



Livrable 4 - Robot Kinocito

Design III

présenté à

M. Dominique Grenier, M. Luc Lamontagne et M. Abdelhakim Bendada

<i>matricule</i>	<i>nom</i>
910 058 073	Émile Arsenault
908 190 985	Philippe Bourdages
910 098 468	Pierre-Luc Buhler
998 107 355	Diane Fournier
908 318 388	Olivier Sylvain
910 055 897	Daniel Thibodeau
910 097 879	Francis Valois

Université Laval
21 avril 2013

Table des matières

Table des figures	iii
Liste des tableaux	iv
1 Matrice de vérification des performances	1
2 Vision 2^e itération	3
2.1 Localisation avec la caméra embarquée	3
2.1.1 Localisation	3
2.1.1.1 Transformations	3
2.1.1.2 Localisation des points	4
2.1.2 Orientation et angle par rapport au mur	4
2.2 Orientation avec la caméra embarquée	4
2.3 Extraire les contours d'un sudocube	5
2.3.1 Performance de l'algo	6
2.4 Localisation avec la Kinect	6
2.4.1 Transformation des distances	6
2.4.2 Localisation de la Kinect à l'aide de la calibration	8
2.4.3 Réduction de la distorsion de la lentille infrarouge	9
2.4.3.1 Correction à l'aide d'une transformation de perspective	9
2.4.3.2 Correction à l'aide d'une courbe de correction d'erreur	10
2.4.4 Détection des obstacles	10
2.4.5 Détection du robot	11
3 Schémas électroniques 2^e itération	12
4 Post-mortem	16
4.1 Vision	16
4.1.1 Orientation avec la caméra embarquée	16
4.1.2 Vision par Kinect	16
4.2 Microcontrôleur	17
4.3 Déplacements	18
4.4 Dessins	18

TABLE DES MATIÈRES

ii

4.5 Circuit électriques	18
-----------------------------------	----

Table des figures

2.1	Exemple de segmentation sur le vert du cadre	6
2.2	Exemple de sudocube nettoyé et prêt à être extrait	7
2.3	Exemple de chiffre extrait	7
2.4	Schéma représentant la table et les différentes mesures obtenues avec la Kinect pour un point quelconque	8
2.5	Schéma représentant la table et les vecteurs nécessaires à la mesure de l'angle de la Kinect	9
2.6	Schéma représentant la table et les vecteurs nécessaires à la localisation de la Kinect	10
3.1	Figure présentant le plan de l'alimentation 5V pour les périphériques	13
3.2	Figure présentant le circuit de protection des circuits électriques	13
3.3	Figure présentant le plan de l'alimentation 5V pour les périphériques	14
3.4	Figure présentant une photo de l'alimentation 24V achetée telle quelle	14
3.5	Figure présentant le plan du récepteur Manchester	15
3.6	Figure présentant le plan du circuit de commande du préhenseur	15
3.7	Figure présentant le plan du circuit de commande de la DEL verte	15

Liste des tableaux

Chapitre 1

Matrice de vérification des performances

Ce chapitre présente la matrice de vérification des performances

Fonctionnalité Commentaires	Sous-Fonctionnalité Commentaire	Paramètre critique Commentaire	Méthode de vérification	Spécification	Performance	Marge	Commentaire
Vision numérique	Déetecter les obstacles	Temps de calcul(s)	Test	< 1s	< 200 ms	+0.8 s	
		Précision en x(cm)	Test	< 1 cm	< 2 cm	-1 cm	
		Précision en z(cm)	Test	< 1 cm	< 2 cm	-1 cm	
	Localiser le robot	Temps de calcul(s)	Test	< 1s	< 0.5 s	+0.5s	
		Précision en x(cm)	Test	< 1 cm	< 2 cm	-1 cm	
		Précision en z(cm)	Test	< 1 cm	< 2 cm	-1 cm	
		Précision angulaire(*)	Test	< 10°	< 5°	+5°	
Traitement numérique	Lire le cube	Temps de calcul(s)	Test	< 10 s	< 2 s	+8s	
	Calculer la trajectoire optimale	Temps de calcul(s)	Test	< 2s	< 0.01 s	+0.99 s	
	Contrôler le robot pour le dessin	Temps d'exécution(s)	Test	< 60 s	< 42 s	+0.18 s	
		Précision (cm)	Test	±1 cm de la ligne centrale	±0.8 cm	+0.2cm	
	Décoder le signal d'antenne	Temps d'exécution(s)	Test	< 5s	0.3 ms	+4.7 s	
	Choisir le cube selon le signal d'antenne		Démonstration				Vérification du cube choisi selon le signal d'antenne émis
	Résoudre le sudocube	Temps de calcul(s)	Test	< 10 s	< 0.05 s	+9.95 s	
		Chiffres initiaux minimum	Test	7	4	+3 s	
Communication	Recevoir signal d'antenne		Démonstration				Vérifier la cohérence entre la réception et l'émission du signal d'antenne
	Communiquer entre le robot et la station de base	Vitesse(Mo/s)	Démonstration	> 0.5Mo/s	2Mo/s	+1.5Mo/s	Vérifier que les informations transmises du robot et de la station de base sont respectivement reçues
		Latence(ms)	Analyse	< 3 s	< 1 s	+2 s	
	Communiquer entre le mac mini et le microcontrôleur	Vitesse(bit/s)	Test	> 57600 bit/s	115200 bit/s	+57600 bit/s	
	Commander les moteurs	Précision(cm)	Test	< 2 cm	0.6cm	+1.4cm	
		Temps de réponse (ms)	Analyse et test	< 1000 ms	10ms	+990ms	
	Réception des images de la Kinect	Taux de transfert (Images/s)	Démonstration				Vérifier que les images sont reçues
	Contrôler la position de la caméra	Temps de réponse(s)	Test	< 5s	< 1 s	+ 4 s	
		Précision (*)	Test	< 5°	< 1°	+ 4°	
Déplacement	Se déplacer sans toucher aux obstacle	Précision(cm)	Test	< 1 cm de la ligne de trajectoire	< 0.8 cm	+0.2 cm	
		Vitesse(cm/s)	Test	> 3 cm/s	21.7 cm/s	+18.7 cm/s	
Alimentation	Utiliser Une pile rechargeable	Énergie(Wh)	Inspection	30Wh	60Wh	+30Wh	Lire le manuel d'utilisation
		Courant maximal (A)	Inspection	10A	50A	+40A	Lire le manuel d'utilisation
		Durée de charge(min)	Test	> 10min	30min	+20 min	
	Alimenter les moteurs	Puissance(W)	Test	> 25W	36W	+11W	
	Alimenter l'ordinateur	Puissance(W)	Test	> 75W	150W	+75W	
		Ondulation de tension(mV)	Test	< 200mV	90mV	+110mV	
	Alimenter les différents périphériques	Puissance(W)	Test	> 10W	15W	+5W	
		Ondulation de tension(mV)	Test	< 200mV	50mV	+150mV	
Affichage	Afficher message d'initialisation de la tâche		Démonstration				Vérifier que le message est bien affiché sur l'écran de base
	Afficher le cube résolu ainsi que la case rouge		Démonstration				Vérifier que le cube résolu et la case rouge sont bien affichés
	Allumer la DEL lorsque tâche complétée		Test				
	Afficher la trajectoire optimale calculée		Démonstration				Vérifier que la trajectoire optimale calculée est bien affichée
	Afficher la position et trajectoire réelle	Temps d'actualisation	Démonstration	< 15 s	< 4 s	+11s	Vérifier le temps de rafraîchissement de l'affichage de la position en temps réel
	Afficher message de fin		Démonstration				Vérifier que le message de fin est affiché
	Afficher sur le LCD		Démonstration				Vérifier que le LCD affiche bien les informations

