

Rapport de Stage

I- HardWare

[La Raspberry](#)

[L'Arduino](#)

[La connection série Arduino-Raspberry](#)

[La boussole](#)

[La carte de puissance Rokraft](#)

[Autre matériels utilisés](#)

II- Software

[L'installation de la carte Raspberry-PI](#)

[Le Système d'exploitation](#)

[Le réseau Ad-hoc avec B.A.T.M.A.N](#)

[Le logiciel de capture vidéo Motion](#)

[Java](#)

[La librairie Rxtx](#)

[Instructions d'installation](#)

[Quelques commandes utiles à connaître](#)

[Le logiciel raspiSoft](#)

[Structure](#)

[MainModel](#)

[Server](#)

[Serial](#)

[Servos](#)

[GPS](#)

[Le logiciel intégré à l'Arduino](#)

[L'IHM](#)

III- Problèmes, remarques et observations

[Le choix du système d'exploitation et du software](#)

[Le dispositif](#)

[Le réseau ad-hoc en mesh-rooting](#)

I- HardWare

Les petits buggys conçus pour la participation au concours sont basés sur le couplage de deux cartes bon marché très largement utilisées dans le domaine de la robotique.

Suivant son modèle, la raspberry-PI coûte une modique somme comprise entre 25 et 35 €. Très fonctionnelle, elle ne dépasse guère la taille d'une carte de crédit et son processeur est basé sur l'architecture ARM. Au-delà de ses faibles dimensions, cette dernière reste un outil puissant pour une multitude d'utilisation.

La carte Arduino se vend entre 20 et 40 € et se dérive en de nombreux modèles apportant chacun leurs fonctionnalités. Elle intègre un micro-contrôleur Atmel AVR pouvant être programmé avec un langage basé sur java et à forte similitude avec le langage C, ce qui permet aux utilisateurs de prendre le contrôle sur la machine.

La Raspberry



Le modèle de la carte installé sur chaque robots est le B.

Il comprend tout ce que comporte le modèle A, supplié par:

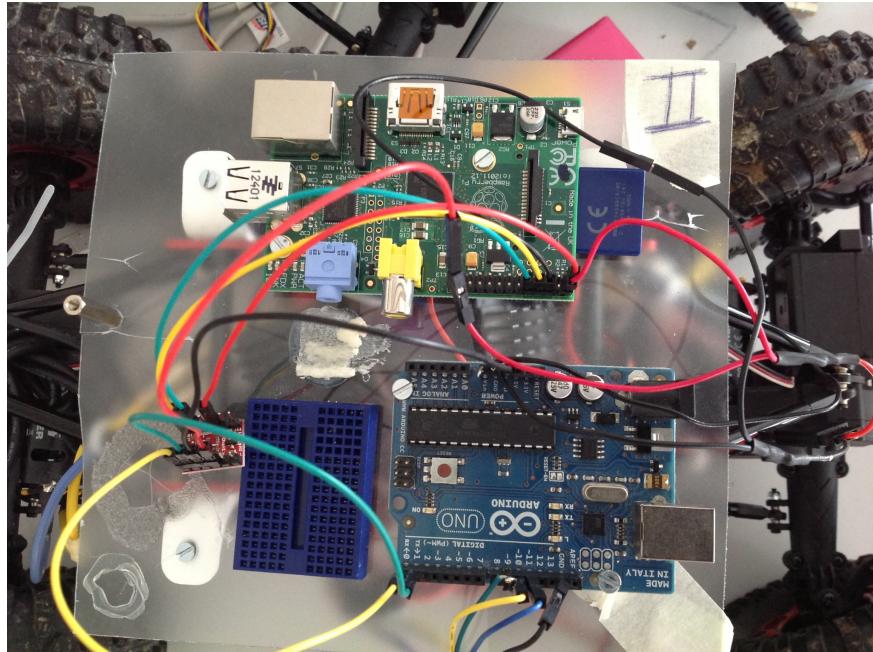
- un port USB
- un port fast ethernet
- 512 Mo de mémoire SDRAM au lieu de 256.

