UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

2022

Programación Orientada a Objetos

PROYECTO INTEGRADOR

DOCENTE: CÉSAR OMAR ARANDA
CARRERA: INGENIERÍA MECATRÓNICA

Alan Vignolo - 12667
Brandon Mamani - 12749
Juan Camaño - 12872

Índice

Introducción

Se debe implementar una solución orientada a objetos aplicando los conceptos fundamentales del ciclo a la manipulación remota de un robot de 3 grados de libertad con efector final.

El robot tiene velocidades y posiciones (lineales y angulares) máximas. El controlador es quién conoce los parámetros geométricos y cinemáticos del robot y realiza un control parcial de la viabilidad de cada orden.

El sistema se implementa usando orientación a objetos y debe tener un cliente y un servidor.

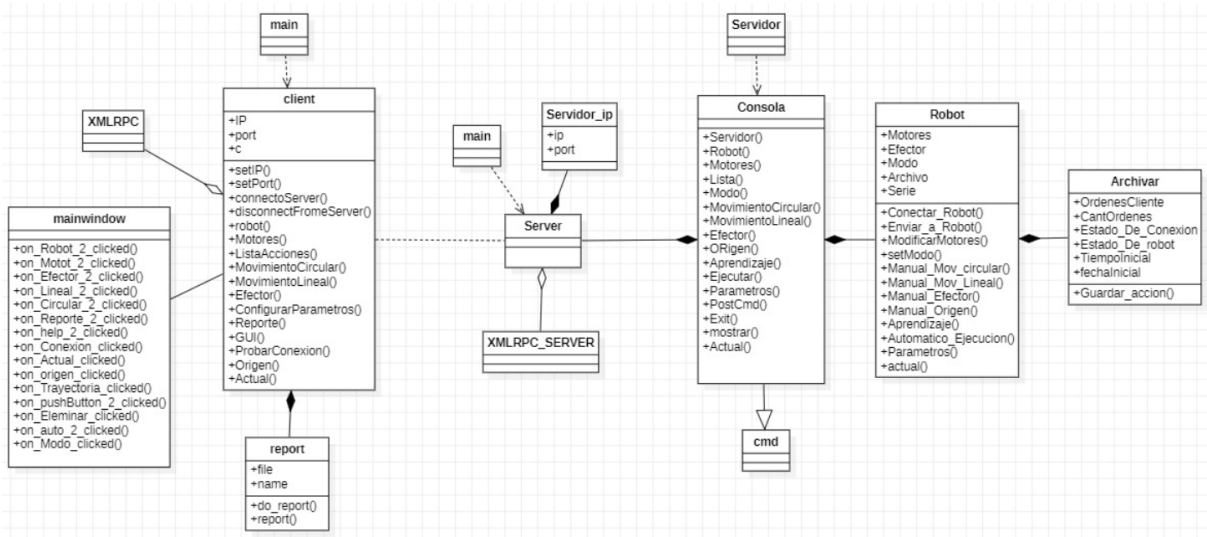
Del lado del servidor, la consola debe poder realizar un conjunto de funciones para controlar. Debe haber un servidor XML-RPC, capaz de ejecutar las mismas operaciones solicitadas en forma remota y la aplicación debe heredar de la clase Cmd.

Desde el lado del cliente, la interfaz debe ofrecer los mismos comandos que el panel de control del servidor y un reporte mostrando el estado de conexión, cantidad de órdenes, estado del Robot, el instante de inicio, fecha y hora de realización y una lista detallada de todas las acciones del robot y quien las solicita, ya sea cliente o servidor.

