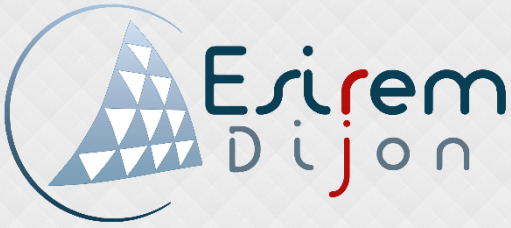
COMPTE RENDU
WIFIBOT

19/06/2017

TP MODULE ITP32

AU SEIN DE

L'ECOLE



RÉALISÉ PAR : VINCENT **CANDAPPANE**
: ANTONIN **JAVELLE**

ENSEIGNANT **PRINCIPAL** : M. **BARTHÉLÉMY** HEYRMAN
ENSEIGNANT **ENCADRANT** : M. **MARROQUIN** CORTEZ

Activités confiées : Réaliser et gérer un projet complet 'wifibot' à travers la manipulation de Qt Creator

REMERCIEMENTS :

Avant de présenter notre rapport, nous tenons à remercier : M. Marroquin Cortez Roberto Enrique et M. Barthélémy Heyrman pour l'encadrement du projet ainsi que pour les nombreux aides et conseils qui nous ont été fournies lors des séances dédiées à la manipulation de Qt Creator et du Wifibot.

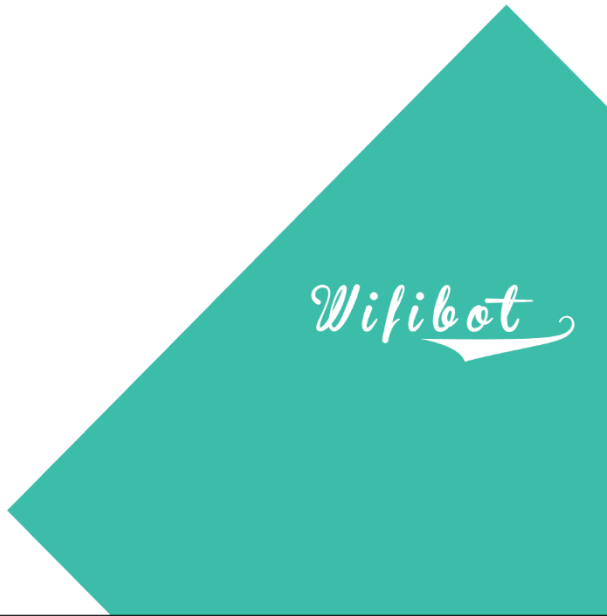


TABLE DES MATIÈRES

INTRODUCTION

Aujourd'hui, de plus en plus de robots sont utilisés, et cela dans divers domaines. Ils sont aussi utilisés dans les laboratoires pour tester les algorithmes, les stratégies avant de les appliquer à l'environnement réel. Les simulateurs, permettant de faire évoluer les robots dans un environnement précis, sont l'outil principal du chercheur.

Dans le cadre du module **ITP31** dirigé par M. Heyrman et M. Marroquin, il nous a été demandé de développer une application graphique sur un environnement de développement spécifique (Qt Creator) capable de commander un robot à distance par une liaison wifi.

De plus, l'ultime objectif, plus implicite, mais tout aussi important, est de nous initier à la gestion de projet. Pour cela, nous devons apprendre à programmer* à plusieurs, nous partager les tâches, planifier au mieux le travail pour avancer le plus efficacement possible.

Ainsi, nous avons subdivisé le projet en trois grandes parties, en effet dans un premier temps nous nous familiariserons avec l'environnement de développement Qt ainsi que SourceTree un logiciel de gestion de version (versioning), puis nous nous sommes intéressés à la compréhension de la documentation disponible sur internet afin d'arriver à la réalisation de l'application et le pilotage du robot.

*Ce rapport présente les principaux travaux d'implantation et de tests du robot. Mots-clés : WiFiBoT, Qt Creator

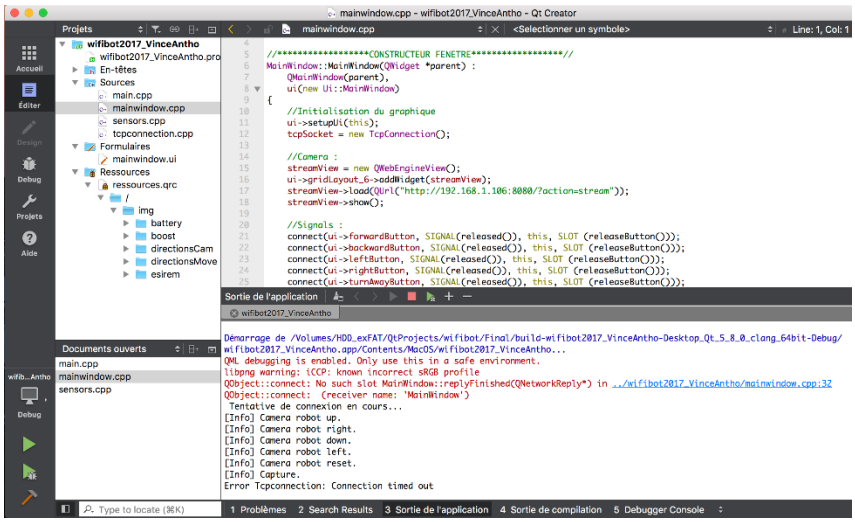
I - QT CREATOR

Tout d'abord, nous avons dû apprendre à nous servir du logiciel qui nous a été imposé pour réaliser ce projet, à savoir Qt Creator. Qt creator est un environnement de développement intégré multi-plateforme faisant partie du Framework Qt, ainsi il a pour avantage d'être un intermédiaire entre l'utilisateur et l'OS c'est-à-dire qu'il traduit les instructions pour l'OS.