L'Arduino



La carte Arduino est l'élément clé. Elle produit les signaux électriques traduisant les ordres à destinations des dispositifs à contrôler. Nous utilisons le modèle Arduino UNO basé sur l'Atmega328. Il possède 14 pins d'entrée/sortie digitale, dont 6 peuvent être utilisés comme sortie de signaux PWM (utilisés pour le contrôle de servos), ainsi que 6 pins d'entrée analogue. Ce composant opère sous une tension de 5 Volts, il faut donc être prudent lorsque l'on raccorde cette dernière avec la carte Raspberry, qui quant à elle ne tolère que du 3,3 Volts pour son GPIO.

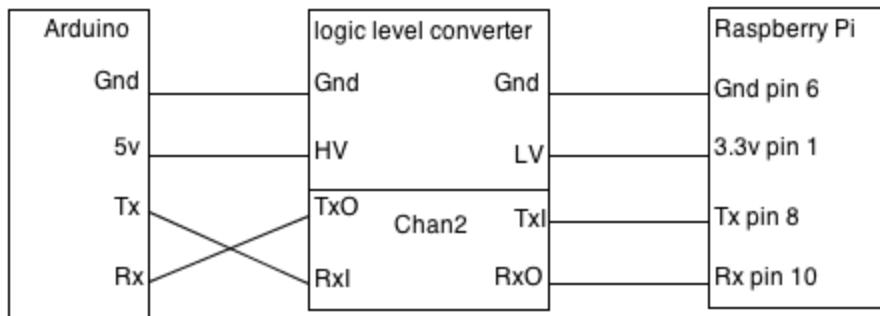
La connection série Arduino-Raspberry



Pour permettre une communication série entre la Raspberry-Pi et l'Arduino, les deux carte sont reliées entre elles par leur pins d'entrée/sortie RX TX.

La connection a été réalisé à l'aide d'un convertisseur de tension qui fait le pont entre les pins digitales (RX -> pin 0 et TX -> pin 1) de la carte Arduino opérant avec une tension fixée à 5 Volts et les pins GPIO (RX -> pin 10 et TX -> pin 8) de la Raspberry qui supportent un seuil de tension s'élevant à 3,3 Volts.

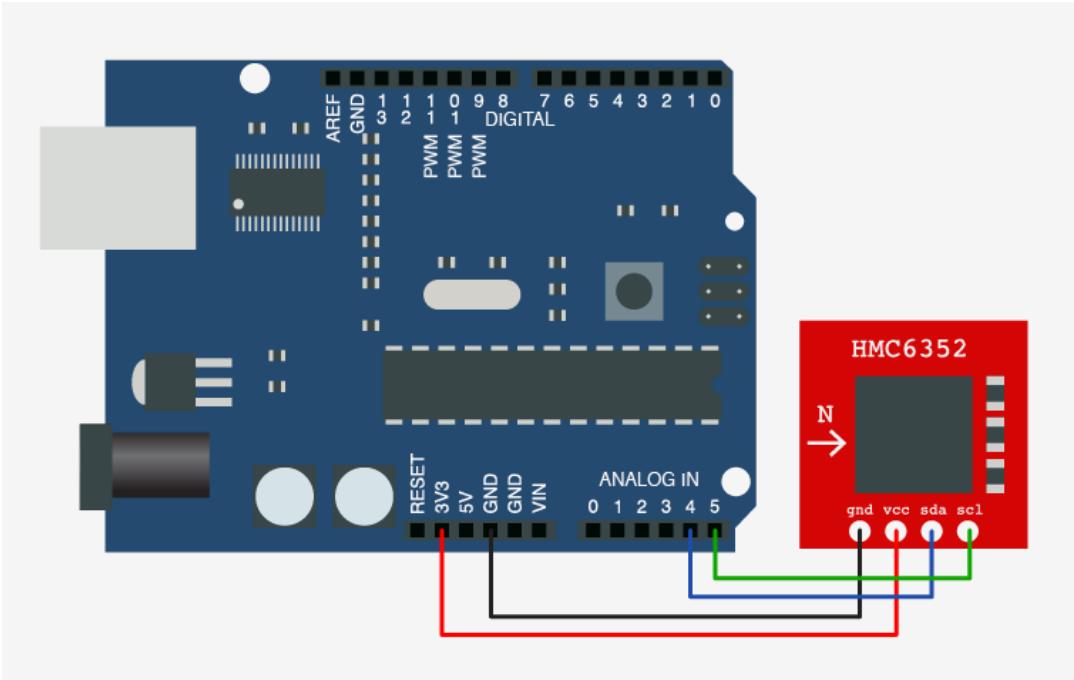
Le montage réalisé est représenté ci-dessous :



