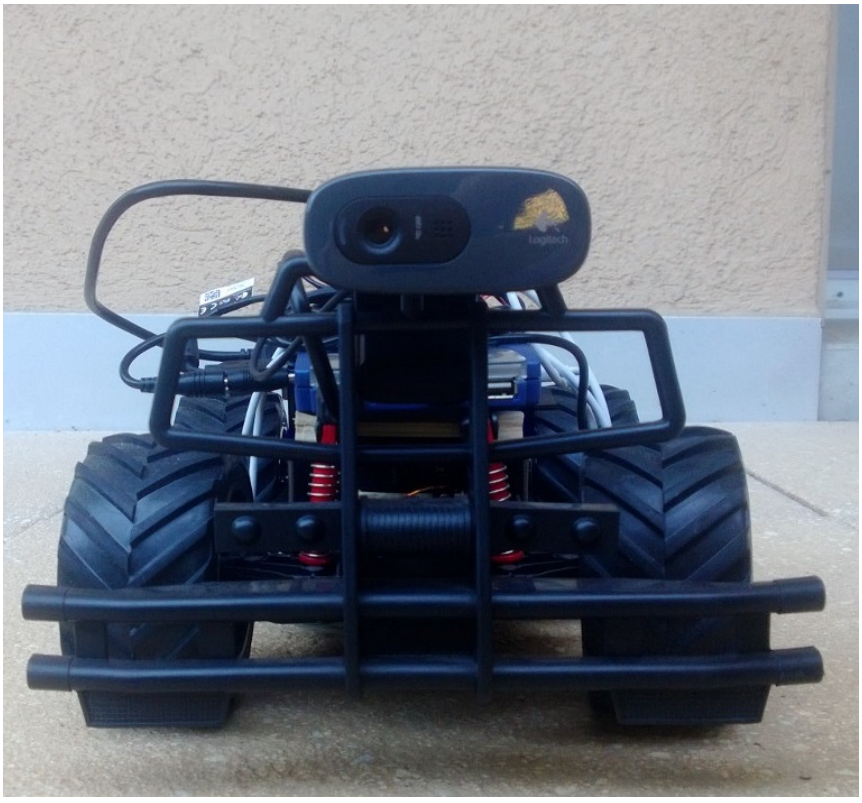
UGV

(Unmanned Ground Vehicle)

Sven Borden — Travail de maturité

Eric Brunner — Gymnase de Morges

2013



Avant-propos

Ce dossier est le résultat de onze mois de recherches effectuées dans le cadre du travail de maturité du gymnase de Morges. Ayant déjà quelques notions en informatique, nous nous sommes dirigés vers un domaine parallèle, la robotique. Le choix de ce sujet est dérivé des quadcopters (hélicoptères à quatre hélices). Bien que très attrayant, nous avons estimé que le travail de réalisation mécanique d'un drone volant risquait de demander trop de ressources économiques et temporelles. Nous avons préféré garder du temps pour l'aspect informatique.

Remerciements

Ce projet n'aurait pu aboutir sans l'aide de nombreuses personnes. Voici l'occasion de les remercier : Mr. Julien Dominski pour le suivi qu'il a fait pour nous ainsi que pour ses conseils en informatique qui nous ont été grandement utiles. Mr. Denis Rochat et Mr. Phillipe Rochat pour leur disponibilité, leurs renseignements ainsi que les prêts matériels. Mr. Jean Rossier, Karl Kangur, Michaël Bolay et Mr. Marco Pagnamenta pour leurs conseils techniques, surtout à propos de la plaque électronique à imprimer. Mr. Frederic Genevey ainsi que son site edurobot.ch pour avoir promu notre projet sur son site internet. Merci à Mr. Frédéric Chaberlot pour ses prêts matériels. Merci à Chloé Berthet pour ses corrections orthographiques. Nous tenions aussi à remercier Stefano Varricchio pour ses informations utiles.

Résumé

Les UGV sont des drones roulants qui sont principalement utilisés par les militaires ou la police. Ils permettent de remplir des missions qu'il serait difficile voire impossible à faire pour l'homme ou qui mettraient sa vie en péril. Ce type de drone existe également sur le marché des jouets, par exemple le *Beewi WiFi Camera Buggy BWZ200-A1* [14], qui coûte septante-neuf francs. Notre projet ne consiste pas à fabriquer un drone ayant un fusil permettant de "dégommer" tout ce qui bouge, mais de faire un drone de reconnaissance à moindre coûts. Les drones existants sont soit trop cher, soit peu performants. L'objectif de ce projet est de développer un prototype qui aurait des avantages face à un produit tel que le Beewi[14] (une portée améliorée, une meilleure autonomie et une meilleure qualité vidéo) . En effet, si nous parvenons à avoir un ensemble possédant un véhicule tout terrain, une interface graphique claire et un guidage simple et efficace, à un prix abordable, nous aurions des arguments de poids face aux autres produits disponibles sur le marché.

Dans la lancée de ce projet, nous nous sommes inscrits au concours Suisse de science *La science appelle les jeunes* [1]. Ce concours réunit presque tous les domaines, pourvu que certains critères soient respectés. Tous les concurrents continuent un projet qu'ils avaient commencé durant leur travail de maturité/diplôme et la démarche utilisée doit être scientifique. Ce concours pourrait, selon les résultats, nous aider à promouvoir notre drone.

Au terme de ce projet nous avons acquis des connaissances dans plusieurs domaines de l'informatique, de l'électronique et de la mécanique. Le résultat final est un véhicule contrôlable à distance, avec une caméra et des senseurs, en plus d'un programme client-serveur Java et C++ pour le transfert du

1. http ://fr.sjf.ch/

flux vidéo, des commandes de direction et des informations recueillies par les senseurs.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le but de ce projet est de construire un véhicule roulant que l'on peut commander à distance. Plus qu'une simple voiture télécommandée, ce drone est capable d'être contrôlé sans avoir une vue directe sur celui-ci, car il possède des capteurs tel qu'une caméra et des capteurs de distances. Ce type d'engin se nomme *UGV (Unmanned Ground Vehicle)* soit "véhicule roulant sans équipage". Surtout utilisés dans l'armée, les modèles qu'on peut trouver sur le marché sont très coûteux, ils varient entre trois cents et mille trois cents francs. Notre but est donc de pouvoir construire un appareil semblable pour moins de deux cent francs. Ce drone peut servir en cas de catastrophes naturelles, néanmoins, notre modèle ne peut pas rouler sur les terrains trop accidentés, mais pour corriger ce problème, il suffirait tout simplement d'implémenter notre système électronique dans un autre type de véhicule. Le développement d'une interface graphique simplifiant l'utilisation est également un but du projet. En effet, cela ne sert pas à grand chose d'avoir un drone que personne ne sait utiliser à moins de s'être entraîné durant des mois.

Nous avons décidé d'initier le rapport avec le chapitre concernant le hardware car il permet une meilleure compréhension du reste et introduit quelques idées et éléments clefs du projet. Ce chapitre décrit principalement l'ordinateur et le micro-contrôleur [10] embarqués sur le drone. Ensuite, vient le chapitre mécanique et électronique qui détaille les modifications faites à la voiture radio-commandée et les recherches faites dans ce domaine. Les deux chapitres qui suivent sont consacrés aux différentes versions des logiciels que nous avons créés et utilisés. Finalement, un dernier chapitre de conclusion clôt ce rapport et ouvre de nouvelles perspectives de développement.

Chapitre 1

Hardware

POUR réaliser ce projet, nous avons dû faire des choix au niveau du hardware. Le hardware est tout ce qui touche à la partie physique du drone. Néanmoins, dans ce chapitre, nous ne traiterons pas de la mécanique et de l'électronique pure, ces sujets seront abordés dans le chapitre 2.

Le drone est basé sur deux éléments centraux. Le premier, l'Arduino, est un micro-contrôleur qui permet de lire et d'émettre des signaux électriques. Le second est le Raspberry Pi, qui est un petit ordinateur bon marché (trente-cinq francs), récemment sorti sur le marché.

La figure 1.1 est une première vue d'ensemble des différents composants utilisés dans ce projet. Comme dit précédemment, tout ce qui concerne moteur, servo et senseurs sera spécifié dans le chapitre suivant.

1.1 Arduino

L'Arduino [9] est un micro-contrôleur *Open Source*, ce qui veut dire que tout le monde peut non seulement avoir accès aux plans et aux codes, mais peut aussi les modifier.[4] Ce micro-contrôleur se programme avec un langage proche du C.

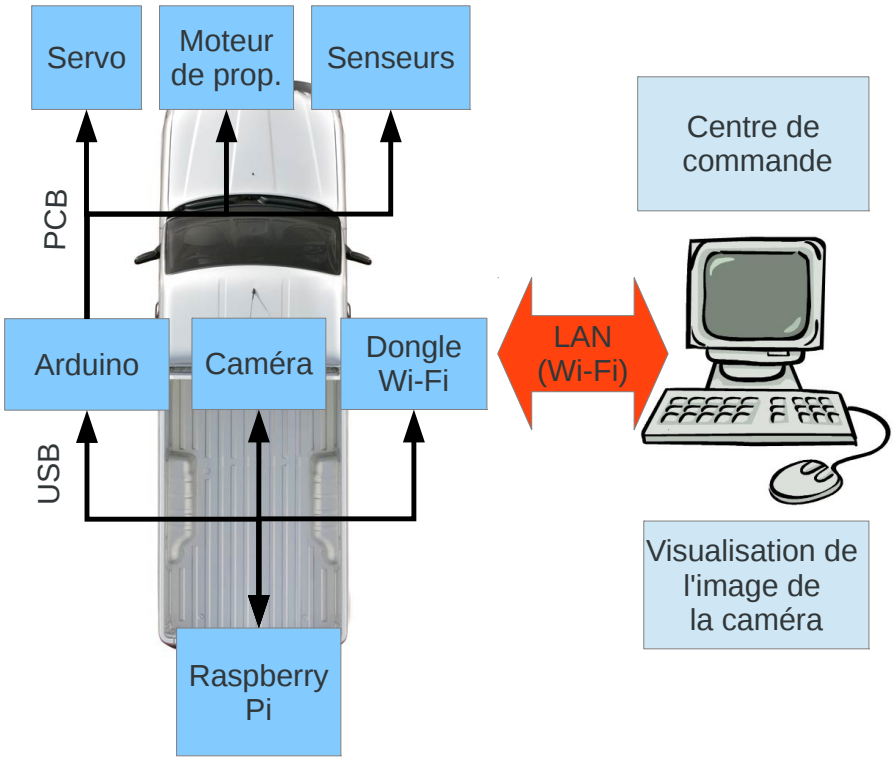


FIGURE 1.1

1.1.1 Choix du type d'Arduino

Pour ce projet, nous avons choisi l'Arduino Uno, c'est l'Arduino de base. Nous avons hésité à prendre l'Arduino Mega, mais les avantages qu'il offre ne sont pas très utiles pour notre projet. Bien qu'il ait une puissance de calcul supérieure à celle de l'Arduino Uno, il est plus coûteux et prend plus de place pour des avantages dont nous n'avons pas besoin. Dans le cadre de ce projet le coût et la place sont des facteurs déterminants. Puisque nous n'avons pas besoin d'une grande puissance de calcul, nous avons choisi l'Arduino Uno (fig : 1.2).

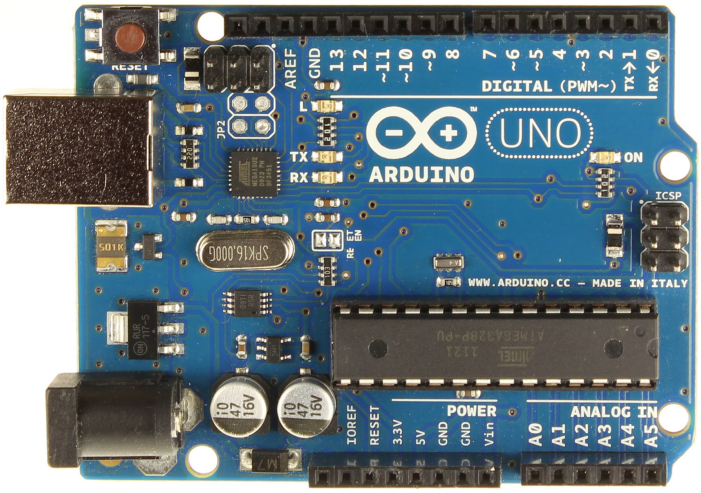


FIGURE 1.2 – Arduino Uno R3, on peut remarquer les pins qui se trouvent de part et d'autre de l'Arduino. Au centre à droite, le microcontrôleur ATmega328, ainsi que le port USB tout en haut à gauche de l'image. Ceci est l'image officielle du produit disponible sur : *http ://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg*

1.1.2 Détails techniques de l'Arduino

Voici une liste des caractéristiques techniques de l'Arduino Uno R3 [9] :

1. 14 pin [1] digitaux (signal haut ou bas) qu'on peut définir en INPUT ou en OUPUT dont 6 d'entres eux peuvent moduler le signal lorsqu'ils sont utilisé en OUTPUT
2. 6 pin analogiques en INPUT
3. Connexion USB
4. Oscillateur à quartz cadencé à 16MHz
5. Courant continu sur les pin digitaux (40mA)

1. Les pin sont des trous où on peut y glisser des fils métalliques. Ce sont les liaisons entre le contrôleur et les senseurs

6. Une sortie 3.3V et une sortie 5V en courant continu
7. 32KB de mémoire flash
8. 2KB de RAM
9. Micro-contrôleur ATmega328

1.1.3 Avantages et alternatives

Nous n'allons pas faire une analyse approfondie des micro-contrôleurs mais il faut se rappeler qu'ils ne doivent pas faire fonctionner un système d'exploitation, mais qu'ils doivent simplement traiter des mesures et envoyer des signaux digitaux ou analogiques. Sa programmation en pseudo C le rend rapide. L'avantage de l'Arduino Uno est qu'il n'y a pas besoin de faire de soudure, les pin sont directement accessibles et on peut y glisser un fil métallique. Dans le cas où nous souhaiterions commercialiser notre produit, nous choisirons probablement l'Arduino mini car il est encore plus petit et nous pouvons directement souder les fils sur le micro-contrôleur.

