Rapport de Projet
WifiBot

Clément BOISSARD
Imad BOUFENCHOUCHE
3e année ITC
Semestre 2, année 2017-2018

ESIREM

Table des matières

Introduction

Ce compte-rendu rapporte le travail et les avancées réalisées au cours des séances de travaux pratique pour le projet WifiBot. Qui consiste en la réalisation d'une application, généralement une application bureau, permettant la manipulation du robot wifibot et de la récupération des informations.

Avant de commencé le développement de l'application il a fallut prendre en main l'outil de développement `Qt` et le langage de programmation `C++` avec lequel il faut programmer.

Nous avons commencé le développement de l'application par la réalisation de l'interface graphique avec les boutons et les informations du robot.

Puis nous avons essayer de mettre en place la manipulation de la caméra et de son déplacement pour voir autour du robot.

Ensuite nous avons essayer de mettre en place la manipualtion du robot en lui même en commandant ses 4 roues motrices.

Chapitre 1

L'interface Graphique

1.1 Introduction

Pour notre application il fallait une interface graphique afin de pouvoir manipuler le robot grâce à celle-ci mais aussi d'avoir des information sur ce dernier ainsi que le retour caméra de la webcam du robot.

1.2 Conception

Nous avons fait le choix, pour prévoir une possible future version mobile au format portrait, d'avoir une interface graphique divisée en 3 parties alignées les unes en dessous des autres.

La premiere partie contiendrait les informations du robots et les boutons de connexions notamment. Cette partie de la fenêtre est la moins accessible, sur mobile en tout cas, c'est pourquoi nous y avons placer les éléments ne nécessitant que peut d'intéraction utilisateur ou de manière occasionnelle.

La deuxième partie est celle qui contient le retour vidéo de la caméra et ce doit donc d'être centré car est la partie la plus intéressante à regarder. C'est aussi la partie la plus importante au niveau taille de l'application.

La troisième et dernière partie est celle qui contient les contrôle utilisateur permettant de controler le robot et la caméra. Cette partie se situe au bas de l'application car elle necessite une interaction avec l'utilisateur continuelle. De plus dans le cadre de l'utilisation mobile, il est logique de mettre les éléments en bas pour ne pas que le reste soit caché par les doigts de l'utilisateur.

Pour les éléments de contrôles nous souhaitions utiliser des "joysticks" comme il peut y en avoir sur les jeux vidéos mobiles. En effet ce sont des éléments graphiques intérectifs pour l'utilisateur.

De plus dans le cas d'une utilisation bureatique classique il est possible de faire le choix d'utiliser les touches du clavier pour contrôler le robot. Les flêches directionnelles permettent de bouger la caméra et les touches `‘Z‘`, `‘Q‘`, `‘S‘` et `‘D‘` pour le déplacement du robot.

1.3 Développement

Le développement de l'interface peut se faire dans le code C++ avec l'instanciation d'objet de type widget. Mais il aussi possible de le faire dans un fichier `‘.ui‘` dans lequel les élements graphiques sont mis en place au format `XML`.

1.3.1 Elements de base

Pour découper les trois parties nous avons utiliser le `Widget ‘QVBoxLayout‘` qui est un élément qui dispose sont contenu de manière verticale c'est-à-dire qu'à chaque ajout d'un widget il est ajouté en dessous des autres.

Nous utilisons des widgets de base tels que des `‘QPushButton‘` pour les boutons, des `‘QProgressBar‘` pour des bars de progression pour l'autonomie restante du robot par exemple, des `‘Spacer‘` qui sont des widget très intéressant qu'on insere entre deux widgets pour les séparer et les écarter.

Pour la partie vidéo on utilise un widget `‘QFrame‘` qui nous permettra de mettre du contenu vidéo à partir d'une url.

1.3.2 Les joysticks

La mise en place des joysticks fut assez compliqué car il n'existe pas de manière native ce genre d'éléments dans Qt, il a donc fallut en trouver réalisé par la communauté de développeur. Le seul code de widget ressemblant aux joysticks que nous désirions était des `‘JoyPad‘` disponible dans le github du développeur.

