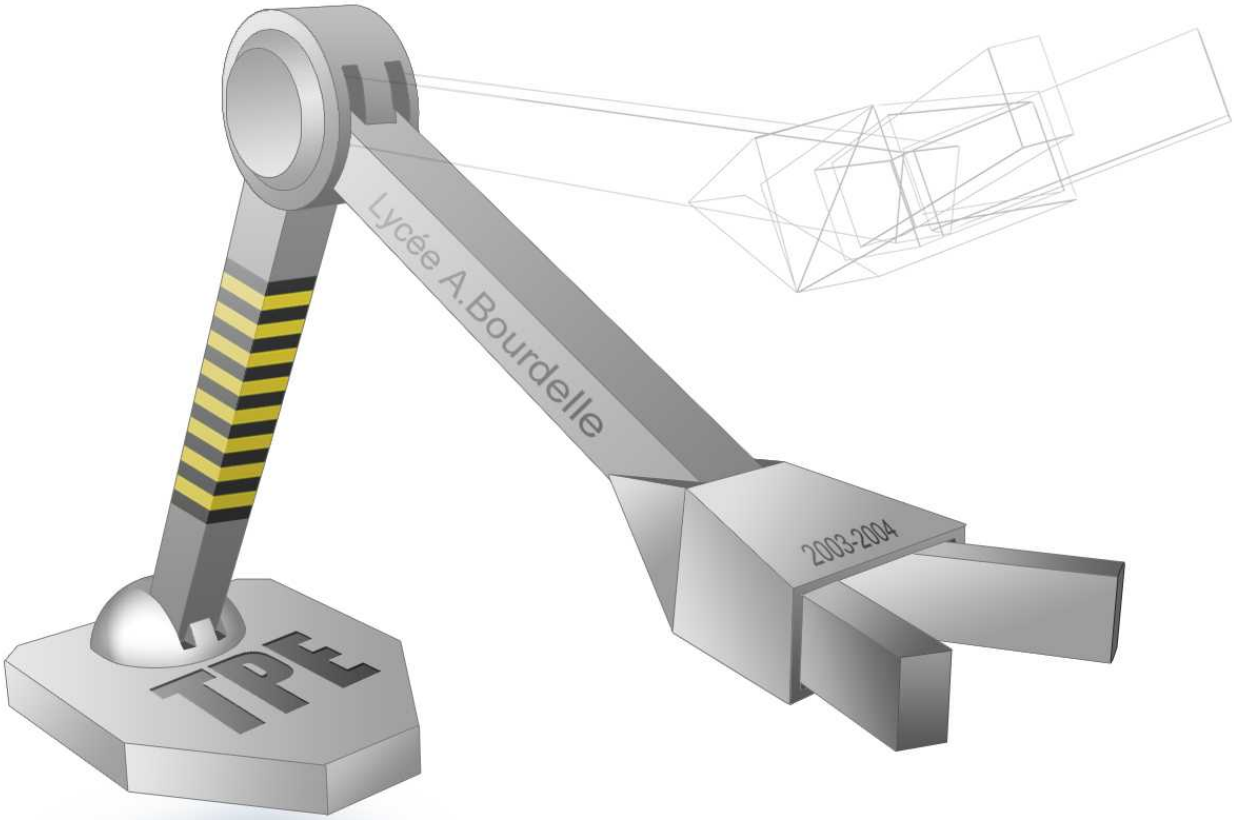
Fau Alexandre, Guillame-Bert Mathieu, Zalewski Frédéric

TPE

Projet de Travaux Pratiques Encadrés

Terminale S : 2003-2004

LYCEE Antoine BOURDELLE - Montauban



Sujet du TPE : Comment contrôler d'un bras mécanique articulé, à partir d'acquisitions numériques réalisées au moyen d'une caméra et d'un algorithme chromatique ?

Mars 2004

Sommaire

1. Introduction et présentation

Aujourd'hui, et de plus en plus dans le futur, le robot sera amené à remplacer l'homme dans certaines taches. Mais celles-ci se bornent encore à des actions simples. De plus si un événement imprévu vient perturber le système, il faut que l'homme intervienne. Des travaux sont encore nécessaires dans les domaines de l'automatique, de l'intelligence artificielle, … pour assurer des progrès

Après une analyse de concept et la réalisation de la maquette d'un bras mécanique articulé et d'une électronique de commande associée, le but de ce TPE est d'étudier comment donner au robot des "yeux" pour voir l'environnement qui l'entoure, un cerveau pour qu'il le comprenne et des bras pour qu'il y agisse. Bien sur la vue chez les robots peut également servir à d'autres taches comme par exemple la surveillance.

Evidemment, notre TPE n'a pas pour but d'inventer un système révolutionnaire, mais tout simplement de comprendre les futurs systèmes munis d'un bras mécanique. La maquette que nous avons réalisée donne des résultats intéressants et encourageants. Des améliorations de commande seront cependant nécessaires pour certains cas où il ne fonctionne pas encore.

2. Partie informatique (logicielle)

La partie informatique ce présente sous forme d'un logiciel.

De manière à ce que l'utilisateur puisse facilement utiliser le robot, des scripts ont été développés pour prédéfinir les actions qu'il attend du robot.

Exemple de script simple : « Prendre un objet »

```
call Initialisation
call Analyse
call CalculeIdealObj
call GoToIdeal
call FermerPince
```

2.1. Analyse de l'image provenant de la caméra

2.1.1. Introduction

Le but de notre TPE étant de contrôler un bras mécanique (ou un robot), à partir d'une image numérique en temps réel, nous avons dû réfléchir à un algorithme et à une technique permettant de le contrôler. Il s'est donc agit d'analyser l'image et d'en retenir les informations les plus importantes afin de les traiter dans la perspective du contrôle. Partant de ce principe nous nous sommes penché sur la détection des articulations du robot et de l'objet. Nous avons obtenu alors à partir d'une image contenant le robot et l'objet, des coordonnées plus facilement exploitables par la suite.

(a) (b)

Figure 1 – Bras articulé équipé de taches colorées

L'analyse de l'image (a) ci-dessus devrait alors par l'intermédiaire d'un algorithme nous renvoyer les coordonnées des points (croix) tels qu'il sont présentés (b).

Nous avons donc pensé à un moyen différent et plus facile pour pouvoir reconnaître les axes du bras ainsi que l'objet : il s'agit de mettre des taches de couleur basique (rouge vert et bleu) se détachant du reste de l'image. Et c'est la détection de ces « taches » que nous allons étudier maintenant.

2.1.2. Algorithme chromatique

2.1.2.1. Le rouge, le vert et le bleu numérique

2.1.2.1.1. Introduction

Une image numérique est constituée d'une multitude de points de couleur, la taille de cette image étant définie par sa résolution (nombre de pixel de large sur nombre de pixel de haut). Chacun de ces points comporte, en bmp (format le plus basique pour stocker des images non compressées), trois informations qui nous permettent de lui attribuer une couleur : c'est le RGB, **R**ed **G**reen **B**leu (ou RVB en français). Le RGB permet à l'aide d'une indication de rouge, de vert et de bleu de savoir si le pixel est de telle ou telle couleur. On attribue 1octet pour chacune des 3 couleurs ce qui nous donne 255 possibilités de variantes sur chaque couleur et donc 24 bits par pixel, ce qui porte à 16 millions le nombre de couleurs possibles. Hors le problème est que pour considérer un point comme rouge il ne faut pas retenir uniquement le bit concernant le rouge du RGB , car le blanc (R=255 , G=255 , B=255) serait considéré comme plus rouge que le rouge lui même (R=128 , G=0, V=0). Il est donc indispensable de définir le rouge, le vert et le bleu numériquement.

Pour définir les couleurs numériques nous avons utilisé un logiciel permettant de voir la palette entière de couleur. Il existe plusieurs types de palettes de couleurs : ici nous utiliserons la palette par défaut de Photoshop.

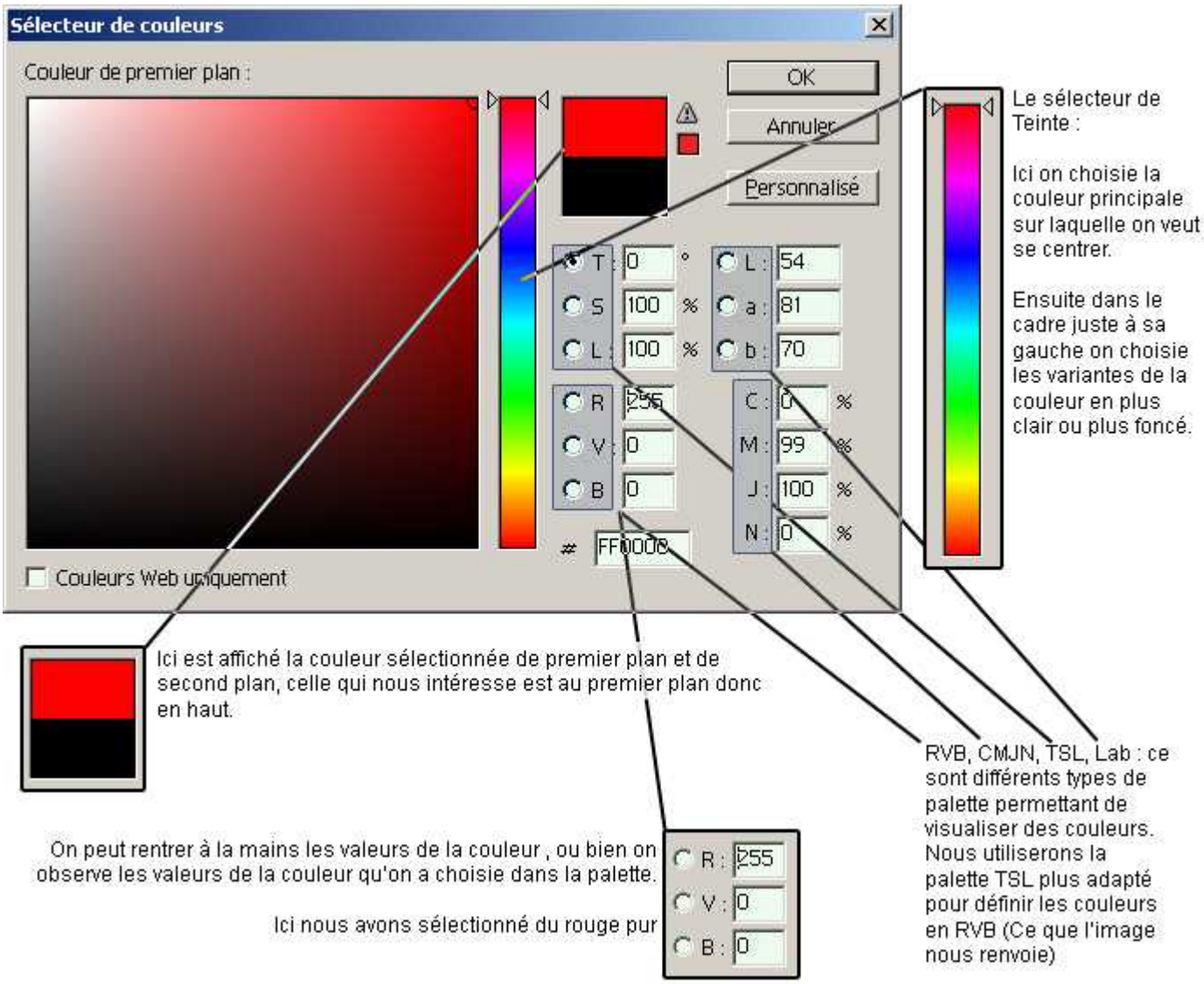


Figure 2 – Sélecteur de couleurs Photoshop

2.1.2.1.2. Le rouge

L'utilisation du 'Sélecteur de couleur' permet de savoir à quoi correspond le rouge numérique dans la palette RGB. Par exemple sur l'image précédente où nous sommes en réglage TSL (Teinte, Saturation, Luminosité), nous pouvons voir que tout le cadran central représente du rouge, en fait le même rouge avec la même teinte.

Donc pour une même teinte, quelque soit la saturation et la luminosité, on reste dans le domaine de la couleur. A partir de cette constatation, il faut définir les limites correspondantes au rouge, pour ne pas passer dans le marron (Rouge très foncé), ni au gris … On retient donc une zone dans ce cadre pour laquelle la couleur sera toujours définie comme équivalente à une même teinte. Un cercle avec pour centre le sommet haut droit du cadre et de rayon 2/3 du cadre semble répondre au besoin d'une plage colorimétrique associée à la couleur.

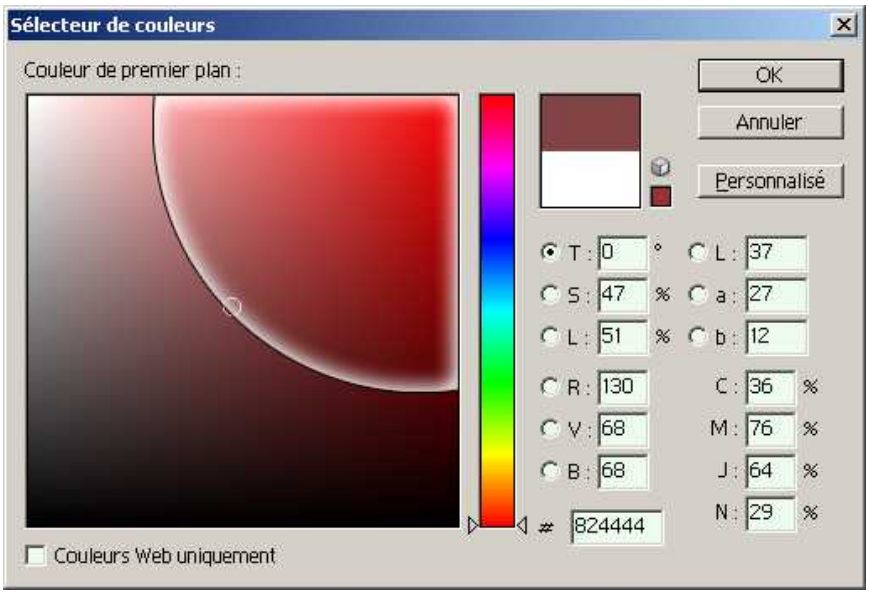


Figure 3 – Plage colorimétrique

A partir de cette image, on se rend compte par exemple que pour la ressemblance entre du rouge pur (R=255, G=0, B=0) et du rouge à la limite du marron (ici sur l'écran : R=130, G=68, B=68), il y a autant de vert que de bleu, et autant de rouge que la somme des deux autres couleurs (68+68=136~130). Il semble donc y avoir une loi de proportion entre le rouge et la somme des valeurs des deux autres variantes, ainsi qu'une loi qui lie le vert et le bleu vers des valeurs proches.

