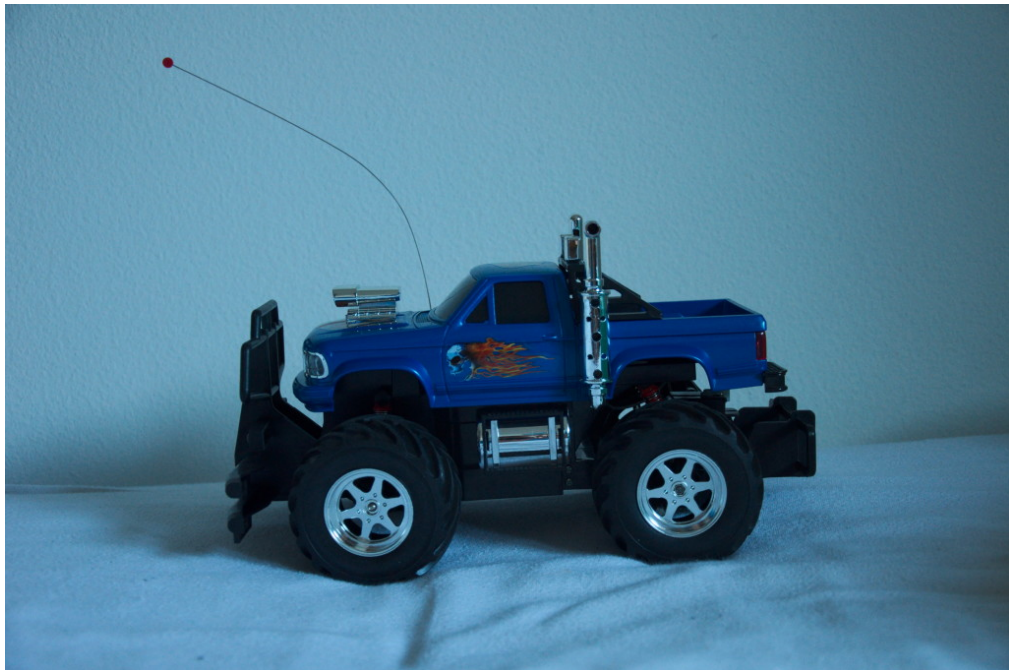


# UGV

## (Unmanned Ground Vehicle)

Sven Borden                      Eric Brunner  
Travail de maturité              Gymnase de Morges

2 septembre 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 quadcoptères (hélicoptères à quatre hélices). Plutôt que de se lancer dans un projet dont nous savons que nous n'allons pas réussir à terminer dans les délais, nous nous sommes plutôt focalisé sur les véhicules terrestres ce qui nous a permis d'aller beaucoup plus loin que le quadcopter.

## Remerciements

Ce projet n'aurait pu aboutir sans l'aide de nombreuses personnes. Voici l'occasion de les remercier : Mr. Denis Rochat et Mr. Phillippe Rochat pour leur disponibilité, leurs renseignements ainsi que les prêts matériels. Mr. Julien Dominski pour son aide précieuse dans le codage. Mr. Jean Rossier, Karl Kangur, Michaël Bolay et Mr. Marco Pagnamenta pour leur conseils techniques, surtout à propos des plaques 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. Mme Pauline Pidoux pour nous avoir aidé lors de la rédaction de ce travail et nous tenions aussi à remercier Stefano Varricchio, du Laboratoire LIS pour ses informations très utiles.

## Résumé

Parti dans l'idée de concevoir un objet déjà existant mais à moindre prix, nous avons choisi des technologies récemment sorties, ce qui fait que nous étions dans les premiers à évoluer dans ce milieu. Néanmoins, un support de développeur a toujours été là pour nous débloquer en cas de pépin. C'est entre autre grâce à eux que nous sommes parvenu à atteindre l'objectif souhaité. 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. 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 existant sont soit trop cher, soit peu performants. Par exemple le Beewi WiFi Camera Buggy BWZ200-A1, qui coûte septante-neuf francs est un bon jouet, mais dans notre travail de maturité, son autonomie est insuffisante (une dizaine de minute tout au plus) et sa portée ne dépasse guère les 25 mètres, ce qui rend la caméra inutile. L'objectif de ce projet n'est pas d'avoir simplement un prototype, mais d'avoir un produit fini qui serait en phase finale de commercialisation. En effet, si nous parvenons avoir un ensemble possédant un véhicule tout terrain, une interface claire et un guidage simple et efficace, nous aurons 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*<sup>1</sup>. Ce concours réunit presque tout les domaines, pourvu que certains critères soient respectés. Tous les concurrents continuent un projets

---

1. [site à insérer]

qu'ils avait fait 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.

