École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

5e année

2012 - 2013

Rapport de projet de fin d'études

Résolution de labyrinthe avec le robot humanoïde Nao

Encadrants
Jean-Louis BOUQUARD
jean-louis.bouquard@univ-tours.fr
Pierre GAUCHER
pierre.gaucher@univ-tours.fr

Étudiants
Damien DETOEUF
damien.detoeuf@etu.univ-tours.fr

DI5 2012 - 2013

Université François-Rabelais, Tours

Version du 30 avril 2013

Table des matières

TABLE DES MATIÈRES

TABLE DES MATIÈRES

Table des figures

TABLE DES FIGURES

Liste des tableaux

Remerciements

Je tiens à remercier mes encadrants, M. Bouquard et M. Gaucher, pour leur disponibilité, leur aide et leurs précieux conseils tout au long de mon projet de fin d'études. Merci d'avoir bien voulu m'encadrer sur ce projet.

Je tiens à remercier M. Rousseau pour l'aide qu'il m'a apportée lors de la prise en main du robot Nao et pour sa disponibilité en tant que référent sur le robot Nao.

Je tiens à remercier M. Mayaud pour son aide lors de la construction du labyrinthe, pour sa disponibilité et pour sa gentillesse.

Je remercie Julien Téruel ainsi que Guillaume Lastecoueres, camarades de promotion, pour leur écoute attentive et les précieux conseils qu'ils m'ont prodigués.

Je remercie Laura, ma concubine, pour sa présence, son soutien, son aide et l'attention qu'elle m'a apportés tout au long de ce projet.

Je tiens à remercier Matthieu Anceret et Florian Lissandres, camarades de promotion, pour ces bons moments passés à travailler nos projets de fin d'études en salle chaîne de production.

Enfin, je remercie tout mon groupe d'amis pour ces bons moments passés ensemble durant les trois années de notre cursus.

Introduction

Au cours de leur dernière année d'étude, les élèves de l'école Polytech Tours sont amenés à travailler sur un projet d'envergure. En début d'année, chacun des élèves se doit donc de choisir puis de réaliser un projet appelé "projet de fin d'études", projet qui sera travaillé tout au long de l'année scolaire.

Ce projet doit représenter l'aboutissement de la formation d'ingénieur : plus qu'une réalisation technique, c'est aussi un travail permettant de mettre en avant la capacité de l'élève à avoir une démarche d'ingénieur. L'élève ingénieur doit montrer qu'il est capable de mener un projet du début à la fin, en respectant des étapes primordiales comme les spécifications ou les réunions avec le client, il doit aussi montrer sa capacité de recherche, de réflexion et de travail. Le projet est supervisé par un encadrant, il n'y a pas de donneur d'ordre, l'élève reçoit son projet ayant un but estimé réalisable et ce même élève doit mettre à profit ses compétences pour atteindre ce but.

Mon projet de fin d'études est intitulé :"Résolution de labyrinthes avec le robot humanoïde Nao". Il s'agit de programmer le robot Nao afin qu'il soit capable de sortir d'un labyrinthe dont il n'a aucune connaissance. Pour que Nao soit capable de sortir du labyrinthe, ce dernier sera modélisé sous la forme d'un graphe. Le graphe sera modélisé au fur et à mesure que le robot explorera le labyrinthe, le but étant de pouvoir utiliser le graphe afin de trouver la sortie du labyrinthe. Pour résoudre ce problème, un module autonome sera développé en C++ via cross-compilation. Ce module autonome, une fois déposé dans la mémoire du robot, permettra à Nao de sortir du labyrinthe. Ce projet est le premier du genre sur le robot Nao. Ce dernier ayant été récemment acheté par l'école, aucun projet de programmation de module n'a eu lieu. Les objectifs sont les suivants : apprendre à programmer et manipuler le robot Nao, et expérimenter des méthodes de résolution algorithmique de graphes.

Le robot humanoïde Nao

3.1 État de l'art

Il n'existe pas de projets exactement identiques au mien. Toutefois, on peut trouver des articles traitant de résolution de labyrinthe par des robots ou bien de traitement d'images avec des robots dans le but de se placer dans l'espace. J'ai pu trouver des articles traitant ces sujets sur la CSDL grâce à l'accès que Polytech Tours nous fournit.

En premier lieu, les auteurs de l'article intitulé : "Monocular visual self-localization for humanoid soccer robots" traitent de méthodes de traitement d'images afin qu'un robot humanoïde soit capable de se placer dans l'espace.

Ensuite, les auteurs de l'article : "A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory" parlent de résolution de labyrinthes avec des robots. Ils remarquent d'ailleurs qu'il est difficile de résoudre un labyrinthe inconnu de manière optimale sans méthode d'intelligence artificielle. Ils utilisent pour ce faire la théorie des graphes et plus particulièrement ils cherchent à prouver la supériorité des algorithmes de théorie des graphes par rapport aux algorithmes qui n'utilisent pas la théorie des graphes pour donner une intelligence artificielle à des robots afin qu'ils résolvent des labyrinthes.

3.2 Présentation de Nao

Nao est le nom du robot humanoïde créé par la société française Aldebaran Robotics. Présenté pour la première fois au public en 2006, Nao est un robot programmable et autonome. Il existe plusieurs versions de Nao, Polytech Tours a fait l'acquisition de la version H25. Cette version est complète (contrairement à certaines qui n'ont pas les jambes, ou qui n'ont que le torse du robot), et dispose de 25 degrés de liberté.

Ce robot est un outil très intéressant dans le domaine de la robotique et de l'aide à la personne puisqu'il est complètement programmable. Cependant Nao n'a pas encore toutes les fonctionnalités qui permettraient d'assister des personnes en situation de handicap. Néanmoins, Aldebaran Robotics travaille à créer le petit frère de Nao : il s'appellera Romeo, mesurera 140 cm et devrait reprendre les fonctionnalités de Nao pour être un robot destiné à l'aide aux personnes en perte d'autonomie. Enfin, outre le fait que Nao puisse être un parfait compagnon de jeu pour les enfants, il est aussi un excellent sujet de travail pour les universités et les chercheurs.

FIGURE 3.1 – Un exemplaire de Nao version H25

3.3 Capacités matérielles et logicielles

Le robot Nao est intéressant de par toutes les possibilités offertes par ses capteurs. Il dispose de caméras, de micros, de haut-parleurs, de doigts permettant la préhension, de capteurs tactiles sur les mains et la tête, de sonars, de capteurs de pressions sous les pieds, de bumpers (boutons poussoirs) sur l'avant des pieds, et d'un gyroscope lui permettant de tenir en équilibre.

Tous ces capteurs donnent la possibilité d'utiliser Nao pour réaliser de nombreuses choses. Par exemple, Nao a été choisi pour la RoboCup (tournoi international de robotique dont le but est de créer des équipes de football) à la place du robot Aibo de Sony. Nao est capable grâce à ses caméras de reconnaître et mémoriser des visages, des formes, des objets ; il est capable grâce à ses micros de reconnaître des sons et grâce à ses sonars et ses caméras, il peut se placer dans l'espace. Il peut aussi anticiper les chutes grâce aux capteurs de pression sous ses pieds et à son gyroscope. Ainsi en cas de chute, ces mécanismes lui permettent de disposer d'un réflexe (placement des bras de façon à se protéger des chocs). Pour finir, Nao peut entendre et reconnaître des mots ou des phrases, mais il est aussi capable de s'exprimer : il peut parler en plusieurs langues (français, anglais, chinois) grâce à ses haut-parleurs.

Nao est un robot, mais il s'assimile aussi à un ordinateur. Il dispose de disques durs, de mémoire vive, de deux processeurs de type Intel Atom et d'un système d'exploitation personnalisé. Le système d'exploitation installé dans la mémoire de Nao s'appelle OpenNao. Il s'agit d'un système d'exploitation de type Linux, basé sur une distribution de type Gentoo. On peut donc utiliser sur Nao la plupart des fonctionnalités offertes par une distribution Linux (SSH, gestion de la mémoire, gestion des dossiers, etc). Nao dispose aussi de trois logiciels importants : Choregraphe, Naoqi et Qibuild. Choregraphe n'est pas un logiciel installé sur Nao, c'est un logiciel qui s'installe sur un ordinateur. Il permet d'utiliser les capacités de Nao en faisant de la programmation graphique grâce à des "boîtes" représentant des comportements définis. Le fonctionnement du logiciel sera expliqué ci-dessous. Naoqi est le nom du logiciel principal de Nao. Il est exécuté en permanence sur Nao et permet de le contrôler. Le framework Naoqi est le framework de

programmation utilisé pour programmer sur Nao. Il permet d'accéder aux différentes ressources de Nao (capteurs, moteurs, ...), il permet le parallélisme, la synchronisation et les évènements. Qibuild permet de compiler du code pour Nao et permet la gestion de projets dédiés à Nao. Ce logiciel gère les dépendances entre projets et permet la compilation croisée. Il est possible de lui donner des chaines de compilation (ce sont des paquets pré compilés) spécifiques afin de programmer pour un système bien précis. Qibuild est basé sur CMake. Après avoir défini ces logiciels, nous pouvons parler de la programmation sur Nao.

3.4 Programmation sur Nao

Au fil de mon étude, par analyse personnelle j'ai pu distinguer trois niveaux de programmation sur Nao.

Au premier niveau, on peut programmer Nao grâce à Choregraphe. Choregraphe est un logiciel développé par Aldebaran Robotics, il permet de faire de la programmation graphique. Dans ce logiciel, des comportements sont déjà programmés : se lever, s'asseoir, marcher sur une certaine distance, etc. et sont modélisés chacun par une "boîte" (ex : la boîte "stand up" correspond à l'action "se lever"). Il s'agit donc de placer sur une aire la "boîte" représentant le comportement souhaité afin que le robot l'effectue. Il y a aussi possibilité de chaîner les boîtes entre elles afin d'obtenir une succession de comportements. Par exemple : se lever, marcher sur deux mètres, dire bonjour puis s'asseoir. Ce principe permet alors d'aboutir à des comportements complexes.

Au deuxième niveau de programmation, nous utilisons toujours Choregraphe mais nous ne nous limitons plus à l'utilisation des boîtes existantes : il s'agit de créer ses propres boîtes afin d'obtenir un nouveau comportement. Il faut savoir qu'une boîte peut soit correspondre elle même à une succession d'autres boîtes, soit correspondre à un script Python (ce script est la traduction écrite des étapes permettant au robot d'effectuer le comportement). Il y a donc possibilité de créer de nouvelles boîtes en les programmant en Pyhton pour ensuite les utiliser dans Choregraphe.

Au troisième niveau de programmation, nous n'utilisons plus du tout le logiciel Choregraphe. Nous utilisons désormais le framework Qibuild. Comme expliqué précédemment, ce framework permet de compiler du code, de gérer les dépendances et permet également la compilation croisée. Grâce à cet utilitaire, nous disposons maintenant de commandes permettant la compilation de code compréhensible par Nao. Pour cette étape, le but est de créer un module indépendant et autonome exécutable sur le robot. Il s'agit donc tout d'abord de créer un arbre de travail (ou worktree en anglais) avec Qibuild. Ensuite, il faut créer son propre projet et y insérer les sources du module que l'on souhaite créer. Je rappelle que ces trois niveaux sont de mon analyse personnelle, le troisième niveau correspond donc pour moi à savoir comment créer un module indépendant et autonome pour Nao. Après avoir crée le projet et ajouté les sources, il faut donc récupérer la chaîne de compilation correspondant à la version du robot sur lequel nous souhaitons développer. Une fois cette chaîne de compilation obtenue, il faut l'utiliser pour créer une chaîne de compilation avec Qibuild. Quand cette étape est validée, il ne reste plus qu'à compiler le projet à l'aide de Qibuild et l'exécutable sera crée. Le détail de toutes ces étapes est écrit et expliqué dans le document que j'ai inséré en annexe et qui s'intitule "Méthode pour créer un module autonome sur le robot humanoïde Nao".

Une fois ces bases posées, je vais présenter l'approche que j'ai adoptée pour résoudre le projet.

Approche et résolution du problème

La finalité du sujet était donc de faire en sorte que le robot Nao soit capable de sortir d'un labyrinthe. Je n'avais pas d'instructions précises pour réaliser le projet, juste Nao et ses possibilités. J'ai donc réfléchi à comment résoudre le problème. Dans un premier temps j'ai pris des décisions au sujet de l'utilisation de Nao, dans un second temps j'ai choisi une méthode algorithmique et enfin il m'a fallu prendre en compte un problème de traitement d'images.

4.1 Utilisation de Nao

Nao dispose de vingt-cinq degrés de liberté et de nombreux capteurs. J'ai donc dû prendre en main le robot au début du projet et réfléchir à comment l'utiliser pour arriver à mes fins. Il me fallait quelque chose qui me permette d'appréhender l'environnement immédiat du robot et pouvoir prendre des décisions sur les actions à faire. Je me suis donc orienté vers les sonars du robot et vers sa caméra. Mon problème était de savoir quel capteur entre la caméra et les sonars serait le plus apte à m'aider pour résoudre le labyrinthe.

Le capteur que j'utiliserais devrait me renvoyer une information immédiate sur la présence d'un obstacle ou l'absence d'un obstacle, sur la présence d'un croisement et le nombre de chemins sortant du croisement, sur la présence d'un cul-de-sac, etc. En premier lieu je me suis orienté vers l'utilisation des sonars. Ils me donnaient une information immédiate sur la localisation de l'obstacle par rapport au robot (droite, gauche, ...) et la distance de l'obstacle. Toutefois, en faisant des tests je me suis rendu compte que les informations renvoyées par les sonars étaient précises, mais tout de même pauvres. En effet, ils me signalent la présence d'un obstacle, mais si jamais il y a un croisement, les informations que je peux exploiter depuis les sonars sont bien trop pauvres pour pouvoir être utilisées afin de déduire la présence d'un croisement et d'autant plus inexploitables pour connaître le nombre de sorties du croisement. L'image suivante présente un exemple de parcours du labyrinthe. Ici, Nao n'utiliserait que ses sonars pour se repérer. Le point orange représente Nao, les deux flèches latérales sont les sonars et la flèche centrale représente l'orientation de Nao. On voit clairement qu'il y un problème, le robot est légèrement orienté vers la droite, le sonar droit détecte le mur à la droite de Nao et le sonar gauche ne détecte pas d'obstacles. Ces informations sont claires, mais elles ne nous permettent pas de prendre une décision sur la présence ou non d'un croisement (voir schéma 4.1). C'est pourquoi dans un second temps, je me suis orienté vers l'utilisation seule de la caméra de Nao pour lui faire appréhender son environnement.

L'utilisation de la caméra de Nao est simple : on peut soit lui faire prendre des photos ou bien obtenir un flux vidéo. J'ai donc commencé à réfléchir à l'utilisation de la caméra pour Nao. En partant du principe qu'on obtient une image de l'environnement immédiat du robot, je pensais pouvoir exploiter cette image pour pouvoir faire prendre des décisions au robot. Il m'aurait suffi de traiter l'image et d'en extraire les

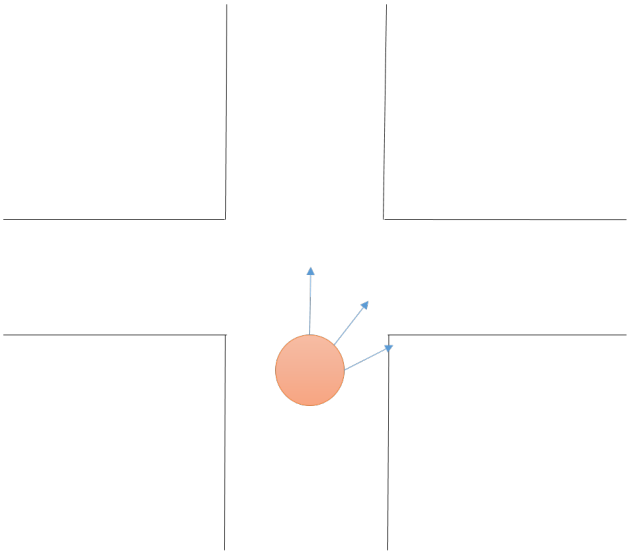

FIGURE 4.1 – Nao dans le labyrinthe à l'approche d'un croisement qui se repère grâce à ses sonars

décisions voulues. Je ne connaissais pas encore la méthode, mais j'en connaissais la faisabilité. Toutefois, à force de recherches, je me suis rendu compte que cette solution serait très compliquée et longue à mettre en place. Détecter des objets, des formes, des changements de couleur est aisé, mais pour s'en servir afin d'évaluer des distances, afin d'évaluer une orientation ou encore un croisement en tant que tel, cette méthode devient très vite compliquée. C'est pourquoi au lieu de choisir l'une ou l'autre des techniques j'ai décidé d'utiliser les deux.