Maintenant, si l'on décale la barre des teintes vers le bas :
- vers le rose (figure 4)
- puis vers le orange jaune (figure 5)

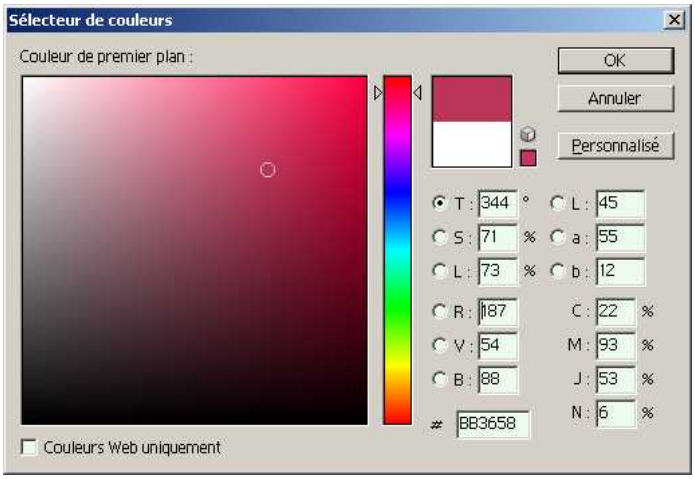


Figure 4 – Plage colorimétrique décalée vers le rose

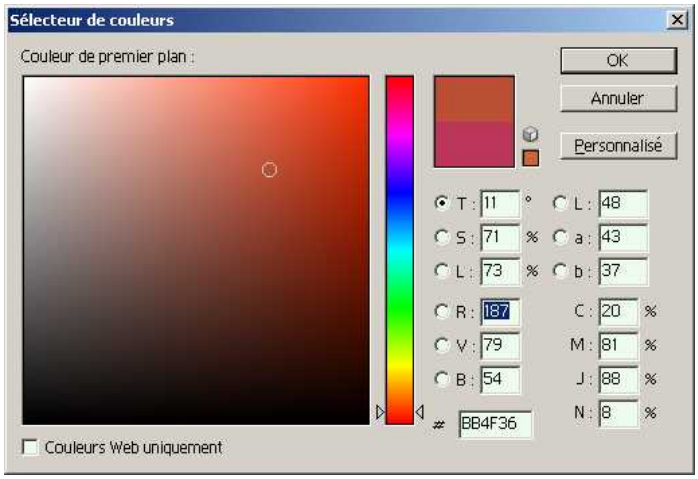


Figure 5 – Plage décalée vers l'orange

on atteint les limites de ce que l'on peut considérer comme rouge sur une image. On remarque que la somme du vert et du bleu est inférieure à la valeur du rouge et que les deux variantes vert et bleu sont très proches. Si l'on augmente vers le rose ou vers le jaune, ces deux valeurs s'éloignent progressivement. A partir de ces deux conjectures on peut d'hors et déjà commencer à élaborer une formule ou un algorithme renvoyant un indice de rouge à partir de ces deux critères.

Il apparaît ainsi nécessaire de mettre des contraintes sur la différence maximale entre le vert et le bleu, puis une luminosité minimale pour ne pas tomber dans le noir trop sombre et enfin formuler un indice de Rouge à partir de la différence entre le rouge et la moyenne des deux autres couleurs. En effet comme on peut le voir sur l'image ci-dessous (figure 6), si on prend un rouge très pale, la somme du vert est du rouge est largement supérieure à la valeur du rouge. Nous prendrons donc la moyenne des deux.

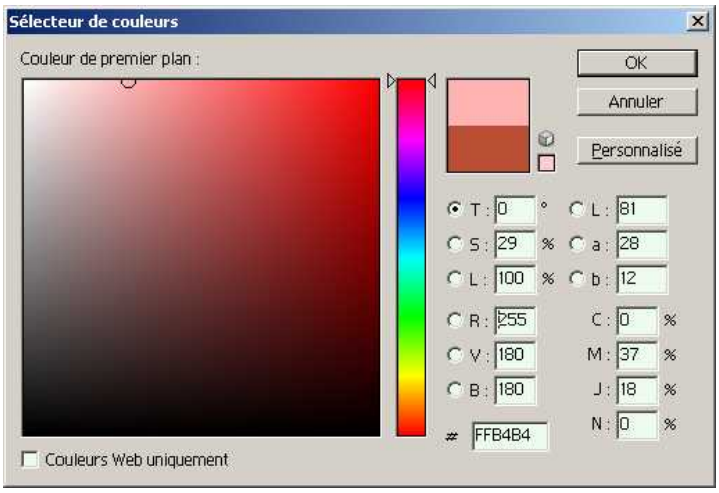


Figure 6 – Rouge pale

D'où l'algorithme :

On ne considère pas la couleur comme rouge si,

1. la différence vert/bleu est trop importante (supérieure à une valeur donnée),
2. la luminosité est trop faible (la couleur peut être rouge, mais trop proche du noir donc vue comme noire, car il faut que l'analyse soit fiable malgré une faible qualité d'image),
3. l'indice de rouge de la couleur étant égal à la différence entre la valeur du rouge et la moyenne de la valeur de bleu et celle du vert (Ir = R – (G+B)/2), est trop faible (inférieure à une valeur donnée).

2.1.2.1.3. Le vert

La définition du vert numérique est recherchée de manière similaire à celle du rouge définie dans la partie précédente. A priori cela devait en tout logique, marcher exactement comme pour le rouge, où les critères n'étaient pas basés sur la couleur de façon absolue, mais uniquement sur la différence de couleur en fonction d'une même teinte.

En déplaçant légèrement le curseur vers le vert, on se rend cependant compte que l'algorithme précédent ne fonctionne pas du tout. En effet sur l'image suivante (figure 7) on peut remarquer que la différence entre le rouge et le bleu (les deux autres couleurs) peut être très grande.

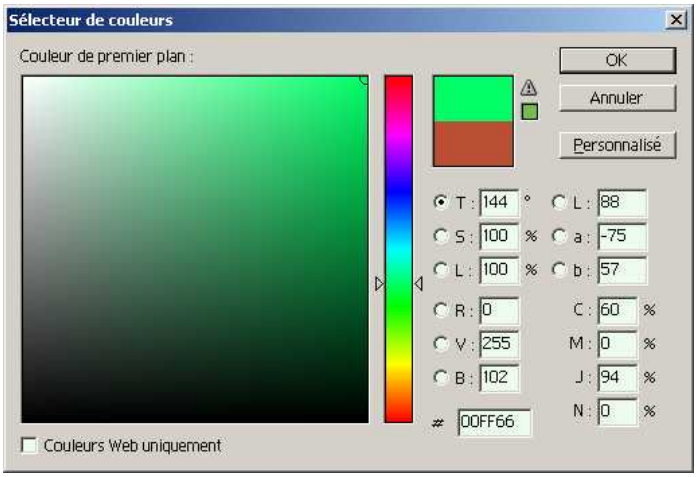


Figure 7 – Variantes du vert

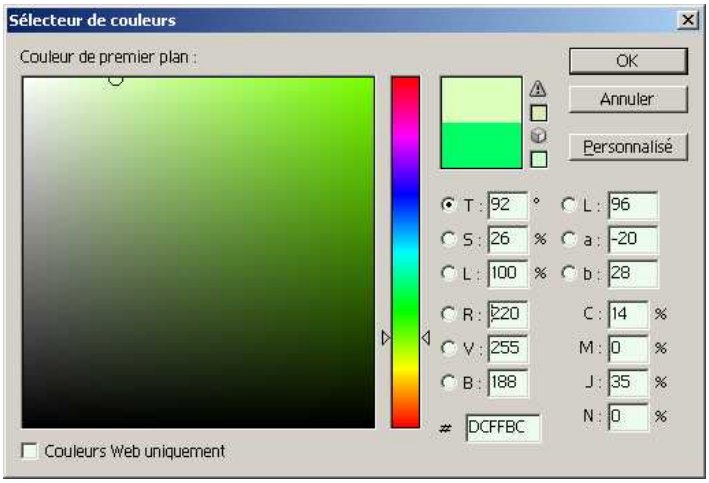


Figure 8 – Variantes du vert

Ici la différence vaut 102, alors qu'avec le rouge cette valeur aurait indiqué que la couleur n'était pas valable. Pour comprendre cette nouvelle logique et en bougeant les couleurs dans les

variantes du vert (figure 8), on se rend compte que le vert est tout le temps la couleur dominante alors qu'une des deux autres couleurs peut passer de 0 jusqu'à la valeur du vert, en obtenant toujours du vert.

A ce stade, on conjecture que l'indice de vert dépend alors de la différence entre la valeur du vert et la valeur la plus grande des deux autres couleurs. Ce qui s'est révélé exact lors des tests effectués sur le Sélecteur de couleurs puis sur des images.

D'où l'algorithme :

On ne considère pas la couleur comme verte si,

1. la luminosité est trop faible,
2. la valeur maximale entre le rouge et le bleu est supérieur à 200 (la couleur est alors trop pale mais elle est aussi trop près d'autres couleurs comme le jaune, et il ne faut surtout pas considérer le jaune comme vert)
3. l'indice de vert, défini comme égal à la différence entre le vert et le maximum des deux autres couleurs, est trop faible.

2.1.2.1.4. Le bleu

Pour le bleu, comme pour le vert, les algorithmes précédents ont été testés. Il s'avère que l'algorithme du rouge a marché parfaitement pour le bleu. Peut être est ce dû au fait que le rouge et le bleu sont des couleurs primaires. La seule différence trouvée est que l'écart entre les deux autres couleurs peut être plus importante que pour le rouge. En effet, on voit sur les images ci-après que le bleu prend une proportion plus importante que le rouge dans la barre des teintes.

D'où l'algorithme :

On ne considère pas la couleur comme bleue si,

1. la différence vert/rouge est trop importante (supérieure à une valeur donnée, autour de 100),
2. la luminosité est trop faible (la couleur est peut être bleue, mais trop proche du noir donc considérée comme noire (similaire au rouge),
3. l'indice de bleu, défini comme égal à la différence entre la valeur du bleu et la moyenne de la valeur de rouge et celle du vert (Ib = B – (R+G)/2), est trop faible (inférieur à une valeur donnée).

Dans les deux images qui suivent on peut voir que la différence entre le rouge et le vert peut atteindre jusqu'à 198 pour la deuxième image, tout en considérant toujours la couleur comme du bleu. Cela traduit simplement le fait qu'on appelle bleu, beaucoup de dérivés du bleu, alors que les dérivés du rouge (orange et rose) ne sont pas considérés comme du rouge.

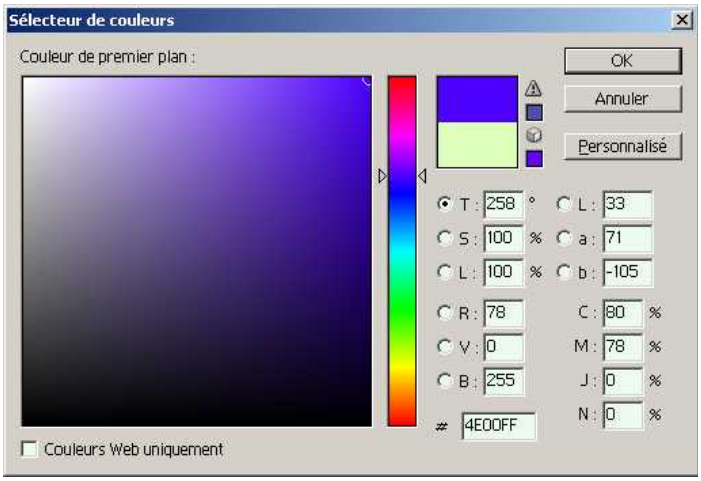


Figure 9 – Variantes du bleu

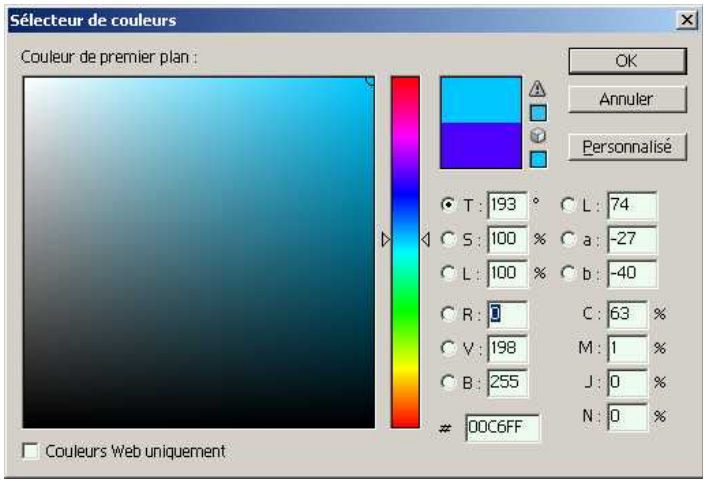


Figure 10 – Variantes du bleu

2.1.2.2. L'algorithme

2.1.2.2.1. Introducti on

Cet algorithme assez compliqué, a été mis au point progressivement lors de nombreuses expériences effectuées avec la caméra. Des parties ont été ajoutées pour améliorer l'analyse. Ces évolutions sont expliquées dans la partie dédiée à l'historique de l'analyse (fin de la partie sur l'analyse).

2.1.2.2.2. Schéma et explications globales

L'algorithme comporte 4 grandes étapes :

1. Analyser l'image pour en sortir des valeurs moyennes et maximales dans chaque couleur. Puis définir une valeur minimale pour chaque couleur.
2. Transformer l'image, de sorte que si elle est trop rouge, bleu ou verte, seules les parties importantes de celle-ci ressortent.
3. Attacher à tous les points une indication sur sa couleur : Pas de couleur, Rouge, Vert ou Bleu
4. Considérer un groupe de points de même indication de couleur et en sortir le centre de gravité.

Le schéma global est celui présenté ci après.