A la différence que la masse de la sortie LV du convertisseur n'a pas été reliée, en effet la masse de la raspberry (pin 6) est directement reliée à la masse de la batterie alimentant le dispositif.

Sur la Raspberry cette communication sera accédé par l'emplacement /dev/ttyAMA0.

La boussole



L'intégration d'une boussole au dispositif est utile à la localisation du buggy dans l'espace. Le montage a du être refait pour simplifier ce qui avait été fait initialement par Mr Murilo. Le modèle utilisé est le HMC6352 du constructeur sparkfun electronics. Cette dernière est directement reliée à l'Arduino tel que cela est représenté sur l'image ci-dessus.

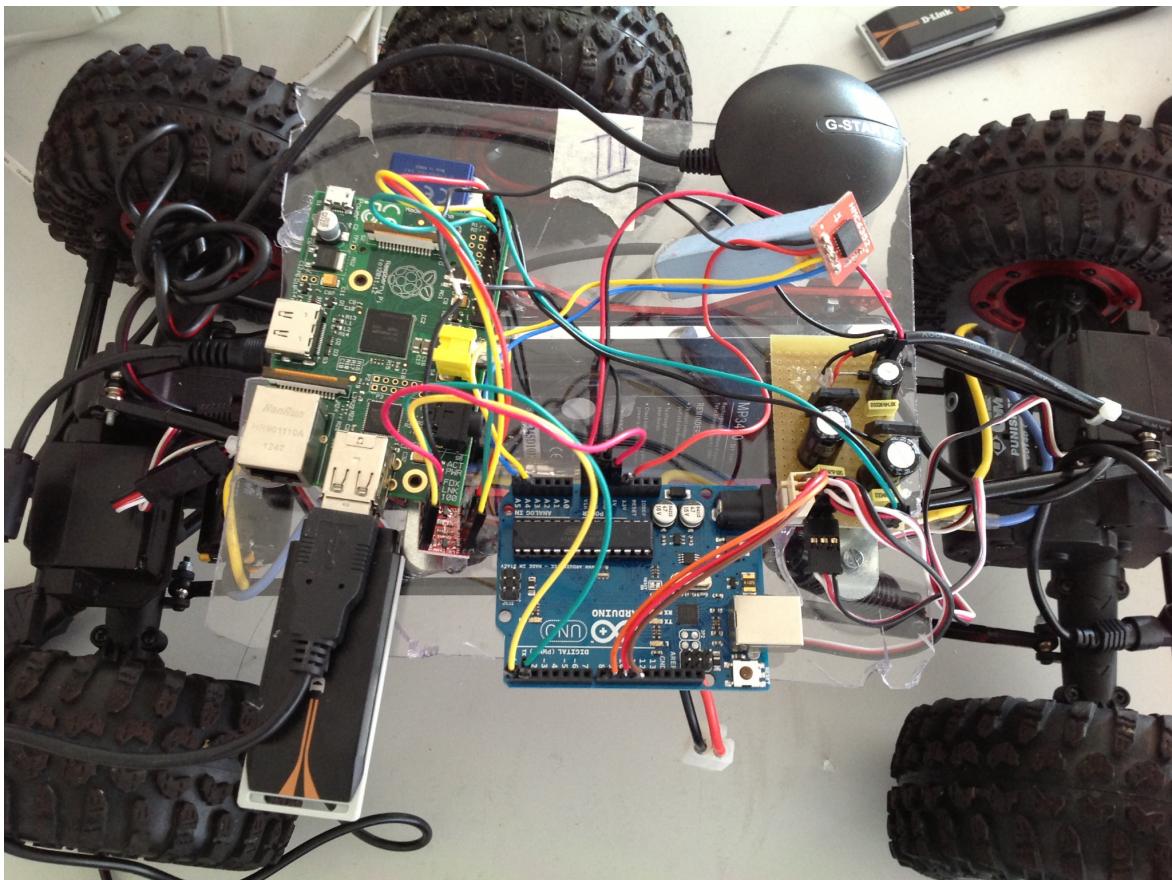


Photo du montage final

La carte de puissance Rokraft



La Rokraft est une carte qui traite les signaux PWM et envoie aux servos la puissance nécessaire qui correspond au signal reçu.

Cette dernière prend le contrôle du servos engine qui est le moteur permettant au buggy d'avancer ou de reculer.

Elle a été calibrée pour s'accorder avec les valeurs choisies lors de la programmation des commandes destinées au moteur principal dans le code du software embarqué (voir notice).

Autre matériels utilisés

Pour le projet, il est également nécessaire d'utiliser une clé wifi qui permet d'assurer le lien entre chaque buggy de la meute, ainsi qu'un caméra de type webcam, utile au téléguidage des robots et à la capture photographique d'éventuels objets d'intérêts qui seront présents sur le circuit.



II- Software

L'installation de la carte Raspberry-PI

Le Système d'exploitation

La carte SD de la raspberry a été installée avec la version Raspbian “wheezy” datée du 25 mai 2013 de la distribution debian.

