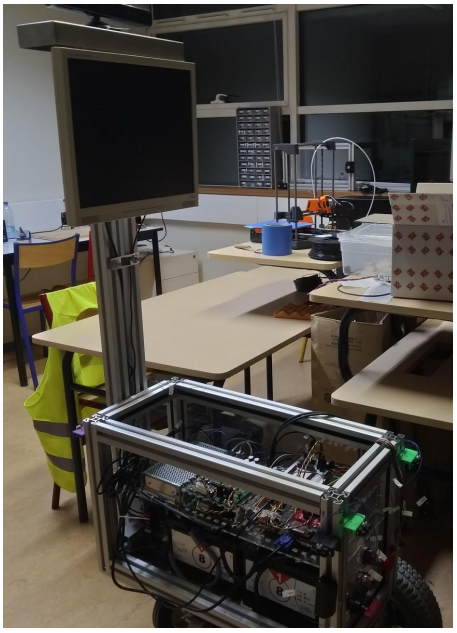
Université de Lille
POLYTECH LILLE
Polytech Lille – Avenue Paul Langevin – 59655 Villeneuve d'Ascq cedex

Rapport Projet IMA3/4

P3 : Robot autonome Centaure



Professeurs encadrants:
BOE Alexandre
REDON Xavier
VANTROYS Thomas

BENAYED Samuel, EVRARD Théo, FROISSART Clément, RINGOT Loïc, SALINGUE Valériane, SANCHEZ Pierre

Remerciements

Nous adressons nos remerciements à Xavier REDON, l'un de nos enseignants encadrants, pour l'aide qu'il nous a apporté tout au long de ce projet, l'apport de matériel et ses réguliers conseils afin d'avancer dans la bonne direction, Thierry FLAMEN pour son assistance lors de nos impressions de cartes, de notre câblage mais également pour son apport de matériels.

Nous remercions également Alexandre BOÉ et Thomas VANTROYS, nos autres enseignants encadrants qui ont suivi l'avancement du projet depuis ses débuts et qui ont donné leur avis afin que nous apportions les corrections nécessaires, qui ont challengé nos solutions et nos choix technique afin que ces derniers soient les meilleurs.

Nos remerciements vont également à Jérémie DEQUIDT et Florian CHEVALIER pour leur expertise en deep learning et en électronique de puissance.

Table des matières

Introduction

Au cours du projet IMA3/4, nous avons travaillé sur le projet de robot autonome, le Centaure. Il s'agit d'un robot d'accueil. Sa fonction principale est d'interagir avec des personnes et de leur proposer ses services : informer ou accompagner ces personnes vers le lieu souhaité. Pour cela, nous travaillons sur différents aspects de ce robot. Tout d'abord, nous le recâblons entièrement. Ensuite, nous réalisons un nouveau code pour ses déplacements. Enfin, nous travaillons sur la manière de commander ce robot (autonome ou téléguidé) et sur sa façon d'interagir avec les utilisateurs.

I. Présentation du projet

a) Etat des lieux

Nous récupérons le robot au début du semestre 6 après le travail des IMA5 dans le cadre de leur projet de fin d'études. Notre objectif est de le rendre autonome et intelligent. Il doit pouvoir communiquer avec ce qui l'entoure et se déplacer en toute sécurité. Cependant, lors de l'état des lieux du robot, d'autres missions vont s'ajouter. Il s'agit d'un robot avec deux roues motrices alimentées par des moteurs de fauteuil roulant, et une roue folle située à l'avant. Les pneus des roues motrices semblent un peu dégonflés. Pour alimenter ces deux moteurs, le robot embarque deux batteries de 12V branchées en série. Il y a aussi deux variateurs de vitesse reliés à l'Arduino. Un convertisseur 24V-12V alimente le PC, la Kinect et l'écran. Des capteurs sont installés sur des supports, mais non utilisés. Ces supports ne sont pas fixes et sont tranchants. Aussi, l'écran ne reste allumé que quelques secondes. Nous remarquons aussi des défauts dans le câblage du robot tel que des câbles volants (qui ne restent pas fixés au shield), des gaines entaillées et réparées avec du scotch. Côté serveur, nous découvrons leur code pour en comprendre le fonctionnement. Nous découvrons aussi le code qui sert à manipuler le robot. Celui-ci ne prend pas en compte les capteurs. Nous devons le modifier en conséquence. Nous testons de commander le robot avec le joystick. Celui-ci est défectueux et devra être remplacé.

b) Cahier des charges

Le robot Centaure est un robot d'accueil. Il peut avoir une discussion avec une personne et pouvoir la guider au sein de Polytech. Pour cela, nous devons réaliser plusieurs tâches.

Tout d'abord, suite à l'état des lieux, une mission principale se dessine, sécuriser le robot en retravaillant le câblage. Nous devons aussi réaliser un nouveau shield pour un meilleur maintien des câbles. Ensuite, nous devons le rendre commandable de trois manières différentes : à l'aide d'un joystick, à l'aide d'un serveur web et de manière autonome. Premièrement, nous devons réaliser le code de l'Arduino pour commander les moteurs ainsi que refaire un joystick. Deuxièmement, nous devons retravailler le serveur pour commander le robot à distance. Enfin, nous devons le rendre autonome et donc prendre en compte les capteurs dans le code de l'Arduino ainsi que reconnaître les obstacles à l'aide de la Kinect. Enfin, nous devons travailler la communication avec le robot, c'est à dire l'interface utilisateur, la synthèse vocale et la reconnaissance faciale et vocale.

