UNIVERSITÉ LAVAL

Livrable 1 - Robot Kinocto

Machines Électriques

présenté à

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

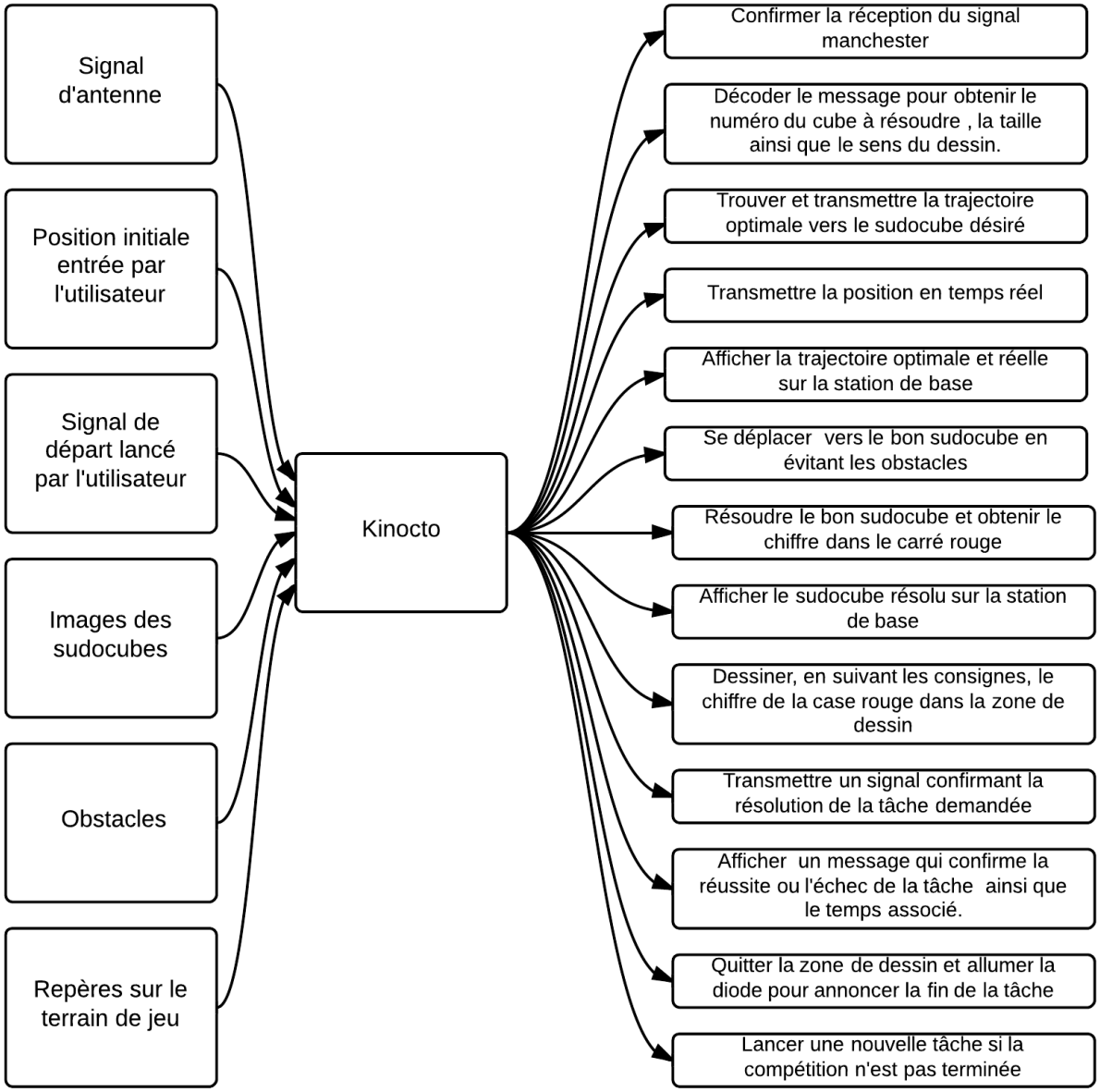
matricule	*nom*
910 058 073	Émile Arsenault
908 190 985	Philippe Bourdages
910 098 468	Pierre-Luc Buhler
998 107 355	Diane Fournier
908 159 170	Imane Mouhtij
908 318 388	Olivier Sylvain
910 055 897	Daniel Thibodeau
910 097 879	Francis Valois

Université Laval

11 février 2012

Chapitre 1

Diagramme de contexte



Chapitre 2

Description des propriétés fonctionnelles

Pour simplifier la lecture du tableau de la description des propriétés fonctionnelles, celui-ci à été séparé sur trois pages en trois différentes sections, soit : vision et traitement numérique représenté dans le tableau 2.1, communication et déplacement dans le tableau 2.2 ainsi qu'alimentation et affichage dans le tableau 2.3.

Tableau 2.1 – Description des propriétés fonctionnelles : section "Vision et Traitement Numérique"

	Fonctionnalités								
	Vision numérique			Traitement numérique					
	Détecter obstacles	Localiser le robot	Lire le cube	Calculer la trajectoire optimale	Contrôler le robot pour le dessin		Décoder le signal d'antenne	Choisir le cube selon le signal d'antenne	Résoudre le sudo-cube
Exigences du client	Temps de calcul (s)	Taux d'erreur (%)	Temps de calcul (s)	Temps de calcul (s)	Temps de calcul (s)	Taux d'erreur (%)			Temps de calcul (s)
Être autonome pendant un minimum de 10 minutes	3	2	5	2	3	3	2	4	2
Se déplacer selon la trajectoire optimale	5	5		5			3	3	
Effectuer une séquence complète en moins de 10 minutes	5	5	5	5	5			3	3
Alimenter le Mac mini avec une ondulation de tension inférieure à 200 mV									
Résoudre sudo-cube			5						5
Dessiner le chiffre selon le signal d'antenne dans une zone prédéfinie (jaune) avec une précision de ± 1cm						5			
Éviter les obstacles	5	5							5
Analyser le bon cube selon le signal d'antenne									
Utiliser la communication sans fil									
Concevoir un système de préhension pour le crayon									
Afficher position réelle	5					5			3
Afficher la trajectoire optimale			5						
Afficher le cube résolu sur la base									
Allumer une DEL lorsque tâche terminée									
Afficher message de fin									
Afficher message de départ									
Afficher trajectoire réelle avec un délai maximum de 15s									
Afficher les informations sur le robot									
Respecter un budget de 250$							3		

Tableau 2.2 – Description des propriétés fonctionnelles : section "Communication et Déplacement"

	Fonctionnalités								
	Communication							Déplacement	
	Recevoir le signal d'antenne	Communiquer entre le robot et la station de base	Communiquer entre le Mac-mini et le microcontrôleur	Commander les moteurs	Transmettre les images de la caméra vers le Mac	Contrôler la position de la caméra	Commander le préhenseur du crayon	Se déplacer sans toucher aux obstacles	
Exigences du client		Vitesse (Mo/s)	Vitesse (bits/s)					Résolution (Degrés)	Vitesse (m/s)
Être autonome pendant un minimum de 10 minutes	4							3	1
Se déplacer selon la trajectoire optimale	3		3	4	2			5	5
Effectuer une séquence complète en moins de 10 minutes	5		3	4				5	5
Alimenter le Mac mini avec une ondulation de tension inférieure à 200 mV									
Résoudre sudo-cube					3	1			
Dessiner le chiffre selon le signal d'antenne dans une zone prédéfinie (jaune) avec une précision de ± 1cm			5	3			5	5	5
Éviter les obstacles			5	3	4			5	
Analyser le bon cube selon le signal d'antenne	5								
Utiliser la communication sans fil									
Concevoir un système de préhension pour le crayon							5		
Afficher position réelle				5					
Afficher la trajectoire optimale	3	5							
Afficher le cube résolu la base		5			3				
Allumer une DEL lorsque tâche terminée			5						
Afficher message de fin		5		5					
Afficher message de départ									
Afficher trajectoire réelle avec un délai maximum de 15s									
Afficher les informations sur le robot									
Respecter un budget de 250$	3		1	5			2		