En effet, je suis arrivé à la conclusion qu'utiliser les sonars seuls ou bien utiliser la vidéo seule ne donnerait pas de bons résultats. J'ai donc décidé d'utiliser ces deux méthodes en combinaison afin de résoudre mon problème. Dans un premier temps, les sonars me serviront à détecter les obstacles proches afin que Nao puisse évoluer en sécurité dans le labyrinthe. A chaque fois qu'un des sonars détecte un obstacle, la distance de l'obstacle est récupérée et Nao tourne selon un certain angle afin d'éviter l'obstacle. Dans un deuxième temps, la caméra me servira à analyser l'environnement de Nao afin de lui faire prendre des décisions. Toutes les secondes, une photo sera prise par Nao. Cette photo sera analysée en interne par Nao, l'image sera soumise à plusieurs algorithmes de traitement d'images qui permettront de mettre en valeur les caractéristiques qui nous intéressent. La partie traitement d'images sera développée ci-après.

Une fois ces déductions faites, il me fallait réfléchir à comment utiliser les informations renvoyées par ces capteurs, comment mettre en place une intelligence capable de tirer parti de l'utilisation de ces capteurs afin de faire évoluer Nao dans le labyrinthe.

4.2 Résolution algorithmique

Avoir des informations fiables et utilisables provenant des différents capteurs de Nao était une première étape, la deuxième étape était de mettre en place une intelligence assez élaborée pour permettre à Nao d'avancer dans le labyrinthe de manière intelligente. Par là j'entends qu'il m'a fallu réfléchir à une méthode

qui, au lieu de faire parcourir de manière erratique le robot dans le labyrinthe, tirerait parti de l'exploration déjà effectuée afin d'éviter de prendre des chemins déjà parcourus ou des directions qui ne mènent forcément pas vers la sortie. J'ai donc décidé d'implémenter une résolution algorithmique afin de pouvoir être capable de donner un comportement intelligent à Nao.

C'est donc sous les conseils de M. Bouquard que je me suis lancé dans la réflexion d'une solution algorithmique qui me permettrait de réaliser ce que je souhaite : faire en sorte que Nao apprenne de ses différents déplacements et surtout qu'il soit capable d'exploiter les informations qu'il a enregistrées. Pour arriver à nos fins, nous avons donc décidé d'utiliser la théorie des graphes. Le labyrinthe sera représenté sous la forme d'un graphe connexe. Le graphe sera créé par Nao au fur et à mesure de son avancée dans le labyrinthe et lui permettra de s'orienter dans ce dernier. Chaque croisement sera représenté par un noeud du graphe, et les chemins qui mènent à ces croisements ou en partent seront représentés par des arêtes du graphe. Les culs-de-sac seront représentés par des noeuds feuilles.

Ainsi, à chaque croisement qu'il rencontrera, Nao créera un noeud dans un graphe qu'il conservera dans sa mémoire. Les noeuds auront donc au minimum une arrête entrante et deux arrêtes sortantes, et au maximum une arrête entrante et trois arrêtes sortantes. Comme Nao aura en mémoire un graphe construit des parties du labyrinthe explorées, il devient possible d'utiliser des algorithmes de résolution de graphes afin de pouvoir décider si une sortie est plus intéressante à emprunter qu'une autre. Pour la résolution du graphe du labyrinthe, j'ai donc décidé d'utiliser un parcours en profondeur d'abord. Ainsi, à chaque fois que Nao arrive à un croisement qu'il a déjà parcouru, il analyse le graphe existant grâce à un algorithme de parcours en profondeur d'abord. Cela lui permet donc de savoir si une sortie du croisement est plus intéressante qu'une autre. En effet, si lors de l'analyse du graphe Nao se rend compte qu'une des sorties mène à un cul-de-sac, il ne faut pas l'emprunter. Au contraire, si une des sorties mène à un endroit du graphe inconnu, il y a potentiellement une chance que cette sortie mène vers la sortie du labyrinthe. Toutefois, à force de réfléchir au problème, je me suis rendu compte de la chose suivante : on peut identifier chaque noeud du graphe par un numéro ou un nom, mais si Nao tourne en rond, sans plus d'informations je suis incapable de le détecter.

En effet, un noeud dans le graphe est créé à chaque fois que le robot rencontre un nouveau croisement, mais je ne dispose d'aucune information sur la localisation de ce croisement et je ne sais pas s'il est vraiment nouveau. Avec le graphe seul, nous sommes dans l'incapacité de savoir si un croisement rencontré est nouveau ou pas. C'est pourquoi j'ai eu l'idée de mettre en parallèle à la création du graphe la création d'un repère en deux dimensions lors de l'exploration du labyrinthe. Ce repère sera marqué par des points à chaque évènement remarquable détecté. Ainsi, lorsque le robot rencontrera un nouveau croisement, il marquera un repère deux dimensions d'un point qui représentera la localisation du croisement dans le labyrinthe. Ce repère sera vérifié à chaque fois que Nao rencontrera un croisement afin de savoir si le croisement a déjà été rencontré ou non. Le robot se localisera lui aussi sur ce repère afin de savoir où il se trouve dans le graphe. Il avancera sa position dans le repère lorsqu'il se déplacera. Nao sera donc aussi en capacité de relier les différents points du repère qu'il parcourt.

Grâce à l'association d'un graphe et d'un repère deux dimensions, j'ai donc pu mettre en place un comportement intelligent pour Nao. Le graphe créé permet à Nao de pouvoir se repérer dans le labyrinthe, pouvoir décider si une sortie est plus intéressante qu'une autre grâce à un parcours de graphe en profondeur d'abord. Le repère 2D permet lui au robot d'avoir une représentation dans l'espace de son environnement, cela lui permet d'éviter de créer un noeud du graphe en double et de tourner en rond indéfiniment. Une fois la théorie mise en place, je savais quels capteurs j'allais implémenter ainsi que quel comportement j'allais donner au robot. Une fois ces prérequis posés, je me suis lancé plus loin dans la réflexion et j'ai remarqué que la partie traitement d'images serait plus difficile à appréhender que je ne le pensais.

4.3 Traitement d'images

Cette partie n'avait pas été envisagée dès le début, mais elle s'est révélée être une partie très importante au fil de l'avancement du projet. En effet, la résolution du problème repose pour beaucoup sur le traitement et l'exploitation des images récupérées par Nao. J'ai cru que cette partie serait facilement solvable, mais elle s'est avérée plus difficile que prévu. Il me fallait être capable d'interpréter les images récupérées depuis la caméra de Nao. Après réflexion et en obtenant des images comme celles-ci 4.2, j'ai décidé de m'orienter vers des solutions de détection de contours.

FIGURE 4.2 – Exemple de photographie prise par les caméras de Nao dans le labyrinthe

En obtenant une telle image, je voulais pouvoir détecter le bord des planches et la délimitation avec le sol. Cela me permettrait d'obtenir dans la mémoire du robot une information claire sur son environnement et non plus juste une image. Or, je n'avais pas de grandes connaissances en traitement d'images. Avec l'aide de M. Gaucher, j'ai pu étudier des algorithmes de traitement d'images efficaces pour ce genre de problèmes. J'ai donc utilisé une librairie C++ contenant des fonctions qui implémentent différents algorithmes de traitement d'images, dont ceux que m'a présentés M. Gaucher. Ces outils m'ont permis d'obtenir plusieurs

résultats. Ma démarche sera présentée dans la partie résultats du rapport.

4.4 Programmation

Pour programmer sur Nao, il faut utiliser le framework Naoqi. Voici une explication de ce qu'il permet et de comment Aldebaran Robotics a pensé la programmation sur Nao.

Naoqi supporte plusieurs langages, mais j'ai développé en C++.

En tout premier lieu, ce qui est exécuté par Naoqi est un broker. Le broker est un objet qui permet deux choses : il permet de trouver les modules et les méthodes chargées de base par Nao et il permet les accès réseau (appeler des méthodes en dehors du processus courant). Les broker sont pensés pour être transparent pour le développeur, ce dernier n'aura pas à s'en soucier, peu importe s'il développe pour un module exécuté en local ou un module exécuté à distance.

Ensuite, on trouve les proxys. Les proxys sont des objets qui permettent d'accéder aux méthodes d'un module existant. Par exemple, si on déclare un proxy sur ALSonar nous disposerons d'un accès aux méthodes permettant de manipuler les sonars.

Enfin, nous avons les modules. Ils se déclinent en deux possibilités : les modules locaux ou les modules distants. Un module local est exécuté depuis la mémoire de Nao, un module distant est exécuté depuis une machine distante.

4.5 Structure de la solution

En définitive, j'ai pris plusieurs décisions qui m'ont servi à mettre en place une structure de solution. Lors du parcours du labyrinthe, le robot commence donc par appréhender son environnement. Pour ce faire, il utilise les différents capteurs à sa disposition : sa caméra et ses sonars. Les sonars lui permettent d'éviter les obstacles immédiats et la caméra lui permet d'obtenir des informations sur le labyrinthe. Une fois que Nao dispose de ces informations, il lui faut les analyser pour pouvoir effectuer une démarche intelligente. C'est pourquoi il faut d'abord qu'il commence par traiter les images obtenues par la caméra à l'aide d'algorithme de détection de contours. Une fois les images traitées, il faut les analyser pour pouvoir en tirer des décisions. A partir des images traitées, je peux donc utiliser une résolution algorithmique. C'est dans cette optique que je mets en place un graphe et un repère deux dimensions au fur et à mesure que Nao explore le labyrinthe.

La solution est donc théoriquement structurée ainsi, nous allons maintenant voir comment j'ai pensé cette solution de manière pratique.

Réalisation

Dès le début du projet, je me suis toujours posé comme règle d'avoir une démarche d'ingénieur. C'est-à-dire que je n'ai jamais cherché à obtenir de mes encadrants des informations précises sur ce que je devais faire, je ne les sollicitais que pour leur rapporter mon travail ou pour leur demander des éclairages sur certaines parties du projet. J'ai toujours souhaité mener le projet à ma manière, en respectant bien sûr les demandes de mes encadrants qui sont censés représenter les donneurs d'ordre. Pour mon projet, je partais du principe que j'étais encadré, mais libre dans mes choix. Je me suis toujours posé comme règle qu'il n'y a pas de bonnes ou de mauvaises décisions pour l'avancement de mon projet, seulement des choix à faire de manière rationnelle et réfléchie, et surtout des choix qui peuvent être justifiés et soutenus afin de montrer leur nécessité à l'avancement du projet. C'est dans cette optique que j'ai mené mon projet de fin d'études. Pour cette partie de mon rapport, je vais présenter les différentes étapes de mon travail, ce que j'ai réalisé tout au long de l'année.

5.1 Gestion de projet

Avant de commencer à parler de mon projet, je vais parler de la manière dont j'ai géré mon projet. J'ai fixé des règles de gestion précises afin de ne jamais perdre pied dans ce que je faisais et de toujours garder à l'esprit ce qui était important. Pour chaque semaine de PFE, je me fixais un objectif à réaliser. Mes encadrants m'ont demandé de rendre compte de mon travail après chaque semaine de PFE. Je me servais donc de ces comptes rendus hebdomadaires afin de fixer clairement ce que j'avais à faire, ce que j'avais fait et ce qu'il me restait à faire. En parallèle de ça, j'ai rédigé des spécifications pour mon projet. Elles m'ont servi à mieux cerner mon projet et ce que j'avais à faire pour avancer. J'ai aussi établi un diagramme de Gantt à suivre afin d'étaler mon travail sur toute l'année. Voici un exemple de rapport hebdomadaire type :

5.1.1 Exemple de rapport hebdomadaire

Projet de fin d'études : résolution de labyrinthe avec le robot humanoïde NAO

Rapport hebdomadaire de Damien DETOEUF DI5.

Rapport de la semaine du 18 mars 2013

Cette semaine, je me suis concentré sur la partie du développement concernant le traitement vidéo. Il s'agit de récupérer une image, la traiter et en extraire des informations pour prendre une décision sur l'exploration du labyrinthe et la tenue du graphe.

J'ai donc mené une réflexion afin de pouvoir prendre des décisions de développement. Pour m'aider à y voir plus clair, j'ai lu quelques articles et j'ai pris rendez-vous avec M. Ramel afin de lui exposer mon

problème. J'ai pu avoir de nombreuses réponses à mes questions lors de ce rendez-vous. M. Ramel m'a assuré qu'adapter le labyrinthe simplifierait la tâche, mais je lui ai répondu que nous ne souhaitons pas adapter le labyrinthe au robot. M. Ramel m'a aussi demandé de réaliser des photos d'exemples de prises de vue depuis Nao dans le labyrinthe afin de voir comment est-ce que nous pourrions chercher des points d'intérêt dans les images dans le but de les traiter.

Pour ce qui est de la résolution et des pistes de recherches, M. Ramel m'a orienté vers la librairie C++ OpenCV, il existe des implémentations d'algorithmes qui pourraient nous être utiles et certains sont robustes aux conditions de luminosité ou autres problèmes qui pourraient se poser. Après discussion avec M. Gaucher, nous allons essayer des techniques existantes qui pourraient être concluantes sur la détection des points d'intérêts dans les images telles qu'une détection de contour par masque de convolution ou bien une détection de lignes droites avec une transformée de Hough. Mardi j'essayerais d'aller voir M. Bouquard pour faire un point sur le PFE.

La semaine prochaine je vais donc continuer l'exploration des solutions de traitement dans les images.

Ce qui était à faire pour la semaine du 18 mars :

- Trier et étudier les articles trouvés, EN COURS
- Rencontrer un professeur spécialiste en traitement d'images pour lui poser des questions sur le traitement d'images, FAIT
- Réaliser un premier traitement d'images dans la mémoire de Nao. EN COURS

A faire pour la semaine du 25 mars :

- Trier et étudier les articles trouvés,
- Continuer à étudier les solutions de traitement d'images et essayer d'en mettre en place dans la mémoire de Nao,
- Etudier OpenCV et surtout vérifier que cette librairie est bien utilisable dans la mémoire de Nao,
- Prendre rendez-vous avec M. Gaucher pour étudier des solutions de traitement d'images.

Les spécifications, les rapports hebdomadaires et le diagramme de Gantt m'ont toujours aidé à garder une ligne directrice quand je travaillais mon PFE. J'ai toutefois toujours gardé un regard critique sur ce que je faisais et comment je le faisais. A chaque créneau de PFE j'essayais d'avancer, mais tout ne se passe toujours pas comme on le souhaite, c'est pourquoi en milieu d'année j'ai revu mon diagramme de Gantt afin de réaliser un diagramme qui serait plus proche de la réalité. J'ai donc travaillé toute l'année en gardant à l'esprit à chaque créneau ce que je devais achever. Je vais maintenant parler de la réalisation de mon projet.

5.2 Prise en main du robot

La première difficulté a été de prendre en main le robot Nao. Ce dernier n'est pas simple à comprendre de prime abord, et je me suis donc laissé du temps pour explorer toutes ses fonctionnalités. J'ai aussi reçu une aide de la part de M. Rousseau qui a pris le temps de me former sur les différentes fonctionnalités du robot sous Choregraphe. J'ai aussi pu lire les rapports et la documentation technique rédigée par un groupe de

projet collectif de l'année 2011-2012. Ces documents étaient instructifs, mais ils se limitaient à l'explication de comment utiliser Nao via Choregraphe. J'ai donc examiné les différentes fonctionnalités de Nao à l'aide de Choregraphe. Au fur et à mesure de mes tests, je me suis rendu compte que Choregraphe est un logiciel très complet et qui permet de faire des comportements complexes et riches. Mais, ce logiciel impose certains comportements lorsqu'on souhaite utiliser les capteurs. Par exemple, sous Choregraphe, la fonctionnalité qui permet d'utiliser les sonars lève un évènement lorsque rien n'est détecté et ne renvoie pas de distance de détection. Pour ma part, j'aurais souhaité avoir un évènement lorsque les sonars détectent quelque chose et pouvoir récupérer la distance entre l'obstacle et le robot. J'en suis donc arrivé à la conclusion que pour la réalisation de mon projet, je n'utiliserais pas les composants de base de Choregraphe.

Toutefois, il est aussi possible de créer ses propres comportements sur Nao en utilisant Choregraphe. En effet, nous pouvons écrire des comportements (des "boîtes" sous Chorégraphe) en utilisant le langage Python. Cela permet de se faciliter la tâche en implémentant certaines fonctions qui n'auraient pas été utilisables si nous nous limitions aux comportements de base, et cela permet surtout d'utiliser les comportements créés en combinaison avec les comportements existants. Disposant de ces informations, j'ai fait un nouveau choix. En effet, il existe un troisième niveau de programmation : il s'agit de programmer ses propres modules indépendants en utilisant le langage C++. J'ai pris la décision de choisir ce style de programmation afin de résoudre le problème : créer un module indépendant en C++. De plus, j'étais plus familier avec le langage C++ qu'avec le langage Python.

Je souhaitais réaliser un module indépendant en C++. En effet, le robot étant nouvellement acquis par l'école, personne ne savait comment le programmer à l'aide du C++ sans utiliser Choregraphe. Choregraphe ne répondait pas complètement à mes exigences de programmation du fait de la limitation imposée par certains comportements déjà existants. Mais outre ce fait, le défi de découvrir comment programmer Nao me motivait beaucoup. En accord avec mes encadrants, j'ai donc décidé de continuer sur cette voie.