Chapitre 2

Vision 2^e itération

2.1 Localisation avec la caméra embarquée

La localisation avec la caméra embarquée est un système de localisation d'appoint dans le projet Kinocto. La classe Localisation doit recevoir une image et une orientation grossière selon le nord, sud, est ou ouest par rapport à la table. Elle doit ensuite être initialisée en fournissant un fichier .xml contenant les paramètres de calibration. Ces paramètres comprennent les paramètres intrinsèques de la caméra obtenus lors de la calibration, les paramètres liés à la distorsion de l'image par la lentille, les paramètres extrinsèques qui permettent de lier les points des images prises par la caméra à leur position par rapport à un système de référence virtuel ainsi que les paramètres qui permettent de recadrer ces coordonnées par rapport au centre du robot.

Avec l'image et les paramètres provenant de la calibration, les méthodes de localisation, de détermination de l'orientation et de mesure de l'angle par rapport au mur peuvent être utilisées.

2.1.1 Localisation

2.1.1.1 Transformations

Pour effectuer la localisation du robot, il faut trouver dans l'image reçue au moins deux points identifiables dont les coordonnées par rapport à la table sont connues. Dans le projet Kinocto, ces points incluent les coins de la table, les coins inférieurs des blocs de couleur, les coins du carré vert dessiné sur la table. Pour passer du système de coordonnées du robot à celui de la table, il faut résoudre le système d'équations suivant, où les points P_{1A} et P_{2A} sont les points dans le système de la table, P_{1R} et P_{2R} sont les mêmes points dans le système du robot et t_X et t_Y sont les coordonnées du robot dans le système de la table :

$$X_{1A} = X_{1R}\cos(\theta) - Y_{1R}\sin(\theta) + t_X \quad (2.1)$$

$$Y_{1A} = X_{1R}\sin(\theta) + Y_{1R}\cos(\theta) + t_Y \quad (2.2)$$

$$X_{2A} = X_{2R}\cos\theta - Y_{2R}\sin(\theta) + t_X \quad (2.3)$$

$$Y_{2A} = X_{2R}\sin(\theta) + Y_{2R}\cos(\theta) + t_Y \quad (2.4)$$

En soustrayant l'équation 2.3 de l'équation 2.1 ainsi que l'équation 2.4 de l'équation 2.2, on élimine t_X et t_Y et on se retrouve avec deux équations que l'on peut combiner pour obtenir une équation à une inconnue, θ . Il s'agit d'une équation de la forme $a\cos(\theta) + b\sin(\theta) = c$. Ce type d'équation peut être résolu en considérant la relation suivante :

$$a\cos(\theta) + b\sin(\theta) = R\cos(\theta - \alpha) \quad (2.5)$$

Où $R = \sqrt{a^2 + b^2}$ et $\tan(\alpha) = \frac{b}{a}$. En isolant θ , on obtient donc :

$$\theta = \cos^{-1}\left(\frac{c}{\sqrt{a^2 + b^2}}\right) + \tan^{-1}\left(\frac{b}{a}\right) \quad (2.6)$$

Il suffit ensuite d'utiliser θ dans deux des équations initiales pour retrouver les coordonnées du robot selon le système de référence de la table t_X et t_Y .

2.1.1.2 Localisation des points

1. Coins de table

Les points correspondant aux coins de table sont trouvés dans l'image en effectuant d'abord une segmentation sur le noir et en utilisant ensuite l'algorithme des lignes de Hough pour retrouver les lignes de bas de mur. L'intersection entre les lignes de bas de mur constitue le coin de la table. Un intérêt de cette méthode est qu'elle permet de retrouver les coins de table même lorsqu'un obstacle se trouve devant celui-ci.

2. Coins du bas des blocs de couleur

Les coins du bas des blocs de couleurs peuvent être retrouvés en trouvant les intersections entre les lignes de bas de mur et la ligne du bas du bloc de couleur, trouvée après segmentation sur le bleu et l'orange.

3. Coins du carré vert

Les coins internes du carré vert peuvent être retrouvés en segmentant sur le vert et en retrouvant les intersections entre les lignes avec la méthode décrite plus haut.

2.1.2 Orientation et angle par rapport au mur

L'angle par rapport au mur peut être utile pour réorienter le robot pour la lecture des sudokus. Il est facilement obtenu en utilisant la ligne de bas de mur détectée précédemment. L'orientation peut être déduite en utilisant cet angle et l'orientation fournie à l'initialisation.

2.2 Orientation avec la caméra embarquée

Après quelques essais avec la matrice de transformation, nous n'avons pas réussi à obtenir une localisation fiable avec la caméra embarquée.

L'orientation avec la caméra embarquée a donc servi de méthode d'appoint à la localisation avec la Kinect pour corriger l'angle du robot à trois moments critiques, soit devant le mur

lors de la lecture du sudocube, dans le carré vert avant le départ vers les sudocubes et dans le carré vert avant de commencer le dessin.

L'angle est évalué en traçant des lignes de Hough sur des images segmentées sur le noir pour les murs et sur le vert pour le carré vert. La ligne pertinente est sélectionnée en trouvant la ligne croisant l'axe des y à la valeur la plus élevée et ayant la pente la plus faible.

