UGV

(Unmanned Ground Vehicle)

Sven Borden — Travail de maturité

Eric Brunner — Gymnase de Morges

23 octobre 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 quadcopter (hélicoptères à quatres 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 informatiques 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 des plaques électroniques 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. Mme Pauline Pidoux pour nous avoir aidé lors de la rédaction de ce travail et 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 mettant sa vie en péril. Ce type de drone existe également sur le marché des jouets, par exemple le *Beewi WiFi Camera Buggy BWZ200-A1* [16], qui coûte septante-neuf francs. Notre projet ne consiste pas à fabriquer un drone ayant un fusil permettant de "dégommer" tous 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[16] (une portée améliorée, une meilleure autonomie et une meilleure qualité vidéo) . En effet, si nous parvenons 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 inscrit au concours suisse de science *La science appelle les jeunes* [1]. Ce concours réunis presque tous les domaines, pourvu que certains critères soient respectés. Tous les concurrents continuent un projet qu'ils avait 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.

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

Table des matières

Avant-propos 1
Remerciements 2
Résumé 1
Introduction 6

1 Hardware **7**
1.1 Arduino 7
1.1.1 Choix du type d'Arduino 7
1.1.2 Détails techniques de l'Arduino 9
1.1.3 Avantages et alternatives 9
1.2 Raspberry Pi 10
1.2.1 Détails techniques du Raspberry Pi 10
1.2.2 Avantages et utilisations 11
1.2.3 Choix de l'OS 11

2 La mécanique et l'électronique **12**
2.1 Caractéristiques du véhicule 12
2.2 PCB 12
2.2.1 Plaque personnelle 13
2.2.2 Schema électronique 15
2.2.3 Design 15
2.2.4 Composants 16

2.2.5 Commande et conception 22
2.3 Système de guidage 22
2.3.1 Système directionel initial 22
2.3.2 Remplacement du Système de guidage 23
2.3.3 Géométrie d'Ackermann 23
2.3.4 Contrôle du servo 23
2.3.5 Programmation pour contrôler un servo 24
2.3.6 Etat actuel du système de guidage 25
2.4 Moteur de propulsion 25
2.4.1 Descriptif 25
2.4.2 H-bridge 25
2.5 Coût des composants 27

3 Software
Première version des systèmes embarqués **29**
3.1 Software de l'Arduino 29
3.2 Software du Raspberry Pi 30
3.2.1 Sans fil 31
3.2.2 Guvcview 32
3.3 Python 32

4 Software
Deuxième version des système embarqués **35**
4.1 Java 36
4.1.1 Client/Serveur 36
4.1.2 UDP ou TCP 37
4.2 OpenCV 38
.1 Sketchbook 39
.1.1 Sketch exemple pour le moteur 39
.1.2 Sketch exemple pour le servo 41

.1.3 Code python sur le Raspberry Pi 43

Table des figures

Image de courverture 1

1.1 Arduino Uno R3 8

2.1 Le PCB. 14
2.2 Exemple de plaque RX2 15
2.3 Prototype du PCB 16
2.4 Schéma sommaire du prototype. 17
2.5 Strip- board 18
2.6 Schéma complet du PCB 19
2.7 Design final du PCB 20
2.8 Utilisation de la diode Zener 21
2.9 Signal modulé 24
2.10 H-Bridge 26

3.1 Environnement de programmation Arduino 30
3.2 *Dongle* Edimax EW7811Un 31
3.3 Interface graphique Python 34

4.1 Structure client-serveur 36
4.2 Connexion d'un nouveau client 37

Liste des tableaux

2.1 Caractéristiques du véhicule 13
2.2 Table de vérité, H-Bridge 27
2.3 Synthèse des composants clef, leur prix et le prix total. 28

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. Équipé d'un répétiteur, un groupe d'UGV peut couvrir une grande surface sans mettre en danger ses pilotes puisqu'ils restent à l'écart. Il peut servir en cas de catastrophe naturelle, néanmoins, notre modèle ne peut pas rouler sur les terrains trop accidentés, mais pour corriger ce problème, il suffit tout simplement d'implémenter notre système 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 est étroitement lié à la mécanique et il permet une meilleure compréhension du reste. Ce chapitre décrit principalement l'ordinateur et le micro-contrôleur [13] embarqués sur le drone. Ensuite, le chapitre mécanique et électronique qui détaille les modifications faites à la voiture radio-commandée et les recherches faites dans ce domaine. Finalement, les deux derniers chapitres sont consacrés aux différentes versions des logiciels que nous avons créées.

Chapitre 1

Hardware

POUR réaliser ce projet, nous avons dû faire des choix au niveau du hardware.Le hardware est tout ce qui touche la partie physique du drone, néanmoins, dans ce chapitre, nous ne traiterons pas de la mécanique, ce sujet sera abordé dans le chapitre 2. Notre choix s'est porté sur deux systèmes. Le premier, l'Arduino, est un micro-contrôleur qui permet de contrôler presque ce qu'on veut grâce à un langage de programmation proche du C.
Le second est le Raspberry Pi, qui est un ordinateur bon marché (trente-cinq francs) qui est récemment sorti sur le marché.

1.1 Arduino

L'Arduino [12] 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. Ce micro-contrôleur se programme avec un langage proche du C.

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 : 1.1). L'Arduino Uno est un micro-contrôleur.

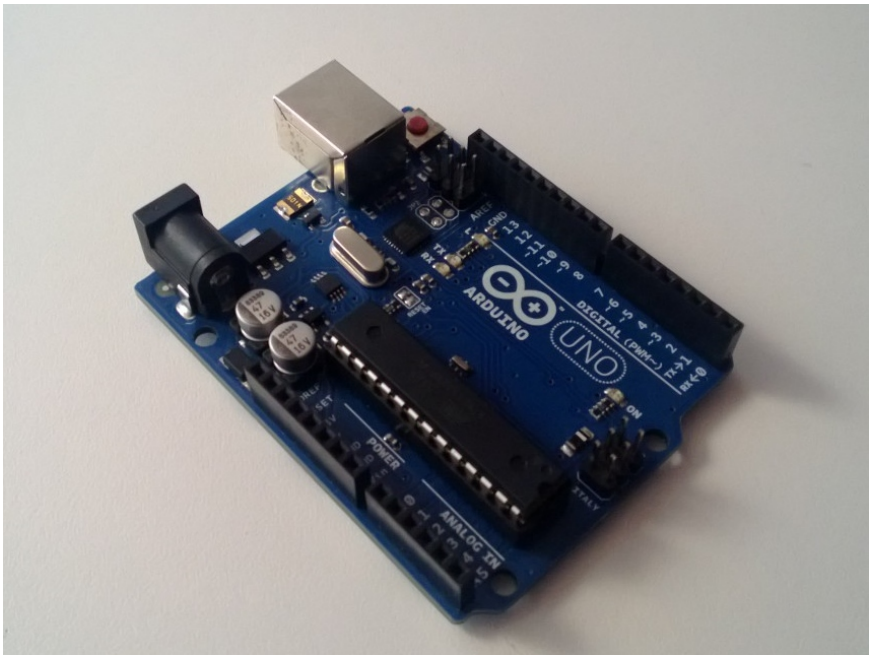

FIGURE 1.1 – Arduino Uno R3, On peut remarquer les pin qui se trouvent de part et d'autre de l'Arduino. On peut remarquer au centre le micro-contrôleur ATmega328 ainsi que le port USB tout en haut de l'image