5.3 Compilation d'un module indépendant en C++ sur Nao

Ce fut une des parties importantes de mon projet. J'ai donc utilisé toutes les documentations qui étaient à ma disposition et en majeure partie celle fournie par Aldebaran Robotics. Ce fut une expérience très intéressante, car la compilation d'un module indépendant en C++ pour Nao n'est pas des plus aisée. Cette étape était d'autant plus importante qu'elle représentait un jalon nécessaire au bon déroulement de mon projet. En effet, si je n'arrivais pas à créer ce module, je ne pouvais pas faire fonctionner Nao comme je l'entendais. Il me fallait donc valider cette étape.

J'ai donc commencé par rechercher un maximum d'informations afin de pouvoir créer ce module et surtout des exemples de code (Aldebaran fournit des exemples très complets dans son SDK). En tout premier lieu, il me fallait récupérer toutes les briques nécessaires à la création de ce module. Pour commencer, il y a plusieurs types d'exécutables qui peuvent être créés : des modules exécutables à distance, c'est-à-dire que le robot est esclave d'un ordinateur distant qui lui envoie ses ordres, des modules exécutables dans la mémoire de Nao, c'est-à-dire qu'il suffit de faire exécuter le module à Nao et il sera complètement autonome. Le but du projet était de faire en sorte que Nao résolve le labyrinthe, il pouvait donc être

esclave d'un ordinateur distant. Toutefois, ce n'était pas mon intention. Je souhaitais vraiment faire en sorte que le robot soit indépendant de tout matériel, et j'avais la possibilité de le faire. Mon choix s'est donc naturellement orienté vers une solution de type module indépendant.

De par ce choix, certaines conditions devenaient obligatoires. En effet, pour pouvoir réaliser un module indépendant sur Nao, il faut absolument rassembler tous ces prérequis :

- Développer le module en Python ou en C++,
- Obligatoirement compiler depuis une plateforme de type Linux,
- Utiliser une chaîne de compilation adaptée au processeur du robot fournie par Aldebaran Robotics,
- Utiliser la bibliothèque logicielle fournie par Aldebaran Robotics,
- Compiler en utilisant qibuild.

Expliquons maintenant ces prérequis point par point. Nao est programmable selon de nombreux langages : Java, Matlab, .NET, Urbi, C, C++ et Python. Toutefois, seuls deux d'entre eux sont utilisables pour réaliser un module compilable et exécutable en mémoire : C++ et Python. C'est un choix fait par Aldebaran Robotics. Ensuite, en ce qui concerne la compilation, on parle ici de compilation croisée (ou "cross-compilation" en anglais). Cela signifie que nous compilons et créons un exécutable depuis un système d'exploitation A afin qu'il soit exécuté sur un système d'exploitation B. Ce processus requiert des règles de compilation correspondant au système d'exploitation cible. Ici, nous devons donc compiler et créer un exécutable qui devra s'exécuter sur Nao. Nao est comme un ordinateur, il dispose d'un système d'exploitation qui lui est propre et qui s'appelle OpenNao. C'est une distribution Linux de type Gentoo, c'est pourquoi afin de réaliser un exécutable qui pourra fonctionner sur Nao, nous devons effectuer une compilation croisée depuis une plateforme Linux. Nous créons donc un exécutable depuis un système d'exploitation Linux vers un système d'exploitation de type OpenNao. Pour ma part, je compilais depuis une distribution de type Linux Mint.

Ainsi, il faut obligatoirement compiler notre code depuis une plateforme Linux. Mais il faut aussi avoir les règles de compilation nécessaires à la compilation croisée. Ces règles sont spécifiques à un type de processeur et à une version. Chaque Nao dispose d'un processeur en particulier et d'une version, pour le Nao de Polytech Tours, le processeur est de type Intel Atom et la version du système est 1.14. Par exemple, lorsque j'ai récupéré Nao au début de mon projet, il n'était qu'en version 1.12. Au cours du projet je l'ai mis à jour, de ce fait les chaînes de compilation ont changé. Je suis passé d'une chaîne pour OpenNao de version 1.12 pour processeur Intel Atom à une chaîne pour OpenNao de version 1.14 pour processeur Intel Atom.

Les deux derniers points font partie des prérequis, mais nous amènent aussi à parler du développement du code du module. Pour pouvoir programmer efficacement, Aldebaran fournit un kit de développement (SDK ou software development kit en anglais) ainsi qu'un utilitaire de compilation. En effet, même si nous programmons en nous dédouanant de Choregraphe, il est inutile de devoir réimplémenter chacune des fonctions de base de Nao. Aldebaran nous fournit un SDK très utile nous permettant de disposer déjà de fonctionnalités de base comme la marche, l'utilisation des caméras, l'utilisation des sonars, les postures, etc. Ce SDK est donc indispensable, dans le sens où il nous permet d'accéder à des primitives

de base, mais aussi dans le sens où il est nécessaire au bon fonctionnement du module. En effet, le SDK contient tous les fichiers de configuration qui permettent au compilateur de comprendre et de créer les liens du programme. Par exemple, lorsque nous programmons sur Nao nous utilisons de nombreuses fois des "proxys" qui permettent d'accéder aux capteurs de Nao. La librairie C++ de base ne connaît pas ces types, mais avec le SDK le compilateur est capable de se retrouver.

Enfin, pour que le module fonctionne il faut utiliser l'utilitaire de compilation fourni par Aldebaran Robotics : qibuild. Ce dernier permet de compiler et de créer des exécutables pour Nao, et il permet surtout la compilation croisée afin de créer des modules indépendants sur Nao. Qibuild est aussi un utilitaire qui permet la création d'arbres de projets de type qibuild, il permet de manipuler plusieurs projets dans une arborescence de fichiers.

Connaissant tous ces prérequis, je me suis donc lancé dans l'écriture de mon propre module autonome. Je me suis beaucoup aidé de la documentation et des exemples fournis par Aldebaran, et réunir toutes les données permettant la compilation du module n'a pas été évident. Premièrement, je disposais d'un code correct, mais je ne possédais pas les chaînes de compilation, ni l'utilitaire de compilation et ni le sdk. En effet, ces fichiers sont propriétaires et appartiennent à Aldebaran Robotics, ils ne sont donc pas disponibles en téléchargement libre. De plus, Aldebaran Robotics est une société qui bien évidemment cherche à gagner de l'argent. La société vend des robots Nao mais propose aussi des formations permettant d'apprendre la programmation et l'utilisation de Nao. On peut donc facilement comprendre qu'il n'est pas dans leur intérêt que les personnes souhaitant programmer sur Nao puissent le faire en toute simplicité grâce à leur documentation. La documentation sur Nao existe et est accessible sur leur site internet, mais cette documentation reste parfois peu claire sur certains points. En ce qui concerne Polytech Tours, l'école a acquis toutes les licences nécessaires et peut donc utiliser tous les logiciels fournis par Aldebaran Robotics. Au début de mes essais, il me manquait plusieurs fichiers nécessaires comme la chaîne de compilation ainsi que l'utilitaire de compilation. Ce n'est que plus tard que j'ai compris que ces fichiers étaient disponibles sur un CD à l'intérieur de la mallette de Nao. J'ai aussi compris encore plus tard que l'école avait accès à un compte développeur grâce à un code situé au dos du livret qui est à l'intérieur de la mallette de Nao. Grâce à ce code, tous les fichiers nécessaires sont disponibles en téléchargement. Nous pouvons même télécharger le système d'exploitation OpenNao, c'est ainsi que j'ai fait la mise à jour de Nao de la version 1.12 vers la version 1.14. Cette nouvelle version me permettait d'accéder à des primitives très utiles, les postures de Nao (lui dire de se lever, de s'asseoir, de s'allonger, etc).

Une fois que je disposais de tous ces éléments (chaîne de compilation adaptée, utilitaire de compilation, sdk adapté à la mise à jour du robot), je pouvais compiler de manière correcte mon code afin de créer un module. La première difficulté que j'ai rencontrée a été le format de l'exécutable. En effet, il est possible de créer un exécutable ou bien une librairie interprétable par Nao. L'exécutable appelle ses propres fonctionnalités tandis qu'une librairie permet d'ajouter des fonctionnalités aux fonctionnalités déjà présentes sur la mémoire de Nao. Au début de mon travail, je me suis trompé dans mon approche et j'ai voulu créer un exécutable et une librairie en même temps alors que je n'avais pas besoin de créer une librairie. Or, la création de la librairie ne marchait pas, mais je n'avais pas encore saisi que je n'avais pas besoin de ça pour faire fonctionner mon exécutable. C'est au hasard d'une discussion avec Guillaume Lastecoueres un

de mes camarades de promotion que j'ai mis le doigt sur le problème. J'ai pu constater que la librairie m'était inutile, et à la suite de quelques modifications mon module s'est bien compilé correctement. C'était une étape importante de mon projet qui fonctionnait correctement, j'avais réussi à compiler et créer un exécutable indépendant.

Une autre étape intéressante était de savoir si le programme s'exécutait correctement. Je me suis donc connecté au robot via SSH et j'ai déposé l'exécutable dans la mémoire de Nao. En exécutant le programme à la main, j'ai pu constater qu'il fonctionnait comme souhaité. Ensuite, je souhaitais faire en sorte que le programme s'exécute sans même qu'on ai à se connecter au robot pour le faire. Aldebaran Robotics a anticipé le problème, il existe donc dans la mémoire de Nao un fichier ini qui est lu au démarrage du robot. Ce fichier s'appelle autoload.ini et se situe à ce chemin dans la mémoire de Nao : /home/nao/naoqi/preference/autoload.ini Si nous écrivons dans ce fichier le chemin de l'exécutable présent dans la mémoire de Nao, Nao va aller exécuter le module au démarrage de son système. Le robot est donc complètement autonome.

5.4 Développement du module

5.4.1 Spécifications

Une fois que je savais que j'avais trouvé comment compiler du code pour Nao et comment créer un exécutable, je pouvais me mettre à développer un premier module indépendant qui implémenterait les fonctions nécessaires à l'exploration du labyrinthe. J'avais donc besoin de développer les fonctions suivantes, fonctions que j'avais spécifiées dans mon cahier de spécification système :

- Parcours du labyrinthe,
- Acquisition et traitement vidéo,
- Gestion du graphe et du repère deux dimensions,
- Détection et évitement des obstacles,
- Gestion des chutes.

En partant de ces spécifications, j'ai donc cherché à développer les composantes qui me permettraient de réaliser ces fonctionnalités. J'ai donc commencé à écrire du code qui me permettrait de faire les actions suivantes :

- Récupérer les informations données par les sonars,
- Permettre à Nao d'éviter les obstacles,
- Prendre une photographie avec les caméras de Nao,
- Créer et gérer un graphe et un repère deux dimensions,
- Faire en sorte que Nao se relève après une chute.

5.4.2 Gestion des sonars et gestion des chutes

Pour réaliser ce programme, je me suis lancé dans l'étude de la documentation fournie par Aldebaran Robotics. Pour chaque composant que je voulais utiliser, il y avait une page de documentation correspondante qui détaillait toutes les fonctions implémentées dans le sdk fourni par Aldebaran Robotics. Je me

suis donc rendu compte que Nao savait gérer les évènements en C++. Par exemple pour les sonars, un évènement est levé à chaque fois que le sonar détecte un objet distant. De plus, à chaque fois que le sonar détecte un objet, en plus de lever l'évènement la distance à laquelle l'objet a été détecté est renvoyée. Ainsi, pour un développeur, il s'agit de déclarer dans son code que l'on se met à l'écoute d'un évènement bien particulier. Si on garde l'exemple des sonars, nous devons déclarer grâce à la méthode "bind" que nous nous mettons à l'écoute de l'évènement "onSonarRightDetected". Nous n'avons donc plus qu'à lier une des méthodes que nous avons écrites à l'évènement "onSonarRightDetected". Nous avons donc déclaré dans notre module que nous souhaitons être informés à chaque fois que l'évènement est levé. En conséquence, à chaque fois que l'évènement sera levé, Nao analysera la liste des modules qui sont inscrits à cet évènement et qui souhaitent recevoir un signal lorsque l'évènement se produit. Au final, le module recevra l'information que l'évènement a été levé et il recevra aussi les informations inhérentes à l'évènement. Comme dans notre module nous avons lié l'apparition de l'évènement à une de nos méthodes, lorsque l'évènement sera levé le code de la méthode que nous avons écrit sera exécuté. Ainsi, nous pouvons choisir de faire ce qu'il nous plait en réponse à l'évènement.

Disposant de cette gestion d'évènements, je pouvais donc gérer facilement les sonars. J'ai donc écrit deux méthodes qui sont appelées lorsque les sonars détectent un obstacle. Une méthode pour le sonar gauche et une autre pour le sonar droit. Ces méthodes récupèrent la distance détectée par le sonar les concernant. J'ai donc mis en place la première fonctionnalité de ma fonction de détection et d'évitement d'obstacles : la détection. Pour l'évitement, il me suffisait donc de récupérer la valeur renvoyée par le sonar et de faire tourner le robot en conséquence.

Pour ce problème, la question à se poser était la suivante : comment faire en sorte que le robot tourne en fonction de la distance renvoyée par le sonar ? Le but était d'obtenir un angle inversement proportionnel à la distance détectée. En effet, plus l'obstacle est éloigné, moins il faut tourner et à l'inverse, plus l'obstacle est proche, plus il faut tourner. J'ai donc décidé d'écrire une fonction mathématique qui prendrait en entrée la distance détectée et qui renverrait en sortie un angle acceptable inversement proportionnel à la distance détectée. La gestion des sonars et la génération des angles renvoyés pour faire tourner Nao ont été un des problèmes de mon projet, j'aborderais ce problème plus tard dans mon rapport.

Les évènements gérés par Nao m'ont beaucoup servi aussi en ce qui concerne la gestion des chutes. En effet, lorsque Nao tombe il est capable de le détecter : Nao dispose d'un réflexe en cas de chute qui lui permet de se protéger, peu importe la manière dont il tombe. Il est capable de se protéger avec ses bras et désactive tous ses moteurs afin d'éviter tout dommage physique. Je me suis donc fait la remarque que ce réflexe doit aussi lever un évènement lorsque Nao tombe. C'est bien le cas, pour pouvoir gérer les chutes il m'a donc fallu mettre mon module en écoute de cet évènement et écrire une méthode qui serait appelée en cas de chutes. Cette méthode a simplement pour but de faire aller Nao de sa position courante à la position debout puis de le faire avancer pour qu'il reprenne l'exploration du labyrinthe.

5.4.3 Gestion de la caméra et traitement d'images

Une fois ces fonctionnalités mises en place, je me suis lancé dans la partie du développement concernant la gestion de la caméra de Nao ainsi que du traitement d'images. Pour gérer la caméra, il ne s'agissait plus

de se mettre en écoute d'évènements. J'avais décidé de ne pas utiliser un flux vidéo pour la résolution, pensant que des photographies prises à intervalles réguliers seraient plus pertinentes et plus faciles à utiliser.

Utiliser la caméra de Nao est plutôt simple, il suffit de créer un proxy d'un certain type et nous avons accès aux méthodes de manipulation de la caméra. J'ai donc appelé une méthode permettant de faire des photographies à intervalles réguliers afin de tout d'abord commencer par faire des tests. Je me suis vite rendu compte d'une chose : l'appel à cette méthode est bloquant. C'est-à-dire qu'il interrompt toute action en cours non bloquante et prends la main pour s'exécuter : aucune autre méthode ne peut s'exécuter en même temps. Il s'agit d'une section critique. Pour l'utilisation de la caméra, j'ai donc décidé d'utiliser des threads. C'est aussi une manière plus logique de gérer le problème. Un thread s'exécutera en tâche de fond tout au long de l'exécution du programme et prendra une photo toutes les secondes. Cette photo sera envoyée à une autre méthode qui sera chargée de l'analyser.

C'est donc ainsi que j'ai pensé le problème concernant la caméra de Nao : un thread qui s'exécutera en tâche de fond s'occupera de capturer les images du labyrinthe que Nao voit pendant que ce dernier poursuit son avancée. Une fois les images obtenues, je me suis heurté à un des problèmes majeurs que j'ai rencontrés : le traitement des images reçues. En théorie, on peut exploiter entièrement une image et en tirer des informations précises et utiles. Dans la pratique, on peut exploiter une image, mais cela s'avère rudement difficile la plupart du temps et difficilement exploitable. De plus, je n'avais pas vraiment les connaissances pour réaliser ces traitements. Je parlerais plus de ces problèmes dans la partie "problèmes" de mon rapport.