c) Travail réalisé lors de la première année

Pendant cette première année, nous avons découpé le travail en deux parties : la partie câblage et la partie software. Nous commençons par la partie câblage. Nous avons tout d'abord résolu le problème de l'écran. Il était sous-alimenté. Nous avons donc commandé un deuxième convertisseur, celui-ci avec une tension de sortie réglable. Ainsi le convertisseur 24-12V alimente la Kinect et le PC, et celui avec une tension de sortie réglable alimente l'écran. En effet, l'écran nécessite 13V pour se maintenir allumé. Nous avons essayé d'améliorer le câblage en usinant une boîte contenant le PC, pour permettre à tous les câbles d'être regroupés, ainsi qu'en réalisant de nouvelles soudures plus propres. Ceci n'étant pas suffisant, nous avons, lors du S8, dû décâbler intégralement le robot pour pouvoir le recâbler entièrement sans contraintes (hormis gabarit du robot). Nous avons remplacé l'ancien shield de l'Arduino. Contrairement à son prédécesseur, le nouveau shield comporte des borniers pour un meilleur maintien des câbles au sein du robot. Ce shield a été validé à la fin du S6 et a été terminé pendant le S7.

La partie software a également était amorcée durant ce semestre pour ne pas prendre trop de retard malgré la refonte du hardware. Nous avons été capable d'utiliser la bibliothèque OpenCV et la reconnaissance vocale de google afin de créer un programme en Python capable d'interagir avec un utilisateur. Malheureusement les programmes en Python semblent trop lourds à supporter pour le PC embarqué.
L'interaction avec l'utilisateur est mise de côté durant le S7 afin de se concentrer sur l'utilisation de la Kinect. Après avoir exploré plusieurs solutions permettant d'exploiter le flux vidéo du capteur Kinect (utilisation du C# et du SDK de Microsoft notamment) nous nous sommes finalement orientés vers l'utilisation du C et de la libfreenect, permettant un traitement d'image de plus "bas niveau" et conçu spécialement pour les systèmes Unix.

Ayant présenté un robot complètement démonté à la fin du S7, voici les objectifs pour le S8 :

- Refaire des parois au robot et câbler l'intégralité du robot.
- Relier le serveur Web à l'Arduino pour récupérer les données d'un joystick virtuel.
- Exploiter le flux vidéo de la Kinect afin permettre l'autonomie du robot.
- Adapter et améliorer les programmes d'interaction Homme-Machine afin d'être supportés par le PC du robot.

II. Architecture du robot

a) Réalisation du châssis

La modification la plus importante du robot est son sens de circulation. Nous avions une roue folle à l'avant qui déstabilisait le robot. Nous avons donc retourné le mât sur lequel est fixé l'écran et la Kinect. Cela nous permet en plus d'avoir une distance minimale entre un possible interlocuteur et la Kinect et donc d'éviter tout problème de détection. Celle-ci a en effet une distance minimale de fonctionnement. Ensuite, nous profitons du fait que le robot soit complètement vidé pour réarranger les différents éléments qui le composent. Tout d'abord, nous modifions la fixation de l'étage. Avant, cette plaque reposait sur les batteries. Nous avons décidé de réaliser des encoches dans les plaques extérieures pour pouvoir emboîter l'étage. Cet étage est aussi maintenu grâce à la plaque sur laquelle sont fixés les fusibles. Toutes ces plaques ont été réalisées sur le logiciel Inkscape, la plupart modifiées à partir des fichier que nous avons récupéré du groupe précédent. Nous avons aussi modifié le bloc PC - Arduino. La boîte en bois a été remplacée par une boîte en plexiglas, qui suit la logique du châssis, avec une meilleure disposition des éléments ainsi qu'une meilleure aération pour éviter la surchauffe.

Enfin, les derniers éléments à ajouter sur les robot sont les capteurs. Nous avons complètement repensé le mode de fixation de ceux-ci. Tout d'abord, nous avons modifié leur emplacement. Pour une meilleure sécurité, deux capteurs sont fixés à l'avant du robot et deux sont fixés à l'arrière. Les deux derniers sont fixés de part et d'autre du robot. Ainsi, nous avons une couverture optimale pour notre robot. Nous avons aussi modifié leur support en les modélisant sur le logiciel Onshape. Comme on peut le voir sur l'image ci-dessous, les capteurs sont vissés sur ce support, qui se glisse à l'intérieur du profilé et y est fixé. Enfin, une rainure permet de faire passer les câbles des capteurs proprement jusqu'à rejoindre le trou prévu à cet effet dans les plaques extérieures.

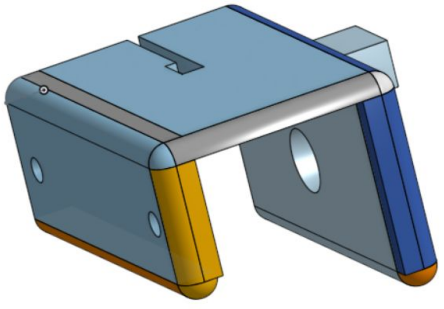


Figure 1 : Capteur modélisé sur Onshape

b) Câblage du robot

Le constat du S7 est le suivant :

- Le système électrique est trop brouillon, il est très compliqué de s'y retrouver.
- Les conducteurs rigides ne correspondent pas aux usages de la robotique mobile.
- Les équipements de puissances (contacteurs, relais, borniers de puissance) sont réparties sur les deux étages du robot, résultant en une difficulté à effectuer des opérations de maintenance.
- Les équipements de sécurité sont trop disparates.