1

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

1.1.2 Détails techniques de l'Arduino

Voici une liste des caractéristiques techniques de l'Arduino Uno R3 [12] :

1. 14 pin digitaux (signal haut ou bas) qu'on peut définit 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)
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 [14], 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 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 calculatoire, mais du point de vue rapport qualité-prix (35.- CHF).

1.2.1 Détails techniques du Raspberry Pi

Voici une liste des caractéristiques techniques du Raspberry Pi modèle B [14] :

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 FullHD 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 varié. En effet, il peut être utilisé pour la photographie, comme base centrale d'un système de surveillance.

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.

Chapitre 2

La mécanique et l'électronique

2.1 Caractéristiques du véhicule

NOUS nous basons sur un modèle réduit télécommandé de type 4x4. Le tableau suivant est un inventaire de ses caractéristiques dans son état actuel :

2.2 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 bas ou haut (0 ou 5V), à la différence de l'Arduino qui a la possibilité de moduler le signal (voir : 2.3.4). Cette caractéristique spécifique nous permet d'avoir

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

un large éventail de vitesses du moteur et d'angles de braquage différents.

2.2.1 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 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.

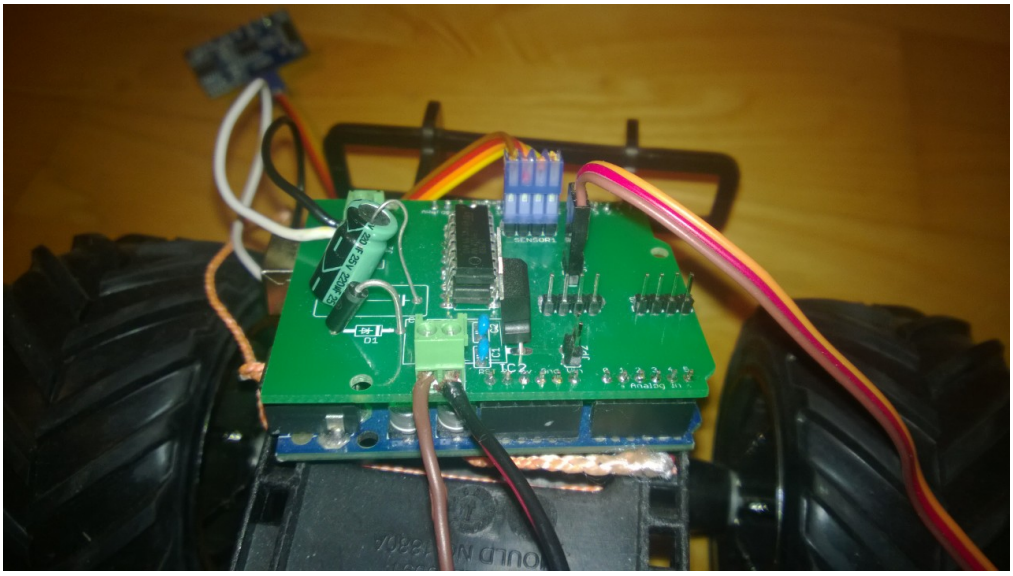

FIGURE 2.1 – Le PCB.

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 que ils seraient probablement inutiles lors de hauts sauts de voltage.
3. Un chip L293D (H-bridge (sec : 2.4.2)) contrôle le moteur de propulsion. Ici, deux chips sont empilés, donc en parallèle, pour pouvoir supporter la charge importante du moteur.

Tous ces composants, en plus des câbles et des connecteurs vers les actuateurs et batterie, sont soudés sur une *strip-board* semblable à celle dans la photo 2.5.C'est essentiellement 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.

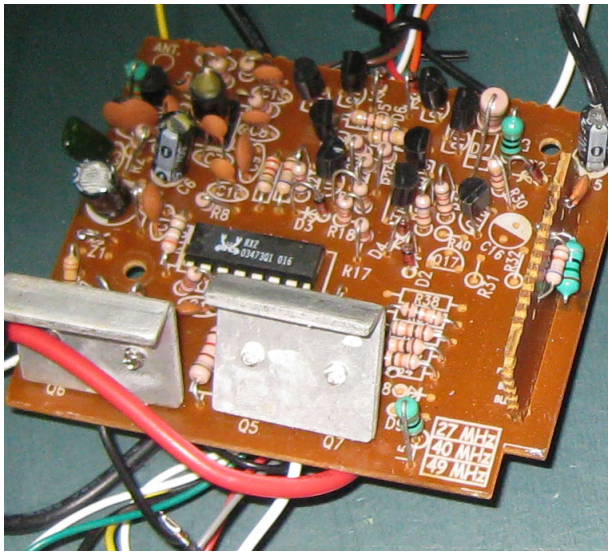

FIGURE 2.2 – Exemple de plaque similaire à ce qu'on a pu trouver dans le véhicule.

2.2.2 Schema électronique

Le schéma a été dessiné à l'aide d'un programme dédié gratuit (version light), Eagle. Les symbols 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

