Université de Technologie de Troyes

Rapport de TP

SY22

Dimitri HURAUX Paul GARNIER Cheng JIANG

Semestre Universitaire P17

Sommaire

1 Introduction

Au cours de ces séances de TP nous avons eu à développer un projet visant à nous familiariser avec la programmation en C++/Qt et la programmation réseau. Cette application suit une architecture client/serveur et sert à envoyer et recevoir des images provenant d'une caméra accompagnées d'un timestamp et de valeurs extraites des capteurs de pression et de température. Il est à noter également qu'une fois l'image récupérée, on fait de la détection de visage sur celle-ci et on encadre le visage d'un rectangle.

Dans ce rapport, nous allons tout d'abord présenter l'environnement, l'application cliente et l'application serveur. Ensuite, on va décrire la manière dont le projet à évoluer. Nous allons ensuite développer la manière dont ont été implémentées les différentes fonctionnalités du programme. Suite à ça, nous allons comparer les délais entre TCP et UDP. Puis nous allons finir par une conclusion.

2 Environnement

Pour mener à bien ce projet, on a utilisé un raspberry pi 2 sous raspbian connecté à une caméra. On a développé le projet sous QtCreator qui est un IDE spécialement conçu pour coder en C++/Qt. La capture et le traitement des images s'effectuent avec la librairie openCV qui est une librairie spécialisée dans le traitement des images en temps réel. Pour la récupération des valeurs du capteur de pression et de température, on se base sur RTIMULib qui est une librairie servant d'interface pour l'accès aux capteurs.