Qt Creator est orienté pour la programmation en C++ et a été modélisé pour être compatible multi-plateforme, ce qui apporte un certain confort d'utilisation que ce soit pour des applications Android, Windows, Apple, Linux ...etc.

Comme la plupart des IDE, Qt Creator intègre directement dans l'interface :

- Un débogueur, très simple à prendre en main et assez puissant
- Un outil de création d'interfaces
- Des outils pour la publication de code sur Git et Mercurial
- Des éditeurs de texte avec autocomplétion et coloration syntaxique
- Une documentation Qt.



Capture d'écran de Qt Creator 5.8.0 et son débogueur sous Mac OS Sierra

Qt Creator utilise sous les systèmes d'exploitation du noyau UNIX, le compilateur **gcc** et intègre un nombre important de librairies permettant le développement simple, rapide et efficace.

II - CARACTÉRISTIQUES DU ROBOT WIFIBOT LAB V3

1. DESCRIPTION DU ROBOT : WIFIBOT LAB V3

Au niveau de la documentation, un site officiel complet dédié pour le robot wifibot est présent, celle-ci nous a été utile pour comprendre le fonctionnement du robot, connaître les différentes caractéristiques techniques de celui-ci (capteurs, batterie, connexion, envoi de trame…).

Le robot en question n'est autre qu'un ordinateur ordinaire connecté à plusieurs capteurs et moteurs. Ces caractéristiques basiques sont quasiment la configuration qu'un ordinateur peut avoir :

- Processeur 4 cœurs
- Mémoire RAM de 4G
- Disque SSD de 64G

Le wifibot utilisé à l'école ESIREM s'agit d'un wifibot Lab qui est une plateforme robotique modulaire créée par une entreprise française, permettant de couvrir un large spectre lié à la robotique mobile, à l'informatique industrielle et aux réseaux sans fil. Le système de base est composé d'un **châssis en aluminium anodisé**, d'une **caméra USB motorisée,** de **4 capteurs infrarouges,** d'une **carte WIFI** et de **4 moteurs** de mouvement.

2. CARACTÉRISTIQUES DE LA LIAISON CARTE WIFI - MODEM WIFI SANS FIL

De plus une carte WIFI assure la liaison sans fils au système avec le point d'accès configuré. Les utilisateurs peuvent ainsi concevoir des programmes directement sur le robot (VGA ou bureau distant via WIFI). Le wifibot que nous avions configuré fonctionnera en mode client/serveur, permettant à un client de contrôler les robots par une connexion de TCP/IP.

3. CARACTÉRISTIQUES DE LA CAMÉRA LOGITECH

Technical Specifications

- Motorized tracking (189° horizontal and 102° vertical)
- Carl Zeiss® optics
- Autofocus lens system
- Ultra-high resolution 2-megapixel sensor with RightLight™ 2 Technology
- Color depth: 24-bit true color
- Video capture: Up to 1600 by 1200 pixels (HD quality)
- Still-image capture: 8 megapixels (with software enhancement)
- Built-in microphone with RightSound™ Technology
- Frame rate: Up to 30 frames per second
- High-Speed USB 2.0
- Logitech QuickCam® software (with Video Effects™, filters, avatars, and face accessories)
- Works with Skype™, Windows Live™ Messenger, Yahoo®, AOL® and other compatible instant messaging applications

Dans les grandes lignes, cette caméra nous offre un **tracking motorisé** (suivi et localisation) offrant un champ de vision de 189 degrés et d'une inclinaison de 102 degrés.

Grâce aux fonctions avancées de **"Premium Autofocus"** et **"2.0 MP performance"**, les images restent nettes, même en gros plan.

Utilisant la technologie **RightLight™ 2**, la caméra s'ajuste intelligemment pour produire la meilleure image possible en fonction des milieux sombre ou éclairée.

III - PARTIE TECHNIQUE : CAHIER DES CHARGES

1. PRINCIPE DU PROJET

Le robot étant fourni sans interface de contrôle, l'objectif du projet repose sur la création et le développement d'une interface homme-machine permettant le contrôle à distance (via réseau Wifi) du Wifibot Lab V3.

Le cahier des charges demandé est constitué de 8 axes majeurs, ici détaillé dans l'ordre de notre réalisation :

- Connexion au robot,
- Pilotage du robot à l'aide du clavier,
- Pilotage du robot grâce à des boutons sur l'interface graphique,
- Affichage de l'image de la webcam,
- Exploiter et contrôler la motricité de la caméra,
- Récupération de la valeur de la batterie,
- Récupération et affichage des informations des différents capteurs : Odométrie, IR,
- Améliorations (détaillés dans la partie 5).

2. ACHEMINEMENT DES DONNÉES

Dans un premier temps il est nécessaire de mettre en place un système de connexion/déconnexion au robot afin de pouvoir contrôler ce dernier. Le robot possède une adresse IP et un port permettant de se connecter à lui via une connexion Wifi.

Deux types de communication sont alors possibles. Soit l'utilisation d'UDP (un protocole sans connexion) ou l'utilisation du protocole TCP (avec connexion). La principale différence entre ces deux types de communications réside dans l'acheminement des données et l'acquittement de ces dernières.

3. CONTRÔLE DU ROBOT

L'implémentation du code permettant le contrôle du robot est la phase maîtresse de notre projet. Pouvoir avancer, reculer, tourner à droite ou à gauche devait être mis en place correctement à la lettre pour un contrôle total du robot en déterminant le type de commandes.

S'offrent alors à nous alors plusieurs possibilités : touches de clavier, contrôle par clic de boutons virtuel via l'interface Qt, manette de jeu, Joystick ou encore détection de mouvement.

