PROJET
WIFIBOT V3

CHAU Alexandre
DANAUDIERE Benjamin

Année 2015/2016

Esirem Dijon

UNIVERSITÉ DE BOURGOGNE

Table des matières

Index des illustrations

Introduction

Au cours de notre formation à l'école d'ingénieur l'ESIREM, en 3ème année, un projet de fin d'étude sera mis en œuvre afin d'exploiter l'apprentissage des langues de programmation, notamment le C++ ici. Le projet s'intitule WIFIBOT Version3, qui sera programmé sur QTCreator, un environnement de développement intégré multi-plate forme faisant partie du framework QT.

Pour cela, nous travaillerons en binôme, qui est composé de CHAU Alexandre, et Danaudière Benjamin.Un cahier des charges nous a été imposé. Le but étant de faire fonctionner le robot, pouvoir le faire rouler , et gérer les commandes. Suite à cela, des améliorations peuvent être rajouté : Récuperer la caméra, les données transmis, «jouer» avec les capteurs etc ..

Durant ce projet , nous serons encadrés par Mr Peillon, Mr Herman et ……
Et nous disposerons de quelques semaines pour réaliser à bien ce projet, suite à cela , une présentation oral du travail réalisé.

Dans un premier temps, nous allons voir l'utilisation du logiciel QTCreator, suite à cela, nous allons présenter rapidement le wifibot , puis nous allons finir par voir les parties importantes du code.

Chapitre 1 : Présentation du projet

1.Utilisation de Qt

1.1 Présentation de Qt

Illustration 1 : Qt

Qt est une bibliothèque multiplate-forme pour créer des GUI (interface utilisateur graphique). Qt est écrite en C++, et est à la base conçue pour être utilisé en C++. Toutefois, il est aujourd'hui possible de l'utiliser avec d'autres langages comme Java ou Python.

Qt est bien plus qu'une bibliothèque, c'est un ensemble de bibliothèques. C'est pour cela qu'on parle plutôt de framework, car on dispose d'un ensemble d'outils pour développer les programmes plus efficacement.

1.2. Qt est multiplate-forme

Qt est un framework multiplate-forme. Il est disponible sur la plupart des système d'explitations d'aujourd'hui, Windows, Linux et Mac Os.

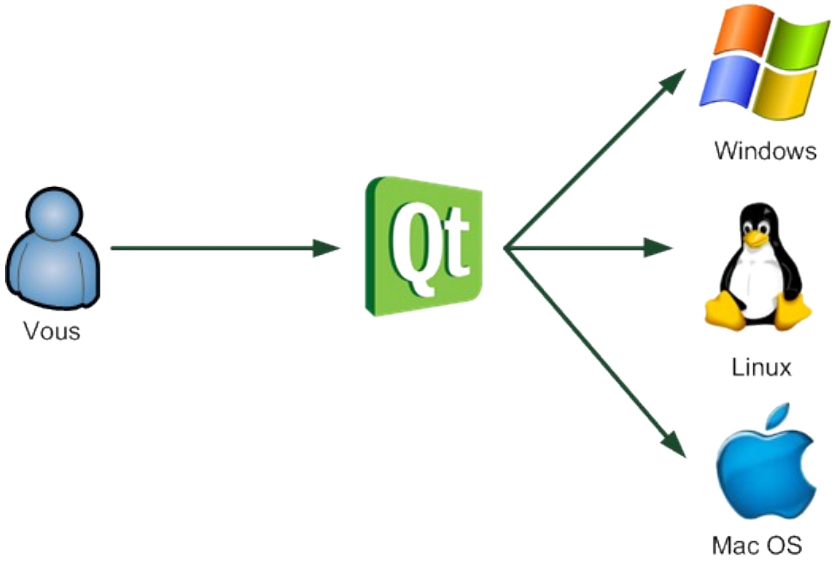

Illustration 2: Qt logiciel multiplate-forme

Grâce à cette technique, les fenêtres qu'on code ont une apparence adaptée à chaque Os. Pour notre part, nous allons utiliser Qt sous Windows.

1.3. La licence de Qt

Qt est distribué sous deux licence au choix : LGPL ou propriétaire. Celle qui nous interesse est la licence LGPL car elle nous permet d'utiliser gratuitement Qt. On peut asussi bien réaliser des programmes libres, c'est a dire des programme dont le code source est public et donc on autorise la modification par d'autres personnes, que des programmes propriétaires.