De ce constat nous avons défini les attentes vis-à-vis du câblage pour le S8 :

- La nécessité d'organiser le câblage.
- L'utilisation de conducteurs souples est impérative.
- Les actionneurs de sécurité doivent être regroupés et facile d'accès.
- La séparation des deux étages du robot doit être clairement établie et visible.

Nous avons donc commencé par réorganiser le placement des équipements composant le robot. Ainsi, l'étage inférieur (étage de puissance) se voit composé des deux batteries 12V, les équipements de sécurité (Sectionneur/Coupe Batterie, Arrêt d'Urgence) et des équipements de puissance (Contacteurs, Relais, Porte fusible, Borniers à fort ampérage). L'intérêt de cette organisation est la proximité des équipements interagissant entre eux, ce qui limite la taille des conducteurs, et, par extension, limite les problèmes de Compatibilité Électromagnétique (CEM) dans les conducteurs.

L'étage supérieur (étage de commande) accueille les deux convertisseurs DC/DC, les deux variateurs de vitesse, l'ordinateur et l'Arduino du robot.

Les divers équipements étant installés, nous avons installés les goulottes, composants majeurs d'un câblage propre. Une fois découpées aux bonnes dimensions et installées dans le robot, nous pouvons réaliser le câblage. Toutefois, afin d'éviter la fâcheuse situation de couper un conducteur trop court, nous utilisons au préalable du ruban pour avoir une estimation des longueurs des conducteurs. Nous avons réalisé le câblage à partir de ces gabarits, en utilisant des conducteurs souples multibrins. Ces conducteurs simplifient la réalisation du câblage mais représentent surtout un intérêt vis-à-vis de la résistance mécanique du système. En effet, un conducteur multibrins est mieux maintenu dans un bornier qu'un conducteur à âme massive. Il était prévu d'ajouter un deuxième arrêt d'urgence (en série avec le premier) sur le dessus du robot, pour cette raison deux câbles reliés par un domino sont présents dans les goulottes, chaque câbles allant sur une borne du nouvel arrêt d'urgence.

Un guide de démontage du robot pour les opérations de maintenance a été réalisé *(Disponible sur le wiki projet P3).*

c) Commande du moteur

Lors de la récupération du projet, nos prédécesseurs utilisaient de vieux variateurs qui avaient un certain âge mais qui n'en restait pas moins fonctionnel. Ils sont également très imposants et un peu complexe d'utilisation. Nous avons choisi de les changer pour des versions plus petites et plus "simples" d'utilisation. Malheureusement le court délai pour passer nos commandes nous ont poussé à faire un mauvais choix pour ces nouveaux variateurs. Nous possédons donc deux variateurs ne convenant pas à notre utilisation car il manque des modules pour pouvoir les utiliser. Pour ne pas perdre plus de temps et d'argent, nous avons fait le choix de recâbler les anciens variateurs.

Le câblage des variateurs est également complexe car ils possèdent beaucoup de pins d'entrées/sorties et nous n'avons aucun schéma de câblage propre à notre disposition.

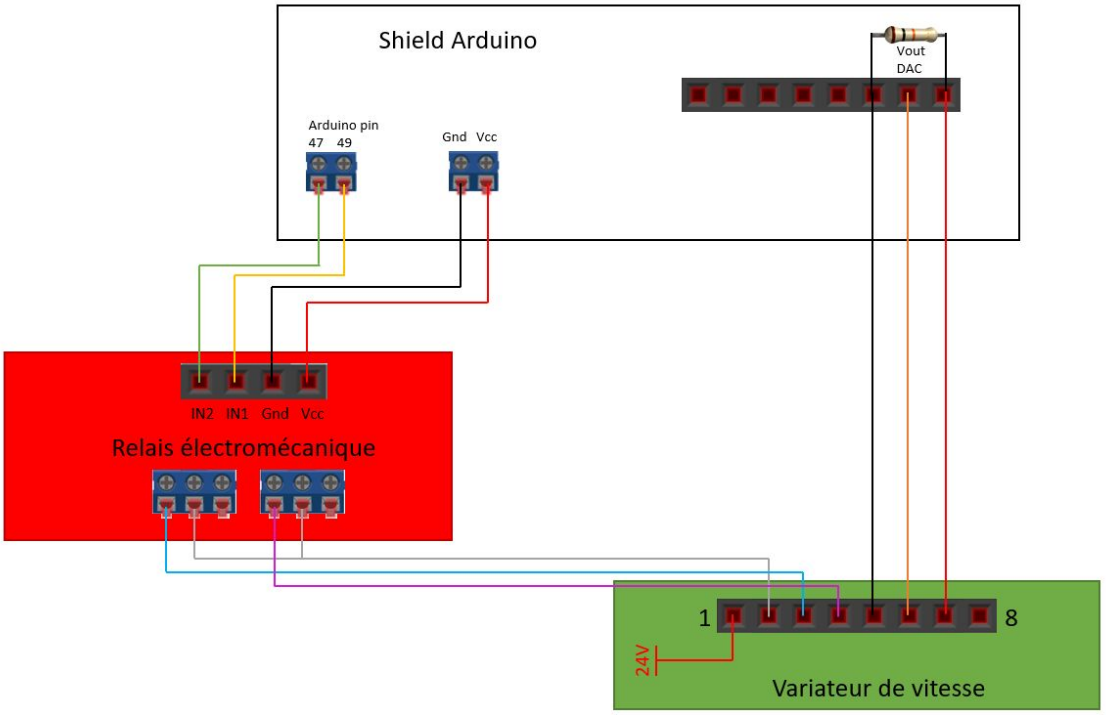


Figure 2 : Schéma de câblage des variateurs

Grâce a la datasheet nous avons pu établir ce schéma de câblage qui permet de contrôler un variateur grâce à notre shield Arduino ainsi qu'une carte de relais supplémentaire. Pour démarrer les contrôleurs moteurs il faut :

- 24V sur la pin 1.
- 0V sur les pins 3,4 et 8.
- Une résistance d'au moins 5K entre les pins 5 et 7.
- Une tension d'au minimum 40mV sur la pin 6.