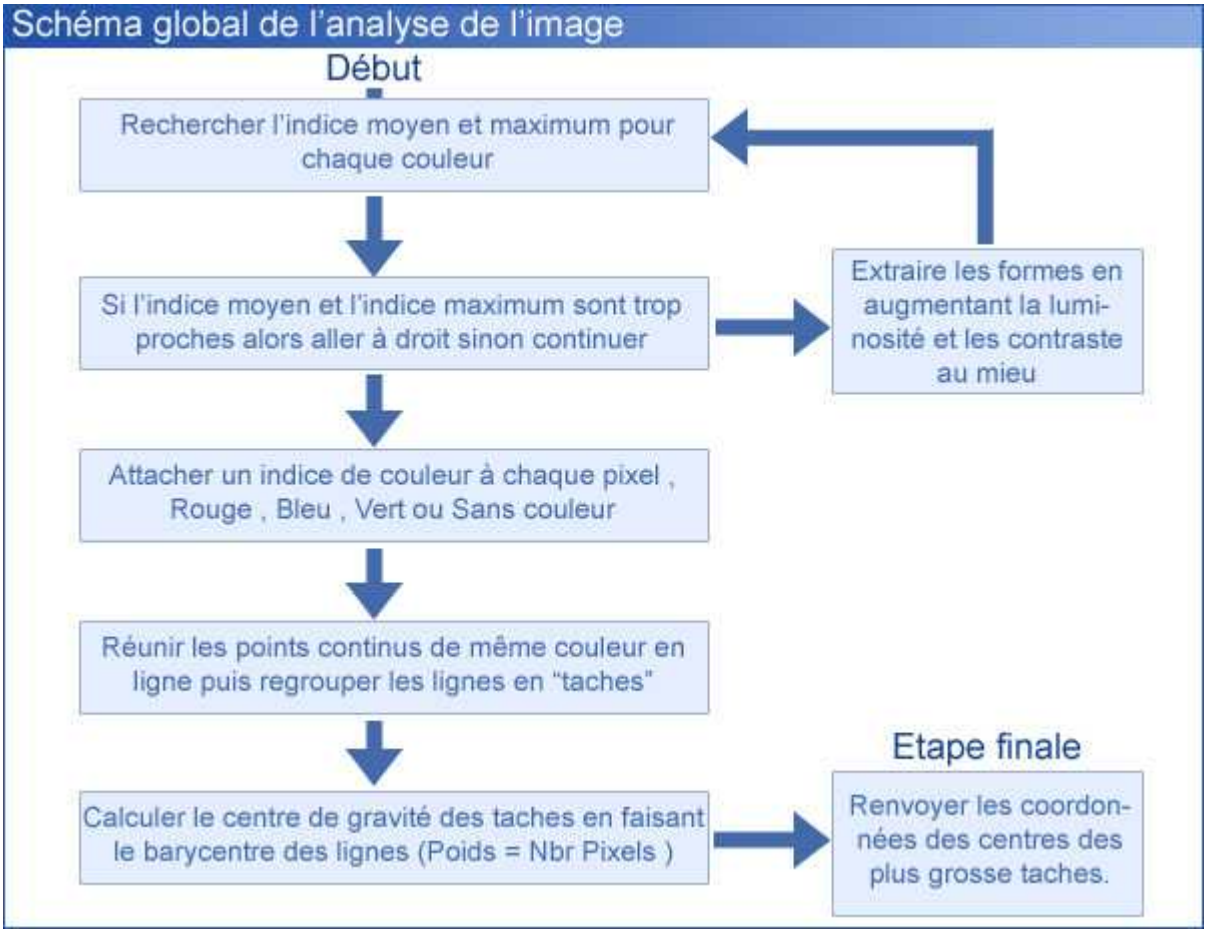
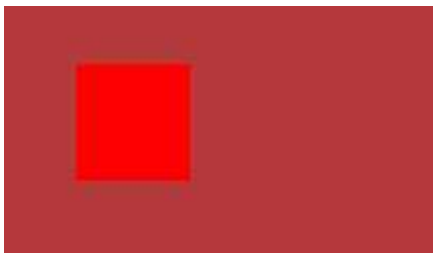


Figure 11 – Organigramme général du traitement

2.1.2.2.3. Indice de couleur minimal (exemple du rouge)

Dans les algorithmes définis dans la partie précédente, nous avons à chaque fois retenu une valeur d'indice minimale à préciser. Plutôt que de la laisser constante, nous avons préféré la faire varier, de manière à ce qu'elle s'adapte parfaitement à l'image. En effet, si nous avions laissé une valeur constante, l'algorithme n'aurait pas pu détecter un carré rouge clair au milieu d'une image légèrement moins rouge comme dans le cas ci-dessous ?



En effet sur l'image ci-contre, on reconnaît facilement une tache rouge alors qu'un algorithme avec des constantes sur la définition du rouge aurait tout considéré comme rouge et aurait renvoyer le centre de l'image. Pour régler ce problème, il a donc été décidé de générer une valeur minimale de rouge, fonction de la moyenne du rouge de l'image (moyenne des indices, voir la partie sur la définition du rouge numérique) et du maximum d'indice de rouge de l'image. La valeur minimale se situe quelque par entre les deux valeurs suivant une fonction comprenant une constante paramétrable qui définit une sensibilité plus ou moins importante : plus la valeur de la constante est grande, plus la sensibilité est forte et inversement.

En appliquant les algorithmes des différentes couleurs, et en se basant sur un indice minimal par couleur dépendant de l'image, on filtre pour obtenir ci-dessous à droite (Noir = Pas de couleur attaché au pixel ; Rouge, vert, bleu = le pixel est considéré comme rouge, vert, bleu) une image contenant une information réduite à l'essentiel :

Figure 12 – Image originale

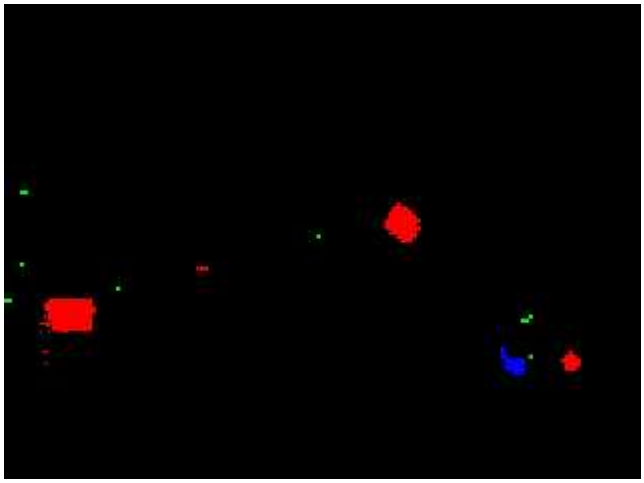


Figure 13 – Image filtrée

2.1.2.2.4. Filtre d'extraction des formes

Après avoir mis au point l'algorithme, nous avons pris en considération les répercutions de la qualité de l'image sur la qualité de l'analyse. Dans nos expériences, les images provenant d'une caméra à faible coût, sont de qualité médiocre. Une image qui à première vue paraît « propre », peut en fait, être assez difficile à analyser. En faisant un zoom important sur une image de la caméra, avec un éclairage assez rouge, on voit apparaître pleins de petits points très rouges au milieu d'autres points beaucoup moins rouges (c'est ce que l'on appelle le bruit). En revenant à l'image non zoomée, on remarque que ces points forment avec leur voisins, une couleur beaucoup moins rouge. L'image qui à première vue semblait tout à fait facile à analyser, peut fausser tous les calculs.

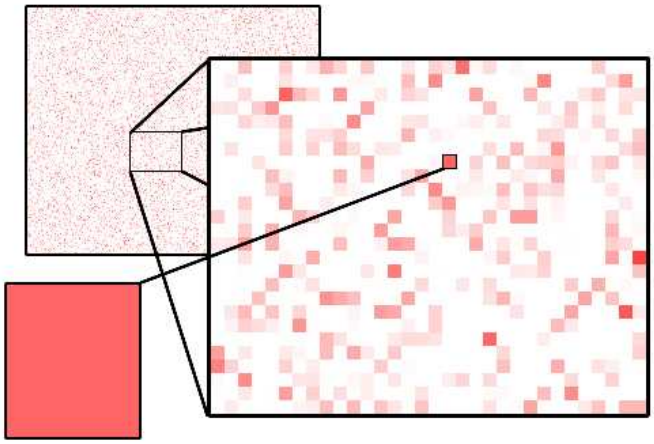


Figure 14 – Fluctuations spatiales de couleur dans une image zoomée

Donc, une image blanche légèrement rougeâtre peut en réalité posséder des points très rouges…. Pour résoudre ce problème, on aurait pu appliquer un flou qui élimine les pixels trop rouges. En fait pour ce genre d'image, l'indice minimum augmente de manière importante en effectuant une moyenne normale avec un maximum assez élevé. Certaines taches qui pourraient être rouges, mais moins que les pixels les plus rouges, ne seraient pas alors détectées. Ajoutons à cela, que des images parfois très rouges du fait d'un mauvais éclairage, ou d'un mauvais réglage de la caméra, deviennent impossibles à analyser (aucune tache n'est trouvée). Nous avons donc envisagé un filtre conditionnel, actif seulement si la moyenne de rouge d'une image devient trop proche du maximum (ou du bleu ou du vert). Ce filtre consiste donc à augmenter les contrastes et la luminosité de telle manière que les taches ressortent de l'image.

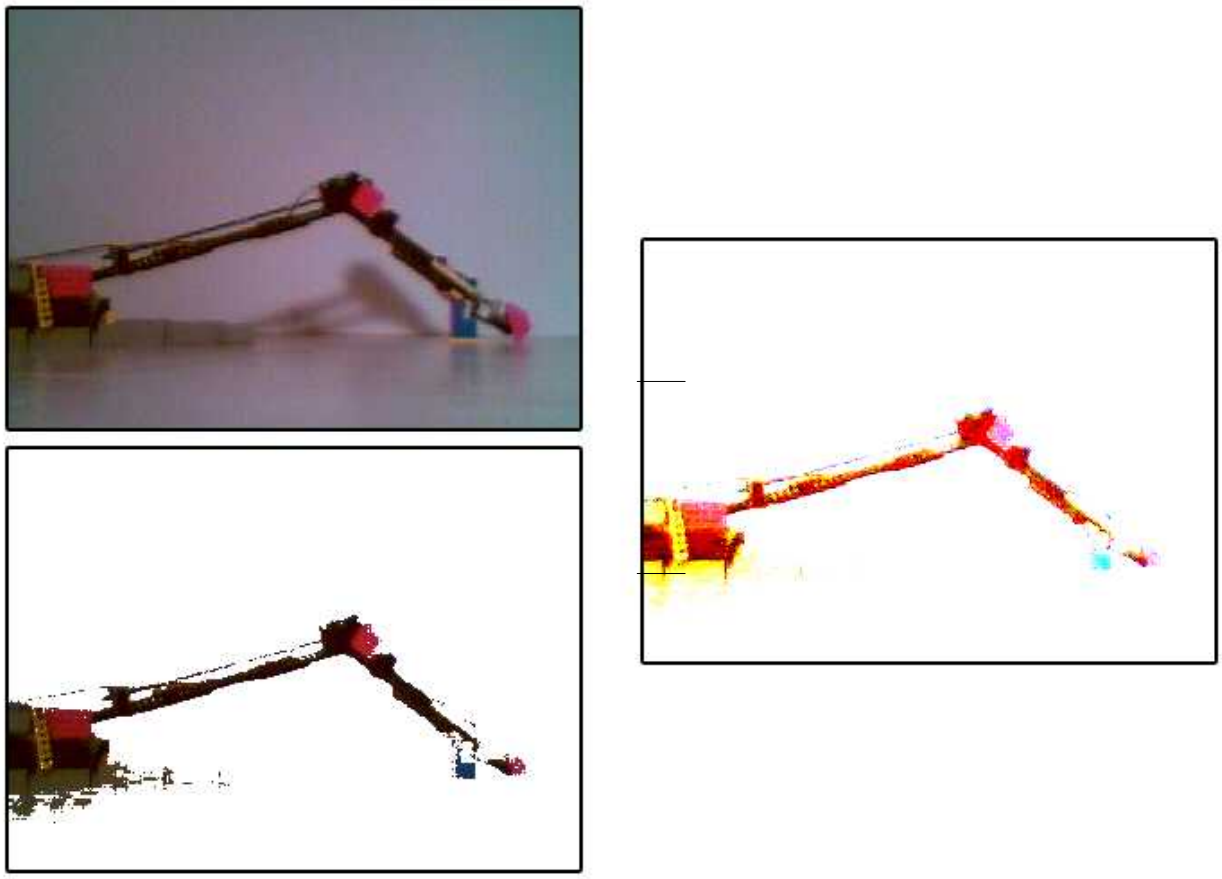


Figure 15 – Résultat de l'application du filtrage

Comme le montre les photos ci-dessus, en augmentant le contraste, le fond rouge et bleu disparaît alors complètement et il ne reste que le robot sombre, avec ses axes en rouge … L'analyse de l'image est alors beaucoup plus facile.

2.1.2.2.5. Rattachement de points à un même ensemble

Cette action comporte 2 principales étapes.

Tout d'abord nous réunissons tous les pixels contigus sur une même ligne, et obtenons ainsi uniquement des lignes de pixels avec leurs coordonnées Y en hauteur et X correspondant au premier pixel de la ligne. Nous communiquons ensuite la taille en pixel de la ligne. Le traitement est fait pour chaque ligne. Les photos ci-dessous montrent respectivement une image et les centres de chaque ligne de rouge trouvés.

Figure 16 – Image originale

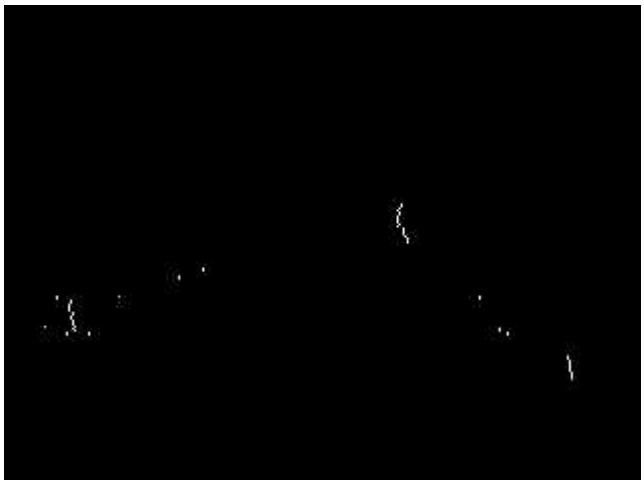


Figure 17 – Centres des lignes rouges

Nous analysons ensuite les lignes dans la hauteur pour trouver des liens. Nous prenons ensuite les lignes qui se suivent et qui se touchent (grâce aux informations de X1 et X2 qui représentent les deux extrémités de la ligne) et nous formons un bloc de pixels.