4. CONTRÔLE DE LA CAMÉRA

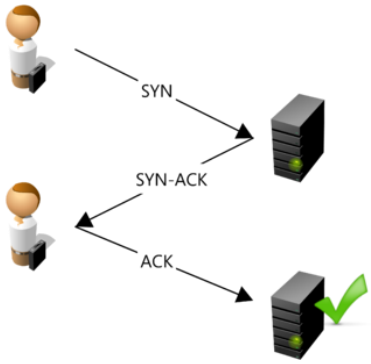
Afin d'obtenir un contrôle total de la caméra et d'assurer la récupération du flux vidéo sous n'importe quel angle il est nécessaire de pouvoir effectuer une rotation haute, basse, droite ou gauche de la caméra. Comme pour le déplacement de notre robot, il est alors nécessaire de déterminer la manière adéquate de commander la caméra.

IV - PARTIE RÉALISATION :

Dans cette partie, afin de détailler le fonctionnement, nous nous appuierons sur la documentation disponible sur le site officiel de wifibot, mais aussi sur celui de Qt Creator.

1. ÉTABLISSEMENT DE LA CONNEXION

La connexion du robot est gérée dans une classe à part entière : **tcpconnexion.cpp**. Nous avons décidé d'implémenter et mettre en place une connexion TCP, car les avantages de ce protocole sont les suivants :



- **TCP** : A l'inverse à l'UDP, TCP est orienté **connexion,**
- **Communication rapide,**
- **Fiabilité :** Système d'acquittement qui permet la confirmation de réception des données envoyées,
- **Contrôle CRC des données :** Celui-ci repose sur une équation mathématique, permettant de vérifier l'intégrité des données transmises.

Le wifibot peut être assimilé à un périphérique serveur qui répond à un client. Il utilise les sockets pour recevoir et émettre des informations en mode connecté via un protocole TCP. Afin de nous connecter au robot, nous avons réglé préalablement le point d'accès du réseau selon la manière suivante :

Port du robot : **15000**	Adresse IP du robot : **192.168.1.106**
Port de la caméra : **8080**	Adresse IP de la caméra : **192.168.1.106**

Par ailleurs nous avons implémenté au sein de notre programme un socket nommé *socket* en utilisant l'objet *QTcpSocket()* grâce à la classe *QTcpSocket* contenue dans une librairie Qt. Cet objet permet de créer directement un socket en mode de communication TCP ayant pour paramètre : @IP et le numéro de port d'écoute du robot. Voici une démonstration de l'établissement de connexion :

```
    socket->abort(); // On désactive les connexions précédentes s'il y en a
    socket = new QTcpSocket(this);
    socket->connectToHost("192.168.1.106",15020);// On se connecte au serveur demandé
```

Une fois que le serveur a répondu à l'appel du client, la connexion est établie de manière permanente jusqu'à la fin de l'envoie des trames (si le robot ne reçoit pas de trames dans un délai de 10ms la connexion se libère).

Ainsi un destructeur *disconnect()* de la classe **tcpconnection.cpp** est appelé pour se déconnecter :

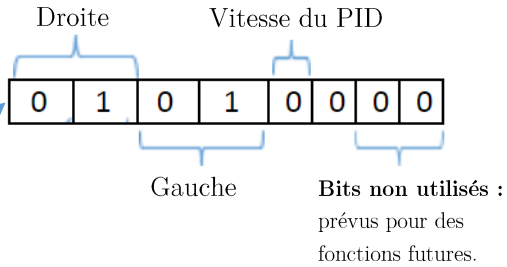
```
    void TcpConnection::disconnect(){
    moveRobot("stop");
    socket->disconnectFromHost();
}
```

À la fin de chaque utilisation du wifibot, il est nécessaire d'appeler le destructeur afin de libérer la connexion et donner main à un autre utilisateur qui pourrait attendre sinon la machine sera théoriquement connectée au programme et pourra créer des conflits lors de connexions simultanées.

2. L'ENVOI DES TRAMES : L'UTILISATION DU CONTRÔLE CRC

Les trames envoyées et reçues par le wifibot ne possèdent pas le même format. Ainsi les trames envoyées au wifibot sont des trames permettant de commander les différentes fonctions du wifibot alors que les trames renvoyées par ce dernier permettent de récupérer les données liées aux différents capteurs.

Char 1 : 255
Char 2 : 0x07(size)
Char 3-4 : Vitesse roue de gauche 0 → 240 tics max
Char 5-6 : Vitesse roue de droite 0 → 240 tics max
Char 7 : Contrôle la direction
Char 8, 9 : CRC (Contrôle de redondance cyclique)



Sur la documentation, il est indiqué que le Wifibot doit recevoir des tableaux de données où chaque case est représentée par un bit/caractère hexadécimal/octet. Pour générer les trames à envoyer, nous avons donc créé un buffer en utilisant l'objet *QByteArray()* que l'on nommera **frame**.

Cela fait, nous avons créé une fonction prenant en paramètre la direction des roues, ainsi qu'une vitesse générale. C'est dans cette principale fonction que le **contrôle directionnel des roues** sera réalisé et répertorié dans à un switch (pour éviter la surcharge inutile du code) :

```
moveRobot(QString direction, int speed){
    QByteArray frame;

    frame.append((char)0xFF);    //char1 : 255
    frame.append((char)0x07);    //char2 : size

    QStringList directions;
    directions << "forward" << "backward" << "left" << "right"<< "turn";
    switch(directions.indexOf(direction)){
        case 0 : //forward
            frame.append((char)((int)(240*speed)/100));
            frame.append((char)0x00);
            frame.append((char)((int)(240*speed)/100));
            frame.append((char)0x00);
            frame.append((char)0xF0);
            break;
        case 1 : //backward
            frame.append((char)((int)(240*speed)/100));
            frame.append((char)0x00);
            frame.append((char)((int)(240*speed)/100));
            frame.append((char)0x00);
            frame.append((char)0xA0);
            break;
}
```

Seuls les cas directionnels **"avancer"** et **"reculer"** sont décrits dans ce bout de code.