L'angle trouvé n'est pas l'angle réel du robot avec la ligne, mais un angle de 0° signifie que l'orientation du robot est correcte. L'erreur devient plus importante avec les grands angles, mais l'erreur d'orientation ne dépasse normalement pas 15°. En appliquant la correction 2 fois, l'orientation du robot est corrigée dans la plupart des cas.

2.3 Extraire les contours d'un sudocube

Au tout début, l'image est nettoyé de son bruit grâce a un flou gaussien.

La première étape d'extraction consiste à segmenter par couleur verte en convertissant l'image en HSV et en appliquant la méthode `inRange()` afin d'isoler le cadre vert du sudocube. Le contours du cadre est nettoyé avec l'application d'opérateur morphologique tel que la dilatation et l'érosion. Puis, les algorithmes d'opencv `findContours()` et `approxPolyDP()` sont appliqués pour trouver deux polygones rectangulaires. Ces polygones sont les bordures intérieures et extérieures du cadre. Enfin, on vérifie que l'aire des rectangles est plus grande que 200 000 pixels afin d'être sûr que ceux-ci sont assez grands pour être ceux du cadre du sudocube. Tous les polygones qui sont en bas de cette valeur sont soit des cadres vert des sudocubes adjacents qui n'occupent pas une grande partie de l'image ou des artefacts dû à la segmentation qui ne nous intéressent pas.

Ensuite, à partir du polygone intérieur du cadre on isole une sous-région de l'image principale que l'on recadre afin qu'elle ai 500 pixels de côté. L'image obtenue est convertie en noir et blanc avec la méthode `threshold()` pour appliquer le même algorithme de contour mentionné dans le premier paragraphe afin de trouver les polygones des cases du sudocube. Une dilatation et une érosion sont appliquées afin d'améliorer les contours des cases. Ensuite, on sélectionne tous les polygones qui ont une aire entre 150 et 700 pixels. Si jamais le nombre de polygones sélectionné n'est pas de 47, on réapplique la méthode `threshold()` avec un seuil de tolérance différent pour l'algorithme de contour et ce jusqu'à ce qu'on obtienne le bon seuil pour avoir 47 cases.

Le polygone de la case rouge est obtenu en segmentant par couleur rouge et en appliquant à nouveau l'algorithme de contour mentionné ci-haut.

Finalement, les images des cases sont triées en fonction de leur position en X et en Y dans l'image du sudocube et un algorithme de reconnaissance des caractères appliqué sur toutes les cases afin d'identifier les numéros. L'algorithme de reconnaissance utilise la technique KNearest. Les chiffres trouvés sont ajoutés dans la structure de données du sudocube et la position de la case rouge est spécifiée.

2.3.1 Performance de l'algo

L'algorithme prend en moyenne 250 ms afin de s'exécuter et réussi à extraire des sudocubes quand la caméra prend des photos entre 20 à 45 cm devant le mur avec un angle maximale de +-20 degrés de chaque côté. Sur 80 essais manuel un seul essai comportait un chiffre mal lu.

2.4 Localisation avec la Kinect

Pour aider au déplacement du robot, la Kinect a été utilisée afin de détecter la position des obstacles, la position du robot lui-même ainsi que sa position angulaire par rapport au point (0,0) de notre représentation cartésienne de la table. Toutefois, comme la Kinect est située à l'extérieur de la table de jeu, il faut une translation et une rotation des données obtenues afin de positionner les objets avec exactitude. Les 4 sections qui suivent expliquent en détail les différentes parties de l'algorithme de vision de la Kinect.

2.4.1 Transformation des distances

En premier lieu, il est important de clarifier quelles distances il est possible d'obtenir à l'aide de la Kinect. Le «Framework» OpenNI est capable de retourner 2 types de distances pour chacun des points dans l'image infrarouge obtenue de la Kinect. Le premier type de distances retourné est la longueur, en mètres, entre l'objectif et un point quelconque sur l'image. Le second type est dérivé du premier et correspond aux composantes X, Y et Z du vecteur de longueur entre l'objectif et la cible. Pour aider à la compréhension, les deux types de distances peuvent être représentés par les lignes vertes sur l'image 2.4. Toutefois, comme l'origine de notre représentation cartésienne de la table est située sur le coin inférieur droit de la table et que la Kinect n'est pas située à cet endroit, il est nécessaire d'obtenir la distance X et Y (lignes bleues sur l'image 2.4) entre l'origine et le point ciblé sur l'image obtenue de la Kinect. Pour y arriver, il suffit d'obtenir la position X et Y (lignes rouges sur l'image 2.4) de la lentille infrarouge de la Kinect ainsi que l'angle de la Kinect avec l'axe X de la table. Pour obtenir ces valeurs, nous utilisons une calibration simple présentée à la section 2.4.2. Une fois la calibration effectuée, il est possible d'effectuer la transformation des distances de la Kinect vers les composantes X et Y recherchées à l'aide de règles de trigonométrie simples (Translation et Rotation). Toutefois, la Kinect possède une erreur de mesure qui augmente avec la distance. C'est pourquoi l'algorithme va toujours posséder une incertitude entre 1 et 3 cm pour chacun des points mesurés.

2.4.2 Localisation de la Kinect à l'aide de la calibration

Le but de la calibration est d'obtenir l'angle de la Kinect avec l'axe X de la table ainsi que la position de son objectif en XY par rapport au coin inférieur droit de la zone de jeu qui constitue notre point (0,0). Pour y arriver, un système, qui possède des dimensions connues, va positionner une plaque constituée d'un "Chessboard" sur la table à un endroit prédéterminé

(ligne noire sur la table dans l'image 2.5). Par la suite, une image RGB de la table est capturée à l'aide de la Kinect et les 2 points les plus éloignés horizontalement sur le "Chessboard" sont localisés. Ces 2 points sont reportés sur l'image de distances correspondante pour obtenir la distance XYZ de chacun des 2 points (lignes vertes sur l'image 2.5). À l'aide de ces deux distances, les dimensions du petit triangle entre les deux points de la plaque de calibration sont trouvées (le triangle se situe en dessous de la plaque de calibration sur l'image 2.5). En ayant les dimensions du triangle, il est possible de trouver l'angle de la Kinect, car celui-ci est le même que l'angle interne droit du triangle. Il suffit d'utiliser l'équation suivante avec les dimensions obtenues précédemment :

$$\text{Angle} = \arctan \left(\frac{\text{oppose}}{\text{adjacent}} \right) \quad (2.7)$$

Pour obtenir la position de la Kinect, il est nécessaire de connaître la position réelle, par rapport au point (0.0), des 2 points utilisés pour la mesure de l'angle. En effectuant la rotation des distances de chacun, des points donnés par la Kinect pour ramener les mesures dans le plan XY de la table, les distances XY entre les points et l'objectif sont obtenus (lignes rouges pleines sur l'image 2.6). En soustrayant les distances précédemment obtenues aux distances réelles mesurées (lignes bleues sur l'image 2.6), la position de la Kinect est obtenue (lignes rouges pointillées sur l'image 2.6).