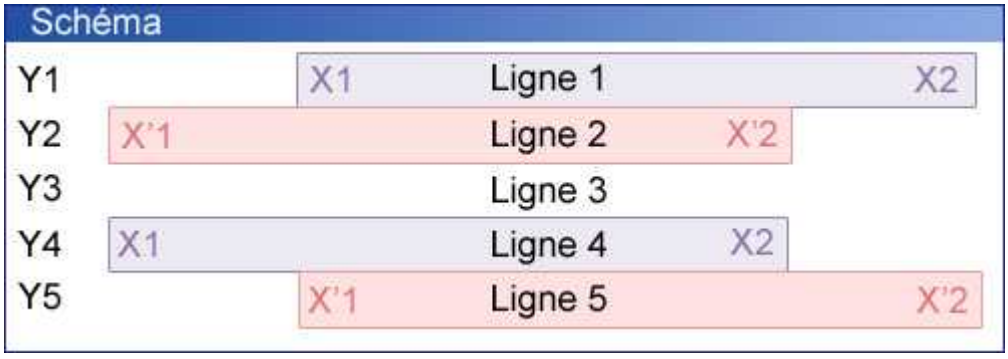


Figure 18 – Rattachement de points à un ensemble

L'algorithme retenu est défini par:

1. Si $Y_{Ligne\ 1} = Y_{Ligne\ 2} + 1$ est vrai est alors les deux lignes dépendent peut être de la même « tache », sinon ne pas effectuer l'action n°2.
2. Si X1 X'1 et X1 X'2 **OU** X'1 X1 et X'1 X2 alors les deux lignes appartiennent à la même « tache »

Nous obtenons des taches, regroupements de plusieurs lignes. Seulement nous oublions les taches en U. Cet algorithme détectera deux taches pour chaque barre avec une image en U … Il faut donc ajouter deux lignes à l'algorithme :

1. Si $Y_{Ligne\ 1} = Y_{Ligne\ 2} + 1$ est vrai est alors les deux lignes sont peut être de la même « tache », sinon ne pas effectuer l'action n°2.
2. Si X1 X'1 et X1 X'2 **OU** X'1 X1 et X'1 X2 alors les deux lignes appartiennent à la même « tache »
3. Si la ligne intercepte deux lignes parallèles, les trois lignes prennent le même numéro de tache.
4. Si deux lignes touchent une même autre ligne, mais que ces deux lignes ne se rencontrent pas, alors les 3 lignes prennent le même numéro de tache.

Il ne reste plus qu'à faire une moyenne des Y et une moyenne des centres de chaque ligne en leur donnant comme « poids » le nombre de pixel qu'elles possèdent (poids qui modifie l'influence de la valeur du centre dans la moyenne) et on obtient le centre de gravité. Il s'agit simplement du barycentre des lignes.

2.1.3. Historique de l'analyse

L'historique de l'analyse rappel la succession des problèmes rencontrés que nous avons dû résoudre.

Dans un premier temps, l'analyse avait été faite beaucoup plus simplement, en se basant sur le principe d'une image de bonne qualité avec des couleurs facilement identifiables. Pour trouver les coordonnées des axes à partir des taches de couleur, une plage de couleur avait été définie pour le rouge, le bleu et le vert. Cette plage pouvait à tout moment être changée. Un problème apparut : souvent l'analyse ne trouvait rien, car l'image provenant de la caméra n'était pas assez nette et les couleurs du fond de scène pourtant blanc, devenaient plus fades avec des teintes de rouge ou bleu aux « yeux » de la caméra.

Nous avons donc du faire évoluer le système pour qu'il fonctionne dans des conditions plus difficiles (actuellement, le système est loin d'être parfait, mais il est beaucoup plus stable). Lors du développement de ce nouvel algorithme, la définition de l'indice de couleur minimal ne se faisait pas avec le maximum de rouge mais juste à partir de la moyenne que l'on multipliait par une constante. Hors cela créait un problème dès que l'image était trop rougeâtre. En effet la moyenne et le maximum (correspondant à la tache en tout logique) étaient trop proches et l'indice minimum ainsi généré à partir de la moyenne était au dessus du maximum : l'analyse était donc incapable de trouver un seul pixel considéré comme rouge.

Une autre étape importante de l'analyse fut la possibilité de détecter une forme en **u** ou en **n**. Comme cela est expliqué dans la partie sur le rassemblement des pixels en un ensemble de points, l'algorithme n'était testé au début, que sur des taches pleines. Si la forme ne faisait pas difficulté, les images étaient difficiles à analyser, car les couleurs étaient trop proches du gris ou trop fades. Hors quand on a mis une forme en U à analyser, l'algorithme a trouvé deux taches (une pour chaque branche du U), et nous avons dû refaire la détection des « taches ».

Un autre problème pour l'analyse, fut de donner à l'algorithme des tailles minimales de tache (pour éviter les regroupements de 2 pixels ou même d'un seul pixel). Hors ces tailles minimales étaient appliquées lors de l'analyse horizontale puis verticale. Donc si une ligne de 2 pixels (pas pris en compte) faisait la jointure entre deux autres lignes (prisent en compte), alors l'analyse détectait deux taches (la ligne de 2 pixels n'est pas considérée comme une ligne « valide »).

Enfin le dernier problème à résoudre est venu des images où la différence entre la moyenne et le maximum des indices de couleur était trop faible. Une image trop rouge était impossible à analyser : l'algorithme détectait soit beaucoup trop de taches, soit aucune tache. Nous avons alors ajouté le **Filtre d'extraction des taches** qui permet de les faire ressortir en modifiant les fonds d'images en blanc (voir image sur la partie concernant le filtre).

2.2. Interprétation des résultats de l'analyse

L'interprétation par l'ordinateur, des résultats de l 'analyse lui permet d'une part de contrôler la position du bras par rapport à l'objet et d'autre part de définir et de commander les opérations qui lui permettront de le saisir. Nous présentons ici les différentes étapes correspondantes. Des extraits des fonctions les plus importantes sont donnés en fin du document.

Contrôle des positions

La partie reconnaissance fourni un ensemble de points de différentes couleurs. La première action pour l'ordinateur est de reconnaître le robot et l'objet à partir de ces points. Pour ce faire l'ordinateur utilise des règles physiques du robot :

- le premier segment du bras ne peut être en dessous de sa base.
- la pince ne peut être au dessus du premier segment.

Les figures ci après illustrent l'interprétation par l'ordinateur, de la situation du bras mécanique et de l'objet (figure 20) à partir de la position des points (figure 19).

Figure 19 – Exemple de position de points

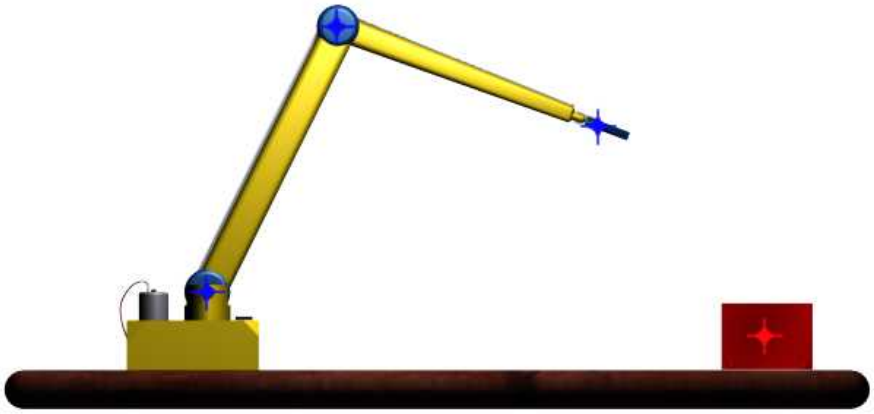


Figure 20 – Interprétation de la position du bras et de l'objet

Une fois les bras du robot et l'objet repérés, l'ordinateur cherche la position que doit avoir le robot pour saisir l'objet. C'est donc pour lui la position idéale à atteindre.

Détermination des mouvements pour atteindre l'objet

Il s'agit de déterminer comment atteindre la meilleure position pour saisir l'objet.

Règle fixant la position relative des segments : La longueur des segments du robot étant fixe, l'ordinateur imagine deux cercles ayant pour centres respectifs la base du robot et le centre le l'objet. Les rayons des cercles sont définis par la longueur des segments du bras du robot.

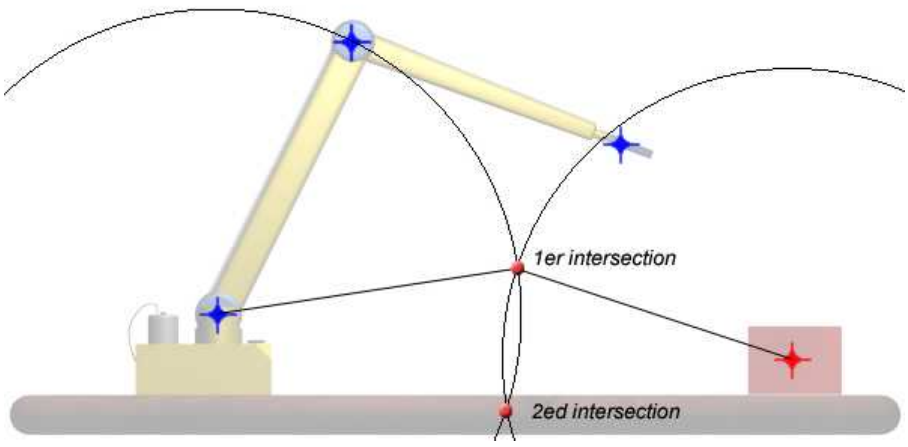


Figure 21 – Choix des positions relatives des segments

Les deux points d'intersection des deux cercles sont les positions idéales du milieu du bras.

- Si deux intersections sont trouvées, alors l'ordinateur prend celle qui réalisable le plus facilement. Sur l'exemple figure 21, l'ordinateur prend la première intersection car la seconde est impossible : en effet le bras ne peut pas passer à travers la table.
- Si une seule intersection est trouvée, l'ordinateur la prend par défaut.
- Si aucune intersection n'est trouvée ou si aucune n'est possible, l'ordinateur considère qu'il ne peut pas prendre l'objet et il envoie un message d'erreur en conséquence.

Détermination des mouvements angulaires : L'ordinateur doit ensuite déterminer les mouvements angulaires nécessaires pour atteindre l'objet. Comme cela est représenté sur la figure 22, il calcul les différences angulaires entre la position initiale et la position à atteindre

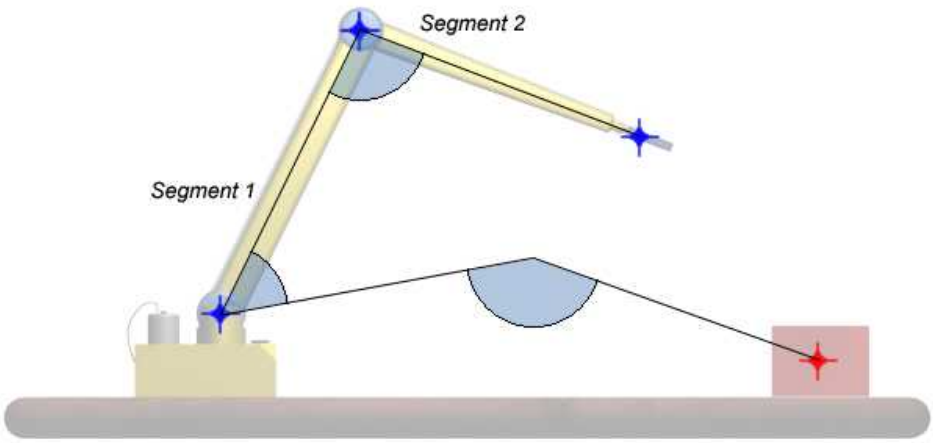


Figure 22 – Détermination des mouvements angulaires

Il trouvera par exemple que le segment 1 doit ce déplacer de 60° et le segment 2 de 90°.

Consignes du mouvement

L'ordinateur a comme variable interne la vitesse de rotation des axes du robot. Il sait par exemple que la vitesse de l'axe 1 est de 5°/s et que celle de l'axe 2 et de 8°/s. Il utilise ces caractéristiques et fait le rapport pour trouver pendant combien de temps il doit actionner les moteurs du robot pour atteindre la position voulue.

Ex : 60 / 5 = 12 90 / 8 = 11.25

Les moteur de l'axe 1 tournerons donc pendant 12 secondes dans le sens indirect et ceux de l'axes 2 tournerons pendant 11.25 s dans le sens direct.

Contrôle permanent du mouvement

L'ordinateur fait bouger ainsi le robot à partir de sa position initiale vers la position finale. Pendant la réalisation du mouvement, il recalcule celui-ci en permanence. Les résultats sont exploités soit pour confirmer le mouvement, soit pour le corriger : au final, il se rapproche progressivement de la position idéale avec une bonne précision. Une fois la position voulue atteinte, il pourra fermer sa pince pour prendre l'objet et continuer selon les instructions qui lui aurons été données, à le manipuler.

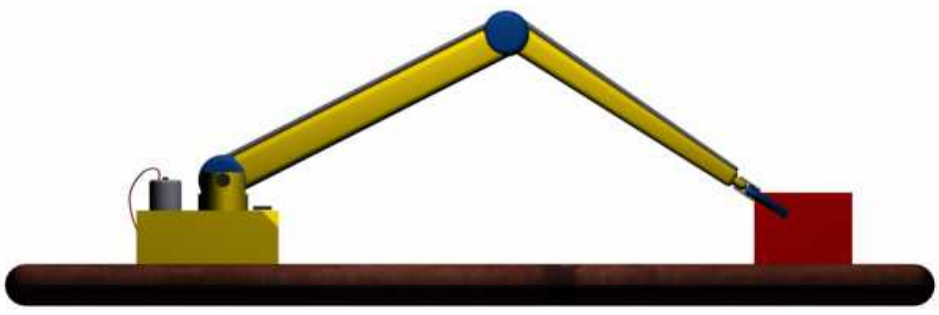


Figure 23 – Cible atteinte par le bras m écanique

2.3. Contrôle du robot

La communication de l'ordinateur avec le robot se fait par l'intermédiaire d'une carte électronique présentée plus loin et connectée avec le port parallèle DB25.

La carte offre le contrôle de 7 moteurs à courrant continu dans les deux sens. Pour notre maquette, seules trois des sept sorties ont été utilisées :

- Rotation du segment 1
- Rotation du Segment 2
- Ouverture/Fermeture de la pince.

Les sorties restantes non inutilisées peuvent servir à une éventuelle amélioration du robot.

Protocole de contrôle du robot :
Le contrôle ce fait par envoi sur le port parallèle d'un octet. Les 7 premiers bits de l'octet de commande servent à sélectionner le moteur à commander. Le bit restant indique le sens de rotation du moteur.