Les **char 8 et 9** quant à eux vont s'assurer de l'intégrité des données. C'est-à-dire elles verifient que trames envoyées ne sont ni corrompues ou ni manquantes, un algorithme de vérification est exécuté à chaque message envoyé : le **CRC** (*Contrôle de redondance cyclique*) codé sur **16 bits.**

```
quint16 crc = crc16(frame);
frame.append((char)(crc));
frame.append((char)(crc>>8));
socket->write(frame);
```

La fonction *moveRobot()* sera donc appelée dans le **mainwindow.cpp** afin de pouvoir être exécutée donnant accès au mouvement du robot à chaque événement crée, c'est-à-dire à chaque bouton correspondant appuyé.

3. LA RÉCEPTION DES TRAMES

De la même manière, lors de la réception nous remplissons un buffer pour la réception des données envoyé par le wifibot. En revanche la structure de la trame reçue possède une structure particulière de 21 char.

```
//On lit ce qui vient du robot
receiveData = socket->read(21);
```

4. LES CAPTEURS : ODOMÉTRIE, VITESSE DES ROUES, BATTERIE

Pour récupérer les valeurs de l'odométrie, des capteurs de vitesse et de la batterie, il a suffi de récupérer le compartiment du socket adéquat à chaque donnée souhaitée. On attribuera donc chacune de ces valeurs à une variable que nous utiliserons lors de l'affichage :

- Les capteurs infrarouge(IR) :

```
quint8 captIR_D, captIR_G;
captIR_G = ((int)(unsigned char)receiveData[3]);
captIR_D = ((int)(unsigned char)receiveData[11]);
```

Remarque : *seuls les capteurs IR de devant sont fonctionnels, ceux de l'arrière sont défectueux.*

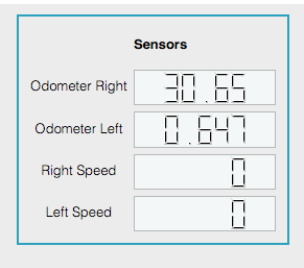
- La batterie :

```
quint8 battery;
battery = ((int)(unsigned char)receiveData[2]);
```

Remarque : *Comme le wifibot s'agit d'un ancien matériel, l'on suppose que la batterie est déchargée à 11V et pleine à 13 et et si c'est au dessus, c'est parce qu'il recharge.*

- Les capteurs de vitesse :

```
qint16 speedL, speedR;
speedR=((int)(unsigned char)receiveData[9]+(((int)(unsigned char) receiveData[10])<<8);
speedL=((int)(unsigned char)receiveData[0]+(((int)(unsigned char) receiveData[1])<<8);
```



5. RÉCUPÉRATION DU FLUX VIDÉO DE LA CAMÉRA

Pour se connecter à la webcam, il a fallu impérativement utiliser l'objet **QWebEngineView** de la librairie **<webenginewidgets>,** car l'ancienne version **QWebView** de la librairie **<webkitwidgets>** n'est plus d'actualité. Puis pour créer la fenêtre afin d'afficher la vidéo nous avons utilisé une fonction *load(QUrl(@IP))*, prenant en paramètre l'adresse du flux vidéo qui est composé de @IP de la caméra ainsi que du port d'écoute de la caméra. On a ainsi l'implémentation suivante qui a permis la lecture des images de la caméra :

```
streamCamera = new QWebEngineView();
ui->gridLayout_6->addWidget(streamCamera);
streamCamera->load(QUrl("http://192.168.1.106:8080/?action=stream"));
streamCamera->show();
```

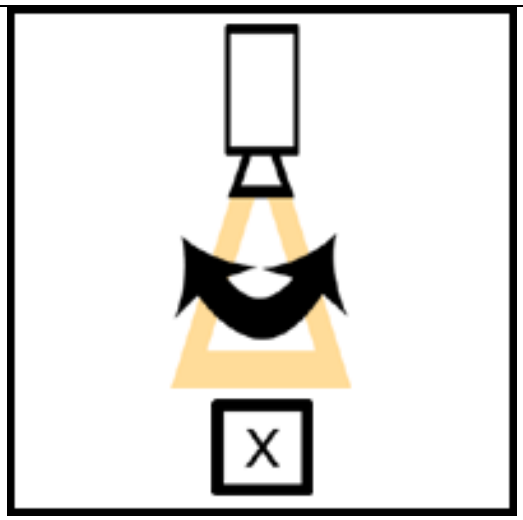
6. CONTRÔLE DE LA CAMÉRA ET AUTOFOCUS

Pour le contrôle de la caméra, nous allons directement interagir avec la page web qui permet de contrôler la caméra. Nous avons récupéré les ID de rotation suivant :

- **1 -** d'inclinaison **'tilt'** (Haut - bas) **: 10094853**
- **2 -** de panning **'pan'** (Gauche - Droite) **: 10094852**

Par exemple, pour pouvoir bouger la caméra en bas via le bouton 'bas', on a la fonction suivante :

```
void MainWindow::on_camDownButton_clicked(){
std::cout << "[Log] Camera robot down." << std::endl;
manager->get(QNetworkRequest(QUrl
("http://192.168.1.106:8080/?action=command&dest=0&plugin=0&id=10094853&group=1&value=250")));
}
```



Rotation caméra horizontale : panning

7. VITESSE DU ROBOT

La mise en place de la vitesse du robot en km/h a été une étape assez complexe à réaliser puisque la vitesse fournie par les capteurs du robot est en ticks/s et il a fallu trouvé une formule adéquate dans la documentation nous permettant de convertir ticks/s en km/h. Nous avons réglé ces configurations évoquées au cas par cas directement dans la fonction **moveRobot()** détaillée plus haut.