2.2.3 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

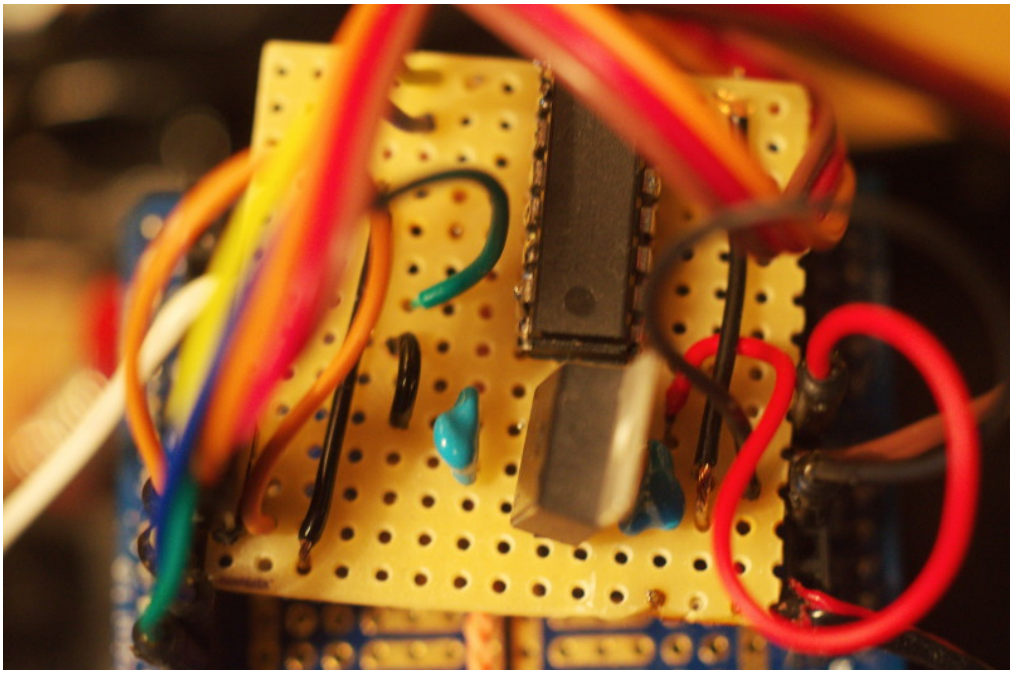

FIGURE 2.3 – Première plaque rassemblant différents composants électriques.

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

2.2.4 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.

(SCHÉMA/PHOTO qui montre où chaque composant est placé)

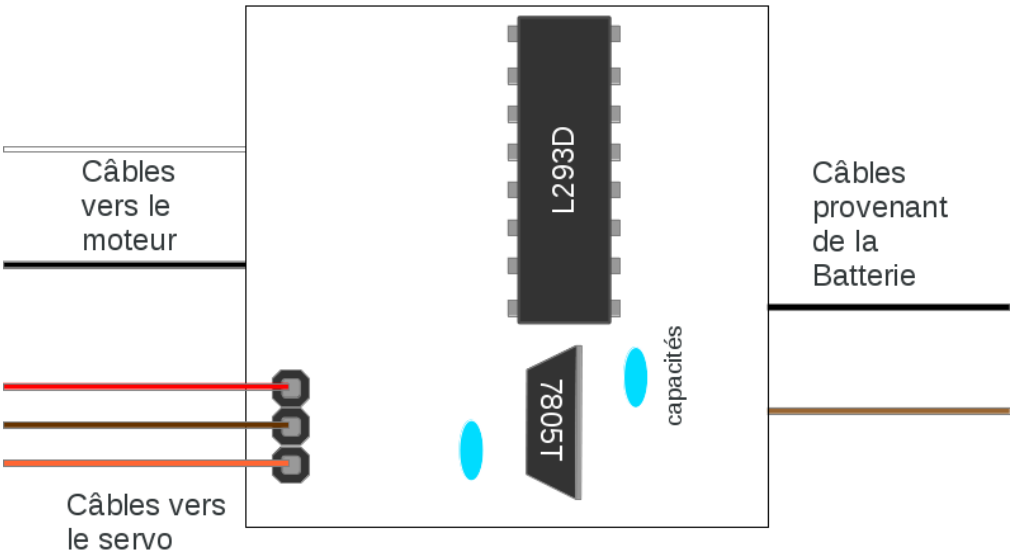

FIGURE 2.4 – Schéma sommaire du prototype.

1. L293D

   Le L293D est à un chip à seize pattes qui encapsule effectivement deux H-bridge comme illustré dans la section 2.4.2. Chaque moitié du pont peut fournir 0.6A de courant continu et 1.2A en pic [8]. 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 sont partagés si 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 [9], 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 pro-

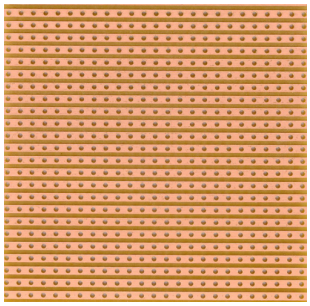

FIGURE 2.5 – Exemple de dessous de plaque de prototypage similaire à la notre. http ://www.verotl.com/images/images/veroboards/l-01-0033.jpg

totype.

2. 7805T

   Le 7805T est un régulateur 5V. Il approvisionne les composants fonctionnant à 5V en courant. Il peut prendre jusqu'à 35V de tension en entrée [6]. 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. 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 [6]. Mais, les tests jusqu'à ce jour montrent que 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 pour lisser le