Mais qui utilise Qt ?

Qt est utilisé par de nombreuses entreprises tels : Adobe, Archos,Boeing,Google,Skype… Qt est utilisée pour réaliser de nombreuses GUI, comme celle d'Adobe photoshop Elements, ou bien de Google Earth.

1.4. Qt Creator

Qt Creator est un programme tout-en-un qui comprend entre autres :

- Un IDE pour developper en C++, optimisé pour compiler des projets utilisant Qt
- un éditeur de fenetre, qui permet de desiner facilement le contenu des interfaces à la souris

0149 une documentation indispensable pour tout savoir sur Qt.

Voici a quoi ressemble Qt Creator lorsqu'on lance le programme

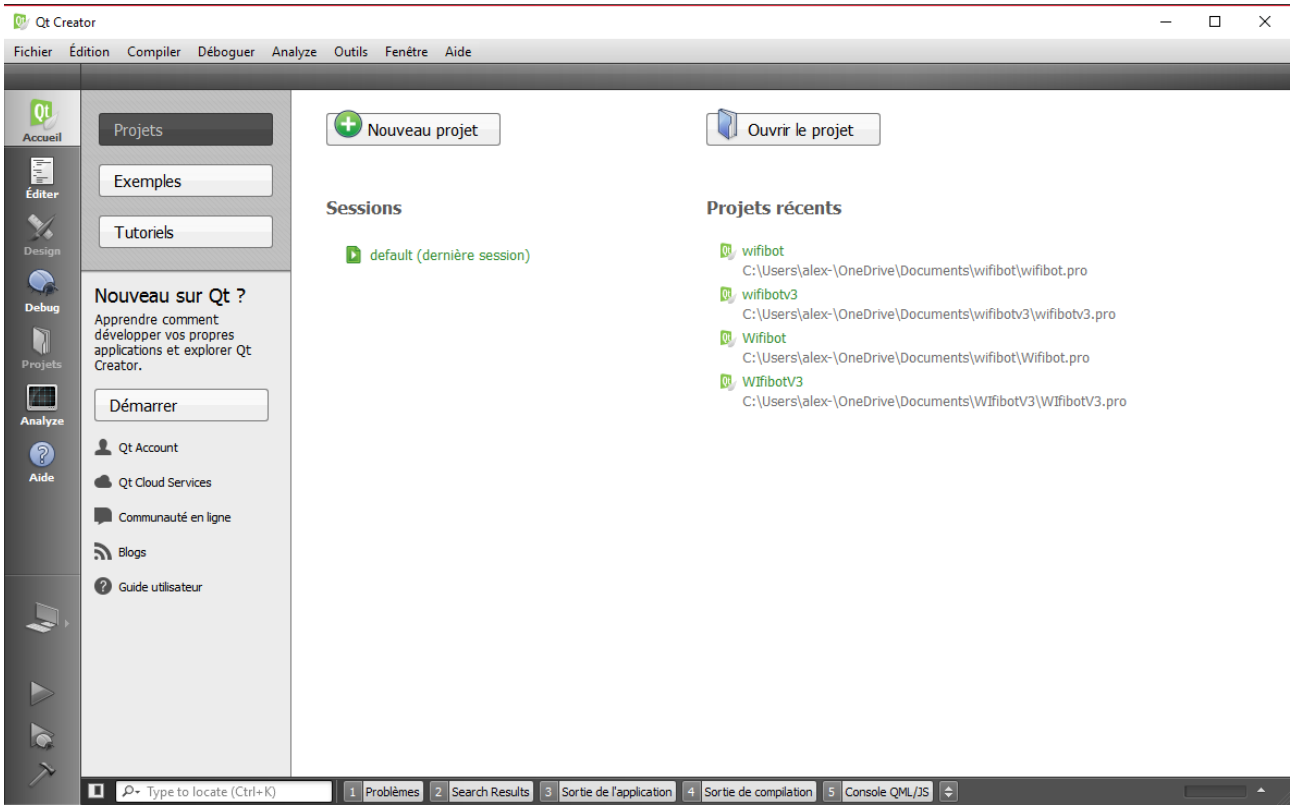

Illustration 3: Qt Creator

Remarques : Qt Creator est facile d'utilisation, cependant avant de l'utiliser, il est necessaire de lire quelques documentations. Car en effet, il existe des objets propres a Qt, comme par exemple pour l'utilisation de socket. (pour la communication client/serveur)

Il nous a fallut plusieurs heures pour s'adapter au logiciel, voir son fonctionnement. Il a été nécessaire de se tourner vers Openclassroom pour justement comprendre son principe, son utilisation , sa création d'une fenêtre.