# Table des matières

Avant-propos . . . . .	1
Remerciements . . . . .	2
Résumé . . . . .	1
Introduction . . . . .	5
<b>1 Hardware</b>	<b>6</b>
1.1 Choix du hardware . . . . .	6
1.1.1 Arduino . . . . .	6
1.1.2 Raspberry Pi . . . . .	8
1.2 PCB . . . . .	10
1.2.1 Plaque personnelle . . . . .	10
1.2.2 Schéma . . . . .	11
1.2.3 Design . . . . .	11
1.2.4 Composants . . . . .	11
1.2.5 Commande et conception . . . . .	12
1.3 Coût des composants . . . . .	12
<b>2 La mécanique et l'électronique</b>	<b>15</b>
2.1 Description du véhicule . . . . .	15
2.2 Système de guidage . . . . .	15
2.2.1 Système directionnel initial . . . . .	15
2.2.2 Remplacement du Système de guidage . . . . .	17

---

2.2.3	Géométrie d'Ackermann . . . . .	17
2.2.4	Contrôle du servo . . . . .	17
2.2.5	Programmation pour contrôler un servo . . . . .	18
2.2.6	Etat actuel du système de guidage . . . . .	18
2.3	Moteur de propulsion . . . . .	19
2.3.1	Descriptif . . . . .	19
2.3.2	H-bridge . . . . .	19
<b>3</b>	<b>Software</b>	
	<b>Première version des systèmes embarqués</b>	<b>22</b>
3.1	Software de l'Arduino . . . . .	22
3.2	Software du Raspberry Pi . . . . .	23
3.2.1	Sans fil . . . . .	24
3.2.2	Guvview . . . . .	25
3.3	Python . . . . .	25
<b>4</b>	<b>Software</b>	
	<b>Deuxième version des système embarqués</b>	<b>27</b>
4.1	UDP ou TCP . . . . .	28
4.2	Java . . . . .	29
4.3	OpenCV . . . . .	29
.1	Sketchbook . . . . .	30
.1.1	Sketch exemple pour le moteur . . . . .	30
.1.2	Sketch exemple pour le servo . . . . .	32
.1.3	Code python sur le Raspberry Pi . . . . .	34

# Table des figures

Image de courverture . . . . .	1
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 microcontrôleur ATmega328 ainsi que le port USB tout en haut de l'image . . . . .	7
1.2 Première plaque rassemblant différents composants électriques.	10
1.3 Schéma sommaire de la plaque. . . . .	11
1.4 Exemple de dessous de plaque de prototypage similaire à la notre. <a href="http://www.verotl.com/images/images/veroboards/l-01-0033.jpg">http ://www.verotl.com/images/images/veroboards/l-01-0033.jpg</a> . . . . .	14
2.1 Schéma de la modulation du signal électrique [2] . . . . .	18
2.2 Schéma d'un H-bridge . . . . .	20
3.1 Environnement Arduino . . . . .	23
3.2 <i>Dongle</i> Edimax EW7811Un. Connecté par USB à la Framboise, il permet au Raspberry Pi de se connecter via Wi-Fi à une borne . . . . .	24
3.3 Fenêtre X du code python qui permet la lecture du clavier. Cette fenêtre permet de contrôler le drone. . . . .	26



# Liste des tableaux

2.1	Caractéristiques du véhicule . . . . .	16
2.2	Table de vérité accompagnant le schéma du H-bridge (fig.(2.2))	21

## 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 incluant une caméra. Ce types d'engins se nomment *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. Equipé d'un répéteur, un groupe d'UGV peut couvrir une grande surface sans mettre en danger son pilote puisqu'il reste 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. L'interface est aussi un des buts, tout comme la simplicité d'utilisation. 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, c'est pourquoi le but de ce projet est d'avoir un programme qu'il suffit d'exécuter pour pouvoir utiliser le drone.