La pin 1 du variateur est une clé d'anti-démarrage, il est nécessaire de lui transmettre 24V au démarrage pour ne pas mettre le variateur en mode erreur. La pin 2 est une sortie,

pour choisir le sens de rotation du moteur on redirige cette sortie vers la pin 3 ou 4. La pin 6 sert à faire varier la vitesse en envoyant une tension analogique comprise entre 40mV et 4,8V. Pour simplifier le câblage, nous avons rajouté des borniers sur le shield qui récupère les sorties des relais pour avoir une gaine contenant les 8 câbles partant du shield et arrivant sur les variateurs.

Concernant le programme Arduino nous avons fait le choix de repartir de zéro car le programme des anciens étudiants travaillant sur le robot était très lourd à comprendre de par sa complexité et sa taille. De plus repartir à zéro nous permet de créer nos propres protocoles de communication.

Le programme Arduino nous permet de lire sur deux ports séries, celui du joystick virtuel et celui de la Kinect. Nous devons également lire la consigne du joystick physique. Un protocole de priorité a donc été établie :

Joystick Physique > Capteurs > Joystick virtuel > Kinect

Nous avons fait le choix de prioriser le joystick physique au cas où un dépannage ou un déplacement forcé du robot serait nécessaire. Viens ensuite les capteurs qui sont nécessaires pour stopper rapidement le robot si un obstacle est trop proche. Nous avons fait le choix de ne pas prendre en compte les capteurs lors de l'utilisation du joystick physique pour ne pas stopper le robot si nous venions à le déplacer dans un espace limité. A l'inverse les consignes venant du joystick virtuel sont vérifiées par les capteurs par sécurité si un problème de réseau ou de serveur survient lors d'un déplacement dans une foule. La Kinect qui est la partie autonome vient en dernier dans cette liste de priorité car si une consigne est détectée sur le joystick physique ou virtuel on les juges plus importantes.

La consigne récupérée par l'Arduino correspond à la position x et y d'un joystick pour simplifier le programme et n'avoir qu'une fonction de calcul de consigne à envoyer aux moteurs. L'utilisation des capteurs se fait en tout ou rien, si un objet est détecté à moins de 40 cm du capteur, le robot s'arrête. Durant notre première soutenance des interrogations ont été émise sur le choix de la technologie (infrarouge) des capteurs de distance. Des tests ont été réalisés avec une luminosité plus ou moins intense et nous ont permis de garder ces capteurs car les valeurs restaient cohérentes. Les quelques variations de la tension de sortie de ces capteurs en fonction de la luminosité n'impactent pas le fonctionnement au vu de l'utilisation en tout ou rien.

La gestion de la priorité s'effectue de la façon suivante : on regarde si le joystick physique est à sa position initiale, s'il l'est, nous regardons si un message est en attente sur la liaison série, si aucune de ces deux conditions n'est présente, nous prenons en compte la consigne de la liaison série de la Kinect. Si l'une ou l'autre des conditions est présente, nous sélectionnons ses données comme consigne des moteurs. Si aucune donnée n'est reçue pendant 5 secondes sur ce channel de communication, nous reprenons la recherche de donnée classique.

III. Déplacement du robot

a) Développement du serveur

Un des axes principaux de notre S8 a été d'implémenter un serveur Web comprenant un joystick virtuel permettant le contrôle du robot en mode manuel et ce grâce à tout PC/Téléphone connecté au même réseau que notre PC embarqué dans Centaure. Nous avons pris comme base le précédent serveur qui avait été développé par des anciens IMA5 et qui d'après leur compte rendu et leurs vidéos proposées sur le Wiki, semblait totalement fonctionnel.

Ce Serveur est écrit en Node.JS qui est une plateforme logicielle en Javascript orientée vers des applications réseau événementielles. Ce choix a été fait par les IMA5 et est tout à fait justifié car cela convient totalement pour notre joystick qui fonctionne par évènements et via notre serveur HTTP. Pour lancer ce serveur et vérifier son efficacité nous avons tout d'abord réinstaller l'ensemble des plugins nécessaire à son démarrage sur notre machine. A cet effet nous avons utilisé le package manager spécialement conçu pour Node.JS : NPM. Ce package manager est très facile d'utilisation et se base notamment sur les dépendances de notre projet qui sont inscrites dans les fichiers package de notre archive, une simple commande « npm update » permet ainsi d'installer la grande partie des dépendances nécessaires, quelques dépendances sont à installer « à la main ».

Après l'installation des différents plugins, le lancement du serveur se fait sans erreurs et warning. Il faut bien entendu changer l'adresse IP et le port d'écoute parti serveur et client. Pour autant des problèmes sont vite remarqués : Tout d'abord le Joystick ne retourne pas dans sa position initiale lors du relâchement de ce dernier, ainsi pour arrêter le mouvement du robot il faut constamment positionner le joystick à sa position initiale. De plus les informations reçues du côté Arduino ne semblent pas optimisées.