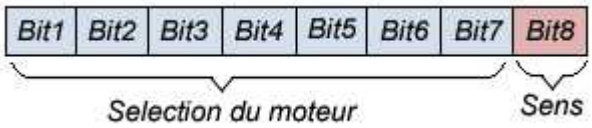


Figure 24 – Octet de commande

Par exemple :

00000000	0x00	Tout les moteurs sont au repos.
10000001	0x81	Fait tourner le moteur 1 dans le sens direct.
10100000	0xA0	Fait tourner le moteur 1 et 3 dans le sens indirect

Dans le protocole, les moteurs ne sont jamais utilisés en même temps.
Pour déplacer par exemple l'axe 1 de 60°, en sachant que sa vitesse de rotation est de 5° par seconde, l'ordinateur envoie l'octet de contrôle 10000001 (0x81), attend 12 secondes puis immobilise tous les moteurs en envoyant l'octet de contrôle 00000000 (0x00).

Le robot ne possédant pas d'autre capteur que la caméra, l'état de la pince n'est pas connu en début de séquence. Pour remédier à ce problème, la pince est toujours ouverte en début de séquence. A la fin d'une action la pince doit donc toujours être réouverte.

3. Partie électronique

La gestion électronique du robot est assurée par une carte qui a été spécialement définie et réalisée pour les besoins de la maquette de notre TPE. Son schéma détaillé est donné en annexe (Annexe 5.2). Alimentée sous 12V, elle se branche sur le port parallèle DB25 du PC avec une tension de 3V. La prise DB est représentée figure 25 ci-dessous.

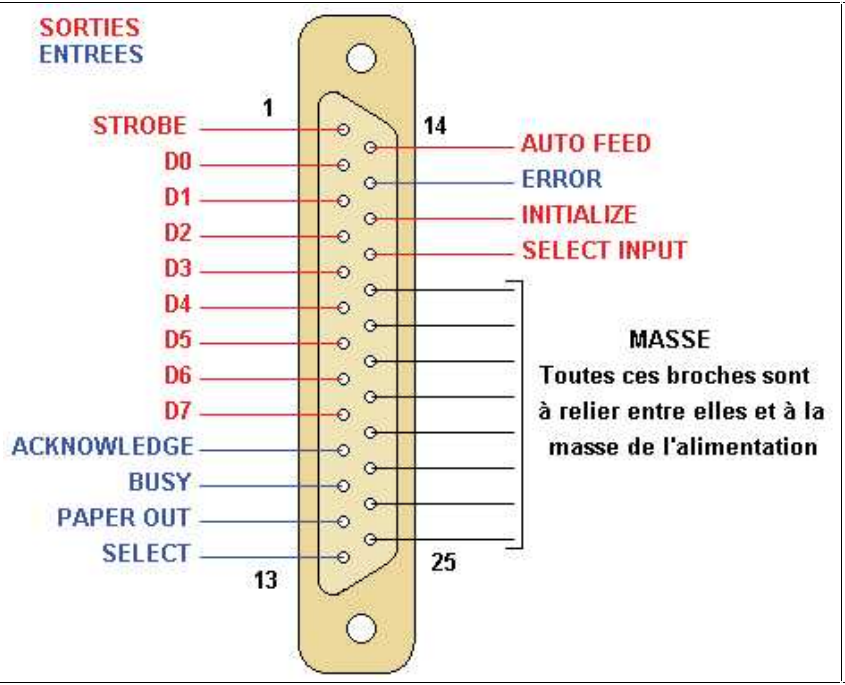
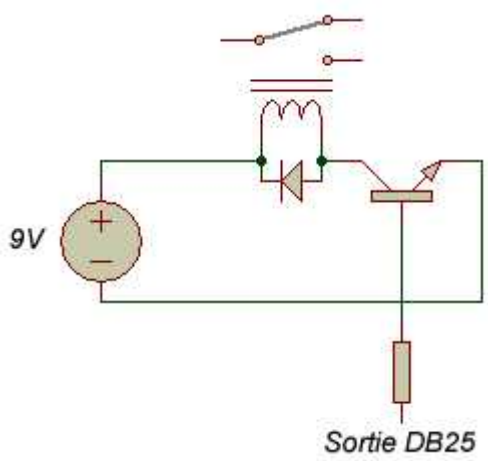


Figure 25 – Prise DB25

Alimentation des moteurs

La prise DB délivre 3V ce qui est très insuffisant pour alimenter les moteurs du robot. La carte joue donc entre autre un rôle d'amplificateur. Elle utilise des relais ce qui lui permet en plus d'isoler l'ordinateur de la partie moteur. Ainsi en cas de problème la carte mère du PC ne risque pas de subir de dommage.



Chaque sortie du DB25 est relier un module de commutation du type représenté figure 26.

Il permet de faire commuter un relais avec les 3V du DB 25 et donc de manipuler de plus grandes tentions.

La carte est alimentée par un transformateur 9V. (Voir le schéma de la carte dans les annexes en fin de dossier)

Figure 26 – Module de commutation

4. Partie mécanique

4.1. Introduction

Pour la réalisation du projet, nous avions besoin d'un bras manipulateur qui se déplace selon deux axes et d'une pince pour la préhension d'un objet.
Nous avons tout d'abord commencé par essayer de modéliser une pince qui fonctionne avec un moteur électrique (contrainte fixée par le cahier des charges). Cette conception est détaillée dans la partie 4.1.

Des raisons de coûts et de difficultés pour trouver et usiner les matériaux, nous ont amené à prendre la décision d'abandonner cette solution pour réaliser le bras mécanique en lego. Et d'autant plus que le pilotage du robot est complexe à cause des différences entre les moteurs (tension d'alimentation qui diffère et rapport de réduction qui diffèrent aussi).

La partie de conception du robot en lego est détaillée dans la partie 4.2

4.1.1. Modélisation de la pince sous Solidworks

La pince du robot a été modélisée sous Solidworks pour nous permettre de déceler des problèmes mécaniques possibles lors de la conception future de la pince.

Pour cela nous avons commencé par la conception d'un doigt de pince avec tout le système de bielles qui doit lui permettre de s'ouvrir ou de se fermer.

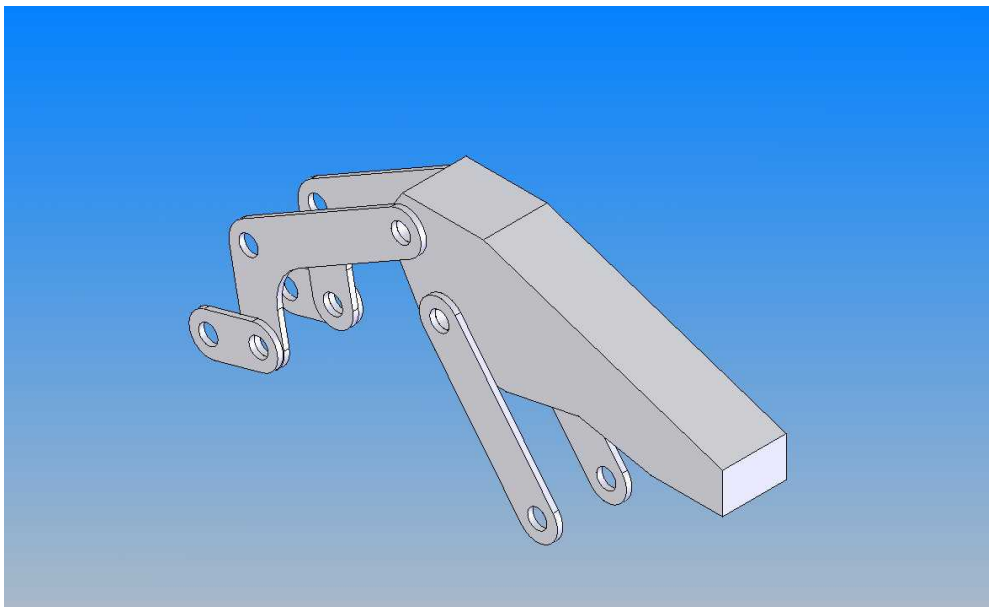


Figure 27 – Modélisation de la pince sous Solidworks

A ce stade de la conception les contraintes de coaxialité ne posent pas de problèmes.

L'étape suivante consiste dans l'assemblage des deux doigts de pince avec l'axe moteur et l'écrou (figure 28).

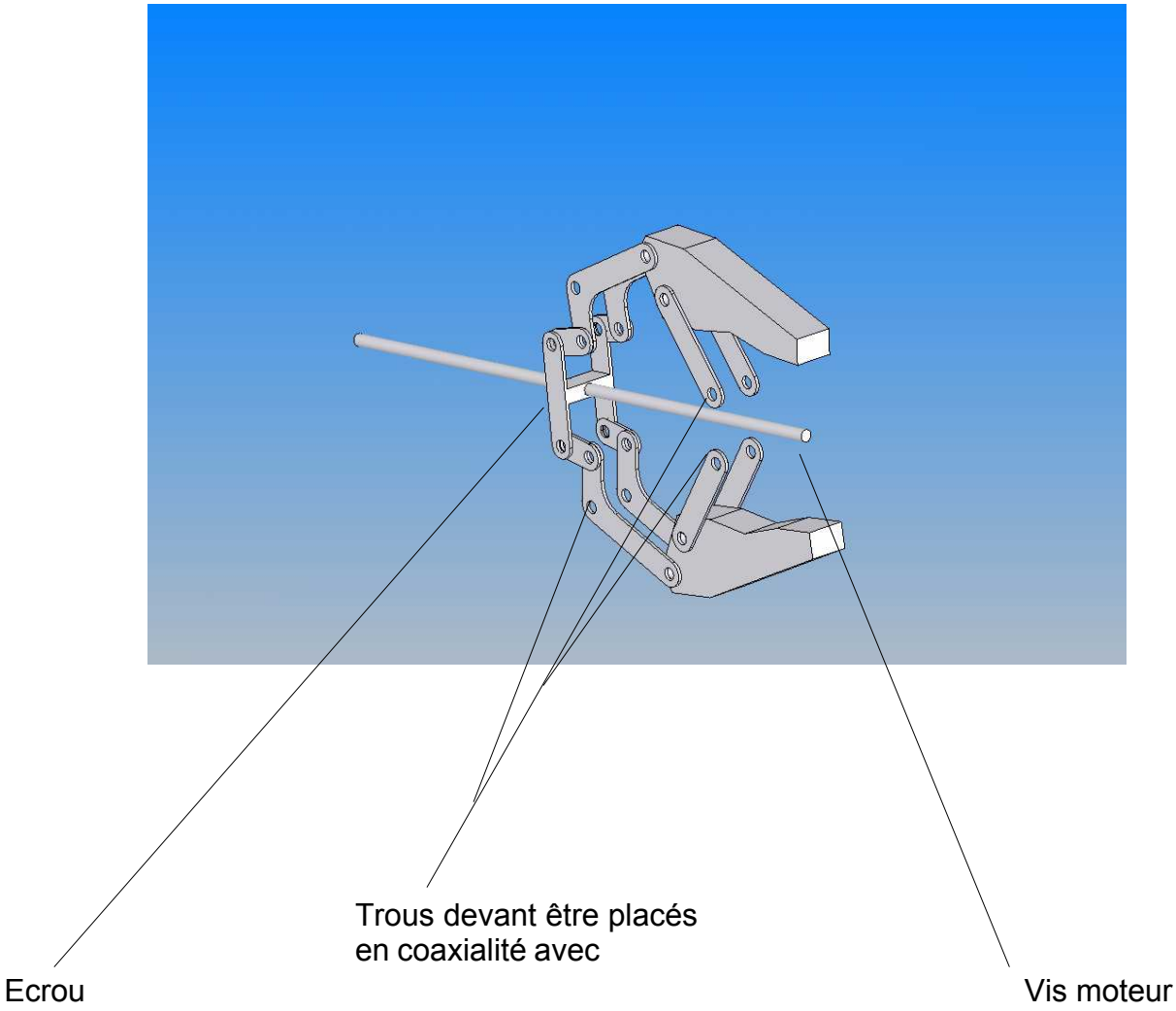


Figure 28 – Assemblage des doigts de la pince avec la vis moteur

Nous avons rencontré des difficultés au niveau des contraintes d'assemblage pour fixer la pince (figure 28) par rapport au bâtit représenté sur la figure 29. Par ailleurs nous avons pu remarquer que nous risquerions d'avoir un problème au niveau de la vis moteur. En effet si l'on veut une bonne ouverture de la pince, il faut une vis moteur assez longue. Elle risquait de gêner lors de la fermeture de la pince. Ce problème restera à résoudre, lors de la conception finale de la pince.

Nous n'avons réussi à fixer la pince (ci-dessus) au bâtit (ci-dessous), pour des raisons de contraintes. En effet quand on insère des assemblages dans des assemblages, le logiciel ne reconnaît plus et ignore les contraintes du premier assemblage qu'il considère comme rigide.

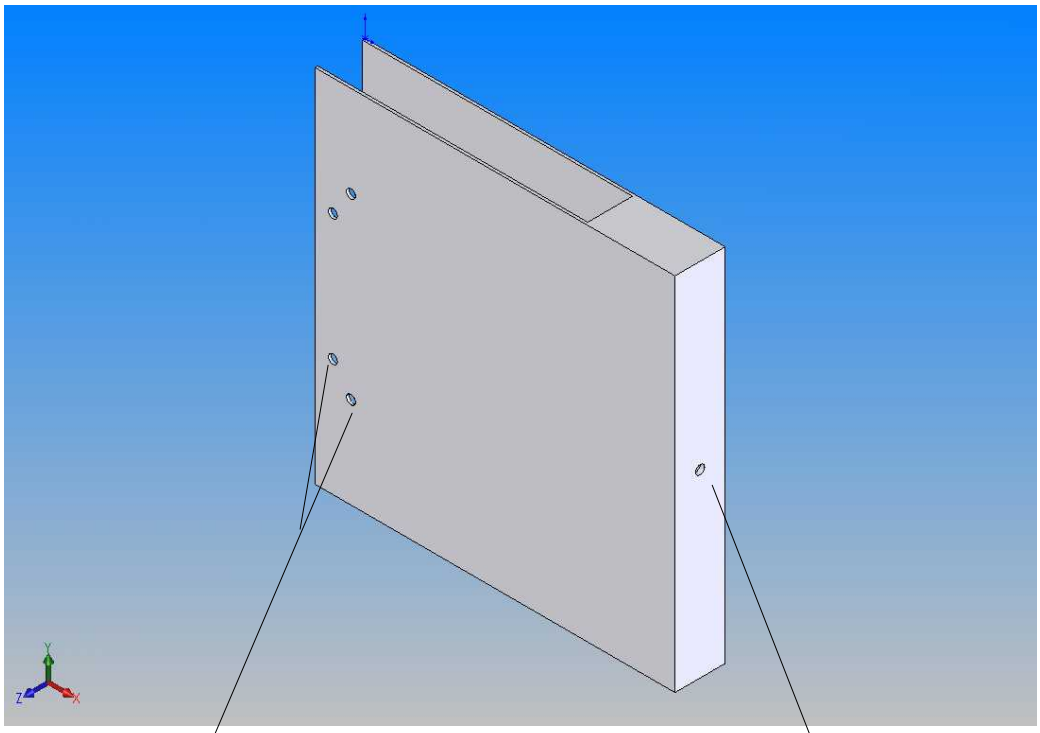


Figure 29 – Bâtit

4.1.2. Recherche des matériaux adéquats en fonction du problème

Pour trouver une solution la plus économique qu'il soit, nous avons pensé découper chacune des pièces de la pince (Doigt, bielles…) dans des plaques de matières plastiques ou de bois assez épaisses. Toutes les liaisons pivots seraient assurées par un ensemble vis écrou rondelle avec deux morceaux de caoutchouc faisant office d'effecteur sur la pince.