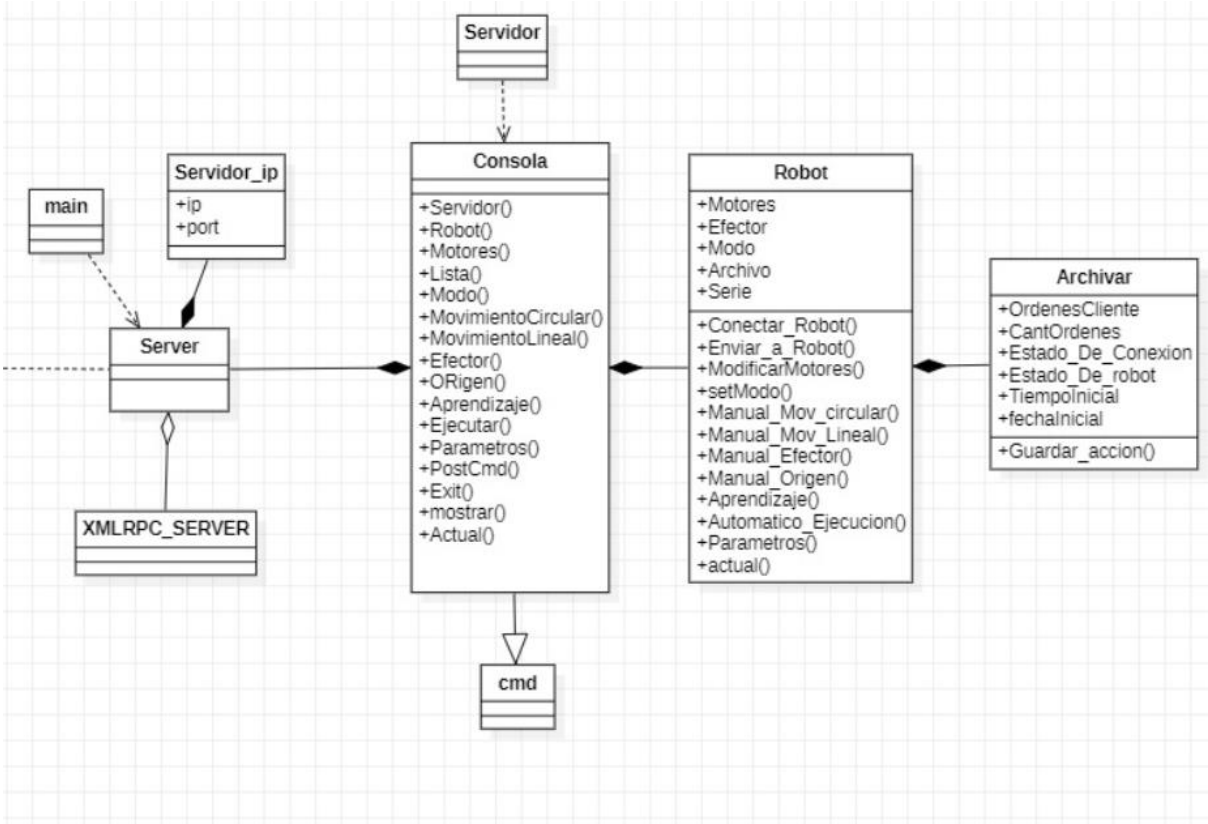
Conceptos y Fundamentos Utilizados en el Trabajo

- **Modelo Orientado a Objetos**

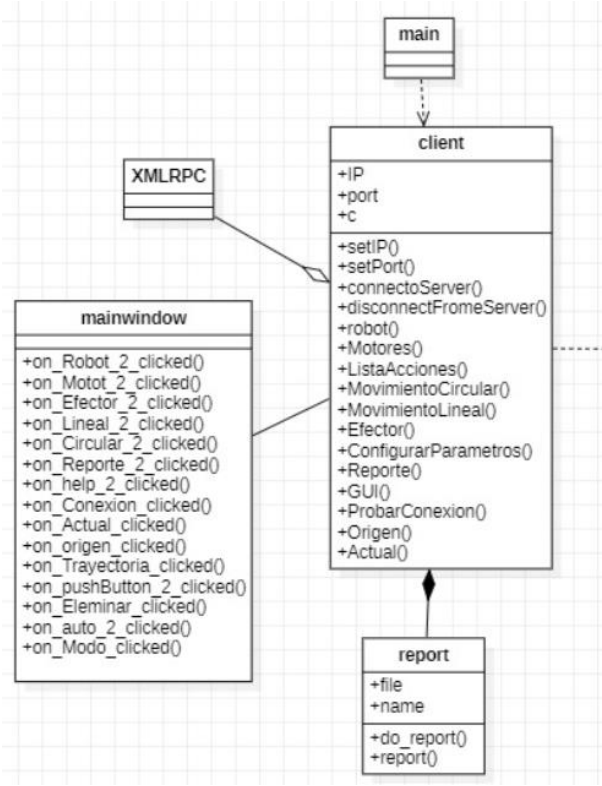
Como parte de la solución se creo un modelo UML representativo, donde se puede observar las clases creadas.



