

Création d'une IHM destinée à la surveillance de zones dangereuses

Auteur:
Gloaguen Thibault

Tuteurs:
Laffont Jacques
Aufrère Romuald

Introduction

► Contexte du projet

- Projet pour la société ERDF
- Travail réalisé par Polytech Clermont Ferrand

► Objectifs

- Faisabilité d'une IHM portable sur support Android
- Application et démonstration

► Environnement

- Framework Qt avec QtCreator 5.3 sous Windows 8
- Raspberry Pi avec un OS Raspbian (Linux)

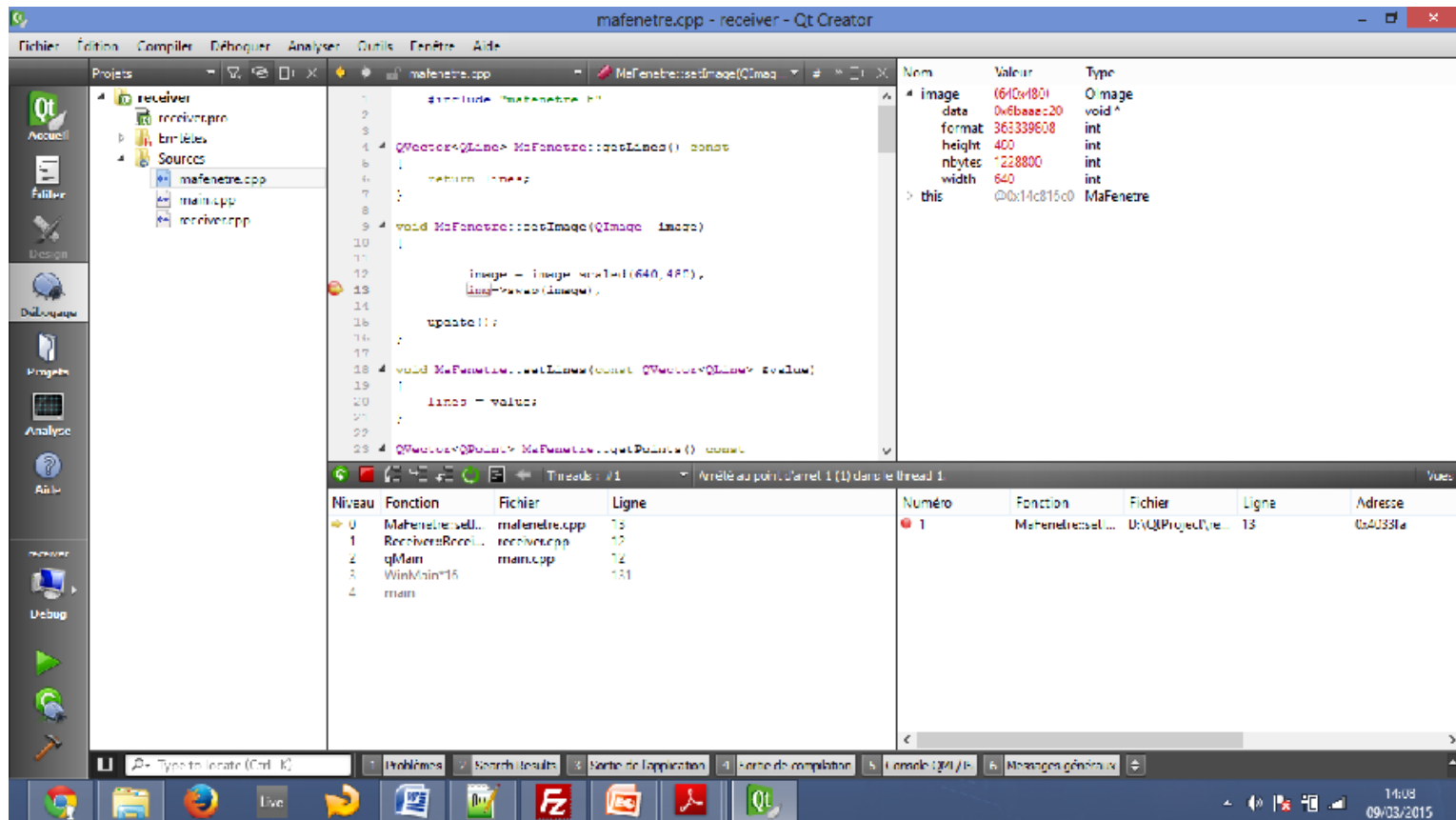
Plan

- ▶ **Ressources et méthodes**
 - L'environnement de travail
 - Méthodologies utilisées
 - Architecture de l'application
- ▶ **La zone de dessin**
 - Composition de la fenêtre
 - Dessin sur image
- ▶ **Le modèle Client Serveur**
 - Le Client coté fenêtre Qt
 - Le Serveur coté Raspberry Pi
 - Portage Android