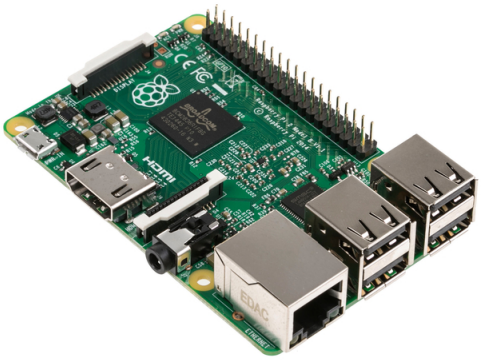

FIGURE 2.1 – Raspberry Pi 2

FIGURE 2.2 – Caméra

3 L'application cliente

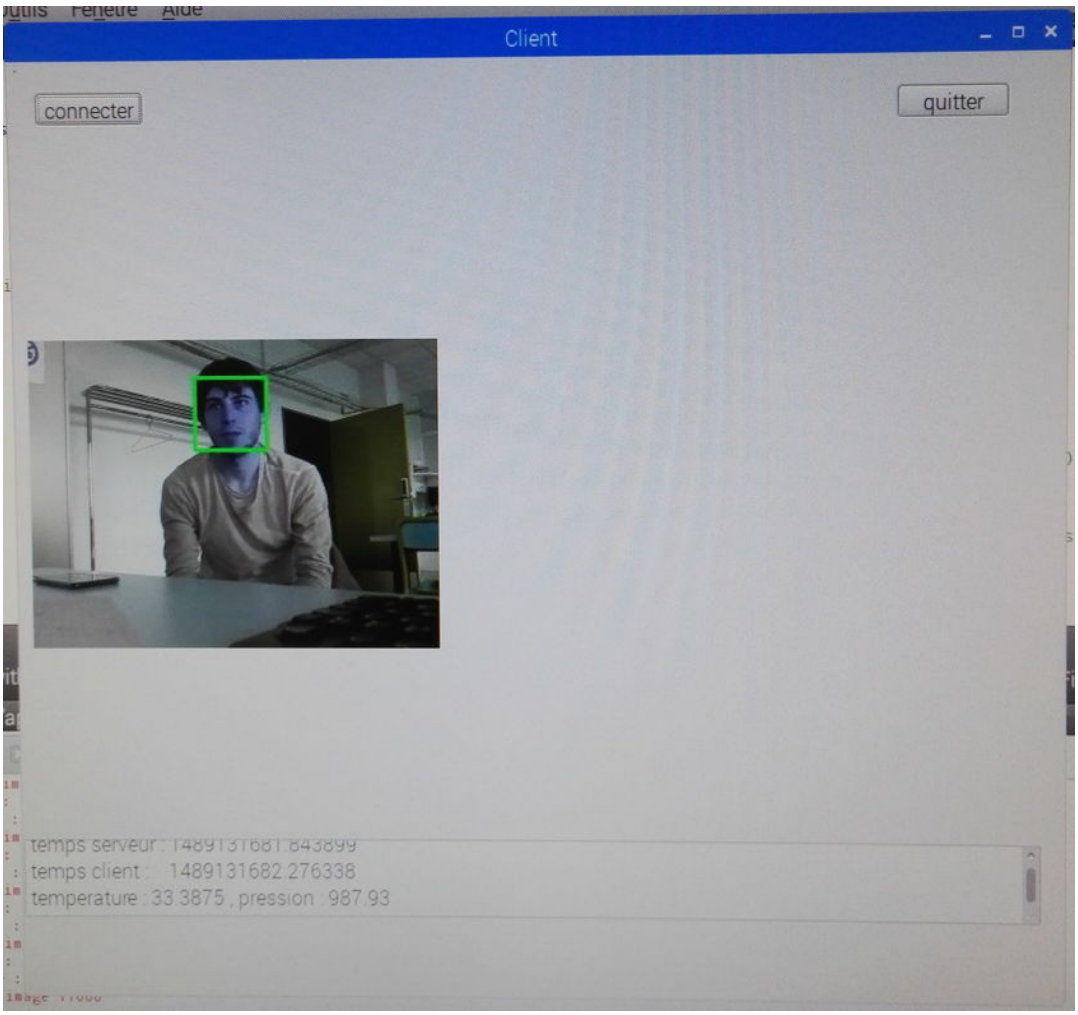

FIGURE 3.1 – Interface du Client

Nous pouvons voir ici, l'application cliente qui reçoit et affiche une image dont le visage est encadré en vert. Cette image est accompagnée de l'affichage des timestamp et des valeurs de température et de pression. Une nouvelle image est reçues toutes les secondes.