Tableau 2.3 – Description des propriétés fonctionnelles : section "Alimentation et affichage"

	Fonctionnalités										
	Alimentation				Affichage						
	Utiliser une pile rechargeable	Alimenter les moteurs	Alimenter le Mac	Alimenter les différents périphériques	Afficher message d'initiation de la tâche	Afficher le cube résolu	Allumer la DEL lorsque tâche complétée	Afficher la trajectoire optimale	Afficher la position réelle	Afficher message de fin	Afficher sur le LCD
Exigences du client		Puissance (W)		Ondulation de tension (V)					Temps d'actualisation (s)		
Être autonome pendant un minimum de 10 minutes	5	3	3	3							
Se déplacer selon la trajectoire optimale	5	5	5	3							
Effectuer une séquence complète en moins de 10 minutes											
Alimenter le Mac mini avec une ondulation de tension inférieure à 200 mV			5	5							
Résoudre sudo-cube	5		5	5							
Dessiner le chiffre selon le signal d'antenne dans une zone prédéfinie (jaune) avec une précision de ± 1cm	3	3	3	3							
Éviter les obstacles	3	3	3	3							
Analyser le bon cube selon le signal d'antenne	5		1	3	3	3	3				
Utiliser la communication sans fil											
Concevoir un système de préhension pour le crayon											
Afficher position réelle			1	3			3				5
Afficher la trajectoire optimale	3		1	3	3	3	3			5	
Afficher le cube résolu sur la base		5	1		3	3	3	5			
Allumer une DEL lorsque tâche terminée			1				2		5		
Afficher message de fin											
Afficher message de départ				3	5						
Afficher trajectoire réelle avec un délai maximum de 15s				3					5		
Afficher les informations sur le robot				3							5
Respecter un budget de 250$	5	2	3	5							

Chapitre 3

Description des cas d'utilisation

Cette section contient un résumé de chacun des différents cas d'utilisation. Elle est divisée en trois sous-sections : soit les cas d'utilisation en lien à l'usager, ceux en lien à la station de base et finalement ceux en lien au robot. Le diagramme des cas d'utilisation est présenté à la section 3.4.

3.1 Cas d'utilisation en lien avec l'usager

3.1.1 Lancer le signal de départ

À l'aide d'un Graphical user interface ou interface graphique pour utilisateur (GUI) installé sur la station de base, l'utilisateur clique sur un bouton qui permet au robot de lancer son exécution (signal de départ transmis au robot pour commencer à effectuer une tâche).

3.1.2 Entrer les coordonnées initiales du robot

Avant d'envoyer le signal de départ au robot, l'utilisateur doit pouvoir entrer dans le GUI de la station de base les coordonnées de la position de départ du robot par rapport au terrain de jeu.

3.2 Cas d'utilisations en lien avec la station de base

3.2.1 Localiser le robot

À l'aide de la Kinect, la station de base doit être en mesure de trouver la position du robot sur le terrain de jeu.

3.2.2 Détecter les obstacles

La station de base, au moyen de la Kinect, doit pouvoir à détecter la position des deux obstacles disposés entre les deux zones principales du terrain de jeu, ainsi que les limites (murs) du terrain.

3.2.3 Afficher la solution du sudocube

La station de base doit être capable d'afficher une image du sudocube résolu et d'indiquer à l'écran le chiffre qui se trouve dans la case rouge du sudocube.

3.2.4 Afficher le message de confirmation du lancement de la tâche

La station doit être capable d'afficher un message de confirmation du lancement de la tâche pour l'usager lorsque le message de confirmation du robot est reçu.

3.2.5 Afficher la trajectoire prévue et réelle du robot

La trajectoire calculée et transmise par le robot doit être affichée à l'écran de la station de base. De plus, à l'aide des données fréquemment transmises par la Kinect contenant la position du robot, la station doit afficher, en comparaison à la trajectoire prévue, la trajectoire réelle du robot.

3.2.6 Transmettre des données au robot

La station doit transmettre au robot, par connexion sans fil, des messages indiquant de lancer une nouvelle tâche, la position des obstacles et la position de ces derniers.

3.2.7 Recevoir des données provenant du robot

La station de base doit être capable de recevoir, par connexion sans fil, des données provenant du robot comme la trajectoire que le robot prévoit emprunter, le sudocube résolu, le message de confirmation du lancement d'une tâche et le message indiquant la fin d'une tâche.

3.2.8 Afficher le message confirmant que la tâche a été complétée avec succès ou a échoué

La station de base doit afficher un message à l'écran confirmant que la tâche a été complétée avec succès ou un message d'échec si cette dernière a pris de plus de 10 minutes pour s'exécuter.

3.2.9 Afficher le temps d'exécution de la tâche

La station de base doit être en mesure d'afficher à l'écran le temps d'exécution de la tâche.

3.3 Cas d'utilisation en lien avec le robot

3.3.1 Déterminer le chemin optimal entre un point A et un point B tout en évitant les obstacles

En connaissant la configuration et la position des obstacles, le robot doit calculer la trajectoire optimale qu'il doit emprunter pour se rendre d'un point à un autre.

3.3.2 Se déplacer d'un point A au point B

Le robot doit être capable de se déplacer, sur le terrain de jeu, d'un point A vers un point B avec un contrôle automatisé des roues.

3.3.3 Indiquer son adresse IP locale pour que la station de base puisse communiquer avec le robot

Le robot doit être capable d'indiquer son adresse IP locale à la station de base pour que celle-ci établisse une connexion sans fil avec le robot.

3.3.4 Extraire les informations d'un sudocube à partir d'une photo

À l'aide d'une caméra embarquée sur le robot, il doit prendre une photo et représenter le sudocube à partir de l'image obtenue.

3.3.5 Résoudre le sudocube et identifier le chiffre dans le carré rouge

Le robot doit effectuer des traitements sur l'image du sudocube afin de le représenter et de le résoudre par la suite. Il identifie également l'emplacement de la case rouge à l'intérieur du sudocube et détermine donc le chiffre qui s'y trouve.

3.3.6 Recevoir des données provenant de la station de base

Le robot doit être capable de recevoir, par connexion sans fil avec la station de base, des messages indiquant de lancer une nouvelle tâche, la position des obstacles et sa position.

3.3.7 Transmettre des données à la station de base

Le robot doit transmettre, par connexion sans-fil, des données à la station de base comme la trajectoire que le robot prévoit emprunter, le sudocube résolu, le message de confirmation du lancement d'une tâche et le message indiquant la fin d'une tâche.

3.3.8 Contrôler une DEL

Une fois le message de fin de tâche transmis et le robot à l'extérieur de l'air de dessin, ce dernier doit allumer une DEL qui est installée sur le robot.

3.3.9 Lancer de nouvelles tâches si la compétition n'est pas terminée

Si le temps alloué pour la compétition (10 minutes) n'est pas écoulé, le robot doit commencer une nouvelle tâche après en avoir terminer la séquence d'opération.

3.3.10 Se localiser

Le robot doit être capablee en tout temps de déterminer sa position et son orientation, à l'aide de la webcam et des données reçues, ce qu'il lui permet de connaître son emplacement par rapport à l'antenne, aux murs, aux obstacles et aux sudocubes. De plus, il doit transmettre ces données à la station de base de façon régulière.

3.3.11 Décoder la transmission de l'antenne

Le robot doit se déplacer au-dessus de l'antenne et de capter le signal transmis par celle-ci. Il doit ensuite le décoder pour trouver les paramètres indiquant le sudocube ciblé ainsi que l'orientation et la taille du chiffre à dessiner dans la zone de base.