# Chapitre 1

## Hardware

### 1.1 Choix du hardware

**P**OUR 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 microcontrôleur 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.

#### 1.1.1 Arduino

L'Arduino [5] est un microcontrôleur *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 microcontrôleur se programme avec un langage proche du C.

### Choix du type d'Arduino

Pour ce projet, nous avons choisi l'Arduino Uno (fig :1.1), c'est l'Arduino de base. En effet, Arduino possède de nombreux produits afin de subvenir aux différents besoin. Nous avons hésité à prendre l'Arduino Mega, mais les avantages qu'il offre ne sont pas très utiles pour notre projet. En effet, l'Arduino Mega est similaire à l'Arduino Uno , sauf qu'il est possède des ressources de calculs plus puissantes et il est aussi deux fois plus grand. Puisque nous n'avons pas besoin d'une grande puissance de calcul, nous avons choisi l'Arduino Uno. L'Arduino Uno est un microcontrôleur qui ressemble à ceci.

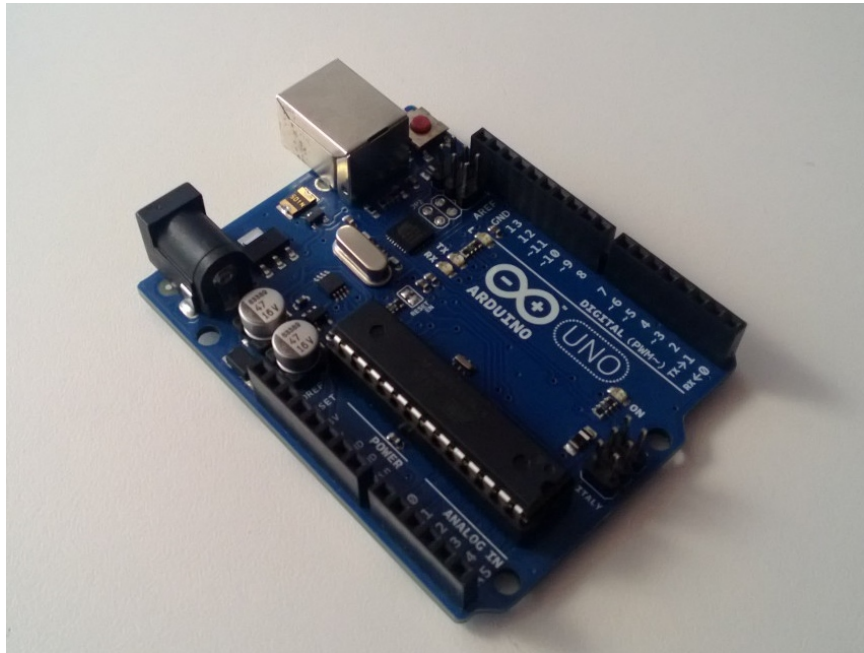


FIGURE 1.1 – Arduino Uno R3, On peut remarquer les pin qui se trouvent de part et d'autre de l'Arduino. On peut remarquer au centre le microcontrôleur ATmega328 ainsi que le port USB tout en haut de l'image

<sup>1</sup> L'Arduino Uno possède les caractéristiques suivantes :

1. 14 pin digitaux (signal haut ou bas) qu'on peut définir en INPUT ou en OUTPUT dont 6 d'entre eux peuvent moduler le signal lorsqu'ils sont utilisés en OUTPUT
2. 6 pin analogiques en INPUT
3. Connection 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. Microcontrôleur ATmega328

Bien qu'à première vue on pourrait se dire qu'il n'est pas très performant. Il faut se rappeler qu'il ne doit pas faire fonctionner un système d'exploitation, mais qu'il doit simplement traiter des mesures et envoyer des signaux digitaux ou analogiques. 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 microcontrôleur.

### 1.1.2 Raspberry Pi

Le Raspberry Pi[1], 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

---

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

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. En effet, pour trente-cinq francs, il a les caractéristiques suivantes :

1. poid de 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. criture 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)

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.