Le réseau Ad-hoc avec B.A.T.M.A.N

Pour la gestion du mesh rooting sur le réseau ad-hoc, il a été décidé d'opter pour l'utilisation du module batctl installé à partir des dépôts de paquets debian.

Au préalable, un réseau ad-hoc a été configuré pour chaque carte raspberry afin que tous les buggys disposent d'une adresse IP static qui leur est propre sur le réseau.

C'est par le biais de cette adresse que pourront communiquer chaque robots.

Le réseau est établit lors du boot, par l'exécution d'un script csh référencé dans le fichier init.d de la carte.

Le logiciel de capture vidéo Motion

Motion est le soft utilisé pour faire le lien entre l'objectif de la webcam embarquée sur chacun des buggys et le l'IHM de contrôle. Il permet de diffuser sur le réseau le flux vidéo filmé en direct. Il a été configuré dans le but d'obtenir une qualité d'image et une fluidité optimum pour que la téléopération du robot soit confortable.

Motion est lancé avec sa configurationn dès le démarrage de la carte raspberry-PI.

Java

Afin de pouvoir exécuter les logiciels programmé sous java, la version de jdk8 est installé sur toutes les cartes.

La librairie Rxtx

Rxtx est une librairie java utilisée pour établir la communication série entre la carte Raspberry et l'Arduino et simplifie largement la programmation du logiciel embarqué. Son fonctionnement s'apparente à celle de la classe Socket de java.net.

Instructions d'installation

Descriptif de l'installation d'une carte étapes par étapes.

1. Système d'exploitation et commandes utiles

[Télécharger la dernière version.](#)

Installation sur la carte SD-HC sous linux:

```
df -h
```

les partitions de la carte apparaissent avec leur chemin, sous la forme : /dev/sdb[x].