C'est donc à ce moment que je suis allé demander conseil à mes encadrants. En effet, je ne voyais pas du tout comment je pouvais utiliser ces images afin de prendre une décision. J'ai commencé par aller voir M. Ramel, qui est un spécialiste en traitement d'images, afin d'obtenir des conseils en traitement d'images. M. Ramel m'a donné plusieurs pistes à étudier, mais il m'a souligné le fait que si nous ne mettons aucun signe distinctif sur le labyrinthe, l'analyse sera difficile. Or, au cours d'un de mes entretiens avec M. Bouquard, j'avais proposé de mettre des signes dans le labyrinthe. Ces signes auraient pu permettre à Nao d'avoir une information locale sur un croisement du labyrinthe ou un cul-de-sac par exemple, aucune information générale n'aurait été marquée. Toutefois, M. Bouquard m'a fait remarquer que mettre ces signes n'était pas une idée intéressante, dans le sens où je tentais d'adapter le problème à ma solution. J'ai écouté ses conseils et j'ai oublié l'idée de mettre des signes. Toutefois, nous sommes tombés d'accord sur le fait qu'il était possible d'ajouter des bandes sur les murs qui pourraient aider à la reconnaissance. Donc, le conseil de M. Ramel n'était pas inadapté, mais ce dernier m'avait surtout conseillé de faire un jeu d'images exemple afin de voir ce qu'on pourrait utiliser comme analyse d'images. J'ai donc commencé à prendre des images du labyrinthe depuis les caméras de Nao.

Toutefois, même en ayant ces images je ne voyais pas comment faire. Après avoir exposé mon problème à M. Gaucher, ce dernier m'a proposé un rendez-vous afin de discuter de méthodes de traitement d'images qui pourraient m'aider. C'est ce que nous avons fait, et M. Gaucher m'a présenté le filtre de Sobel et la transformée de Hough. Le filtre de Sobel analyse une image et permet de détecter tous les contours. La transformée de Hough est une technique de reconnaissance de formes, elle permet de détecter les lignes

droites les plus significatives qui passent par un point de l'image et les mettre en valeur. Pour chaque point de l'image, le nombre de lignes droites significatives ainsi que leur angle sont représentés dans un repère à deux dimensions appelé accumulateur de Hough. Une fois ces deux méthodes étudiées, je me suis lancé dans différents tests afin d'obtenir des images exploitables. J'ai donc commencé par récupérer des images de test dans le labyrinthe. En voici une : 5.1

FIGURE 5.1 – Image du labyrinthe prise depuis la caméra de Nao et sur laquelle j'ai effectué mes premiers tests

Ensuite, j'ai dû chercher une méthode pour utiliser ces algorithmes de traitement d'images. Mon choix s'est donc naturellement tourné vers l'utilisation de la librairie OpenCV. En effet, cette librairie implémente beaucoup de méthodes de traitement d'images très utiles, dont le filtre de Sobel et la transformée de Hough. J'ai donc commencé par appliquer sur les images le filtre de Sobel en utilisant OpenCV. Voici le résultat sur une image : 5.2

On voit ici que les contours sont très nettement soulignés. Ce fut un résultat encourageant. Pour continuer, j'ai décidé d'appliquer à mon image l'algorithme de Laplace, qui a le même but que le filtre de Sobel : détecter les contours. Voici le résultat de l'application de l'algorithme de Laplace : 5.3

Les contours sont bien détectés, mais de manière un peu plus floue que dans l'algorithme de Sobel. Une fois que je disposais de ces images, j'ai donc essayé d'appliquer la transformée de Hough sur ces dernières. Les débuts furent difficiles, pour obtenir un meilleur résultat, je me suis aussi mis à utiliser le filtre de Canny afin de détecter les contours de l'image au lieu d'utiliser le filtre de Sobel ou bien l'algorithme de Laplace. Malgré tout, j'obtenais des images comme celle-ci : 5.4

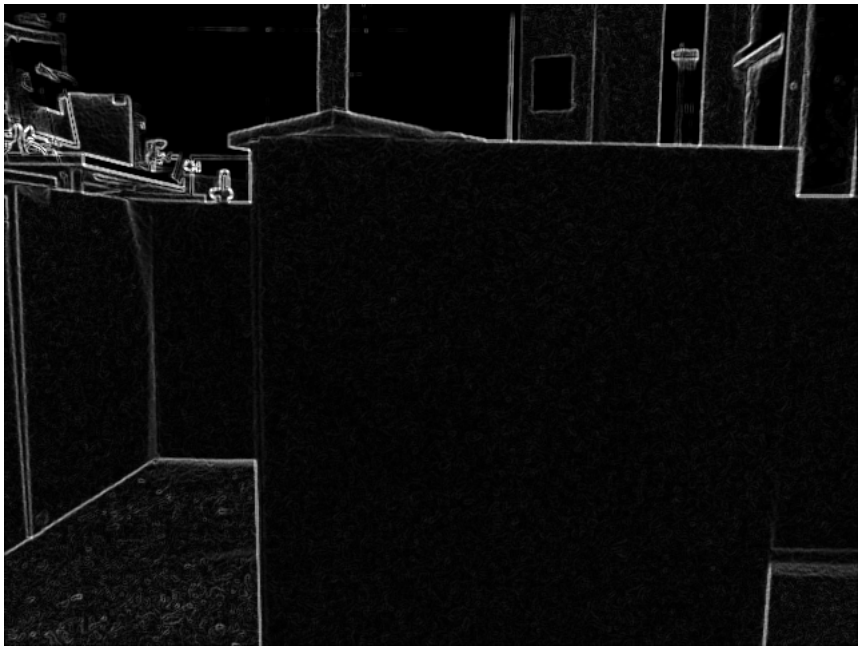

FIGURE 5.2 – Résultat de l'application du filtre de Sobel sur l'image de test

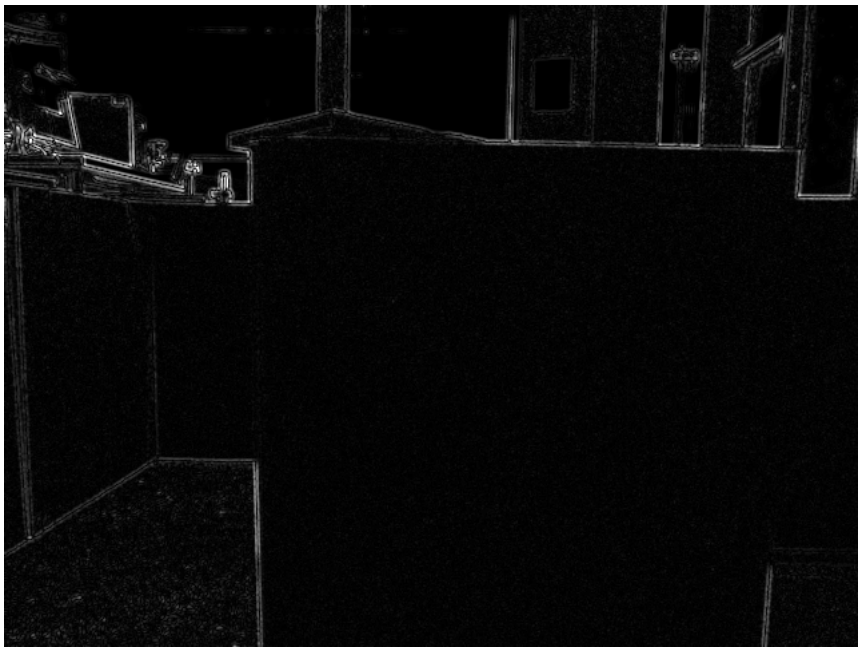

FIGURE 5.3 – Résultat de l'application de l'algorithme de Laplace sur l'image de test

Ici, à chaque droite significative détectée une droite rouge parcourant toute l'image est affichée. Le résultat était donc peu concluant. Les résultats étaient un peu meilleurs, mais j'ai découvert une autre méthode bien plus concluante pour utiliser la transformée de Hough : j'ai essayé d'utiliser la transformée de Hough probabiliste. Cette dernière, à la différence de la transformée de Hough généraliste, calcule seulement une fraction des points du plan accumulateur en les choisissant aléatoirement. Cette méthode est donc plus rapide, mais elle effectue aussi des calculs de longueur de ligne, ce qui lui permet de détecter des segments.

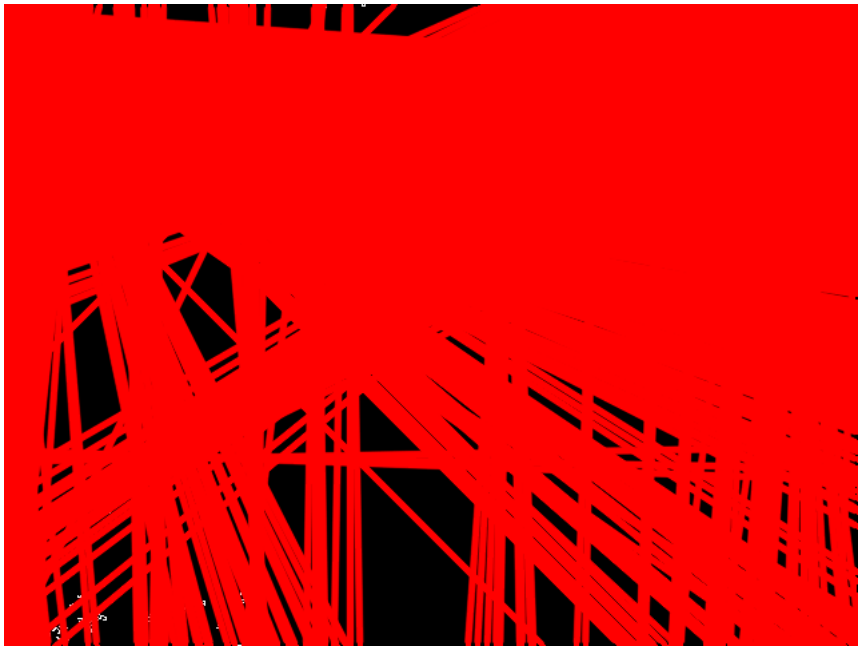

FIGURE 5.4 – Résultat de l'application de la transformée de Hough sur l'image de test filtrée à l'aide du filtre de Canny

Après beaucoup de tentatives infructueuses en essayant tour à tour la transformée de Hough et la transformée de Hough probabiliste, après avoir changé maintes fois les paramètres, j'avais toujours un problème. Je n'arrivais pas à afficher un nombre suffisant de droites sur l'image : j'en avais toujours trop pour la transformée de Hough et toujours pas assez pour la transformée de Hough probabiliste. Il me fallait donc seuiller les résultats, c'est-à-dire n'afficher que les droites significatives des points ayant un nombre de droites significatives dans l'accumulateur de Hough supérieur ou égal à un nombre défini. Après avoir remarqué que le seuil pouvait tout simplement être passé en paramètre de l'appel de la fonction, voici le résultat que j'ai obtenu avec l'application de la transformée de Hough probabiliste en seuillant : 5.5

FIGURE 5.5 – Résultat de l'application de la transformée de Hough probabiliste sur l'image de test sur laquelle nous avons préalablement appliqué le filtre de Canny

Cette image résultant de l'application de la transformée de Hough probabiliste était donc très encourageante. Voici quelques images originales ainsi que le résultat de l'application du filtre de Canny et de la transformée de Hough probabiliste sur ces images.

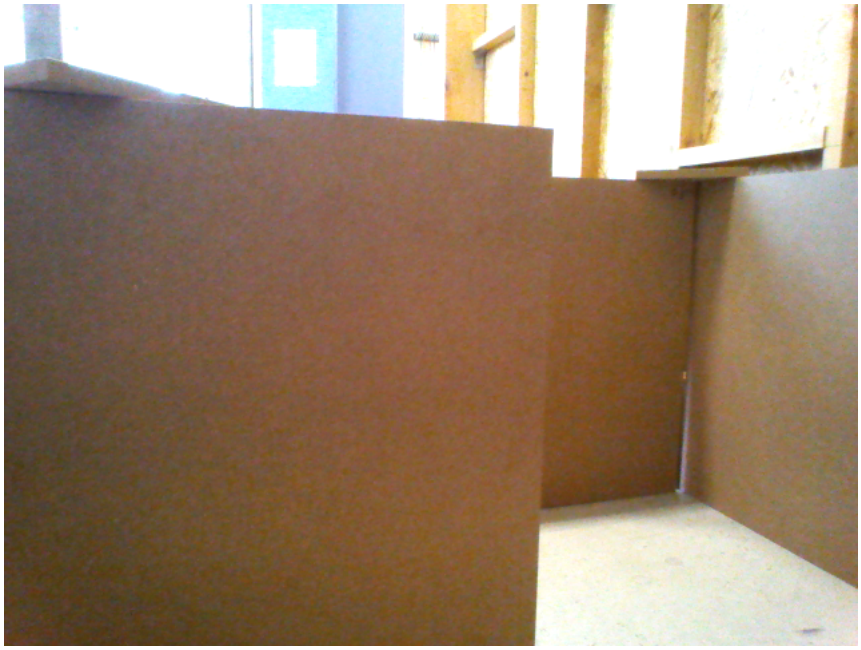

FIGURE 5.6 – Image du labyrinthe capturée par la caméra de Nao (1)

FIGURE 5.7 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao (1)

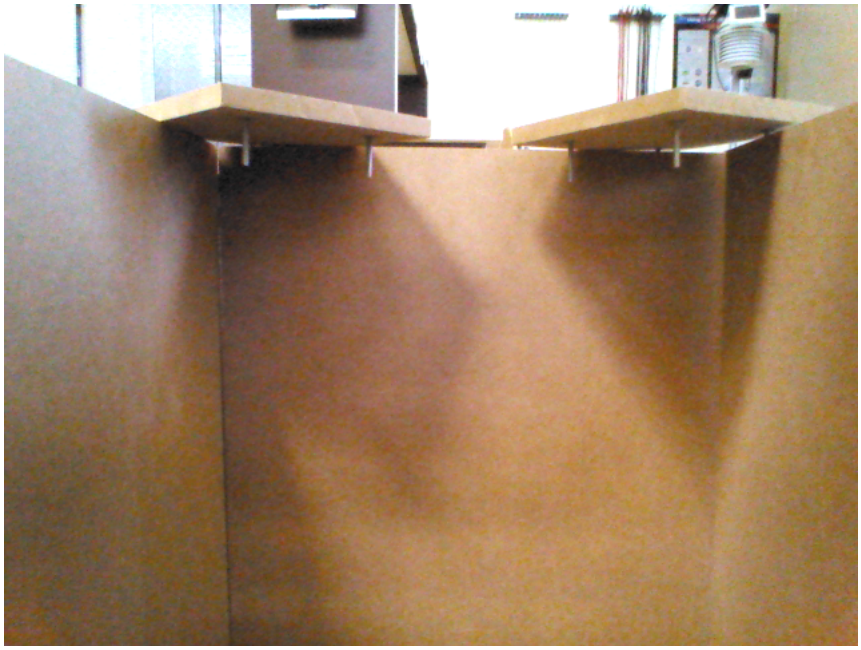

FIGURE 5.8 – Image du labyrinthe capturée par la caméra de Nao (2)

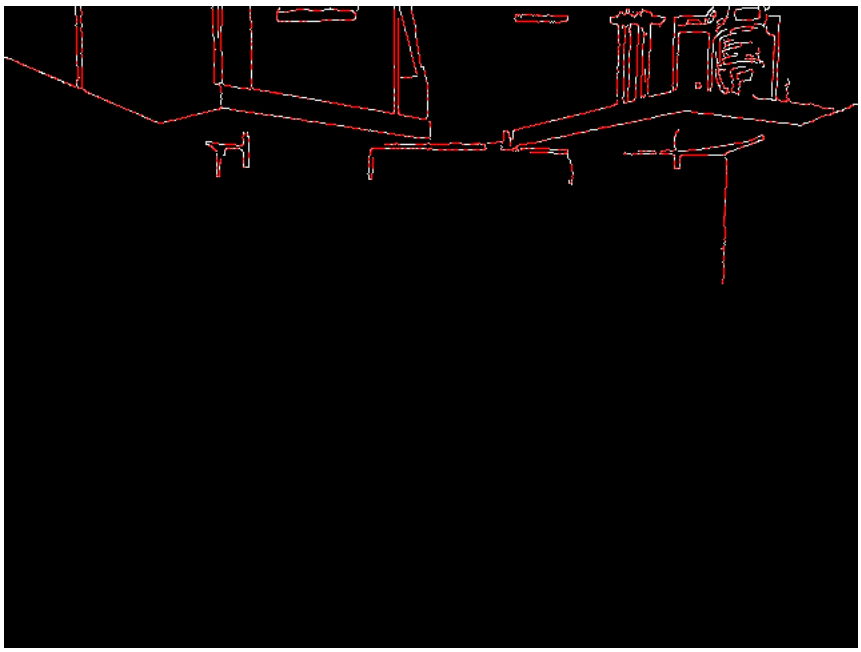

FIGURE 5.9 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao (2)