2.4.3 Réduction de la distorsion de la lentille infrarouge

Dû à la distorsion présente dans la lentille de la caméra infrarouge de la Kinect, il a été nécessaire d'utiliser 2 algorithmes pour réduire la distorsion et améliorer la précision des distances obtenues.

2.4.3.1 Correction à l'aide d'une transformation de perspective

Le framework OpenCV possède une fonction de transformation de perspective qui se nomme "getPerspectiveTransform". Celle-ci reçoit en entrée une matrice de 4 points de distance selon un axe ainsi que les 4 mêmes points selon un autre axe. Par la suite, la fonction va produire une matrice de taille 3x3 pouvant être appliquée à n'importe quelle mesure qui doit passer d'un domaine à l'autre. Dans le cas du projet, le premier domaine est celui des distances retournées par la Kinect et le second est celui des distances réelles mesurées sur la table. Pour corriger la distorsion, il suffit alors de multiplier la matrice produite 3x3 par une matrice 3x1 composé des distances en x et z ainsi que du chiffre 1. Le résultat va être composé d'une mesure en x et z ainsi qu'un d'un facteur correctif devant être appliqué aux 2 mesures associées pour appliquer la correction. Ainsi, grâce à cette méthode, la précision augmente beaucoup en périphérie de l'image de points où la distorsion était très importante et pouvait causer des erreurs allant jusqu'à 8 centimètres.

2.4.3.2 Correction à l'aide d'une courbe de correction d'erreur

Le principal problème de la méthode précédente provient de son très petit nombre d'échantillons. Avec seulement 4 points, la distorsion ne peut être corrigée qu'aux alentours des points mesurés et il devient nécessaire de trouver une meilleure méthode pour corriger sur toute la plage de distances prévues. Comme la distorsion dépend de la distance avec l'axe optique, une courbe d'erreur de niveau 3 a été mise en place. Ainsi 16 différentes mesures situées un peu partout sur la table ont été effectuées à l'aide de la Kinect. Avec ces mesures, l'erreur de chacun des points avec la mesure réelle a été comptabilisée. Par la suite, une courbe d'interpolation a été créée pour relier les différents points d'erreurs et permettre une correction efficace de la distorsion en l'appliquant sur chacune des mesures effectuées.

2.4.4 Détection des obstacles

Comme il est possible d'obtenir n'importe quelle distance entre l'origine et un point quelconque sur l'image, un algorithme de recherche d'obstacles a été créé. Sachant que les obstacles sont situés dans une zone précise sur la table et que ceux-ci font 40cm de hauteur, il suffit de rechercher tout objet de cette hauteur situé dans la zone pré définie. La Kinect retourne la hauteur de chacun des points sur l'image infrarouge et permet de localiser les obstacles. De plus, à l'aide de calculs statistiques, l'algorithme est en mesure de trouver les obstacles lorsqu'ils sont presque alignés ou obstrués par un autre objet comme le robot. Comme cet algorithme dépend entièrement de l'algorithme de transformation des distances, les positions obtenues pour chacun des obstacles possèdent la même incertitude de 1 à 3 cm sur chaque mesure de distance effectuée. Pour améliorer la validité des mesures obtenues, une moyenne sur 3 images de distance est effectuée. En ce qui concerne l'efficacité de l'algorithme, différents tests montrent un temps de calcul d'environ 30 ms sur un Core 2 Duo 2.4 GHz pour obtenir la position du centre des deux obstacles.

2.4.5 Détection du robot

En ce qui concerne la détection du robot, un algorithme semblable à la détection des obstacles a été utilisé. Comme le robot possède une hauteur d'environ 20cm et une profondeur semblable, il est possible d'isoler, dans la matrice de distance, un carré qui correspond aux dimensions du robot. En plus de cette méthode, 2 "Chessboard" entourés d'un cadre bleu foncé sont placés sur 2 faces opposées du robot. En recherchant les carrés ainsi que les cadres bleus sur le robot, il est possible de déterminer sa position comme l'algorithme précédent, mais en plus d'obtenir son angle avec l'axe X de la table. Pour contrer l'erreur de position intrinsèque causée par la Kinect, la méthode de moyenne des distances utilisée pour la détection des obstacles est aussi appliquée à la détection du robot. Le temps de calcul est un peu plus élevé que la détection du robot et se situe aux alentours de 50 à 60 ms et il est possible d'obtenir une précision de moins de 2 centimètres dans le carré vert de la table. La précision à l'extérieur de celui-ci n'est pas nécessaire, car le robot se repositionne à l'aide de la Kinect seulement lors de la réception du signal Manchester et avant un dessin.