1.2 Raspberry Pi

Le Raspberry Pi [11], ou la Framboise pour les francophones, est un ordinateur de la taille d'une carte de crédit sur lequel on peut installer différents systèmes d'exploitations dérivés de UNIX/Linux. Le Raspberry Pi est acheté nu, c'est-à-dire que cet ordinateur ne possède pas d'écran, ni de clavier ou de souris, néanmoins le Raspberry Pi possède plusieurs ports où on peut brancher un écran (via l'interface HDMI ou Composite), un câble Ethernet et presque ce qu'on veut grâce aux deux ports USB. Le Raspberry Pi est très intéressant non pas du point de vue de sa puissance de calcul, mais du point de vue rapport qualité-prix (35.- CHF).

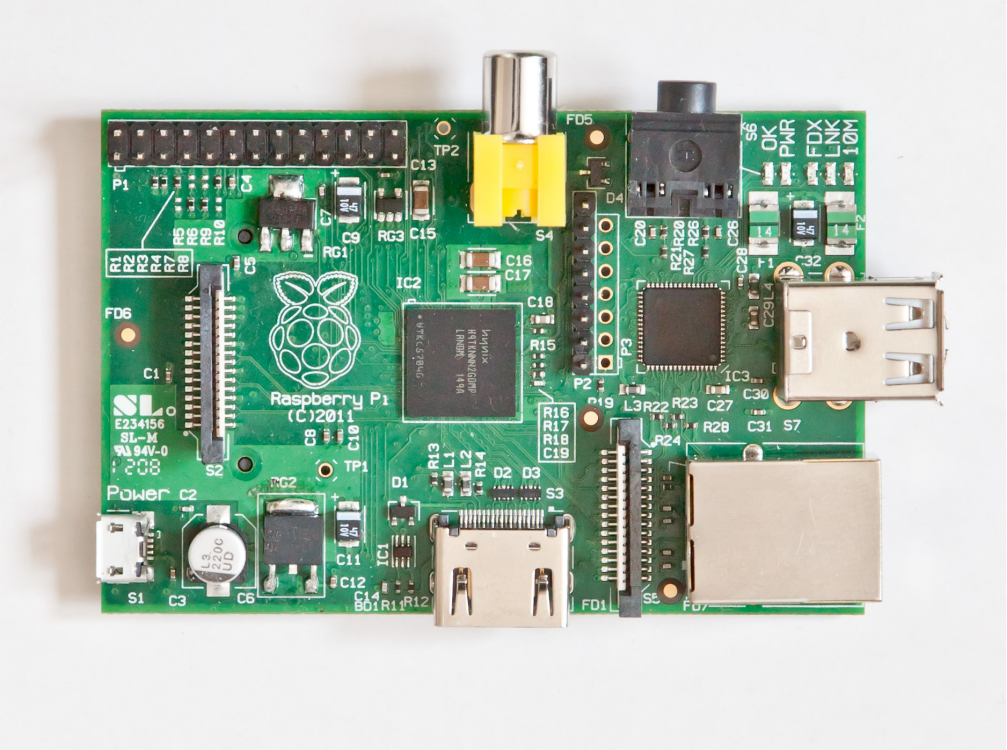


FIGURE 1.3 – Le Raspberry Pi est un petit ordinateur linux. Il est de la taille d'une carte de crédit.

1.2.1 Détails techniques du Raspberry Pi

Voici une liste des caractéristiques techniques du Raspberry Pi modèle B [11] :

1. 45g environ
2. Processeur ARM1176JZF-S (ARMv6) 700MHz Broadcom 2835
3. 512Mo de RAM (sur la version B, soit celle que nous avons choisie)
4. 2 sorties vidéo (HDMI et Composite)
5. Sortie audio stéréo Jack (3.5mm) (le son passe aussi par le HDMI en sortie 5.1)
6. Écriture et lecture possible sur une carte mémoire sous forme de carte SD (supporte les formats : SDHC, MMC et SDIO)

7. 2 ports USB 2.0 et 1 port Ethernet
8. Alimentation par câble micro USB
9. Faible consommation (5W, 5V, 1A)
10. Communication possible via les pin GPIO
11. Décodeur permettant de lire le Full HD 1080p
12. API logiciel vidéo (OpenGL)

1.2.2 Avantages et utilisations

Bien qu'à première vue la Framboise ne semble pas très performante, il faut prendre en compte son prix qui est bas, sa taille ainsi que les possibilités qui sont presque infinies. Les projets qu'on peut mener grâce au Raspberry Pi sont des plus variés. En effet, il peut être utilisé pour la photographie, comme base centrale d'un système de surveillance ou comme ordinateur central pour un drone par exemple.

1.2.3 Choix de l'OS

Une quinzaine de systèmes d'exploitations fonctionnant sur le Raspberry Pi existent. Parmi les plus connus, il y a Androïd, Arch Linux ARM et Debian/Raspbian. Notre choix à été porté sur Raspbian, qui est un dérivé de Debian, pour plusieurs raisons. Tout d'abord, cet OS a été développé spécialement pour le Raspberry Pi, c'est donc un OS actif et vivant continuellement développé par la communauté Raspberry. Cet OS est basé sur un environnement Linux, ce qui offre un grand nombre de libertés afin de travailler dessus. Egalement, Raspbian est gratuit, ce qui est à prendre en compte puisque nous essayons de réduire les coûts.

1.3 Sans fil

Au niveau hardware, afin que le Raspberry Pi puisse se connecter sur le réseau sans fil, nous utilisons un *dongle* (fig :??) Wi-Fi [2] connecté par USB. Cet accessoire ne nécessitant pas de *driver* [3], il est directement reconnu et peut rapidement être configuré par le Raspberry Pi.

1.4 Webcam

La webcam utilisée dans notre projet est tout à fait standard. D'ailleurs, la grande majorité des caméras USB récentes pourraient fonctionner. Il faut juste s'assurer qu'elles utilisent le driver UVC (USB Vidéo Class)[1]. Le modèle est la C270 de Logitech qui a une résolution de 720p et qui coûte 35.- CHF.

2. Le modèle utilisé est le suivant : EW7811Un fabriqué par Edimax

3. Un driver est un petit logiciel permettant de faire l'interface entre les périphériques et l'ordinateur.

Chapitre 2

La mécanique et l'électronique

NOUS nous basons sur un modèle réduit télécommandé de type 4x4. Le tableau (2.1) est un inventaire de ses caractéristiques dans son état avant d'implémenter notre système. Dans ce chapitre nous allons aborder tout ce qui se réfère aux actuateurs (moteur et servo-moteur), comment ils ont été installés et comment les utiliser. Nous commencerons par le plus gros travail, la conception d'un circuit électrique imprimé, dit aussi PCB.

E

2.1 PCB

La décision de retirer l'électronique de base du véhicule télécommandé provient d'une curiosité de comprendre comment elle est faite et surtout, du besoin d'avoir un degré supérieur de contrôle sur la voiture. A la base, le cerveau de la voiture se trouvait dans un petit chip. Très courant dans les jouets radio-commandés bas de gamme, le RX2 ou similaire (fig. 2.2), permet d'interpréter un signal radio et émettre des signaux pour actionner des moteurs, par exemple. Ce micro-chip va de paire avec le TX2, qui se trouverait dans la télécommande (TX pour "Transmit" et RX pour "Receive").

La nature simple du chip RX2 ne lui permet que d'émettre un signal

Grandeurs	Longueur	35 cm
	Longueur (centre de roue à centre de roue)	17 cm
	Largeur	22 cm
	Largeur (centre de roue à centre de roue)	17 cm
	Hauteur au sol	4 cm
Moteur de propulsion	Voltage de marche	˜5V - ˜10V
	Courant min (roue libre)	˜2A
	Courant max (roue bloquée)	˜3A
Servo-moteur de guidage	Fabriquant	Corona
	Modèle	Metal gear DS558HV
	Voltage de marche	˜6V - ˜7.4V
	Courant	300mA - 400mA
	Charge maximale	12kg - 14kg

TABLE 2.1 – Caractéristiques du véhicule

bas ou haut (0 ou 5V), à la différence de l'Arduino qui a la possibilité de moduler le signal (voir : 2.2.3). Cette caractéristique spécifique nous permet d'avoir un large éventail de vitesses du moteur et des angles de braquage différents.

2.1.1 Plaque personnelle

En attendant de recevoir la plaque sortie d'usine et pour tester une partie design, nous avons conçu un prototype du résultat final (fig.2.3). Cette version préliminaire nous a permis de faire les premiers pas avec

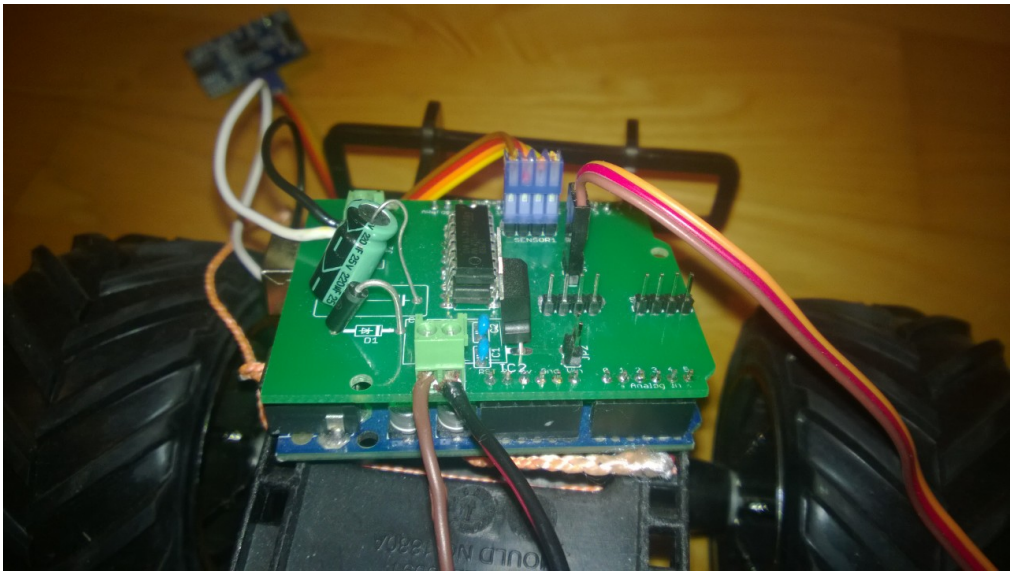


FIGURE 2.1 – Le PCB que nous avons nous avons conçu grâce au logiciel Eagle.

le véhicule. Elle permet de contrôler le moteur de propulsion ainsi que le servo-moteur directionnel. Pour une vue plus claire du prototype et une description des composants principaux, voir la figure 2.4.

Quelques explications sur le contenu et la conception de la plaque prototype :

1. Un régulateur de voltage 5V (7805T) transforme le courant 9V de la batterie en 5V.
2. Le régulateur est entouré de capacités pour lisser d'éventuels sauts de tension, mais remarquez qu'ils seraient probablement inutiles lors de hauts sauts de voltage.
3. Un chip L293D (H-bridge (sec : 2.3.2)) contrôle le moteur de propulsion. Ici, deux chips sont empilés, donc en parallèles, pour pouvoir supporter la charge importante du moteur.

Tous ces composants, en plus des câbles et des connecteurs vers les actuateurs et la batterie, sont soudés sur une strip-board. C'est essentiellement

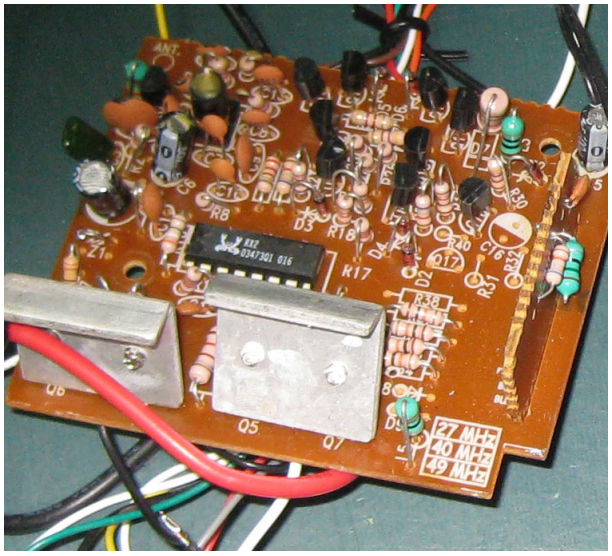


FIGURE 2.2 – Exemple de plaque similaire à celle qu'on a pu trouver dans le véhicule.

un alignement des bandes conductrices avec des trous pour placer les composants. Les trous ont une distance standard de 0.5 pouces. Le détail des connexions ne sera pas précisé ici, mais il ressemble évidemment au design de la plaque qui est illustré plus tard.

2.1.2 Schema électronique

Le schéma a été dessiné à l'aide d'un programme dédié gratuit (version light), Eagle. Les symboles utilisés sont donc standardisés. (voir fig : 2.6)

Quelques indications pour lire le schéma :

- Les fils portant le même nom sont connectés
- Toutes les terres sont connectées ensemble
- Il y a un circuit 9V et un circuit 5V

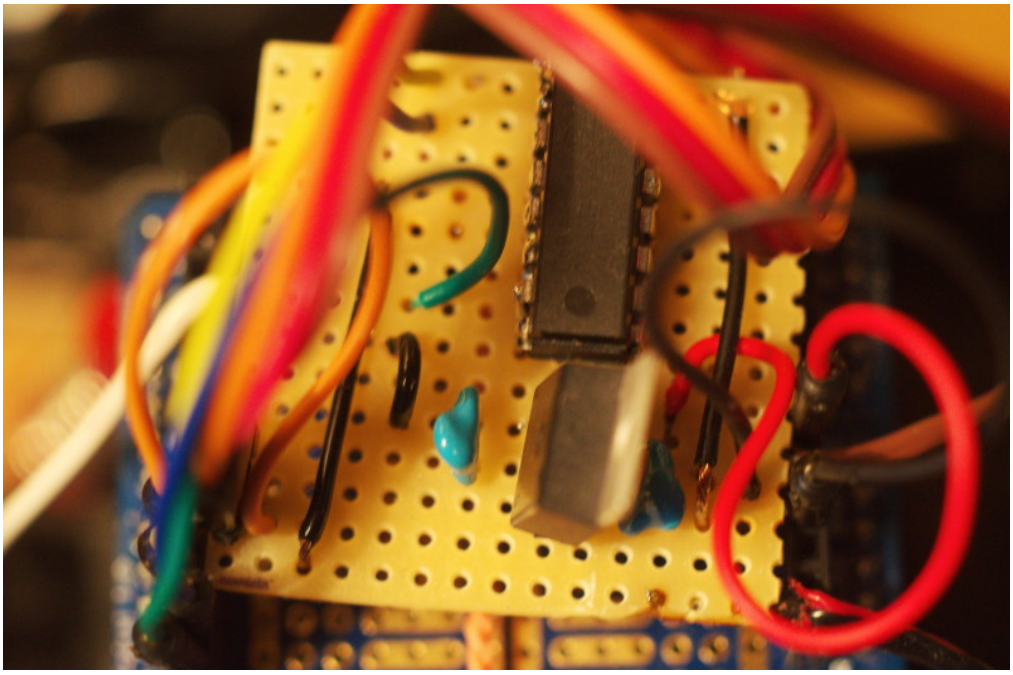


FIGURE 2.3 – Première plaque rassemblant différents composants électriques.

2.1.3 Design

Une fois le schéma dessiné, Eagle crée une plaque sur laquelle il faut disposer chaque composant. Le programme se charge de contrôler que tout est relié comme dans le schéma. Il pourrait même faire le ”routing” lui-même, mais le résultat est très décevant. Des étudiants de l'EPFL nous ont conseillé de faire ça nous-mêmes. Ceci demande de la patience et de la réflexion, comme un puzzle. Nous voulions essayer d'imprimer une version à l'EPFL, il fallait donc faire un design dont les routes se trouvaient uniquement sur le derrière de la plaque (routes bleues)... Le jeu ce complique ! En effet, la meilleure solution que nous avons trouvée nous laisse avec une connexion à faire à la main et un connecteur pour senseur en moins.

D'ailleurs, il faut remarquer que la plaque maison ne permet pas au robot d'interagir avec son environnement. En effet, elle peut juste contrôler

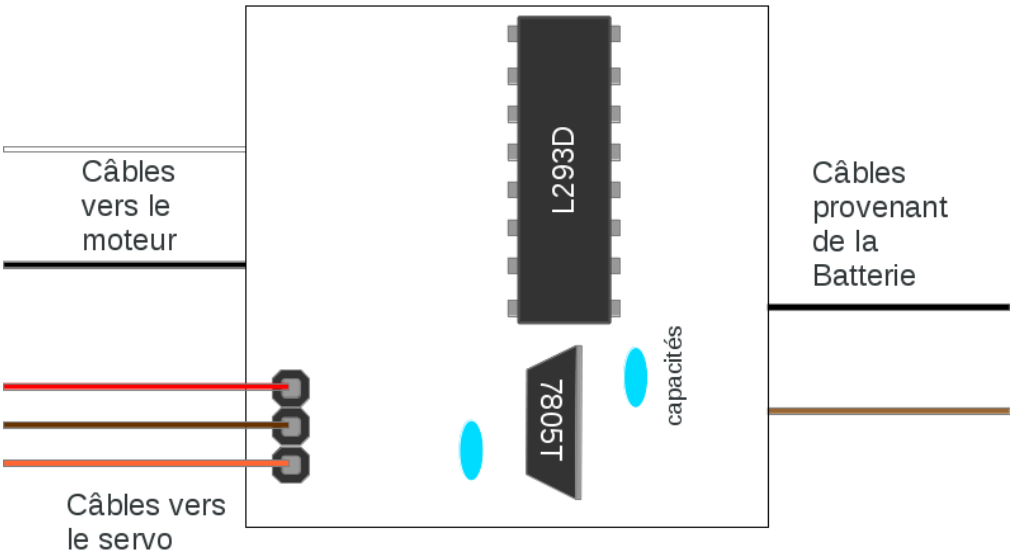


FIGURE 2.4 – Schéma sommaire du prototype.

des moteurs. La plaque usinée prévoit trois connecteurs pour des senseurs de distance.

2.1.4 Composants

Plus que de comprendre comment les composants sont connectés ensemble il est important de bien discerner à quoi sert chacun d'entre eux. Ici, nous détaillerons, un à un, chaque composant, ses propriétés et sa fonction.

1. L293D

   Le L293D est à un chip à seize pattes qui encapsule effectivement deux H-bridge comme illustré dans la section 2.3.2. Chaque moitié du pont peut fournir 0.6A de courant continu et 1.2A en pic [5]. Sachant que le moteur peut tirer jusqu'à 3A lorsque la roue est bloquée, il est clair qu'un seul de ces demi-chip est loin d'être à la hauteur. Les avis

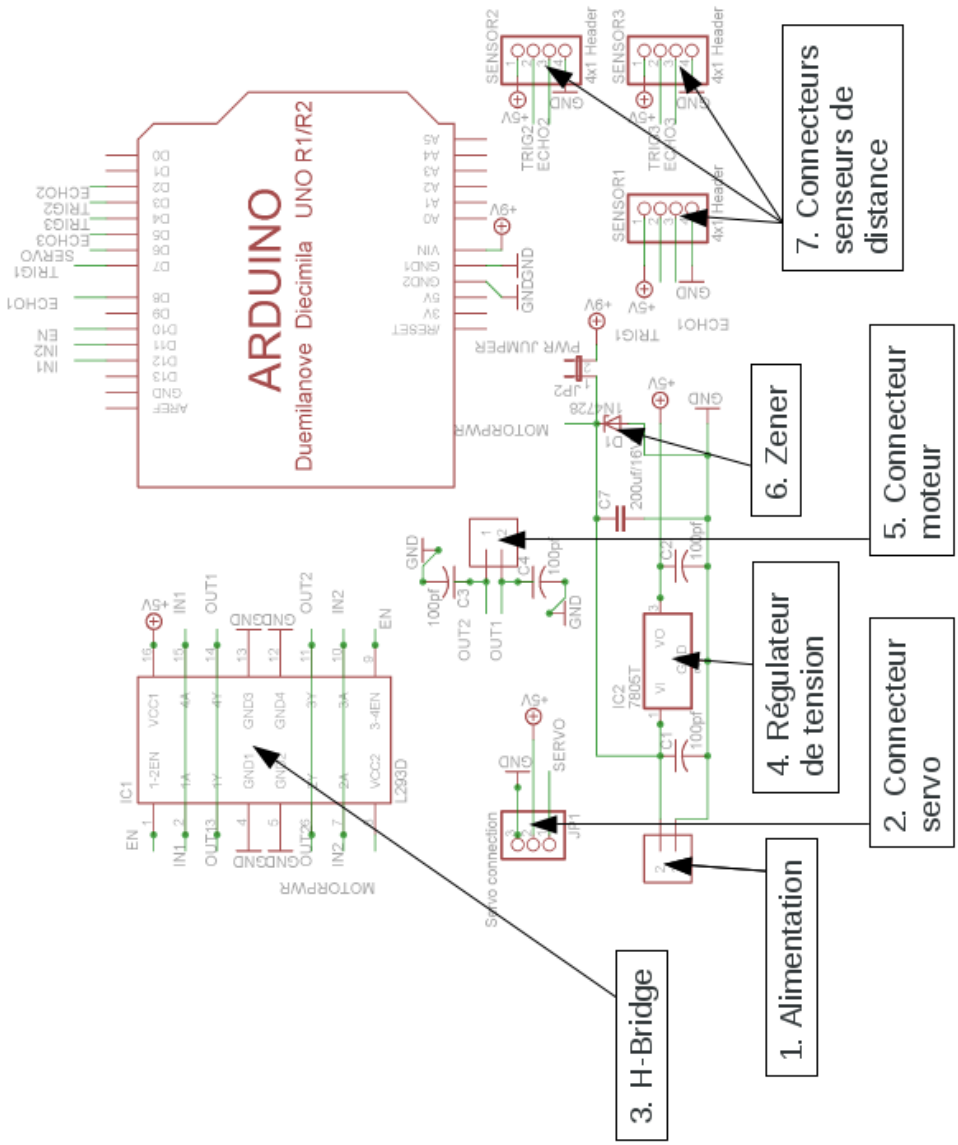


FIGURE 2.5 – Schéma complet du PCB

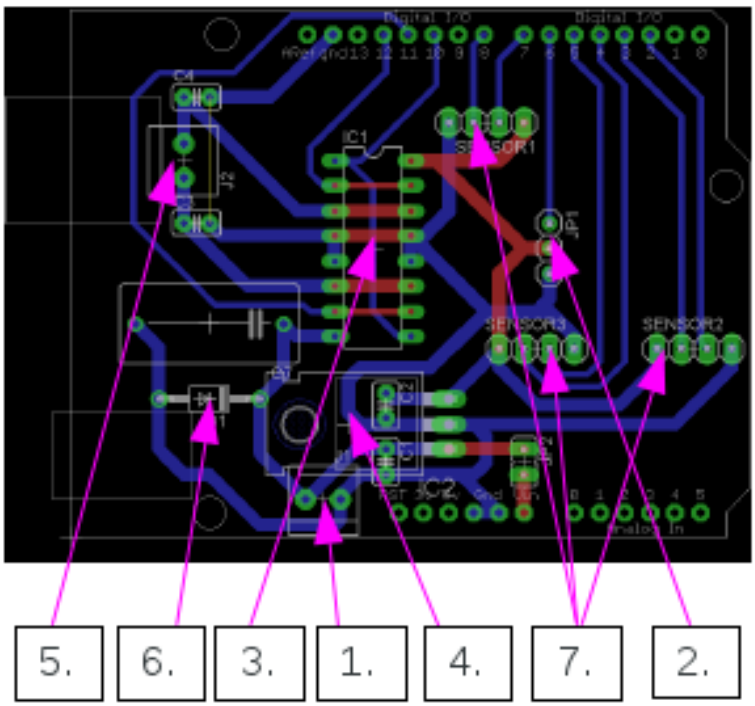


FIGURE 2.6 – Image du design final à l'ordinateur. (Voir figure 2.6 pour les annotations)

sont partagés s'il est possible de mettre ces chips en parallèle (certains transistors et autres composants ne se partagent pas la charge comme prévu lorsqu'ils sont mis en parallèle), mais, basé sur les indications d'un grand producteur d'électronique pour amateurs [6], nous avons décidé d'empiler deux L293D et de mettre leurs moitiés en parallèle, ce qui s'additionne à 2.4A de courant continu et 4.8A en pic. Nous avons pu tester et valider cette configuration avec la plaque prototype.

2. 7805T

Le 7805T est un régulateur 5V. Il approvisionne les composants fonctionnant à 5V en courant continu. Il peut prendre jusqu'à 35V de tension en entrée [3]. Il faut savoir que c'est un appareil très peu efficace car la tension non utilisée est dissipée en chaleur. Il est donc nécessaire de s'assurer que le composant arrive à dissiper suffisamment de courant en chaleur sans griller. Selon les graphes, il a une dissipation d'environ 2.5W. Si on somme les courants des différents composants fonctionnant à 5V on trouve qu'ils requièrent environ 1A au maximum. La plus grande partie de la consommation vient du servo-moteur qui ne demande pas un approvisionnent continu. On calcule donc une dissipation de 5W (5V*1A) ce qui excède largement les spécification du composant [3]. Mais, les tests jusqu'à ce jour montrent qu'il n'y a pas de problème vu que le servo-moteur n'a pas besoin d'un approvisionnement régulier. Pour être sûr, on pourrait fixer un *heatsink* [1] sur le composant.

3. Condensateur $200\mu f$ ou $500\mu f$

   Cette capacité travaille en parallèle avec la diode Zener (fig : 2.8) pour lisser le signal vers le moteur et absorber de forts sauts de tension à la suite, par exemple, d'un blocage soudain des roues.

4. Condensateur $100pf$

   Ces petites capacités servent surtout à lisser les petites variations dans le signal, comme par exemple en sortie du régulateur de voltage.

5. Diode Zener

   La diode Zener a la particularité, en plus d'être une diode, d'avoir une tension seuil au-delà de laquelle elle ne remplit plus sa fonction de diode et laisse librement passer le courant. Ceci est très pratique en

1. un *heatsink* est une pièce métallique qui permet de dissiper plus rapidement la chaleur d'un composant.

cas de forts sauts de tension (fig : 2.8).

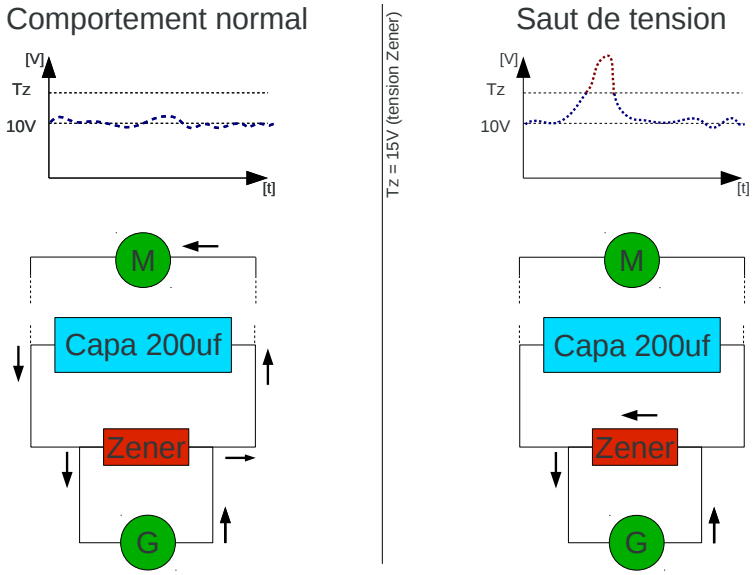


FIGURE 2.7 – Schéma d'un circuit ayant une diode Zener, expliquant comment cette diode permet d'éviter de griller des composants en cas de sauts de tensions.

6. Connecteurs senseurs

   Ces connecteurs facilitent simplement la connectique et le remplacement d'un senseur au cas où il était cassé. Il y a quatre pins : deux pour l'alimentation (5v, 15mA, pour un senseur à ultrason HC-SR04) [2], un pour envoyer le signal et le dernier pour recevoir l'écho.

7. Connecteur servo-moteur

   Tout comme les connecteurs pour senseurs à la différence qu'il n'y a que trois pins : deux pour l'alimentation (5V, 600mA) (tab : 2.1)

8. Prises block

   Ces connecteurs peuvent connecter de façon provisoire des plus gros

câbles (moteur, batterie). Les câbles sont immobilisés au moyen de vis.

2.1.5 Commande et conception

Une fois le design terminé et les composants choisis, nous pouvons envoyer notre projet à une fabrique spécialisée. Des étudiants de l'EPFL, du club RoboPoly, nous ont conseillé une compagnie basée en chine, SeeStudio, qui imprime des plaques à de bons prix. La procédure de perçage et d'impression est évidemment standardisée. Le fabriquant requiert donc des fichiers bien précis à donner à ses machines, dénommés fichiers Gerber. Ces fichiers contiennent de l'information en coordonnées x,y et en commandes que la machine interprète. Il serait évidemment très complexe d'écrire un tel fichier à la main. Heureusement, le CAD que nous utilisons prévoit des scripts capable de transformer notre design en de tels fichiers. Le fabriquant requiert exactement huit fichiers différents, compressés dans un zip et envoyés avec la commande.

2.2 Système de guidage

2.2.1 Système directionel initial

Dans son état initial, le système de guidage pouvait se comparer à un servo-moteur très rudimentaire. Il en comptait toutes les caractéristiques, mais en un état simplifié, en particulier le système de positionnement. Dans un servo-moteur classique, il s'agit d'un potentiomètre (résistance variable) qui permet de savoir en tout moment la position de la corne [2]. Par contre, dans le cas de la voiture, un système de balais assure cette tâche. Il en résulte une identification de position très basique : gauche,devant ou droite.

Sachant que nous avons retiré l'électronique de la voiture, il nous restait deux possibilités : utiliser le système de guidage rudimentaire, mais déjà en

2. La corne est le bras qui sort du servo-moteur et qui permet de déplacer des masses

place ou tout remplacer avec un servo-moteur conventionnel.
Après beaucoup de temps perdu à tenter de contrôler le guidage de base avec notre électronique importée, nous avons décidé de passer à un servo-moteur. Nous avons acheté un puissant servo-moteur de haute qualité chez une connaissance qui en avait commandé un gros lot pour la modique somme de 10.- CHF.

2.2.2 Remplacement du Système de guidage

Une installation fiable d'un objet étranger dans un ensemble usiné tel que la voiture n'est pas une tâche facile. Il fallait pourtant que le résultat final soit solide si l'on voulait pouvoir compter dessus. C'est pour cela que nous avons créé une base en contreplaqué pour y loger le servo-moteur.

Ce montage permet de retirer le servo-moteur en cas de besoin, donc de pouvoir le remplacer. En effet, la plaque supérieure est fixée au moyen de vis à bois. Le servo-moteur est accompagné, dans son logement, d'un morceau de gomme adhérente (morceau de chambre à air). La structure épouse les formes de la voiture pour un maximum de rigidité. La transmission de la force aux roues se fait par l'intermédiaire d'une tige métallique. Celle-ci est fixée à la corne du servo-moteur et possède une boucle soudée à l'ancien axe de transmission. Nous utilisons justement l'ancien axe de transmission pour une raison de géométrie. [3]

Le montage final est illustré par les figures (2.9) et (2.10).

2.2.3 Contrôle du servo

Le contrôle d'un servo-moteur est une tâche qu'un micro-contrôleur tel que l'Arduino (section : 1.1) effectue avec aisance. On peut indiquer à un servo-moteur de se rendre vers un de ses $180°$ de liberté en lui envoyant

3. voir : http ://en.wikipedia.org/wiki/Ackermann_steering_geometry sur "Ackerman steering"

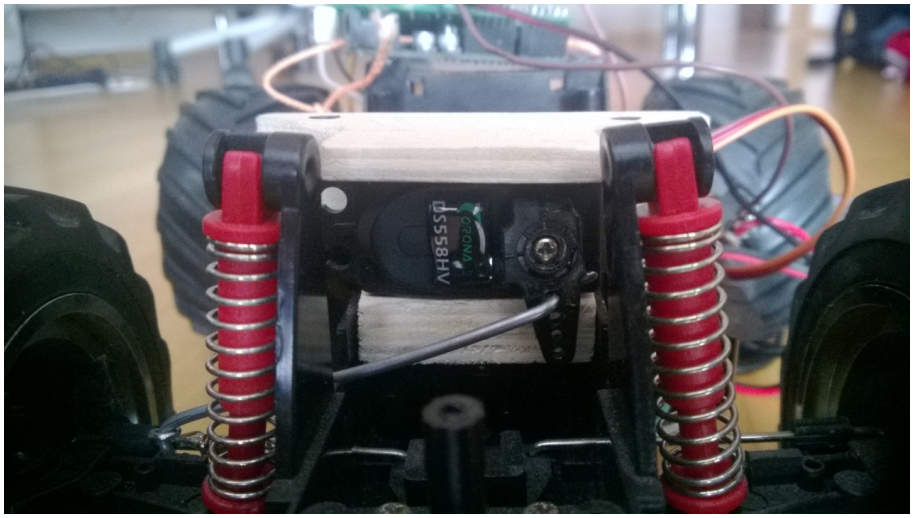


FIGURE 2.8 – Vue frontale du montage pour le système de direction.

un signal électrique dit modulé. Ce signal est modulé d'une manière compréhensible pour le servo-moteur. L'illustration 2.11 pourrait davantage éclairer le lecteur.

Comme l'on peut voir, le dit signal est formé de hauts et de bas. Lorsqu'il est *BAS*, cela veut dire que la tension est basse ou égale à 0V. Lorsqu'il est *HAUT*, cela veut dire que la tension est à une valeur définie auparavant, standard, différente de 0V. Dans notre cas, le *HAUT* est à 5V (standard pour le modélisme et l'électronique en général). On appelle la période pendant laquelle le signal est *HAUT* une pulsation.

Chaque début de pulsation est séparé par un temps bien défini de 20ms. Ce qui peut varier d'une pulsation à l'autre, donc ce qui informe le servo-moteur en quel angle il doit se positionner, est la longueur de la pulsation. Comme indiqué, celle-ci peut varier de 1ms à 2ms.

2.2.4 Programmation pour contrôler un servo

Le programme se trouvant en annexe (A.2) est un exemple proposé dans la section "apprentissage" du site officiel d'Arduino [8]. Il utilise la librairie

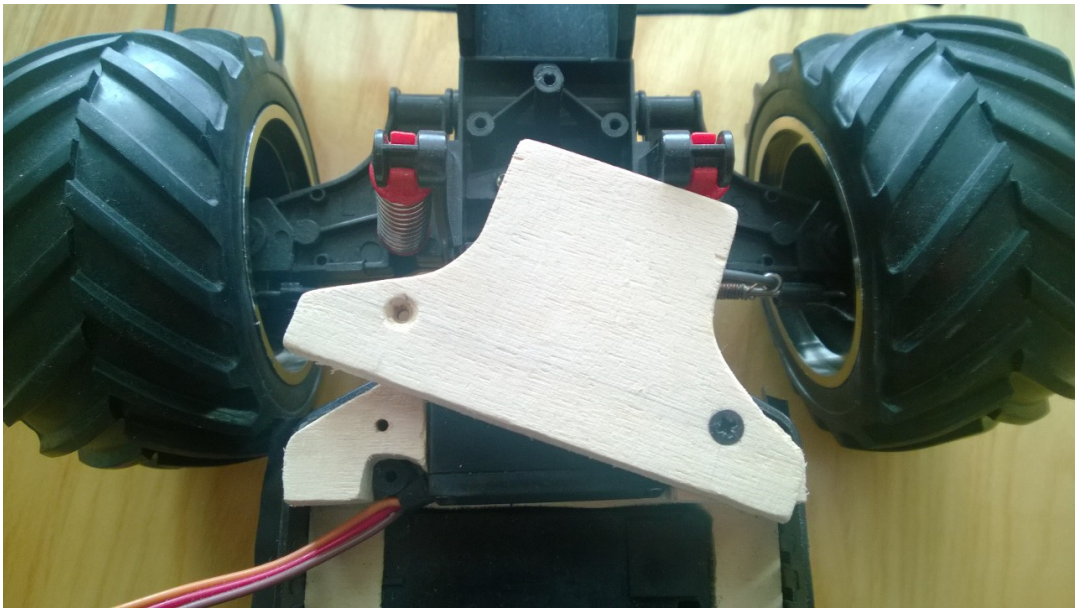


FIGURE 2.9 – Vue aérienne du montage pour le système de direction, avec le couvercle entrouvert.

"Servo" installée avec l'IDE Arduino. Ce que font les méthodes de cette classe est de produire un signal comme celui discuté à la section précédente en l'émettant par un des pins de l'Arduino capable de faire cette modulation. D'ailleurs, ce programme est très pratique pour tester le fonctionnement d'un servo-moteur.

2.3 Moteur de propulsion

2.3.1 Descriptif

Le moteur de propulsion, au contraire du servo, n'a pas été changé. Ses caractéristiques électriques (données que nous avons mesurées à l'aide d'un multimètre du gymnase) se trouvent dans le tableau (2.1). Le moteur est muni d'une boîte à vitesse ainsi qu'un différentiel. Nous avons estimé qu'il aurait été inutilement compliqué d'y apporter des modifications. La config-

FIGURE 2.10 – Schéma de la modulation du signal électrique [12]

uration déjà existante, à l'exception de l'électronique subvient tout à fait à nos besoins.

2.3.2 H-bridge

Faire tourner l'axe d'un moteur électrique pose peu de problèmes. Il suffit de connecter l'un des pôles à la tension positive et l'autre à la tension négative. Ceci fera tourner l'axe du moteur dans un sens. Si vous souhaitez le faire tourner dans le sens inverse, il vous suffira d'échanger les fils électriques aux pôles du moteur.

Le problème suivant se pose alors : comment inverser le sens de marche du moteur sans intervention manuelle ?

La réponse est donnée par un astucieux circuit composé de transistors. Il permet, au moyen de deux signaux actionnant les transistors, de contrôler le sens du courant passant dans le moteur. La figure 2.12 pourra d'avantage éclairer le lecteur.

Pour fabriquer un H-bridge, il faut utiliser entre autres des transistors NPN(Négatif-Positif-Négatif) et PNP(Positif-Négatif-Positif). La différence pratique entre ces deux types de transistors est que l'un demande un signal déclencheur haut pour être ouvert tandis que l'autre en demande un bas

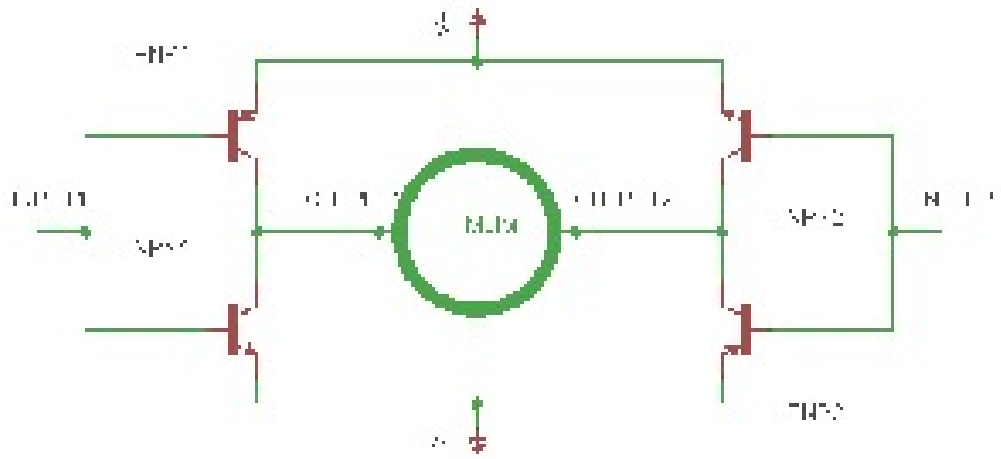


FIGURE 2.11 – Schéma d'un H-bridge

ou la terre. Dans ce cas, on utilise deux types de transistors différents (en grande partie pour clarifier le schéma), mais l'on pourrait très bien utiliser uniquement des transistors du même type. On obtiendrait le même résultat en connectant chaque *INPUT* à une paire de transistors diagonalement opposés [7] dans la figure 2.12.

On peut voir que selon le *INPUT*, on obtiendra des tensions au bornes du moteur ou *OUTPUT* variables.

Table de Vérité

Rédigeons un tableau de vérité pour mieux illustrer la situation, où H (*high*) signifie haut et L (*low*) signifie bas (voir tableau 2.2).

2.4 Senseur de Distance

Nous utilisons un senseur de distance nommé HC-SR04. Son fonctionnement est basé sur l'émission puis la détection d'ultrasons. En d'autres termes, c'est un sonar. Les dauphins et les chauve-souris utilisent le même système pour s'orienter. Son spectre d'opération s'étend de 2 à 400cm et

INPUT1	*INPUT2*	Tension
H	L	$OUTPUT1 = OUTPUT2$
L	H	$OUTPUT1 = OUTPUT2$
H	H	$OUTPUT2 > OUTPUT1$
L	L	$OUTPUT1 > OUTPUT2$

TABLE 2.2 – Table de vérité accompagnant le schéma du H-bridge (fig : 2.12)

Quand les signaux *INPUT* sont opposés, le moteur est à l'arrêt. Quand ils sont équivalents, le moteur est en marche, dans un sens ou dans l'autre.

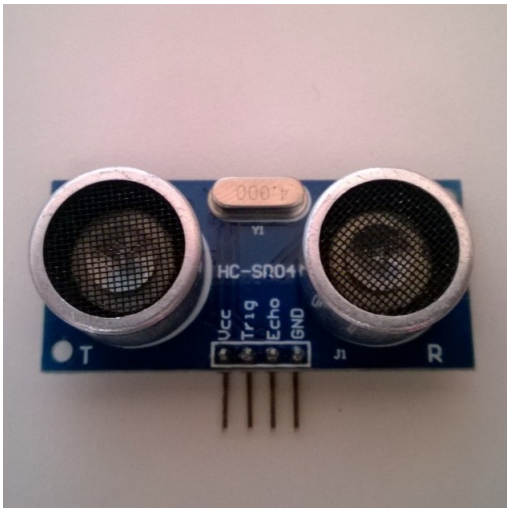


FIGURE 2.12 – Le senseur de distance à ultrason HC-SR04

n'est pas influencé par les conditions de lumière. Une matière avec des propriétés sonores particulières, tel que le tissu, peut biaiser les mesures du senseur.[2]

L'idée, c'est que les senseurs puissent nous apporter des informations supplémentaires à celles apportées par la caméra sur l'environnement dans lequel se trouve le véhicule. De plus, c'est de l'information simple, juste une distance, un entier, donc la transmission de cette donnée est très rapide par

rapport à celle d'une image vidéo.

Quelques caractéristiques intéressantes :

1. Voltage de fonctionnement : 5V
2. Courant en veille : $<$ 2mA
3. Courant en utilisation : 15mA
4. Angle de mesure efficace : $< 15^{\circ}$
5. Angle de mesure total : 30°
6. Spectre d'opération : 2-400cm
7. résolution : 0.3cm
8. Dimensions : 45mm x 20mm x 15mm

La figure (2.14) montre tout d'abord le signal à donner au senseur pour qu'il lance l'émission de la pulsation ultrason, soit un signal *HAUT* de $10us$. Ensuite, le senseur va effectivement émettre un signal d'ultrason modulé. Lorsqu'il reçoit le son en retour, le senseur va donner un signal *HAUT* de longueur proportionnelle au temps qu'il a attendu pour recevoir l'écho.

2.4.1 Programmation pour utiliser le senseur

Le code Arduino exemple se trouve en annexe, section A.3. Comme expliqué précédemment, l'Arduino va recevoir un signal d'une longueur dans le temps proportionnel à la distance mesurée par le senseur. Plus précisément, c'est le temps t en microsecondes que le senseur a pris pour recevoir l'écho en retour. Sachant que la vitesse du son est de 343.5m/s (donc le temps pour franchir 1cm équivaut à 29.1 us) et que il doit faire un aller-retour du véhicule à l'obstacle, on en déduit que la distance d en fonction du temps s'écrit comme suit :

$$d(t) = \frac{t}{2 * 29.1} \tag{2.1}$$

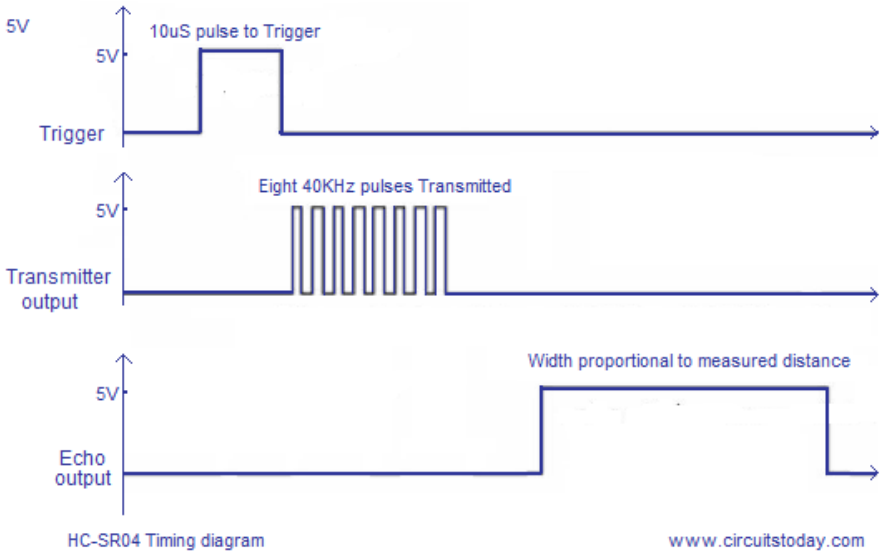


FIGURE 2.13 – Fonctionnement du HC-SR04 en fonction du temps

2.5 Coût des composants

Le tableau 2.3 contient les éléments clefs que nous avons utilisés pour construire notre drone.

Composant	Quantité	Prix [CHF]
Voiture radio-commandée	1	30.00
Arduino Uno	1	20.00
Raspberry Pi	1	35.00
Batterie externe (7000mAh)	1	28.00
Webcam	1	35.00
Edimax EW7811Un (wi-fi)	1	18.40
Hub USB	1	5.60
Capteur de distance (HC-SR04)	3	9.60
L293D	1	3.00
Câble micro USB	1	1.40
total	10	186.00

TABLE 2.3 – Synthèse des composants clés, leur prix et le prix total.

Chapitre 3

Software
Première version des systèmes embarqués

La première version du software consistait en une étape préliminaire dont le but était de prouver qu'on pouvait contrôler la voiture à distance par un réseau local Wi-Fi. Sa simplicité relative réside dans le fait que nous avons légué tout ce qui est transmission de données par le réseau à un logiciel tiers. Nous nous sommes surtout occupés ici de la communication Raspberry Pi, Arduino par USB et de trouver une première solution pour transmettre la vidéo.

3.1 Software de l'Arduino

Pour programmer l'Arduino, nous utilisons le logiciel dédié, *Arduino IDE*[1]. C'est un environnement *Open source* qui fonctionne sur Windows, Linux et Mac OS X. Il a été écrit en Java et est basé sur le logiciel Process-

1. IDE signifie *Integrated Development Environment*. C'est un ensemble d'outils qui sont mis à la disposition des programmeurs afin de simplifier et augmenter leur productivité.

ing. Ce logiciel nous permet de programmer, de compiler et de téléverser un code à un Arduino. Il possède aussi des exemples de codes afin de pouvoir apprendre à utiliser un Arduino sans avoir à chercher sur le web. Il prend en compte des librairies qu'on peut rajouter simplement en mettant la librairie dans le dossier "*arduino**librairies*". Ce software permet aussi à l'utilisateur d'envoyer des caractères ou des chaînes de caractères à l'Arduino par le biais du port *Serial*, il faut néanmoins appuyer sur entrée pour envoyer le caractère ou la chaîne de caractères.

3.2 Software du Raspberry Pi

Sans les logiciels, notre projet aurait bien du mal à se réaliser. Dans cette section, nous allons parler de tout les softwares que nous utilisons sur la Framboise pour la première version du drone. Dans un premier temps, nous allons rapidement aborder la question du sans fil et de la connexion que nous avons mise en place afin de contrôler le drone, avant de continuer sur le logiciel Guvcview, qui permet d'afficher une image. Par la suite, nous allons brièvement parler de notre programme python qui permet de contrôler le véhicule avant de finir le chapitre avec un résumé de points forts ou moins forts de cette première version.

3.2.1 Partage sécurisé

Nous utilisons un type de partage sécurisé entre deux ordinateurs, qui se nomme *ssh* pour *Secure Shell*. C'est un protocole permettant à un ordinateur de faire une connexion sécurisée avec un autre afin de le contrôler. Meilleur que VNC (*Virtual Network Computing*) dans le sens où il ne demande pas au Raspberry Pi de dupliquer le *Desktop* (bureau), ce qui nous permet d'améliorer les performances, tant du côté de la réactivité des commandes, que du nombre d'images par secondes de la caméra envoyé par le

Raspberry Pi. Il y a deux inconvénients majeurs à ces solutions, la première, c'est qu'il faut établir la connexion et lancer les programmes manuellement. Le deuxième inconvénient, c'est qu'il est beaucoup plus compliqué, voire impossible à un utilisateur n'utilisant pas un système UNIX de se connecter au Raspberry Pi. Ce qui fait en sorte que notre drone ne toucherait pas un grand public.

3.2.2 Guvcview

Guvcview est un software permettant d'enregistrer des séquences vidéos ou des images, il fournit aussi une image de contrôle, que nous utilisons dans la première version du drone. Ce logiciel très facile d'utilisation permet d'avoir près de sept images par secondes lors d'une connexion *ssh* avec une résolution d'environ 240x160 pixels. L'inconvénient de ce logiciel est que l'image de contrôle ne peut pas être affichée seule, elle s'accompagne toujours du panneau de réglages. De plus, il y a un temps de latence d'environ 700ms.

3.3 Python

Outre les programmes mentionnés auparavant, nous avons développé notre propre code python qui permet de contrôler le véhicule. L'interface graphique est très simple (fig : 3.1). Il s'agit simplement d'une fenêtre qui est à l'écoute du clavier et qui, lorsque l'utilisateur presse une touche, va envoyer le caractère à l'Arduino via le port de communication *Serial*. L'Arduino va ensuite, grâce à son propre code définir quelle action le véhicule doit faire. Ce programme permet non seulement de choisir la direction à prendre, mais aussi de régler l'angle de braquage ou la vitesse maximale. Le programme ne prend pas en compte l'utilisation de capteurs, tel que le capteur de distance. Il permet simplement de diriger le véhicule. Une

zone de texte non éditable par l'utilisateur permet d'afficher des messages, signalant, par exemple, que le pilote a atteint le braquage maximal. Le programme n'est pas conscient de l'état physique de la voiture, mais se base uniquement sur les paramètres sélectionnés par le pilote pour imprimer de tels messages.

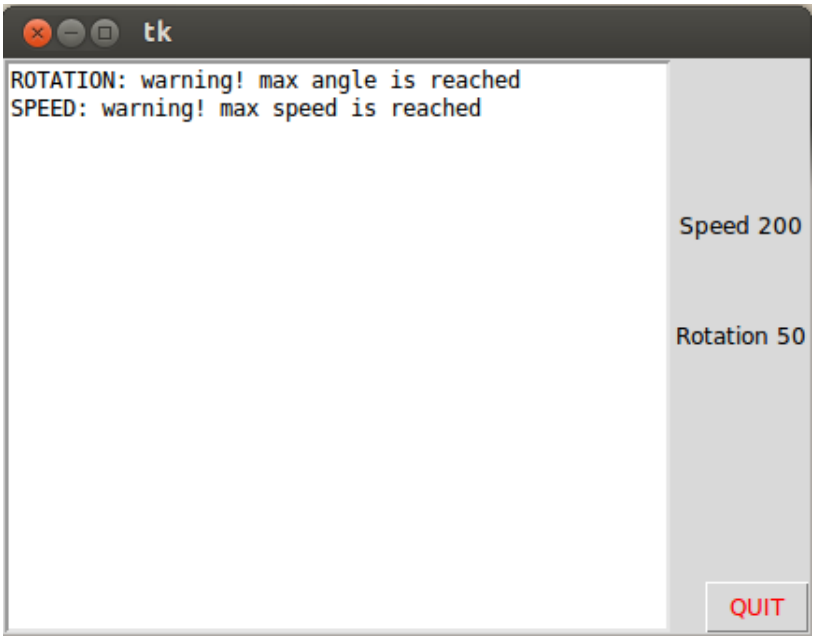


FIGURE 3.1 – Fenêtre X du code python qui permet la lecture du clavier. Cette fenêtre permet de contrôler le drone. Le programme affiche sur la colonne de droite l'angle de rotation des roues ainsi que la vitesse maximale du drone fixée par le pilote.

3.4 Évaluation de la première version

Avant de commencer à parler de la deuxième version, nous allons analyser quels sont les points forts ou points faibles de cette première version. Les avantages de cette version sont surtout au niveau des programmeurs.

En effet, cette version est assez simple à programmer, dans le sens où nous utilisons des programmes qui existent déjà et nous nous sommes surtout occupé de l'IHM (Interface Homme Machine). Les communications (ssh ou serial) sont très simple à utiliser. Les inconvénients se placent surtout au niveau de l'utilisateur. En effet, il n'est pas très pratique de devoir ouvrir un Terminal, se connecter au drone via ssh, lancer trois programmes manuellement avant de pouvoir utiliser le drone. De plus, la connexion cryptée fait baisser la rapidité du transfert de données (qui est surtout visible avec l'image de la caméra). C'est pourquoi nous nous sommes lancés dans une deuxième version du drone, qui, en plus d'avoir de nouvelles fonctions, est plus simple d'utilisation et plus fluide.

Chapitre 4

Software
Deuxième version des systèmes embarqués

DANS notre projet, nous avons prévu une deuxième version du drone, dans le sens où la première version ne remplissait pas tout à fait les buts que nous nous étions fixés. Cette version, principalement programmée en Java et en C++, nous permet de faire nous-même la connexion entre l'ordinateur et le Raspberry Pi. Le C++ a été utilisé dans tout ce qui touche la caméra. Il a été choisi car il est beaucoup plus rapide que le Java, et donc essentiel au niveau de l'image, où nous cherchons les meilleures performances. Le reste a été codé en Java. Nous avons choisi ce langage pour plusieurs raisons. Tout d'abord, nous avions des notions en Java, ce qui n'est pas négligeable quand on doit choisir un langage de programmation, c'est aussi un langage constamment mis à jour et il possède une large communauté, ce qui fait que de nombreuses classes existent, notamment du côté des interfaces graphiques, sans lesquelles nous aurions de la peine à rendre le programme agréable à utiliser. Dans ce chapitre, nous allons tout d'abord

aborder le type de structure employé et le choix de protocole que nous avons adopté, nous allons ensuite voir à quoi ressemble notre programme Java et ce qu'il fait. Avant de parler de la vidéo avec le programme en C++ ainsi que OpenCV.

4.1 Client/Serveur

Le modèle client-serveur est composé de deux parties : les serveurs qui fournissent un service et les clients qui bénéficient de celui-ci. Un client ne partage pas ses ressources avec les autres, mais profite de celles du serveur. Le serveur attend qu'un client vienne se connecter et faire des requêtes auxquelles il répond. Il doit généralement être capable de gérer plusieurs clients à la fois.

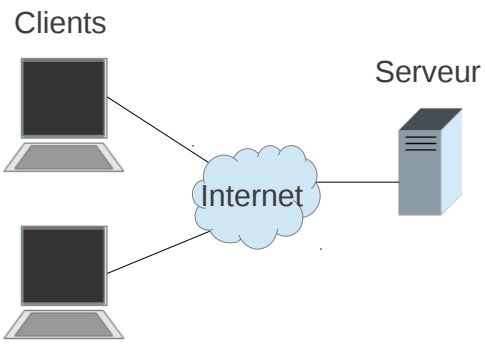


FIGURE 4.1 – Structure client-serveur.

Nous avons établi que la voiture occupe le rôle de serveur qui est exécuté sur le RaspberryPi. Il contrôle les mouvements de la voiture par l'intermède de l'Arduino. Le client peut être n'importe quelle machine sur le réseau local utilisant le côté client du code. Le code est effectivement constitué de deux "main class", c'est à dire des applets, "Server" et "Client".

Un Thread est initié pour chaque nouvelle demande de connexion. C'est-à-dire que le serveur peut s'occuper en parallèle des requêtes de différents clients (voir fig. 4.2).

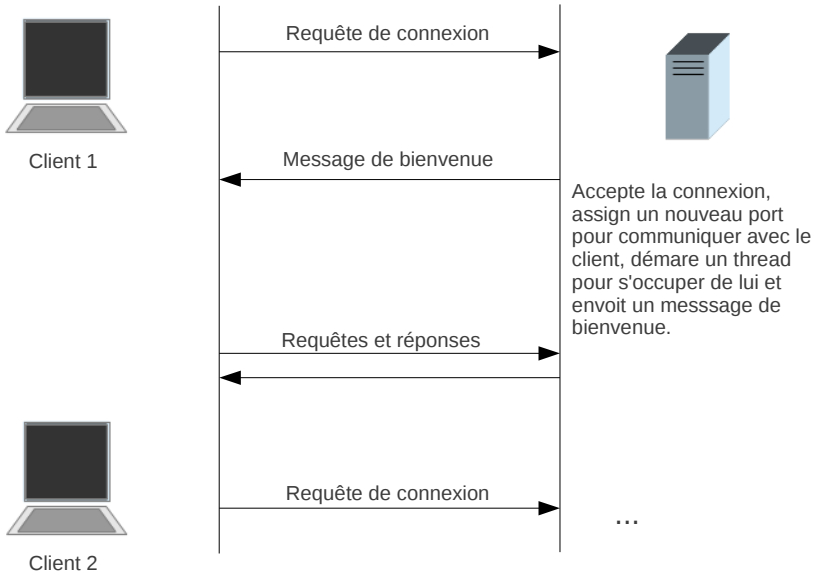


FIGURE 4.2 – Schéma de la procédure suivie à chaque nouvelle connexion.

4.2 UDP ou TCP

Avant de parler de notre choix, nous allons un peu expliquer ce que sont les protocoles UDP et TCP et leurs différences. UDP et TCP sont deux protocoles de communication réseau qui permettent d'envoyer des données, (*Packets*), d'un ordinateur à un autre.

Le protocole UDP [1] est un protocole qui n'est pas orienté connexion. Les paquets qui transitent via un protocole UDP ont l'adresse IP du récepteur,

1. *User Datagram Protocol*, soit protocole des datagrammes d'utilisateurs

comme la poste, mais il n'y a aucune garantie que les paquets soient bien arrivés sans parler de leur ordre. Par contre, il n'y a pas le souci de perte de connexion puisque il n'y en a pas. Ce protocole est particulièrement adapté pour la transmission de vidéo. Pour contrôler les mouvements du véhicule, il est impératif qu'il sache à tout moment s'il y a au moins un client qui est au contrôle, au risque que le véhicule soit livré à lui-même. Ici rentre en jeu le protocole TCP.

Le protocole TCP [2] est l'opposé de l'UDP, il est orienté connexion. Lorsque le client envoie des données au serveur, ce premier est informé de l'arrivée des dites données. Il y a donc un système de reçu qui confirme la réception des packets. S'il y a des fichiers corrompus, l'expéditeur va renvoyer les fichiers manquants. On peut, par analogie, comparer ce protocole à la communication téléphonique : on remarque assez rapidement quand il y a une coupure dans la ligne.

4.3 Java

Le code complet Java est très long et est composé de beaucoup de classes. Plutôt que de tout détailler ou pire, juste mettre le code cru, nous avons sélectionné les thèmes les plus intéressants que nous avons rencontrés pendant l'écriture du programme. Nous allons expliquer comment nous avons structuré le système, selon l'architecture et le protocole discutés en section 4.1 et 4.2.

4.3.1 Gestion des Requêtes de connexion

Pour établir une connexion avec le serveur, un client va ouvrir un port de communication avec lui, appelé "Socket". Une fois cette opération complétée

2. *Transmission Control Protocol*, soit protocole de contrôle des transmissions

avec succès, il sera possible d'obtenir les "Streams" [3] entrant et sortant pour effectivement envoyer des données.

```
//ouverture du port de communication avec le host(Adresse IP ou domain name)
Socket serverSocket = new Socket(host, port);
//Obtention des Input et Output Stream encapsules dans un PrintStream et un BufferedReader
PrintStream out = new PrintStream(serverSocket.getOutputStream());
BufferedReader in = new BufferedReader(new InputStreamReader(serverSocket.getInputStream()));
```

De l'autre côté, le serveur attend les demandes de connexion, ceci grâce à la classe welcomeSocket qui s'occupe de ce genre de requêtes. Une fois la requête de connexion acceptée le socket de communication est récupéré et un thread est lancé avec en argument le socket de communication pour pouvoir poursuivre l'échange de données. Finalement, le thread est stocké dans une liste, ainsi, quand le serveur doit envoyer un message aux clients il itère simplement à travers la liste de thread.

```
ServerSocket welcomeSocket = new ServerSocket(4444);
System.out.println("Server is running...");
while(true){
  // Socket de transfert de donnees
  final Socket clientSocket = welcomeSocket.accept();
  // Creation d'un nouveau Thread
  ServerThread st = new ServerThread( clientSocket );
  st.start();
  serverThreads.add(st);
}
```

Puisqu'il est dans une boucle while, il se remet en attente de la prochaine requête de connexion.

3. Tuyau pour transmettre des bytes

4.3.2 Gestion des requêtes suivantes

Le Client dispose d'une variété de requêtes prédéfinies. A titre d'exemple on va définir des pseudos requêtes avancer, reculer et s'arrêter. De plus, lié à la requête vient s'ajouter une option pour donner plus de précision sur, dans ce cas, la vitesse. Une pseudo requête se présenterait alors sous cette forme :

Avancer(20)

Ce qui voudrait dire : "Rouler vers l'avant à 20cm/s". Pour bien comprendre la demande d'un client le serveur a également cette même liste de requêtes prédéfinies dont il sait s'occuper. Une fois que le serveur a trouvé la requête dans sa liste, il exécute l'opération prévue pour cette situation. Le schéma 4.3 illustre cette procédure. Le tableau de requêtes est en fait un HashMap, les identifiants des requêtes se trouvent à gauche en majuscule et les références aux fonctions à exécuter se trouvent à droite.

```
1 public static HashMap<String, RequestHandler> RequestHandlers = new HashMap<String,
      RequestHandler>();
2 RequestHandlers.put("ON", new ONRequestHandler());
3 RequestHandlers.put("OFF", new OFFRequestHandler());
4 RequestHandlers.put("FOR", new FORRequestHandler());
5 RequestHandlers.put("BAK", new BAKRequestHandler());
6 RequestHandlers.put("STO", new STORequestHandler());
7 RequestHandlers.put("LEF", new LEFRequestHandler());
8 RequestHandlers.put("RIG", new RIGRequestHandler());
9 RequestHandlers.put("CNT", new CNTRequestHandler());
```

4.3.3 librxtx

Afin de pouvoir communiquer entre l'Arduino et le RaspberryPi via USB, nous avons fait appel à une librairie Java supplémentaire qu'il a fallu installer. librxtx s'occupe de gérer les ports de communication USB et d'établir

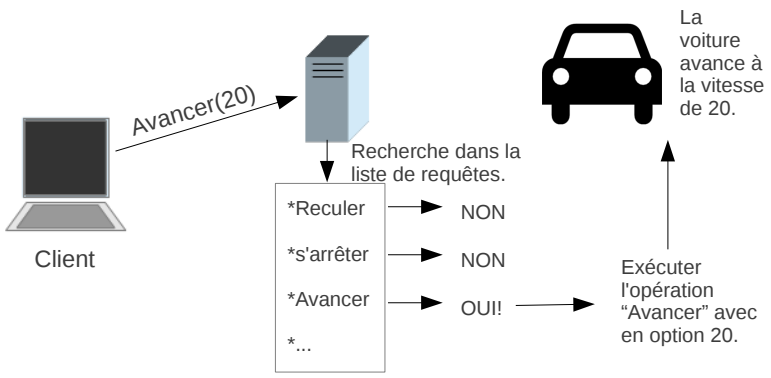


FIGURE 4.3 – Schéma de la procédure suivie pour une requête au serveur.

une communication avec un appareil connecté par ce moyen.

Grâce à cette librairie nous avons pu écrire notre propre classe qui permet de facilement sonder le système pour un appareil tel que l'arduino, établir une connexion à cette appareil et lui envoyer des données. Le système de réception de messages USB est basé sur un modèle évènementiel, expliqué dans la section 4.3.4.

4.3.4 Evénements et écoute d'un Stream

La programmation événementielle permet dans certains cas d'améliorer l'efficacité d'une application. Elle est basée sur des évènements ou changements d'états qui déclenchent des autres morceaux de code. Ce type de programmation s'oppose à la programmation dite séquentielle.[13]

Pour illustrer cette architecture, imaginons deux téléphones, l'un est dépourvu de haut-parleurs (voir fig.4.4). Pour chatter, l'utilisateur du téléphone défectueux devra régulièrement contrôler s'il a reçu un nouveau message et souvent il contrôlera pour rien, tandis que l'autre sera averti d'un nouveau message (changement d'état, événement !) et contrôlera uniquement à ce moment là.

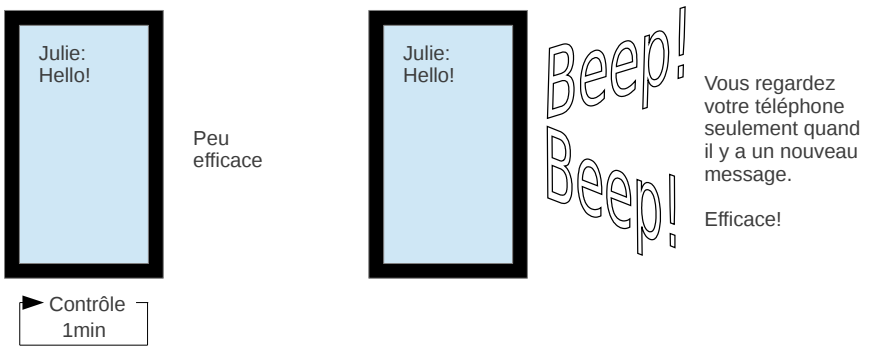


FIGURE 4.4 – L'utilisateur du téléphone sans haut-parleur contrôlera régulièrement son téléphone et souvent pour rien.

Notre code java utilise ce système pour tout ce qui concerne la réception de message, donc l'écoute de Stream, c'est à dire que dès qu'un message est reçu, aussi bien par le client que par le serveur, le code pour gérer ce message est exécuté. L'interface graphique (GUI) est basée par principe sur des événements, par exemple : "boutton pressé", "champ de texte modifié", "touche enfoncée". Voici justement un exemple de programmation événementielle pour l'interface graphique :

```
1  //Creation d'un boutton avec l'ecriture "CONNECT".
2  JButton connectButton = new JButton("CONNECT");
3  //On dit de lancer un avertissement quand le boutton est presse.
4  connectButton.addActionListener(new ActionListener(){
5  //On definit ensuite le code a executer en consequence.
6  //Dans ce cas, la procedure de connexion est executee.
7  @Override
8    public void actionPerformed(ActionEvent e) {
9      ipAddress = ipAddressTextField.getText();
10     port = Integer.parseInt(portTextField.getText());
11
```

```
12      Client.connect(ipAddress, port);
13    }
14  });
```

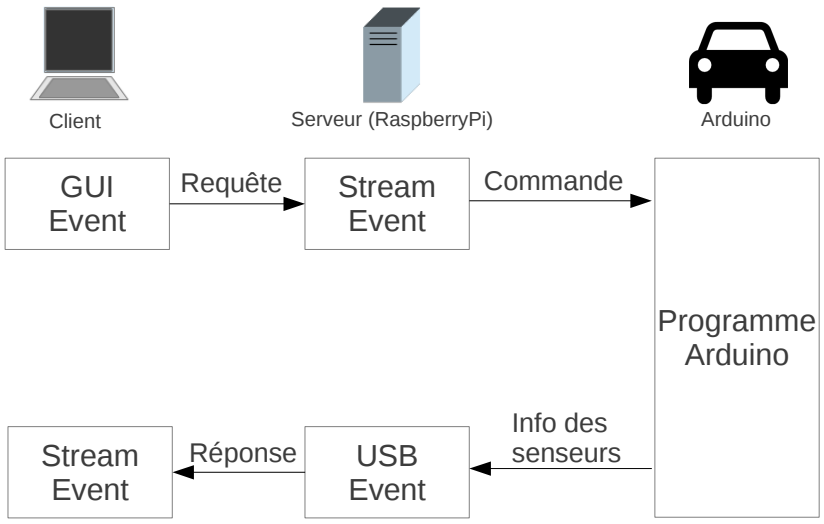


FIGURE 4.5 – Cette figure illustre une suite d'événements possible dans l'architecture de notre projet

4.3.5 IHM

Pour faciliter l'utilisation du programme Java et pour capter les événements du clavier afin de contrôler la voiture, nous avons créé une interface graphique en Swing [4]. Au travers de l'onglet connexion (voir fig. 4.6) de l'IHM, l'utilisateur peut se connecter au serveur en entrant l'adresse IP de celui-ci. Une fois la connexion établie, le pilote se dirige ensuite vers l'onglet contrôle pour commencer à guider le véhicule (voir fig. 4.7).

4. Swing est un package qui rassemble les classes nécessaires à créer une interface graphique.

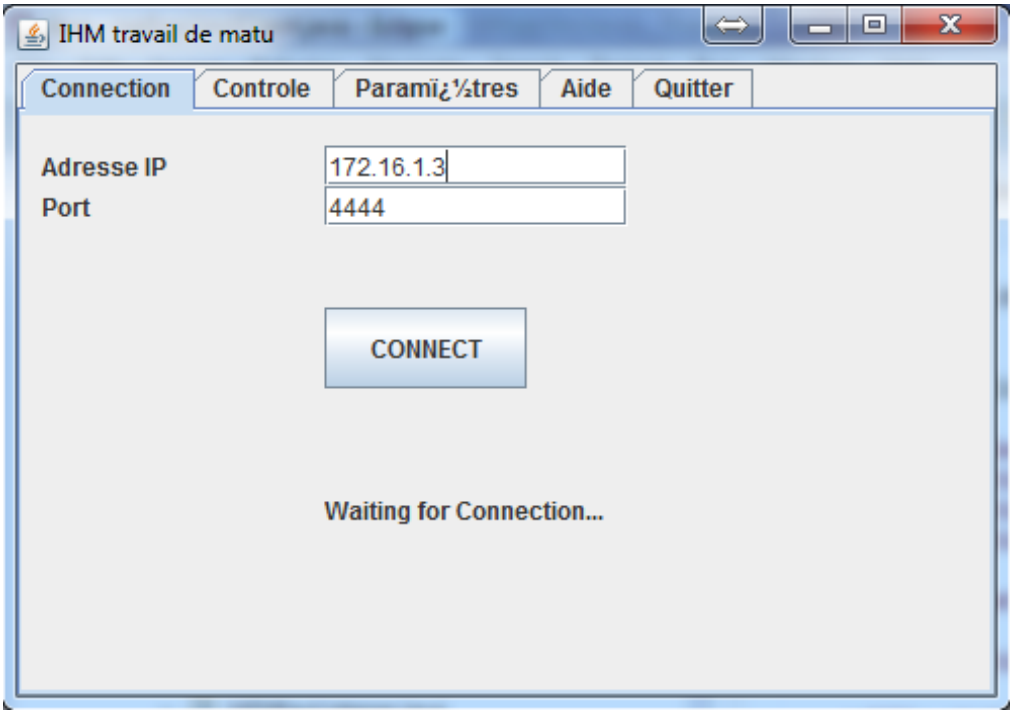


FIGURE 4.6 – Onglet de connexion de l'interface graphique Java

Les paramètres par défaut pour guider le véhicule sont les touches fléchées pour avancer, reculer et tourner, ainsi que les touches a, s, d, et w pour changer la vitesse et le braquage des roues. Ces derniers paramètres sont représentés par des glisseurs que l'on peut aussi modifier à l'aide de la souris.

4.4 C++ et vidéo

Après avoir parlé de ce que faisait notre code Java, nous allons ici aborder la partie touchant à la vidéo, et par conséquent, au code C++. Comme dit précédemment, nous utilisons le protocole de réseau UDP pour transmettre la vidéo. La raison du choix de ce langage réside dans le fait que c'est un langage qui est exécuté beaucoup plus rapidement qu'un langage de haut niveau tel que Java (la communauté considère que cette différence

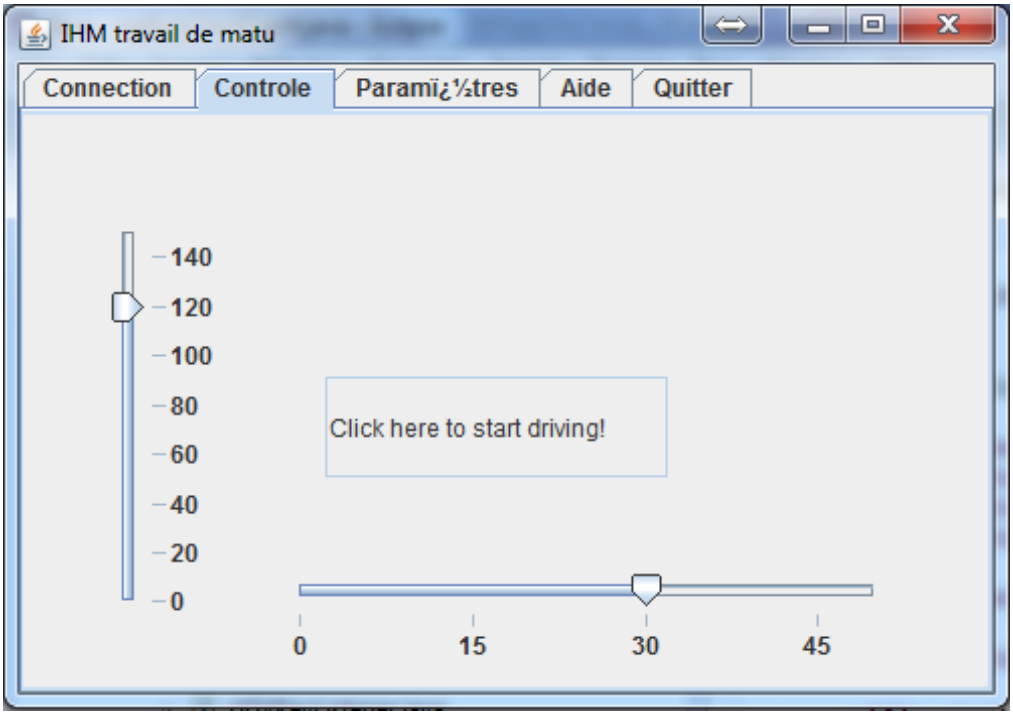


FIGURE 4.7 – Onglet de contrôle de l'interface graphique Java, permettant à l'utilisateur de guider la voiture.

de rapidité équivaut à un facteur trois tant que le programme est codé en C++ de bas niveau). Le programme en C++ est un programme invisible par l'utilisateur. En effet, il fonctionne en arrière plan. Voici le plan du programme :

1. Ouverture d'un socket du côté du Raspberry Pi et de l'ordinateur distant
2. Capture du flux vidéo de la caméra connectée à la Framboise
3. Compression du flux video
4. Envoi du flux vers l'ordinateur distant
5. Réception et décompression des packets reçus
6. Affichage de l'image dans le programme Java

4.4.1 Client/Serveur

Nous allons maintenant aborder une petite présentation du code pour comprendre comment fonctionne notre client serveur. Par soucis de simplicité, nous n'allons que parler du code écrit pour le système Windows.