Pour mettre les JoyPads dans le design de l'interface graphique il a fallut ajouter 2 fois l'arbre XML suivant au endroit désirés :

Programme 1.1 – JoyPad XML

```
<item>
    <widget class="JoyPad" name="joypad" native="true">
        <property name="minimumSize">
            <size>
                <width>250</width>
                <height>0</height>
            </size>
        </property>
    </widget>
</item>
```

Il était aussi important de préciser au compilateur ce qu'est `‘JoyPad‘` en ajoutant ces lignes à la fin du fichier :

Programme 1.2 – JoyPad Custom Widget

```
1 <customwidget>
2     <class>JoyPad</class>
3     <extends>QWidget</extends>
4     <header>joypad.h</header>
5     <container>1</container>
6 </customwidget>
```

Une fois cela terminé, la réalisation de l'interface graphique était minimale afin de pouvoir avoir une futur application bureau permettant de commander le robot Wifibot.

Chapitre 2

La caméra

2.1 Introduction

L'une des fonctionnalités les plus intéressantes est l'utilisation de la caméra IP du Robot Wifibot. En effet il est très intéressant de pouvoir récupérer le flux vidéo que la caméra capte mais aussi il est nécessaire de pouvoir piloter cette caméra de manière à la faire tourner sur elle même suivant deux axes, un vertical et un horizontal.

2.2 Fonctionnement

La manipulation de la caméra s'éffectue par une connexion avec le protocole applicatif `HTTP` en effet un serveur web est en place au sein de la caméra est traite les requêtes HTTP. De cette manière il est possible de faire bouger la caméra du robot sur ses deux axes mais aussi de récupérer le flux vidéo.

Pour le contrôle de la caméra nous avons un `'JoyPad'` prévu à cet effet, celui de droite il faut donc que suivant la position de son joystick à l'intérieur que la caméra se déplace.

2.3 Développement

Le développement de la partie qui gère la caméra et la première par laquelle nous avons véritablement commencé en programmation C++.

2.3.1 Rotation Caméra

Il est facile de récupérer les évenements de mouvement du joypad grâce à une connection d'un « signal », mis en place par le développeur du widget, à un « slot » à qui nous permet de savoir lorsque le joystick du joypad à sa position `x` ou `y` a changé.

Pour connecter un signal à un slot il faut utiliser la fonction Qt `'connect'`.

C'est donc ce que nous faisons en connectant le signal `'xChanged'` et `'yChanged'` du joypad de notre interface graphique manipulant la caméra au slot d'une classe de cette manière :

Programme 2.1 – Evenement joypad caméra

```
// Connexion evenement du joypad sur l'axe x
connect(ui->widget1, &JoyPad::xChanged, this, [this](float x){
    joypadRobot(x, ui->widget1->y());
});
// Connexion evenement du joypad sur l'axe y
connect(ui->widget1, &JoyPad::yChanged, this, [this](float y){
    joypadRobot(ui->widget1->x(),y);
});
```

On peut voir ici que nous appellons un slot « anonyme » c'est-à-dire que ce n'est pas une méthode, et que cette méthode slot est composé d'une ligne appelant notre fonction prenant les coordonnnées `x` et `y` du joystick.

Cette fonction traite les coordonnées afin de choisir le déplacement à effectuer ainsi que sa vitesse :

Programme 2.2 – Traitement évenements joypad caméra

```
void MainWindow::joypadCamera(float x, float y){
    qDebug() << "Camera : x = " << x << ", y = " << y ;
    float xAbs = x < 0 ? x * -1 : x, yAbs = y < 0 ? y * -1 : y;

    if(x < 0 && xAbs >= yAbs)
        turnCamera(Direction::rightward,xAbs);
    else if(x > 0 && xAbs >= yAbs)
        turnCamera(Direction::leftward,xAbs);

    if(y < 0 && yAbs >= xAbs)
        turnCamera(Direction::backward,yAbs);
    else if(y > 0 && yAbs >= xAbs)
        turnCamera(Direction::forward,yAbs);

    if(x == 0 && y == 0) turnCamera(Direction::none,0);
}

void MainWindow::turnCamera(Direction direction, float speed){
    output->moveCamera(direction,speed);
}
```

Cette fonction calcul donc dans quelle direction est ce qu'il faut faire tourner la caméra et à quelle vitesse et appelle la méthode `‘moveCamera(Direction direction, float speed)‘` de la classe commandant le robot.