8. COMMANDE DU ROBOT

Afin de commander le robot, nous avons implémenté plusieurs types de moyens via :

- Les touches du clavier
- L'interface graphique
- Une manette Joystick **PlayStation® 3 DualShock**

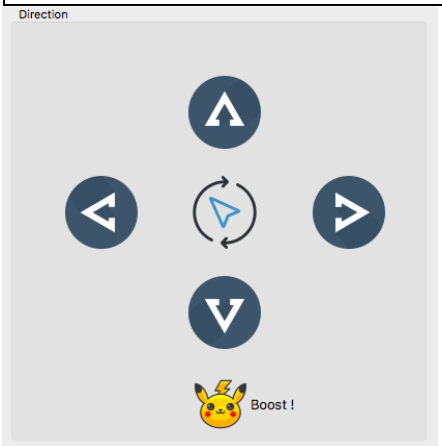
A. CONTRÔLE VIA LES TOUCHES DU CLAVIER

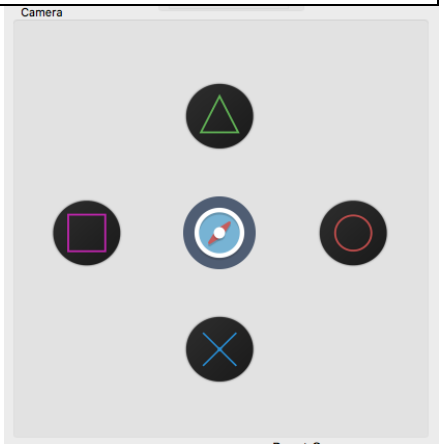
Afin de réaliser la commande du robot à l'aide des touches du clavier nous avons utilisé la classe **<QKeyEvent>** qui permet de décrire un événement du clavier :

```
void MainWindow::keyPressEvent(QKeyEvent *ev){
    switch(ev->key()){
    //1 - Directions
    case Qt::Key_Z:
        tcpSocket->moveRobot("forward", ui->speedSlider->value());
        std::cout << "[Log] Robot go forward." << std::endl;
        break;
}
```

B. CONTRÔLE VIA L'INTERFACE

```
void MainWindow::on_forwardButton_pressed(){
    tcpSocket->moveRobot("turn", ui->speedSlider->value());
    std::cout << "[Log] Robot turning." << std::endl;
}
```





C. CONTRÔLE VIA MANETTE PLAYSTATION® 3 DUALSHOCK

Ce mode de contrôle sera détaillé dans la section **"Améliorations"**.

V - INTERFACES

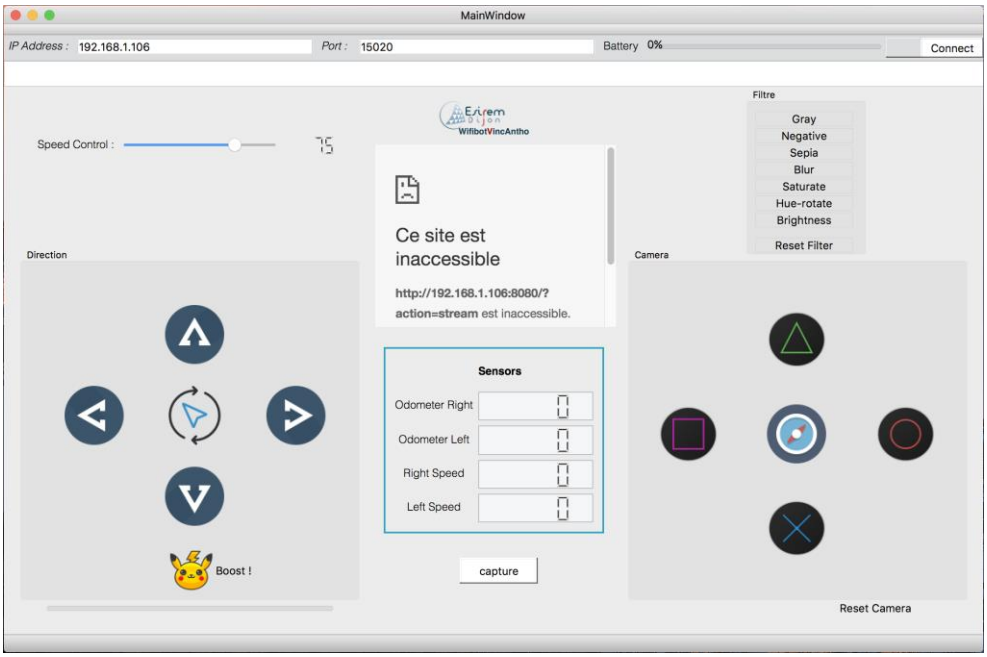
Pour la réalisation de l'interface graphique nous avons utilisé **Qt designer** qui propose des outils graphiques permettant d'implémenter une interface sans utiliser de ligne de codes.

Ainsi pour créer les **touches directionnelles** pour le déplacement du wifibot, la rotation de la caméra et divers nous avons tout simplement utilisé des éléments **QPushButton**.

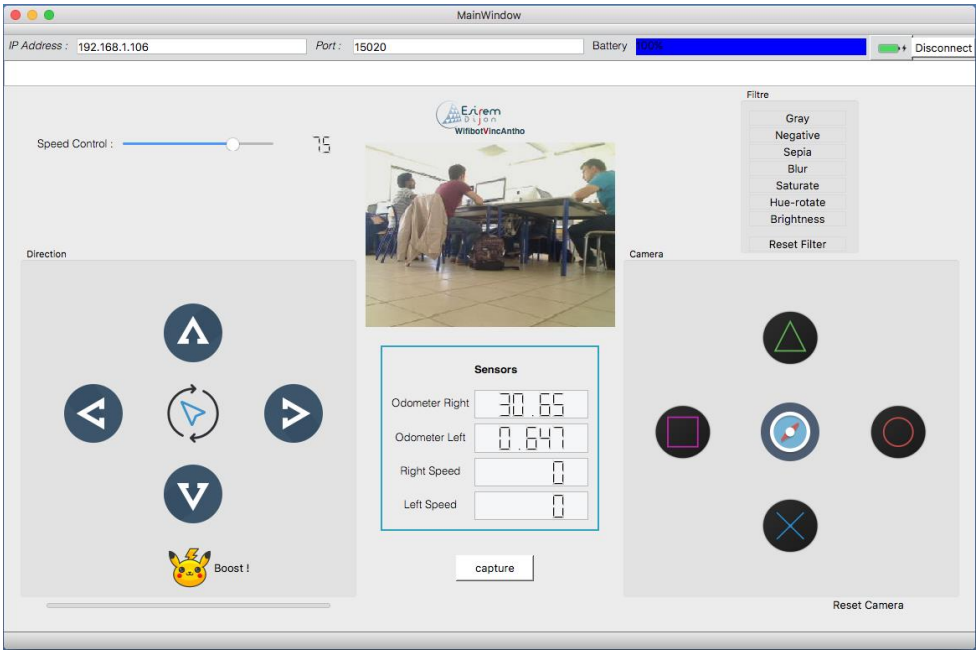
Le **contrôle de vitesse** et le pourcentage de batterie est tout simplement contrôlés par un élément **Slider**.