## Choix de l'OS

Une quinzaine de systèmes d'exploitations fonctionnant sur le Raspberry Pi existent. Parmi les plus connus, il y a Android, Arch Linux ARM et Debian/Raspbian. Notre choix a 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 et il est donc continuellement développé par la communauté du Raspberry Pi. Cet OS étant basé sur un environnement Linux, cela offre un grand nombre de liberté afin de travailler dessus. Raspbian est aussi gratuit, ce qui est à prendre en compte puisque nous essayons de réduire les coûts.

## 1.2 PCB

### 1.2.1 Plaque personnelle

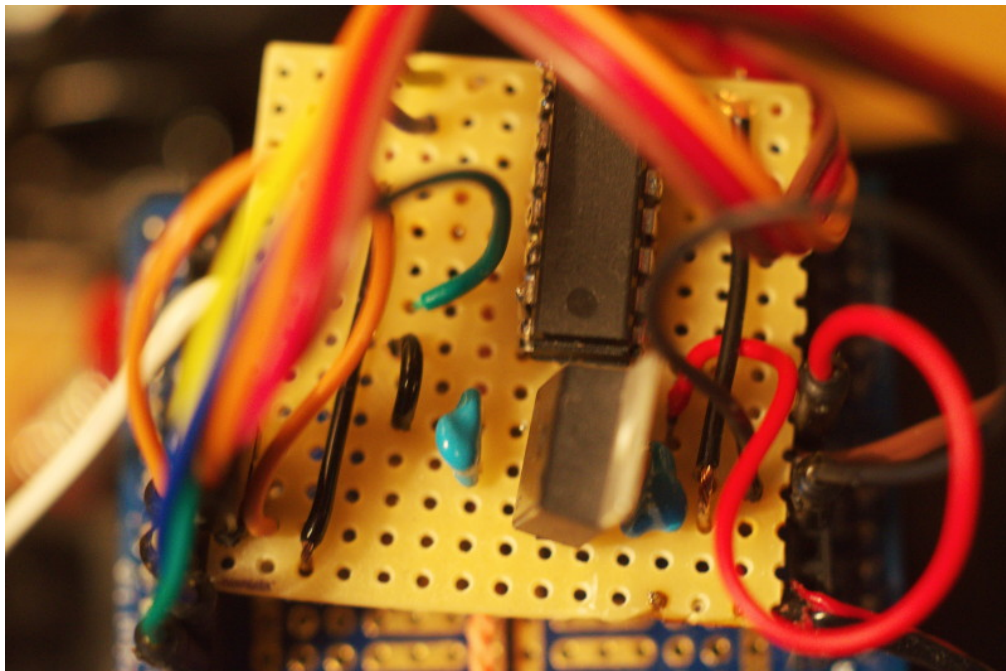


FIGURE 1.2 – Première plaque rassemblant différents composants électriques.

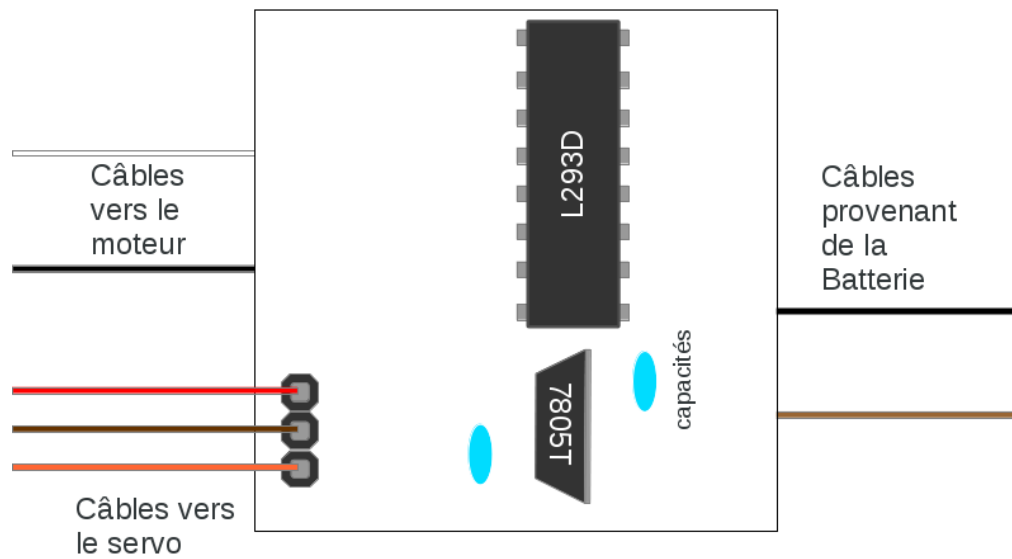


FIGURE 1.3 – Schéma sommaire de la plaque.