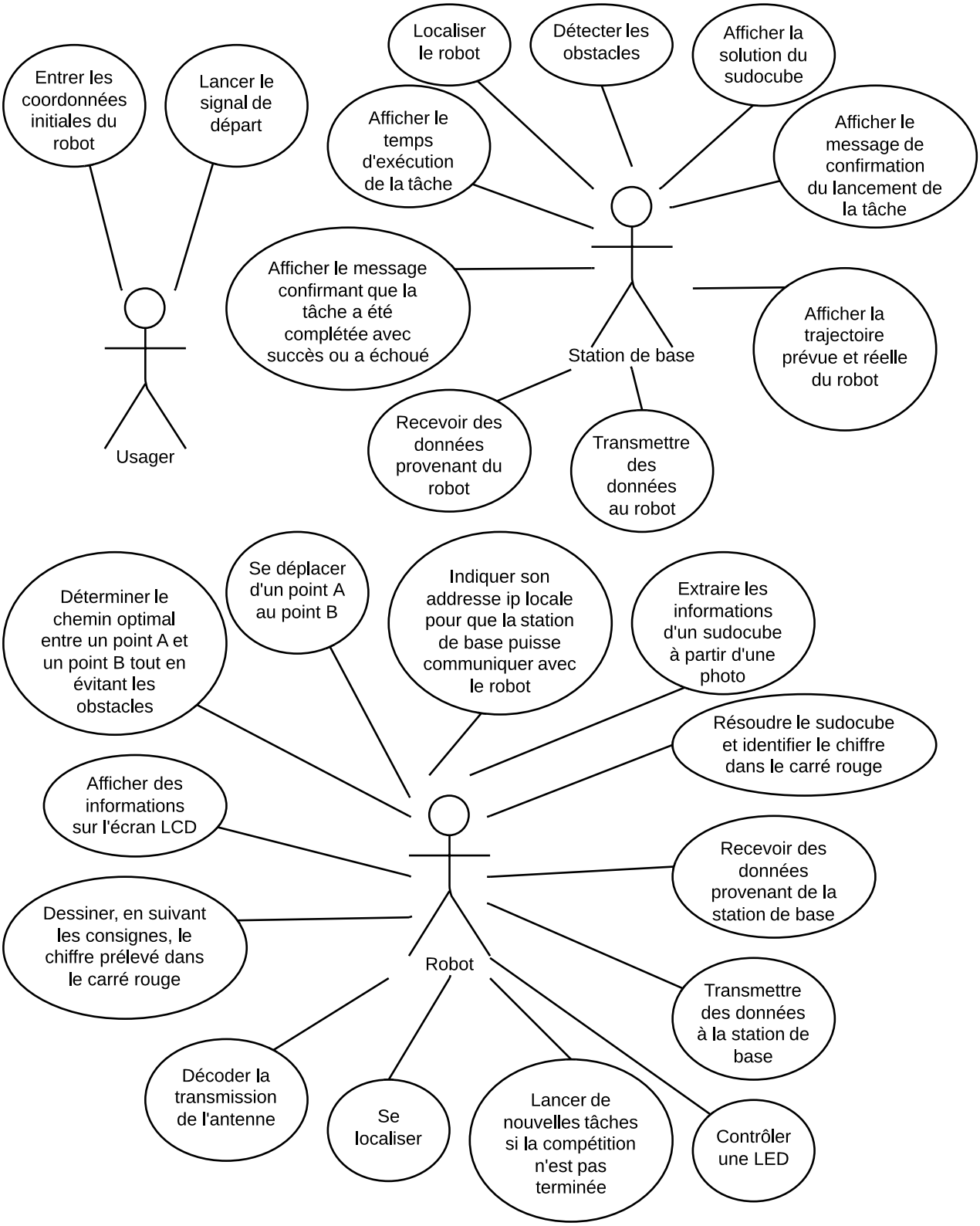
3.3.12 Dessiner, en suivant les consignes, le chiffre prélevé dans le carré rouge

Le robot doit utiliser un préhenseur afin de poser la mine du crayon sur la table, une fois dans l'air de dessin. Il doit ensuite se déplacer de façon à dessiner le chiffre trouvé dans la case rouge du sudocube, tout en suivant les dimensions exigées et les paramètres décodés.

3.3.13 Afficher des informations sur l'écran LCD

Les paramètres trouvés lors du décodage du signal de l'antenne doivent être affichés sur un écran LCD installé sur le robot.

3.4 Diagramme des cas d'utilisation


Entrer les coordonnées initiales du robot
Lancer le signal de départ
Usager
Localiser le robot
Détecter les obstacles
Afficher la solution du sudocube
Afficher le temps d'exécution de la tâche
Afficher le message de confirmation du lancement de la tâche
Afficher le message confirmant que la tâche a été complétée avec succès ou a échoué
Station de base
Afficher la trajectoire prévue et réelle du robot
Recevoir des données provenant du robot
Transmettre des données au robot
Déterminer le chemin optimal entre un point A et un point B tout en évitant les obstacles
Se déplacer d'un point A au point B
Indiquer son addresse ip locale pour que la station de base puisse communiquer avec le robot
Extraire les informations d'un sudocube à partir d'une photo
Résoudre le sudocube et identifier le chiffre dans le carré rouge
Afficher des informations sur l'écran LCD
Recevoir des données provenant de la station de base
Dessiner, en suivant les consignes, le chiffre prélevé dans le carré rouge
Robot
Transmettre des données à la station de base
Décoder la transmission de l'antenne
Se localiser
Lancer de nouvelles tâches si la compétition n'est pas terminée
Contrôler une LED

Chapitre 4

Diagrammes de séquences

Ce chapitre présente les différentes figures associées au diagramme des séquences. Ce diagramme est séparé selon cinq portions relatives aux fonctions particulières du robot. La figure 4.1 présente les liens entre l'utilisateur et la kinocto. La figure 4.2 présente les liens entre la kinocto et son environnement afin de déterminer sa position. La figure 4.3 présente les liens entre la kinocto et la station de base pour la transmission et l'affichage de la trajectoire optimale.La figure 4.4 présente les liens entre la kinocto et la table de jeu dans l'optique du déplacement de celle-ci à travers les obstacles vers la zone de lecture ainsi que les liens avec la résolution du sudocube. La figure 4.5 présente les liens entre la kinocto et ses différents périphériques lors de la production du dessin et de la confirmation de la résolution de la tâche.

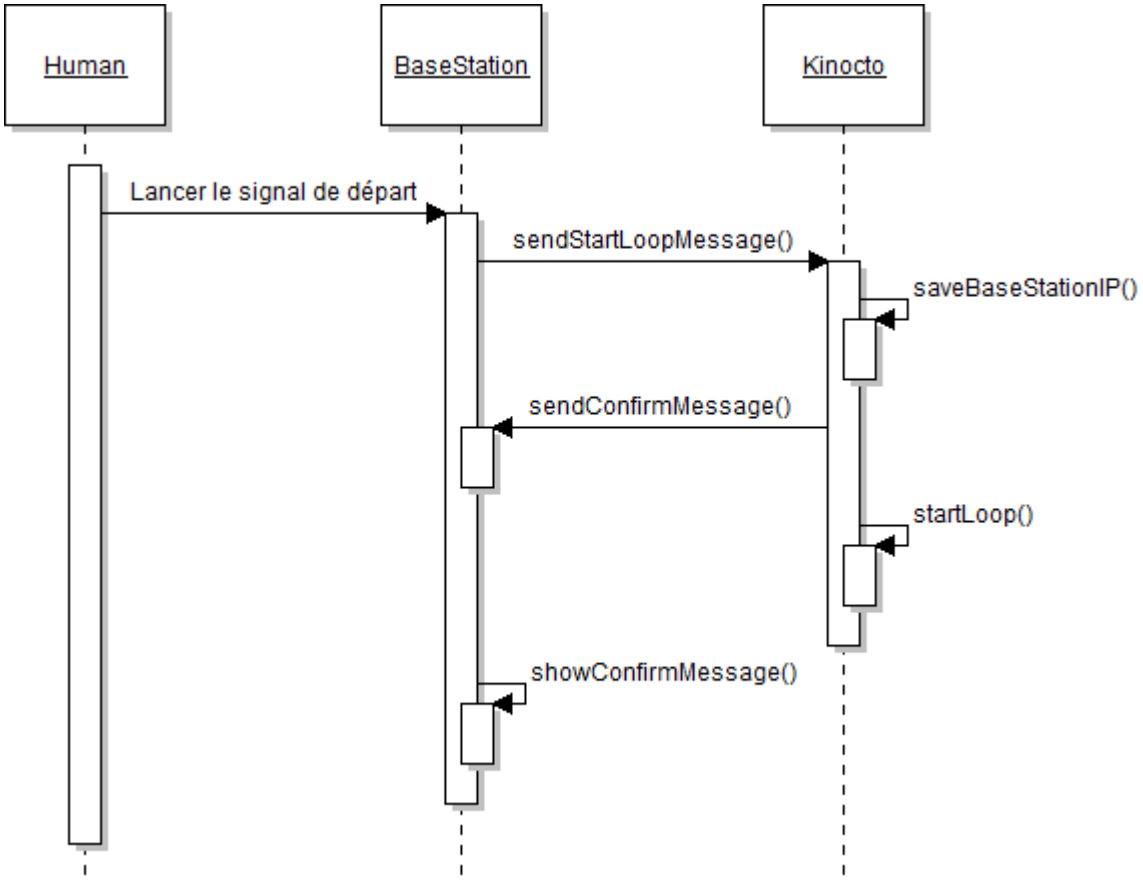


Figure 4.1 – Diagramme des séquences présentant les liens entre l'utilisateur et la kinocto