Ressources et méthodes

Environnement de travail

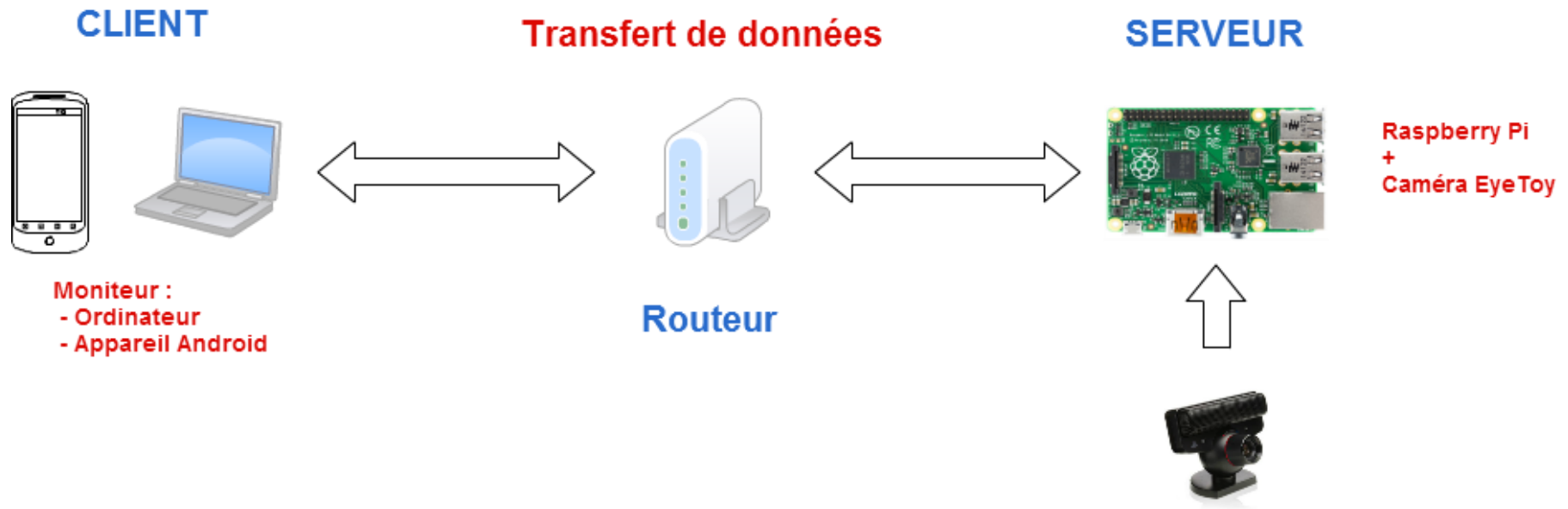
► QtCreator



Ressources et méthodes

Environnement de travail

► Mise en place d'un réseau local



Ressources et méthodes

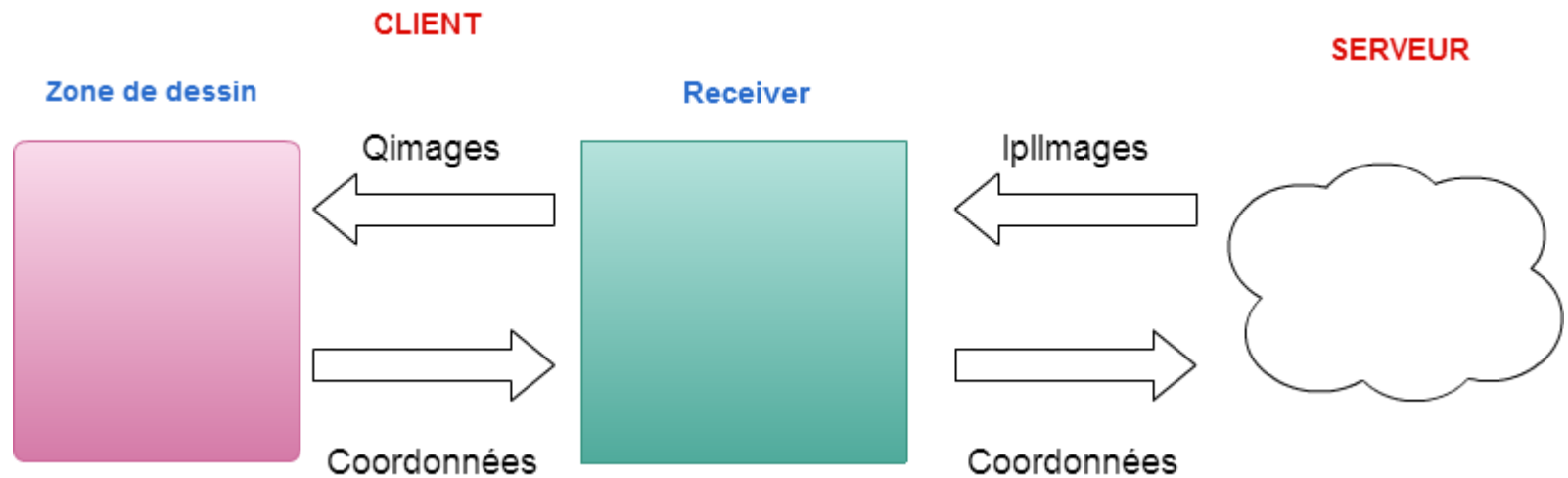
Méthodologies utilisées

► Démarche

- Etude des exemples d'applications dessin avec Qt
- Réalisation d'une fenêtre de dessin
- Etude des exemples Client Serveur en Qt
- Réalisation d'une application Client Serveur
- Intégration de la zone de dessin à l'application Client
- Adaptation du serveur sur Raspberry Pi