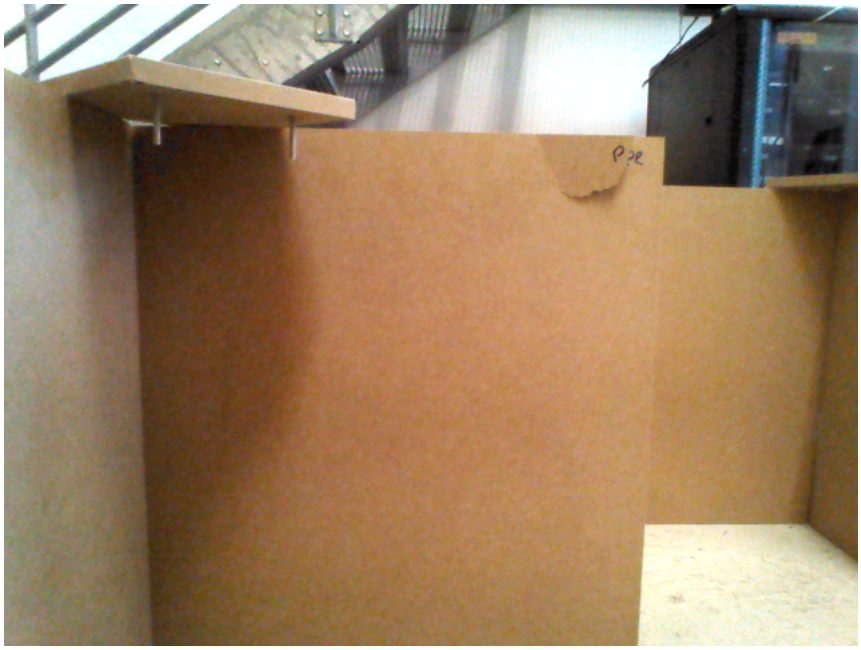

FIGURE 5.10 – Image du labyrinthe capturée par la caméra de Nao (3)

FIGURE 5.11 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao (3)

FIGURE 5.12 – Image du labyrinthe capturée par la caméra de Nao (4)

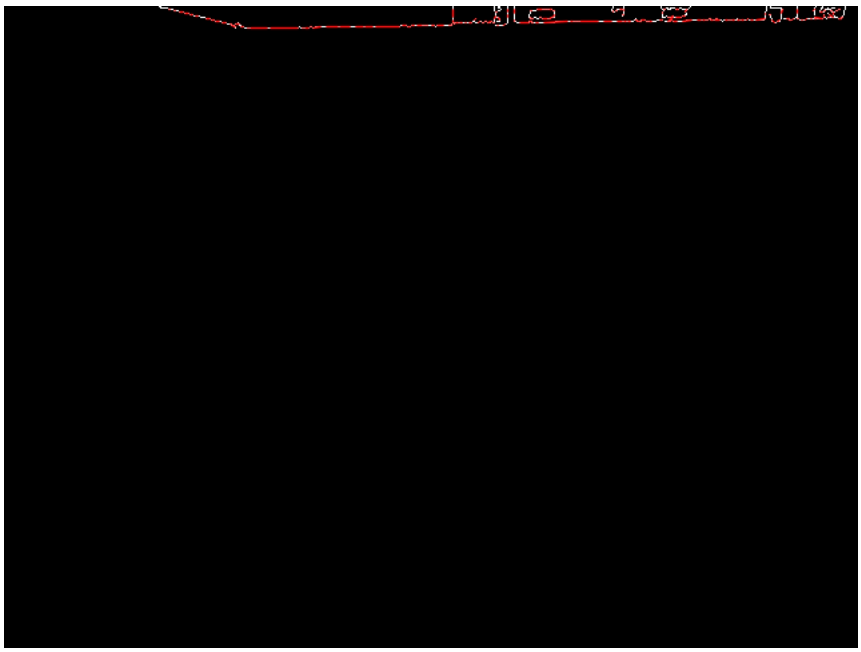

FIGURE 5.13 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao (4)

Les résultats obtenus sont très satisfaisants et encourageants. Nous constatons que les contours sont bien détectés et que la transformée de Hough probabiliste a bien fonctionné. Toutefois, nous remarquons aussi que des détails, qui à l'oeil humain sont évidents, ne sont pas détectés. Par exemple lorsque deux planches se chevauchent en perspective. C'est pourquoi j'ai voulu réaliser une nouvelle batterie de tests. Mais cette fois-ci, j'ai appliqué des bandes blanches sur les bords de chaque planche dans le but de marquer une séparation nette. Voici les tests que j'ai réalisés, à chaque fois en premier l'image originale puis l'image résultante sur laquelle nous avons utilisé le filtre Canny puis appliqué la transformée de Hough probabiliste.

FIGURE 5.14 – Image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (1)

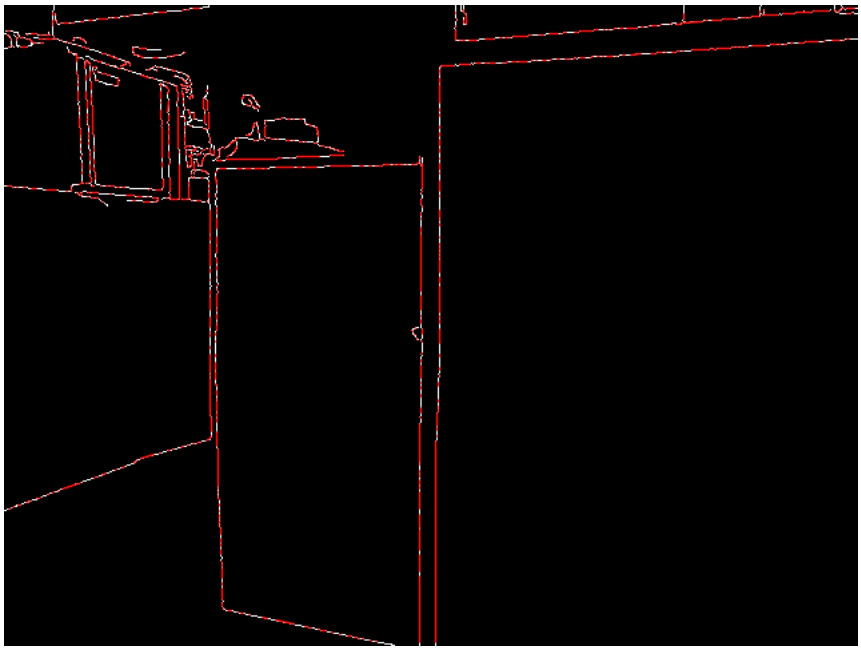

FIGURE 5.15 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (1)

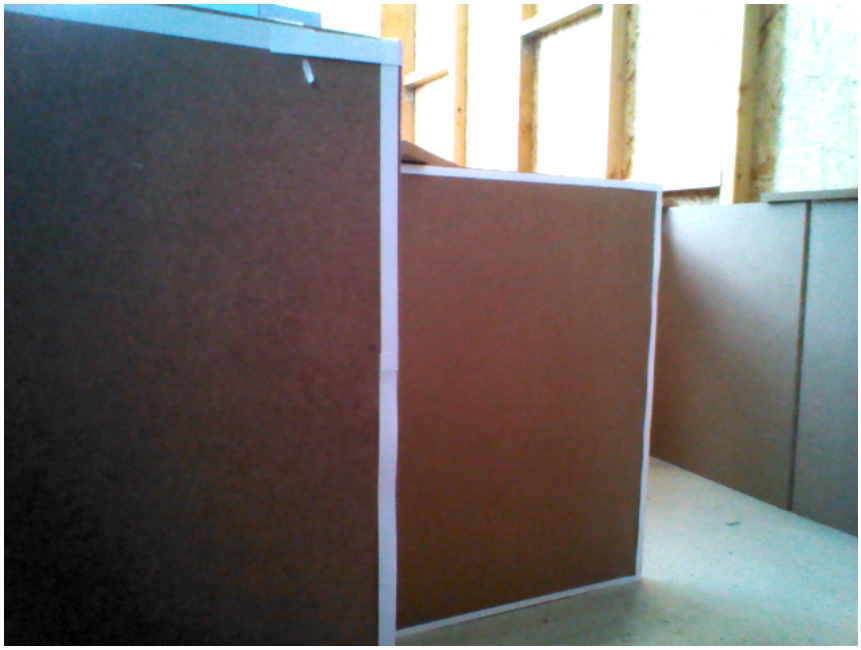

FIGURE 5.16 – Image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (2)

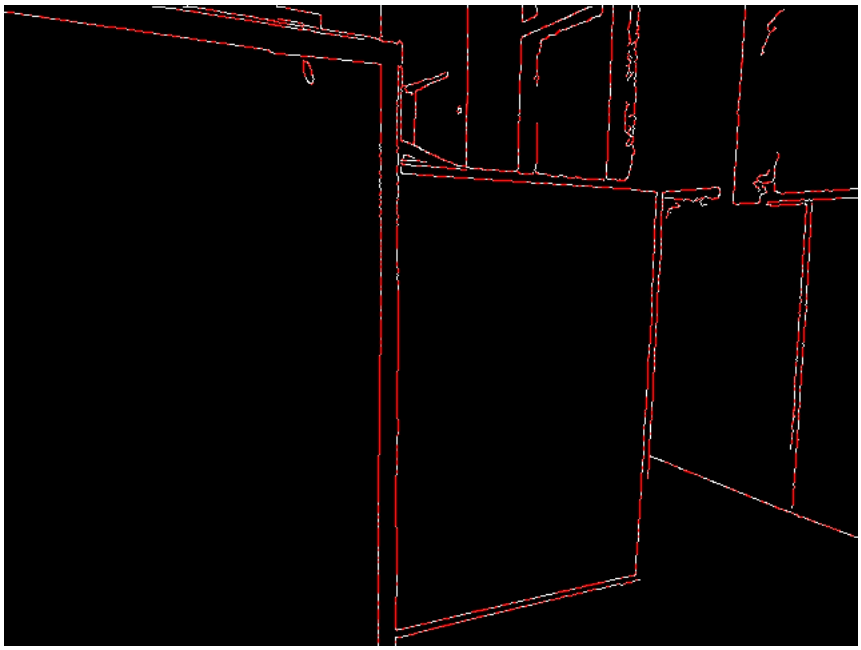

FIGURE 5.17 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (2)

FIGURE 5.18 – Image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (3)

FIGURE 5.19 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (3)

FIGURE 5.20 – Image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (4)

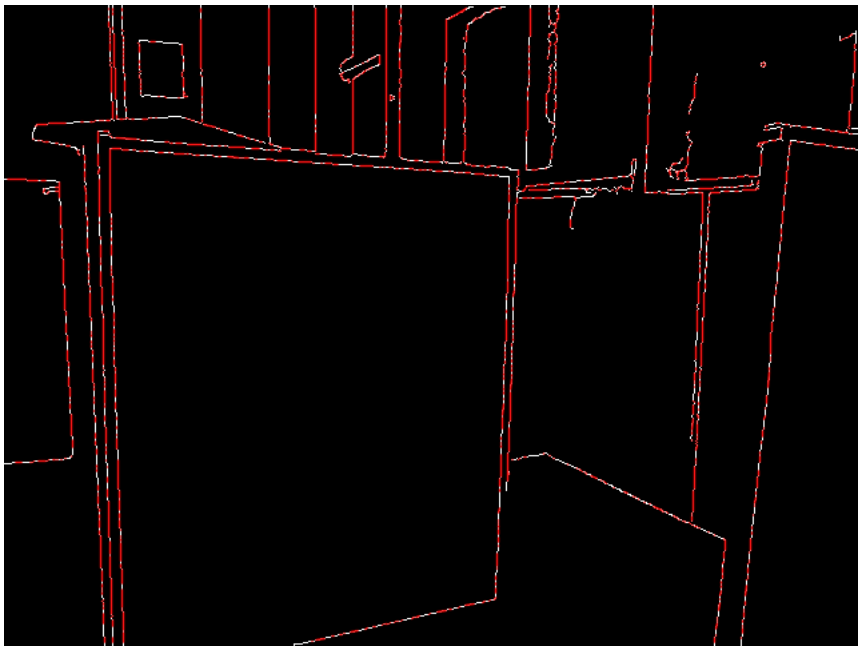

FIGURE 5.21 – Résultat de l'application d'un filtre de Canny puis de la transformée de Hough probabiliste sur une image du labyrinthe capturée par la caméra de Nao. Des bandes blanches ont été appliquées sur les planches du labyrinthe. (4)

Ces résultats étaient très concluants. C'est vers la fin de mon projet que j'ai réussi à obtenir de tels résultats. Par manque de temps, je n'ai pas réussi à aller plus loin dans la résolution. Il reste encore beaucoup de travail à faire pour pouvoir exploiter correctement ces données. Il nous faudrait nous fixer des formes reconnaissables à détecter. Nous pourrions imaginer chercher à détecter des angles ou de grandes lignes droites qui symboliseraient des croisements ou des planches, il nous faudrait aussi faire en sorte que le robot se déplace et prenne d'autres photographies si les résultats obtenus ne sont pas bons.

Du fait que je n'ai pas réussi à analyser les images et à en tirer des informations, je n'ai pas pu mettre en place la partie création du repère 2D et création du graphe. Il aurait fallu analyser les images comme je l'ai indiqué ci-dessus pour en tirer des informations exploitables afin de créer le graphe et un point dans le repère. Enfin, il aurait fallu récupérer les distances parcourues par Nao afin de déplacer Nao lui même dans le repère.

Résultats et problèmes rencontrés

6.1 Problèmes

Au cours de mon travail sur ce projet, j'ai rencontré plusieurs problèmes. Premièrement, le développement d'un premier module autonome. J'ai eu des difficultés à obtenir ce que je voulais. En effet, je ne disposais d'aucune information. Je ne connaissais pas du tout la manière de programmer sur Nao et je n'avais pas de méthode définie. La documentation, en anglais, était parfois peu claire sur les méthodes et les différents documents que je pouvais trouver sur Internet étaient souvent adaptés à une autre version de Nao. J'ai réussi à développer ce module, mais j'ai dû croiser beaucoup d'informations et étudier longuement la documentation avant d'arriver à compiler correctement le module.

J'ai aussi eu des problèmes à la programmation de Nao. Non pas à comment développer le module, mais à comment mettre en place tout ce que j'avais développé. En effet, j'ai toujours eu du mal à appréhender la manière de programmer sur Nao et je pense que je n'ai pas encore tout à fait saisi. J'ai pensé mon développement de la manière suivante : je réalisais toutes mes actions dans un module seul et l'exécution de ce module réalisait l'exploration du labyrinthe. Toutefois, j'ai réalisé ce choix tout en doutant, car j'ai eu l'impression qu'Aldebaran avait pensé faire réaliser les programmes comme une accumulation de plusieurs modules, plusieurs programmes pour réaliser un comportement bien défini et non pas un seul gros module pour réaliser une action. Il y a aussi possibilité de créer une librairie personnelle afin de réaliser les actions voulues. J'ai fait mon choix en connaissance de cause et cela n'a pas porté préjudice à mon développement. Toutefois, j'ai eu beaucoup de problèmes inexpliqués lors de l'exécution du module.

En effet, en développant sur Nao nous développons sur un système temps réel. Il y a donc des contraintes de concurrence entre processus à respecter scrupuleusement. Par exemple, on ne peut pas demander à Nao de se lever et de marcher en même temps, l'un des deux doit être bloquant, c'est-à-dire qu'il doit empêcher tout autre processus de prendre la main pendant son exécution. En l'occurrence, l'action de se lever est bloquante, cela a été prévu par Aldebaran. Certaines actions de base de Nao sont bloquantes, de plus, dans la librairie fournie par Aldebaran Robotics il existe des méthodes pour manipuler des verrous logiciels afin de protéger certains appels de fonctions.

Ainsi, tout semble être mis en oeuvre afin de permettre la gestion de sections critiques pour éviter les appels concurrents. Toutefois, malgré tout cela, j'ai rencontré des problèmes à l'exécution de mon module. Certaines méthodes, bien que déclarées bloquantes dans la documentation, ou bien certains appels de fonction que je faisais moi même et que j'avais protégés étaient interrompus. Ce n'était pas forcément une interruption par un autre processus pour prendre la main sur le processus en cours, mais par moment, lorsque Nao réalisait une action bloquante, son action était saccadée comme si un autre processus tentait

de prendre la main ou comme si le robot avait des ralentissements. Je ne m'explique toujours pas ce comportement, et ces saccadements sont un véritable problème.

J'ai aussi rencontré des problèmes avec les sonars de Nao. En effet, je me suis rendu compte tardivement que le niveau de détection était haut. C'est-à-dire que les sonars détectent les obstacles entre vingt et soixante-dix centimètres. Toutefois, ce problème est résolu grâce aux bumpers de Nao qui permettent de détecter les objets immédiats. Malgré tout, j'ai rencontré beaucoup de problèmes pour faire en sorte que les deux sonars ne rentrent pas en conflit. En effet, si le sonar gauche détecte un obstacle, il faut qu'on ait le temps d'exécuter la réaction voulue. Le souci est que le sonar droit détecte aussi pendant l'exécution de la réaction à la détection du sonar gauche. De ce fait, j'ai eu du mal à mettre en place un comportement en réaction à la détection des sonars, car les deux sonars entrent parfois en concurrence. J'ai pensé à une solution, mais je ne l'ai pas essayée, il faudrait faire en sorte de bloquer les évènements de l'autre sonar lorsqu'un des deux sonars a détecté un obstacle proche. Nous pourrions donc désinscrire notre module des évènements de l'autre sonar pendant que la réaction est exécutée.