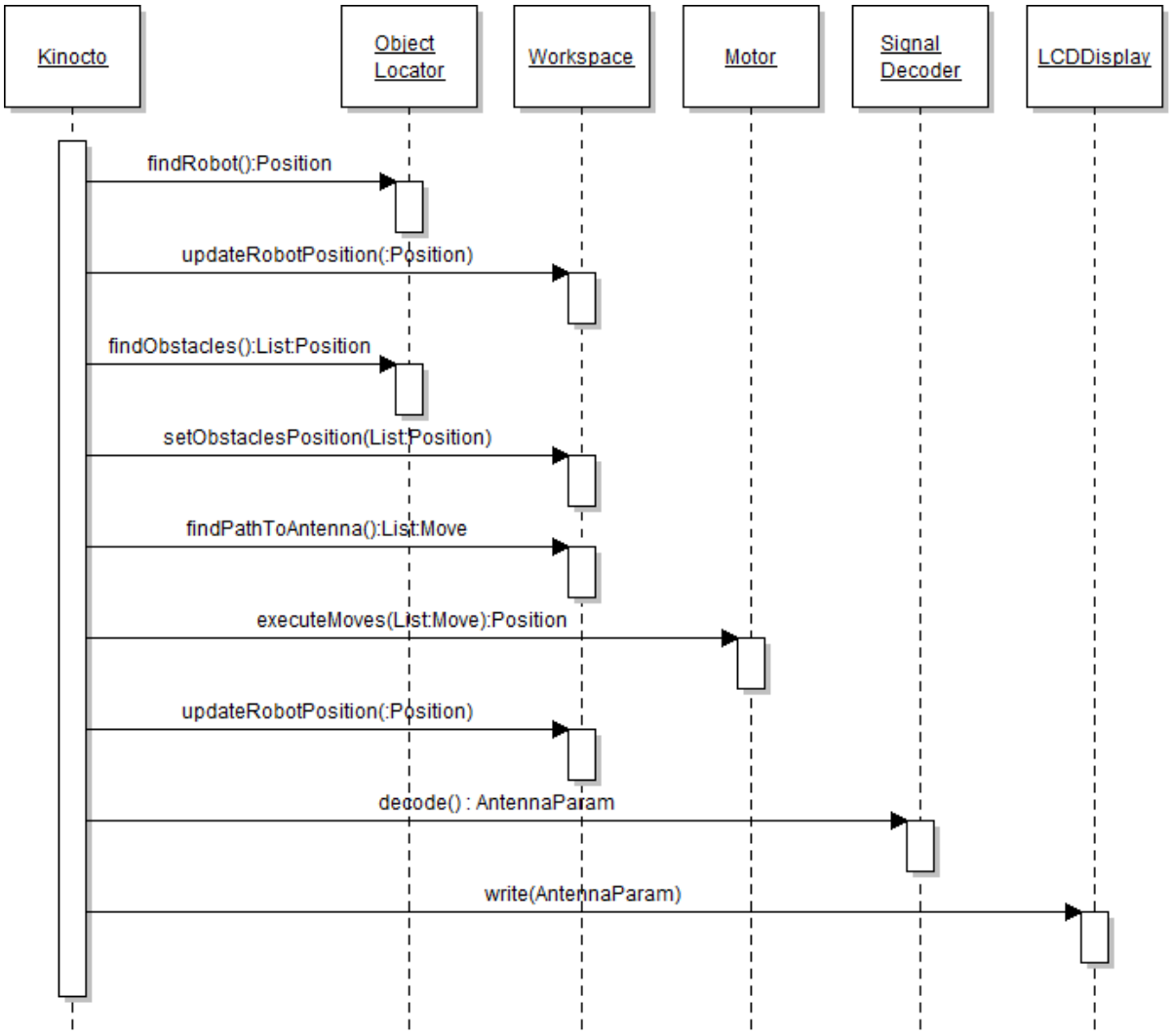


Figure 4.2 – Diagramme des séquences présentant les liens entre la kinocto et son environnement afin de déterminer sa position

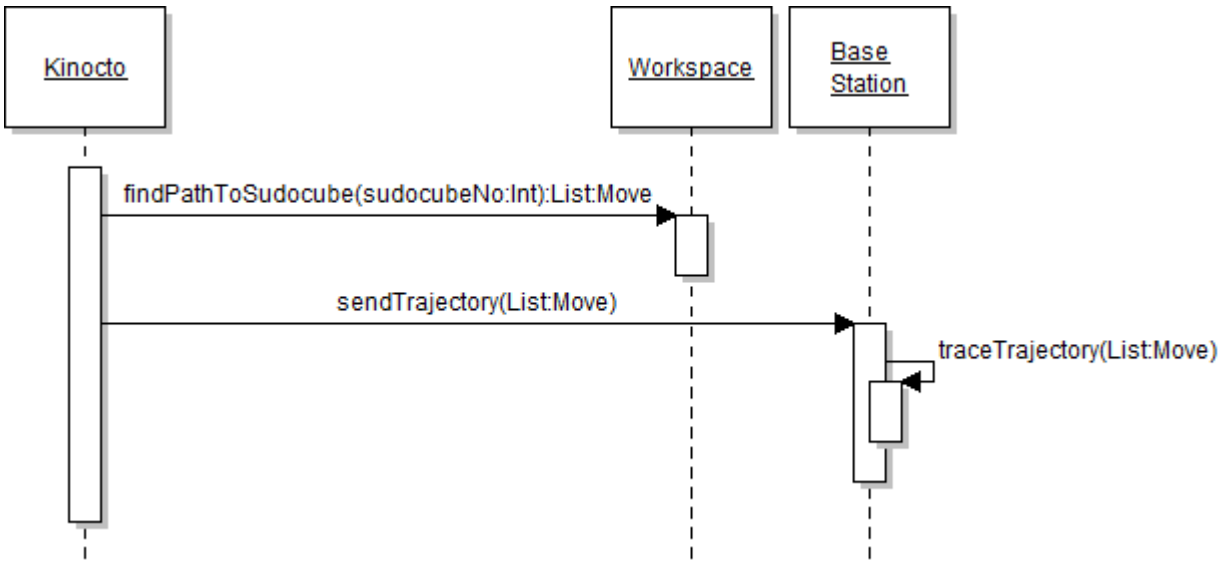


Figure 4.3 – Diagramme des séquences présentant les liens entre la kinocto et la station de base pour la transmission et l'affichage de la trajectoire optimale