Lado Servidor



Lado Cliente



- **Clases y Objetos**

Las clases definidas para solucionar el problema planteado son:

Del lado del servidor:

- Robot: es la clase que se encarga de la comunicación serie con el Arduino.
- Consola: esta clase se encarga de realizar la entrada de ordenes por parte del lado servidor, además de conectar las ordenes del cliente con el robot.
- Server: aquí se ejecuta la conexión XMLRPC, esta recibe las ordenes del lado cliente las conecta con el servidor.
- Archivar: clase utilizada para el guardado de las acciones realizadas por el robot en un archivo .txt.
- Server_ip: clase que conserva los datos para realizar la conexión XMLRPC.

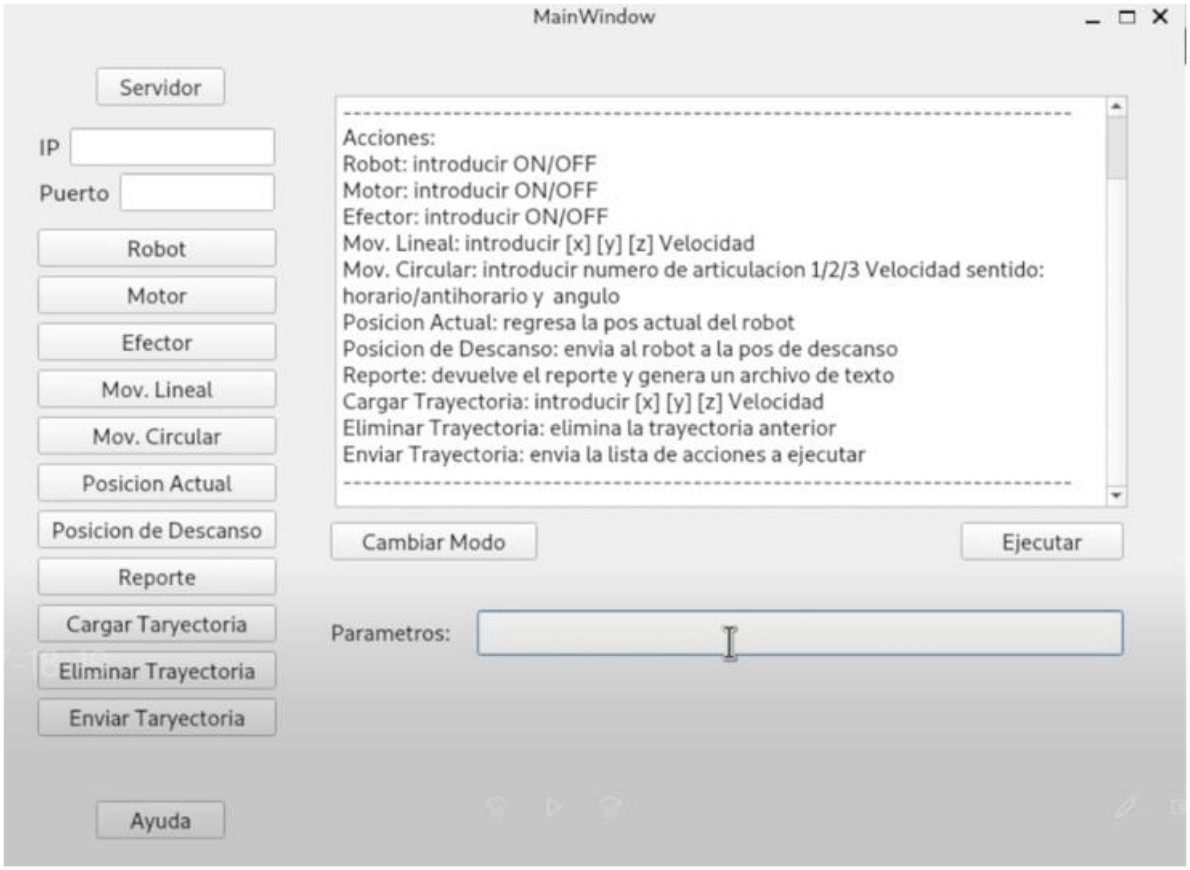
Del lado del cliente:

- Client: esta clase es la que recibe las ordenes por parte de los clientes y los envia para que sean ejecutadas por el servidor.
- Report: encargada de realizar el reporte de las acciones ejecutadas.