Le problème majeur de mon projet aura été la partie concernant le traitement d'images. C'est sur cette partie que je me suis posé le plus de questions. Je n'avais pas beaucoup de compétences dans ce domaine, mais j'ai beaucoup réfléchi et avec l'aide de M. Gaucher j'ai réussi à avancer. Au début, j'avais pensé à une solution : nous aurions pu mettre en place des symboles dans le labyrinthe afin de donner une information locale sur les croisements et les culs-de-sac. Ces symboles n'auraient pas aidé le robot à se retrouver dans le labyrinthe, mais ils l'auraient aidé à avoir une information locale sur son environnement afin de la traiter. Toutefois, cette solution a vite été abandonnée après réflexion avec M. Bouquard. Cela aurait été comme adapter le problème à notre solution.

J'ai donc cherché à faire autrement et à me lancer dans l'analyse d'images. J'ai réussi à récupérer des images depuis le robot et à les traiter dans la mémoire du robot. Le but était de passer les images en niveau de gris, détecter les contours puis enfin arriver à détecter les contours les plus significatifs. J'ai réussi à faire cela grâce à l'algorithme de Canny et à la transformée de Hough probabiliste. Une fois les résultats obtenus, je ne savais pas quoi en faire et malheureusement il ne me restait plus assez de temps pour développer une solution. J'ai pensé à parcourir l'image pour récupérer les contours les plus significatifs détectés et puis essayer de reconnaître des formes. En apprenant des formes à Nao, j'aurais pu essayer de lui faire reconnaître les formes détectées dans les images. J'ai toutefois manqué de temps pour essayer cette solution.

6.2 Résultats

Premièrement, j'ai réussi à développer un module autonome pour Nao. Dans le but de capitaliser les compétences acquises, j'ai rédigé une documentation détaillant la démarche que j'ai suivie pour réaliser ce module. Ensuite, j'ai commencé à programmer le module en tant que tel. J'ai commencé à programmer en vue des journées portes ouvertes de Polytech Tours puis j'ai continué sur cette lancée pour réaliser plus de composantes du programme. Voici ce que j'ai développé dans le module :

- Gestion des sonars et récupération de la valeur de distance détectée,

- Rotation de Nao avec l'angle de rotation inversement proportionnel à la distance détectée,
- Gestion des chutes, Nao se relève en cas de chute,
- Lancement de thread afin de capturer des images du labyrinthe.

Disposant du module et des différentes fonctionnalités, je me suis lancé dans le traitement et l'analyse d'images afin de pouvoir prendre des décisions en fonction de l'environnement immédiat du robot. J'ai réussi à détecter précisément les contours des images que j'avais, mais cela m'a pris du temps. Et malheureusement j'ai manqué de temps pour continuer mon étude. Ne disposant pas de la partie traitement d'image, je n'ai pas pu me lancer dans la partie théorie des graphes et repère deux dimensions.

Je n'ai pas mené le projet à son terme. Toutefois, j'ai tiré plusieurs conclusions de mon étude et certains résultats sont exploitables. Mon but a aussi été de capitaliser un maximum d'information avec à l'esprit de faciliter le travail d'une personne qui reprendrait éventuellement le projet. Ce rapport et la documentation que j'ai rédigés recensent les remarques et déductions que j'ai faites sur ce projet et les difficultés que j'ai pu rencontrer.

6.3 Livrables

Je fournis à mes encadrants ce que j'ai rédigé au cours de mon projet. Les livrables comprennent :

- Le code source du module sur lequel je travaillais,
- La documentation que j'ai rédigée pour expliquer comment compiler un module,
- Mon cahier de spécifications système,
- Mon rapport de projet de fin d'études.

Le code source est compilable et permet la création d'un module, mais à l'heure actuelle il ne crée pas un module fonctionnel pour explorer le labyrinthe. Il me servait surtout à réaliser différents tests sur Nao.

Bilan technique, bilan humain

7.1 Bilan technique

Ce projet a été pour moi une grande découverte. Nao n'est pas un robot si commun, et il est très onéreux. Ce dernier fait les gros titres en France actuellement, car Aldebaran Robotics est une société française et les médias mettent en avant sa réussite. C'est pourquoi avoir un robot Nao à Polytech Tours est une chance. J'ai pu avoir accès à un robot Nao régulièrement et j'ai pu l'étudier et travailler dessus, c'est une chance que je mesure. J'ai découvert une partie du monde de la robotique et l'énorme travail de la société Aldebaran Robotics.

Nao est un robot très bien fait, le travail réalisé en amont par Aldebaran Robotics nous permet d'utiliser le robot de manière efficace, sans avoir à tout refaire. J'entends par là que lorsqu'on veut que le robot se lève, il suffit d'appeler la méthode correspondante et non pas de tester la jambe droite, récupérer son orientation en collectant les données des différents moteurs et de donner une orientation à chaque moteur pour faire que le robot se lève. Bien évidemment, ce que je viens de décrire n'est pas ingrat, mais je pense personnellement que cela ne fait pas partie du domaine de compétence d'un ingénieur en informatique généraliste. Le fait que ces fonctionnalités soient déjà en place nous permet de nous concentrer sur des domaines plus éloignés du matériel. Et c'est ce que j'ai beaucoup apprécié au cours de ce projet.

J'ai pu découvrir l'environnement de développement inhérent à Nao. J'ai eu une longue phase de recherche au début du projet, je devais me renseigner un maximum sur le robot pour pouvoir l'utiliser au mieux. J'ai donc découvert la manière dont Aldebaran Robotics a pensé son robot, le fait d'utiliser des proxys pour accéder aux différents capteurs, les différentes approches de programmation et bien d'autres. Cela m'a aussi permis de travailler en C++, ce n'est pas mon langage de prédilection. Pour pouvoir programmer sur Nao, j'ai aussi été confronté à des concepts que je ne maîtrisais pas, comme la compilation croisée. J'ai beaucoup appris sur ce concept. J'ai aussi pu observer les contraintes des systèmes temps réels en activité, en effet, le robot ne peut pas réaliser toutes les actions en simultané. Il faut donc pouvoir gérer des sections critiques et savoir quand donner la main à une action et quand ne pas le faire.

Ce projet m'aura aussi permis de mettre à l'épreuve mes connaissances en traitement d'images. En effet, au cours de mon cursus à Polytech Tours j'ai étudié le traitement d'images en informatique, mais cela n'était pas une étude extrêmement approfondie. Je maîtrisais des concepts clés, je savais comment appliquer des masques sur une image, j'avais des notions de ce qu'on pouvait faire ou ne pas faire. Mais, ce n'est qu'un petit aperçu du traitement d'images. Grâce à M. Gaucher, j'ai pu apprendre de nouvelles techniques de traitement d'images et apprendre comment me servir d'une image pour arriver à mes fins.

En définitive, je me suis beaucoup enrichi en ce qui concerne la technique. Mais, j'ai aussi énormément appris au niveau humain tout au long de ce projet, et j'ai réalisé beaucoup de choses concernant le fait d'être un ingénieur.

7.2 Bilan humain

Au cours de ce projet, j'ai réellement saisi le sens de ce qu'est être un ingénieur, avoir une démarche d'ingénieur. Non pas que j'ai eu une révélation lors de mon projet de fin d'études, mais tous les projets que nous faisons à Polytech Tours nous apprennent à avoir des réflexes et à devenir des ingénieurs. C'est au cours de mon projet de fin d'études plus que jamais que j'ai eu le sentiment de mener une démarche d'ingénieur. Le contexte dans lequel nous sommes placés joue pour beaucoup, nous travaillons en autonomie avec un donneur d'ordre sur un projet avec un but bien précis, mais sans avoir un chemin tracé vers la solution.

Chaque jour de projet, je réfléchissais à ce que je faisais et j'essayais de mener mon projet dans le bon sens. Je cherchais des moyens de résolution et je savais qu'il me fallait prendre des décisions. C'est là que j'ai pu constater que j'avais appris tout au long de ma scolarité à Polytech Tours à travailler en autonomie, à faire mes propres choix. Je gardais toujours à l'esprit cette phrase : Pour mener à bien mon projet je dois prendre des décisions et il n'y a pas de bonnes ou de mauvaises décisions, il n'y a que mes choix et la justification que je donnerai pour prouver que ces choix font aller le projet dans le bon sens.

Pour ce projet, j'ai été confronté à un problème inconnu dans un milieu inconnu. Je n'avais jamais eu entre les mains un robot, et encore moins un robot comme Nao. J'ai donc dû passer beaucoup de temps à étudier de la documentation et réaliser des tests sur le robot. Au cours de mon projet, j'ai particulièrement apprécié le travail que j'ai eu à fournir pour le développement du module et le résultat fut gratifiant. En effet, jusque là personne dans l'école n'avait réussi à réaliser un module indépendant sur Nao. Cela me motivait beaucoup et je tenais à passer cette étape, c'était un jalon important de mon projet et j'étais très satisfait lorsque j'ai réussi à compiler mon module.

Pour finir, ce projet fut donc vraiment enrichissant. J'ai beaucoup aimé travailler conjointement avec mes encadrants, la partie gestion de projet a été intéressante et c'était la première fois que j'avais un projet de cette envergure à réaliser seul. J'ai beaucoup appris au cours de ce projet, je sentais que je menais une démarche d'ingénieur.

Conclusion

Ce projet fut une expérience très enrichissante et intéressante. J'ai appris, tant au niveau technique qu'au niveau humain. J'ai pu étudier et travailler sur un robot à la pointe de la technologie, ce fut une chance. Le sujet était intéressant et pour la première fois nous étions en réelle autonomie sur un projet. Il nous était demandé du travail, de la rigueur et d'avoir une démarche d'ingénieur pour réaliser le projet. Il nous fallait mettre en place une gestion de projet, j'ai aussi du rédiger un cahier de spécifications système pour le projet. C'était aussi une expérience intéressante dans le sens où nous avions des jours banalisés dédiés au travail de ce projet.

Pour réaliser le projet, j'ai commencé par prendre en main le robot. Ensuite, j'ai fait mes choix sur la manière dont je souhaitais programmer, c'est-à-dire un module indépendant sur Nao. J'ai réussi à trouver comment compiler le module et je me suis lancé dans la programmation du module. J'ai réussi à programmer le module, à accéder aux différents capteurs de Nao grâce au C++, mais la partie la plus difficile fut la partie concernant le traitement d'images. J'ai réussi à traiter les images, mais je ne disposais plus de temps afin de les exploiter pour la suite du module.

Pour finir, même si je n'ai pas réussi à terminer mon projet je suis content d'avoir travaillé dessus. J'ai appris beaucoup de choses et j'ai vraiment aimé travailler avec mes encadrants. Ce fut un plaisir de travailler sur un projet intéressant avec des personnes motivantes et intéressantes. Je regrette de n'avoir pu amener le projet à son terme, mais je laisse mes sources, les documentations que j'ai rédigées et ce rapport dans l'éventualité ou un étudiant voudrait reprendre mon travail afin de porter le projet encore plus loin. Dans un futur projet, si Nao arrive à sortir du labyrinthe, nous pourrions ajouter des composantes d'optimisation comme le faire sortir le plus vite possible.

Annexe

Cahier de spécification système

1.1 Introduction

Dans le cadre des projets de fin d'études, un projet ayant pour but d'étudier et d'utiliser le robot Nao crée par la société Aldebaran a été proposé. Le projet vise à faire évoluer le robot dans un labyrinthe pour lui en faire trouver la sortie. Pour ce faire, le labyrinthe sera perçu comme un graphe. Une résolution algorithmique respectant la théorie des graphes sera donc utilisée. Nao est programmable en C++ et Python et peut embarquer des modules programmés dans sa mémoire.

1.2 Contexte de la réalisation

1.2.1 Contexte

Le projet s'inscrit dans un contexte de nouvelles technologies robotiques avec l'utilisation du robot Nao. Nao est un robot crée par la société Aldebaran Robotics. Il n'a pas de réelle utilité mise à part être un outil de recherche et de développement pour les écoles ou les universités. Il peut aussi être un compagnon de jeu pour les enfants. L'implémentation d'une résolution algorithmique est une contrainte forte pour la résolution du problème, tout au long de ce dernier l'accent sera mis sur la théorie des graphes. Nous aurons aussi à traiter des images, rechercher des formes dans des images.

1.2.2 Objectifs

Pour ce projet, nous disposons donc du robot humanoïde Nao. Il s'agit d'exploiter les possibilités offertes par cette plateforme. Le robot dispose de primitives programmables pour évoluer dans son environnement. L'enjeu est clair : il s'agit de prouver que le robot Nao est capable d'évoluer dans un environnement complexe grâce à une résolution algorithmique et en utilisant ses capteurs.

1.2.3 Hypothèses

Plusieurs hypothèses peuvent être faites, ainsi que des constatations qui pourraient avoir une incidence sur le projet dans le futur. Le projet n'étant pas qu'un développement logiciel, nous devons tenir compte de certaines contraintes physiques inhérentes à l'utilisation du robot.

La première contrainte à prendre en compte est le fait que les pieds du robot sont plats et glissent selon le revêtement. Il faut absolument en tenir compte, car si nous voulons faire tourner le robot d'un certain angle il y aura forcément une erreur et le robot ne tournera pas selon l'angle désiré. C'est un problème qu'il faut envisager lors de la résolution du labyrinthe. Les glissements peuvent aussi provoquer des chutes, le robot dispose d'un très bon système de reprise sur les glissements et d'auto protection lors des chutes mais il faut être en mesure de faire en sorte que le robot se relève après une chute.

Pour la détection des obstacles, il a été envisagé d'utiliser les sonars de Nao. Toutefois, ces derniers ont une portée courte, ce qui réduit leur utilité pour la résolution du projet. De plus, l'utilisation du code Python existant déjà pour la gestion du sonar sous Choregraphe ne convient pas à nos besoins. Il faudra donc retravailler ce code afin de pouvoir l'utiliser.

Une problématique à gérer est l'utilisation de la batterie : le robot dispose d'une autonomie limitée avec laquelle il faudra travailler. Nous ne pouvons pas permettre que la résolution du labyrinthe prenne trop de temps.

Le robot chauffe aussi très vite, il faudra surveiller les données fournies par le robot sur la température de ses composants afin de prévoir des solutions de secours immédiates pour éviter tout dommage matériel. Par exemple, en cas de surchauffe du processeur, le robot s'éteint tout seul. S'il tente de s'éteindre en étant debout, il risque de tomber et de possiblement endommager un composant.

Une contrainte très importante est aussi la méthode de perception qu'utilisera le robot. Une grosse phase de réflexion devra être effectuée afin de prendre une décision. Il faudra choisir entre utiliser les sonars du robot ou bien ses caméras pour une détection via analyse des images. Les contraintes sont les suivantes : la portée du sonar est limitée, et le sonar ne nous permet pas d'affiner la détection. Ce dernier sera capable de nous signaler un obstacle, mais pas de nous signaler un embranchement ou un carrefour du labyrinthe. Ensuite, concernant la détection par vidéo, la contrainte est que le processus de détection devient beaucoup plus lourd qu'avec le sonar. Toutefois, la capacité d'analyse est grandement améliorée grâce à la vidéo et cela nous ouvre des perspectives très intéressantes. La détection par l'image est difficile, mais faisable. Nous pourrons utiliser des différences de contrastes entre le sol et les murs afin de réaliser une détection de contours.

En conclusion, nous pourrions utiliser conjointement la détection vidéo et le sonar. Le sonar pour détecter les obstacles imminents, la détection vidéo pour pouvoir gérer le déplacement selon la théorie des graphes.

1.2.4 Bases méthodologiques

En tout premier lieu, ce projet est réalisé avec un outil bien particulier : le robot humanoïde Nao. Ce robot est le pilier central du projet, c'est sur lui que l'étude est réalisée. Nao peut être programmé : nous le programmerons en C++ ou en Python. Ces deux langages permettent de créer des modules que nous placerons dans la mémoire du robot, ces modules seront chargés au démarrage et pourront être lancés sur demande. Nous devrons aussi nous servir de la suite logicielle fournie par Aldebaran. Elle est composée des logiciels suivants : Choregraphe, Naoqi et Qibuild dont la fonction sera détaillée ultérieurement.