### 1.2.2 Schéma

### 1.2.3 Design

### 1.2.4 Composants

1. L293D
2. 7805T
3. capacités
4. diode zener
5. connecteurs senseurs
6. connecteur servo
7. prises block



### 1.2.5 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.

## 1.3 Coût des composants

Voici un tableaux contenant tout ce dont nous avons utiliser 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

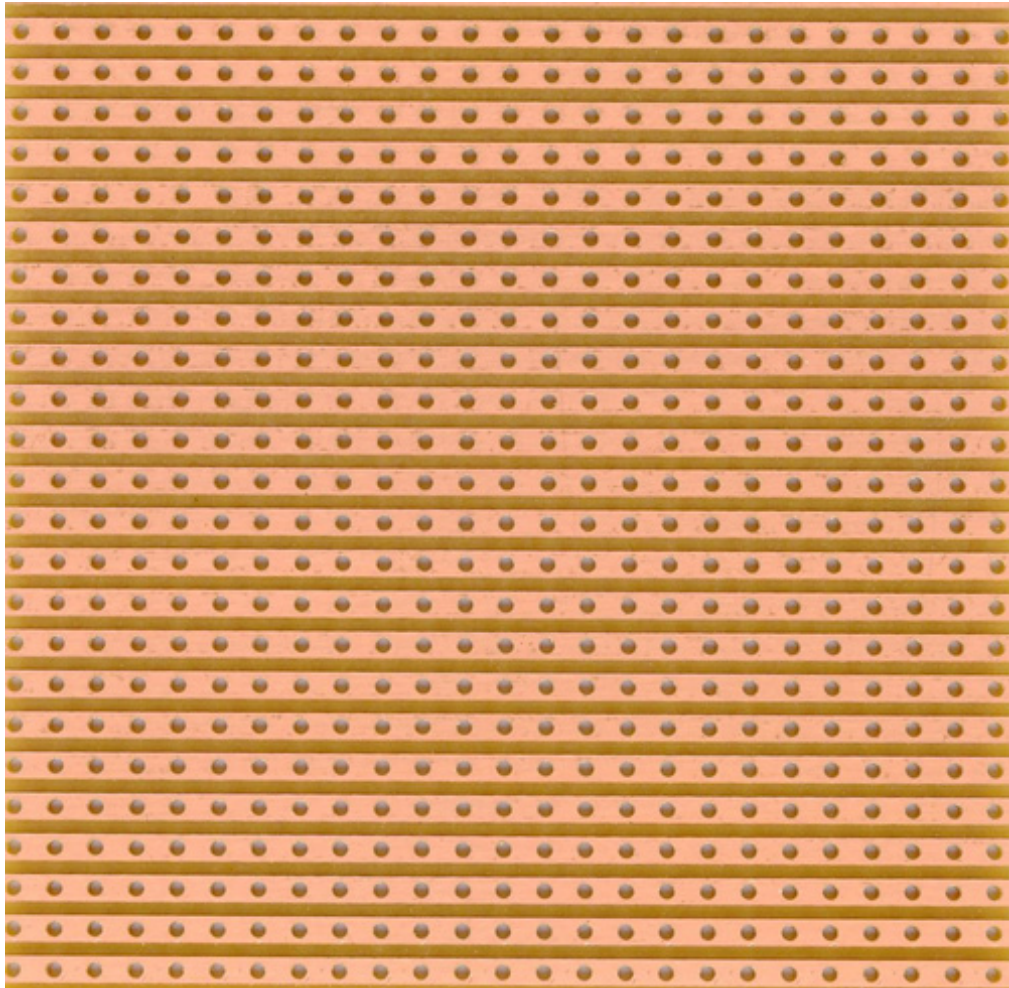


FIGURE 1.4 – Exemple de dessous de plaque de prototypage similaire à la notre.