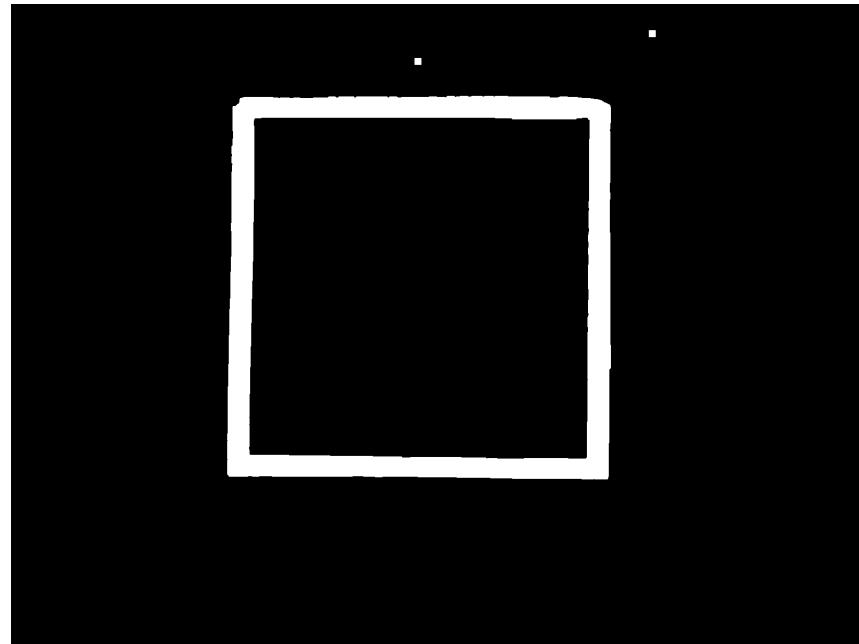


Figure 2.1 – Exemple de segmentation sur le vert du cadre

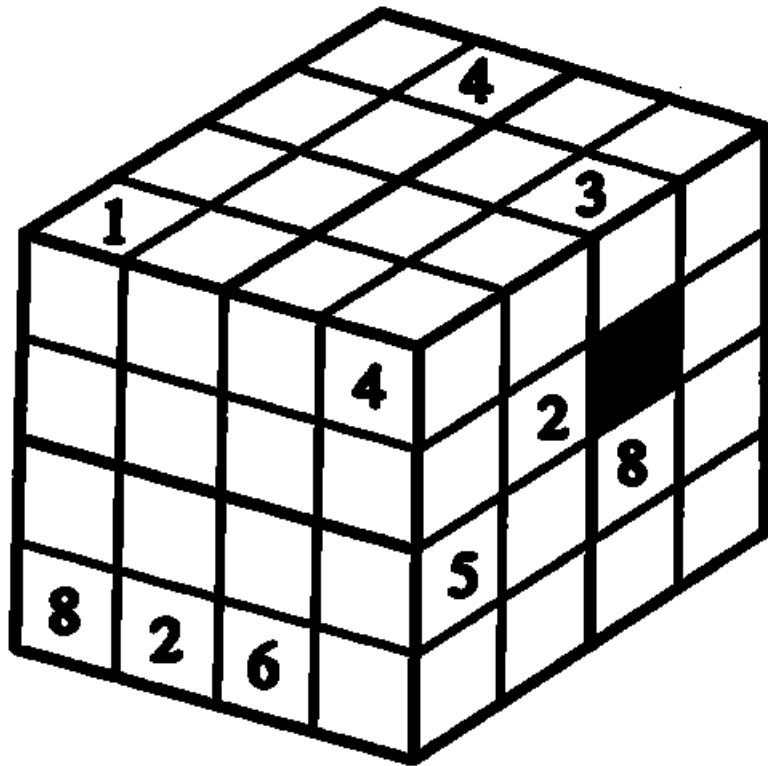


Figure 2.2 – Exemple de sudocube nettoyé et prêt à être extrait



Figure 2.3 – Exemple de chiffre extrait

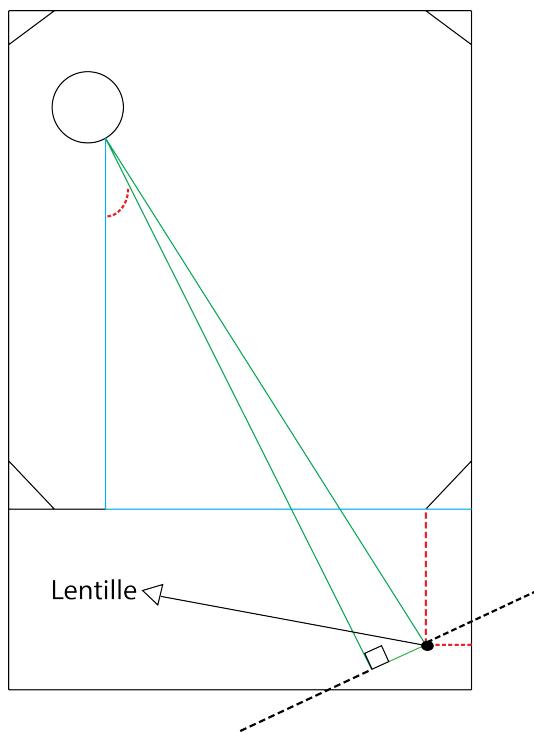


Figure 2.4 – Schéma représentant la table et les différentes mesures obtenues avec la Kinect pour un point quelconque

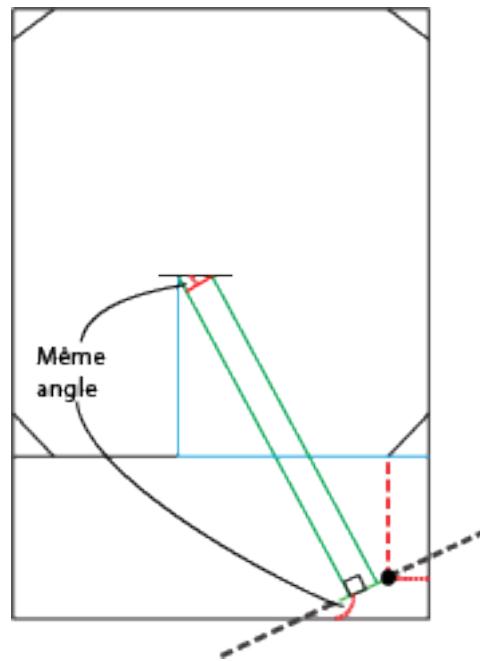


Figure 2.5 – Schéma représentant la table et les vecteurs nécessaires à la mesure de l'angle de la Kinect

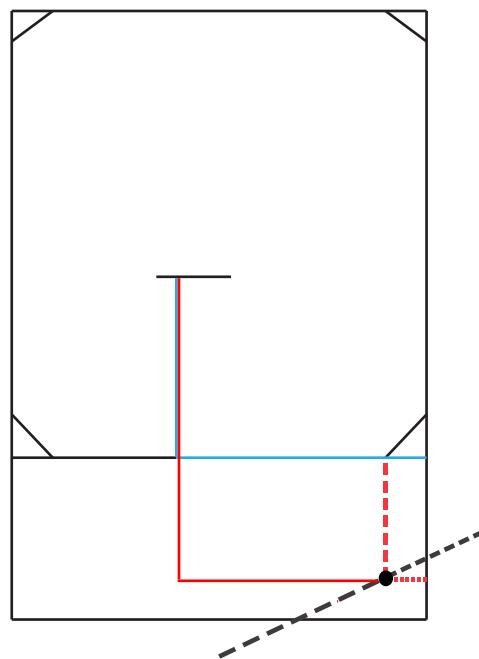


Figure 2.6 – Schéma représentant la table et les vecteurs nécessaires à la localisation de la Kinect

Chapitre 3

Schémas électroniques 2^e itération

Le modèle de microcontrôleur employé est un Stellaris LM3S9B92. Les différentes entrées et sorties ont été fixées sur un circuit imprimé (PCB) dont le schéma est présenté à la figure 3.1.