C'est cette fonction ensuite qui va envoyer les requêtes HTTP à la caméra pour la manipuler :

Programme 2.3 – Déplacement caméra

```
void RobotOutputManager::moveCamera(Direction direction, float speed){
    int value = 200*speed, id = 0;
    switch(direction){
    case Direction::forward :
        value *= -1;
        id = 53;
        break;
```

```
    case Direction::backward :
        id = 53;
        break;
    case Direction::rightward :
        value *= -1;
        id = 52;
        break;
    case Direction::leftward :
        id = 52;
        break;
    default : return;
    }
    string str = "http://192.168.1.106:8080/?action=command&"
                +"dest=0&plugin=0&id=100948";
    str.append(to_string(id));
    str.append("&group=1&value=");
    str.append(to_string(value));
    request.setUrl(QUrl(QString::fromStdString(str)));
}
```

2.3.2 Retour Caméra

Ensuite nous souhaitons avoir le retour caméra sur la QFrame.

Programme 2.4 – Retour caméra

```
void MainWindow::cameraStream(QString ip, QString port){
    QUrl url = QUrl("http://"+ip+":"+port+"/?action=stream");
    qDebug() << url;
    QWebEngineView *view = new QWebEngineView(ui->video);
    view->load(url);
    view->show();
}
```

Ce code affiche le retour vidéo sur la QFrame très simplement depuis l'url en effectuant une requête HTTP.

La méthode ‘`cameraStream`‘ est appelée au lancement de l'application et à chaque fois qu'une connexion est effectué afin de prevoir le changement d'adresse Ip et/ou de port.

2.4 Conclusion

Après avoir réalisé ce code il était possible de faire bouger la caméra et de voir à travers elle.

Pour autant un problème se pose en effet le mouvement du joystick sur le joypad est très précis et donc lors d'un déplacment de nombreuses positions sont retenues et donc de nombreuses requêtes HTTP sont effectuées sur la caméra et on lui demande donc de bougé énormément en un très faible lapse de temps qui lui est bien plus important pour réaliser tout les déplacement demandés il arrive donc fréquement

qu'elle se bloque sur le côté visé et force alors qu'elle est déjà au maximum, ce qui est assez inquiétant quant à l'état de la caméra.

C'est notamment pour cette raison que les flêches directionnelles du clavier sont paramètrées pour déplacer aussi la caméra.

Car c'est plus facile à manipuler de cette manière.

La récupération n'a pas été mis en place encore à l'heure de l'écriture de ce rapport mais était en cours.

Chapitre 3

Le déplacement du Robot

3.1 Introduction

Après la gestion de la caméra terminée, ou presque, on souhaitait pouvoir manipuler le robot en lui même est donc c'est déplacement, pour cela il faut être capable ne manipuler la mise en rotation des roues par l'intermédiaire de leur moteur.

3.2 Fonctionnement

Si l'on souhaite que le robot avance il faut que les roues tournes dans le sens avant et à l'inverse pour que le robot recule il faut faire tourner toutes les roues dans le sens opposé.

Mais pour tourner il faut que les 2 roues de chacun des deux côtés du robot tourne dans le sens opposé.

Pour pouvoir contrôler la mise en rotation des moteurs, il faut lui envoyer des messages par protocole internet en mode connecté TCP/IP.

Les informations transimises indiquent le sens de rotation des roues, horaire ou anti-horaire, ainsi que leur vitesse de rotation.

De la même manière que pour la caméra il y a un joypad permettant à l'utilisateur de gérer la mise en mouvement du robot, c'est donc le joypad de gauche.

3.3 Développement

Pour contrôler le robot, il faut, de la même manière que pour contrôler la caméra, récupérer les informations liées au joypad désiré.

Programme 3.1 – Evenement joypad robot

```
1    connect(ui->widget1, &JoyPad::xChanged, this, [this](float x){
2        joypadRobot(x, ui->widget1->y());
3    });
4    connect(ui->widget1, &JoyPad::yChanged, this, [this](float y){
5        joypadRobot(ui->widget1->x(),y);
6    });
```

Le code est similaire à celui de la caméra sauf que cette fois on appelle une autre méthode pour traiter les coordonnées.

