Manuel de l'utilisateur

- **Pilotage du Robot en console**

Prérequis :

- Logiciel minicom sur Unix (sudo apt-get install minicom), Hyperterminal sur Windows
- le bras robot dûment connecté avec le programme téléversé

Régler minicom (ou HyperTerminal) de manière à avoir le port correspondant au robot sur écoute (sur Ubuntu par exemple : **/dev/ttyACM0**) et à bien envoyer des caractères ASCII (format décimal...)

On communique avec le robot avec la syntaxe suivante :

- On commence par initier une nouvelle trame avec le caractère **&**
- On entre ensuite le numéro du servo (voir schéma) : ex : 1 pour le servo de base
- On entre ensuite la valeur de l'angle que l'on veut lui donner. Ex :120

On a donc finalement une instruction de la forme *&1120*.
Une fois le caractère **&** détecté, le robot écrit sur la liaison série les caractères reçus :
>> $ &1120 (données expédiées, non visible sur la fenêtre)
>> 120 Fin de Trame (Ce que renvoie le robot)
Il se déplace ensuite jusqu'à la position donnée par pas de 2°. En effet il est illusoire d'avoir une précision au degré avec le robot.

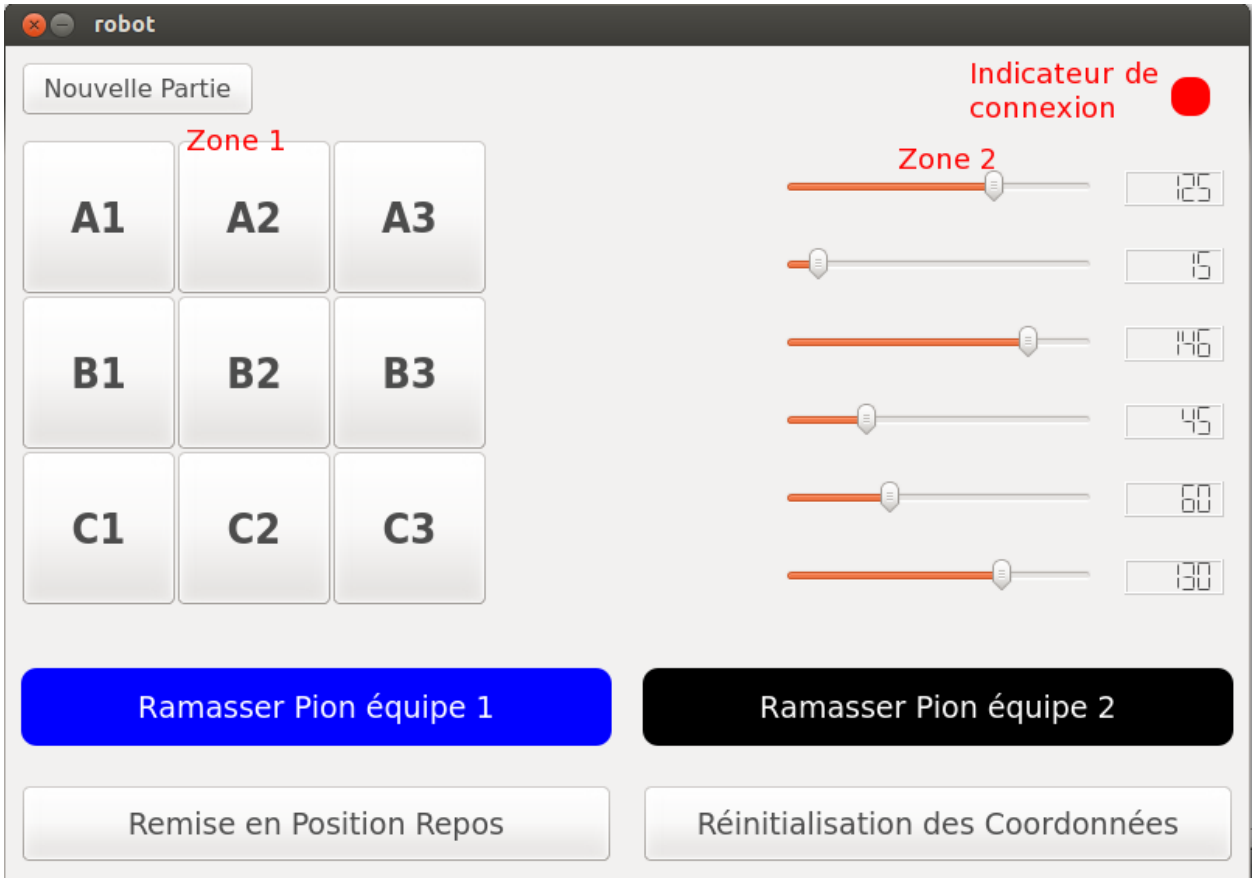
Ceci permet donc de contrôler le robot à la main.

