UNIVERSIDAD DE OVIEDO

Departamento de Ingeniería Eléctrica,
Electrónica, de Computadores y Sistemas

Guía de Prácticas

ASIGNATURA:	Implementación de Sistemas de Control		
CENTRO:	Centro Internacional de Postgrado		
ESTUDIOS:	Master en Ingeniería Mecatrónica		
CURSO:	2º	CUATRIMESTRE:	1
CARÁCTER:	Obligatoria	CRÉDITOS ECTS:	6
PROFESORADO:	Ignacio Alvarez, Fernando Briz		

PRACTICA 04: Comunicación TCP/IP para control de motores con Qt-SDK/C++

Realizar un programa cliente TCP orientado a eventos en modo consola, que se comunique con las controladoras del simulador de plotter vertical, cuya documentación e instalación se encuentra disponible en: http://isa.uniovi.es/~ialvarez/Curso/descargas/SimuladorWSL2/Simulador-VerticalPlotter-WSL.pdf

El programa pedirá por teclado un texto xml con el formato:

```
<Motor><name>left</name><cmd>message to send</cmd></Motor>
```

El texto a enviar (`message to send`) debe seguir el formato indicado en la documentación de las controladoras (se deberá añadir `"\r\n"` al final del texto extraído del campo `<cmd>`).

Cada vez que haya una respuesta de una controladora, se escribirá dicha respuesta en la consola.

Se dispone de las siguientes herramientas en Qt-SDK:

1. Clase `QTcpSocket` (añadir `QT += network` en archivo `.pro`): permite (de una manera similar a QSerialPort) establecer una conexión por red con un servidor remoto, e intercambiar datos con él. El protocolo de datos a intercambiar está determinado por las controladoras (ver documentación). El cliente se conecta al servidor de la controladora con la función `connectToHost()`.

   Se creará una clase MyMotorController con un objeto privado de tipo `QTcpSocket`:

MyMotorController.h

```
#ifndef MYMOTORCONTROLLER_H
#define MYMOTORCONTROLLER_H
#include <QObject>
#include <QTcpSocket>

class MyMotorController : public QObject
{
    Q_OBJECT
private:
    QString name;
    QTcpSocket client;
    ...
public:
    void MyMotorController::MyMotorController(datos del motor y el servidor,QObject* parent=nullptr);
public slots:
    void OnClientConnected();
    void OnClientDisconnected();
    void OnClientDataReceived();

    void OnCommandReceived(QString msg);
};


#endif // MYMOTORCONTROLLER_H
```

MyMotorController.cpp

```
#include "MyMotorController.h"
```

```
void MyMotorController::MyMotorController(datos del motor y el servidor,QObject* parent) :
                                                                        QObject(parent)
{
    name=nombre del motor;
    Conectar señales de client con slots de this: connected(), disconnected(), readyRead()

    client.connectToHost(datos del servidor);

}

void MyMotorController::OnClientConnected() {
   escribir texto en consola
}

void MyMotorController::OnClientDisconnected() {
   escribir texto en consola
}

void MyMotorController::OnClientDataReceived() {
   QString txt=client.readAll();
   escribir txt en consola
}

void MyMotorController::OnCommandReceived(QString msg) {
   if (el campo <name> de msg se corresponde con el name de este motor) {
       QString cmd=extraer campo <cmd> de msg
       escribir cmd en consola
       QString toSend=cmd+"\r\n";
       client.write(toSend.toLatin1());  // se necesita un QByteArray
   }
}
```

2. Utilizar la clase siguiente para la espera en un hilo (thread) separado de la entrada del usuario por teclado; de esta forma se evita detener el bucle de eventos principal:

MyConsoleInput.h

```
#ifndef MYCONSOLEINPUT_H
#define MYCONSOLEINPUT_H
#include <QObject>
#include <QThread>

class MyConsoleThread : public QThread
{
    Q_OBJECT
private:
    void run() override;
signals:
    void KbdInputAvailable(const QString& txt);
};

class MyConsoleInput : public QObject
{
    Q_OBJECT
private:
    MyConsoleThread consoleThread;
public:
    explicit MyConsoleInput(QObject *parent = nullptr);
signals:
    void KbdInputAvailable(const QString&  txt);
};

#endif // MYCONSOLEINPUT_H
```

MyConsoleInput.cpp

```
#include "MyConsoleInput.h"
#include <QTextStream>

void MyConsoleThread::run()
{
    QTextStream qt_out(stdout),qt_in(stdin);

    while (1)
    {
        qt_out << "INPUT: ";
        qt_out.flush();
        QString txt=qt_in.readLine();
        emit KbdInputAvailable(txt);
    }
}

MyConsoleInput::MyConsoleInput(QObject *p) : QObject{p}
{
    connect(&consoleThread,SIGNAL(KbdInputAvailable(QString)),
            this,SIGNAL(KbdInputAvailable(QString)));
    moveToThread(&consoleThread);
    consoleThread.start();
}
```