4 L'application serveur

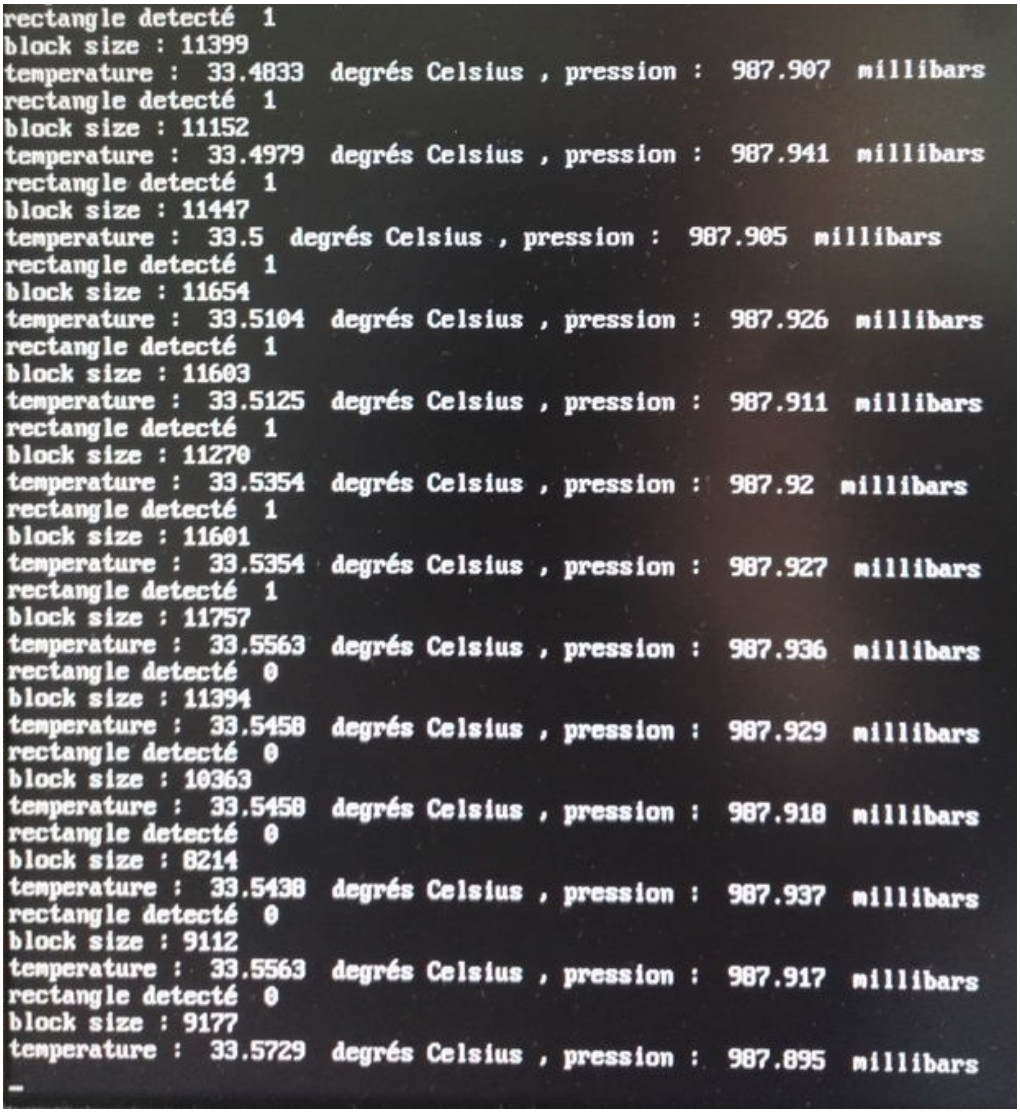

FIGURE 4.1 – Affichage console du serveur

Nous pouvons voir ici, l'application serveur qui est en ligne de commande. L'application serveur à chaque seconde récupère les images de la caméra via openCV, applique la reconnaissance de visage, ajoute le rectangle vert et envoie l'image accompagnée de son timestamp et des valeurs de température et de pression récupérées sur les capteurs.

5 Évolution du projet

Le projet a connu un certain nombre d'étapes avant d'aboutir.

On a implémenté tout d'abord une simple application client/serveur pour transmettre un message avec le protocole TCP.

Ensuite on a intégré la capture d'image provenant de la raspicam pour émettre celles-ci à intervalles réguliers.

Une fois la transmission d'image fonctionnelle, on l'a optimisé en compressant les images en JPEG.

Il nous a fallu ensuite mettre en place la détection des visages et des rectangles pour encadrer ceux-ci.

Après, on a optimisé la transmission en utilisant UDP à la place de TCP, on est même passé à UDP multicast pour pouvoir avoir plusieurs clients recevant les données d'un même serveur.

Pour finir, on a cherché également à transmettre les valeurs provenant des capteurs de température et de pression puis à éliminer l'interface graphique du serveur pour qu'il puisse fonctionner en mode console.

6 Implémentation des fonctionnalités

6.1 Réseau TCP

Pour implémenter les fonctionnalités réseaux en TCP, on s'est surtout basé sur les fonctions et objets Qt suivants :

- QTcpSocket qui représente un socket TCP avec lequel il faut se connecter à un serveur via connectToHost()
- QTcpServer qui représente un serveur TCP avec lequel il faut écouter sur un port via listen()
- QByteArray que l'on utilise pour stocker les données que l'on va émettre et recevoir
- QDataStream qui représente une abstraction de plus haut niveau que les QByteArray et que l'on utilise pour placer et récupérer les données dans ceux-ci (les données étant sérialisées et désérialisées)
- QTimer qui nous permet de déclencher l'émission de nouvelles données toutes les secondes
- connect() qui permet de lier un signal à un slot. On l'utilise notamment pour la réception de données sur les sockets et les nouvelles connexions sur le serveur

On prend bien évidemment le soin de transmettre tout d'abord la taille des données attendue pour ne procéder à la lecture de celles-ci qu'une fois qu'elles sont entièrement reçues.