Nous avons donc pris le code des IMA5 comme base et nous avons adapté ce code à notre utilisation. Voici l'architecture générale de notre serveur :

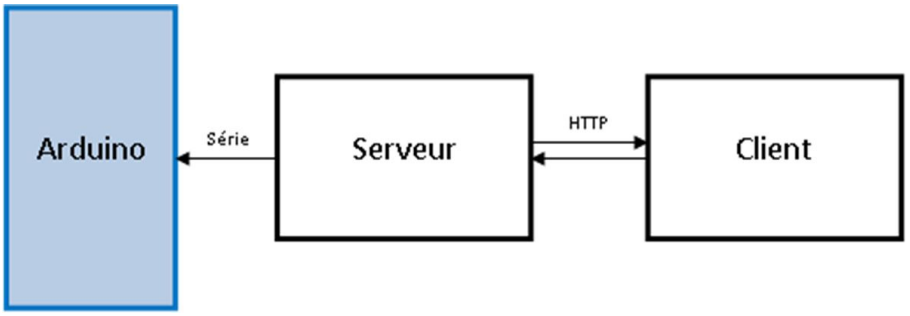


Figure 3 : Architecture du serveur

Parmi les modules de Node.js, nous retrouvons le module HTTP qui permet le développement de serveur HTTP. Dans notre cas nous établissons une communication entre un serveur et un client. Cette communication client/serveur était totalement fonctionnelle lors de la reprise du projet, les problèmes venaient de l'utilisation du joystick et de l'envoi sur la liaison série des données reçue de la part du client.

Notre première tâche a été de corriger le joystick. Pour cela nous avons raisonné sur les événements renvoyé par le joystick et ce grâce à la documentation du joystick que nous utilisons, dans notre cas : NippleJs (Documentation disponible à l'adresse suivante : https://github.com/yoannmoinet/nipplejs). Lorsque l'utilisateur appuie sur la zone active et relâche la zone active, les valeurs renvoyées sont celles de l'initialisation à savoir le joystick en position centrale (X=63,Y=63). Par la suite nous avons modifié la plage des valeurs renvoyées par le joystick pour ainsi correctement se lier à notre partie Arduino et être cohérent avec la partie Kinect. Voici la représentation des valeurs renvoyés par le joystick.

Les valeurs X et Y du joystick sont représentées sur 8 bits, ainsi le bit de poids fort nous permet de distinguer les valeurs X /Y et les 7 autres bits permettent de définir sa valeur (allant de 0 à 127). Nous avons désormais une trame de 2 octets pour les valeurs X et Y, par rapport à la tram de 10 octets de caractère précédemment mise en place par les IMA5. Cela est moins lourd, plus rapide d'envoi et plus rapide d'analyse côté Arduino. Ces données sont ensuite envoyées sur la liaison série grâce à la librairie SerialPort et correctement reçues du côté de l'Arduino.

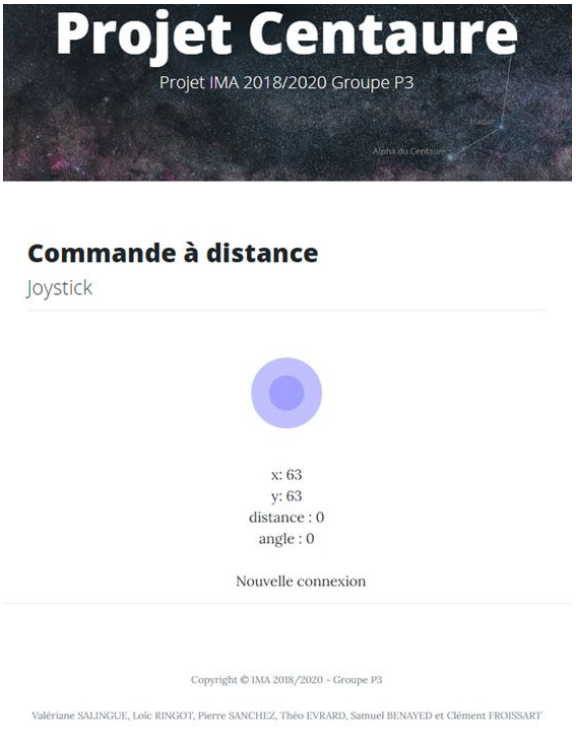


Figure 4 : Page Web du projet Centaure

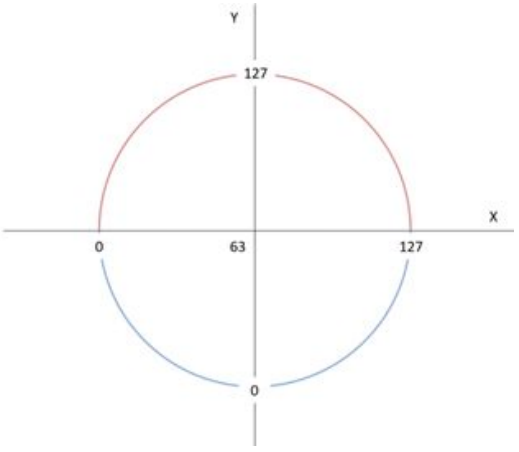


Figure 5 : Schéma de la consigne envoyée à l'Arduino

b) Utilisation de la Kinect

Dans le but de conférer de l'autonomie à notre robot, il nous faut être capable de récupérer et d'exploiter le flux vidéo provenant de la Kinect. L'avantage de la Kinect par rapport à une caméra classique est qu'elle dispose d'un capteur infrarouge. Ainsi, pour chaque pixel de la vidéo, on dispose des taux de rouge-vert-bleu (RGB) de celui-ci ainsi que de sa distance à la caméra. On utilise la libfreenect, qui permet à un système UNIX de contrôler et d'exploiter un capteur Kinect.
On conçoit donc un programme permettant de déterminer une consigne de direction en fonction de l'obstacle le plus proche. Pour y parvenir, on parcourt chaque pixel de la vidéo pour déterminer la position du pixel le plus proche. Selon les coordonnées y, x et z de ce pixel, on génère une consigne de déplacement. Cette consigne est linéaire, ainsi, plus un obstacle et centré par rapport au robot, plus le changement de direction sera important. De la même façon, plus un obstacle est proche, plus la vitesse du robot sera faible.

Cette consigne peut ensuite être transmise à l'Arduino via une communication série. On remarque cependant un problème majeur : le capteur infrarouge est assez sensible à la lumière naturelles. Cela crée de nombreuses zones d'ombre, notamment au niveau des fenêtres, dont on ne parvient pas à connaître la distance. Ils sont par défaut ignoré par le capteur, ce qui peut générer des collisions.

Un algorithme de récupération et de gestion des ombres est donc créé. On utilise les pixels aux alentours des zones d'ombre approximer et leur attribuer une distance par rapport au robot :

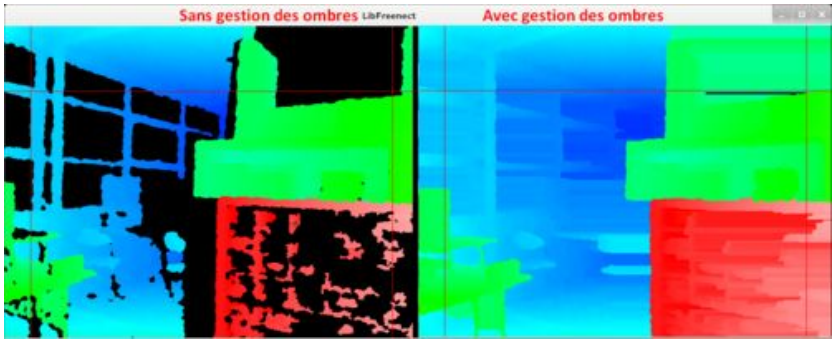


Figure 6 : Traitement d'image sans/avec correction des ombres

c) Suivi de balise

Pour que le robot puisse se repérer, nous avons imaginé l'utilisation de balises au sein de l'école.

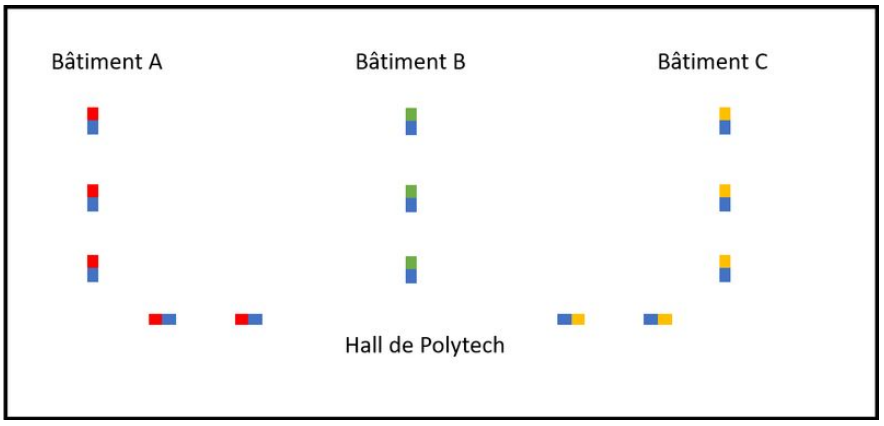


Figure 7 : Proposition de balisage au sein de l'école

Grâce à OpenCV nous avons implémenté une solution qui permet de détecter les couleurs via les images de la Kinect. Si l'utilisateur souhaite se diriger vers le bâtiment C, le robot devra suivre les balises jaunes par exemple. L'objectif est donc de repérer les couleurs à l'image et de déterminer dans quel sens le robot devra se déplacer. Pour récupérer ces valeurs nous calculons le barycentre des pixels d'une couleur définie. Nous comparons ensuite ces barycentres pour pouvoir prendre une décision et donner une consigne au robot. Chaque calcul de barycentre est lancé dans un thread différent et la prise de décision est faite dans le main.

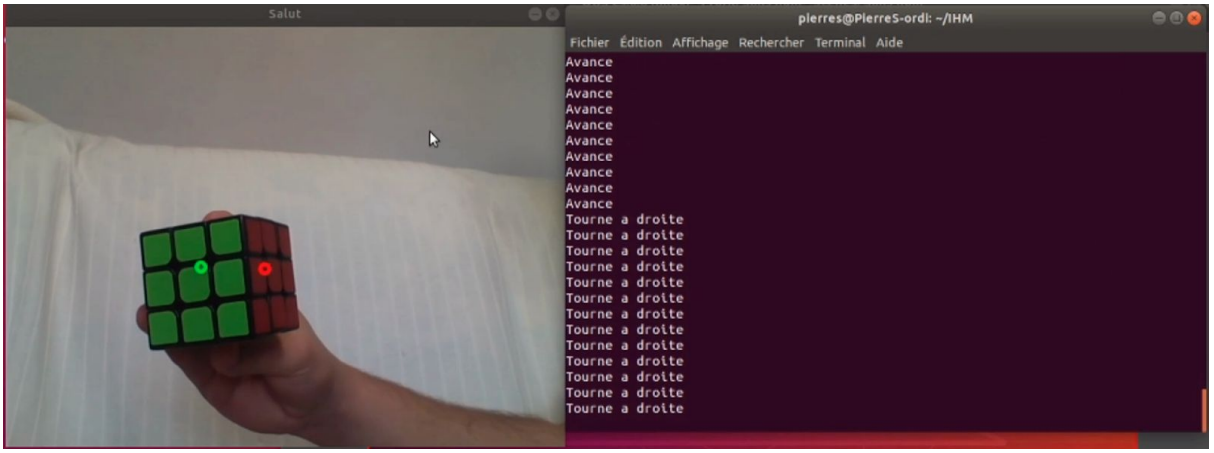


Figure 8 : Récupération des barycentres et envoie de l'ordre au robot

Pour éviter de détecter des formes qui ne correspondent pas aux balises on ne prend en compte que des formes ayant une surface minimale.

d) Apprentissage de circuit

Une autre solution proposée pour que le robot se déplace de manière autonome est de faire de l'apprentissage de circuit. Le but est que le robot s'entraîne sur un circuit donné, et l'apprenne pour ensuite se diriger de lui-même. Pour cela, nous utilisons Google Colab comme plateforme et TensorFlow comme logiciel. Grâce à cette plateforme, nous pouvons coder facilement sans avoir d'installations à faire. Pour apprendre un circuit à un robot, nous devons suivre 4 étapes :

- Prendre en photo le circuit
- Labelliser ces images
- Entraîner le réseau de neurones
- Vérifier le réseau avec d'autres images du circuit

Le circuit que nous prenons pour le test est le suivant : aller d'un bout du couloir à l'autre puis tourner à droite dans la dernière chambre.

Figure 9 : Circuit que doit apprendre le robot

Pour permettre au robot de se déplacer tout au long du circuit, nous devons établir un lien entre l'image et la direction que doit prendre le robot dans cette situation pour respecter le circuit. Par exemple, si le robot voit le mur de gauche, pour s'en éloigner il doit aller vers la droite plus ou moins fortement. Pour cela, nous choisissons de labelliser les images selon 5 directions :

- Aller vers la gauche
- Aller en diagonale vers la gauche
- Aller tout droit
- Aller en diagonale vers la droite
- Aller vers la droite

Chaque direction est reliée à des coordonnées X et Y compatibles à celles du joystick. Pour pouvoir utiliser ces images pour entraîner le réseau, il faut les convertir en matrices et leur associer le bon label. Nous pouvons maintenant entraîner le réseau. Nous choisissons le modèle présent dans le guide de TensorFlow qui utilise Keras, une bibliothèque de Python, pour l'entraînement et l'évaluation de réseaux. Une fois entraîné, nous pouvons l'évaluer à l'aide d'autres images du circuit. Nous labellisons aussi ces images pour nous permettre de vérifier facilement ses prédictions. Voici certaines prédictions que nous obtenons :

Figure 10 : Prédictions du modèle

Sous les images, nous pouvons lire la prédiction du réseau, le pourcentage de certitude de cette prédiction et enfin la bonne direction. Les prédictions en bleu sont bonnes et celles en rouge sont fausses. Pour diminuer le taux de mauvaises prédictions, nous pouvons essayer un autre modèle pour entraîner notre réseau ou entraîner ce réseau avec beaucoup plus d'images. Ici, le modèle a été entraîné avec 365 images.

Pour utiliser ce réseau de neurones sur le Centaure, nous devons récupérer les images de la Kinect. Une fois celles-ci converties et mises dans le bon répertoire, nous pourrons les évaluer pour en sortir une prédiction. Enfin, nous devons envoyer cette direction aux moteurs. Pour cela, nous communiquons avec la Kinect pour lui transmettre la direction. Cette dernière vérifiera qu'il n'y a pas d'obstacles pour suivre cette direction.

IV. Interaction Homme-Machine

a) Interface graphique

Une caractéristique importante d'un robot d'accueil est sa capacité d'interagir avec un utilisateur lambda. Pour y parvenir, celui-ci doit disposer d'une interface claire, intuitive mais également ludique afin d'attirer l'attention. Nous décidons donc de mettre à profit l'écran du Centaure afin d'accomplir cette tâche. On s'impose de plus d'utiliser le langage C afin de garder une cohérence globale dans le software de notre projet. L'interface visible à l'écran est donc gérée par un fichier source multi-threadé "display.c" qui fait appel à la libSDL (utilisée pour la création d'applications multimédia).
Notre interface se compose de 3 éléments distincts :

- Un fond uni gris
- Une bouche
- Des yeux

Ces éléments sont en réalité de simple images fixes au format bitmap. Plus précisément, ces images sont des "sprites", qui représentent un élément dans plusieurs positions différentes. Pour donner l'illusion d'un mouvement, on affiche différentes zones de l'image successivement (comme dans certains jeux vidéo) :

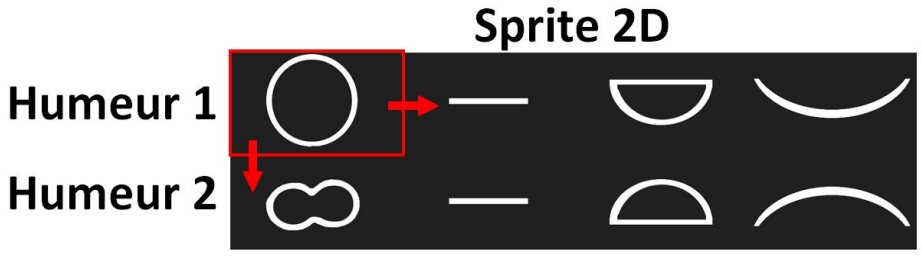


Figure 11 : Sprite d'animation de la bouche

Dans le but d'avoir des animations fluides et indépendantes, on gère l'animation de la bouche et des yeux dans deux threads différents. Un 3e thread nous permet, toujours grâce à la libSDL, de charger et jouer des fichiers sonores.
On synchronise alors l'animation de la bouche avec le lancement ou non d'un fichier sonore (la durée de l'animation est déterminer grâce à une formule liant la taille et le bitrate du mp3). En dehors de tout mp3, la bouche reste fixe.