Ex:

```
sudo umount sdb1
```

```
sudo umount sdb2
```

```
sudo dd bs=1M if=/chemin/vers/image/raspbian of=/dev/sdb
```

Une fois l'os installé:

```
username : pi
```

```
password : raspberry
```

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

2. Configuration du réseau

```
apt-get install batctl
```

Création du réseau ad-hoc:

```
ifconfig wlan0 down
```

```
iwconfig wlan0 mode ad-hoc channel 1 essid MyAdhoc key 1234567890 ap 02:12:34:56:78:90
```

```
ifconfig wlan0 mtu 1528
```

```
ifconfig wlan0 192.168.0.[10X] netmask 255.255.255.0
```

Création du réseau avec l'interface bat0

```
modprobe batman-adv
```

```
batctl if add wlan1
```

```
ifconfig bat0 192.168.1.[10X]/24 netmask 255.255.255.0
```

```
ifconfig bat0 up
```

3. Installation et configuration de Motion

```
sudo aptitude motion
```

```
sudo nano /etc/default/motion
```

```
start_motion_daemon=yes
```

Écraser le fichier de configuration motion.conf dans /etc/motion/ avec notre propre fichier motion.conf (fourni)

4. Installation de java

[Télécharger la dernière version](#) (linux ARMv6-7).

```
scp jdk-8-ea-b99-linux-arm.tar.gz pi@192.168.1.[10X]:/home/pi
```

```
ssh pi@192.168.1.[10X]
```

```
tar zxvf jdk-8-ea-b99-linux-arm.tar.gz
sudo mv jdk1.8.0 /usr/local/java
export PATH=$PATH:/usr/local/java
export JAVA_HOME=/usr/local/java/jdk1.8.0/
java -version
sudo reboot
```

5. Installation de RXTX

Noter le nom du kernel obtenu grâce à la commande:

```
sudo uname -r
sudo nano /usr/include/linux/version.h
```

Ajouter la ligne:

```
#define UTS_RELEASE "nom_du_kernel"
wget http://rxtx.qbang.org/pub/rxtx/rxtx-2.2pre2.zip
unzip rxtx-2.2pre2.zip
cd rxtx-2.2pre2.zip
./configure --disable-lockfile
sudo nano src/gnu/io/RXTXCommDriver.java
```

Aller à la ligne 577 et rajouter les lignes:

```
"ttyACM",
"ttyAMA",
make
sudo make install
sudo reboot
```

6. Copier l'image de l'installation pré-configuré sur carte SD

Backup raspberry pi:

```
dd if=/dev/sdb | gzip > ~/image.gz
```

Restaurer :

```
chmod 777 /dev/sdb
gzip -dc ~/image.gz | dd of=/dev/sdb
```

Quelques commandes utiles à connaître

Démarrer/Arrêter motion :

```
sudo service motion start/stop
```

Lancement du soft embarqué :

```
raspiSoft
```

Le logiciel raspiSoft

Structure

Le logiciel embarqué sur la Raspberry-PI a été organisé de la manière suivante :

Le programme est composé de quatre classes dites “modèle” qui contiennent toutes les déclarations et initialisations de variables utiles à la gestion des différentes parties composant le logiciel. Elles sont regroupées dans le package com.models du projet et sont instanciées à partir de la classe MainModel.

Pour ce dernier, nous avons besoin de gérer une communication réseau, une communication série, le calcul des valeurs de commande des servos moteurs, ainsi que le calcul lié au positionnement GPS du robot.

Toutes les opérations de calcul sur les données de gps et de servos sont codées dans les classes Servos et GPS situées dans le package com.computing, et toutes les opérations d'émission et de réception de données sur les communications se trouvent dans les packages com.server et com.serial.

Pour finir, le programme est exécuté à partir du pacakage com.main, dans lequel se trouve la classe Buggy contenant la méthode main.

MainModel

La classe MainModel instancie toutes les autres classes modèles du projet dans son constructeur. Elle initialise également une instance de l'énumération Mode servant à la gestion des différents modes de fonctionnement du programme en fonction des contextes d'opération du robot. Un buggy peut donc fonctionner en mode automatique (AUTO), manuel (MAN), injoignable (UNREACHABLE) et arrêt (STOP).

Le mode automatique n'a pu qu'être ébaucher, tout comme la classe GPS.