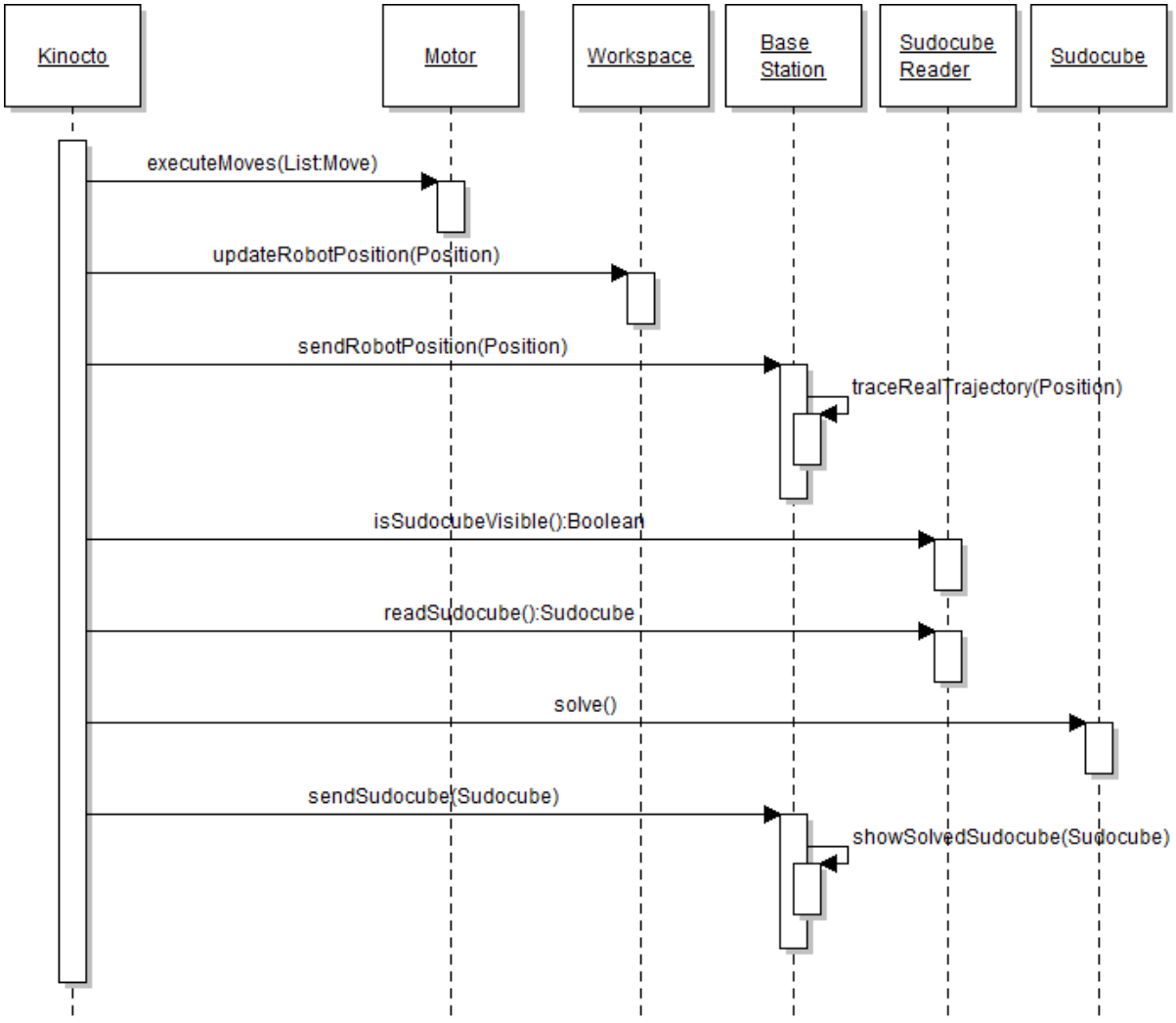


Figure 4.4 – Diagramme des séquences présentant les liens entre la kinocto et la table de jeu dans l'optique du déplacement de celle-ci à travers les obstacles vers la zone de lecture ainsi que les liens avec la résolution du sudocube

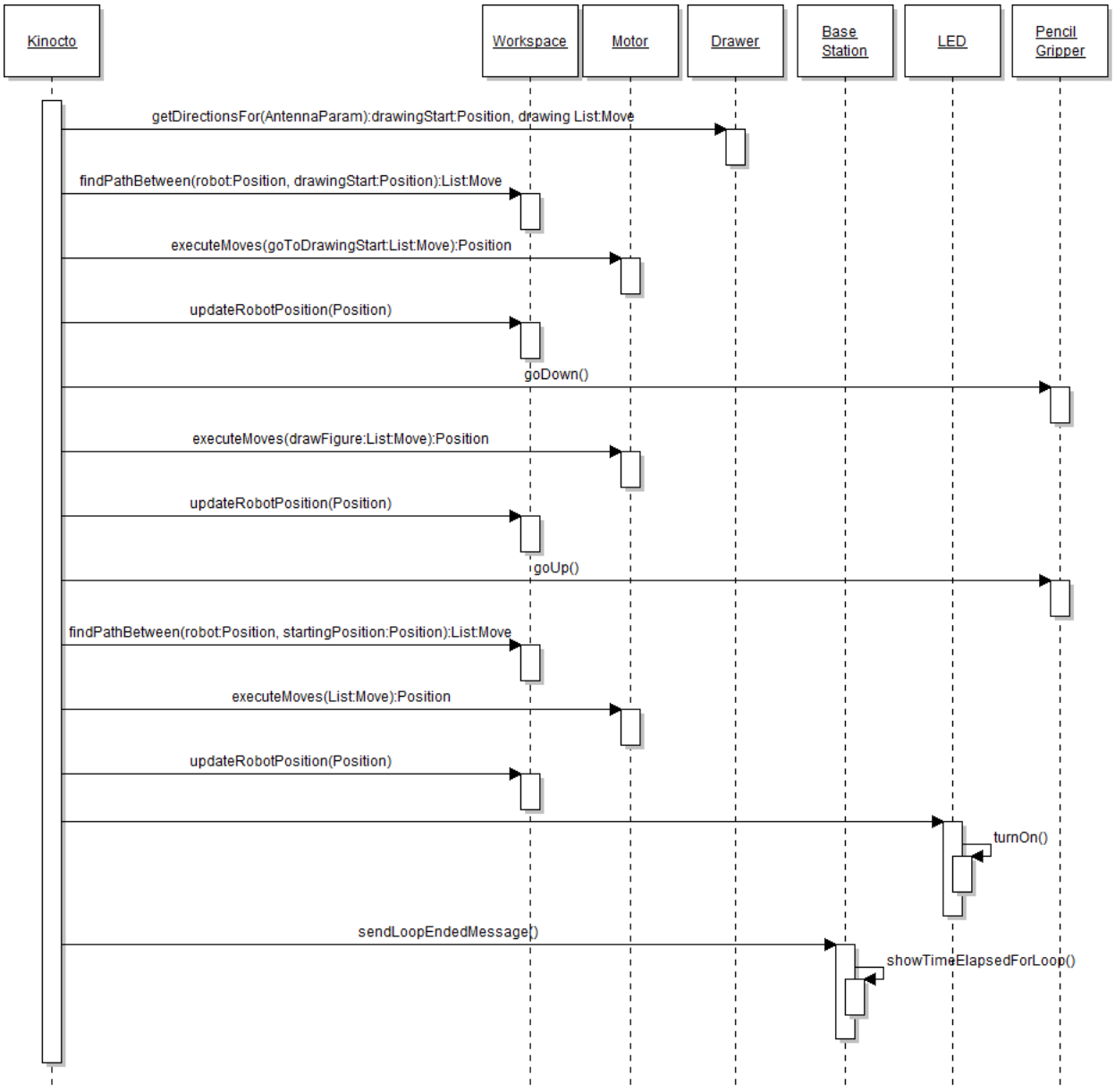
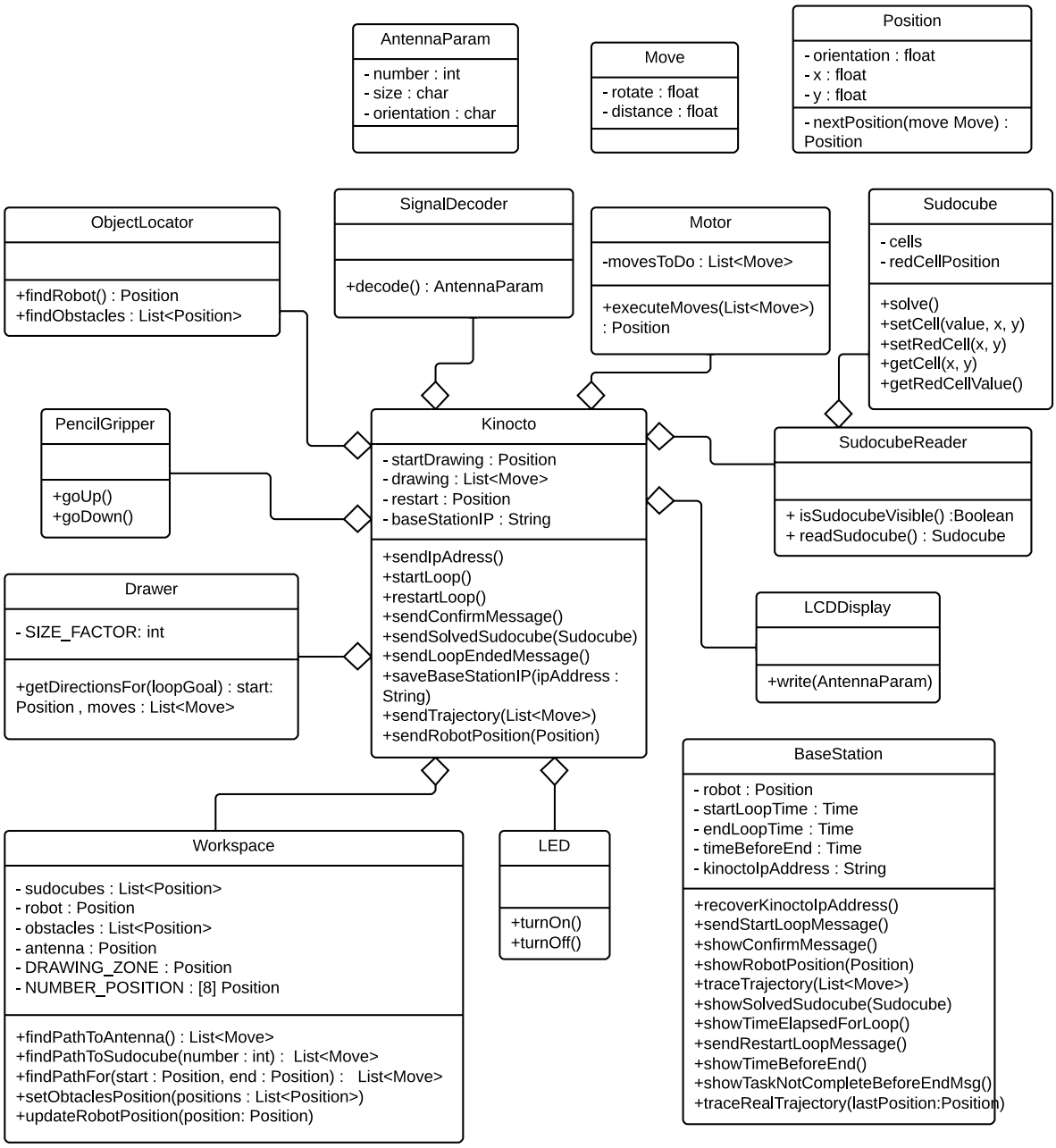