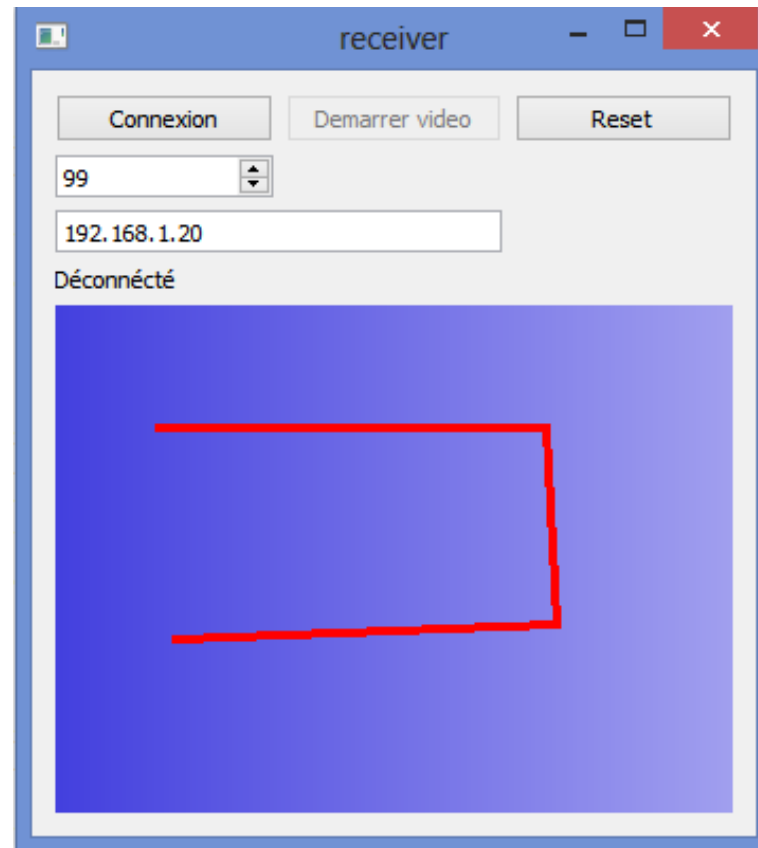
Ressources et méthodes

Architecture de l'application



La zone de dessin

Composition de la fenêtre



La zone de dessin

Dessin sur image

- ▶ Gestion des événements sur Qt
 - void *paintEvent*(QPaintEvent *event)
 - void *mousePressEvent*(QMouseEvent *event)

La zone de dessin

Dessin sur image

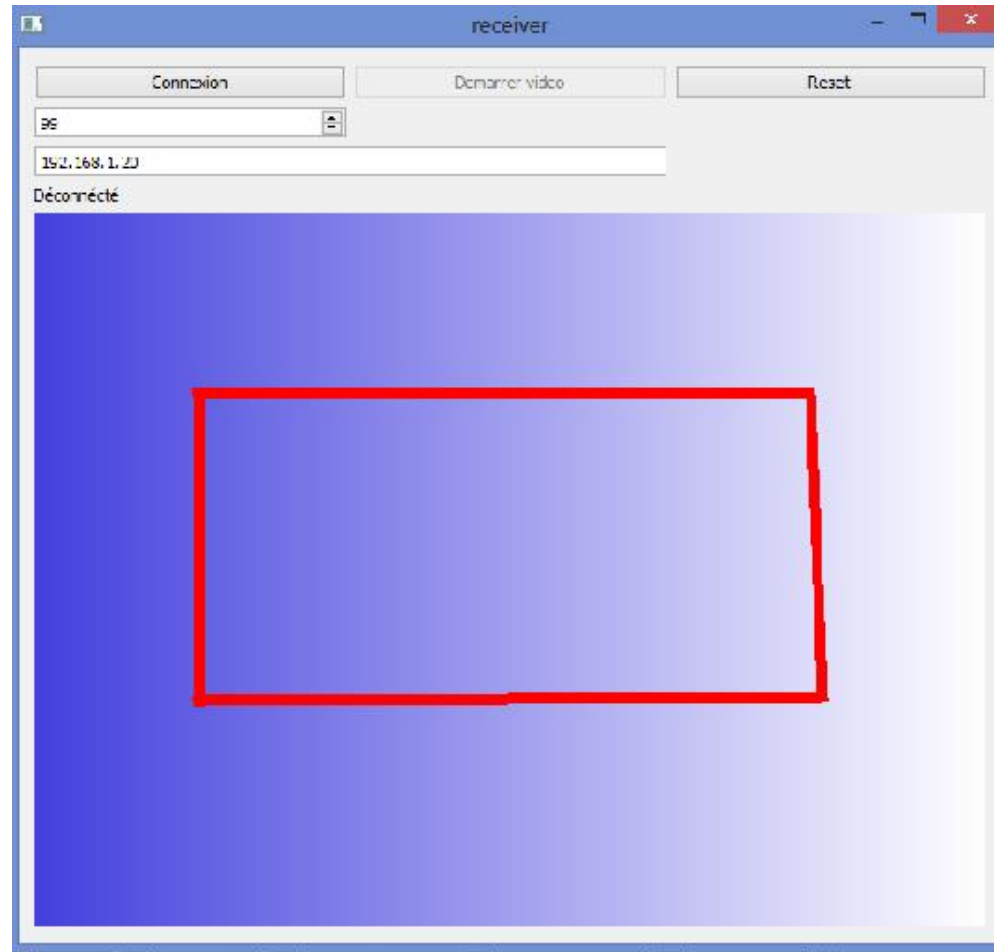
```
void MaFenetre::drawLines(QPainter *p)
{
    if (!startPos.isNull() && !endPos.isNull())
    {
        p->drawLine(startPos, endPos );
    }

    p->drawLines(lines);
}

void MaFenetre::paintEvent(QPaintEvent *event)
{
    QPainter p (this);
    QRect dirtyRect = event->rect();
    p.drawImage(dirtyRect,*img, dirtyRect);
    QPen pen;
    pen.setColor(Qt::red);
    pen.setWidth(7);
    p.setPen(pen);
    drawLines(&p);
}
```