Ensuite, pour la motorisation de l'ensemble nous avons fait des tests avec les moteurs dont on disposait, soit :

- un moto réducteur 6 vdc qui tourne à 40 Tr/Min avec un rapport de réduction égal à 75
- un moto réducteur 12 vdc qui tourne à 10 Tr/Min avec un rapport de réduction égal a 500

Pour permettre une meilleure localisation de la pince par la partie électronique et sachant que le moteur 12 vdc a un rapport de réduction important (il a donc un meilleur couple), nous avons choisi de faire tourner l'axe moteur avec le moteur 12 vdc.

Ce sont ces problèmes de conception et d'études qui nous ont incité a réaliser le bras avec des legos avec un objectif de simplicité.

4.2. Conception du bras manipulateur en lego

4.2.1. Schéma cinématique du bras

La conception du bras en lego s'appuie sur les schémas cinématiques suivants. Ils concernent le bras manipulateur et la pince.

Bras manipulateur

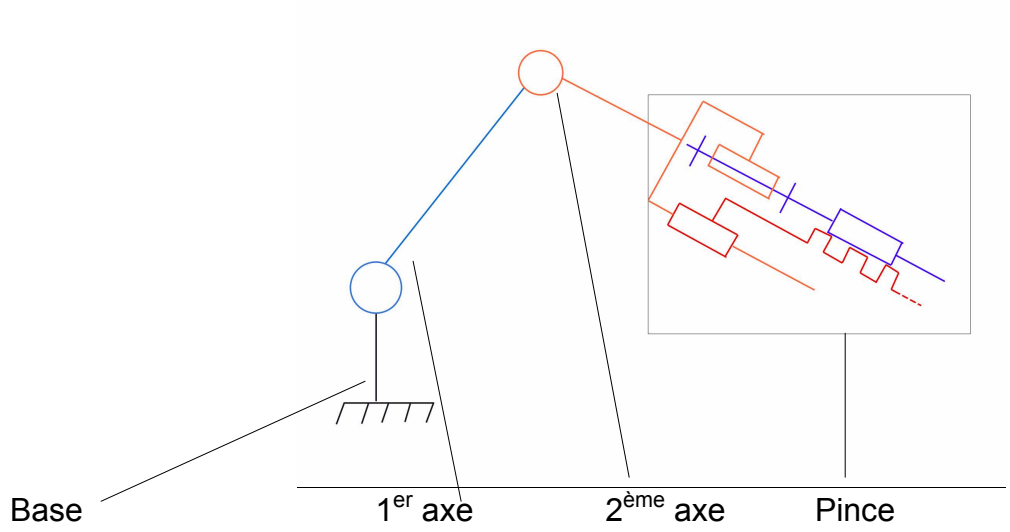


Figure 30 – Schéma cinématique du bras

Pince

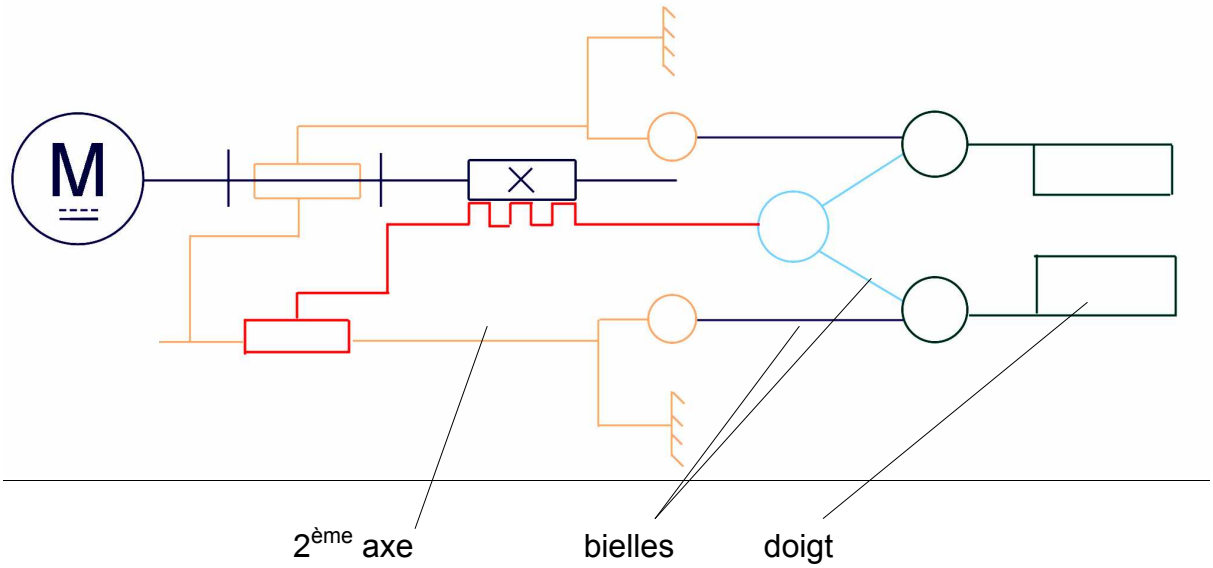


Figure 31 – Schéma cinématique de la pince

4.2.2. Description du fonctionnement

4.2.2.1. Motorisation et réduction

Étant donné que nous avons choisi de réaliser le bras manipulateur en lego, nous adoptons pour motoriser ce dernier, des moteurs lego qui sont alimentés en 9V. A cause du poids des axes et de la pince, les moteurs n'ont pas un couple suffisant pour lever les deux axes, la pince, et en

plus éventuellement un objet. En plus, la vitesse de rotation de moteur est trop élevée pour permettre à l'ordinateur de pouvoir faire l'analyse. La vitesse de déplacement des axes est trop importante, il faut donc adapter ce mouvement de sortie moteur. Nous avons donc choisi d'utiliser un réducteur pour pouvoir adapter le mouvement, ce qui doit permettre de diminuer la vitesse de rotation et d'augmenter le couple en sortie moteur . C'est en prenant toutes ces contraintes de base que nous avons commencé à réaliser le bras.

4.2.2.2. Réalisation des liaisons pivots des deux axes

Nous avons réalisé des liaisons pivot sur le bras pour lui permettre de s'articuler selon ses axes. Nous les avons réalisées en faisant passer un axe qui assure la mise en position de la liaison, puis nous avons mis des pignons en liaison encastrement de chaque coté de l'axe pour assurer le maintient en position. Les liaisons de bases étant ainsi réalisées il ne manque plus qu'à les motoriser.

4.2.2.3. Motorisation de la liaison pivot du premier axe

La liaison pivot du premier axe qui lie ce dernier à la base, doit supporter le poids des deux axes, plus celui de la pince qui porte éventuellement un objet; donc le rapport de réduction qui lie la sortie du moteur à la sortie du réducteur doit être assez important. En montant le bras nous nous sommes aperçus, que malgré toutes ces précautions, le moteur forçait pour faire monter le bras. Alors nous avons couplé deux moteurs qui tournent en sens opposé, pour faire tourner le pignon d'entrée réducteur; Le problème du poids du bras a été ainsi d'autant mieux résolu, que pour assurer encore une meilleure fiabilité, nous avons rajouté un élastique, qui aide les moteurs à faire monter le bras et qui limite la conséquence du poids lors de sa descente.

Quand au réducteur, il est réalisé en couplant des réducteurs à train d'engrenage avec des liaisons roue et vis sans fin. Pour motoriser la liaison pivot entre la base et le premier axe, nous avons relié le réducteur à un pignon monté en liaison encastrement par rapport au premier axe. Pour des raisons encore de fiabilité, nous avons transmis le mouvement de rotation en sortie du réducteur sur les deux cotés de l'axe du premier axe, afin que les efforts qui agissent au niveau de la liaison soient parallèles.

Le rapport de réduction a été calculé et il est de 1/5184. Cet axe met alors 13 secondes pour parcourir sa course angulaire maximale.

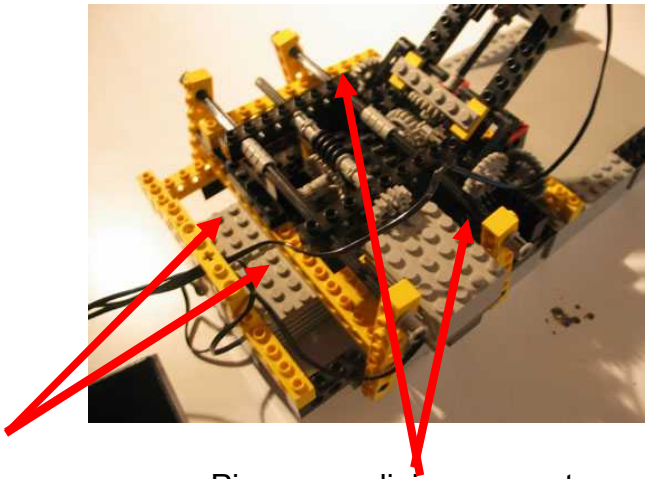
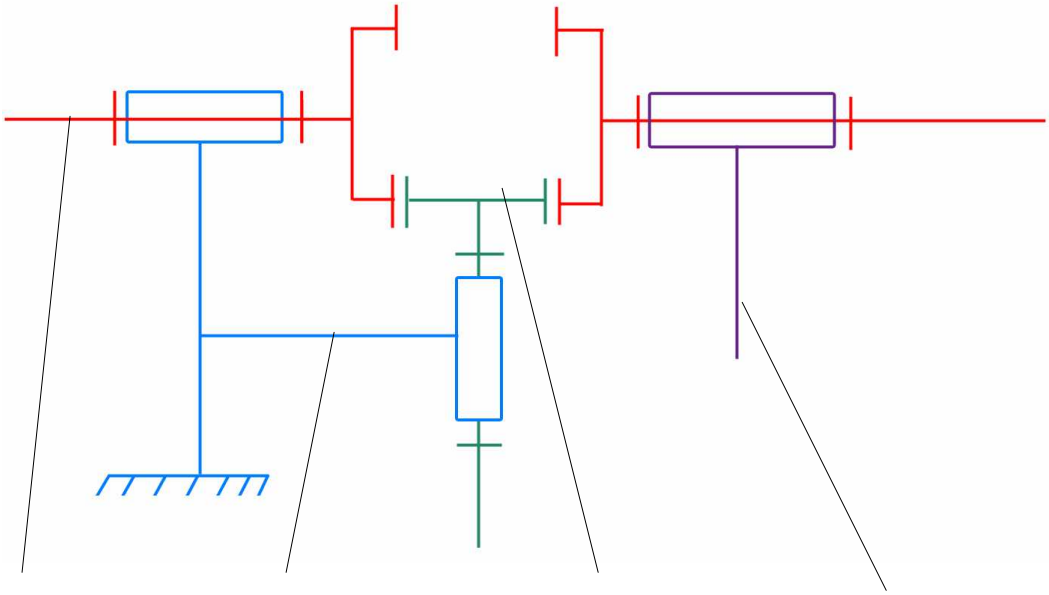


Figure 32 – Motorisation de la liaison pivot du premier axe

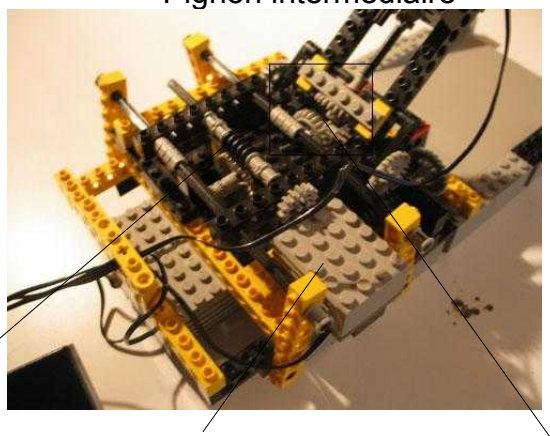
4.2.2.4. Motorisation de la liaison pivot du second axe

La liaison pivot de cet axe ne supporte pas autant de poids que celle de l'autre axe, donc par conséquent le rapport de réduction peut être plus grand. Nous n'avons besoin que d'un moteur, car il faut moins de couple pour motoriser cette liaison. Le réducteur est basé, comme pour l'axe précédent sur un accouplement entre réducteur à train d'engrenage, et des réducteurs de type roue et vis sans fin. Le rapport de réduction résultant est de 1/3456.

A ce stade, il faut résoudre un autre problème : il s'agit de transmettre le mouvement de rotation, qui sert à motoriser cet axe, au travers du premier axe, car à cause du poids, le réducteur et le moteur de cet axe ont été placés sur la base. Nous avons donc mis au point une liaison utilisant deux pignons à dentures latérales et un pignon à denture droite et nous les avons placés comme sur le schéma cinématique ci dessous, de façon à ne pas influer sur le rapport de réduction total du réducteur. Enfin les deux pignons en liaison encastrement avec le deuxième axe permettent de mobiliser celui-ci.



Réducteur Base Pignon intermédiaire 1er axe



Réducteur roue et vis sans fin Moteur adaptateur permettant de changer d'axe

Figure 33 – Motorisation de la liaison pivot du second axe

4.2.2.5. Réalisation et motorisation de la pince

Pour la pince un moteur plus petit donc moins lourd est utilisé pour pouvoir mobiliser celle-ci. Pour l'instant elle ne doit prendre que des objets de faible masse (stylo, papier, plaque de lego…). Le moteur est donc monté directement sur le deuxième axe du bras, pour éviter d'avoir à faire

passer le couple nécessaire à la fermeture de la pince au travers de différents groupes qui ne sont pas cinématiquement liés.

Nous avons ensuite adapté le mouvement de rotation de sortie moteur, en mouvement de translation, grâce à une liaison vis sans fin crémaillère. Grâce à un jeu de bielles fixées soit au deuxième axe, soit aux doigts de la pince, ou soit à la crémaillère, nous pouvons actionner la pince grâce au mouvement de translation issue de la liaison entre la crémaillère et la vis sans fin. Le couple et la vitesse de rotation en sortie moteur sont satisfaisants pour notre étude et nous n'avons pas besoin d'adapter ce mouvement avec un réducteur.
Son schéma cinématique complet est présenté dans la partie 4.2.1.