Figure 4.5 – Diagramme des séquences présentant les liens entre la kinocto et ses différents périphériques lors de la production du dessin et de la confirmation de la résolution de la tâche

Chapitre 5

Diagrammes de classes



Chapitre 6

Description des prototypes

6.1 Préhenseur

Comme dispositif pour le traçage des dessins par le robot, nous opterons pour une solution simple et robuste. Afin d'éviter d'augmenter inutilement la charge de travail du microcontrôleur, nous avons décidé d'éviter d'utiliser des solutions complexes qui pourraient exiger la production d'un signal particulier par le microcontrôleur. Le signal de sortie qui devra être produit sera simplement un signal binaire : « 1» lorsque le crayon devra être en mode traçage et « 0 » lorsque le crayon sera soulevé et donc en mode « attente ». Pour maintenir le crayon en position « attente » durant la durée du trajet, nous utiliserons un système de maintien magnétique qui gardera le crayon en position relevé lorsqu'il recevra le signal « 1 » logique. Pour abaisser le crayon et ainsi entrer en mode traçage, nous utiliserons un ressort à faible constante de rappel qui entrera en jeu lorsque le système de maintien magnétique relâchera le crayon « 0 » logique. Il est bien important que la constante de rappel du ressort soit faible pour ne pas briser la mine lors du contact de celle-ci avec la surface de traçage. La constante de rappel du ressort devra également être inférieure à la force exercée par le système de maintien magnétique qui relèvera le crayon en mode attente, une fois le dessin terminé.

6.2 Récepteur et décodeur du signal Manchester

Pour le récepteur du signal magnétique, nous utiliserons un « tone decoder » qui est utilisé dans de nombreuses applications qui nécessitent la reconnaissance de certaines tonalités bien précises. Comme nous avons besoin de détecter une série de tonalités identiques qui se suivent en formant un code, ce genre de dispositif nous convient parfaitement. Le « tone decoder » recevra le signal par une boucle de fil installée à un endroit sur le robot qui captera la variation du flux magnétique produite par le fil sous la table et détectera si la tonalité inaudible est émise ou non. Lorsque la fréquence (ou tonalité) sera reçue par le récepteur, celui-ci présentera zéro volt en sortie et à l'opposé, lorsque le récepteur ne détectera pas la fréquence ciblée, il présentera 5V à sa sortie. Il exécutera ces opérations d'une façon suffisamment rapide pour que le segment binaire reçu soit exactement le même que celui transmis, mais complètement opposé étant donné la configuration logique du « tone decoder ». Ce message sera ensuite inversé par l'unité de traitement suivante et traité pour que le robot exécute la tâche adéquate. L'avantage de cette approche est que le « tone decoder » est un circuit intégré très robuste aux perturbations magnétiques de son environnement, il est peu énergivore et très compact.

6.3 Microcontrôleur

La composante qui effectuera le lien entre le pont en H qui contrôle les moteurs, les encodeurs sur les moteurs, le signal décodé par le récepteur du signal d'antenne, le préhenseur, la D.E.L., l'écran LCD et le Mac mini sera un microcontrôleur modèle Texas Instrument Stellaris LM3S9B92. Nous avons choisi ce microcontrôleur, car il possède un grand nombre de broches d'entrée/sortie, 4 sorties de signaux PWM, 2 interfaces d'encodeur à quadrature qui permettront le contrôle et l'asservissement des moteurs et il possède également deux ports de communication UART ainsi qu'une interface USB pour la communication avec le Mac mini. De plus, le microcontrôleur est programmé en C à l'aide d'un compilateur, d'un IDE et de librairies de pilotes de périphériques fournies.

6.4 Affichage sur le robot

Pour afficher les informations nécessaires sur le robot, nous utiliserons un écran à cristaux liquides 16 x 2 caractères. Cet écran est commandé par 8 lignes de données qui permettent de transmettre des caractères ASCII et 3 lignes de contrôle, RS, R/W et E. Le huitième bit de données peut également servir de « busy flag », ce qui permet de savoir si le contrôleur de l'écran a terminé d'exécuter les dernières instructions. En plus de la banque de caractères prédéfinie, il est possible d'ajouter des caractères "maison" dans des espaces libres de la mémoire de caractères. Le contraste peut être contrôlé par un potentiomètre qui fournit la référence sur la broche 3. En ce qui concerne l'alimentation, l'écran peut être alimenté à partir du microcontrôleur, en excluant le rétroéclairage, qui n'est pas nécessaire pour distinguer les caractères dans un environnement éclairé. Si le rétroéclairage devait être utilisé, il serait nécessaire de l'alimenter à l'aide d'une source indépendante du microcontrôleur, car il nécessite 150 mA. Les fonctions codées dans les fichiers ecran.h et ecran.c de l'annexe A.1 permettent d'envoyer des caractères et des chaînes de caractères sur le LCD et de contrôler le déplacement du curseur.

6.5 Communication entre le microcontrôleur et le Mac mini

Comme le microcontrôleur possède une interface USB qui permet la communication avec l'ordinateur par un port série UART, nous utiliserons ce protocole pour communiquer. Le port UART lève une interruption lorsqu'il reçoit un caractère dans son registre de données. Les données reçues peuvent être enregistrées dans un tampon circulaire durant l'interruption pour être traitées dans le processus principal. Du côté du Mac mini, un terminal série permettra d'envoyer des données par l'interface USB.

6.6 Alimentation du Mac mini

En ce qui a trait au système d'alimentation du Mac mini, le dispositif à concevoir doit réussir à élever la tension de la pile jusqu'à 24V. Cette tension correspond à un point de fonctionnement qui est jugé comme idéal selon les spécifications techniques de l'appareil. On constate par la suite que la puissance demandée par le Mac mini sera la plus grande dépense énergétique du système. Il faut donc envisager une alimentation avec un très bon rendement, de manière à limiter le dimensionnement de la pile. L'usage d'un régulateur de tension conventionnel, qui utilise une tension supérieure à l'alimentation, n'est pas envisageable pour le projet, dans la mesure où le rendement dépasse rarement les 50%. Ce qui s'offre à la conception de l'alimentation est un système de type «Boost» qui effectue une amplification de la tension continue. Ce type de circuit est le pendant en tension continue du transformateur. En soi, il est possible de réaliser la fonction d'amplification au moyen de composants discrets. Cependant, l'implantation pratique demande beaucoup de temps et de ressources et est généralement moins robuste qu'une alimentation utilisant des composants intégrés. Il est préférable, vu les coûts de l'électronique actuelle, d'opter pour des composants intégrés qui effectuent le même travail avec des rendements très élevés (de l'ordre du 90% et plus). Aussi, vu le coût de l'ordinateur et sa sensibilité aux oscillations de tension, il est très important de considérer la stabilité de la tension de sortie de l'alimentation. Les régulateurs envisageables pour l'application et les spécifications en courant présentent une ondulation de tension inférieure aux niveaux maximaux du Mac mini ($\pm 200mV$). Il est donc tout indiqué d'opter pour un circuit intégré de type «Boost», vu son bon rendement, sa robustesse, sa facilité d'implantation ainsi que sa stabilité de tension de sortie.

6.7 Alimentation de l'électronique embarquée

En ce qui a trait au système d'alimentation de l'électronique embarquée, l'alimentation de l'électronique autre que le Mac mini doit se faire optimalement en 5V, puisqu'il s'agit d'une tension utilisable pour le servomoteur de la tourelle de la caméra et permet aussi de donner un point de référence au pont en H. Par ailleurs, la plupart des microcontrôleurs ont des entrées 5V, il est donc de mise d'utiliser cette tension pour l'électronique auxiliaire. Le dispositif devra abaisser la tension de la pile à 5V puisque celle-ci sera sélectionnée avec une tension supérieure. On peut tout de suite penser à l'usage d'un régulateur, cependant, pour les mêmes raisons qu'énoncées à la section 6.6, un faible rendement de l'alimentation pourrait dégrader la durée de vie de la pile. Vu qu'il est nécessaire d'abaisser la tension, l'utilisation d'une technologie de type «Boost» est impossible. Il existe cependant la technologie de type «Buck», qui produit exactement l'effet contraire. L'utilisation de composés discrets requiert davantage de temps et est moins robuste. Une solution impliquant des composés intégrés doit être envisagée. Par ailleurs, les rendements de ce type d'alimentation peuvent s'élever à plus de 90%. Les composantes comme le microcontrôleur et le pont en H requièrent une tension d'alimentation stable et l'emploi d'une alimentation faite à partir de composants intégrés permet de convenir à ce besoin. Il est donc tout indiqué d'opter pour un circuit intégré de

type «Buck», vu son bon rendement, sa robustesse, sa facilité d'implantation ainsi que sa stabilité de tension de sortie.

6.8 Communication sans fil

Pour amorcer la communication sans fil entre le robot et la station de base, il faut que cette dernière sache l'adresse IP du robot. Il nous est impossible de brancher un clavier et un écran sur le robot afin de récupérer celle-ci et le réseau sans fil de l'université empêche l'utilisation de service tel que www.NOIP.com pour obtenir de façon dynamique l'IP de la carte réseau. Nous avons donc créé un script «bash» pour récupérer l'adresse IP du robot et l'envoyer à une application en ligne appelée «Pastebin» qui permet d'enregistrer du texte. Le script est placé dans le fichier «/etc/rc.d/rc.local» afin qu'à chaque démarrage, l'adresse IP soit connue. Pour valider l'adresse IP obtenue, il suffit d'essayer de se connecter par «SSH» sur le robot ce qui confirme la possibilité d'utiliser le réseau sans fil pour communiquer avec le robot.

Ultérieurement, nous utiliserons une api en C++ ou en Python pour communiquer.

6.9 Lecture des informations d'un sudocube avec la caméra

Pour réaliser ce prototype, OpenCV fut utilisé ainsi que quelques photos des sudocubes de l'environnement de travail du robot. Le langage de programmation C++ fut employé afin de garantir un temps de traitement relativement faible. Les opérations sur les images exigeant beaucoup de cycles processeur. De plus, l'accès à un plus grand nombre d'exemples de code OpenCV en C++(contrairement au Python) a aussi motivé le choix de ce langage.

L'algorithme testé dans ce prototype commence par effectuer une segmentation par couleur verte sur la photo afin d'isoler le cadre du sudocube. Puis, un algorithme de détection des contours est appliqué. OpenCV retourne, après ces opérations, deux polygones rectangulaires que l'on peut utiliser pour vérifier l'existence du cadre vert dans l'image. On s'assure que le cadre remplit une bonne partie de l'image en calculant l'aire de celui-ci. Cela permet de garantir qu'il y a assez de détails pour lire correctement les cases du sudocube. Ensuite, l'image originale est convertie en tons de gris pour appliquer encore une fois un algorithme de détection des contours. Cela permet de récupérer les polygones de toutes les cases du sudocube. Comme il y a quelques polygones qui ne sont pas des cases, on les élimine en calculant leur aire et en ne gardant que ceux qui sont suffisamment grands.

Afin de garantir la lecture de toutes les cases du sudocube, on pourrait ajouter une détection automatique du seuil de l'algorithme de détection des contours en vérifiant le nombre de polygones trouvés. Il faut en trouver exactement 47.

De plus, pour assurer une bonne lecture des caractères, on pourrait effectuer un réalignement du cadre vert.

Ce qui reste à faire : effectuer une segmentation sur la couleur rouge en mode HSV afin d'identifier la case rouge et créer un algorithme de reconnaissance des caractères pour lire les chiffres dans les cases.

6.10 Recherche de chemin

Afin de résoudre le problème de recherche d'un chemin à parcourir par le robot Kinocto afin d'éviter les obstacles, nous avons dû nous pencher sur certaines contraintes particulières à notre situation. Nous avons tout d'abord décidé de représenter la table à l'aide d'un nuage de points quadrillés, un peu à la manière d'un graphe ayant des coordonnées (x,y). Ce type de représentation facilite beaucoup la recherche de chemin puisque les algorithmes les plus efficaces travaillent sur des graphes. Par la suite, il a fallu considérer que notre robot n'est pas un point ponctuel sur la table. Ce problème a été contourné en insérant une marge autour des obstacles. En ayant des obstacles plus gros, ceci nous permet de garder une représentation du robot comme un point et utiliser un algorithme de recherche de chemin. L'algorithme A* a été choisi, car il est très rapide et facile à implémenter. Finalement, il y a une dernière contrainte à considérer, venant du fait que nous utilisons un nuage de point comme référence sur la table. Il est impossible de se déplacer en diagonale sans "zigzaguer" à travers les obstacles. Ceci engendre beaucoup de rotations nécessaires afin d'arriver au point final ce qui contribue à l'incertitude de la position réelle du robot. Nous ajouterons donc une couche logicielle permettant d'identifier les points critiques du chemin trouvé par l'algorithme A*. Ainsi, cette liste de points permettra au robot de se déplacer en ligne droite lorsqu'il fera des diagonales en ignorant les points inutiles entre les points critiques qui créaient le mouvement involontaire, et réduira par le fait même le nombre de rotations nécessaires avant d'arriver au point final.