[http ://www.verotl.com/images/images/veroboards/l-01-0033.jpg](http://www.verotl.com/images/images/veroboards/l-01-0033.jpg)

## Chapitre 2

# La mécanique et l'électronique

### 2.1 Description du véhicule

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

### 2.2 Système de guidage

#### 2.2.1 Système directionnel initial

Dans son état initial, le système de guidage pouvait se comparer à un servo très rudimentaire. Il en comptait toutes les caractéristiques, mais en un état simplifié, en particulier le système de positionnement. Dans un servo classique, il s'agit d'un potentiomètre (résistance variable) qui permet de savoir en tout moment la position de la corne<sup>1</sup>. 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

---

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

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 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

deux possibilités : utiliser le système de guidage rudimentaire, mais déjà en place ou tout remplacer avec un servo 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. Nous avons acheté un puissant servo 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 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.

Ce montage permet de retirer le servo en cas de besoin, donc de pouvoir le remplacer. En effet, la plaque supérieure est fixée au moyen de vis à bois. Le servo est accompagné, dans son logement, d'un morceau de gomme adhésive (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 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.<sup>2</sup>

### 2.2.3 Géométrie d'Ackermann

### 2.2.4 Contrôle du servo

Le contrôle d'un servo est une tâche qu'un microcontrôleur tel que l'arduino 1.1.1 effectue avec aisance. On peut indiquer à un servo de se rendre vers un de ses 180° de liberté en lui envoyant un signal électrique dit modulé. Ce signal est modulé d'une manière compréhensible pour le servo. L'illustration suivante pourrait d'avantage 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

---

2. voir : [http://en.wikipedia.org/wiki/Ackermann\\_steering\\_geometry](http://en.wikipedia.org/wiki/Ackermann_steering_geometry) sur "Ackerman steering"

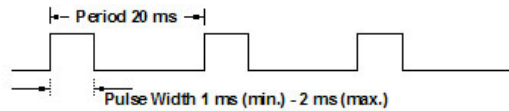


FIGURE 2.1 – Schéma de la modulation du signal électrique [2]

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

### 2.2.5 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 [4]. 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.

### 2.2.6 Etat actuel du système de guidage

Le servo n'est pas fonctionnel sur la plaque électronique que nous avons conçu nous-même. Selon nos diagnostics, il s'agit d'une mauvaise soudeure et donc le servo 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.

## 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 mesuré à l'aide d'un multimètre du gymnase) se trouvent à la section (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 configuration 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.

Le schéma suivant pourra d'avantage éclairer le lecteur.

#### **Explications**

Pour fabriquer un H-bridge, il faut utiliser entre autres des transistor



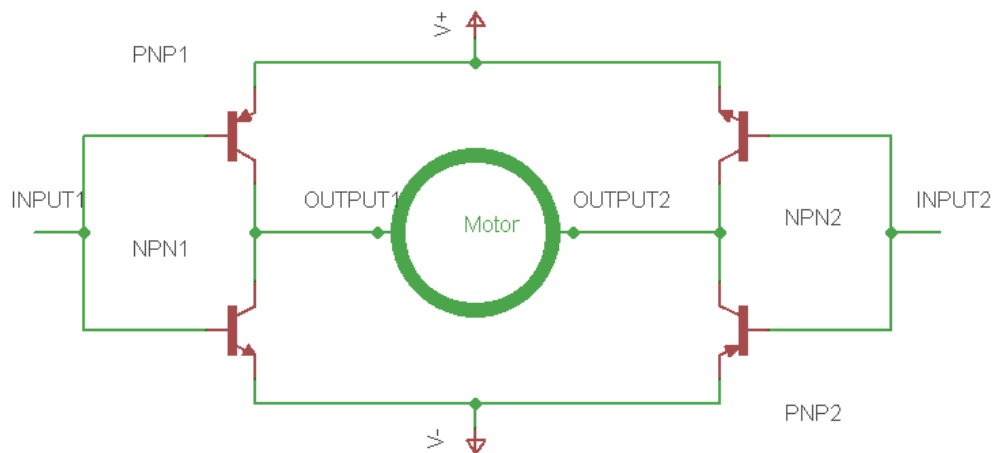


FIGURE 2.2 – Schéma d'un H-bridge

NPN<sup>3</sup> et PNP<sup>4</sup>. 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[3].