La zone de dessin

Dessin sur image



Le modèle client serveur

Le client

► Le Receiver

- Création d'une Qsocket

- `socket = new QTcpSocket(this);`

- Connexion au serveur

- `socket->connectToHost(ip->text(), port->value());`

- Réception des données

- `connect(socket, SIGNAL(readyRead()), this, SLOT(donneesRecues()));`

- Conversion de l'IplImage en QImage

- `image = Ipl2QImage(newIplImage);`

Le modèle client serveur

Le client

- La méthode DonnéesRecues()
 - Deux parties :
 - Réception de la taille de message et de l'objet IplImage (width, height, widthStep)

```
bufferObj = new char[sizeof(IplImage)];  
bufferData= new char[tailleMessage*sizeof(IplImage)]  
in.readRawData(bufferObj,sizeof(IplImage));  
newImage = (IplImage*) bufferObj;  
tailleMessage=tailleMessage*sizeof(IplImage);
```

Le modèle client serveur

Le client

- La méthode DonnéesRecues()
 - Réception des datas image

```
if (socket->bytesAvailable() < tailleMessage) return;  
in.readRawData(bufferData,tailleMessage);  
newImage->imageData = bufferData;  
image = Ipl2QImage(newImage);  
drawFenetre->setImage(image);
```

Le modèle client serveur

Le client

- Envoi des coordonnées
 - Utilisation de la Classe QPoint pour récupérer les coordonnées
 - `Position = event->pos();`
 - Emission d'un signal dès qu'on a 4 Qpoints
 - `emit pointPret();`
 - Connexion du signal à la méthode d'envoi
 - `connect(drawFenetre, SIGNAL(pointPret()), this, SLOT(envoyerMessage()));`

Le modèle client serveur

Le client

- Envoi des coordonnées
 - La méthode EnvoyerMessage()

```
void Receiver::envoyerMessage()
{
    QVector<QPoint> data = drawFenetre->getPoints();
    QString messageAEnvoyer;
    for (int j = 0; j < data.size(); ++j)
    {
        messageAEnvoyer += " " + QString::number(data[j].x())
            + " " + QString::number(data[j].y()) + "|";
    }
    QByteArray paquet;
    QDataStream out(&paquet, QIODevice::WriteOnly);
    out << messageAEnvoyer;
    socket->write(paquet);
}
```


Le modèle client serveur

Le serveur

- Création d'une socket

- `int sock=0;`
- `bind(sock, (struct sockaddr*)&to, sizeof(to));`

- Capture d'image de la caméra

- `CvCapture *capture = cvCreateCameraCapture(CV_CAP_ANY);`
- `frame = cvQueryFrame(capture);`