6.2 Capture d'image

Pour la capture d'image on se base sur les fonctions suivantes de la librairie raspicam :

- grab() qui capture l'image
- retrieve() qui place l'image capturée dans une matrice openCV

Pour la capture d'image on se base sur les fonctions et objets suivants de la librairie openCV :

- cv : :Mat qui représente une matrice OpenCV
- flip() qui sert à retourner l'image dans le cas où elle est capturée à l'envers
- cvtColor() qui permet de changer d'espace colorimétrique (de nuances de gris à RGB par exemple)

— type() qui permet de connaitre l'espace colorimétrique actuel

Ensuite, on construit un objet QImage en se basant sur les attributs data, cols et rows de la matrice openCV qui représente l'image.

Pour des raisons de performance liées au Raspberry Pi, on ne capture qu'une image par seconde et dans une résolution réduite à 320x240.

6.3 Compression JPEG

Pour mettre en place la compression JPEG, on s'est surtout basé sur les fonctions et objets Qt suivants :

— QImage qui représente une image dont on utilise la fonction save() pour sauvegarder l'image (en JPEG) dans le QIODevice servant à émettre (côté serveur) et la fonction load() pour charger l'image depuis le QIODevice servant à recevoir (côté client)

— QDataStream qui permet de sérialiser des données binaires vers un QIODevice. On peut accéder à ce QIODevice avec la fonction device() (ce qu'on utilise notamment lors de l'appel aux fonctions save() et load() de QImage)

6.4 Reconnaissance faciale

Pour la reconnaissance faciale, on se base sur le CascadeClassifier de openCV.

Il faut commencer par appeler sa fonction load() pour charger le fichier de configuration pour la reconnaissance des visages (/usr/share/opencv/haarcascades/haarcascade_frontalface_alt.xml).

Ensuite, il faut égaliser l'histogramme de l'image avec la fonction equalizeHist().

Puis, on procède à la détection des visages sur l'image égalisée avec la fonction detectMultiScale().

Enfin, pour chaque visage détecté on crée le rectangle correspondant sur l'image d'origine en appelant le constructeur rectangle().

6.5 Réseau UDP

Pour passer l'application en UDP, on a remplacé QTcpSocket et QTcpServer par QUdpSocket. UDP étant un protocole non connecté, il n'y a plus besoin de faire appel à connectToHost(). Il faut néanmoins faire appel à la fonction bind() pour assigner un port à la communication.

6.6 Multicast

Pour l'implémentation du Multicast, on a dû choisir une IP sur la plage 224.0.0.1 à 239.255.255.254 (adresse multicast). Puis pour chaque client on a fait appel à la fonction joinMulticastGroup() de QUDPSocket. Il suffit ensuite que le serveur émette sur cette IP et tous les clients recevront les données.

Il ne faut pas oublier de définir le TTL avec la fonction setSocketOption(QAbstractSocket : :MulticastTtlOption,QVariant(nombre)) a une valeur plus élevée car par défaut, ce nombre est défini à 2.

6.7 Récupération de la valeur des capteurs

Pour récupérer la valeur des capteurs, on se base sur RTIMULib.

Tout d'abord, on crée la configuration avec "new RTIMUSettings("RTIMULib")".

Ensuite on utilise la fonction createIMU() pour créer l'entité qui représentera l'instrument qui accède aux capteurs.

Puis on utilise la fonction createPressure() pour créer l'entité correspondant au capteur de pression.

Il faut maintenant initialiser l'IMU avec la fonction IMUInit() et initialiser le capteur de pression avec pressureInit().

Pour récupérer les valeurs de capteurs, il faut appeler la fonction getIMUData() et la fonction pressureRead(). il reste maintenant à accéder aux attributs "temperature" et "pressure". la température est en degrés celsius et la pression en millibars.