Server

Les objets et variables servant à la création et au maintient du serveur sont regroupés dans la classe ServerModel (entre autre: Socket, ServerSocket, Reception, Transmission, TIME_OUT, ...).

La méthode serverSettings configure un serveur et met le programme en pause jusqu'à la connection d'un client. Dès lors que la connection est établie, la variable mode est mise à jour vers la valeur MAN, ce qui permettra la téléopération du buggy par l'opérateur.

Un time-out a été implémenté pour que la connection soit réinitialisé au bout de 5 secondes à partir du moment où il y a perte de connection, ou si aucun message du client n'est reçu par le buggy.

Par défaut, client et serveur s'échange le message "ping" à intervalle régulier d'une seconde.

Protocole d'échange d'informations entre serveur et client :

Client -> opérateur.

Serveur -> buggy.

Client : AUTO

Serveur : passe en mode automatique

Client : START

Serveur : passe en mode manuel

Client : STOP

Serveur : passe en mode stop

Client : MOVE val [R/L]

Serveur : affecte la valeur val à la variable engineForce et incrémente/décrémente les variables frontAngle et rearAngle en fonction de la commande suivante (si commande il y a).

Client : CAM [R/L]

Serveur : emplacement prévu pour la gestion du mouvement du mât de la caméra.

Serial

La communication série est implémentée par la classe SerialModel et gérée à partir de la classe Communicatoin du package com.serial.

Pour son fonctionnement, la librairie Rxtx doit être installée et paramétrée sur l'hôte du logiciel (voir installation de RXTX). Selon le moyen de raccordement utilisé entre la carte Raspberry et l'Arduino, cette communication passe par différents ports :

-si le branchement est effectué en USB, l'identifiant du port sera "/dev/ttyACM[x]"

-si le branchement est effectué par gpio - digital pins, l'identifiant sera "/dev/ttyAMA[x]"

Le data-rate de la communication série a été initialisé à 115 200 bauds.

Protocole d'échange d'informations entre Raspberry et Arduino :

R -> Raspberry.

A -> Arduino.

R : E

R : val

R : .

A : envoie la valeur val au moteur principale.

R : F

R : val

R : .

A : envoie la valeur val au moteur asservis en position avant.

R : R

R : val

R : .

A : envoie la valeur val au moteur asservis en position arrière.

R : C

A : demande le résultat à la boussole et renvoie ce dernier.

A : C

A : val

Servos

Pour la gestion des moteurs lors de la téléopération, une méthode simple de régulation d'angle est implémentée dans la classe Servos.

Cette dernière s'effectue dans la méthode steeringMAN et utilise les variables frontAngle, rearAngle et engineForce de la classe ServosModel. steeringMAN repose sur l'affectation de la variable news du modèle, qui influe sur les données envoyées par le port série à l'Arduino.

Un pas de calcul, qui est utilisé pour l'incrémentation ou la décrémentation de ses variables déclenché par action de direction sur le buggy, est initialisé à 10; ainsi les roues avant et arrière effectueront un mouvement de 10° vers la droite ou la gauche, en fonction de la direction décidée par l'opérateur.

Une méthode de calcul des valeurs servos fonctionnant grâce au principe de la régulation en cap a également été programmée, mais n'a pas pu être approfondie et testée par manque de temps. Nous avons du nous focaliser sur d'autres objectifs.

GPS

La programmation de la gestion du positionnement par données gps du buggy n'est pas allé très loin, par manque d'information sur les méthodes utilisées pour l'odométrie du robot. Cependant, le stockage de ces informations a été pensée pour leur utilisation dans une repère cartésien, où l'unité est exprimée en mètres.

Un objet de la classe File et un FileOutputStream sont initialisés dans GPSModel et sont prévus afin d'exporter les données gps lié au buggy hôte dans un fichier gps_data.txt afin d'y être sauvegardées et éventuellement réutilisées.