Tout d'abord, il faut indiquer quelles librairies nous utilisons, ce sont les lignes suivantes. La première ligne permet d'utiliser les socket Windows. Les autres lignes sont des librairies standards en C++.

```
1 #include <winsock2.h> //fonction socket windows
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <string>
5 #include <cstdio> //pour les Sprintf
```

Une fois que ceci est fait, nous initialisons quelques variables dont il n'est pas utile de préciser à quoi elle vont servir. Ensuite, nous initialisons deux objets aux deux lignes ci-dessous. Nous verrons un peu plus tard à quoi ils servent.

```
1 SOCKET socketId;
2 SOCKADDR_IN destInfo;
```

Le premier objet va se voir attribuer les données contenues dans la fonction **socket()**

```
1    socketId = socket(AF_INET, SOCK_DGRAM, 0);
```

La fonction **socket(famille, type, protocole)** possède 3 arguments. Le premier, **AF_INET** est la famille de protocole utilisée, elle utilise une adresse internet IPV4, c'est a dire qui est encapsulée sur 4 octets. Le deuxième argument détermine le type de communication à faire, dans notre cas, **SOCK_DGRAM** signifie que nous sommes en UDP. Le troisième argument permet de spécifier un protocole qui permet de fournir un service désiré, dans notre cas, nous

n'avons besoin de rien, il est donc à 0. **socketId** va donc recevoir les argument de la fonction **socket()**. Ils seront utilisés un peu plus tard.

Le code suivant explique comment envoyer des données à une adresse et un port donné. La première ligne appelle la fonction **strcpy** qui copie le contenu du pointeur qui se trouve dans le tableau de caractères **Image** dans le tableau de caractères **buffer**. La deuxième ligne est à proprement parler la ligne qui envoie des données au serveur. Le **numberChar** en début de ligne ne sert à rien dans le processus d'envoi, mais il sert à contrôler par la suite si les données ont été correctement envoyées.

```
1  strcpy(buffer, Image);
2  numberChar = sendto(socketId, buffer, strlen(buffer), 0, (struct sockaddr*)&destInfo,
       sizeof(destInfo));
3  //sendto(int socket, char * buffer, int len, int flags, const struct sock_addr *
       dest_addr, int sock_len dest_len).
```

Vient ensuite la fonction **sendto()** qui prend les arguments comme mentionné à la ligne 3 du code. Une petite explication sur ces arguments. Comme vu précédemment, l'argument **socketId** contient les indications sur la destination. L'argument **buffer** représente un tampon [5] qui contient les octets à envoyer. L'argument **strlen(buffer)** indique le nombre d'octets à envoyer, soit la taille du tableau. Le quatrième argument spécifie le type d'envoi, dans notre cas, l'envoi est normal, donc nous assignons la valeur 0 à cet argument. L'argument suivant (soit le cinquième) permet de pointer sur la structure **destInfo** qui contient des informations sur la destination. Enfin, le dernier argument spécifie la longueur de la structure **destInfo**.

Nous allons maintenant voir la fonction qui permet au serveur d'ouvrir

5. Un tampon, ou plutôt une mémoire tampon est une allocation d'une partie de la mémoire vive (RAM) qui est utilisée pour stocker temporairement les données. Il est utilisé dans ce cas car le processus d'envoi est plus lent que le processus de lecture

un socket et se mettre en écoute. Le code suivant permet de créer un socket. Elle se construit de la manière suivante, **bind(int socket, socketaddr * description, int len)**. En d'autres termes, ces arguments permettent de savoir sur quel port il faut ouvrir un socket et quel type d'adresse utiliser.

```
1 erreur = bind(socketId,(struct sockaddr*)&sourceInfo,sizeof(sourceInfo));
```

Le code suivant correspond a la réception de donnée. Nous utilison la fonction **recvfrom()** qui contient les arguments comme écrit à la ligne 3 du code. Le nombre maximum de bytes que le serveur va accepter de lire en un envoi à été arbitrairement choisi puisqu'il est plus grand que un bloc de bytes (soit 1024 bytes).

```
1       tempo = sizeof(sourceInfo); // Passe par une variable afin d'utiliser un pointeur
2       numberChar = recvfrom(socketId,buffer,1515,0,(struct sockaddr*)&sourceInfo,&tempo);
3       //recvfrom(int socket, char *data, int maxnumbytes, int flags, sosckaddr *source, int
            source_len)
4       buffer[numberChar] = 0;
```

La dernière ligne du code permet de fermer le tableau après avoir reçu les données car la fonction **recvfrom()** ne le fait pas.

4.4.2 OpenCV

Ces programmes client/serveur ont pour but de transmettre l'image de la caméra, mais jusqu'ici, nous n'avons pas parlé de l'image en soit, c'est ce dont nous allons aborder maintenant. Ce sujet n'a pas encore abouti pour l'instant, c'est pourquoi le conditionnel sera utilisé dans cette section. Nous aimerions utiliser OpenCV pour pouvoir capturer le flux vidéo de la caméra. OpenCV est un ensemble de librairies qui sont utilisées pour le traitement d'image. Ces librairies sont écrites en C et C++. Non seulement, elle nous permettraient d'avoir l'image de la caméra, mais en plus, de pouvoir faire du traitement sur celle-ci (aucun traitement d'image ne se ferait sur le Rasp-

berry Pi car il ne possède pas la puissance nécessaire pour le faire). Un problème que nous avons rencontré est la difficulté d'installer ces librairies sur les ordinateur, en particulier sur l'OS Raspbian où chaque compilation dure plusieurs heures. Néanmoins, nous comptons continuer dans cette voie car ce sont les meilleures librairies que nous avons trouvés.

4.5 Codes

Concernant les codes complets, le lecteur intéressé pourra les trouver dans le projet publique UGV sur `http://github.com/rico500/UGV` .

Chapitre 5

Conclusion

DÉBUTÉ il y a maintenant une année, ce projet n'était qu'un rêve avant de devenir un travail de maturité pour finalement se concrétiser. Le véhicule a beaucoup évolué depuis sa première ébauche et a pris autant de directions aussi inattendues qu'intéressantes et instructives. Bien que le résultat final ne soit pas aussi raffiné qu'on l'espérait, il est tout de même le témoin de l'important investissement de temps et d'énergie dans ce travail.

Au point où nous en sommes, le drone ne pourrait pas mener de vraies missions de reconnaissance ou de recherche. Limité par sa construction fragile et sa portée qui laisse à désirer. Par contre, grâce à ce prototype, nous avons pu comprendre, en profondeur, l'investissement qui demande la construction d'un engin pareil. Nous avons développé une structure informatique et appris à concevoir des circuits imprimés. De ce fait, nous pouvons transposer ces connaissances à un autre véhicule plus solide que le notre. De plus, ce travail nous ouvre des portes sur des projets plus complexes, tel que des véhicules collaborateurs qui pourraient étendre la zone de couverture WI-FI avec des répétiteurs mobiles ou un robot capable de cartographier son environnement.

Concernant le prix du drone, nous arrivons à un total de 186.- CHF, ce qui reste tout de même en dessous de la barre des 200.- CHF. Nous pour-

rions encore améliorer ce prix tout en gardant les mêmes fonctionnalités. Ceci, en utilisant un Arduino plus petit ou en s'en débarrassant totalement et tenter de tout faire avec le Raspberry Pi.

Annexe A

Sketchbook

A.1 Sketch exemple pour le moteur

```
#define A 12 // define input pin "A"
#define B 11 // define input pin "B"
#define E 10 // define enable pin "E"

byte i;

void setup(){

// tous les pins sont des "OUTPUT", ils vont contoler le moteur

  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(E, OUTPUT);
}

void loop(){

// boucles faisant varier la vitesse lineairement en avant et en arriere

```

```
20    for(i = 0; i < 200; i++){
21      forward(A, B, E, i);
22      delay(50);
23    }
24    for(i = 200; i > 0; i--){
25      forward(A, B, E, i);
26      delay(50);
27    }
28    for(i = 0; i < 200; i++){
29      backward(A, B, E, i);
30      delay(50);
31    }
32    for(i = 200; i > 0; i--){
33      backward(A, B, E, i);
34      delay(50);
35    }
36  }
37
38  //methode globale pour controler le moteur, fait appel aux
39  //methodes halt, backward et forward.
40  void motor(int speedo){
41    if (abs(speedo) > 255){
42      halt(A,B);
43      break;
44    }
45    else if (speedo > 0){
46     forward(A, B, E, speedo);
47    }
48    else if (speedo < 0){
49      backward(A, B, E, speedo);
50    }
51    else if (speedo == 0){
52      halt (A, B);
```

```
  }
}

//fonction faisant tourner le moteur en avant ou l'inverse d'en arriere.
void forward(byte pin1, byte pin2, byte pinEnable, byte speedo){
  digitalWrite(pin1, HIGH);
  digitalWrite(pin2, LOW);
  analogWrite(pinEnable, speedo);
}

//fonction faisant tourner le moteur en arriere ou l'inverse d'en avant.
void backward(byte pin1, byte pin2, byte pinEnable, byte speedo){
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, HIGH);
  analogWrite(pinEnable, speedo);
}

//fonction servant a arreter le moteur.
void halt(byte pin1, byte pin2){
  digitalWrite(pin1, LOW);
  digitalWrite(pin2, LOW);
}
```

A.2 Sketch exemple pour le servo

[8]

```
// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.


#include <Servo.h>
```

```
7
8  Servo myservo; // create servo object to control a servo
9                 // a maximum of eight servo objects can be created
10
11 int pos = 0;   // variable to store the servo position
12
13 void setup()
14 {
15   myservo.attach(9); // attaches the servo on pin 9 to the servo object
16 }
17
18
19 void loop()
20 {
21   for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
22   {                                 // in steps of 1 degree
23     myservo.write(pos);             // tell servo to go to position stored
24                                     // in variable 'pos'
25     delay(15);                      // waits 15ms for the servo
26                                     // to reach the position
27   }
28   for(pos = 180; pos>=1; pos-=1)  // goes from 180 degrees to 0 degrees
29   {
30     myservo.write(pos);
31     delay(15);
32   }
33 }
```

A.3 Sketch exemple pour le senseur HC-SR04

```
1 //HC_SR04
2 //Authors: Eric Brunner & Sven Borden
```

```
3  //Inspired by example code on trollmaker.com
4  //more at:
5  //http://trollmaker.com/article3/arduino-and-hc-sr04-ultrasonic-sensor
6
7  #define trigPin 7
8  #define echoPin 8
9
10 void setup() {
11   Serial.begin (9600);
12   pinMode(trigPin, OUTPUT);
13   pinMode(echoPin, INPUT);
14 }
15
16 void loop() {
17   long microseconds, cm;
18
19   /*Pour que le signal declencheur HAUT soit propre,
20   on commence par lancer un court signal BAS*/
21   digitalWrite(trigPin, LOW);
22   delayMicroseconds(2);
23
24   //Ensuite le signal que attend le senseur!
25   digitalWrite(trigPin, HIGH);
26   delayMicroseconds(10);
27   digitalWrite(trigPin, LOW);
28
29   /*On enregistre la longueure du signal HAUT
30   renvoye par le senseur*/
31   microseconds = pulseIn(echoPin, HIGH);
32
33   //la distance en cm est calculee puis...
34   cm = (microseconds/2) / 29.1;
35   //...affichee
```

```
36    Serial.println(cm);
37
38    delay(100);
39
40  }
```

Bibliographie

[1] ?? Linux uvc driver and tools, ?? http://www.ideasonboard.org/uvc/#documentation.

[2] Cytron Technologies Sdn. Bhd. User's manual, Mai 2013.

[3] Futurlec. Technical information - national semiconductor 7805t datasheet, Novembre 2004. www.futurlec.com/Linear/7805T.shtml (voir partie TO-220).

[4] From The Open Source Initiative. Open source definition, ?? http://opensource.org/osd.

[5] Texas Instrument. L293, l293d quadruple half-h drivers, Juin 2002. http://idmax.free.fr/Aide/Stepper/l293.pdf.

[6] Ladyada. Adafuit motor shield, power usage, août 2012. http://learn.adafruit.com/adafruit-motor-shield/power-requirements.

[7] Robot Room. H-bridge motor driver using bipolar transistors, ?? http://www.robotroom.com/BipolarHBridge.html.

[8] Barragan Studio. Servo sweep, Septembre 2010. http://arduino.cc/en/Tutorial/Sweep.

[9] Arduino Team. Arduino spec., ?? http://arduino.cc/en/Main/ArduinoBoardUno.

[10] From Wikipedia the free encyclopedia. Microcontrôleur, ?? http://fr.wikipedia.org/wiki/Microcontr\%C3\%B4leur.

[11] From Wikipedia the free encyclopedia. Raspberry pi, ?? http://en.wikipedia.org/wiki/Raspberry_Pi.

[12] From Wikipedia the free encyclopedia. Servo control, Octobre 2012. http://en.wikipedia.org/wiki/Servo_control.

[13] From Wikipedia the free encyclopedia. Event-driven programming, Octobre 2013. http://en.wikipedia.org/wiki/Event-driven_programming.

[14] BeeWi Simply wireless. Bwz200 - buggy wifi avec camera, 2013. http://www.bee-wi.com/buggy-wifi-avec-camera-bwz200-a1-beewi,fr,4,BWZ200-A1.cfm.