Il ne faut pas oublier d'arranger un intervalle de 1000*IMUGetPollInterval() entre 2 collectes de valeur.

6.8 Élimination de l'interface graphique du serveur

Pour l'élimination de l'interface graphique du serveur, il faut remplacer dans la fonction main(), le type QApplication par QCoreApplication.Il faut également retirer toutes les appels aux fonctions show() et remplacer l'héritage de notre classe MainWindow de QMainWindow vers QObject.

7 Guide d'utilisation

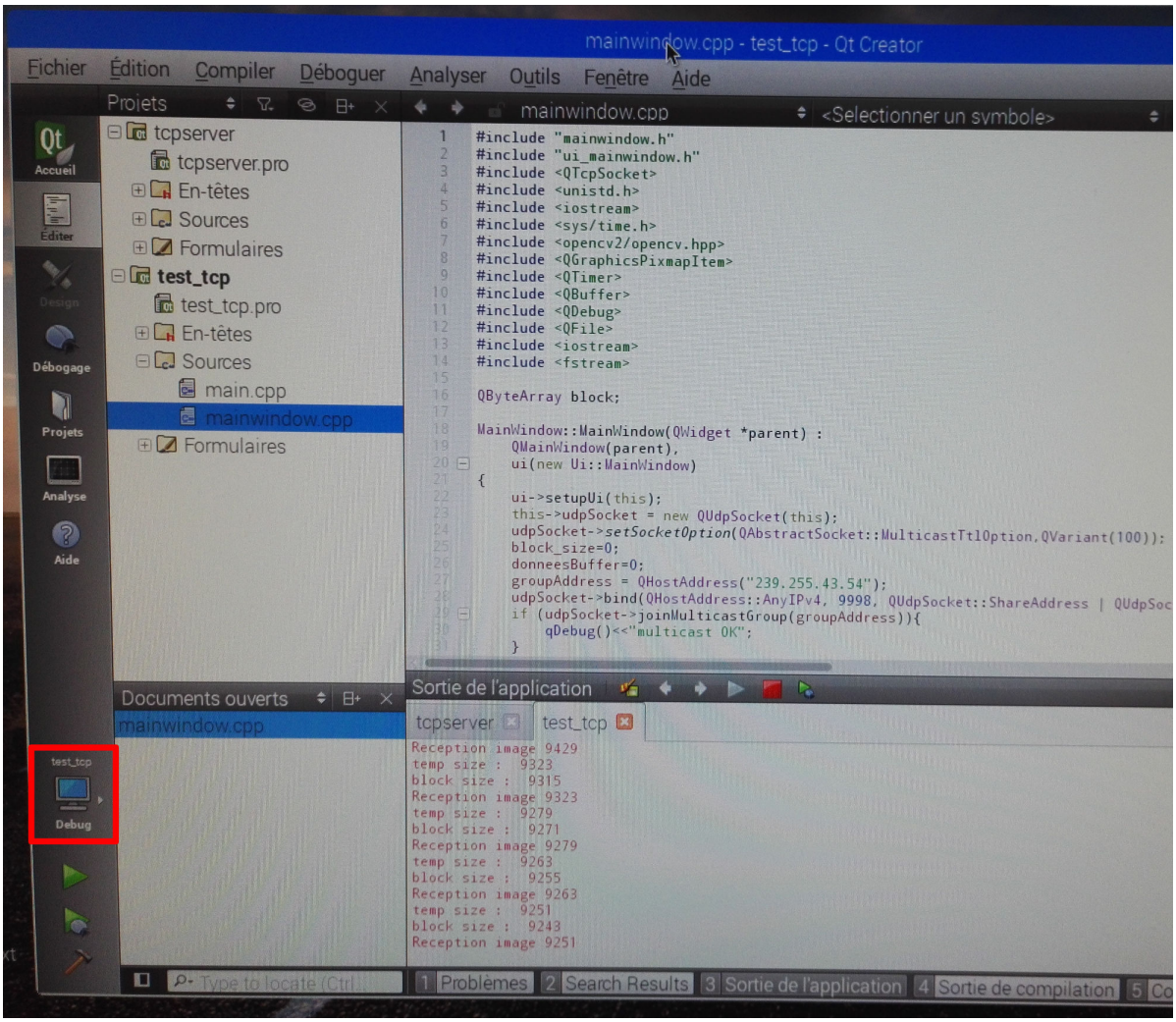

FIGURE 7.1 – interface de Qt Creator

Il faut tout d'abord lancer Qt Creator, charger le projet du serveur et le projet du client. Puis sélectionner le projet voulu en cliquant sur la petite puce dans l'encadré rouge. une fois le projet sélectionné, le compiler en cliquant sur l'icône en forme de marteau. Vous pouvez maintenant lancer le programme que ce soit en cliquant sur l'icône en forme de triangle vert en dessous de l'encadré rouge ou en lançant le programme compilé depuis un shell (utile notamment pour le serveur qui peut fonctionner sans interface graphique).