1.3 Description générale

1.3.1 Environnement du projet

La robotique a toujours été un grand défi pour l'Homme. Des premiers automates aux robots évolués tels que nous les connaissons aujourd'hui, la recherche et le développement en robotique permettent de pousser toujours plus loin les limites. Nao est une création de la société Aldebaran Robotics. Il embarque de multiples capteurs et un système d'exploitation. Ce robot est fourni avec trois logiciels : Naoqi, Qibuild et Choregraphe. Naoqi est le logiciel principal qui est en permanence exécuté sur Nao et permet de le contrôler. Le framework Naoqi est le framework programmable qui permet de programmer sur Nao. Qibuild est une librairie qui permet la compilation des sources utilisant les méthodes manipulant Nao. Qibuild gère la dépendance entre projets et supporte la cross-compilation (pour pouvoir faire exécuter le code sur le robot). Enfin, Choregraphe est un logiciel qui est utilisé sur un ordinateur et permet de contrôler le robot à distance afin de lui faire réaliser certains comportements. La problématique du projet est de faire résoudre des labyrinthes au robot en utilisant sa capacité de calcul et ses capteurs. Nous utiliserons une résolution algorithmique pour le parcours, résolution qui s'appuiera la théorie des graphes.

1.3.2 Caractéristiques des utilisateurs

Pour ce projet, il n'y aura pas d'utilisateurs au sens que personne ne devra utiliser une interface pour piloter le robot. Nao aura le code permettant son fonctionnement embarqué dans sa mémoire.

1.3.3 Fonctionnalités et structure générale du système

Le système est basé sur Nao. Nao communique via wifi ou câble ethernet pour recevoir les commandes de Choregraphe ou bien pour qu'un développeur se connecte à sa mémoire. Ensuite, une fois que le module développé est dans sa mémoire, Nao est indépendant et interagit avec le labyrinthe.

1.3.4 Contraintes de développement, d'exploitation et de maintenance

Contraintes de développement

Il y a de nombreuses contraintes à respecter pour le développement sur Nao. Premièrement, les limites physiques du robot. Nous ne pouvons pas faire plus que ce que nous permettent les différents capteurs et les capacités physiques du robot (marcher, tenir des objets etc). Il nous faudra tenir compte de ces restrictions. Nao est utilisable via Choregraphe sur toutes les plateformes. Nous pouvons programmer Nao à l'aide de nombreux langages et sur toutes les plateformes, toutefois, il n'y a qu'en C++ et en Python que nous pouvons faire du code exécutable directement depuis la mémoire du robot. Si nous souhaitons créer un module qui fonctionnerait de manière autonome sur le robot, il faut programmer et compiler depuis un système Unix. Il nous faut aussi utiliser Naoqi qui est un logiciel exécuté en permanence sur le robot et permettant de le contrôler. Nous devrons utiliser Qibuild, qui est un framework permettant de programmer Nao. Naoqi et Qibuild sont des logiciels de base à utiliser pour le développement : ils font partie de l'environnement nécessaire pour programmer Nao. Il existe aussi un logiciel du nom de Naosim qui permet de simuler l'utilisation du robot dans un environnement afin de tester les programmes réalisés. Pour la réalisation du projet, nous avons fait le choix d'utiliser un algorithme de parcours de graphe. Chaque noeud du graphe représentera un carrefour du labyrinthe. L'arc entrant du noeud sera le chemin par lequel Nao est arrivé au carrefour, le noeud pourra avoir deux ou trois arcs sortants selon le type de carrefour. Un virage sera représenté par un arc simple. Qibuild propose une librairie de méthodes permettant d'accéder aux différents capteurs du robot. Le robot est capable de se connecter à un réseau wifi et d'obtenir une adresse IP, nous pourrons donc communiquer via wifi avec le robot.

1.4 Description des interfaces externes du logiciel

1.4.1 Interfaces matériel/logiciel

Le robot dispose d'un port USB et d'un port ethernet. Nous pouvons nous connecter grâce à un câble ethernet afin de pouvoir accéder au même réseau que le robot. Une fois sur le même réseau via un ordinateur par exemple, nous pouvons nous connecter au robot via Choregraphe ou en SSH afin de lui envoyer des commandes. Nous pouvons aussi accéder à son interface de gestion en tapant son adresse IP dans un navigateur. Via cette interface de gestion, nous pouvons configurer le robot afin de faire en sorte qu'il se connecte à un réseau wifi. Nao est capable de se connecter à un réseau wifi et d'obtenir une adresse IP, si

un des réseaux wifi spécifiés dans sa liste de réseaux est joignable. Une fois connecté sur le même réseau que Nao, en wifi ou en ethernet, il est possible de se connecter au robot via Choregraphe ou en SSH.

1.4.2 Interfaces homme/machine

Il n'y aura pas d'interface homme/machine à développer car il n'y a pas de perspective utilisateur pour le projet.

1.4.3 Interfaces logiciel/logiciel

Nao est accessible via Choregraphe ou via SSH. Nous pouvons transférer des fichiers en utilisant ces deux méthodes. Nao est aussi capable de scanner sa mémoire pour avoir accès aux modules qui sont enregistrés sur cette dernière. Le programme qui sera développé utilisera le framework Naoqi pour accéder aux différents capteurs.

1.5 Architecture générale du système

Le système en tant que tel est le robot. Le but est de créer un programme suffisamment abouti pour qu'il permette au robot de trouver la sortie d'un labyrinthe. Le programme devra suivre une structure algorithmique utilisant la théorie des graphes. Le robot utilisera son sonar pour détecter les obstacles. Ensuite, nous utiliserons la caméra de Nao pour pouvoir détecter les carrefours du labyrinthe, à l'aide de logiciels permettant la détection de contours ou de contrastes. Grâce à la détection des carrefours, nous pourrons créer un graphe du labyrinthe et résoudre le labyrinthe.

1.6 Description des fonctionnalités

Le développement du module permettant à Nao de sortir d'un labyrinthe suivra cinq grands axes. Ces grands axes sont liés à des problématiques matérielles.

- La perception, l'acquisition. Cela se fera par le biais des capteurs,
- les actions que Nao est capable de faire, nous parlons ici de ses effecteurs,
- la communication,
- la mémorisation, liée étroitement à la capacité de la mémoire physique de Nao,
- la capacité à prendre des décisions.

Lors du développement, nous pourrons distinguer quatre grandes fonctions :

- Fonction d'acquisition et traitement vidéo
- Détection et évitement des obstacles
- Gestion du graphe et du repère 2D
- Gestion des chutes
- Parcours du labyrinthe

1.6.1 Définition de la fonction de parcours du labyrinthe

Identification de la fonction de parcours du labyrinthe

- Parcours du labyrinthe ;
- fonction principale, elle permettra au robot de parcourir le labyrinthe en utilisant la théorie des graphes ;
- priorité : primordiale.

Description de la fonction de parcours du labyrinthe

Cette fonction sera la fonction principale, elle appellera successivement les autres fonctions afin de déplacer le robot en respectant les retours que ces fonctions lui donnent. En effet, si la fonction d'acquisition et traitement vidéo retourne la présence d'un croisement, il faudra agir en conséquence. Cette fonction n'aura pas d'entrée ni de sorties.

1.6.2 Définition de la fonction d'acquisition et traitement vidéo

Identification de la fonction d'acquisition et traitement vidéo

- Fonction d'acquisition et traitement vidéo ;
- obtenir une image nette de l'environnement immédiat pour ensuite prendre une décision ;
- primordiale.

Description de la fonction d'acquisition et traitement vidéo

Cette fonction ne prendra rien en entrée. Pour ce qui est de sa sortie, elle renverra une information sur l'environnement immédiat enregistré par la caméra du robot. La fonction récupèrera un flux vidéo, les images seront analysées afin de connaître l'environnement immédiat du robot. Nous enverrons une information différente selon le résultat de la détection (si nous détectons un croisement et le nombre de sorties du dit croisement, si nous détectons un cul-de-sac).

1.6.3 Définition de la fonction de gestion du graphe et du repère 2D

Identification de la fonction de gestion du graphe et du repère 2D

- Gestion du graphe et du repère 2D ;
- recevoir les informations d'acquisition afin de maintenir le graphe et le repère 2D ;
- primordiale.

Description de la fonction de gestion du graphe et du repère 2D

Cette fonction sera appelée lorsque la fonction d'acquisition retournera un lieu remarquable du labyrinthe. La fonction de gestion du graphe et du repère 2D aura pour finalité de créer et maintenir le graphe et le repère 2D tout au long de l'exploration.

1.6.4 Définition de la fonction de détection et évitement des obstacles

Identification de la fonction de détection et évitement des obstacles

- Détection et évitement des obstacles ;
- obtenir une information précise venant des sonars du robot et indiquant la distance d'un obstacle ;
- primordiale.

Description de la fonction de détection et évitement des obstacles

Cette fonction fera appel aux sonars de Nao. Ces derniers renverront une distance en centimètre, distance séparant Nao d'un quelconque obstacle. Il nous faudra exploiter ces informations pour permettre à Nao de ne pas entrer en collision avec les murs du labyrinthe. Nous pourrons fixer un seuil en centimètres en dessous duquel il faudra mettre en place des mesures d'évitement.

1.6.5 Définition de la fonction d'après chute

Identification de la fonction de gestion des chutes

- Gestion des chutes ;
- doit vérifier si le robot a chuté ou non et permettra au robot de se relever et de reprendre l'exploration s'il a chuté ;
- primordiale.

Description de la fonction d'après chute

Cette fonction n'aura pas de données en entrée, elle sera appelée lorsque le robot aura chuté. Elle lui permettra de se relever et de reprendre l'exploration du labyrinthe.

1.7 Conditions de fonctionnement

Le but du projet est de faire en sorte que le robot évolue dans un labyrinthe. Il faudra donc s'assurer que le labyrinthe soit construit, que le robot soit chargé et que le module programmé soit en mémoire dans le robot. Il faudra aussi que le robot soit accessible par wifi, pour pouvoir commander l'exécution du module programmé.

1.7.1 Performances

Le robot doit être capable de répondre en temps réel aux stimuli de son environnement. Les temps de réponse doivent être raisonnables, le robot doit se mouvoir de manière fluide dans le labyrinthe. Le temps de calcul qui permettra de provoquer le mouvement ne doit pas excéder quelques secondes. L'acquisition sera faite en permanence, ainsi que les calculs permettant d'obtenir une information sur l'état de l'environnement.

1.7.2 Capacités

Nao dispose d'un panel de capteurs variés. Des capteurs de pressions aux caméras sur sa tête, ils ont tous une utilité bien définie. Toutefois, sur l'utilisation de certains nous pouvons observer quelques limites qui contraignent leur utilisation. Par exemple, nous nous rendons compte que les sonars du robot ont une portée plutôt courte. Le robot a aussi tendance à glisser sur certains revêtements, bien qu'il dispose de moyens de gérer son équilibre, le risque de chute est bien présent. Les limites sont claires : le robot peut glisser et chuter et les capteurs peuvent être difficiles à utiliser pour la résolution.

1.7.3 Modes de fonctionnement

Le robot s'allume par simple pression du bouton présent sur sa poitrine. Le programme que nous développerons, lui, ne sera pas lancé dès le démarrage du robot. Il faudra se connecter en SSH au robot et demander l'exécution du module programmé que nous aurons mis dans la mémoire du robot. Nous essayerons de faire en sorte qu'à l'allumage du robot le programme se lance automatiquement. Nous pouvons aussi imaginer par exemple une application Android qui permettrait de lancer le programme à distance.

1.7.4 Contrôlabilité

Pour suivre le bon fonctionnement du programme, en premier lieu nous observerons le comportement du robot lorsqu'il exécutera le programme. Ensuite, nous ferons en sorte d'enregistrer dans des fichiers de log les prises de décisions du robot lorsqu'il rencontrera une situation à laquelle il devra répondre par un choix (exemple : un carrefour). Nous garderons ces logs dans le but de pouvoir vérifier le bon fonctionnement du robot.

1.7.5 Sécurité

Le robot est accessible en réseau seulement via Choregraphe, en SSH ou par un câble ethernet. Nous pouvons aussi accéder à sa mémoire par USB. Toutefois, le programme que nous développerons ne permettra pas de fournir plus de connectivité au robot donc il n'y aura pas plus de possibilités de se connecter.

1.7.6 Intégrité

A tout moment, le robot peut perdre l'équilibre et chuter. Toutefois, des mesures de protection sont prévues de base lorsque le robot détecte une chute. De notre côté, nous aurons à détecter que le robot a détecté une chute et agir en conséquence. Cela se fera par faire se relever le robot et lui faire reprendre son exploration du labyrinthe.

Plan de développement

2.1 Découpage du projet en tâches

2.1.1 Tâche 1 : Découverte du projet, rapport et guide projet collectif année dernière

Description de la tâche

Cette tâche consiste à découvrir le projet. Plusieurs rapports ont déjà été faits sur Nao lors d'un projet collectif d'un groupe de quatrièmes années. Il faudra donc étudier les différents rapports déjà produits et les documents fournis par la société Aldebaran sur leur site internet.

Estimation de charge

Tâche estimée à un jour homme.

2.1.2 Tâche 2 : Prise en main du robot : formation

Description de la tâche

Formation planifiée avec M. Rousseau. M. Rousseau présente le robot et ses différentes fonctionnalités. Parcours et utilisation des différents capteurs. Découverte et utilisation de Choregraphe et Naosim. Utilisation du mode apprentissage du robot.

Estimation de charge

Tâche estimée à un jour homme.

2.1.3 Tâche 3 : Prise en main de la suite logicielle fournie par Aldebaran (Choregraphe, Naoqi etc.)

Description de la tâche

Téléchargement et installation des différents composants de la suite logicielle fournie par Aldebaran. Cette tâche consistera à étudier ces composants et à apprendre leur fonctionnement par différents tests et implémentations.

Estimation de charge

Tâche estimée à deux jours homme.

2.1.4 Tâche 4 : Essai des différents capteurs du robot et étude des possibilités

Description de la tâche

Découverte et utilisation des différents capteurs du robot. Étude des avantages et inconvénients offerts par chacun d'entre eux pour le développement du projet.

Estimation de charge

Tâche estimée à deux jours homme.

2.1.5 Tâche 5 : Prise en main de la programmation du robot

Description de la tâche

Cette tâche consiste à étudier comment programmer le robot en C++ et en Python. Il s'agit de se familiariser avec la programmation sur Nao via Naoqi sans passer par Choregraphe.

Estimation de charge

Tâche estimée à deux jours homme.

2.1.6 Tâche 6 : Rédaction du cahier de spécifications système

Description de la tâche

Rédaction du cahier de spécification système du projet.

Livrables

Il y aura un seul livrable : le cahier de spécifications système.

Estimation de charge

Tâche estimée à quatre jours homme.

2.1.7 Tâche 7 : Structure algorithmique : comment réaliser l'algorithme de résolution de labyrinthes

Description de la tâche

Il faut étudier l'approche algorithmique du projet. Le parcours du labyrinthe correspondra à un parcours de graphe, il faudra donc étudier la théorie des graphes et l'appliquer au parcours de labyrinthe. Une étude de plusieurs documentations pourra être effectuée, plusieurs documents présents sur la IEEE CSDL sont en relation avec le projet.

Estimation de charge

Tâche estimée à quatre jours homme.
Structure algorithmique : comment réaliser l'algorithme de résolution de labyrinthes

2.1.8 Tâche 8 : Coder un premier programme interprétable en C++

Description de la tâche

Cette tâche consiste à créer un premier programme en C++ utilisant les capteurs du robot et réalisant différentes actions. Le but est de pouvoir voir comment utiliser les capteurs en C++ et cerner les contraintes qui pourraient exister. Si jamais la programmation en C++ s'avère trop contraignante, une programmation en Python pourra être envisagée.

Estimation de charge

Tâche estimée à trois jours homme.

2.1.9 Tâche 9 : Prendre des mesures pour stabiliser le robot, éviter les glissements et pertes d'équilibre

Description de la tâche

Le but de cette tâche est de prendre une décision sur les moyens de stabiliser le robot. Il a été observé que le robot glisse facilement sur certaines surfaces ce qui peut entraîner des déplacements non désirés et des pertes d'équilibre, il faudra donc pouvoir pallier ce problème.

Estimation de charge

Tâche estimée à deux jours homme.

2.1.10 Tâche 10 : Coder l'implémentation de l'algorithme sur le robot

Description de la tâche

Cette tâche est primordiale. Il s'agit de développer l'implémentation de l'algorithme du robot, implémentation qui lui permettra de parcourir le labyrinthe.

Livrables

Module à déployer sur le robot. Ce module est le code C++ ou Python compilé.

Estimation de charge

Tâche estimée à six jours homme.

2.1.11 Tâche 11 : Tester le code

Description de la tâche

Cette tâche consiste à tester le code sur le robot et observer le bon fonctionnement grandeur nature. Il faudra surtout veiller à ce que l'intégrité physique du robot ne soit pas menacée.

Estimation de charge

Tâche estimée à cinq jours homme.

2.1.12 Tâche 12 : Construction du labyrinthe

Description de la tâche

Mise en place du labyrinthe.

Estimation de charge

Tâche estimée à cinq jours homme.