Donc c'est sous ce logiciel, que nous allons coder afin de faire fonctionner le Wifibot version 3. Pour rappel, l'objectif est de pour le faire rouler, et le contrôler, obtenir ses relevés (vitesses, capteurs, caméra etc..)

Chapitre 2 : Le robot Wifibot v3

1. Wifibot

Illustration 4: logo Wifibot

Le wifibot Lab est une plate forme robotique modulaire qui permet 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 infra rouge. Le châssis du robot est contrôlable en utilisant un port RS232. L'unité de calcule embarquée qui envoie les commandes au robot est une carte industrielle Intel Atom D510 double cœurs au format 3,5 pouces avec une image du système d'exploitation XP embedded SP3. Une carte wifi assure la liaison sans fil au système avec le point d'Accès configuré fourni gratuitement. Ainsi les utilisateurs peuvent modifier ou concevoir des programmes directement sur le robot (VGA ou par WIFI) (voir annexe)

Illustration 5: Wifibot lab v3

2. Diagramme de Gantt

Nous disposons de 11 séances de 2h, soit 22h théoriques afin de mener à bien ce projet. Celui ci a débuté le 22/01/2016 et ce finit le 29/04/2016 par une présentation oral du travail réalisé.

Nous n'avons pas réellement de contrainte de temps, d'objectif. C'est un projet indépendant, donc nous devons gérer notre temps, et notre avancé. C'est pourquoi je n'ai pas affiché les contraintes lié au temps.