- **QT Creator**

Como parte del problema se proponia de manera opcional, la creacion de una interfaz grafica que opere del lado cliente. Decidimos utilizar QT creator, el cual nos proporcionaba las herramientas para poder crear una. Qt funciona como IDE para trabajar en C++, lo cual nos resultaba bastante conveniente.

La interfaz creada se muestra a continuación:



Como se puede apreciar, esta posee todas las funcionalidades del robot, mas un apartado de Reporte, que muestra todas las ordenes ejecutadas por el robot hasta el momento, junto con la cantidad de órdenes, hora en la que se inicio el trabajo y estado actual del robot.

- **Control de Errores mediante Manejo de Excepciones**

Las excepciones son la indicación de que se produjo un error en el programa. Se producen cuando un método no se ejecuta correctamente. En el código se puede definir qué hacer cuando ocurra una excepción para tratarla sin que falle todo el programa. En python, se usa “try” para lanzar la excepción, “except (tipo de excepción)” para especificar que hacer al lanzarse una excepción de un tipo determinado, “except” para definir qué

hacer al ocurrir una excepción no especificada, “else” para decidir qué hacer en caso de que no haya excepciones.

Para implementar el control de excepciones en C++, se usan las expresiones try, throw y catch. Se utiliza try para lanzar la excepción, se debe usar un bloque try para incluir una o más instrucciones que pueden iniciar una excepción. Para controlar las excepciones que se puedan producir, se implementa el bloque catch inmediatamente después de un bloque try. Cada bloque catch especifica el tipo de excepción que puede controlar.

Aplicación de excepciones en nuestro código del lado Servidor:

```
try:
    self.Serie = serial.Serial(port='COM9', baudrate=9600, timeout=2, write_timeout=2)
    self.Archivo.setEstado("Robot Conectado")
    return "INFO: ROBOT CONNECTED"
except:
    return "INFO: SERIAL ERROR"
```

Aplicación de excepciones del lado Cliente:

Aquí intentamos conectar el robot y en caso de no poder hacerlo el servidor no se detiene, si no que no se realiza la conexión.
Otro ejemplo es en el código cliente hecho en C++:

```
void MainWindow::on_auto_2_clicked()
{
    try {
        QString aux = ui-> lineEdit ->text();
        ui->label_2->setText(QString::fromStdString(user->Automatico(aux.toStdString())));
    } catch (...) {
        ui->label_2->setText(QString::fromStdString("PARAMETROS INCORRECTOS"));
    }
}
```

Aquí evitamos que el programa colapse por un error en la introducción de los parámetros. En el caso de que esto suceda, se retorna un mensaje con “Parámetros Incorrectos”.

- **Librerías utilizadas**

En nuestra implementación utilizamos las siguientes librerías de Python:

Sys: proporciona acceso a algunas variables utilizadas por el intérprete para manipular el entorno de tiempo de ejecución de Python.

Threading: los hilos permiten a nuestras aplicaciones ejecutar múltiples operaciones de forma concurrente en el mismo espacio de proceso.

Socket: provee clases específicas para manejar el transporte común.

Datatime: incorpora los tipos de datos date, time y datetime para representar fechas y funciones para manejarlas. En nuestro caso se utilizo para agregar la hora y fecha del inicio de actividad del robot.

Re: el modificador de búsqueda re.ASCII fuerza que los símbolos \w, \W, \b, \B, \d, \D, \s y \S se basen en el código ASCII para buscar coincidencias de textos, en lugar de basarse en Unicode.

Cmd: creamos nuestra CLI heredando de la clase Cmd de Python, que ofrece atributos y métodos muy útiles a la hora de diseñar una CLI, a la que solo debimos agregar nuestras funciones particulares.

Io: para crear el archivo de aprendizaje y leer un archivo en modo automático.

Os: Para resolver un problema de ejecución, en el que creamos y borramos un archivo temporal.

Xmlrpc.server: utilizada para crear la conexión del lado del servidor.

Las librerias utilizadas en C++

Xmlrpc.client: utilizada para crear la conexión del lado del cliente.

QT Creator: de aquí utilizamos varias herramientas de la librería propia del framework.