2.2 Planning

Voici le diagramme de Gantt de mon projet :

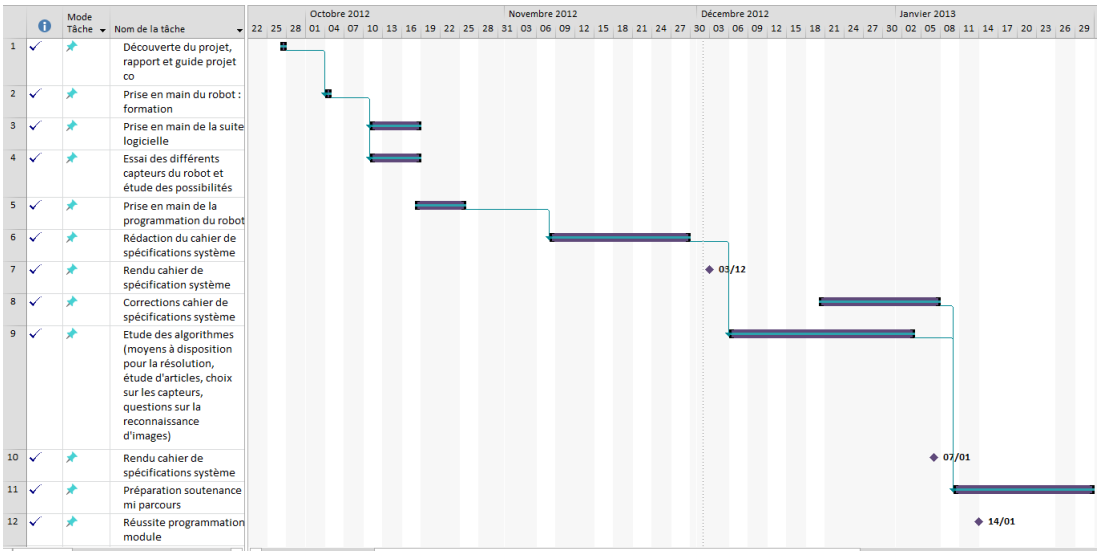

FIGURE 2.1 – Diagramme de Gantt de mon projet, première partie

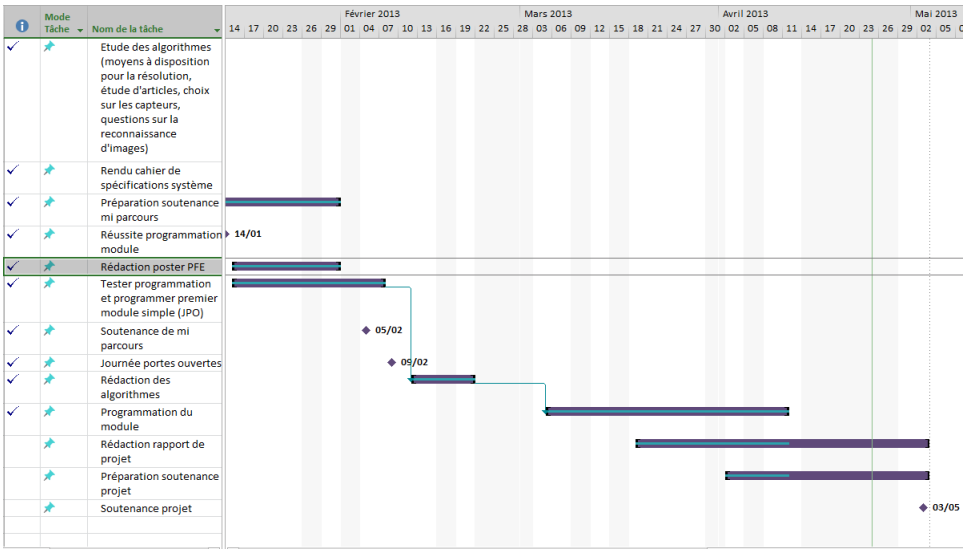

FIGURE 2.2 – Diagramme de Gantt de mon projet, deuxième partie

Méthode pour créer un module autonome sur le robot humanoïde Nao

2.3 Préambule

Ce document présente une méthode pour créer un module développé en C++ dans le but de l'exécuter sur la mémoire du robot ou à distance.

Ce module sera réalisé en cross-compilation, c'est-à-dire que les étapes de compilation, création de liens, etc. jusqu'à la création de l'exécutable seront réalisées sur un système d'exploitation A dans le but d'utiliser l'exécutable sur un système d'exploitation B.

Ce manuel est réalisé pour le robot Nao de l'école Polytech Tours.

ATTENTION

Avant toute chose, notez bien que pour que le module créé fonctionne sur le robot, la compilation du module devra absolument se faire sur un environnement **Linux**.

En cas de problèmes, référez-vous à la documentation en ligne fournie par Aldebaran Robotics.

2.4 Réalisation du module

2.4.1 Créer un worktree

Il suffit de se placer à l'endroit où vous voulez travailler (nous appellerons ce dossier workspace) et de faire la commande :

- **qibuild init**

2.4.2 Créer un projet

Placez-vous dans votre répertoire de travail (workspace) et exécutez la commande :

- **qibuild create nomprojet**

Si tout s'exécute normalement, un dossier nomprojet sera créé dans votre workspace.

2.4.3 Configurer et construire le projet

Toujours dans le workspace, exécutez les commandes suivantes :

- **qibuild configure nomprojet**
- **qibuild make nomprojet**

Une fois ces commandes exécutées, vous disposerez de fichiers automatiquement générés dans un dossier se nommant "nomprojet".

2.4.4 Etape de cross-compilation pour le robot

Cette partie est importante, car ce sont ces manipulations en particulier qui nous permettront de créer un fichier qui pourra être exécuté sur le robot.

En tout premier lieu, il faut récupérer la **cross-toolchain** qui correspond à notre version du robot (atom pour les versions V4 ou supérieur, geode pour les autres). Il s'agit d'un dossier compressé qui devrait s'appeler (pour le Nao de Polytech Tours) **"nao-atom-cross-toolchain-1.14"**. Vous trouverez ce dossier en téléchargement sur le site users de Aldebaran Robotics (http ://users.aldebaran-robotics.com). Les logs sont situés au dos de la notice dans la mallette de Nao (vous ne pourrez pas vous y inscrire, il vous faut obligatoirement les logs situés au dos de la notice).

Une fois ce dossier récupéré et décompressé, vous serez en mesure de créer la cross-toolchain. Il vous suffira de taper la commande suivante :

- **qitoolchain create nomDeLaChaine /chemin/nao-atom-cross-toolchain-1.14/toolchain.xml**

Vous pouvez fixer le nom de la chaine à votre guise (ex : cross-atom) et "chemin" correspond au chemin permettant d'accéder à votre dossier nao-atom-cross-toolchain-1.14.

Une fois cette étape validée, il vous faudra exécuter les deux commandes suivantes :

- **qibuild configure -c nomDeLaChaine nomDuProjet**
- **qibuild make -c nomDeLaChaine nomDuProjet**

NB : L'argument nomDuProjet n'est pas nécessaire si vous vous trouvez dans le dossier correspondant à votre projet.

Une fois toutes ces étapes validées, vous disposez d'un fichier exécutable sur la mémoire de Nao. Ce fichier se trouve dans le dossier de votre projet, dans le dossier : **build-nomDeLaChaine/sdk/bin**

Si vous souhaitez ajouter votre code, il vous suffira de **modifier les fichiers sources** de votre projet en plaçant les vôtres à la place. Il vous faudra exécuter de nouveau la commande :

- **qibuild make -c nomDeLaChaine**

Et l'exécutable sera de nouveau généré.

2.4.5 Exécution du programme

Pour pouvoir exécuter le programme que vous venez de créer, vous disposez de deux méthodes :

- **Exécuter à distance**

Il vous faudra appeler votre programme de la manière suivante :

- **./monProgramme @IPduRobot port**

En supposant que vous gérez l'adresse IP et le port dans votre programme via les variables **argc et argv.**

- **Exécuter sur le robot**

Ici, il vous faudra tout d'abord **copier le programme** sur votre robot.

Nao est accessible via **SSH**, vous pouvez donc utiliser la commande **scp** pour copier votre fichier sur le robot. Vous pouvez aussi atteindre la mémoire de Nao en branchant un **câble USB** derrière sa tête ou encore en utilisant le gestionnaire dévolu à cette tâche dans **Choregraphe** (j'ai pour ma part utilisé SSH en copiant le fichier grâce à la commande scp).

Nous cherchons à faire en sorte que **le programme se lance au démarrage du robot.** Pour ce faire, il faut aller éditer le fichier **autoload.ini** qui se trouve dans la mémoire de Nao à cet emplacement :

- **/home/nao/naoqi/preference**

Il vous faudra éditer le fichier **autoload.ini** en respectant la syntaxe indiquée à l'intérieur du fichier.

Pour que notre programme s'exécute au démarrage du robot, nous allons donc écrire cette ligne en dessous du tag **[program]** dans le fichier autoload.ini (le tag [program] indique les exécutables à lancer au démarrage de Nao) :

- **/chemin/menant/à/votre/programme/monProgramme**

Nous rappelons que votre programme doit se trouver dans la mémoire interne de Nao.

ATTENTION

Afin que la manipulation fonctionne et que votre programme se lance au démarrage de Nao, votre programme doit gérer les arguments **argc et argv** et particulièrement à travers ces deux arguments correspondant aux arguments envoyés en ligne de commande, votre programme doit gérer ces deux options :

- **– pip et – pport**
- **– pip** correspond à l'adresse IP du robot et **– pport** correspond au port.

La raison en est simple : si ces deux options ne sont pas gérées dans votre programme, **il ne se lancera pas.** En effet, lorsque Nao appelle automatiquement un programme, **il met automatiquement ces deux arguments** (–pip et –pport) à l'appel de l'exécutable. Donc, si votre programme ne gère pas ces arguments, il ne fonctionnera jamais de manière automatique sur Nao.

Exemple :

Si vous placez le chemin de votre programme dans la mémoire de Nao sous le tag [program] dans le fichier autoload.ini de Nao, votre programme sera automatiquement appelé au démarrage de Nao de la manière suivante :

- **./monProgramme –pip –pport**

Avec –pip l'adresse IP du robot et –pport le port correspondant.

Il est donc important de gérer ces deux arguments. Comme par exemple ci-dessous :

```
// Sepcified IP or PORT for the connection
if (argc == 5)
{
  if (std::string(argv[1]) == "--pport"
      && std::string(argv[3]) == "--pip")
  {
    pport = atoi(argv[2]);
    pip = argv[4];
  }
  else if (std::string(argv[3]) == "--pport"
           && std::string(argv[1]) == "--pip")
  {
    pport = atoi(argv[4]);
    pip = argv[2];
  }
  else
  {
    std::cerr << "Wrong number of arguments!" << std::endl;
    std::cerr << "Usage: mymodule [--pip robot_ip] [--pport port]" << std::endl;
    exit(2);
  }
}
```

FIGURE 2.3 – Exemple de code fourni par Aldebaran Robotics qui montre comment gérer les arguments –pip et –pport

2.5 Conclusion

Si vous avez suivi toutes ces étapes sans encombre, vous devriez avoir réalisé un module cross-compilé exécutable sur le robot. Je ne certifie pas l'exactitude de ce document compte tenu des mises à jour rapides et de l'évolution de Nao. En cas de problèmes, veuillez consulter la documentation disponible (en anglais seulement) sur Internet sur le site de la société Aldebaran Robotics.

Document écrit le 18/01/2013.

Ces pages ont été particulièrement utiles pour la rédaction de ce document :
- http://www.aldebaran-robotics.com/documentation/qibuild/qibuild_in_five_minutes.html
- http://www.aldebaran-robotics.com/documentation/dev/cpp/tutos/create_module.html

Poster réalisé pour mon projet de fin d'études

Projet de fin d'études

Résolution de labyrinthe avec le robot humanoïde Nao

Damien DETOEUF (encadrants : *P. Gaucher, J-L. Bouquard*)

Objectif

• Programmer le robot Nao à l'aide d'un algorithme de graphe afin qu'il soit capable de parcourir un labyrinthe et d'en trouver la sortie.

Problématiques

• Robotique :
  • Utilisation des capteurs
  • Découverte du robot
• Algorithmique :
  • Théorie des graphes
  • Apprentissage
• Recherche de points importants dans une image

Nao

• Processeur Intel Atom
• Capacités audios et visuelles (reconnaissance d'objets, de sons, de visages…)
• Réflexes naturels (gestion des chutes, rigidité des membres…)
• Capteurs de pression, capteurs tactiles, bumpers, sonars…
• Système d'exploitation Linux embarqué (OpenNAO)

Département Informatique – Polytech'Tours, 64 av. Portalis 37200 Tours
www.polytech.univ-tours.fr

POLYTECH TOURS *Informatique*

FIGURE 2.4 – Poster réalisé pour mon projet de fin d'études

Références

- Le site web de la société Aldebaran Robotics : http://www.aldebaran-robotics.com/ . J'y ai trouvé toute la documentation relative à Nao, concernant son utilisation, les définitions de certains termes comme Naoqi et Qibuild. J'y ai aussi trouvé la documentation des méthodes fournies pour utiliser les différents matériels dont dispose Nao.
- Les articles présents sur le référencement IEEE CSDL : http://www.computer.org/ . L'école nous offre la possibilité d'accéder gratuitement à beaucoup d'articles sur différents sujets. J'y ai trouvé quelques articles qui me serviront lors de ma résolution du problème et qui seront analysés plus tard.
- Le site de Stack Overflow : http://stackoverflow.com/ . Ce site regroupe une communauté importante de programmeurs,sur de très nombreux domaines. Les personnes qui ont un problème peuvent poser leurs questions librement et des réponses leur sont apportées. Du fait de la taille de la communauté les sujets abordés sont variés et j'ai pu trouver des informations concernant Nao.
- Le rapport de projet collectif ainsi que le guide utilisateur rédigés par le groupe 1 de projet collectif de l'année 2011 - 2012.
- Le site web de M. F.Legrand qui m'a permis d'en apprendre plus sur la transformée de Hough et la transformée de Hough probabiliste : http://www.f-legrand.fr/scidoc/index.html
- Le site web et la documentation de la librairie OpenCV, je me suis servi de cette documentation pour réaliser les traitements d'images : http://opencv.org/

Glossaire

- Choregraphe : Logiciel qui est utilisé sur un ordinateur et permet de contrôler le robot à distance afin de lui faire réaliser certains comportements pré programmés.
- C++ : Langage de programmation permettant la programmation orienté objet et la programmation procédurale.
- Graphe : Ensembles de sommets (noeuds du graphe) reliés entre eux par des arcs.
- Nao : Robot créé par la société Aldebaran Robotics.
- Naoqi : Logiciel exécuté en permanence sur Nao et permettant de le contrôler. Permets d'accéder aux primitives de programmation de Nao.
- Python : Langage de programmation permettant la programmation impérative et la programmation orientée objet.
- Qibuild : Permets la compilation des sources utilisant les méthodes qui manipulent le matériel de Nao.
- Sonar : Matériel permettant la détection et l'obtention de la distance d'objets par rapport à un point grâce à l'envoi et la réception d'ondes sonores.
- Théorie des graphes : Ensembles de principes exploitant les graphes.

Résolution de labyrinthe avec le robot humanoïde Nao

Département Informatique
5e année
2012 - 2013

Rapport de projet de fin d'études

Résumé : Le but du projet est de faire en sorte que le robot Nao soit capable de sortir d'un labyrinthe de manière intelligente. Premièrement, la découverte et la prise en main du robot ont été étudiées. Ensuite, il a fallu étudier les différents capteurs qui permettraient de réaliser l'exploration du labyrinthe et l'analyse de l'environnement. Puis, il fallait donner un comportement intelligent au robot. J'ai donc utilisé l'algorithmique et en particulier la théorie des graphes. Enfin, le projet comporte aussi une étude du traitement d'images avec Nao afin d'analyser le labyrinthe.

Mots clefs : Nao, Robotique, Projet de fin d'études, Polytech Tours, Labyrinthe, Résolution de labyrinthes, Traitement d'images, Théorie des graphes, Algorithmique

Abstract: The goal of the project is to make Nao get out of a maze in an intelligent way. First, the discover of the robot has been studied. Then, I had to study the different sensors which would allow to realize the maze's exploration and the analysis of the environment. Moreover, I had to give an intelligent behaviour to the robot. So, I used algorithmics and in particular the graph's theory. Finally, the project also contains a study of image processing with Nao in order to make an analysis of the maze.

Keywords: Nao, Robotics, End-of-studies project, Polytech Tours, Maze, Maze's resolution, Image processing, Graph's theory, Algorithmic

---

Encadrants
Jean-Louis BOUQUARD
jean-louis.bouquard@univ-tours.fr
Pierre GAUCHER
pierre.gaucher@univ-tours.fr

Étudiants
Damien DETOEUF
damien.detoeuf@etu.univ-tours.fr

DI5 2012 - 2013

Université François-Rabelais, Tours