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

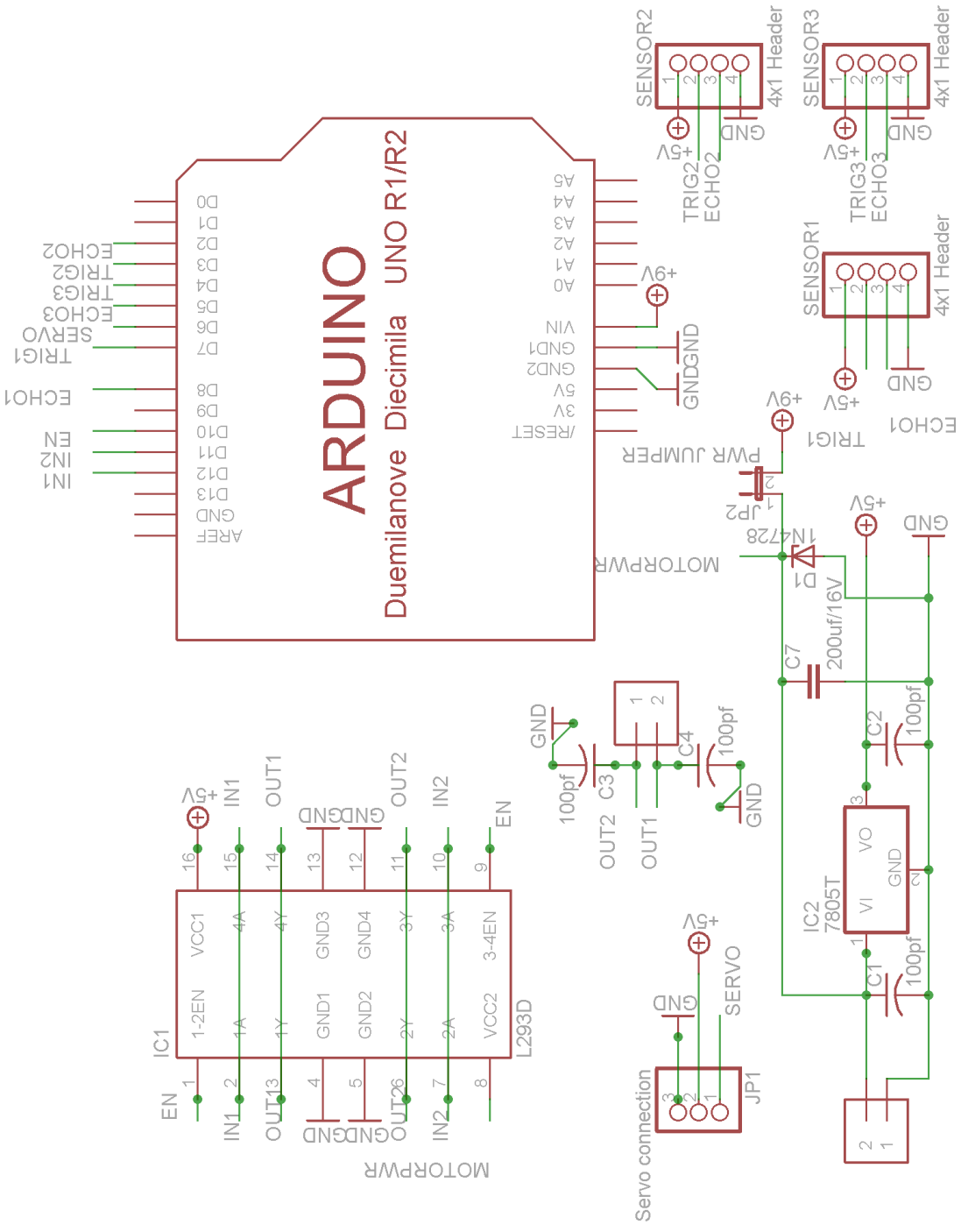

FIGURE 2.6 – Schéma complet du PCB

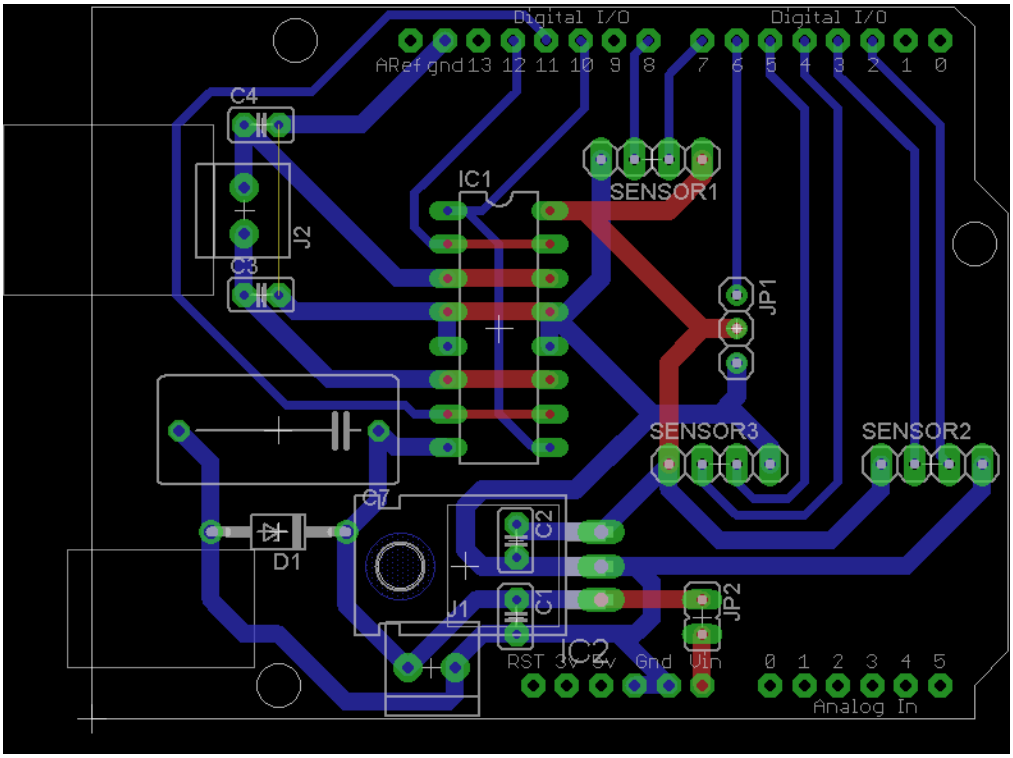

FIGURE 2.7 – Image du design final à l'ordinateur.

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

6. connecteurs senseurs

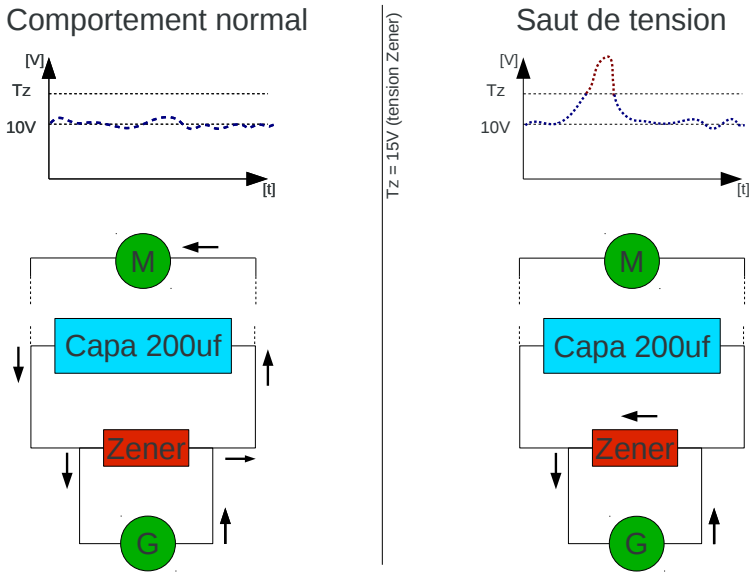

FIGURE 2.8 – Schéma d'un circuit ayant une diode Zener, expliquant comment cette diode permet d'éviter de griller des composants en cas de sauts de tensions.

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 ultrasonic 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.

2.2.5 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 bon 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, stockés dans un zip et envoyés avec la commande.

2.3 Système de guidage

2.3.1 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 (voir image) assure cette tâche. Il en résulte une identification de position très basique : gauche,devant ou droite.
(insérer images)
Sachant que nous avons retiré l'électronique de la voiture, il nous restait deux possibilités : utiliser le système de guidage rudimentaire, mais déjà

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

en place ou tout remplacer avec un servo-moteur plus 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.