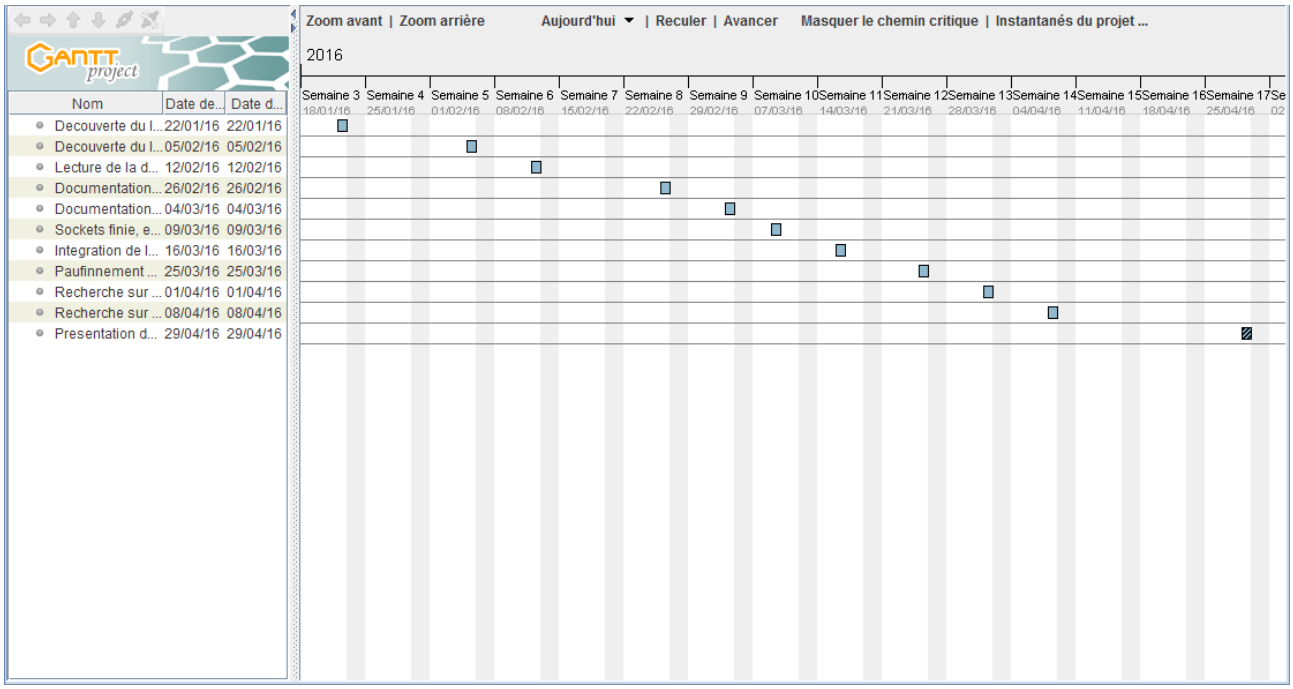

Illustration 6: Diagramme de Gantt

3. Diagramme de PERT

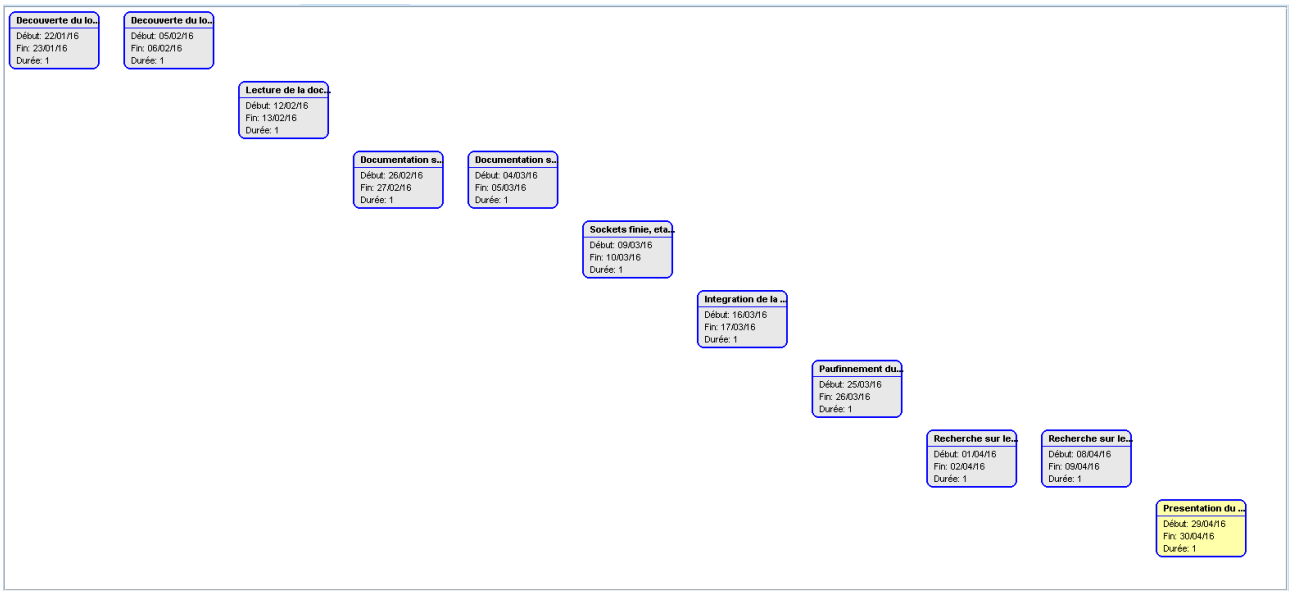

Illustration 7: Diagramme de Pert

Le diagramme ci dessus, représente les actions réalisés au cours des séances. Un fichier joint sera mis à disposition pour une meilleur lecture des données.

4. Cahier des charges

L'objectif de se projet est de :

- Pouvoir se connecter au robot
- Pouvoir contrôler le wifibot, à l'aide de commande
- Pouvoir recevoir les informations transmises par le robot (vitesse, capteur, batterie etc..)

Bien sur, des options sont possibles, comme :

- Avoir la caméra et pouvoir la contrôler
- « Jouer » avec les capteurs pour se déplacer, a l'aide de checkpoint
- Pouvoir contrôler le robot a l'aide d'un joystick.