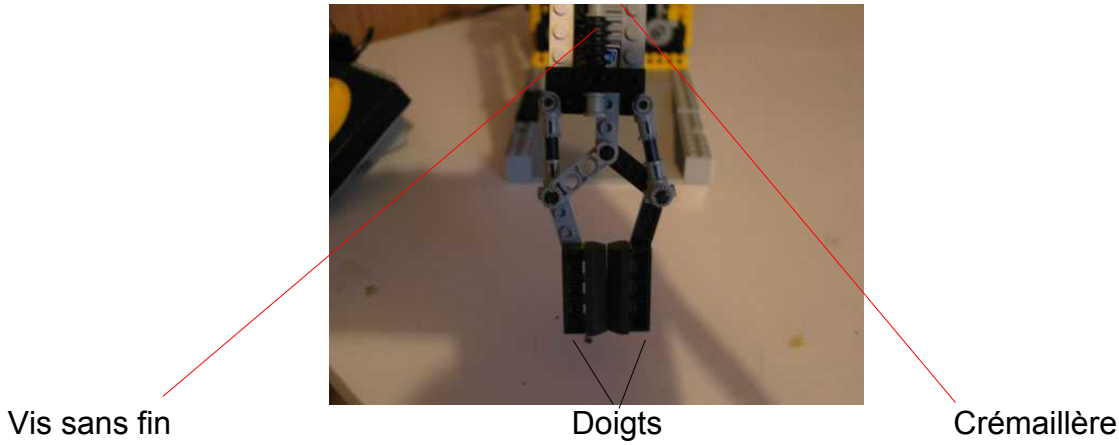


Figure 34 – Motorisation de la pince

4.3. Conclusion

Le robot est désormais monté et est prêt pour les tests expérimentaux. Pour être complet, il reste le montage des pièces rouges sur les liaisons, pour que la partie commande puisse fonctionner par repérage et puisse piloter les différentes actions du bras manipulateur.

Figure 35 – Bras mécanique articulé complet avec ses motorisations

5. Annexes

5.1. Annexe informatique

5.1.1. Analyse de l'image

Voici le code de toute la partie analyse en vb (elle consiste en un module de classe). Il existe aussi une version cpp que nous ne mettons pas à disposition, car il y a encore des bugs (ne pas surcharger les annexes pour rien). La fonction utilisée pour l'analyse est getPoints.

Liste des fonctions :

- getPoints : fonction principale qui renvoie les coordonnées des taches
- drawCross : dessine une croix de couleur sur l'image aux coordonnées données
- drawPixel : dessine un pixel de couleur sur l'image aux coordonnées données
- deleteTache : supprime une tache de la liste des taches (utilisé quand deux taches se touchent)
- PixelIsRed, PixelIsBlue, PixelIsGreen : dit si un pixel a telles coordonnées est rouge, bleu ou vert.
- AnalyseTachesAndDrawCross : Renvoie les coordonnées des taches à partir d'une liste de lignes de couleur et dessine une croix de la couleur de la tache en son centre de gravité.

```
Private Type BITMAP
    bmType As Long
    bmWidth As Long
    bmHeight As Long
    bmWidthBytes As Long
    bmPlanes As Integer
    bmBitsPixel As Integer
    bmBits As Long
End Type
Private Type Coord
    X As Long
    Y As Long
    Height As Integer
    Width As Integer
End Type
Private Declare Function GetObject Lib "gdi32" Alias "GetObjectA" (ByVal hObject As Long, ByVal nCount As Long,
 lpObject As Any) As Long
Private Declare Function GetBitmapBits Lib "gdi32" (ByVal hBitmap As Long, ByVal dwCount As Long, lpBits As Any
) As Long
Private Declare Function SetBitmapBits Lib "gdi32" (ByVal hBitmap As Long, ByVal dwCount As Long, lpBits As Any)
 As Long
Private Type TacheLine
    Xmin As Integer
    Xmax As Integer
    Y As Integer
    Tache As Integer
End Type
Private indiceRougeMinimum As Long
Private indiceBleuMinimum As Long
Private indiceVertMinimum As Long


Private Enum CrossColors
    RED_CROSS = 1
    BLUE_CROSS = 2
    GREEN_CROSS = 3
End Enum

Private Type TachesCoords
```

```
    X() As Long
    Y() As Long
End Type

Private PicBits() As Byte
Private tmpPicBits() As Byte
Private newPicBits() As Byte
Private PicInfo As BITMAP
Private maxDistColor As Integer
Private minLuminance As Integer
Private minDiffMaxMoy As Integer
Private IntensityFactor As Single
Private Cnt As Long, BytesPerLine As Long

Public Sub getPoints(hwndPict As Long, ByRef RedX() As Long, ByRef RedY() As Long, ByRef BlueX() As Long, ByR
ef BlueY() As Long, ByRef GreenX() As Long, ByRef GreenY() As Long, Optional objMinSize As Integer = 8, Optional
objMaxSize As Integer = 14, Optional distMaxColor As Integer = 70, Optional LuminanceMin As Integer = 40, Optional
 MinColorIntensity As Integer = 30, Optional ColorIntensityFactor As Single = 15, Optional ShowMask As Boolean = F
alse)
    Dim tmp As Long
    Dim tmp2 As Integer
    Dim a As Long
    Dim i As Long
    Dim k As Long
    Dim Luminance As Long

    Dim RedLines() As TacheLine
    Dim BlueLines() As TacheLine
    Dim GreenLines() As TacheLine

    Dim tacheMinSize As Integer
    Dim tacheMaxSize As Integer

    Dim redMax As Integer
    Dim blueMax As Integer
    Dim greenMax As Integer

    Dim factor As Single

    Dim redMoy As Long
    Dim blueMoy As Long
    Dim greenMoy As Long

    Dim LookForRedPix As Boolean
    Dim LookForBluePix As Boolean
    Dim LookForGreenPix As Boolean


    LookForRedPix = True
    LookForBluePix = True
    LookForGreenPix = True

    'defini la taille de la tache mini pour etre considere comme une tache et non un defaut de l'image et les differents
parametres par defaut
    tacheMinSize = objMinSize
    tacheMaxSize = objMinSize
    maxDistColor = distMaxColor
    minLuminance = LuminanceMin
    minDiffMaxMoy = MinColorIntensity
    IntensityFactor = ColorIntensityFactor / 10


    'mange un bmp et le traduit en variable byte exploitable
```

```
GetObject hwndPict, Len(PicInfo), PicInfo
BytesPerLine = (PicInfo.bmWidth * 4 + 3)
ReDim PicBits(1 To BytesPerLine * PicInfo.bmHeight) As Byte
ReDim tmpPicBits(1 To UBound(PicBits)) As Byte
ReDim newPicBits(1 To UBound(PicBits)) As Byte

GetBitmapBits hwndPict, UBound(PicBits), PicBits(1)
tmpPicBits = PicBits

'trouve la moyenne du rouge, le max puis en deduit le niveau de rouge minimum pour etre considere comme rouge
redMax = -100
blueMax = -100
redMoy = 0
blueMoy = 0

For Cnt = 1 To UBound(PicBits) - 2 Step 4

   'luminance
   Luminance = Luminance + CLng(PicBits(Cnt)) + CLng(PicBits(Cnt + 1)) + CLng(PicBits(Cnt + 2))

   'rouge
   tmp = CLng(PicBits(Cnt + 2)) - Int(CLng(PicBits(Cnt + 1)) + CLng(PicBits(Cnt))) / 2
   tmp = tmp - Abs(CLng(PicBits(Cnt + 1)) - CLng(PicBits(Cnt)))
   redMoy = redMoy + tmp
   If tmp > redMax Then redMax = tmp

   'bleu
   tmp = CLng(PicBits(Cnt)) - Int(CLng(PicBits(Cnt + 2)) + CLng(PicBits(Cnt + 1))) / 2
   blueMoy = blueMoy + tmp
   If tmp > blueMax Then
      blueMax = tmp
   End If

   'vert
   If CLng(PicBits(Cnt)) > CLng(PicBits(Cnt + 2)) Then
      tmp = CLng(PicBits(Cnt + 1)) - CLng(PicBits(Cnt))
   Else
      tmp = CLng(PicBits(Cnt + 1)) - CLng(PicBits(Cnt + 2))
   End If
   greenMoy = greenMoy + tmp
   If tmp > greenMax Then
      greenMax = tmp
   End If
Next
Luminance = Luminance / ((UBound(PicBits) - 2) / 4)
redMoy = redMoy / ((UBound(PicBits) - 2) / 4)
blueMoy = blueMoy / ((UBound(PicBits) - 2) / 4)
greenMoy = greenMoy / ((UBound(PicBits) - 2) / 4)
If redMoy = 0 Then redMoy = 1
If blueMoy = 0 Then blueMoy = 1
If greenMoy = 0 Then greenMoy = 1

If redMoy > 30 Or blueMoy > 30 Or greenMoy > 30 Then
   If Luminance > 400 Then
      SimplifyPict PicBits 'simplify                                                    l'image
   End If
End If

factor = (redMax / redMoy) / IntensityFactor 'rouge
If redMax - redMoy < minDiffMaxMoy Then LookForRedPix = False 'pour kil ne trouve rien si l'image est juste
blanche
indiceRougeMinimum = redMoy * factor
```

```
    factor = (blueMax / blueMoy) / (IntensityFactor)'bleu
    If blueMax - blueMoy < minDiffMaxMoy Then LookForBluePix = False 'pour kil ne trouve rien si l'image est juste
blanche
    indiceBleuMinimum = blueMoy * factor

    If (greenMax / greenMoy) < 0 Then
        factor = (greenMax / greenMoy) / (IntensityFactor * 3)'vert
    Else
        factor = (greenMax / greenMoy) / (IntensityFactor - 0.2)'vert
    End If
    If greenMax - greenMoy < minDiffMaxMoy Then LookForGreenPix = False 'pour kil ne trouve rien si l'image est
juste                                                                                                    blanche
    indiceVertMinimum = greenMoy * factor


    If Not (LookForRedPix) And Not (LookForBluePix) And Not (LookForGreenPix) Then Exit Sub

    'detecte toute les lignes continue de point rouge ou bleu et les archives avec les Y , les MinX et MaxX de chacunes
 de ces lignes
    ReDim RedLines(0) As TacheLine
    ReDim BlueLines(0) As TacheLine
    ReDim GreenLines(0) As TacheLine
    For Cnt = 0 To PicInfo.bmHeight - 1
        For i = 0 To PicInfo.bmWidth - 1
            If PixelIsRed((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 1, PicBits) And LookForRedPix Then
                newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 1) = 255
                newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 2) = 255
                newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 3) = 255
                If i < PicInfo.bmWidth - 5 Then
                    a = i
                    tmp = 0
                    Do Until Not (PixelIsRed((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 1, PicBits)) Or a >= PicInfo.bmWidth - 1
                        newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 1) = 0
                        newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 2) = 0
                        newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 3) = 255
                        tmp = tmp + 255
                        a = a + 1
                    Loop
                    If tmp > 255 Then
                        ReDim Preserve RedLines(UBound(RedLines) + 1) As TacheLine
                        RedLines(UBound(RedLines)).Xmin = i
                        RedLines(UBound(RedLines)).Xmax = i + Int(tmp / 255)
                        RedLines(UBound(RedLines)).Y = Cnt + 1
                    End If
                    i = a
                End If
            End If
        Next
    Next

    For Cnt = 0 To PicInfo.bmHeight - 1
        For i = 0 To PicInfo.bmWidth - 1
            If PixelIsBlue((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 1, PicBits) And LookForBluePix Then
                newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 1) = 255
                newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 2) = 255
                newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 3) = 255
                If i < PicInfo.bmWidth - 5 Then
                    a = i
                    tmp = 0
                    Do Until Not (PixelIsBlue((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 1, PicBits)) Or a >= PicInfo.bmWidth - 1
                        newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 1) = 255
                        newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 2) = 0
                        newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 3) = 0
```

```
                tmp = tmp + 255
                a = a + 1
            Loop
            If tmp > 255 Then
                ReDim Preserve BlueLines(UBound(BlueLines) + 1) As TacheLine
                BlueLines(UBound(BlueLines)).Xmin = i
                BlueLines(UBound(BlueLines)).Xmax = i + Int(tmp / 255)
                BlueLines(UBound(BlueLines)).Y = Cnt + 1
            End If
            i = a
        End If
    End If
  Next
Next

For Cnt = 0 To PicInfo.bmHeight - 1
  For i = 0 To PicInfo.bmWidth - 1
    If PixelIsGreen((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 1, PicBits) And LookForGreenPix Then
      newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 1) = 255
      newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 2) = 255
      newPicBits((Cnt * 4 * PicInfo.bmWidth) + (i * 4) + 3) = 255
      If i < PicInfo.bmWidth - 5 Then
        a = i
        tmp = 0
        Do Until Not (PixelIsGreen((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 1, PicBits)) Or a >= PicInfo.bmWidth - 1
          newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 1) = 0
          newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 2) = 255
          newPicBits((Cnt * 4 * PicInfo.bmWidth) + (a * 4) + 3) = 0
          tmp = tmp + 255
          a = a + 1
        Loop
        If tmp > 255 Then
          ReDim Preserve GreenLines(UBound(GreenLines) + 1) As TacheLine
          GreenLines(UBound(GreenLines)).Xmin = i
          GreenLines(UBound(GreenLines)).Xmax = i + Int(tmp / 255)
          GreenLines(UBound(GreenLines)).Y = Cnt + 1
        End If
        i = a
      End If
    End If
  Next
Next

Dim tmpTC As TachesCoords
tmpTC = AnalyseTachesAndDrawCross(RedLines, tmpPicBits, tacheMinSize, tacheMaxSize, RED_CROSS)
ReDim RedX(UBound(tmpTC.X))
ReDim RedY(UBound(tmpTC.X))
RedX = tmpTC.X
RedY = tmpTC.Y
tmpTC = AnalyseTachesAndDrawCross(BlueLines, tmpPicBits, tacheMinSize, tacheMaxSize, BLUE_CROSS)
ReDim BlueX(UBound(tmpTC.X))
ReDim BlueY(UBound(tmpTC.X))
BlueX = tmpTC.X
BlueY = tmpTC.Y
tmpTC = AnalyseTachesAndDrawCross(GreenLines, tmpPicBits, tacheMinSize, tacheMaxSize, GREEN_CROSS)
ReDim GreenX(UBound(tmpTC.X))
ReDim GreenY(UBound(tmpTC.X))
GreenX = tmpTC.X
GreenY = tmpTC.Y

If ShowMask Then
  SetBitmapBits hwndPict, UBound(tmpPicBits), newPicBits(1)
Else
```

```
        SetBitmapBits hwndPict, UBound(tmpPicBits), tmpPicBits(1)
    End If
End Sub

Private Sub drawCross(ByRef pictB() As Byte, X As Long, Y As Long, Color As CrossColors)
On Local Error Resume Next
drawPixel pictB, X, Y, Color
drawPixel pictB, X - 1, Y, Color
drawPixel pictB, X - 2, Y, Color
drawPixel pictB, X - 3, Y, Color
drawPixel pictB, X - 4, Y, Color
drawPixel pictB, X + 1, Y, Color
drawPixel pictB, X + 2, Y, Color
drawPixel pictB, X + 3, Y, Color
drawPixel pictB, X + 4, Y, Color
drawPixel pictB, X, Y - 1, Color
drawPixel pictB, X, Y - 2, Color
drawPixel pictB, X, Y - 3, Color
drawPixel pictB, X, Y - 4, Color
drawPixel pictB, X, Y + 1, Color
drawPixel pictB, X, Y + 2, Color
drawPixel pictB, X, Y + 3, Color
drawPixel pictB, X, Y + 4, Color
End Sub

Private Sub drawPixel(ByRef pictB() As Byte, X As Long, Y As Long, Color As CrossColors)
Select Case Color
    Case RED_CROSS
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 1) = (255 - pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 1)) / 1.2
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 2) = (255 - pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 2)) / 1.2
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 3) = 255
    Case BLUE_CROSS
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 1) = 255
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 2) = (255 - pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 2)) / 1.2
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 3) = (255 - pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 3)) / 1.2
    Case GREEN_CROSS
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 1) = (255 - pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 1)) / 1.2
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 2) = 255
        pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 3) = (255 - pictB((Y * PicInfo.bmWidth * 4) + (X * 4) + 3)) / 1.2
End Select
End Sub

Private Sub deleteTache(ByRef Lines() As TacheLine, ByVal tacheAvant As Integer, ByVal tacheApres As Integer)
Dim i As Integer
For i = 1 To UBound(Lines)
    If Lines(i).Tache = tacheAvant Then Lines(i).Tache = tacheApres
Next
End Sub

Private Function PixelIsRed(Coord As Long, Pict() As Byte) As Boolean
Dim indiceRouge As Integer
Dim diffBleuVert As Integer
Dim Luminance As Integer

indiceRouge = CLng(Pict(Coord + 2)) - (Int(CLng(Pict(Coord + 1)) + CLng(Pict(Coord))) / 2)
diffBleuVert = CLng(Pict(Coord)) - CLng(Pict(Coord + 1))
Luminance = CLng(Pict(Coord)) + CLng(Pict(Coord + 1)) + CLng(Pict(Coord + 2))

If indiceRouge > indiceRougeMinimum And indiceRouge > 5 And Luminance > minLuminance And Abs(diffBleuVert)
< maxDistColor Then
    PixelIsRed = True
End If
End Function
```

```
Private Function PixelIsBlue(Coord As Long, Pict() As Byte) As Boolean
Dim indiceBleu As Integer
Dim diffRougeVert As Integer
Dim Luminance As Integer

indiceBleu = CLng(Pict(Coord)) - (Int(CLng(Pict(Coord + 2)) + CLng(Pict(Coord + 1))) / 2)
diffRougeVert = CLng(Pict(Coord + 2)) - CLng(Pict(Coord + 1))
Luminance = CLng(Pict(Coord)) + CLng(Pict(Coord + 1)) + CLng(Pict(Coord + 2))

If indiceBleu > indiceBleuMinimum And indiceBleu > 5 And Luminance > minLuminance And Abs(diffRougeVert) < m
axDistColor * 1.5 Then PixelIsBlue = True
End Function

Private Function PixelIsGreen(Coord As Long, Pict() As Byte) As Boolean
Dim indiceVert As Integer
Dim MaxOthers As Integer
Dim Luminance As Integer

If CLng(Pict(Coord)) > CLng(Pict(Coord + 2)) Then
   indiceVert = CLng(Pict(Coord + 1)) - CLng(Pict(Coord))
   MaxOthers = CLng(Pict(Coord))
Else
   indiceVert = CLng(Pict(Coord + 1)) - CLng(Pict(Coord + 2))
   MaxOthers = CLng(Pict(Coord + 2))
End If
Luminance = CLng(Pict(Coord)) + CLng(Pict(Coord + 1)) + CLng(Pict(Coord + 2))

If indiceVert > indiceVertMinimum And indiceVert > 5 And Luminance > minLuminance And MaxOthers < 200 Then Pi
xelIsGreen = True
End Function

Private Function AnalyseTachesAndDrawCross(ColorLines() As TacheLine, PictureBits() As Byte, minSize As Integer,
 maxSize As Integer, CrossColor As CrossColors) As TachesCoords
   Dim TachesColor() As Coord
   Dim TachesColorNbr As Integer
   Dim i As Long
   Dim k As Integer
   Dim tmp As Long
   Dim tmp2 As Integer
   ReDim AnalyseTachesAndDrawCross.X(0)
   ReDim AnalyseTachesAndDrawCross.Y(0)

   'deduit si deux lignes sont continu dans la hauteur et appartienne a la meme "tache" et donne un n° de tache a cha
que ligne continue de point de                                                                                  couleur
   ReDim TachesColor(0) As Coord
   For k = 1 To UBound(ColorLines)
      For i = k - 1 To 1 Step -1
         If ColorLines(k).Y = ColorLines(i).Y + 1 Then
            If ColorLines(k).Xmax > ColorLines(i).Xmin And ColorLines(k).Xmin < ColorLines(i).Xmax Then
               ColorLines(k).Tache = ColorLines(i).Tache
               For Cnt = i - 1 To 1 Step -1
                  If ColorLines(k).Xmax > ColorLines(Cnt).Xmin And ColorLines(k).Xmin < ColorLines(Cnt).Xmax And
 ColorLines(k).Y = ColorLines(Cnt).Y + 1 And ColorLines(Cnt).Tache <> ColorLines(k).Tache Then
                     deleteTache ColorLines, ColorLines(Cnt).Tache, ColorLines(k).Tache
                     Exit For
                  End If
               Next
               GoTo suite
            End If
         End If
      Next
      TachesColorNbr = TachesColorNbr + 1
```

```
        ColorLines(k).Tache = TachesColorNbr
suite:
    Next

    'pour chaque TachesColor il fait la somme des X et des Y et icremente le nombre de point (milieu de chaque ligne de rouge continue) qu'elles possedent
    For k = 1 To UBound(ColorLines)
        If UBound(TachesColor) < ColorLines(k).Tache Then
            ReDim Preserve TachesColor(ColorLines(k).Tache)
        End If
        TachesColor(ColorLines(k).Tache).Height = TachesColor(ColorLines(k).Tache).Height + 1
        TachesColor(ColorLines(k).Tache).X = TachesColor(ColorLines(k).Tache).X + ((ColorLines(k).Xmin + ColorLines(k).Xmax) / 2)
        TachesColor(ColorLines(k).Tache).Y = TachesColor(ColorLines(k).Tache).Y + ColorLines(k).Y
        If ColorLines(k).Xmax - ColorLines(k).Xmin > TachesColor(ColorLines(k).Tache).Width Then TachesColor(ColorLines(k).Tache).Width = ColorLines(k).Xmax - ColorLines(k).Xmin
    Next

    'fait la moyenne des X et des Y pour trouver le centre de la TachesColor et verify que la tache fait plus de Xpixels de haut
    tmp2 = 0
    For k = 1 To UBound(TachesColor)
        If TachesColor(k).Height > minSize And TachesColor(k).Width > maxSize Or TachesColor(k).Height > maxSize And TachesColor(k).Width > minSize Then
            tmp2 = tmp2 + 1
            i = Int(TachesColor(k).X / TachesColor(k).Height)
            tmp = Int(TachesColor(k).Y / TachesColor(k).Height)
            drawCross PictureBits, i, tmp, CrossColor
            ReDim Preserve AnalyseTachesAndDrawCross.X(UBound(AnalyseTachesAndDrawCross.X) + 1)
            ReDim Preserve AnalyseTachesAndDrawCross.Y(UBound(AnalyseTachesAndDrawCross.Y) + 1)
            AnalyseTachesAndDrawCross.X(UBound(AnalyseTachesAndDrawCross.X)) = i
            AnalyseTachesAndDrawCross.Y(UBound(AnalyseTachesAndDrawCross.Y)) = tmp
        End If
    Next
End Function

Private Sub SimplifyPict(ByRef PictureBits() As Byte)
Dim Cnt As Long
Dim LumMoy As Long
Dim u As Long
Dim i As Integer

For Cnt = 1 To UBound(PictureBits) - 2 Step 4
    For i = 0 To 2
        LumMoy = LumMoy + PictureBits(Cnt + i)
    Next
    u = u + 1
Next
LumMoy = LumMoy / u / 3
For Cnt = 1 To UBound(PictureBits) - 2 Step 4
    u = 0
    For i = 0 To 2
        If PictureBits(Cnt + i) > LumMoy / 2.5 Then u = u + 255
    Next
    If u = 765 Then
        PictureBits(Cnt) = 0
        PictureBits(Cnt + 1) = 0
        PictureBits(Cnt + 2) = 0
    End If
Next
End Sub
```