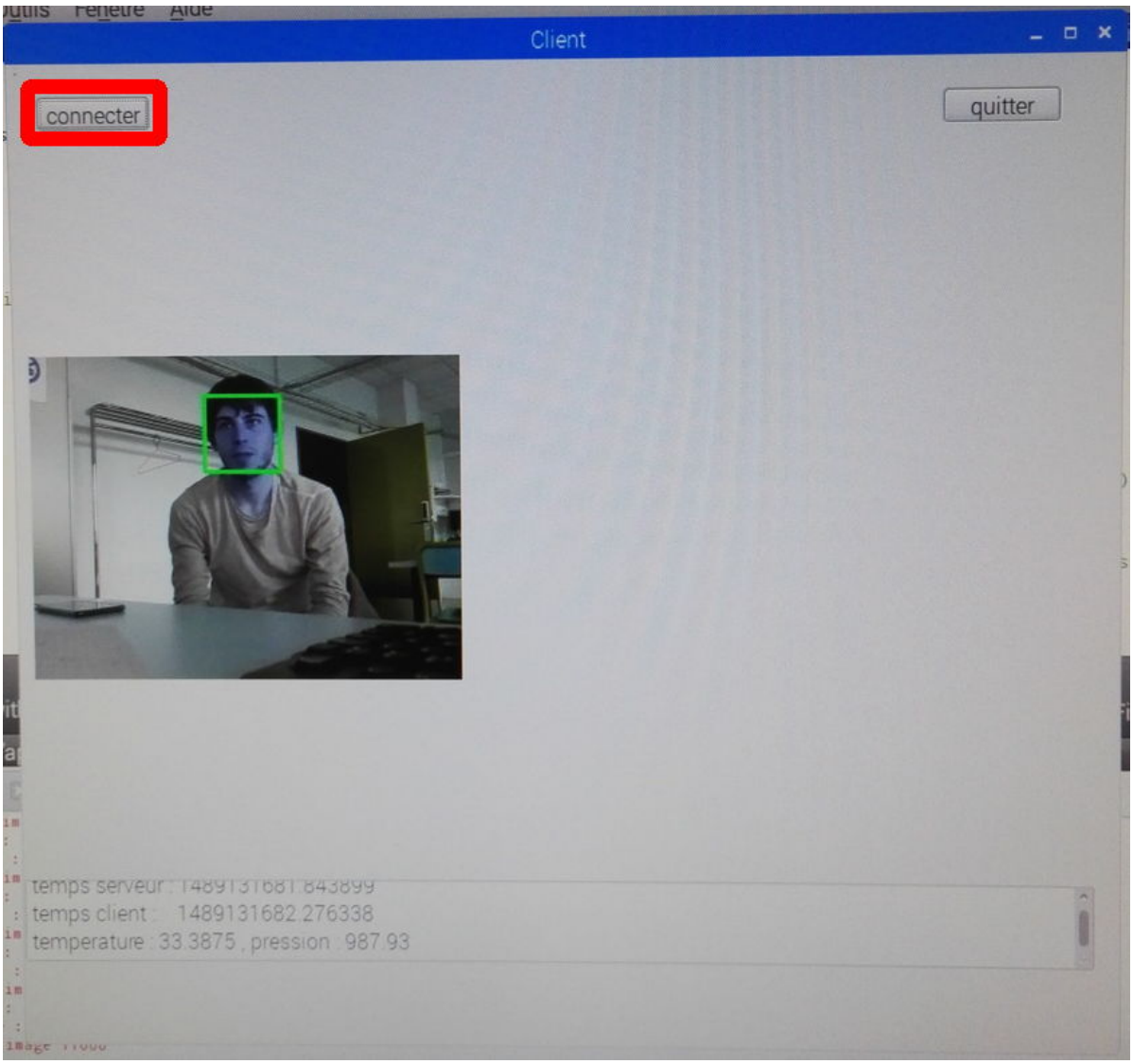

FIGURE 7.2 – Interface du client

Une fois le serveur lancé. Il suffit d'exécuter le client et de cliquer sur le bouton "connecter" (encadré en rouge) pour voir les images et les données les accompagnant s'afficher.