Ces données sont sauvegardées sous la forme suivante :

HH:MM:SS=longitude latitude

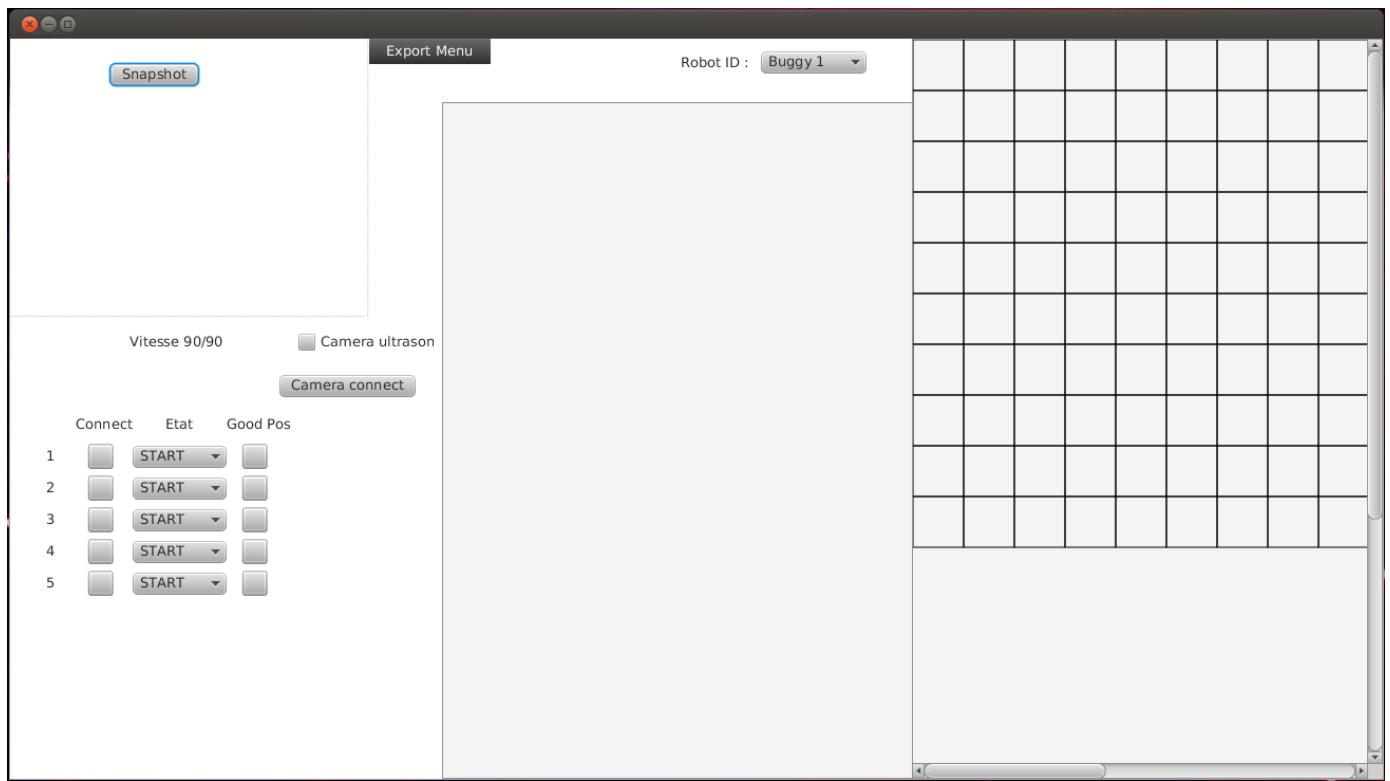
Le logiciel intégré à l'Arduino

Ce programme a été codé pour qu'il soit le plus simple possible dans l'état des choses.

Très court, il remplit son rôle: il communique les commandes servos envoyées par le logiciel embarqué sur la Raspberry aux moteurs connectés aux pins PWM et envoie des requêtes à la boussole avant d'expédier le résultat vers raspiSoft.

Afin de synchroniser l'envoie des données par la raspberry, la réception de ces dernières et leur traitement par l'Arduino, et dans le but d'éviter la perte de paquet lors de ce processus; des délais de mise en pause des programmes ont du être rajouter. Cela a été fait par tâtonnement, mais de façon à ce que la communication soit optimum en qualité de temps et de traitement.