Tous les circuits électriques sont placés derrière un circuit de protection utilisant des fusibles dont le schéma est présenté à la figure 3.2. Ce circuit est prévu pour limiter les dégâts lors d'erreurs de manipulation ou de défauts électroniques. Par ailleurs, ces circuits sont actionnables par des interrupteurs placés devant les circuits respectifs.

L'alimentation 5V des périphériques est réalisée au moyen d'un circuit Buck 5V, le schéma est présenté à la figure 3.3.

L'alimentation 24V du préhenseur et du Mac mini est réalisée au moyen d'un circuit Boost acheté chez un détaillant, pour lequel on ne dispose pas du plan de circuit en raison du secret d'ingénierie du fournisseur. La tension de sortie est variable avec un potentiomètre. La tension d'entrée est directement celle de la batterie. Une photo du dispositif est disponible à la figure 3.4.

Le circuit de démodulation du signal Manchester émis sur les tables est réalisé au moyen du circuit présenté à la figure 3.5. La clé de ce circuit est le LM567 qui est un circuit intégré permettant de syntoniser le signal d'antenne selon une largeur de bande que l'on établit au moyen des capacités présentes sur les différentes pines du LM567. Le LM567 permet d'isoler directement l'enveloppe du signal. Un comparateur à hystérésis est placé à la sortie du LM567 de manière à rendre plus uniformes et abruptes les transitions dans le signal d'enveloppe.

Le circuit de commande du préhenseur est présenté à la figure 3.6. Il n'est composé que d'un transistor Mosfet qui permet d'activer la conduction du préhenseur (avec un courant proche de 150mA), directement à partir du microcontrôleur. Une diode de roue libre permet de décharger l'énergie de la bobine (préhenseur) lors des transitions.

Le circuit de commande de la DEL est présenté à la figure 3.7. Il n'est composé que d'un transistor Mosfet qui permet d'activer la conduction de la DEL, une résistance de 100Ω permet de limiter le courant dans la DEL.

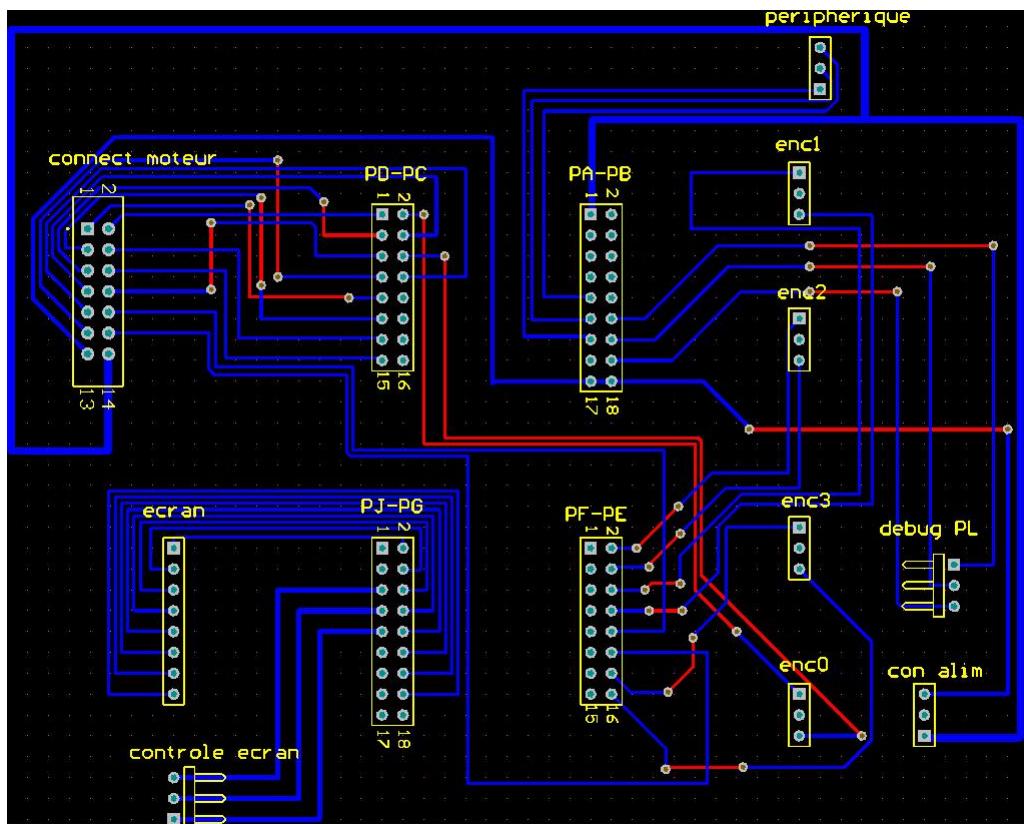


Figure 3.1 – Figure présentant le plan de l'alimentation 5V pour les périphériques

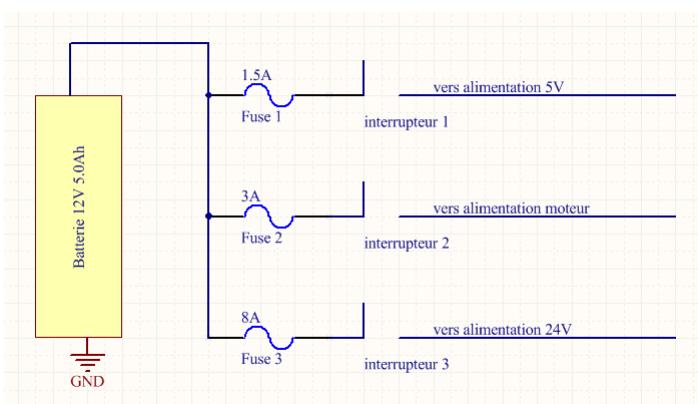


Figure 3.2 – Figure présentant le circuit de protection des circuits électriques

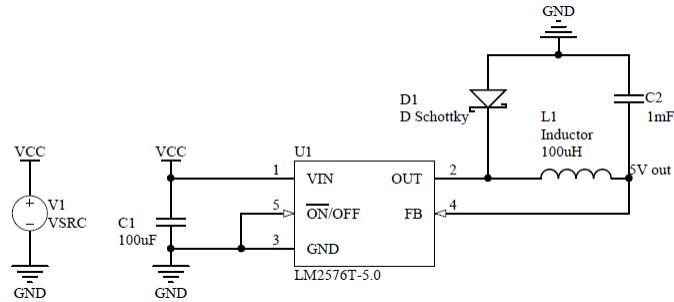


Figure 3.3 – Figure présentant le plan de l'alimentation 5V pour les périphériques