2.3.2 Remplacement du Système de guidage

Une installation fiable d'un objet étranger dans un ensemble usiné tel 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 développée plus tard. [3]

2.3.3 Géométrie d'Ackermann

2.3.4 Contrôle du servo

Le contrôle d'un servo-moteur est une tâche qu'un micro-contrôleur tel que l'Arduino (sec : 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 un signal

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

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

FIGURE 2.9 – Schéma de la modulation du signal électrique [15]

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 a 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.

2.3.5 Programmation pour contrôler un servo

Le programme se trouvant en annexe (.1.2) est un exemple proposé dans la section "apprentissage" du site officiel d'Arduino [11]. Il utilise la librairie "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 cette modulation.

2.3.6 Etat actuel du système de guidage

Le servo-moteur n'est pas fonctionnel sur la plaque électronique que nous avons conçu nous-même. Selon nos diagnostiques, il s'agit d'une mauvaise soudure et donc le servo-moteur ne peut pas être alimenté en courant. Ce problème devrait se résoudre lorsque nous recevrons le PCB.

D'ailleurs, ce programme est très pratique pour tester le fonctionnement d'un servo-moteur.

2.4 Moteur de propulsion

2.4.1 Descriptif

Le moteur de propulsion, au contraire du servo, n'a pas été changé. Ses caractéristiques électriques (données que nous avon mesuré à l'aide d'un multimètre du gymnase) se trouvent à la section (2.1). Le moteur est muni d'une boite à vitesse ainsi qu'un différentiel. Nous avons estimé qu'il aurait été inutilement compliqué d'y apporter des modifications. La configuration déjà existante, à l'exception de l'éléctronique subvient tout à fait à nos besoins.

2.4.2 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.

Le schéma suivant pourra d'avantage éclairer le lecteur.

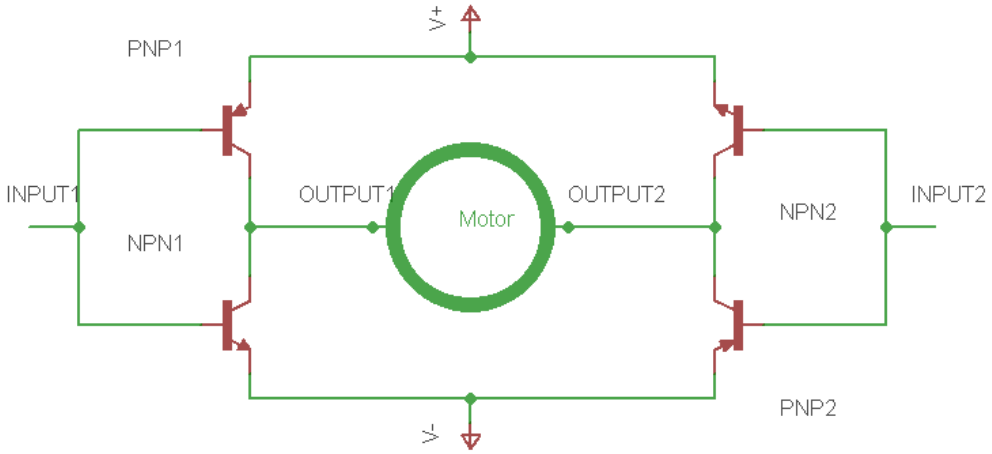

FIGURE 2.10 – Schéma d'un H-bridge

Explications

Pour fabriquer un H-bridge, il faut utiliser entre autres des transistor NPN [4] et PNP [5]. 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 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 les *INPUT* à des transistors diagonalement opposés[10].

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

4. Négatif-Positif-Négatif
5. Positif-Négatif-Positif

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 : 2.10) 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.

Table de Vérité

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

2.5 Coût des composants

Le tableau 2.3 contient les éléments clefs que nous avons utilisé 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)	1	3.20
L293D	1	3.00
Câble micro USB	1	1.40
total	10	179.60

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

Chapitre 3

Software
Première version des systèmes embarqués

3.1 Software de l'Arduino

POUR programmer l'Arduino, nous utilisons le logiciel dédié, *Arduino* (fig : 3.1), C'est un environnement *open source* [1] qui fonctionne sur Windows, Linux et Mac OS X. Il a été écrit en Java et est basé sur le logiciel Processing. 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 car-

1. Terme de *Open source*[7] signifie littéralement "Libre de droits", ceci indique donc qu'un logiciel *Open source* permet à n'importe qui d'avoir accès au code source dans son intégralité. "L'Open Source est un monde de partages où chacun peut apporter une pierre à l'édifice pour améliorer encore et toujours les solutions retenues"[3]

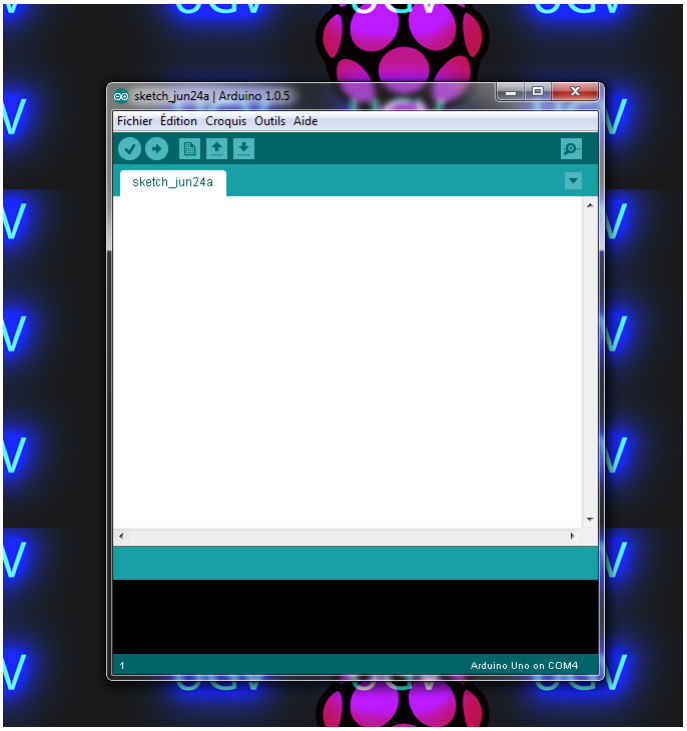

FIGURE 3.1 – Environnement Arduino, C'est la fenêtre qui nous permet d'écrire un code dans un pseudo C, de le compiler et le téléverser sur l'Arduino. Ce programme a un sorte de petit déboggeur qui est très basique, mais très souvent suffisant.

actères à l'Arduino par le biai 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 afin de rendre le drone fonctionnel. Dans un premier temps, nous allons parler de notre première version, fonctionnant sous python avec comme logiciel de vidéo Guvcview (sec :3.2.2). Dans un second temps, nous

allons aborder la deuxième version du drone, plus complexe et fonctionnant grâce au langage de programmation Java

3.2.1 Sans fil

Afin que le Raspberry Pi puisse se connecter sur le réseau sans fil, nous utilisons un *dongle* (fig :3.2) Wi-Fi [2]. Cet accessoire ne nécessitant pas de *driver* [3],il est donc directement reconnu et peut rapidement être configurer par le Raspberry Pi.

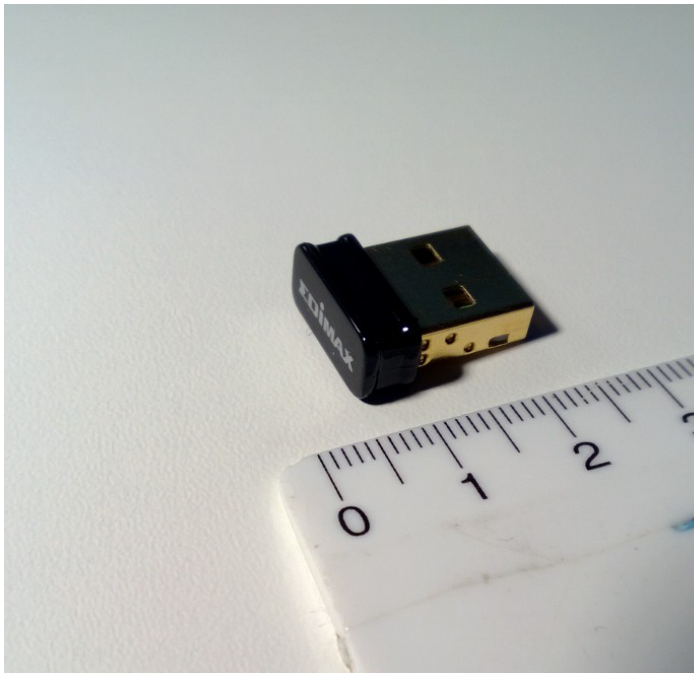

FIGURE 3.2 – *Dongle* Edimax EW7811Un. Connecté par USB à la Framboise, il permet au Raspberry Pi de se connecter via Wi-Fi à une borne

2. Le modèle utilisé est le suivant : EW7811Un fabriqué par Edimax
3. Un driver est un petit logiciel permettant d'exploiter le Hardware

Partage sécurisé

Le partage sécurisé entre deux ordinateurs, que nous allons appeler ici *ssh* pour *Secure Shell* 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 affiché par le Raspberry Pi. Il y a deux inconvénients majeurs à ces solutions, la première, c'est qu'à moins d'avoir un *Shell Script*, il faut établir la connexion et lancer les programmes manuellement., la deuxième, c'est qu'ils ne sont utilisable que sous les système UNIX, c'est à dire que sous cette version, notre drone ne toucherais pas le grand public, mais seulement les professionnels qui auraient plutôt tendance à choisir des modèles plus chers, mais aussi plus performants.

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 dix-huit images par secondes lors d'une connexion *ssh* avec une résolution d'environ 600x380 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.

3.3 Python

Outre les programmes mentionnés ci-dessus, nous avons développé notre propre code python (annexe : .1.3) qui permet de contrôler le véhicule. Le code est Visuellement, l'interface graphique est très simple (fig : 3.3). 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. Il affiche entre autre les messages d'erreur ainsi que des informations à propos de la vitesse maximale ou du braquage maximal. Ce programme ne prend pas en compte l'utilisation de capteur, tel que capteur de distance. Il permet simplement de diriger le véhicule. Une zone de texte non éditable par l'utilisateur permet d'afficher des messages, avertissant, 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.

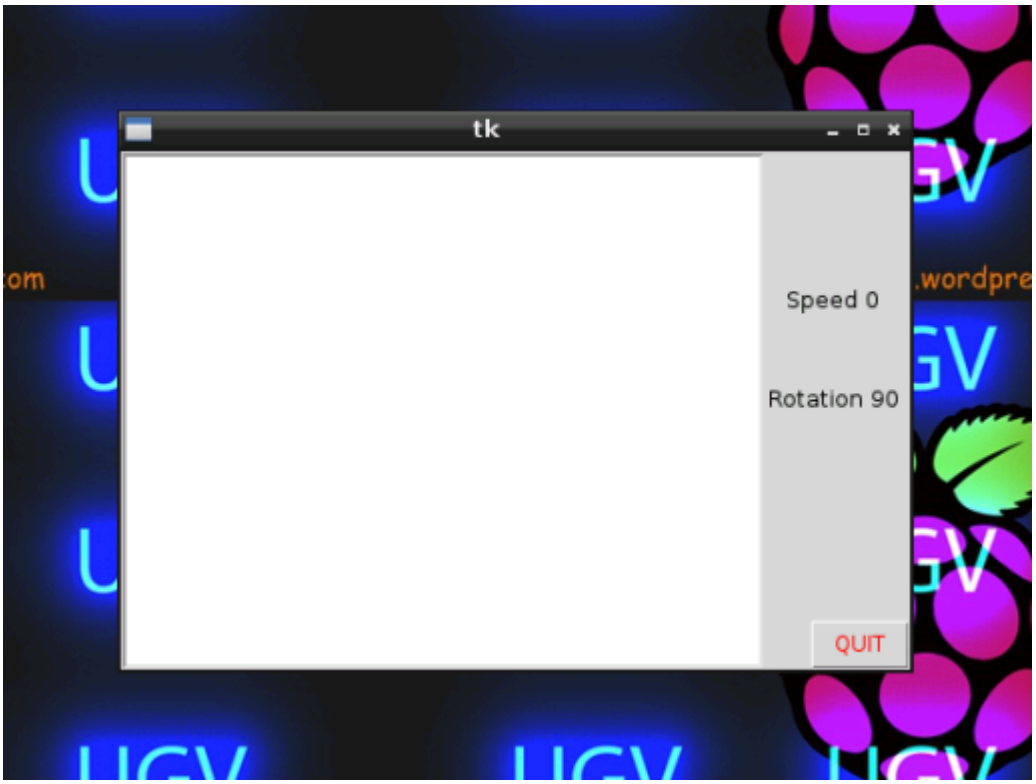

FIGURE 3.3 – Fenêtre X du code python qui permet la lecture du clavier. Cette fenêtre permet de contrôler le drone.

Chapitre 4

Software
Deuxième version des système 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 avec le langage Java, nous permet de faire nous même la connexion entre l'ordinateur et le Raspberry Pi. Nous avons choisis ce langage pour plusieurs raisons. Tout d'abord, nous avions des notions en Java, ce qui n'est pas négligeable. 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 employée 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 OpenCV.