Pour les **descriptions informatives**, des **labels** ont été utilisés.

OFFLINE Mode : **Avant** la connexion du robot, l'interface se présente de la manière suivante :



ONLINE Mode : **Après** la connexion réussie, l'interface principale apparaît comme suit :



VI - AMÉLIORATIONS : NOUVELLES FONCTIONNALITÉS

Le cahier des charges a été respecté sur tous les points et nous étions même amenés à optimiser le programme afin de gagner en simplicité ainsi qu'en rapidité (exemple : utilisation de signal en C++, séparation en classe ...).

Nous avons aussi rajouté des fonctions supplémentaires pour rendre le pilotage du robot agréable et intuitif. Les améliorations envisagées sont les suivantes :

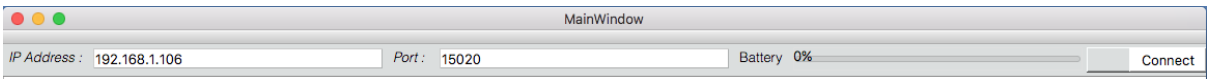
- **Interface de pilotage intuitif** et agréable à l'utilisation
- **Affichage de la vitesse** du robot en km/h
- **Slider :** Modification de la vitesse du robot
- **Batterie intelligente :** logo informatif en fonction du pourcentage de batterie
- **Manette de jeu PS3 :** Pilotage du robot à l'aide d'un contrôleur de Joystick
- **Le Nitro :** Accélération soudaine de la vitesse à l'appui du bouton « boost »
- **Ajout de différents filtres :** traitement d'image
- **Capture :** Snap de l'image de la Webcam du WIFIBOT

1. BATTERIE INTELLIGENTE

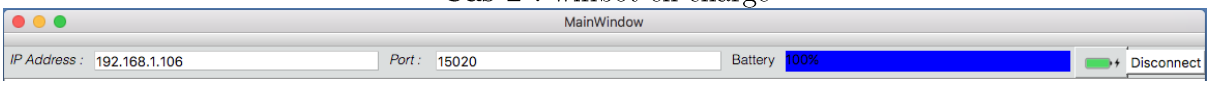
Il s'agit ici d'une indication du niveau de batterie via des icônes selon le pourcentage de la batterie :

Charging **Full** **Medium** **Low**

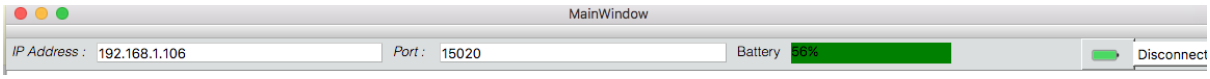
Cas 1 : wifibot éteint



Cas 2 : wifibot en charge



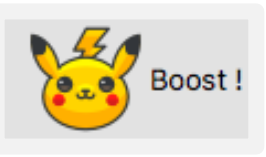
Cas 3 : wifibot à 58%



2. LE NITRO : UN ESPRIT DE COMPÉTITION

Le nitrométhane est utilisé comme carburant pour les véhicules de course, en particulier pour les dragsters, afin de fournir plus de puissance. Il est également utilisé comme comburant, pour donner un apport en oxygène dans les moteurs thermiques de voitures et avions-radiocommandes. Dans notre cas, il nous a paru amusant de reprendre le concept qui s'y rapproche. Implémenté en dernière minute, il s'agit d'une accélération du wifibot à sa vitesse maximale à l'appui du **bouton "Boost"** sur l'interface, **touche "ESPACE"** sur le clavier et **R1** sur la manette PS3 :

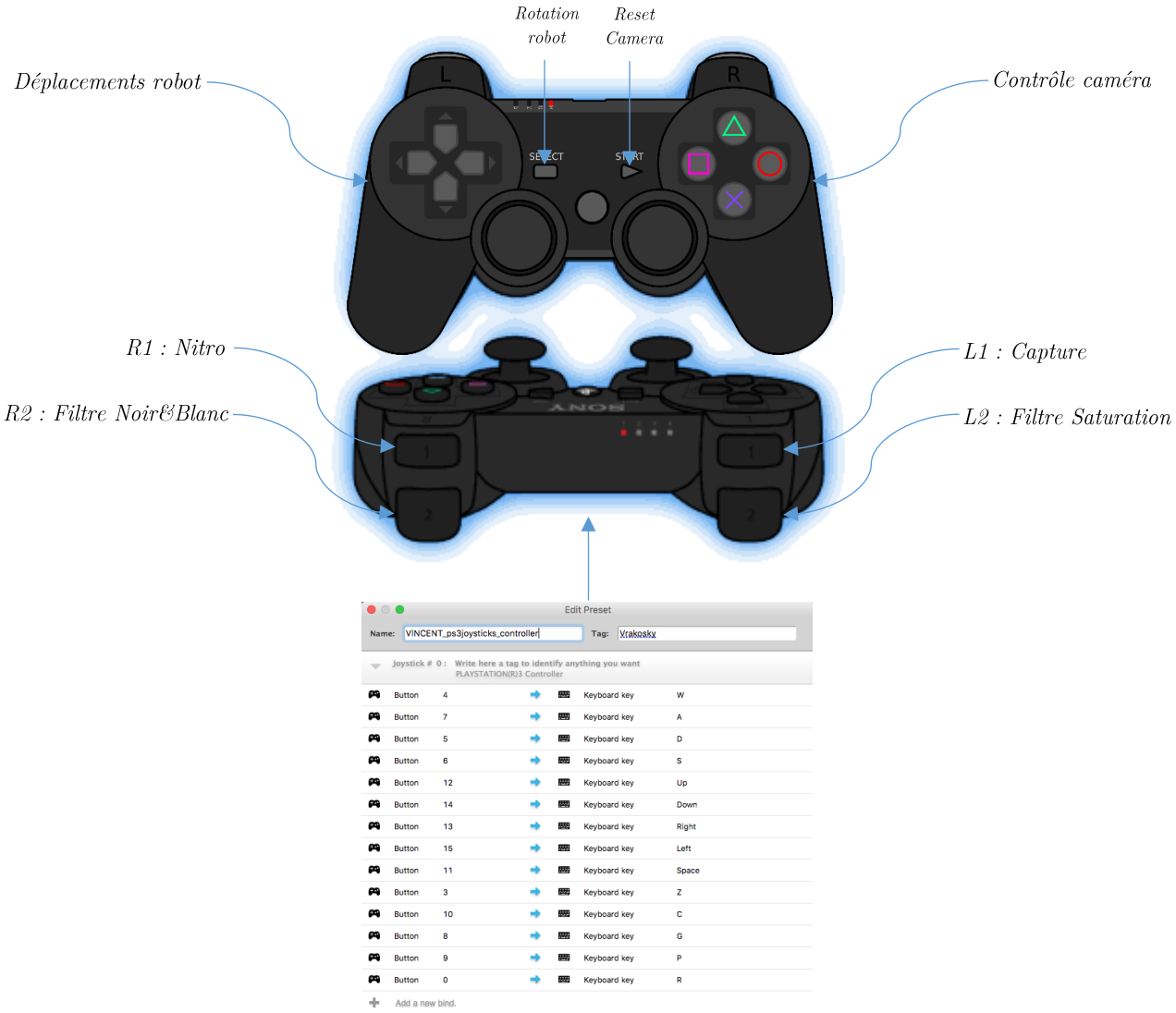
À l'appui de ce bouton, la vitesse est à son maximum, cependant nous avions pensé à récupérer l'ancienne vitesse stockée dans une variable **previousSpeed**, afin de revenir à la vitesse de base lorsque l'utilisateur relâche le bouton d'accélération. Pour cela, nous avions utilisé la méthode événementielle suivante :




```
void MainWindow::keyReleaseEvent(QKeyEvent *ev)
```

3. CONTRÔLE VIA MANETTE PLAYSTATION® 3 DUALSHOCK

Nous avions fait le choix* d'intégrer un moyen de pilotage plus intuitive : *Manette PlayStation® 3 DualShock.* Après installation des drivers correspondant à notre **Controller Playstation**, plusieurs configurations ont été réalisées par rapport à la détection de signaux pour chaque événement créé de la manette (appui de bouton, rotation des joysticks, …) grâce au logiciel **"Joystick Mapper"** spécialement conçu pour les systèmes d'exploitation **MacOS**. Chaque bouton de la manette sera lié à un événement adéquat du clavier :



*D'autres méthodes été possible comme l'implémentation des événements de la manette via la **classe Controller**. Cependant cela n'est pas fonctionnel à 100% par rapport à nos nombreux essais réalisés et ne fonctionne qu'exclusivement sur les manettes XBOX360, d'où notre choix de méthode.

4. LES FILTRES

La gestion des filtres pour le traitement d'image a été mise en place avec succès grâce à la fonction javascript et au filtre CSS déjà précodé. Le code fait appelle à la propriété **filter** comme nous pouvons le voir dans l'exemple ci-dessous, s'agissant d'un filtre noir et blanc :

```
streamCam->page()->runJavaScript(
"var_filtres=document.body.firstChild.style.webkitFilter;document.body.firstChild
.style.webkitFilter = filtres+' grayscale(100%)';");
```

La propriété **filter** permet d'appliquer des filtres et d'obtenir des effets graphiques de flou, de saturation, etc. Les filtres sont généralement utilisés pour ajuster le rendu d'une image, d'un arrière-plan ou des bordures.
Plusieurs fonctions sont incluses dans le **standard CSS** et permettent d'obtenir des effets prédéfinis. L'interface propose en tout 7 filtres en tout ayant pour possibilité de se fusionner, voici une simulation complète :

5. CAPTURE D'ÉCRAN

À tout cela, nous avons implémenté une fonctionnalité permettant la capture d'écran de la webcam et l'enregistrement de l'image. Il suffit pour cela de diriger la Webcam dans le plan désiré, cliquer sur le bouton « CAPTURE » et l'image sera déjà enregistré.

Pour s'y faire, il nous a fallu enregistrer l'image adéquate sous la forme d'un *QPixmap* grâce à la fonction *grab()*, qui fait tout simplement la capture de la zone demandée, ici *streamCamera*.

```
void MainWindow::on_capture_clicked()
{
//Chemin de base : racine
      nowDate = QDateTime::currentDateTime();
      fileNameDate = nowDate.toString("dd_MM_yyyy_hh_mm_ss").toUtf8();
      path = "/Users/esirem/Documents/wifibot2017_VinceAntho/capture/";
      extension = ".png";
//Enregistrement des photos
      pathCapture = path + fileNameDate + extension;
      QPixmap pixMap = streamCamera->grab();
      pixMap.save(pathCapture, "PNG");

   std::cout << "[Log] Capture." << std::endl;
}
```

Les variables `path`, `fileName` et `extension` représentent le chemin, le nom du fichier (on utilise un `QDateTime` afin de nommer les fichiers avec l'heure précise de la capture qui évitera l'écrasement des fichiers précédents. Renommée, via des noms de fichiers distincts et uniques sous l'extension .png, l'image sera enregistrée à la racine du projet plus précisément dans le dossier 'capture'.

VII - FUTURE WORK : PROGRAMMER UNE INTELLIGENCE

Au cours de la réalisation de ce projet, nous avons rencontré un seul problème au niveau des améliorations escomptées : programmer une intelligence artificielle pour le wifibot lab v3.

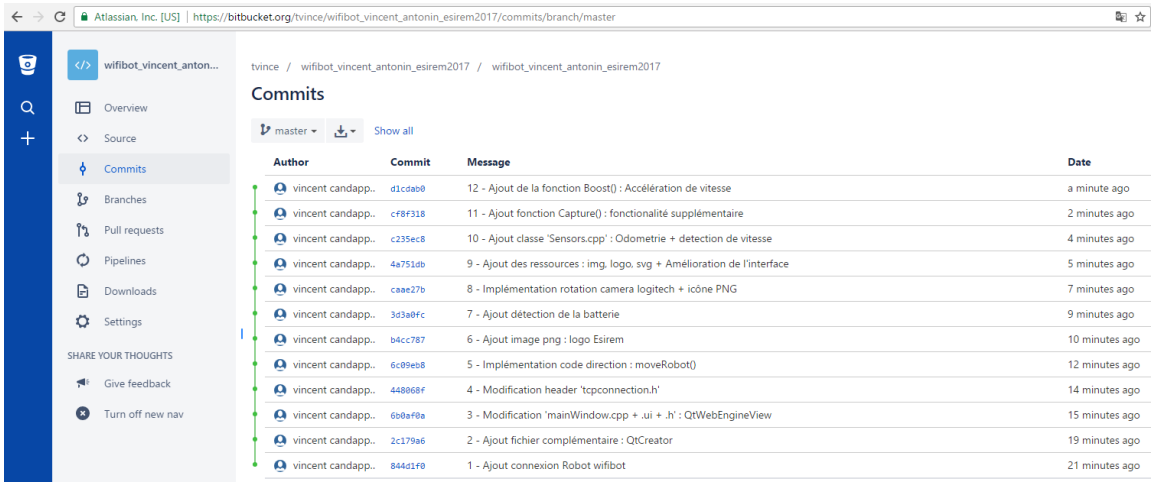
Oui, il s'agissait de l'ultime étape à réaliser, développer une tel capacité nécessite à utiliser les 4 capteurs infrarouges IR (que nous avons réussi à stocker via la classe Sensors.cpp).

Nous avons eu à l'idée d'optimiser le wifibot de façon à la rendre plus intelligente de la manière suivante :

- L'arrêt automatique du robot en cas de détection d'obstacle à moins de 40 cm afin de réduire les dégâts de la machine.
- Ou même la rotation du robot en cas de détection d'objet(s) à proximité et sa redirection dans une zone avec moins d'obstacle.

VIII - PARTIE GESTION : Atlassian SourceTree

Pour la réalisation de ce projet, nous nous sommes réparti les tâches en fonctions de nos connaissances et de nos compétences. Pour cela, nous devons apprendre à programmer à plusieurs, nous partager les tâches, planifier et avancer efficacement le travail, et cela sur le même code. Nous avons donc fait le choix d'utiliser un logiciel de ***versioning*** nous permettant de travailler facilement en groupe : **SourceTree**.



SourceTree est un logiciel de gestion de version et permet donc d'apporter un certain confort lors d'un travail en équipe. Dans le cadre de ce projet nous avons travaillé en binôme et l'utilisation de SourceTree a permis d'éviter au maximum les problèmes que l'on peut rencontrer en groupe, car il permet d'avoir un listing clair et précis de toutes les modifications apportées au code sources ainsi qu'une sauvegarde de ces dernières ce qui permet en cas d'erreur de remonter à une version antérieure du programme qui elle sera opérationnel. Bien évidemment, les **commit** sont choisi intelligemment, seulement les améliorations du code fonctionnelles sont commités. **SourceTree** et capable d'assembler leurs modifications afin d'éviter que le travail d'une personne soit écrasé.

CONCLUSION

Ce projet informatique a eu pour but de travailler sur une application concrète suivant un cahier des charges général avec le but principal suivant :

"commander à distance un wifibot par liaison wifi."

À travers ce projet, plusieurs facteurs bien que technique que théorique sont intervenues. Le fait de nous avoir laissés dans un milieu totalement autonome, nous ont montré qu'il était nécessaire dans un premier temps d'analyser par soi-même les différentes possibilités et fonctionnalités à mettre en place. Le cahier des charges ainsi modélisé nous a beaucoup simplifié les tâches à réaliser afin de pouvoir mener chaque phase du projet.

L'utilisation de Qt Creator afin d'effectuer les développements était très intuitive, a permis la réalisation assez rapide et la mise en place de bon nombre de fonctionnalités très simplement.

Ainsi nous avons pu noter l'importance que peut avoir un logiciel de gestion de version tel que SourceTree dans le développement d'un projet. En effet, ce logiciel minimise et évite problèmes que l'on peut rencontrer lors d'un projet en collaboration et surtout lorsque plusieurs personnes interagissent sur un même travail.

Au final, nous avons pu réaliser et finaliser notre projet wifibot à bien, toute fonctionnalité primaire a été exploitée correctement. Au-delà de tout cela, nous avons su combiner plusieurs de nos connaissances pour améliorer le wifibot.

"Ce fut une expérience remarquable et innovante, nous réaliserons avec plaisir d'autres projets aussi complexes qu'amusants afin de pouvoir contrôler d'autres gadgets utiles pour des utilisations futures"

CANDAPPANE Vincent
JAVELLE Antonin
ESIREM IT 3A
19/06/2017 10:30