8 Comparaison de délai entre TCP et UDP

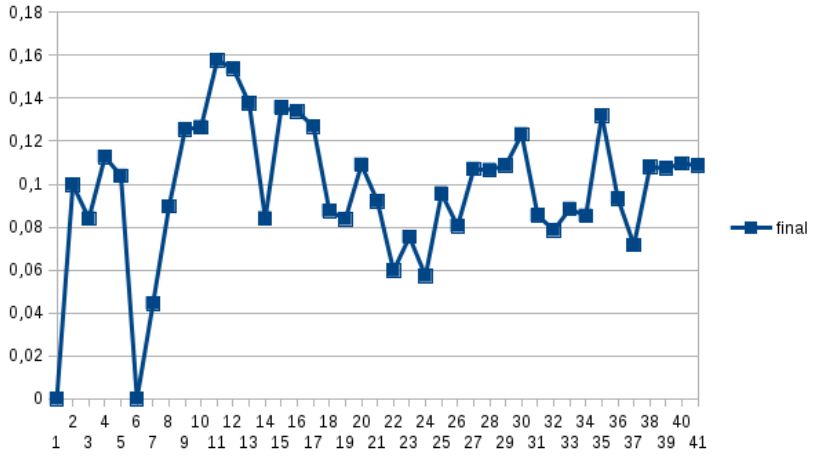

FIGURE 8.1 – Variation de délai en local avec TCP (en s)

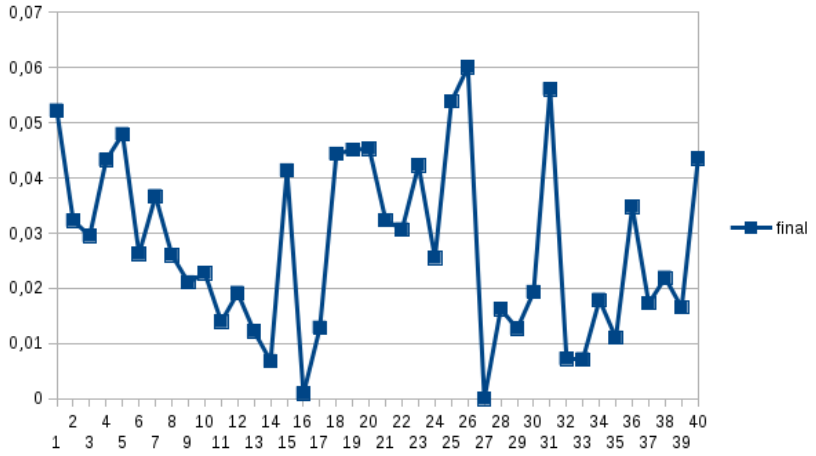

FIGURE 8.2 – Variation de délai en local avec UDP (en s)