El programa principal crea objetos de ambas clases, establece la conexión signal/slot, y deja la ejecución a app.exec():

main.cpp

```
int main(int argc,char* argv[])
{
    ...
    MyMotorController motorLeft(datos del servidor);
    MyConsoleInput console;
    QCoreApplication app(argc,argv);


    QObject::connect(&console,SIGNAL(KbdInputAvailable(QString)),
                     &motorLeft,SLOT(OnCommandReceived(QString)));

    ...
    return app.exec();
}
```

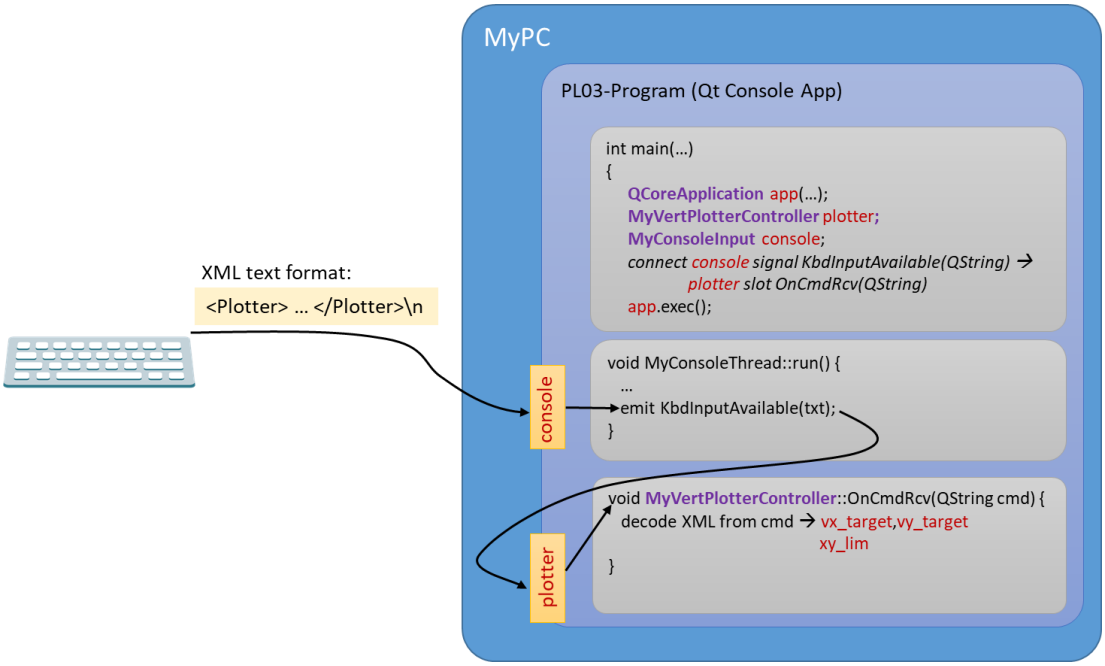
3. Ampliación propuesta: Realizar el control de ambos motores en una única clase MyVerticalPlotterController, de forma que se pueda utilizar el mensaje de texto siguiente para establecer la velocidad vxy del trazador (en mm/s), y asegurar que se detiene el mismo si se sobrepasan los límites indicados (en mm):

```
<Plotter>
   <vx_target>20</vx_target>
   <vy_target>40</vy_target>
   <xlim>[-500,500]</xlim>
   <ylim>[-1500,-100]</ylim>
</Plotter>
```

   Para ello, será necesario:

   4. Recibir y decodificar el mensaje xml en un slot de la clase a desarrollar. Guardar vx_target , vy_target, xlim e ylim en variables miembro de la clase.
   5. Lanzar un timer cada 100ms en dicha clase, y en el slot correspondiente emitir el comando G (Get) a ambos motores.
   6. Cuando ambos motores hayan respondido al comando Get :
      - Obtener los valores de pulsos de encoder y, a partir de ellos, calcular la posición xy utilizando las ecuaciones proporcionadas en la documentación del simulador.
      - Comparar xy con los límites xlim e ylim; si alguno es excedido, poner a cero los valores vx_target y vy_target.
      - Si no se exceden los límites, utilizando la posición xy calculada, convertir vx_target (mm/s), vy_target (mm/s) a velocidades necesarias en ambos motores (v1, v2, medidas en mm/s).
      - Solicitar a cada motor establecer la velocidad deseada en mm/s.

Esquema de recepción de comandos de consola:



Esquema de control del plotter vertical: