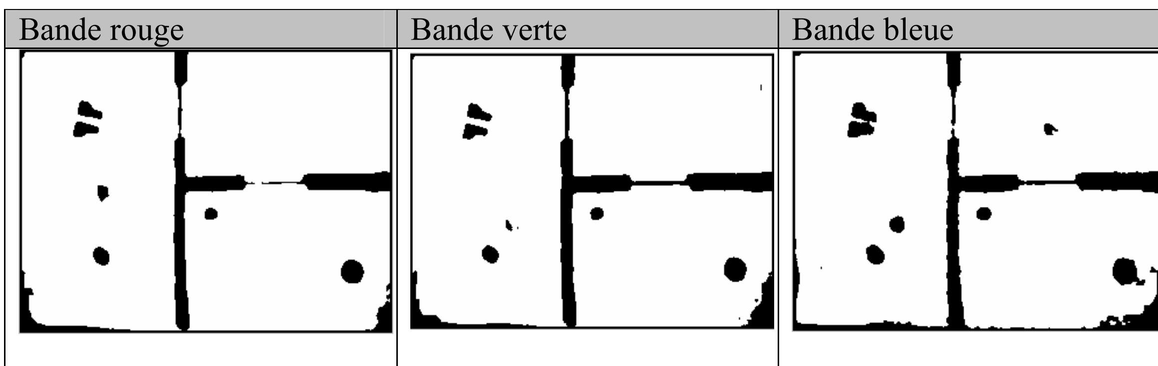


- L'image seuillée en couleurs : Cette image est calculée grâce a l'utilisation d'une lookup table, toute les bande sont soit misent a 0 soit a 255 selon si la valeurs du pixel est en dessous ou en dessus du seuil fixé.
  Comme on le voit cette image est très pratique, pour détecter le robot et certain cylindre

- Reste encore les images des composantes des couleurs. Pour obtenir ces images, le procédé a été de sélectionner les valeurs, pour lesquels, la couleur choisie est au dessus d'un certain seuil, alors que les autres couleurs sont elles en dessous d'un autre seuil, fixé également a travers l'interface.
  Cette image permet la détection des cylindres, si celle-ci est impossible sur l'image seuillée en couleurs



- Il reste de plus quelque autre image accessible, parmi celle-ci les image en bande, le principe est pratiquement le même que les images de sélection des couleurs, a la différence près, qu'il n'y a la pas de contraintes sur les autre couleurs, seulement sur celle qui est sélectionné. De plus on aura en résultat une image composée d'une seule bande, ce qui nous donne à l'écran une image noir et blanc. La différence du nombre de bande n'est pas primordiale, mais elle allège les traitements et la taille de l'image.



- Finalement on a une image qui conserve seulement les pixel dont la différence de couleurs entre les trois composante est supérieur a une certaine valeurs, cette image fais ressortir spécialement bien toutes les composante rouge et peut donc en conséquent être utilisé pour la détection du cylindre rouge.

La détection des différents objets :

Bien sure tout ces traitements et différentes images doivent nous amener finalement à la détection des objets sur le terrain, on aimerais détecter :

- Les murs
- Les portes
- Le robot ainsi que son orientation
- Les positions des différents cylindres

La détection des murs et des portes:

Pour la détection des murs, on part de l'image noir et blanc, on va simplement effectuer un parcours de l'image a partir d'une position décalée en X et en Y a partir du centre, on parcours l'image horizontalement et dès que l'on trouve un pixel noirs cela signifie que l'on touche le mur vertical, on connaît donc sa position. Pour le mur horizontal, pas de grande différence, il faut juste savoir si le mur se trouve a gauche ou a droite, afin de savoir de quel coté le rechercher.
Pour les portes, il a suffit de remarquer que les ombres étaient plus fine la ou se trouve les portes, on va donc chercher sur toutes la longueur des murs la position là ou le mur est le plus fin cela nous indiquera la position des portes.

La détection des cylindres :

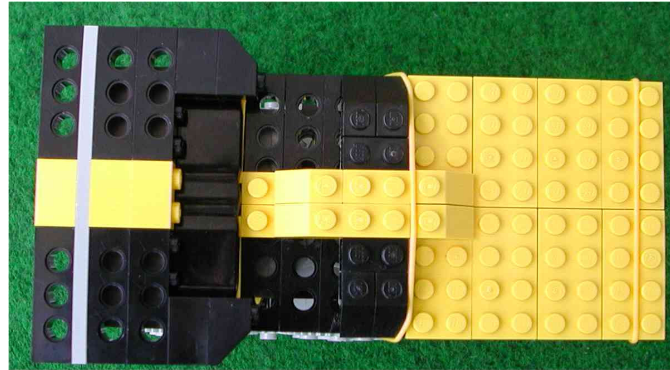
L'image choisie pour le repérage des cylindres dépend beaucoup de la qualité de la lumière ainsi que la couleur du cylindre.
Pour le cylindre rouge, l'image seuillée en couleur marche normalement relativement bien, pour le trouver on essaye de faire matcher un pattern d'un carré rouge sur l'image, On va essayer de trouver la correspondance, mais seulement dans la partie de l'image qui est susceptible de les accueillir, c'est-à-dire a droite ou a gauche du mur vertical et non pas sur tout le terrain. Cela évite également de confondre les cibles et les cylindres.
Pour réduire d'avantage le nombre de comparaison, on va essayer de faire matcher le pattern seulement quand l'on tombe sur un pixel rouge. De plus le pattern doit matcher et es pixel alentour au pattern ne doivent pas être rouge.

Si on essaye de faire matcher un pattern 15X15 sur toute l'image, on aura 320X240*15X15 = 20 mio de comparaison, par contre si on ne le fait que sur les pixel rouge = 1000 on aura 1000*15X15 comparaison = 2 mio de comparaison donc un gain de facteur 10.

Lorsque l'on a la meilleure correspondance, on estime que c'est position du cylindre.

Pour les autre cylindre on utilise le même système, mais pas forcement sur la même image, on va faire typiquement une recherche sur les image des composantes.

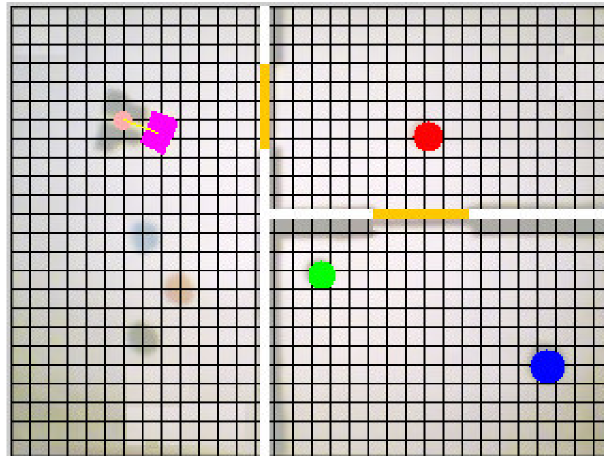
La détection du robot :
La détection du robot ainsi que son angle est sûrement la partie la plus ardue. Le repérage s'effectue à partir du toit du robot, il a fallut donc choisir un pattern facilement reconnaissable, qui de plus, nous permet de déterminer la position du robot.
Après plusieurs essais le toit retenu a été le suivant :



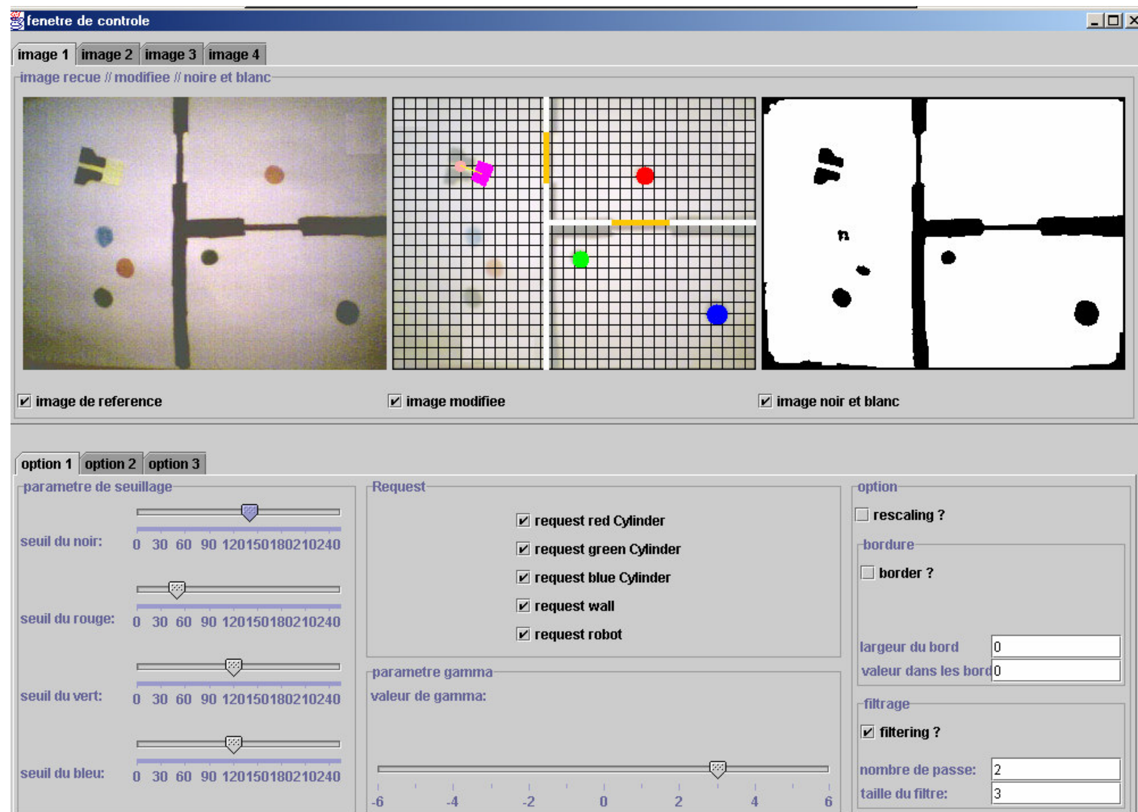
On va effectuer les recherches sur l'image en couleur seuillée, qui nous permet une détection du jaune plus ou moins bonne selon la qualité de l'éclairage. Le choix de la couleurs jaune a été motivé par trois raisons, premièrement, c'était la seul couleur présente en suffisante quantité pour faire un toit (a part le noirs) dans la boite lego Mindstorm. Deuxièmement le jaune n'est pas utilisé par un des éléments de l'environnement. Finalement le jaune est composé d'une part de rouge et de vert égale.
On a également l'avant du robot qui pourra être repéré en matchant un pattern de couleur jaune sur tout le terrain, de la même manière que cela été réalisé avec les cylindre. Une fois le robot situé, on va chercher les points extrêmes de la surface jaune, ce qui nous permettra de calculer le centre de gravité. L'étape suivante est de détecter la partie arrière, on match a nouveau un pattern spécial, en faite on cherche un endroit ou la somme des pixel noir et jaune est le plus proche possible. Grâce à ces deux points, on peut calculer l'angle du robot.

Résultat de la recherche :

Si la recherche a bien fonctionné on se retrouve dans la situation ci-dessous, tous les éléments sont correctement identifiés et le robot pourra atteindre son but sans problème.



Vue de l'interface :



7. Le programme TestIR :

Ce petit utilitaire permet de tester la qualité de la réception Infrarouge. Il permet de s'assurer que la tour est bien placée et que le robot ne risque pas de louper une commande. Son code est extrêmement simple. Le code tournant sur le PC émet un signal toute les secondes environ. Dès que le RCX reçoit le signal correctement, il émet un bip. Ainsi, la cadence des bips est corrélée avec la qualité de la réception. Nous avons écrit se programme afin de déterminer où nous allions placer notre émetteur.

8. Le programme de calibration :

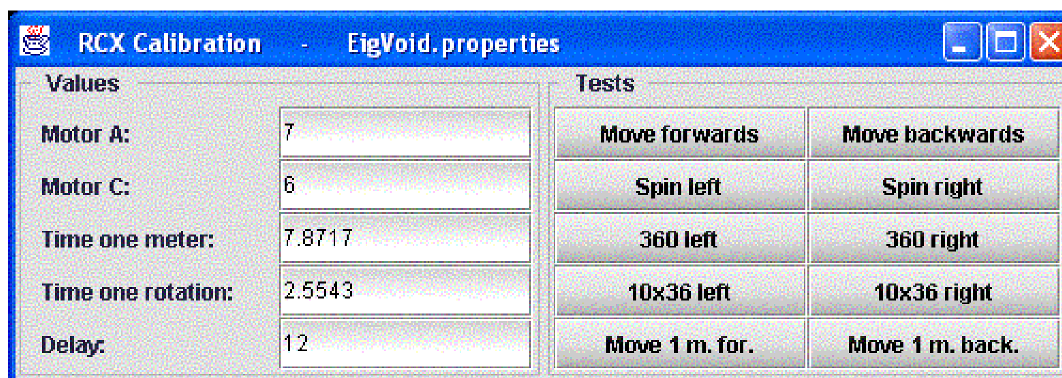
Notre set de calibration est constitué de 5 attributs:

- Motor A: Puissance du moteur A. Les moteurs n'ont pas exactement la même puissance. On s'en aperçoit très vite en demandant au robot d'aller tout droit... La puissance est un nombre entre 1 et 7; on choisira donc 7 pour le moteur le moins puissant, et on diminuera la valeur du moteur le plus puissant jusqu'à ce que le robot se déplace de manière sobre...
- Motor C: Puissance du moteur C. (le moteur A entraîne les roues gauches, le moteur C les roues droites et le moteur B la pince).
- Time one meter: Temps mis pas le robot pour parcourir 1 mètre.
- Time one rotation: Temps mis par le robot pour tourner de 360°
- Delay: Lors de chaque mise en mouvement, les moteurs mettent un certain temps pour réagir et se mettre à tourner. De plus, et spécialement avec nos roues dentées, il y a un certain glissement supplémentaire lors de l'accélération. Et finalement, lorsque les moteurs s'arrêtent, l'inertie acquise fait que le robot ne s'arrête pas instantanément. Ces différents effets cumulés font que, pour chaque mouvement, le robot a besoins d'un petit temps de plus pour accomplir sa tâche. Comme se temps est fixe pour tout mouvement (on aura le même dérapage initial si on entame un 360° que si on tourne seulement de 20°...), on peut donc l'ajouter systématiquement à chaque commande. C'est ce que fait notre paramètre Delay, exprimé en millisecondes.

Pour tester la calibration, il est possible de le faire avec le programme principal:

1. Tapez la commande "0 1000" et s'assurer que le robot avance tout droit, et de exactement 1 mètre.
2. Tapez les commandes "360" puis "-360" et vérifiez que le robot effectue un tour complet de manière précise.
3. Tapez les commandes spéciales "601" et "602", qui permettent de faire tourner le robot de 10 * 36° à gauche et à droite, et s'assurer que là aussi le robot fasse bien un tour complet.

Comme le robot ne réagit pas de la même manière avec ou sans cylindre, il faut en réalité répéter ce test deux fois. Le robot possède d'ailleurs deux sets de calibration, et passe automatiquement de l'un à l'autre lorsqu'il prend ou dépose un cylindre. Cependant, ces trois tests mettent en jeu l'ensemble des paramètres de calibrations; ils sont donc suffisants pour **vérifier** que le robot est bien calibré. Par contre, pour **déterminer** ces paramètres, il est utile d'avoir recours à un programme particulier. En effet, car trouver les paramètres par tâtonnement est ambitieux et surtout très long, car il faut à chaque fois recharger le code sur le RCX.



Ce programme permet de rentrer les paramètres, puis lorsque l'utilisateur sélectionne un test, le programme transmet dynamiquement les paramètres ainsi que le code du test demandé. Le robot commencera donc par se calibrer puis effectuera le test choisis.

Lorsque l'utilisateur lance le programme, il est invité à spécifier sur la ligne de commande un nom de fichier (sans extension) dans lequel il désire enregistrer ses paramètres. Ainsi, lors de la prochaine session de calibration, le programme récupérera automatiquement les données précédentes. Pour le concours nous avons créé 4 fichiers:

- EigFull.properties: Attributs pour l'EIG, lorsque le robot transporte un cylindre.
- EigVoid.properties: Attributs pour l'EIG, lorsque le robot ne transporte pas de cylindre.
- HomeFull.properties: Attributs pour la maison, avec cylindre.
- HomeVoid.properties: Attributs pour la maison, sans cylindre.

9. Conclusion :

En résumé, on peut dire que ce concours fût une expérience inoubliable. Le fait de travailler avec du matériel si capricieux, avec des sources d'imprécisions si nombreuses, nous a appris un nouveau style de programmation : la programmation tolérante. C'est-à-dire faire des programmes très facilement adaptables, et surtout qui ne plantent pas lors d'une erreur, mais qui au contraire ont la faculté de « retomber sur leurs pattes », lorsque des incohérences apparaissent.

En faisant le bilan du concours, nous nous sommes penché sur les améliorations que l'on pourrait envisager. Le principal point que nous avons retenu était une collaboration plus importante entre les classes de la caméra et celles du programme principal. L'objectif initial était de respecter au maximum un des concepts de la programmation objets, à savoir limiter le plus possible les dépendances inter-objets). Le programme principal connaît en fait qu'une seule méthode de la partie caméra. C'est la méthode getPositions() qui retourne toutes les positions des différents objets présents sur le terrain. L'amélioration consisterait à transmettre à la partie caméra la position approximative du robot après un déplacement (en fonction de la commande qui lui est transmise). On évite ainsi d'une part que la caméra perde du temps à chercher le robot sur toute la scène, mais surtout une incohérence serait détectée bien plus vite (si la caméra trouve le robot dans une zone inattendue).

Cette expérience nous sera utile pour le concours de l'année prochaine…