5.1.2. Interprétation de l'analyse

Fonction permettant de trouver les angles idéaux en fonction de l'objet et de la position du robot.

Entrée :

CPoint2D	pt1	:	position de la base du robot
Single	r1	:	Taille du premier segment du robot
CPoint2D	pt2	:	position à atteindre
Single	r2	:	Taille du seconde segment du robot

Sortie :

CPoint2D	:	angles pour atteindre la position idéale

```
Public Function Analise_FindAngle(pt1 As CPoint2D, r1 As Single, pt2 As CPoint2D, r2 As Single) As CPoint2D
Dim i As Long
Dim MinDist As Single
Dim MinAngle As Single
Dim TmpDist As Double
MinDist = 0
MinDist = 16000
Dim p3x As Single
Dim p3y As Single
For i = -180 To 0
p3x = Cos(i / 180 * 3.1415) * r1 + pt1.X
p3y = Sin(i / 180 * 3.1415) * r1 + pt1.Y
TmpDist = Abs(Sqr((p3x - pt2.X) * (p3x - pt2.X) + (p3y - pt2.Y) * (p3y - pt2.Y)) - r2)
If TmpDist < MinDist Then
MinDist = TmpDist
MinAngle = i
End If
Next i
p3x = Cos(MinAngle / 180 * 3.1415) * r1 + pt1.X
p3y = Sin(MinAngle / 180 * 3.1415) * r1 + pt1.Y
Analise_FindAngle.X = MinAngle
Analise_FindAngle.Y = InvCos((pt2.X - p3x) / r2) * 180 / 3.1415
End Function
```

5.2. Annexe électronique

Le schéma de la carte électronique est présenté en dessous figure 36.

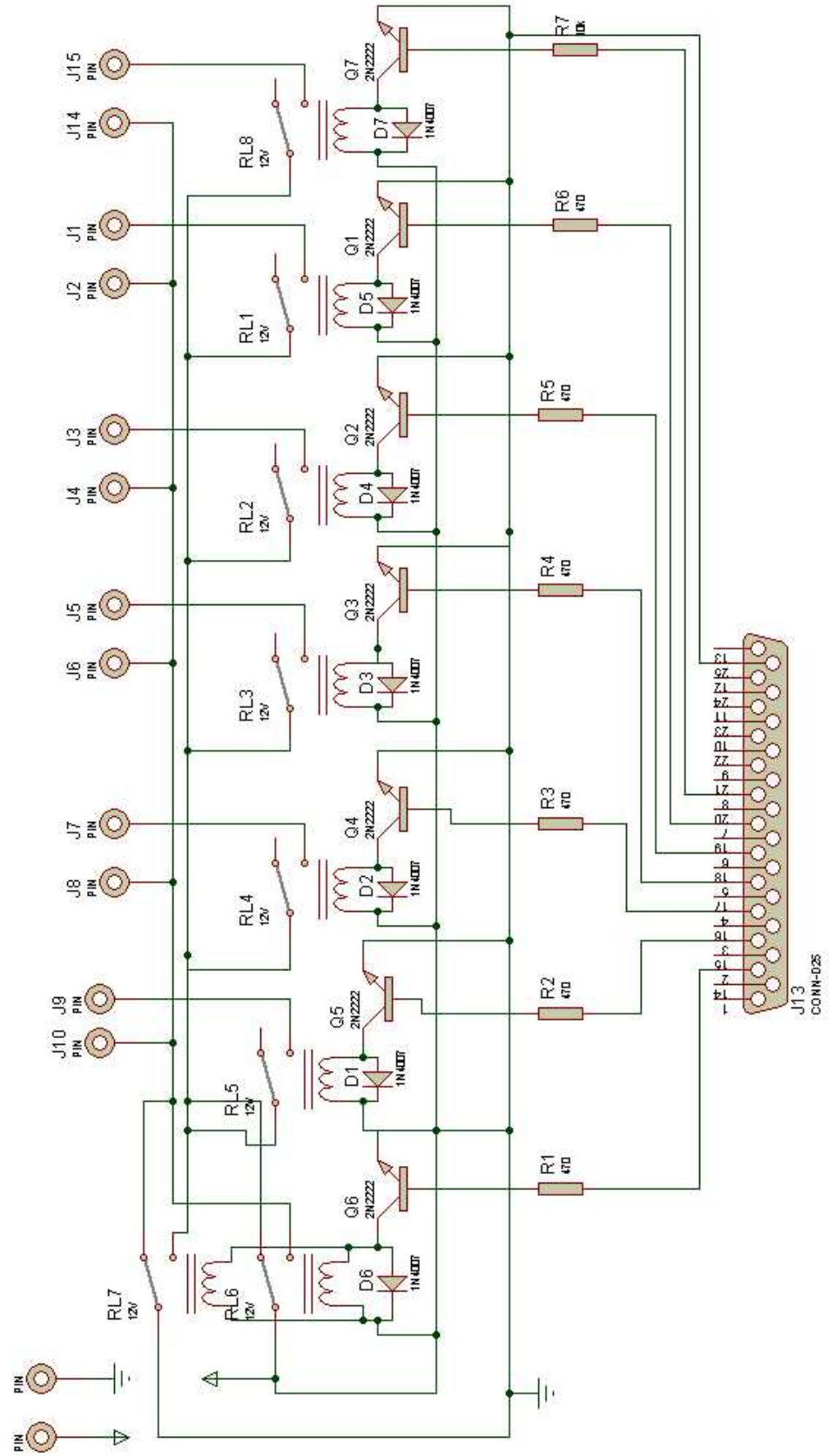


Figure 36 – Schéma de la carte électronique

6. Table des légendes