On peut voir que selon le *INPUT*, on obtiendra des tensions aux 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 :

3. Négatif-Positif-Négatif

4. Positif-Négatif-Positif

<i>INPUT1</i>	<i>INPUT2</i>	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.(2.2))

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.

## Chapitre 3

### Software

# Première version des systèmes embarqués

### 3.1 Software de l'Arduino

**P**OUR programmer l'Arduino, nous utilisons le logiciel dédié, *Arduino* (fig :3.1), 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 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 classes qu'on peut rajouter simplement en mettant la classe dans le dossier arduino. Ce software permet aussi à l'utilisateur d'envoyer des caractères à l'arduino par le biais du port *Serial*.



FIGURE 3.1 – Environnement Arduino

## 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 Guvview (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<sup>1</sup>. Cet accessoire ne nécessitant pas de *driver*<sup>2</sup>, il est donc directement reconnu et peut rapidement être configuré par le Raspberry Pi.

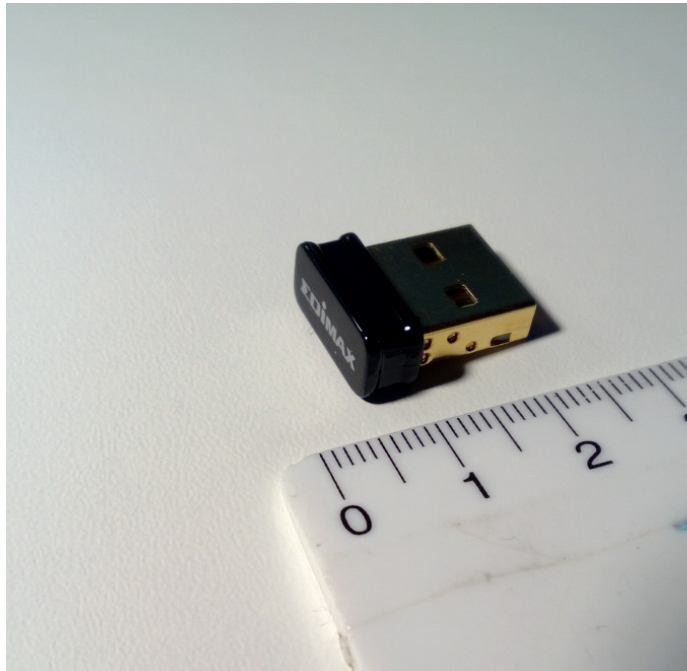


FIGURE 3.2 – *Dongle* Edimax EW7811Un. Connecté par USB à la Framboise, il permet au Raspberry Pi de se connecter via Wi-Fi à une borne

### 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

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

2. Un driver est un petit logiciel permettant d'exploiter le Hardware

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 toucherait 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 Gvvcview

Gvvcview 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 Série. 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.