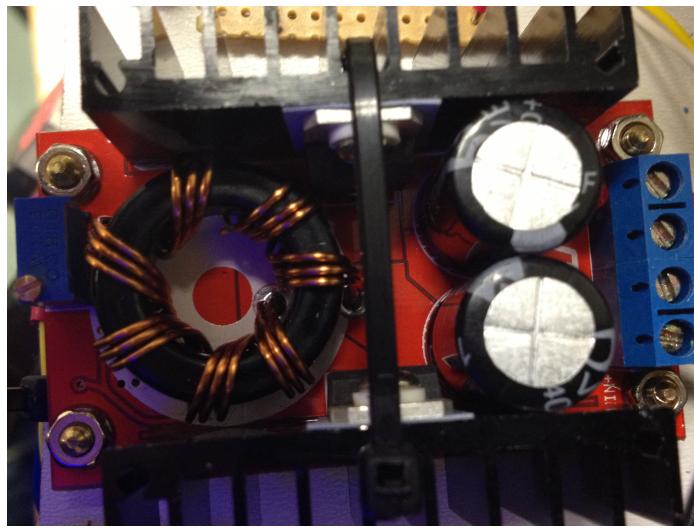


Figure 3.4 – Figure présentant une photo de l'alimentation 24V achetée telle quelle

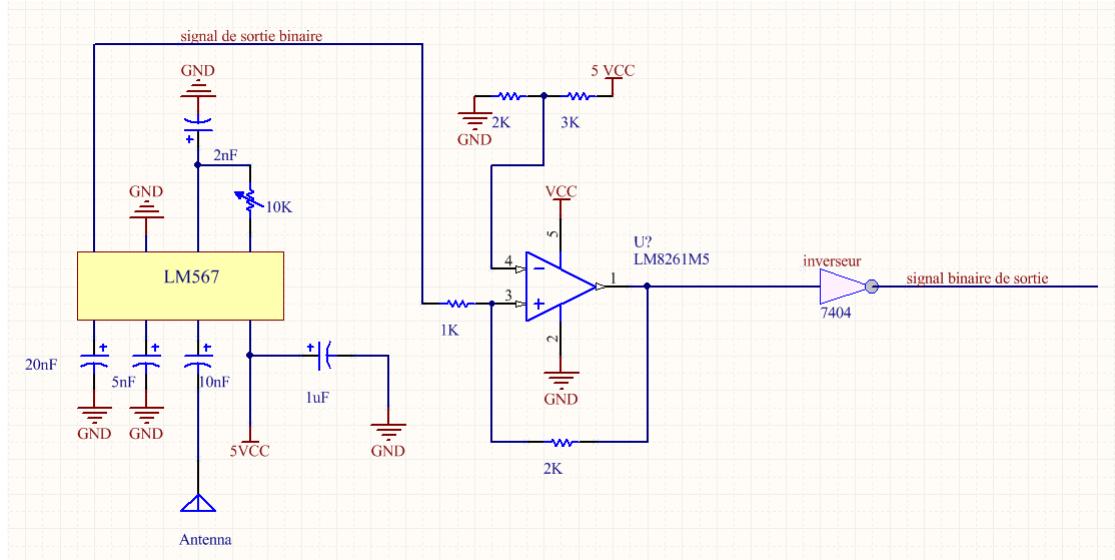


Figure 3.5 – Figure présentant le plan du récepteur Manchester

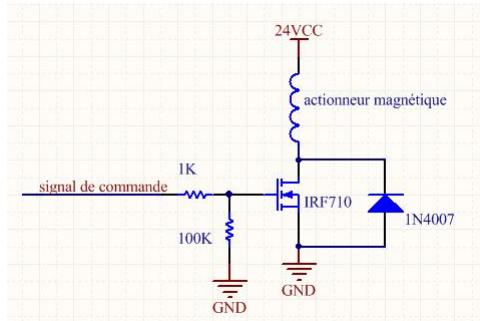


Figure 3.6 – Figure présentant le plan du circuit de commande du préhenseur

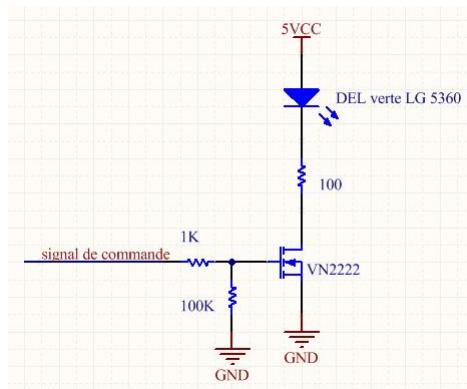


Figure 3.7 – Figure présentant le plan du circuit de commande de la DEL verte

Chapitre 4

Post-mortem

4.1 Vision

4.1.1 Orientation avec la caméra embarquée

L'utilisation de la matrice de transformation pour la localisation exige une matrice extrinsèque la plus parfaite possible. Avec plus de temps, il aurait peut-être été possible d'obtenir une bonne matrice et de la tester adéquatement. Il existe aussi une fonction dans opencv, `reprojectImageTo3D` pour la localisation des objets dans un espace 3D avec deux images prises avec deux caméras ou deux angles différents qui auraient pu être utilisées avec plus de temps et une calibration adéquate.

Une autre méthode qui aurait pu être envisagée est une look-up table constituée en prenant une image d'une grille avec correspondance entre des points dont les coordonnées par rapport au robot sont connues. La transformation de la position du pixel dans l'image vers leurs coordonnées par rapport au robot aurait été faite en localisant les valeurs auxquelles elles correspondent dans la table et en effectuant une interpolation linéaire. Cette méthode aurait eu l'avantage d'être plus facilement et plus intuitivement testable que celles utilisant des matrices.

La localisation avec la caméra embarquée aurait potentiellement été plus précise que celle avec la Kinect, et particulièrement d'obtenir la position du robot lorsqu'il n'est pas possible de le faire avec la Kinect, ce qui survient surtout dans les positions les plus éloignées.

La fiabilité de la méthode de localisation des points à l'intersection des Hough lines aurait également dû être améliorée pour pouvoir l'utiliser dans un algorithme de localisation. En ajustant la saturation des images prises et la segmentation, il était possible d'avoir la position des coins inférieurs des blocs de couleur dans la plupart des cas, mais il arrivait souvent que le positionnement s'écarte beaucoup de la position réelle à cause de l'erreur sur les droites retenues.

4.1.2 Vision par Kinect

La vision à l'aide de la Kinect a été une des parties les plus ardues tout au long du projet. Sachant que la technologie est relativement nouvelle, la documentation à son sujet est très restreinte et souvent incomplète. Il a donc fallu effectuer plusieurs tests et développer nos propres algorithmes pour arriver à quelque chose de fonctionnel. De plus, les frameworks ne semblent pas encore tout à fait matures pour en faire une utilisation simple. Nous avons qu'à penser à la distorsion de la caméra infrarouge qui a causé d'énorme problème lors des tests

de nos algorithmes de transformation de distances. Cette distorsion pourrait être simplement corrigée au travers du framework et la documentation pourrait en faire mention. Une autre idée possible pour améliorer la précision est l'utilisation d'une lookup table de correction en XZ. En effectuant la même méthodologie que pour la courbe de correction en X présenté à la section 2.4.3.2, mais à plus grande échelle, soit plus de 50 points partout sur la table, il serait possible d'obtenir une très bonne correction.

Toutefois, outre le problème majeur de distorsion, la Kinect permet d'obtenir facilement la distance de tout objet dans son champ de vision avec une certaine précision. Ainsi, à l'aide d'algorithme de détection et de statistique, il a été fort simple d'obtenir la position en 3D des obstacles et du robot sur la table. Comme la Kinect effectue tout le traitement de conversion IR-Distance, nos algorithmes sont très efficaces en temps d'exécution. Ainsi, pour la détection des obstacles, l'algorithme prend entre 30 et 60 ms ce qui est très rapide pour une telle détection. Du côté de la détection du robot, il a été nécessaire d'utiliser la caméra RGB en plus de la caméra infrarouge, car le robot est presque aussi haut que les murs de la table. Ainsi, il devient ardu d'extraire le robot du nuage de point lorsque celui-ci est proche d'un mur ou bien des obstacles. La principale difficulté de l'utilisation de la caméra RGB est de savoir ce que nous devons ignorer et ce que nous devons garder. Dans la première version de la détection du cadre bleu sur le robot, dès que quelque chose de bleu entrait dans le champ de vision de la caméra, l'algorithme n'était plus apte à trouver la position du robot et retournait des valeurs aberrantes. Dans la deuxième version, chacune des zones bleues trouvées par l'algorithme est comparée aux distances de la Kinect. Ainsi, lorsque les zones bleues sont situées à l'extérieur de la table, il suffit de les ignorer. Cette composition IR-RGB est vraiment intéressante. Toutefois, en se servant de la caméra RGB, il faut ajouter un temps considérable de traitement pour trouver le cadre bleu ainsi que les différents petits carrés, car la détection n'est pas effectuée directement par la Kinect comme la conversion IR-Distance. Ce traitement de l'image ajoute environ 200ms de plus aux 30 à 60ms de détection de forme dans le nuage de point. Ce temps d'exécution est toutefois en déca de la seconde et est donc extrêmement rapide et adapté à la mise à jour en temps réel de la position du robot.

En écartant la précision imparfaite de notre implantation qui ne demande qu'à être améliorée, le placement des cadres bleus oblige le robot à se placer d'une certaine façon pour que celui-ci soit détecté. Une amélioration possible aurait été de placer des carrés sur les faces non détectées pour permettre la détection à 360 degrés, mais il aurait été nécessaire de créer un algorithme de choix de carrés pour en choisir 1 seul, car dans la plupart des cas, deux carrés auraient été visibles.

4.2 Microcontrôleur

Dans le but de permettre à plusieurs membres de l'équipe de s'impliquer dans la partie microcontrôleur, il aurait fallu rendre le code plus lisible, plus simple et ajouter des commentaires. Bref, de clarifier celui-ci. Un exemple de clarification aurait été de diviser le code en plusieurs fichiers ".h" plutôt que d'utiliser seulement des fichiers sources ".c" et de supprimer les déclarations multiples "extern" de plusieurs variables globales. Excepté ce point, il est

difficile de trouver des améliorations dans cette partie.

4.3 Déplacements

Dans l'ensemble, le robot a un asservissement linéaire en vitesse qui dépasse largement les autres équipes en termes de temps de réponse et de stabilité directionnelle. La stratégie de déplacement aurait pu être optimisée avec l'utilisation des déplacements en diagonal. Comme l'asservissement était en vitesse, nous aurions eu à intégrer des plages de décélérations calibrées selon les plages d'asservissements. Afin d'assurer l'unicité de l'ensemble, d'autres calibres d'asservissements auraient dû être ajoutés. Finalement, il aurait également fallu intégrer un asservissement en position pour les rotations. La méthode implantée pour les rotations n'est pas un asservissement, seulement une correction de l'angle du robot si la consigne de position est dépassée. Pour avoir une meilleure précision et une meilleure vitesse de rotation, il aurait fallu planter cette méthode.

4.4 Dessins

Afin de compenser les disparités de constitutions des roues de notre robot, qui avaient énormément d'impact sur nos performances, un asservissement en position utilisant la vision aurait pu être implanté afin de compenser les erreurs. Pour se faire, nous aurions pu centrer le préhenseur et effectuer des rotations lors des croisements afin de toujours détecter le cadre vert. L'implantation de l'asservissement de déplacement pour les diagonales aurait pu être employé à cet effet, jumelé à la détection du cadre vert, qui aurait du être adaptée pour un robot qui se déplace. Évidemment, le temps requis de conception est particulièrement énorme et un tel ajustement aurait requis plusieurs semaines.

4.5 Circuit électriques

Afin de permettre une plus grande compacité et une clarté des circuits électriques, nous aurions pu planter l'ensemble de ceux-ci sur un même PCB. De cette façon, nous aurions pu éliminer tous les fils électriques, ou presque, qui alimentaient et permettaient la communication des divers modules électronique entre eux. Étant donné que le préhenseur est alimenté à 24 volts tout comme le Mac-mini, il aurait été pratique d'installer un interrupteur pour couper simplement l'alimentation du Mac-mini tout en continuant d'alimenter le préhenseur. Cette petite modification du circuit aurait facilité les tests de dessin lorsqu'on ne désirait pas alimenter le Mac-mini sur la batterie, mais que l'on souhaitait utiliser le préhenseur pour pratiquer des dessins.