6.11 Kinect

La Kinect est l'outil qui va nous aider à localiser les obstacles et localiser notre robot sur la table. Après l'installation d'OpenNI et OpenCV, nous avons testé la Kinect avec le fichier test.cpp qui permet de faire une capture d'image.
Pour localiser le robot par rapport à la Kinect, nous allons le modéliser avec une figure spécifique sur celui-ci (exemple : deux cercles). Ensuite, grâce à la connaissance de la distance entre les deux cercles, nous pourrons déterminer à chaque rafraichissement de position : la distance du robot par rapport à la kinect et par rapport aux obstacles ainsi que sa position angulaire approximative. Pour les équations, nous allons utiliser des équations simples comme les règles de trigonométrie. De plus, afin d'éviter le bruit dans nos images, il est judicieux d'appliquer des filtres correctifs. Le filtre passe-bas est le plus réputé pour enlever le bruit. Toutefois, il rend les images "flou", alors il faut chercher les meilleurs paramètres pour obtenir un filtre optimal et efficace. En ce qui concerne la détection d'obstacles, nous avons deux choix. Le premier choix consiste à faire une capture avec la caméra et en faire l'analyse 2D

tandis que le deuxième choix consiste à utiliser l'image de profondeur. Pour le moment, la solution qui semble la plus efficace est l'utilisation de l'image de profondeur qui consiste tout d'abord à isoler l'obstacle qui possède une forme déjà connue et conserver les points dont la profondeur est proche de celle des obstacles. Par la suite, nous conservons une image en teinte de gris, car c'est le contour de l'obstacle qui nous intéresse. Finalement, on se sert d'OpenCV pour définir un détourage de l'obstacle et l'affiner.

Chapitre 7

Gestion de projet

Le projet a été décortiqué en diverses tâches et les ressources humaines ont été associées à celles-ci. Les tâches dépendent entièrement des fonctionnalités du robot ainsi que des exigences du client. Le tout est présenté dans le fichier kinocto.gan, présent dans l'archive equipe05_livrable1.zip.

Annexe A

Annexes

A.1 Code test pour affichage sur LCD

A.1.1 ecran.h

```
#ifndef ECRAN_H_
#define ECRAN_H_

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/timer.h"
#include "inc/lm3s9b92.h"

//definition de trucs qu vont etre pratiques

//#define STCTRL (*((volatile unsigned long *)0xE000E010))
//#define STRELOAD (*((volatile unisgned long *)0xE000E014))
//#define attend(t) SysCtlDelay((SysCtlClock()/3)/(1000/t))


#define ECRAN_DATA GPIO_PORTE_DATA_R
#define ECRAN_CTRL GPIO_PORTA_DATA_R
//volatile unsigned long ulLoop;

//bit de data du LCD
#define ECRAN_D0 0x01 //PE0
#define ECRAN_D1 0x02 //PE1
#define ECRAN_D2 0x04 //PE2
#define ECRAN_D3 0x08 //PE3
#define ECRAN_D4 0x10 //PE4
#define ECRAN_D5 0x20 //PE5
#define ECRAN_D6 0x40 //PE6
#define ECRAN_D7 0x80 //PE7

//bit de control du LCD
#define ECRAN_RS 0x01 //PA0  reset
#define ECRAN_RW 0x02 //PA1  read/write
#define ECRAN_EN 0x04 //PA2  enable
#define ECRAN_BF 0x08 //PA3 "busy flag", indique que l'ecran est occupe

//volatile unsigned long ulLoop;
void ecranClear(void);
void ecranInit(void);
void ecranWriteChar(char caractere);
void ecranWriteLine(char * line, unsigned short size);
```

```
46 void ecranSetPosCursor(short pos);
47 void ecranAttend(void);
48 void ecranControl(unsigned long input);
49
50 #endif /*ECRAN_H_*/
```

A.1.2 ecran.c

```
1  #include "ecran.h"
2
3
4  /*
5  * Cette fonction permet de s'assurer que l'ecran n'est pas occupe avant d'ecrire
6   */
7  void ecranAttend(void)
8  {
9      ECRAN_CTRL &= ~(ECRAN_RS); // RS = 0
10     ECRAN_CTRL |= ECRAN_RW; // RW = 1
11     volatile unsigned long busyflag = ECRAN_CTRL & ECRAN_BF;
12     while (busyflag != 0) //on regarde le busyflag pour s'assurer que l'ecran est pas ↩
           occupe
13     {
14             busyflag = GPIO_PORTD_DATA_R & ECRAN_BF;
15     }
16      return;
17 }
18
19 /*
20  * Cette fonction execute la routine d'initialisation de l'ecran dans le mode qu'on veut
21  */
22 void ecranInit(void)
23 {
24     ecranControl(ECRAN_D5 | ECRAN_D4 | ECRAN_D3 | ECRAN_D2);
25     ecranControl(ECRAN_D3 | ECRAN_D2 | ECRAN_D1);
26 }
27
28 /*
29  * Cette fonction permet de faire des commandes tel que l'initalisation, changement de ↩
       positions et
30  * trucs du genre, tout ce qui est pas l'ecriture de caracteres sur l'ecran
31  */
32 void ecranControl(unsigned long input)
33 {
34 ecranAttend();
35  volatile unsigned long ulLoop;
36 ECRAN_DATA = input; //on met notre entree sur le port de donnees
37 ECRAN_CTRL &=~(ECRAN_RS | ECRAN_RW); //reset et read/write a zero pour pas qu'il re-ecrive↩
       par erreur
38 ECRAN_CTRL ^= ECRAN_EN;
39 ulLoop = SYSCTL_RCGC2_R;
40         for (ulLoop = 0; ulLoop < 200; ulLoop++)
41         {
42              // on met un delai pour que le LCD puisse voir le enable
43         }
44 ECRAN_CTRL ^=ECRAN_EN;
45 ulLoop = SYSCTL_RCGC2_R;
46         for (ulLoop = 0; ulLoop < 200; ulLoop++)
47         {
48         // on attend un peu pour que l'instruction s'execute avant de changer la pin 7
49         }
```

```
50
51 ECRAN_DATA &= ~(ECRAN_D7); //on reset la pin 7 parce elle output aussi le busy flag et ont↩
       pas veux etre pogner dans ecranAttend()
52 }
53
54
55
56 /*
57  * Cette fonction efface le contenu de l'ecran
58  */
59 void ecranClear(void)
60 {
61     ecranControl(ECRAN_D0);
62     volatile unsigned long ulLoop;
63
64     for (ulLoop = 0; ulLoop < 2000; ulLoop++)
65     {
66         //loop pour etre sur que ton est bien clearer
67     }
68 }
69
70 /*
71  * Cette fonction change la position du curseur d'ecriture de l'ecran
72  */
73 void ecranSetPosCursor(short pos)
74 {
75     ecranControl(GPIO_PIN_7 | pos);
76 }
77
78 /*
79  * Cette fonction permet d'ecrire un caractere sur l'ecran
80  */
81 void ecranWriteChar(char caractere)
82 {
83     volatile unsigned long ulLoop;
84     ecranAttend();
85     ECRAN_DATA = caractere;
86     ECRAN_CTRL &= ~(ECRAN_RW); //RW = 0
87     ECRAN_CTRL |=  ECRAN_RS; //  rs = 1
88     ECRAN_CTRL ^=  ECRAN_EN; //En = 1
89     ulLoop = SYSCTL_RCGC2_R;
90     for (ulLoop = 0; ulLoop < 200; ulLoop++)
91     {
92         //petit delai, juste pour etre sur
93     }
94     ECRAN_CTRL ^= ECRAN_EN; //En = 0
95 }
96
97
98 /*
99  * Cette fonction permet d'ecrire une chaine de caracteres sur l'ecran
100  */
101 void ecranWriteLine(char * line, unsigned short size) //todo
102 {
103     unsigned short i=0;
104     for(i=0; i <size;i++)
105     {
106          ecranWriteChar(line[i]);
107     }
108 }
```