Les yeux en revanche sont mobiles à tout moment, une position aléatoire est générée pour les iris. Afin qu'une personne se sente "concerné" lorsque le robot parle, une file de message IPC est créée entre Display.c et le programme d'analyse vidéo capable de détecter des visages. Ainsi, lorsqu'un visage est détecté, la position de celui-ci est transmise à l'interface et les yeux peuvent fixer celui-ci plutôt que de bouger aléatoirement.

b) Reconnaissance/Synthèse vocale

Lors de la première année du projet, nous avions mis en place une solution de synthèse et de reconnaissance vocale développée en Python. Après nos différents tests il s'est avéré notre PC embarqué sur le Centaure ne permettait pas de bien faire tourner le programme en Python. Nous avons donc recherché une solution plus légère qui pourrait tourner sur des systèmes légers.

Grâce à nos recherches nous avons trouvé un projet gitHub "Jarvis", projet créé par un français dont la dernière mise à jour date d'il y a 2 ans. Jarvis est un assistant vocal ultraléger et multilingue (notamment Français). Il a été imaginé pour la domotique et peut notamment tourner sur des systèmes très léger (ex: Raspberry Pi). Il dispose d'une reconnaissance et d'une synthèse vocale. Un autre point fort de ce projet est qu'il dispose d'une interface qui est utilisée pour mettre à jour le projet, installer les différents prérequis, ajouter de nouvelles commandes ou encore entraîner la reconnaissance de nouveaux mots. Ce projet est parfait pour notre application car il est tout d'abord très facile à utiliser et à prendre en main et est extrêmement personnalisable via l'ajout de commande à exécuter ou l'entraînement de nouveaux mots à reconnaître.

L'assistant vocal Jarvis fonctionne sur un principe de "mot magique". Tant que le mot magique (qui est dans notre cas "Centaure") n'a pas clairement été entendu, le programme ne peut effectuer des actions à la suite de commande vocale et reste dans l'attente du mot magique. Ces commandes sont situées dans le fichier jarvis-commands, ce fichier est très facile à modifier et peut notamment être consulté depuis l'application. Depuis l'application nous pouvons également entraîner la reconnaissance vocale des ordres.

Notre utilisation de l'assistant vocal Jarvis est très simple. Lors de la reconnaissance d'un ou d'une suite de mots correspondant à notre grammaire, ce dernier peut lancer diverses actions. Ces actions peuvent être un dialogue dans lequel le robot attend des réponses de l'utilisateur ou encore le lancement de fonctions et de procédures.

Figure 12 : Exemple utilisation Jarvis

Bilan

Après plus d'un an d'émotions et de rebondissement, c'est avec une certaine frustration que nous quittons ce cher Centaure. Notre frustration vient du fait d'être stoppé au moment où le robot était à nouveau fonctionnel et mobile.
Ce projet, de par la diversité des éléments qui le compose, nous a permis de mettre en œuvre de façon concrète différents pans d'apprentissage de la filière IMA :

- L'électronique de puissance avec la partie motorisation avec un aspect sécurité électrique plus poussé qu'avec un simple Arduino.
- De notions de système avec l'utilisation de thread et de liaison IPC entre différents programmes.
- Des notions de réseau avec la création d'un serveur Web.
- De la CAO pour le châssis du robot et les différents PCB.
- Des notions de deep learning.

Malgré son design tout droit sorti de l'an 2000, ce projet était donc très complet et devenait de plus en plus intéressant à mesure que nous avancions dans notre cursus.

On peut cependant déplorer une mise en route un peu lente du projet. Là où nous aurions dû récupérer un Hardware fonctionnel, nous avons finalement dû reprendre tout depuis le début (ce qui ne fut pas inintéressant loin de là !). Le lancement compliqué du projet est principalement dû à une méconnaissance de celui-ci et des différentes entités qui le composent. A cela s'ajoute la crainte de "casser" des éléments que nous comprenions mal en les démontant. Nous aurions dû au contraire dès le S6, à la vue du câblage, démonter et étudier chaque éléments du robot, pour le remonter de façon plus sûre et propre.
Dans le but de ne pas imposer ce problème à nos successeurs, nous avons mis à disposition sur le wiki un récapitulatif complet du hardware présent au sein du robot, de l'utilité de chaque composant, et comment les faire fonctionner.
Finalement, après bien des embûches, le robot est à nouveau capable de se mouvoir, le câblage interne semble fixé pour un moment. De plus le software permet de faire cohabiter différents mode de déplacement (manuel ou autonome) et d'avoir un début d'interaction avec des utilisateurs. (L'ensemble des éléments du software est disponible dans une archive GIT).

Bien que stoppé par le confinement, la majeur partie des objectifs fixés en début de semestre ont pu être accomplis voir poussés plus loin.

Annexe : Schéma de notre logiciel