- Attente de connexion du client

- `wait=accept(sock, (struct sockaddr*)NULL, NULL);`

- Envoi des données

- `write(wait, frame, sizeof(IplImage));`

Le modèle client serveur

Le serveur

- Envoi des données
 - Deux parties
 - Envoi du corps de l'objet IplImage et de la taille

```
int longueur = frame->imageSize+sizeof(IplImage);  
wait=accept(sock,(struct sockaddr*)NULL, NULL);  
write(wait,&longueur,sizeof(longueur));  
write(wait,frame,sizeof(IplImage));  
wait=accept(sock,(struct sockaddr*)NULL, NULL);  
longueur=longueur-sizeof(IplImage);
```

Le modèle client serveur

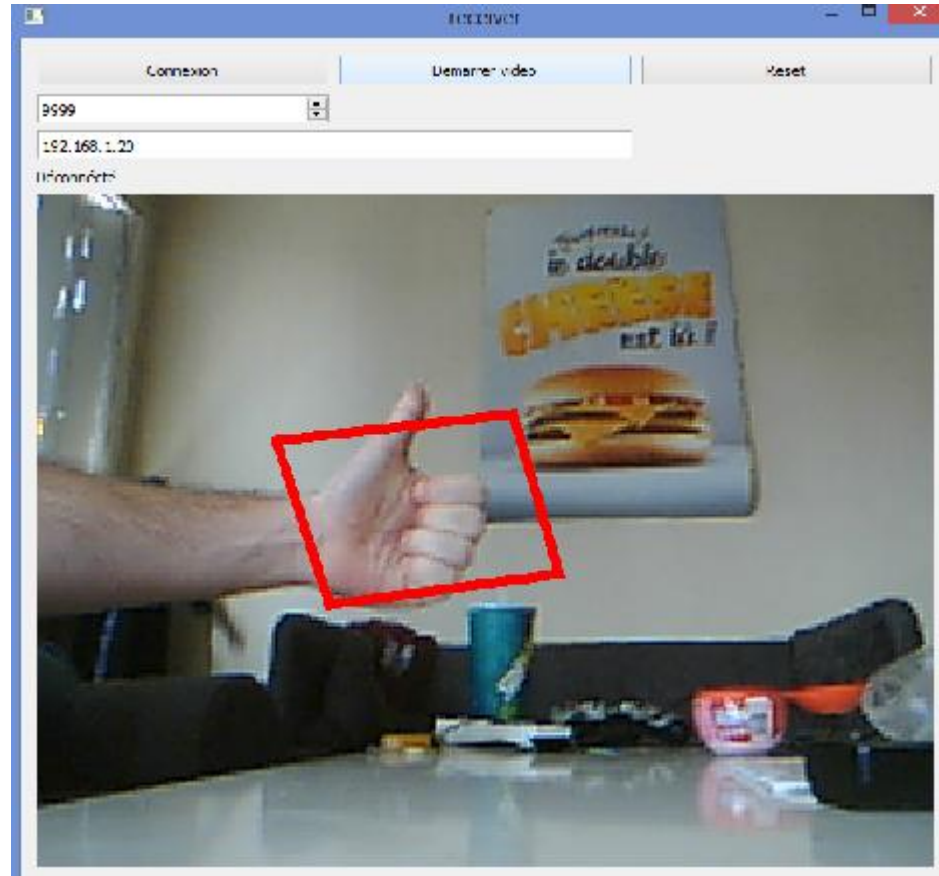
Le serveur

- Envoi des données
 - Envoi du tableau de pixels en boucle

```
while(1)
{
    int n;
    frame = cvQueryFrame( capture );
    write(wait,frame->imageData,longueur);
    printf("Attention retour n° image\n");
    read(wait,&n,sizeof(n));
    printf ("  %d |",n);
    l=cvWaitKey(1);
    if (l=='q')
        break;
}
```