5. Connexion au robot

Il existe deux modes pour se connecter au réseau. Toute les deux basées sur le protocole ETHERNET , filaire ou par wifi.

Le robot a pour adresse IP : 192.168.1.106.
Une fois le routeur branché et le robot allumé, nous pouvons nous connecter a celui ci via un routeur nommé «wifibotlab»

6. La webcam

Afin d'obtenir l'image de la webcam du wifibot, il suffit d'abord de se connecter au réseau «wifibotlab» puis ensuite , sur un navigateur web (Mozilla, Chrome) , il faut taper dans la barre d'adresse IP, l'adresse IP du robot suivis du numero port , par default :8080. Dans le cas présent, il a fallut taper :
"http://192.168.1.106:8080"

Néanmoins, c'est une image fixe. On ne peut contrôler la webcam. Ainsi, pour pouvoir la contrôler, une petite astuce consiste a rajouter en plus de l'adresse IP+ le port , une suite de commande qui permet de faire bouger la caméra. (cf code) . Les caractéristiques de la webcam se trouveront en annexe.

Chapitre 3 : Le code

Nous pouvons découper le code en 4 grandes parties :

- La connexion/déconnexion du robot
- Envoie de trames pour les commandes du robot
- Récupération de l'image de la webcam + déplacement de la webcam
- La fonction main

1. Connexion/Déconnexion du robot

Pour la connexion/déconnexion du robot, nous avons utilisé les sockets propre au logiciel Qt. (Car on pouvais aussi passé par les sockets normales (Sockaddr). Il est plus facile à utiliser et à instancier.

Pourquoi TCP ?

Nous avons pris le choix de prendre une connexion via wifi, plus exactement avec le protocole TCP, afin de s'assurer d'un bon transfert de donnée, et avoir un accusé de réception. Ainsi, lors de la connexion, on envoie un acquittement, pour demander si on peut se connecter. Celui ci, renvoie l'acquittement (positive ou négative). Si celui est positif, on a fait en sorte de mettre un message, comme quoi nous sommes connectés. Sinon, il y a une erreur de compilation.

Nous avons aussi fait en sorte, de synchroniser le tout, avec l'interface graphique. En créant deux boutons : Connexion et Déconnexion.

Ainsi lorsqu'on clique le bouton connexion, on fait en sorte aussi que la webcam s'allume, et qu'on obtient l'image de la webcam.

Lorsqu'on clique le bouton Déconnexion, nous coupons la connexion avec le robot.

```
/**** Connexion au robot ****/
void wifibotv3::connexionviaTCP(QString ip,int port)
{
     cout << "connexion au robot" << endl;
     socket.connectToHost(ip, port);
     connect(&socket, SIGNAL(connected()),this,
SLOT(acquittement_de_la_connexion()));
     cout << "connecté" << endl;
}

/**** Gestion de la connexion ( ui ) ****/
void wifibotv3::on_connect_clicked()
```

```
{
    socket.connectToHost(ui->AdresseIp->text(),15020);
    connect(&socket, SIGNAL(connected()),this, SLOT(Connexion()));
    connect(&socket, SIGNAL(disconnected()),this, SLOT(Disconnexion()));
    timer→start();

}

/**** Si on est connecté,on active la camera + Annonce qu'il est connecté****/
void wifibotv3::Connexion()
{

    ui->AdresseIp->setText("Connecté !");
    QString webcam = "http://192.168.1.106:8080/javascript_simple.html";
    ui->cam->load(QUrl("http://192.168.1.106:8080/javascript_simple.html"));
    ui->cam->show();
    timer->start();

}

/**** Envoie un signal d'acquittement pour la connexion ****/
void wifibotv3::acquittement_de_la_connexion()
{
    emit vers_IHM_acquittement_connection();
    cout <<"connexion établie "<< endl;
}

/**** Bouton pour se déconnecter ****/
void wifibotv3::on_Disconnexion_clicked()
{
    socket.disconnectFromHost();
    cout << "deconnexion du robot" <<endl;
}

void wifibotv3::Disconnexion()
{
   ui->AdresseIp->setText("Déconnecté");
}
```

Illustration 8 : Partie d'un code concernant la connexion / déconnexion

2. Envoie de trames pour les commandes du robot

Nous nous sommes focalisés sur la documentation donnée dont voici ci dessous (A noter que la documentation est principalement en anglais.) :