4.1 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 rencontré pendant l'écriture du programme. Nous allons expliquer comment nous avons structuré le système, selon quelles lois et discuter des avantages et des inconvénients.

4.1.1 Client/Serveur

Le model client-serveur est composé de deux partis : 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 auxquels il répond. Il doit généralement être capable de gérer plusieurs clients à la fois.

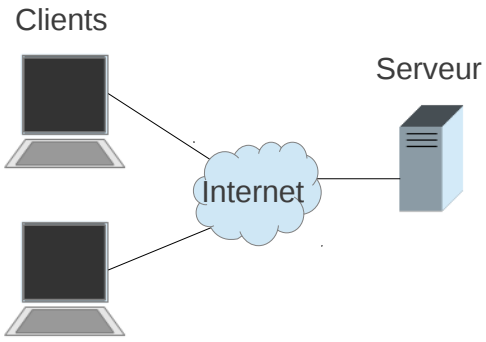

FIGURE 4.1 – Structure client-serveur.

Nous avons établi que la voiture occupe le rôle de serveur qui est effectivement 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).

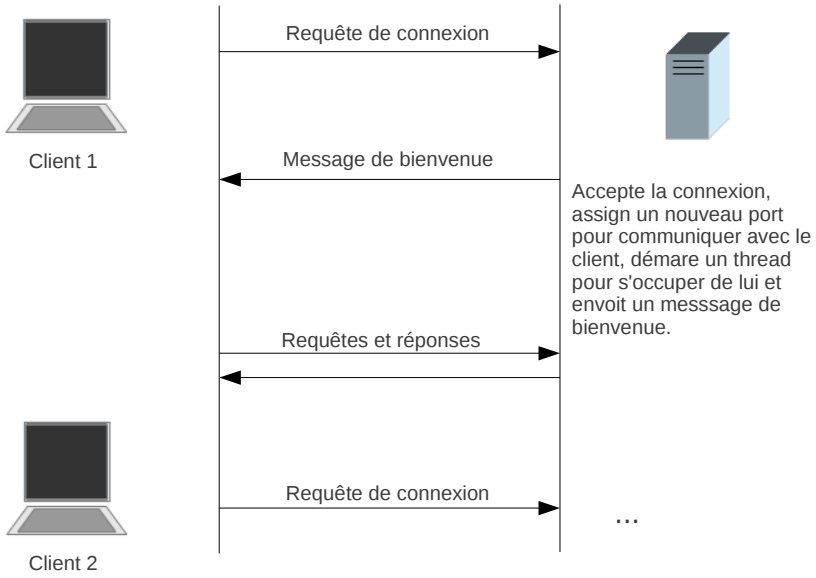

FIGURE 4.2 – Schéma de la procédure suivie à chaque nouvelle connexion.

4.1.2 UDP ou TCP

Avant de parler de notre choix, expliquons un peu que sont les protocoles UDP et TCP et leurs différences. UDP et TCP sont deux protocoles de connexion réseau qui permettent d'envoyer des données (*Packets*) d'un ordinateur à un autre. Un des deux ordinateurs sera considéré comme étant

un serveur tandis que l'autre sera considéré comme un client. La différence entre le client et le serveur à l'air assez simple à première vue, mais est plus complexe lorsque nous devons l'appliquer. Mais revenons tout d'abord aux différences.

Le protocole UDP [1] est un protocole qui n'est pas orienté connexion. En effet, dans notre cas, l'ordinateur envoie des paquets au Raspberry Pi sans le prévenir, ce dernier ne va pas non plus confirmer la réception des paquets. C'est un flux unidirectionnel qui est dû à l'encapsulation des données. En effet, les paquets qui transitent via un protocole UDP ne contiennent que l'adresse IP et pas d'autres informations concernant l'émetteur. Ce qui voudrait dire que le Raspberry Pi ne s'est pas s'il est toujours à portée de l'ordinateur.

Le protocole TCP [2] est l'opposé de l'UDP, il est orienté connexion. Lorsque l'ordinateur envoie des données au Raspberry Pi, ce dernier est informé de l'arrivée des dites données. Il donne aussi une sorte de reçu qui confirme la réception des packets. S'il y a des fichiers corrompus, l'ordinateur va renvoyer les fichiers manquants. On peut, par analogie, comparer ce protocole comme la communication directe téléphonique.

Dans notre cas, nous choix s'est porté sur le protocole TCP. En effet, il est plus pratique de savoir que l'ordinateur n'envoie plus de données et qu'il faille stopper le véhicule car la connexion est simplement perdue, tandis qu'avec le protocole UDP, nous ne savons pas s'il y a une perte de connexion, puisqu'il n'y a pas eu de connexion.

4.2 OpenCV

1. *User Datagram Protocol*, soit protocole des datagrammes d'utilisateurs
2. *Transmission Control Protocol*, soit protocole de contrôle des transmissions

.1 Sketchbook

.1.1 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

  for(i = 0; i < 200; i++){
    forward(A, B, E, i);
    delay(50);
  }
  for(i = 200; i > 0; i--){
    forward(A, B, E, i);
    delay(50);
```

```
  }
  for(i = 0; i < 200; i++){
    backward(A, B, E, i);
    delay(50);
  }
  for(i = 200; i > 0; i--){
    backward(A, B, E, i);
    delay(50);
  }
}

//methode globale pour controler le moteur, fait appel aux
//methodes halt, backward et forward.
void motor(int speedo){
  if (abs(speedo) > 255){
    halt(A,B);
    break;
  }
  else if (speedo > 0){
   forward(A, B, E, speedo);
  }
  else if (speedo < 0){
    backward(A, B, E, speedo);
  }
  else if (speedo == 0){
    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);
}
```

.1.2 Sketch exemple pour le servo

[11]

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


#include <Servo.h>
```

```
Servo myservo;  // create servo object to control a servo
                // a maximum of eight servo objects can be created

int pos = 0;    // variable to store the servo position

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}



void loop()
{
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degre
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position sto
                                     // in variable 'pos'
    delay(15);                       // waits 15ms for the servo
                                     // to reach the position
  }
  for(pos = 180; pos>=1; pos-=1)     // goes from 180 degrees to 0 degre
  {
    myservo.write(pos);
    delay(15);
  }
}
```

Commentaires

On commence par inclure la classe Servo, puis on crée un objet *Servo*. Dans la fonction *setup* du programme, on lie l'objet *myservo* au pin neuf de

l'arduino. Ensuite, dans la fonction *loop*, on fait varier la position du servo grâce à la méthode *write*. Un délai (le programme s'arrête en ce point) de 15ms pour permettre au servo-moteur d'atteindre la position demandée. Etant donnée que cette opération est ittérée plusieurs fois au moyen d'une boucle *for*, on pourra voir le servo décrire un mouvement de balayage.