L'IHM



Raccourcis du soft :

z -> avance.
s -> recule.
q -> gauche.
d -> droite.
r -> accélère.
f -> décélère.
a -> tourne gauche caméra.
e -> tourne droite caméra.
&, 1 -> focus buggy1.
É, 2 -> focus buggy2.
", 3 -> focus buggy3.
", 4 -> focus buggy4.
(, 5 -> focus buggy5.

Pour le code il reste des modifications à effectuer:

- la fonction initExport() située dans ControlVueController
 - la carte, j'ai mis deux cartes :
 - CarteCanvas qui affiche une grille et un triangle pour les buggys.
- "gc.setFill(color. X)" Permet de changer facilement la couleur du buggy, j'ai aussi tenté de faire

une fonction de rotation du triangle pour qu'il soit incliné en fonction de l'angle de la boussole mais ça ne marche pas (mauvaise formule sûrement).

CarteCanvasOld est une carte pour les données GPS (sera peut-être utile ?) bref je l'ai gardé. Le code est bien commenté normalement, il ne devrait pas avoir de problème de compréhension.

III- Problèmes, remarques et observations

Sur l'ensemble de la période de travail sur le projet Eurathlon, beaucoup de problèmes et divers difficultés ont été rencontrées et ont parfois ralenti l'avancement.

De plus il a fallu passer énormément de temps à se documenter dans divers domaines car pour la plupart des tâches à réaliser, nos connaissances n'étaient pas assez poussées pour pouvoir se lancer efficacement dans leur réalisation. La progression a donc été plutôt lente au commencement.

Le choix du système d'exploitation et du software

A plusieurs reprises, nous avons changé d'avis sur le choix de la distribution du système d'exploitation à installer sur la Raspberry-PI et des softs utilisés pour l'implémentation d'un mesh-rooting sur réseau ad-hoc et la diffusion du flux de capture vidéo.

En effet, nous étions parti sur l'installation de la version soft-float de la distribution debian avant de changer pour la distribution Arch Linux, pour finir par revenir sur la version hard-float "wheezy" de debian.

Le dispositif

Initialement, nous avions connecté la carte Arduino et la Raspberry-PI par cable USB mais afin d'économiser un port USB et d'éviter l'utilisation d'un HUB, ce qui pourrait poser un problème au niveau de l'alimentation du dispositif, nous avons opté pour faire passer la communication série par le biais des GPIO de la Raspberry et des pins digitaux de la carte Arduino.

A ce jour le buggy que nous avons le plus transformé et le buggy n°3 :

- Le raccordement UART passe par un convertisseur de tension.
- La boussole est directement branchée sur l'Arduino.
- La masse de la boussole et du convertisseur de tension se rejoignent sur la masse de l'Arduino reliée à celle de la batterie.
- Le connecteur de la Rokraft a été refait (une de ses fiches s'était déssoudé).
- La Rokraft a été calibrée en fonction des valeurs extrémum de commande du moteur principal, qui sont :

maximum marche avant : 180
minimum marche avant : 100
neutre : 90
minimum marche arrière : 80
maximum marche arrière : 0.

Le réseau ad-hoc en mesh-rooting

Au niveau du réseau ad-hoc, quelques tests on été effectués sur le terrain de foot de l'école. En pratique le mesh-rooting géré par le module B.A.T.M.A.N fonctionne correctement, mais nous avons remarqué que les clé wifi des buggy n'avait pas une très grande portée, de plus lorsque le flux vidéo passe par le réseau, le débit reste insuffisant pour pouvoir faire parvenir une image fluide, lorsque que le buggy est contrôlé par un opérateur.

Nous avons également constaté qu'il s'avère souvent que la carte Raspberry soit capricieuse au moment du démarrage après sa mise sous tension: il ne faut pas hésiter à être franc lorsque l'on branche la batterie.