TCP :

The computer-WIFIBOT control interface protocol:
Here the communication takes place through one TCP socket. The robot is the server and the interface the client. The communication channel is through robot's port 15020 and unlike the other protocol it is used for both sending and receiving data.
Messages to port 15020:

You need to send 9 char using a TCP socket:

Char 1 is 255
Char2 is size (here is 0x07)
Char 3-4 is the left speed 0 -> 240 tics max
Char 5-6 is the right speed 0 -> 240 tics max
Char 7 is the Left / Right speed command flag : Forward / Backward and speed control left & right ON / OFF.
Char 7 is decomposed as follow (1 byte char -> 8 bits) :

(128) Bit 7 Left Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(64) Bit 6 Left Side Forward / Backward speed flag ::1 -> Forward / 0 -> Reverse
(32) Bit 5 Right Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(16) Bit 4 Right Side Forward / Backward speed flag :: 1-> Forward / 0 -> Reverse
(8) Bit 3 PID speed 0 is 50 ms 1 is 10 ms (50 ms is recommended)
(4) Bit 2 Not Used (Relay 3 On/Off (future option))
(2) Bit 1 Not Used (Relay 2 On/Off (future option))
(1) Bit 0 Not Used Relay 1 for CPU or sensors. On/Off: 0 is OFF 1 is ON (DSUB9 POWER Pin 3)

Char 8-9 is the CRC 16 bits (char 7 low char 8 high, see end
of document for details) not used in TCP

So to control for example the left side and the right side (Left & Right 2 motors, 2 encoders) :
The speed is between 0-240

If we want the left side & right side to move at speed
120 forward without motor control we send:
Char 1 is 255
Char2 is 0x07
Char 3 is 120
Char 4 is 0
Char 5 is 120
Char 6 is 0
Char 7 is 80 (0 + 64 + 0 + 16)
Char 8 – Char 9 = CRC16(data) not used in TCP

Il est indiqué ici que sur les char 8 et 9, qui correspond au CRC16 qu'il n'est pas necessaire pour le protocole TCP. Or il nous a fallut l'utiliser pour le bon fonctionnement.

Indication :

char 3-4 : concerne la vitesse gauche
char 5-6 : concerne la vitesse droite
char 7 : flag qui concerne si on avance ou on recule, coté gauche ou coté droit , ou les deux.

Voici la fonction réaliser pour le contenu des trames :

```
/**** Trame pour communiquer le déplacement du robot****/
void wifibotv3::Tramemove(quint8 Goche, quint8 Droit, int direction)
{

    trame.clear();
    trame.append((char)0xff);
    trame.append((char)0x07);
    trame.append((quint8)Goche);
    trame.append((char)(0x00));
    trame.append((quint8)Droit);
    trame.append((char)(0x00));
    if(direction==0){
        trame.append((char)0x00);
    }
    else if(direction==1){
        trame.append((char)80);
    }
    else if(direction==2){
        trame.append((char)64);
    }
    else if(direction==3){
        trame.append((char)16);
    }

   // bit 7 et 8 CRC
    quint16 crc = Crc16(&trame, 1);
    trame.append((char)crc);
    trame.append((char)(crc>>8));

}

/**** Definition du CRC ****/

quint16 wifibotv3::Crc16(QByteArray* byteArray, int pos){
    unsigned char *data = (unsigned char* )byteArray->constData();
    quint16 crc = 0xFFFF;
    quint16 Polynome = 0xA001;
    quint16 Parity = 0;
    for(; pos < byteArray->length(); pos++){
        crc ^= *(data+pos);
        for (unsigned int CptBit = 0; CptBit <= 7 ; CptBit++){
            Parity= crc;
            crc >>= 1;
            if (Parity%2 == true) crc ^= Polynome;
        }
    }
    return crc;
```

```
}
```

Illustration 9 : Partie d'un code concernant les trames a envoyés pour les commandes

De même, il a fallut synchroniser les boutons pour les commandes du robot, et le code.
Ainsi, nous avons relié les boutons à un code, qui réalise la fonction. (avancer , reculer, tourner a droite , tourner a gauche) .
Le CRC16, a été mis sur les conseils du professeur surveillant. Vaut mieux le mettre, cela n'engendre pas d'erreur. (cf annexe)

```
/******  Gestion mouvement de direction ( lorsqu'on reste appuyé ) ******/
void wifibotv3::on_BoutonHaut_pressed()
{
    if(capteur1==0)
    wifibotv3::Tramemove(ui->vitesse->value(),ui->vitesse->value(),1);
    else{
         wifibotv3::Tramemove(0,0,1);
    }
}

void wifibotv3::on_BoutonGauche_pressed()
{

    wifibotv3::Tramemove(ui->vitesse->value(),ui->vitesse->value(),3);
}

void wifibotv3::on_BoutonDroite_pressed()
{

    wifibotv3::Tramemove(ui->vitesse->value(),ui->vitesse->value(),2);
}

void wifibotv3::on_BoutonBas_pressed()
{
     if(capteur2==0)
    wifibotv3::Tramemove(ui->vitesse->value(),ui->vitesse->value(),0);
     else
     {
            wifibotv3::Tramemove(0,0,0);
     }
}
/****** Gestion mouvement de direction ( lorsqu'on relache ) ******/

void wifibotv3::on_BoutonHaut_released()
{
    wifibotv3::Tramemove(0,0,1);
}

void wifibotv3::on_BoutonDroite_released()
{
   wifibotv3::Tramemove(0,0,2);
}

void wifibotv3::on_BoutonGauche_released()
{
    wifibotv3::Tramemove(0,0,3);
}
```

```
void wifibotv3::on_BoutonBas_released()
{
    wifibotv3::Trammove(0,0,0);
}
```

Illustration 10 : Partie d'un code concernant les boutons commandes pour le déplacement

La partie la plus difficile, étant de synchroniser le tout, avec l'interface graphique qui a engendré pas mal d'erreur..

3. Récupération de l'image de la webcam + déplacement de la webcam

Encore une fois, il a fallut être astucieux pour récupérer l'image de la webcam. On a pris le choix de via son adresse IP + le numero de port 8080 + un code supplémentaire afin de commander le mouvement de la caméra.
Nous avons également, utiliser les fonctions Qt, notamment QnetworkAccesManager come son nom l'indique pour avoir accès a la webcam, et la fonction QNetworkRequest qui permet de récupérer l'image de la webcam, via l'URL.

```
 //Caméra
    Camera = new QNetworkAccessManager(this);
    Haut="/?action=command&dest=0&plugin=0&id=10094853&group=1&value=-200";
    Droite="/?action=command&dest=0&plugin=0&id=10094852&group=1&value=-200";
    Gauche="/?action=command&dest=0&plugin=0&id=10094852&group=1&value=200";
    Bas="/?action=command&dest=0&plugin=0&id=10094853&group=1&value=200";


/******** Gestion des boutons pour la direction de la caméra *********/
void wifibotv3::on_camversleHaut_pressed()
{
    QUrl url("http://192.168.1.106:8080"+Haut);
            Camera->get(QNetworkRequest(url));
}

void wifibotv3::on_camversleBas_pressed()
{
    QUrl url("http://192.168.1.106:8080"+Bas);
            Camera->get(QNetworkRequest(url));
}

void wifibotv3::on_camverslaGauche_pressed()
{

    QUrl url("http://192.168.1.106:8080"+Gauche);
            Camera->get(QNetworkRequest(url));
}

void wifibotv3::on_camverslaDroite_pressed()
{
```

```
    QUrl url("http://192.168.1.106:8080"+Droite);
            Camera->get(QNetworkRequest(url));
}
```

Illustration 11 : Partie d'un code concernant la récupération de l'image de la webcam

4. La fonction main

La fonction main ici , est très court, et très simple. Car en fait, nous avons utiliser une interface graphique propre à Qt, (QtApplication). Ainsi, pour l'interface graphique, nous n'avons que choisi l'emplacement des boutons, les séparateurs , les capteurs pour recevoir la vitesse, ou bien un «levier» pour mettre plus d'intensité à la vitesse.
Le plus dur étant de synchroniser le tout (les boutons etc..) avec le code.

```
#include <iostream>
#include "wifibotv3.h"
#include <QApplication>

using namespace std;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    wifibotv3 fenetre;

    fenetre.show();
      return a.exec();
}
```

Illustration 12 : La fonction main

Conclusion

La programmation du wifibot a été très enrichissent dans le sens où il y a énormément de chose possible à programmer là dessus, comme le traitement d'image, le déplacement via les capteurs infrarouges, ou bien une assistance au pilotage etc..) .
Cela à aussi permis d'exploiter nos acquis en programmation C++ ,travailler en autonome, chercher ses propres erreurs, chercher à comment trouver une solution pour mener a bien le projet (notamment a bien gérer le logiciel)

Remerciement

Je remercie les professeurs encadrants pour l'aide apporté. Notamment dans les sockets, où le début était assez compliqué.

Annexes

SET SPEED to move (Send periodically at max 250ms, else Timeout occurs and speed is set to 0):

You need to send 9 char at 19200 bauds.
Char 1
is 255
Char2
is size (here is 0x07)
Char 3-4
is the left speed 0 -> 240 tics max
Char 5-6
is the right speed 0 -> 240 tics max
Char 7
is the Left / Right speed command flag : Forward / Backward and speed control left & right ON / OFF.
Char 7 is decomposed as follow (1 byte char -> 8 bi ts):
(128) Bit 7 Left Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(64) Bit 6 Left Side Forward / Backward speed flag :: 1 -> Forward / 0 -> Reverse
(32) Bit 5 Right Side Closed Loop Speed control :: 1 -> ON / 0 -> OFF
(16) Bit 4 Right Side Forward / Backward speed flag :: 1 -> Forward / 0 -> Reverse
(8) Bit 3 PID speed 0 is 50 ms 1 is 10 ms (50 ms is recommended)
(4) Bit 2 Not Used (Relay 3 On/Off (future option))
(2) Bit 1 Not Used (Relay 2 On/Off (future option))
(1) Bit 0 Not Used Relay 1 for CPU or sensors. On/ Off: 0 is OFF 1 is ON (DSUB9 POWER Pin 3)
Char 8-9
is the CRC 16 bits (char 7 low char 8 high, see en d of document for details)
So to control for example the left side and the rig ht side (Left & Right 2 motors, 2 encoders):
The speed is between 0-240
If we want the left side & right side to move at sp eed 120 forward without motor control we send:
Char 1 is 255
Char2 is 0x07
Char 3 is 0
Char 4 is 120
Char 5 is 0
Char 6 is 120
Char 7 is 80 (0 + 64 + 0 + 16)
Char 8 – Char 9 = CRC16(data)

To GET DATA from the chassis:

The DSPIC sends you 21 chars in continuous at 10ms:

```
//DSPIC C code to show you the sending part to the
PC:
bufsend[0]=(-speedlab);
bufsend[1]=(-speedlab >> 8); //speed is a short and
it is tics / 50 ms
bufsend[2]=(
unsigned
char
)(tmpadc2 >> 2);
//Bat Volt:10.1V 1.28V 404/4->101
bufsend[3]=(
unsigned
char
)(tmpadc4 >> 2);
//3.3v->255 2v-> 624/4 -> 156
bufsend[4]=(
unsigned
char
)(tmpadc3 >> 2);
//3.3v->255 2v-> 624/4 -> 156
bufsend[5]=bufposition[0];
//Acumulated odometrie is a float
bufsend[6]=bufposition[1];
//12ppr x 4 x 51 gear box = 2448 tics/wheel turn
bufsend[7]=bufposition[2];
bufsend[8]=bufposition[3];
bufsend[9]=(speedlab2);

bufsend[10]=(speedlab2 >> 8);
bufsend[11]=(
unsigned
char
)(tmpadc0 >> 2);
bufsend[12]=(
unsigned
char
)(tmpadc1 >> 2);
bufsend[13]=bufposition2[0];
bufsend[14]=bufposition2[1];
bufsend[15]=bufposition2[2];
bufsend[16]=bufposition2[3];
bufsend[17]=0;//robot current
// *0.194-37.5 = I in Amp / * 10 for the GUI
: 10 -> 1 A (ACS712 30Amp chip) // *0.129-25 =I (fo
r ACS712 20A chip)
bufsend[18]=14;
//firmware version
bufsend[19]=crc16 low;
bufsend[20]=crc16 high;
```