- **Pilotage du Robot par interface graphique**

Prérequis :

- Avoir Qt installé (et un environnement UNIX)
- Le bras robot dûment connecté avec le programme téléversé
- **s'assurer que le robot est bien connecté sur le port /dev/ttyACM0 (seul supporté pour l'instant)**

Ci-dessus on trouvera une capture d'écran de l'interface.



La zone 1 correspond à la zone de jeu en tant que telle. L'appui d'un bouton provoque le déplacement du robot à la position référencée pour ce bouton (pour l'instant ces positions ne sont pas modifiables sans toucher au code, mais c'est un axe de développement prévu).
La Zone 2 permet de jouer directement sur la position des 6 servomoteurs du robot. Ainsi un déplacement du premier curseur provoquera le déplacement du servomoteur de la base. Ceci pourra servir à l'avenir pour corriger la position du bras robot, voire pour modifier la grille (avec le bouton réinitialisation des coordonnées, non implémentée pour l'instant).

L'indicateur de connexion en haut à droite permet de savoir si la liaison est opérationnelle (vert), ou non (rouge). Si le voyant est rouge, il peut être judicieux de relancer le programme et de s'assurer que le robot est bien connecté sur le port /dev/ttyACM0 (minicom peut fournir cette information).

Les boutons « Ramasser Pion équipe 1/2 » ordonnent au robot suivant le tour en cours d'aller chercher un pion dans la réserve de l'équipe considérée. A noter que la sélection d'une case après usage d'un de ces boutons la colorent en la couleur considérée, permettant un suivi de la partie en temps réel sur l'interface.

- **Pilotage du Robot par Leap Motion**

- Avoir le Leap Motion branché et le SDK installé
- Avoir Qt installé sur un environnement UNIX
- Avoir Qt Creator
- Le bras robot dûment connecté avec le programme téléversé
- **s'assurer que le robot est bien connecté sur le port /dev/ttyACM0 (seul supporté pour l'instant)**

Une fois le Leap Motion branché, il faut lancer QtCreator et sélectionner le projet dans le dossier /ROBOT_SANS_GUI/robot. Il faut ensuite lancer le programme en pressant CTRL R.

Le Leap Motion est alors actif, il faut commencer par désigner les deux angles extrêmes du damier virtuel pour l'initialiser. Pour cela pincez avec vos doigts dans les airs. Une fois ceci fait, en déplaçant vos doigts comme cela est montré sur la vidéo du projet vous pouvez sélectionner une case où placer le pion. Dans la console s'affiche les informations que remonte le Leap Motion, en particulier la case survolée puis sélectionnée.

Manuel du programmeur

1. Les outils indispensables

Minicom

Dans le terminal
$ sudo apt-get install minicom
puis pour faire les réglages :
$ minicom -s
Régler le port écouté sur /dev/ttyACM0

Qt Creator

L'IDE incluant l'API Qt est disponible sur le site officiel :
http://qt-project.org/downloads

Pour installer QextSerialPort :
https://code.google.com/p/qextserialport/

Un README explique bien les modalités d'installation. Voilà un rapide résumé des étapes pour une utilisation directe:
- Télécharger la librairie
- La placer dans un dossier de votre projet, par exemple LIBS
- Ajouter à votre fichier de projet :
include(LIBS/qextserialport/src/qextserialport.pri)
et #include "qextserialport.h" dans vos .h

Leap SDK

Les librairies pour le Leap Motion sont téléchargeables sur le site officiel, selon votre plateforme.
https://developer.leapmotion.com/downloads
Pour l'installer et l'utiliser sous Linux, cette vidéo montre la marche à suivre :
http://vimeo.com/71036624

2. Réglages en tous genres

- Utilisation de l'interface graphique sous Windows :
Il faut modifier le code fourni. Ouvrez MaFenetre.cpp et en haut du fichier modifiez :
m_serialPort = new QextSerialPort("/dev/ttyACM0"); // établit la connexion avec le port
en
m_serialPort = new QextSerialPort("COM1"); // établit la connexion avec le port

où COM1 est le nom du port utilisé.
Nota : La même méthode peut être utilisée pour modifier le nom du port sous UNIX.

- Utilisation du programme avec le Leap Motion sur une machine 64 bits
Avec Qt Creator, ouvrez le projet, remplacez x86 par x64, ça doit donner quelque chose comme ça :

```
win32:CONFIG(release, debug|release): LIBS += -L$$PWD/lib/x64/release/ -lLeap
else:win32:CONFIG(debug, debug|release): LIBS += -L$$PWD/lib/x64/debug/ -lLeap
else:unix: LIBS += -L$$PWD/lib/x64/ -lLeap
```

- Utilisation du programme avec le Leap Motion sur Windows
Bien que théoriquement possible, cela n'a pas été mené au bout. Néanmoins les librairies spécifiques à Windows existent, reste à savoir comment dire à Qt de se servir des DLL.

3. Le code

a. Liaison série

L'instanciation de la classe Qextserialport est faite dans le constructeur de MaFenetre pour la partie interface graphique, à l'initialisation du Leap Motion pour la partie sans interface.

On utilise ensuite une simple fonction sendData**N** où N est le numéro du servo concerné pour envoyer les données au robot suivant le formalisme vu dans le rapport. Pour mémoire il faut s'exprimer à lui de la manière suivante :
& N ANGLE
où N est le numéro du servomoteur à piloter et ANGLE la valeur d'angle à lui donner.

b. Partie code avec Interface Graphique :

Pour obtenir cette partie du code : ouvrir Qt puis sélectionner **file>open File or Project** et sélectionner le fichier **robot.pro** contenu dans le dossier **interface_graphique/robot.**

Le code se compose de trois fichiers :
- MaFenetre.h : qui définit le prototype de la classe MaFenetre
- MaFenetre.cpp : qui explicite le constructeur de la classe MaFenetre ainsi que les slots créés
- main.cpp : qui instancie la classe MaFenetre et l'affiche

Une particularité du développement de projet sous Qt est qu'il crée un fichier .pro (ici robot.pro) qui représente une sorte de Makefile permettant de lier les headers aux codes sources et d'inclure les librairies nécessaires au projet.

Dans notre projet, nous avons notamment besoin de la librairie *qextserialport* ainsi que de la configuration qwidget.

MaFenetre.h :

Commençons par le header de notre projet. Il inclut bon nombre de librairies permettant de mener à bien le projet :

#include <QApplication> -> créationd'une application
#include <QWidget> -> création de widgets
#include <QPushButton> -> création de boutons poussoirs
#include <QSlider> -> création de sliders
#include <QLCDNumber> -> création d'afficheur LCD
#include <QMessageBox> -> création de message d'erreur ou d'information
#include <qextserialport.h> -> permet de gérer la liaison série
#include <stdio.h>

Ensuite apparait le prototype de la classe MaFenetre qui contient des méthodes publiques : son constructeur et son destructeur; des slots publics (toutes les méthodes implémentées pour permettre le déroulement de la partie de morpion) et des attributs privés. Ces attributs sont tous les widgets présents dans la fenêtre, ainsi que la liaison série et les variables joueur, tour_eq1 et *tour_eq2* permettant le bon déroulement de la partie.

MaFenetre.cpp :

Cette partie du code est la plus compliquée et, comme vous pouvez le voir, la plus longue.

Tout d'abord, on a le **constructeur** de la fenêtre : MaFenetre(). Dans ce constructeur, d'abord on l'hérite de la classe QWidget pour permettre de créer une fenêtre. Ensuite, on commence par instancier la liaison port série. Puis on met en place tous les éléments de la fenêtre : d'abord la fenêtre en elle-même puis ensuite tous les widgets (boutons poussoirs, sliders, …). Finalement, on finit par connecter à l'aide d'un *connect* tous les widgets aux slots dont nous avons créé le prototype dans le header.

Ensuite viennent les **slots**. Le tout premier est *RemisePosRepos()* qui permet, comme son nom l'indique de remettre le robot en position de repos, mais pour cela, il faut, s'il possède un pion, qu'il le dépose d'abord d'où le *if(...)*.

Après viennent les **slots correspondant à l'action de déposer un pion sur une case** du damier de jeu. Chaque slot est développé de la même façon : d'abord on regarde la valeur de la variable *joueur* qui va nous indiquer quel joueur a pris un pion (équipe 1 ou 2) et si le robot possède bien un pion. La case va alors être coloré avec la couleur du pion sélectionné (bleu pour l'équipe 1 et noir pour l'équipe 2) grâce à la fonction *setStyleSheet(...)* et ne pas être

modifiée si aucun pion n'est déposé. Par ailleurs, le code permet de désactiver la case afin qu'aucun autre pion ne puisse être déposé sur cette même case (*setEnable(false))*. Enfin, chacun de ses slots se finira par une remise en position du robot pour plus d'esthétique et pour limiter les pics de courant trop importants du robot.

Après nous avons les **slots de ramassage de pions**. Les deux fonctionnent de manière identiques. D'abord on vérifie que le robot n'est pas déjà en possession d'un pion, auquel cas il doit d'abord le déposer avant d'en ramasser un autre. On vérifie également quel est le joueur qui a joué au tour précédent et on affiche un message d'erreur si jamais le même joueur essaye de reprendre un pion à l'aide de QMessageBox. Ensuite, on regarde à quel tour on se trouve à l'aide de la variable *tour_eq**n*** pour savoir quel pion aller chercher et on va le chercher. On affichera également un message quand l'équipe n'a plus de pion, ce qui ne doit, normalement, pas se produire en mode de jeu normal.

Ensuite, on a le **slot de réinitialisation de la partie** : *NouvellePartie().* On vérifie que le robot n'ait pas de pion en sa possession (avec *joueur*). Si c'est le cas il le dépose dans une partie hors du damier de jeu (pour pas le poser sur une case déjà pleine en cas de partie où le damier est entièrement rempli). Puis ensuite, toutes les cases sont réinitialisées (*setStyleSheet(default)* et *setEnable(true)*) et on réinitialise les variables de jeu (*joueur = 0 ; tour_eq**n** = 1*).

Le **slot de réinitialisation des coordonnées** : *reinit()* n'a pas été implémenté ici. Comme expliqué dans le rapport, nous n'avons pas eu le temps de mettre en place les équation de cinématique inverse, ainsi cette méthode ne fait qu'afficher un message d'information.

Finalement, nous avons les **slots d'envoi des données au robot**. Dans ces slots on crée un tableau de caractère dans lequel on stocke la donnée des angles à transmettre (qui sont donnés par la valeur appliquée aux sliders) + le numéro du servo (donné par le slider modifié) + le caractère spécial qui détermine le début de la trame envoyée au bras de robot (&).

Petite subtilité qu'il faut bien comprendre : Pour contrôler le robot, on passe par l'intermédiaire des **sliders**. En effet, lorsqu'on clique sur une case, le slot en question va modifier la valeur des sliders et c'est cette modification des sliders qui va engendrer un signal qui va appeler les slots d'envoi de données au robot via liaison série.

Remarque : l'envoi des données via liaison série se fait séquentiellement. Si le slider1 est modifié puis le 2, alors le slot *sendData1* va s'effectuer (le robot va alors déplacer selon le servo 1) puis ensuite le slot *sendData2* va provoquer le déplacement du servo 2 etc…

main.cpp :

Dans ce fichier, il n'y a rien de plus à dire si ce n'est que la fenêtre est affichée grâce à la méthode *show()*.

c. Avec le Leap Motion

Le détail du fonctionnement du détail est explicité directement dans le rapport. La connexion avec la liaison série se fait au moyen d'une classe *Serie* contenant l'ensemble des fonctions relatives à la liaison série et au déplacement du robot. Elle est instanciée au sein de la classe Sample Listener qui définit la routine que doit exécuter le Leap Motion.