.1.3 Code python sur le Raspberry Pi

```
#Import Tkinter module
from Tkinter import *

# Xwindow libs
import Xlib.display as display
import Xlib.X as X

#For clean exit
import atexit


#Establish serial communication
import serial
#create serial object for serial comunication over port 'ttyACM0' or 'tty
#at 9600 bauds
try:
    print 'trying /dev/ttyACM0/'
    ser = serial.Serial("/dev/ttyACM0", 9600)
except serial.SerialException:
    print '/dev/ttyACM0/ not found'
    pass
try:
```

```
        print 'trying /dev/ttyACM1/'
        ser = serial.Serial("/dev/ttyACM1", 9600)
    except serial.SerialException:
        print '/dev/ttyACM1/ not found'
        pass



#Put autorepeat in normal mode at exit
@atexit.register
def autorepeat():
    d=display.Display()
    d.change_keyboard_control(auto_repeat_mode=X.AutoRepeatModeOn)
    x=d.get_keyboard_control()



#This class contains all elements in the gui, manages the interactions u
#the user.
class UgvGui:


#Variables are stored in class attributes
    speed = 0
    rotation = 90

    def __init__(self, master):
```

```
#Initiate grafical interface
self.master = master
#Make and place buttons, labels and text zones
self.speed_label = Label(master, text="Speed %i " % (self.speed)
self.speed_label.grid(column= 1, sticky= S)

self.rotation_label = Label(master, text="Rotation %i "
                            % (self.rotation))
self.rotation_label.grid(column= 1)

self.exit = Button(master, text= "QUIT", fg="red", command=maste
self.exit.grid(column= 1, sticky= SE)

self.messages = Text(master, height= 20, width= 50,
                     background= "white", border= 2)
self.messages.grid(column= 0, row= 0, rowspan= 3)


#Initiate serial communication

#Bind key events to class functions
master.bind("<Escape>", self.escape)
master.bind("<Shift-Up>", self.shift_up)
master.bind("<Shift-Down>", self.shift_down)
master.bind("<Shift-Right>", self.shift_right)
master.bind("<Shift-Left>", self.shift_left)
master.bind('<KeyPress-Up>', self.upPress)
master.bind('<KeyPress-Down>', self.downPress)
master.bind('<KeyPress-Right>', self.rightPress)
```

```
        master.bind('<KeyPress-Left>', self.leftPress)
        master.bind('<KeyRelease-Up>', self.upRelease)
        master.bind('<KeyRelease-Down>', self.downRelease)
        master.bind('<KeyRelease-Right>', self.rightRelease)
        master.bind('<KeyRelease-Left>', self.leftRelease)

    #Define on key-stroke-event functions
    def escape(self, event):
        print "Quit: UgvGui"
        #Quits the application
        self.master.destroy()

    def shift_up(self, event):
        if self.speed < 200:
            #set new speed if in range
            self.speed = self.speed + 20
            self.speed_label.config(text = "Speed %i" % (self.speed))
            print "SPEED: new speed is %i" % (self.speed)
        else:
            #if the the new speed should be out of range warn the user
            print "SPEED: warning! max speed is reached"
            self.messages.insert(END, "SPEED: warning! max speed is reac

    def shift_down(self, event):
        if self.speed > 0:
            self.speed = self.speed - 20
            self.speed_label.config(text = "Speed %i" % (self.speed))
            print "SPEED: new speed is %i" % (self.speed)
        else:
```

```
            print "SPEED: warning! minimal speed is reached"
            self.messages.insert(END, "SPEED: warning! minimal speed is

    def shift_right(self, event):
        if self.rotation < 140:
            self.rotation = self.rotation + 10
            self.rotation_label.config(text = "Rotation %i" % (self.rota
            print "ROTATION: new angle is %i" % (self.rotation)
        else:
            print "ROTATION: warning! max angle is reached"
            self.messages.insert(INSERT, "ROTATION: warning! max angle i

    def shift_left(self, event):
        if self.rotation > 50:
            self.rotation = self.rotation - 10
            self.rotation_label.config(text = "Rotation %i" % (self.rota
            print "ROTATION: new angle is %i" % (self.rotation)
        else:
            print "ROTATION: warning! minimal angle is reached"
            self.messages.insert(INSERT, "ROTATION: warning! minimal ang



    def upPress(self, event):
        print 'f: %i' % (self.speed)
        ser.write('f')

    def downPress(self, event):
        print 'b: %i' % (self.speed)
        ser.write('b')
```

```
    def rightPress(self, event):
        print 'r: %i' % (self.rotation)
        ser.write('r')

    def leftPress(self, event):
        print 'l: %i' % (self.rotation)
        ser.write('l')

    def upRelease(self, event):
        print 'stop'
        ser.write('s')

    def downRelease(self, event):
        print 'stop'
        ser.write('s')

    def rightRelease(self, event):
        print 'center'
        ser.write('c')

    def leftRelease(self, event):
        print 'center'
        ser.write('c')


#set auto repeat mode to off
d=display.Display()
d.change_keyboard_control(auto_repeat_mode=X.AutoRepeatModeOff)
```

```
x=d.get_keyboard_control()

#Make root Tkinter object
root = Tk()

#launch gui
ugv = UgvGui(root)



#Update the message box to always show the last message
def update_messages():
    ugv.messages.see(END)
    root.after(1000, update_messages)



root.after(500, update_messages)

#initiate gui mainloop
root.mainloop()
```

Bibliographie

[1] Atef Akremi. Fastest way to hack rc car h-bridge, juin 2012. http://www.instructables.com/files/deriv/F46/2LOX/H3XME4X0/F462LOXH3XME4X0.LARGE.jpg.

[2] Cytron Technologies Sdn. Bhd. User's manual, ?? https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit.

[3] Metycea blog. Définition : Qu'est-ce que l'open source ? (1/4), Septembre 2008. http://blog.metycea.com/actus-internet/definition-qu-est-ce-que-l-open-source-1-4.html.

[4] Talking Electronics. Transmitters, Juillet 2011. http://www.talkingelectronics.com/projects/27MHz%20Transmitters/imagesP2/4ChRx2B.jpg.

[5] Jon from jbproject. Wifi robot, août 2008. http://www.jbprojects.net/projects/wifirobot/rx2.jpg.

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

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

[8] Texas Instrument. L293, l293d quadruple half-h drivers, Juin 2002.

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

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

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

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

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

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

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

[16] 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.