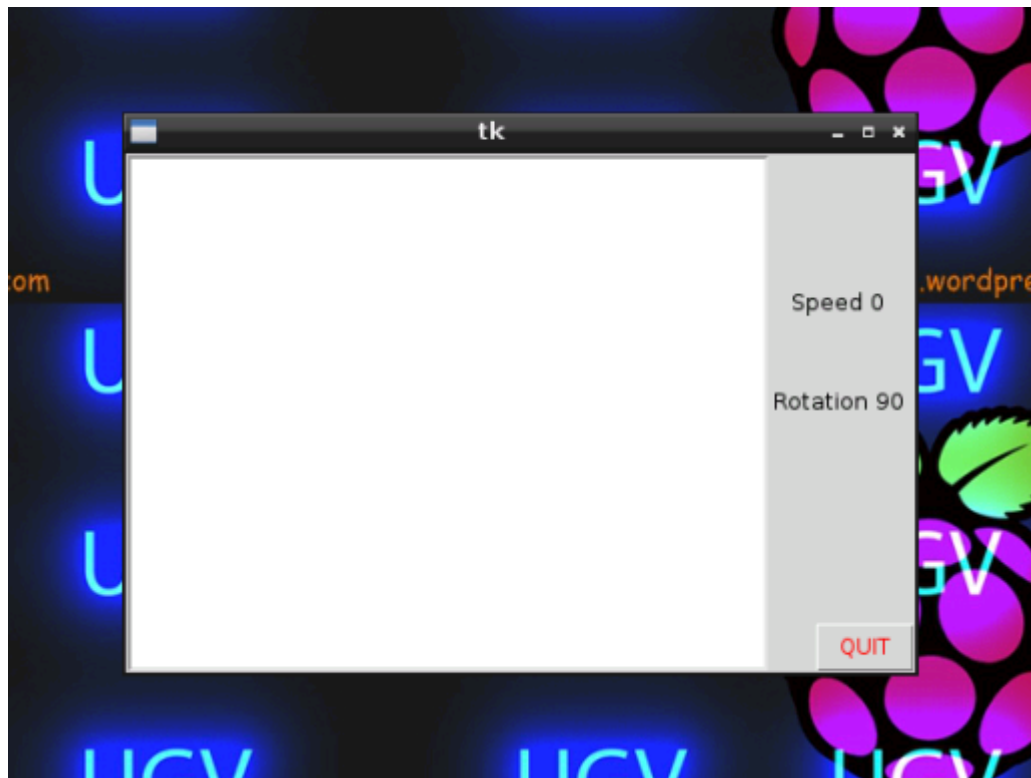


FIGURE 3.3 – Fenêtre X du code python qui permet la lecture du clavier. Cette fenêtre permet de contrôler le drone.

## 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 quand on doit choisir un programme, c'est aussi un langage constamment mis à jour et il possède une large communauté, ce qui a 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 aborder tout d'abord le choix de transmission sans fil 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 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 connection wi-fi qui permettent d'envoyer des données (*Packets*) d'un ordinateur à un autre. Un des deux ordinateur 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. Mai revenons tout d'abord aux différences.

Le protocole UDP<sup>1</sup> est un protocole qui n'est pas orienté connection. 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<sup>2</sup> est l'opposé de l'UDP, il est orienté connection. 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 reception 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 connection est simplement perdue, tandis qu'avec le protocole UDP, nous ne savons pas s'il y a une perte de connec-

---

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

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

tion, puisqu'il n'y a pas eu de connection.

## 4.2 Java

## 4.3 OpenCV

## .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

[4]

```
// 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 degree
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position stored
    // 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 degree
  {
    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 d'atteindre la position demandée. Etant donné que cette opération est itérée 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 graphical 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=master.quit)
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_reached")

    def shift_right(self, event):
        if self.rotation < 140:
            self.rotation = self.rotation + 10
            self.rotation_label.config(text = "Rotation_%i" % (self.rotation))
            print "ROTATION: _new_angle_is_%i" % (self.rotation)
        else:
            print "ROTATION: _warning! _max_angle_is_reached"
            self.messages.insert(INSERT, "ROTATION: _warning! _max_angle_is_reached")

    def shift_left(self, event):
        if self.rotation > 50:
            self.rotation = self.rotation - 10
            self.rotation_label.config(text = "Rotation_%i" % (self.rotation))
            print "ROTATION: _new_angle_is_%i" % (self.rotation)
        else:
            print "ROTATION: _warning! _minimal_angle_is_reached"
            self.messages.insert(INSERT, "ROTATION: _warning! _minimal_angle_is_reached")

    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] the free encyclopedia From Wikipedia. Raspberry pi,?? [http://en.wikipedia.org/wiki/Raspberry\\_Pi](http://en.wikipedia.org/wiki/Raspberry_Pi).
- [2] the free encyclopedia From Wikipedia. Servo control, Octobre 2012. [http://en.wikipedia.org/wiki/Servo\\_control](http://en.wikipedia.org/wiki/Servo_control).
- [3] Robot Room. H-bridge motor driver using bipolar transistors,?? <http://www.robotroom.com/BipolarHBridge.html>.
- [4] Barragan Studio. Servo sweep, Septembre 2010. <http://arduino.cc/en/Tutorial/Sweep>.
- [5] Arduino Team. Arduino spec.,?? <http://arduino.cc/en/Main/ArduinoBoardUno>.