Programme 3.2 – Traitement évenements joypad robot

```
1  void MainWindow::joypadRobot(float x, float y){
2      qDebug() << "Robot x: " << x << " y: " << y;
3      float xAbs = x < 0 ? x * -1 : x, yAbs = y < 0 ? y * -1 : y;
4
5      if(x < 0 && xAbs >= yAbs)
6          moveRobot(Direction::rightward,xAbs);
7      else if(x > 0 && xAbs >= yAbs)
8          moveRobot(Direction::leftward,xAbs);
9
10     if(y < 0 && yAbs >= xAbs)
11         moveRobot(Direction::backward,yAbs);
12     else if(y > 0 && yAbs >= xAbs)
13         moveRobot(Direction::forward,yAbs);
14 }
15
16 void MainWindow::moveRobot(Direction direction, float speed){
17     output->moveRobot(direction,speed);
18 }
```

Une fois la direction est la vitesse de mouvement calculer il faut calculer le sens de rotation des roues et la vitesse de rotation de leur moteur.

Programme 3.3 – Calcul vitesse roues robot

```
1  void RobotOutputManager::moveRobot(Direction direction, float speed){
2      int speedI = speed*CAMERAPORCENT;
3      int speedL, speedR;
4      //qDebug() << "Le robot se dplace dans la direction " <<
           (int)direction << "  la vitesse " << speedI;
5      switch(direction){
6          case Direction::forward :
7              speedL = speedI;
8              speedR = speedI;
9          break;
10         case Direction::backward :
11             speedL = -speedI;
12             speedR = -speedI;
13         break;
14         case Direction::leftward :
15             speedL = speedI;
16             speedR = -speedI;
17         break;
18         case Direction::rightward :
19             speedL = -speedI;
20             speedR = speedI;
21         break;
22         default :
```

```
23          speedL = speedR = 0;
24      }
25
26      QByteArray* sendingByteArray;
27      sendingByteArray = new QByteArray();
28      quint32 Lvalue=qMin(qAbs(speedL),CAMERAPORCENT);
29      quint32 Rvalue=qMin(qAbs(speedR),CAMERAPORCENT);
30      sendingByteArray->clear();
31      sendingByteArray->append((char)0xff);
32      sendingByteArray->append((char)0x07);
33      sendingByteArray->append((char)(Lvalue*240/CAMERAPORCENT));
34      sendingByteArray->append((char)0x00);
35      sendingByteArray->append((char)(Rvalue*240/CAMERAPORCENT));
36      sendingByteArray->append((char)0x00);
37      sendingByteArray->append(((char)connexion->pidMode&0b10101000)
38                               |(speedL<0?0b00000000:0b01000000)
39                               |(speedR<0?0b00000000:0b00010000));
40      connexion->send(sendingByteArray);
41  }
```

Après cela il suffit de transmettre le message construit au robot.

Programme 3.4 – Transmition message au Robot

```
1  void TCPConnexionManager::send(QByteArray* sendingByteArray){
2     quint16 crcValue = ConnexionManager::crc16(sendingByteArray,1);
3
4     sendingByteArray->append((char)(crcValue));
5     sendingByteArray->append((char)(crcValue>>8));
6
7     tcpSocket->write(*sendingByteArray);
8     receive();
9  }
```

3.4 Conclusion

Après cela il était possible de déplacer le robot dans tous les sens et à toutes les vitesses bien plus facilement qu'en utilisant des boutons pour la direction accompagnés d'un `‘QSlider‘` pour la vitesse, et cela sans avoir cet effet d'interruption relativement désagréable, pour autant il est possible d'utilisé les touches de clavier `‘Z‘`, `‘Q‘`, `‘S‘` et `‘D‘` si l'utilisateur souhaite être plus précis sur son déplacement en appuyant par intermitance.

Conclusion

A la fin des séances de TP la réalisation de l'application n'est pas très avancée mais elle a le mérite de ne pas être comme les autres et d'utiliser des éléments plus intéressant mais moins facile à mettre en place et à utiliser.

En essayant de "réparer" certain(s) élément(s) défectueux nous avons perdu beaucoup de temps pour avancer dans le développement de partie(s) plus importante(s) tel que la réception vidéo de la caméra.

La réalisation de ce projet n'a pas été simple d'autant que n'avions reçu aucune information au niveau code et fonctionnel du robot ce qui a poussé bon nombre d'entre nous à rechercher et recopier le travail des autres année ce qui n'est pas ma philosophie d'apprentissage.

Annexe A

Programmes