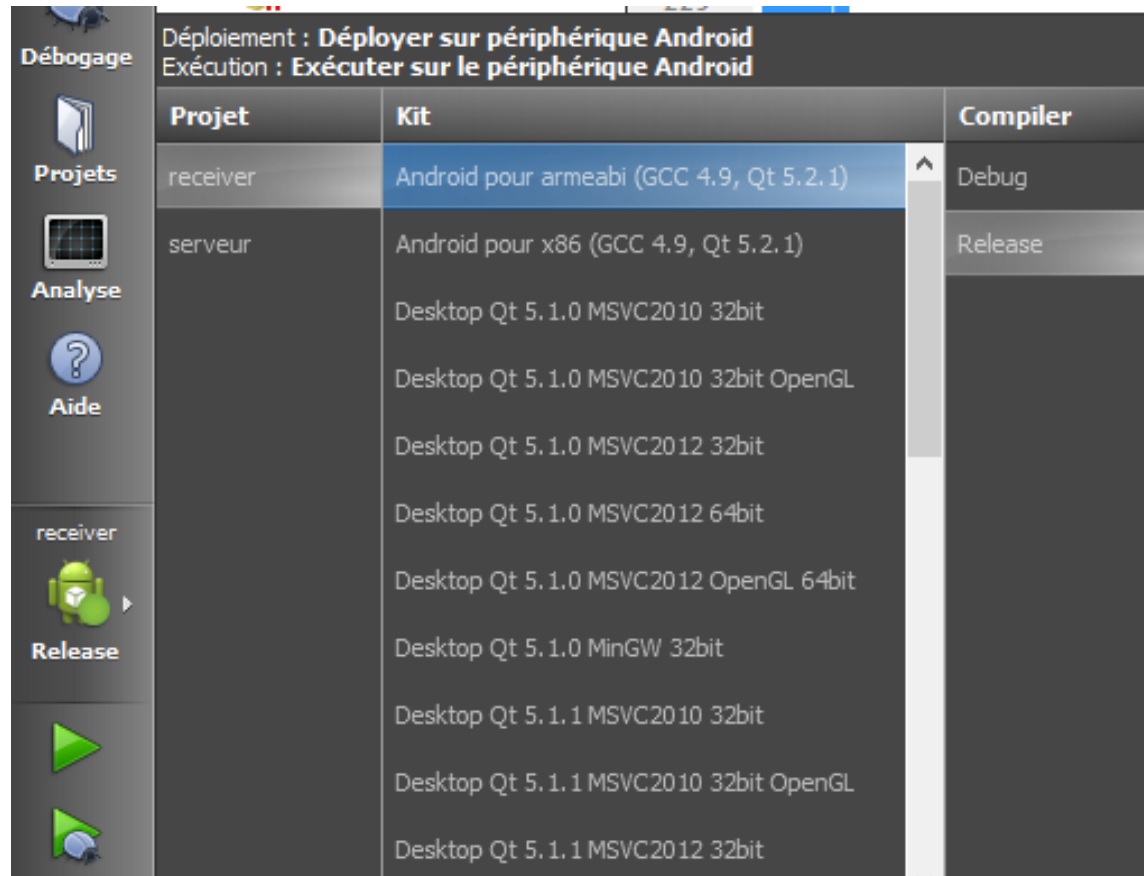
Le modèle client serveur

Le serveur



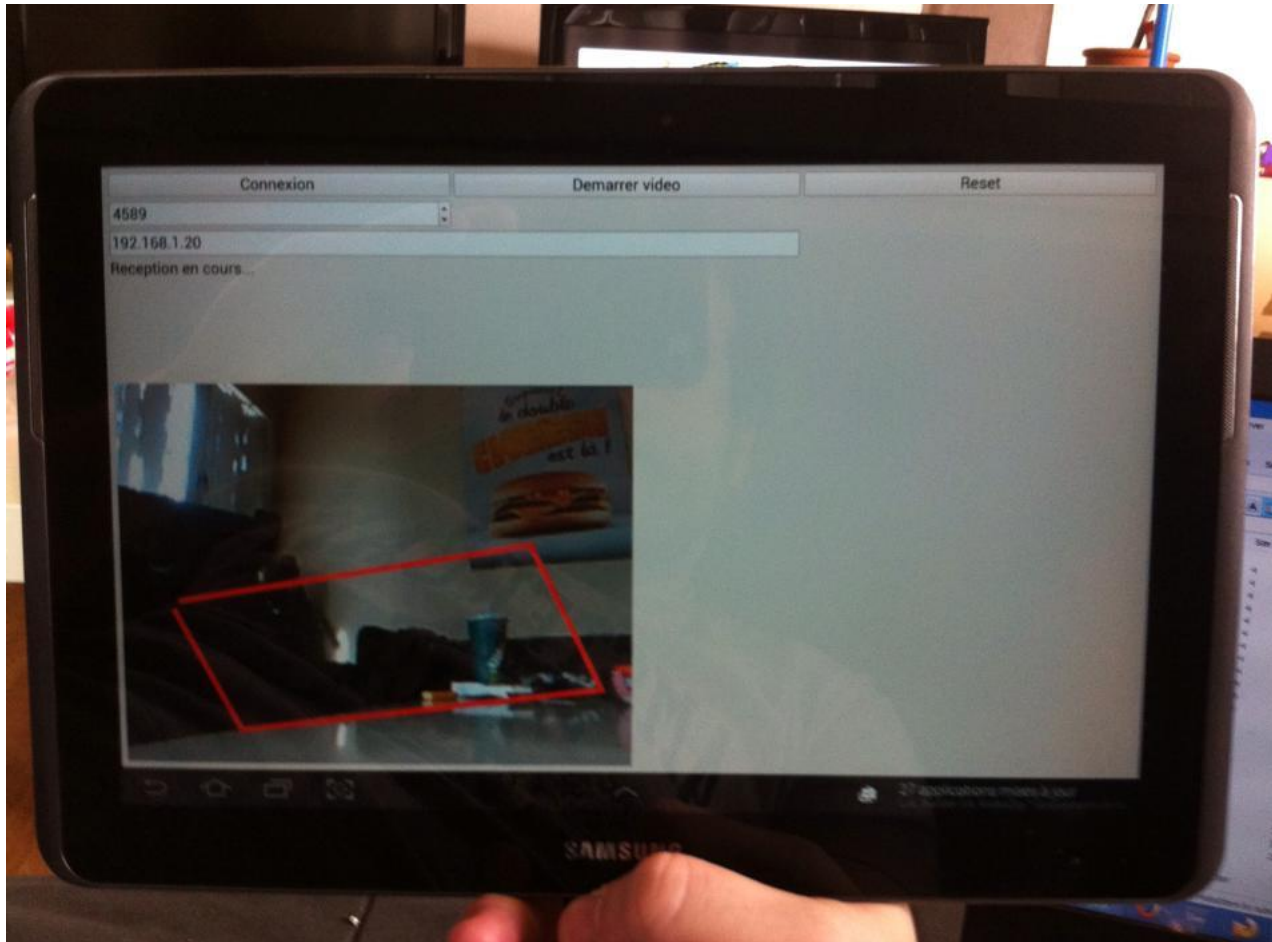
Le modèle client serveur

Portage Android



Le modèle client serveur

Portage Android



Conclusion

- ▶ Application fonctionnelle
 - Retransmission vidéo
 - Dessin sur vidéo
 - Récupération et envoi des coordonnées
- ▶ Portable sur Android
- ▶ Bonne compréhension du framework Qt

Perspectives

- ▶ Améliorer le Design de l'application
- ▶ Optimiser l'algorithme de traitement coté serveur
- ▶ Porter le programme de l'équipe Polytech sur Raspberry Pi et connecter l'IHM avec ce dernier

Merci de votre attention