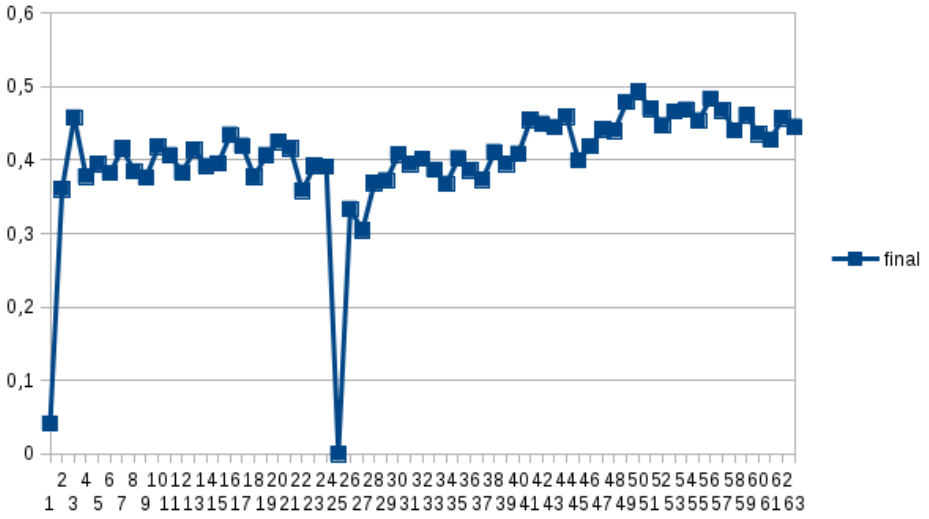

FIGURE 8.3 – Variation de délai entre 2 machines distantes avec TCP (en s)

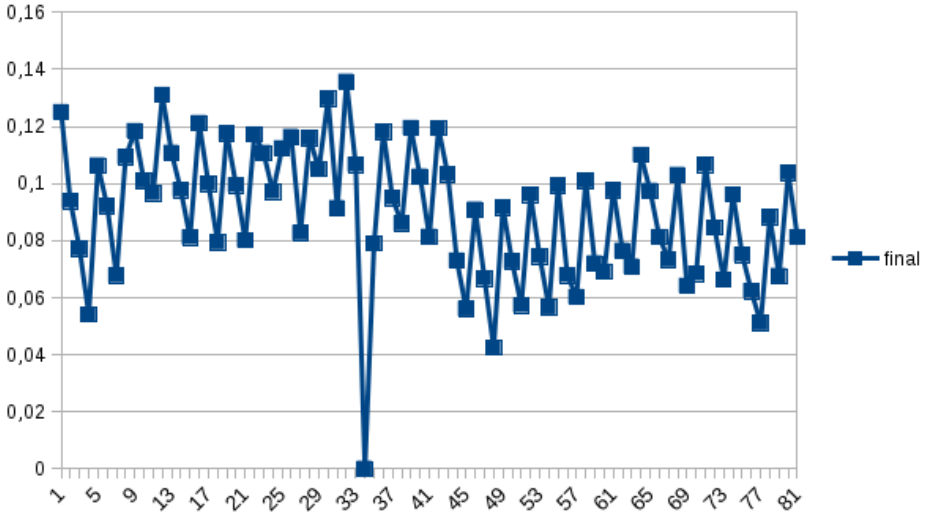

FIGURE 8.4 – Variation de délai entre 2 machines distantes avec UDP (en s)

Nous voyons ici, 4 schémas qui indiquent la variation du délai lors de tests distants et locaux (le zéro représentant le délai minimum obtenu lors des mesures). On remarque aisément que l'usage du protocole UDP améliore significativement les délais de transmission aussi bien à distance qu'en local. On en déduit que le protocole UDP est mieux adapté pour ce type d'application.

9 Conclusion

Au cours de ces séances de TP nous avons donc appris les bases de la programmation C++/Qt et de la programmation réseau. Nous avons pu également mesurer l'avantage que représente l'usage d'UDP plutôt que TCP dans ce genre d'application. La plus grande difficulté dans la mise en oeuvre de ce projet a été de se familiariser avec C++ et Qt "à la volée".