Cstdlib: esta librería fue importante para poder hacer la implementación de alarmas sonoras.

- **Alarmas sonoras y Alertas**

  Gracias a la librería Cstdlib que nos proporciona un medio para reproducir archivos .MP3 desde el código, pudimos implementar alertas sonoras en las acciones de prendido/apagado de robot, encendido/apagado de los motores y movimiento del efector final. Esto lo logramos colocando los archivos dentro de la carpeta donde se ubican los códigos y ejecutándolos cada vez que se detecta un cambio en alguno de los parámetros antes mencionados.

Uso de la aplicación específica

Acceso remoto cliente-servidor

Inicialmente se lanza el servidor, con su puerto y dirección IP correspondientes (los cuales fueron elegidos en el programa, por defecto, pero a su vez pueden ser cargados manualmente).

Posteriormente se corre el cliente, que solicita conexión al puerto y dirección IP anteriores. Al conectarse inicia la ejecución de nuestra GUI análoga en el lado del cliente, que permitirá acceder a todas las funciones del controlador.

Las funciones y sus implementaciones son explicadas con mayor detalle en el video.

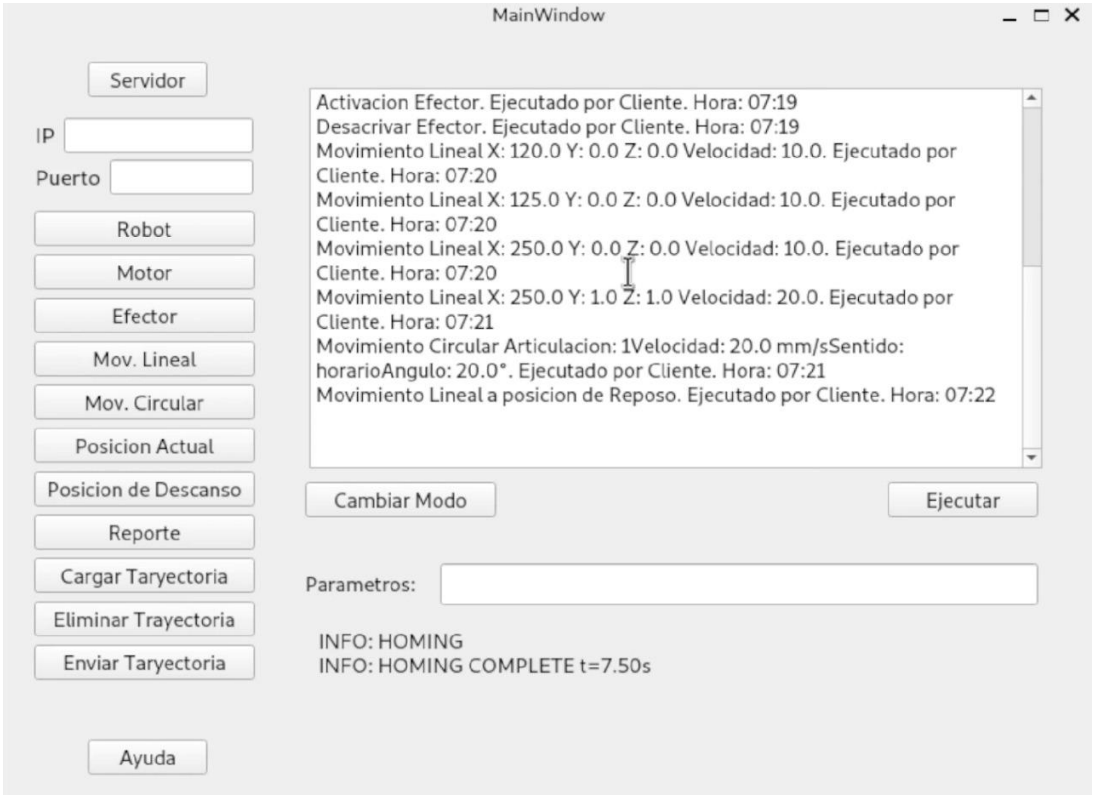
A continuación, vemos la ejecución del cliente y servidor:

Servidor:

```
PS C:\Users\brand\Desktop\Servidor> python servidor.py
Estado de Robot: Conectado
Servidor iniciado en el puerto ('192.168.1.38', 8900)
VENTANA DE COMANDOS ROBOT ABB IRB 460
>>lista
Conectar/Desconectar Servidor --> servidor + on/of
Conectar/Desconectar Robot --> robot + on/off
Encender/Apagar Motores --> Motores + on/off
Ayuda --> help
Reporte --> Reporte
Si modo manual ON --> Movimiento Circular --> MovimientoCircular + Articulacion(1/2/3) + velocidad + horario/antihorario + angulo
Si modo manual ON --> Movimiento lineal --> movimientolineal + X + Y + Z + velocidad
Si modo manual ON --> Activar Efector Final --> efector on/off
Si modo manual ON --> Volver a posicion original --> origen
Si modo manual ON --> Aprendizaje de trayectoria --> aprendizaje
Si modo automatico ON --> Ejecutar archivo --> ejecutar
>>servidor off
Servidor desactivado
>>servidor on
Servidor iniciado en el puerto ('192.168.1.38', 8900)
>>robot on
INFO: ROBOT ALREADY CONNECTED
>>robot off
INFO: ROBOT DISCONNECTED
>>motores on
INFO: NO ROBOT CONNECTED
>>robot on
INFO: ROBOT CONNECTED
>>motores off
INFO: MOTORS OFF
>>movimientocircular 1 10 horario 90
INFO: MOTORS OFF
>>motores on
INFO: MOTORS ON
>>movimientocircular 1 10 horario 90
INFO: CIRCULAR MOVE: [ART: 1 VEL: 10.0 mm/s SENT: horario, ANG: 90.0°]
>>movimientolineal 100 0 0 10
ERROR: POINT IS OUTSIDE OF WORKSPACE

>>movimientolineal 250 0 0 90
INFO: LINEAR MOVE: [X:250.00 Y:0.00 Z:0.00 E:0.00] t=0.28s

>>origen
INFO: HOMING
INFO: HOMING COMPLETE t=7.50s

>>actual
INFO: ABSOLUTE MODE
INFO: CURRENT POSITION: [X:0.00 Y:170.00 Z:120.00 E:0.00]
```

```
>>efector on
INFO: GRIPPER ON

>>efector off
INFO: GRIPPER OFF

>>aprendizaje
Nombre del archivo: Prueba2
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   motores
Ingrese los parametros (on/off)  on
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   modo
Ingrese los parametros (manual/automatico)  manual
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   movimientolineal
Ingrese X Y Z Velocidad  10 20 30 50
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   movimientocircular
Ingrese Articulacion Velocidad Sentido Angulo  1 100 horario 90
['1', '100', 'horario', '90']
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   efector
Ingrese on/off  on
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   efector
Ingrese on/off  off
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   origen
Ingrese una accion  (motores | modo | movimientoCircular | movimientoLineal | efector | origen | carga | 0 para salir) :
   0
Aprendizaje finalizado
>>ejecutar Prueba1
INFO: ROBOT IN MANUAL MODE
```

Cliente:


MainWindow
Servidor
IP
Puerto
Robot
Motor
Efector
Mov. Lineal
Mov. Circular
Posicion Actual
Posicion de Descanso
Reporte
Cargar Taryectoria
Eliminar Trayectoria
Enviar Taryectoria
Ayuda
Activacion Efector. Ejecutado por Cliente. Hora: 07:19
Desacrivar Efector. Ejecutado por Cliente. Hora: 07:19
Movimiento Lineal X: 120.0 Y: 0.0 Z: 0.0 Velocidad: 10.0. Ejecutado por Cliente. Hora: 07:20
Movimiento Lineal X: 125.0 Y: 0.0 Z: 0.0 Velocidad: 10.0. Ejecutado por Cliente. Hora: 07:20
Movimiento Lineal X: 250.0 Y: 0.0 Z: 0.0 Velocidad: 10.0. Ejecutado por Cliente. Hora: 07:20
Movimiento Lineal X: 250.0 Y: 1.0 Z: 1.0 Velocidad: 20.0. Ejecutado por Cliente. Hora: 07:21
Movimiento Circular Articulacion: 1Velocidad: 20.0 mm/sSentido: horarioAngulo: 20.0°. Ejecutado por Cliente. Hora: 07:21
Movimiento Lineal a posicion de Reposo. Ejecutado por Cliente. Hora: 07:22
Cambiar Modo
Ejecutar
Parametros:
INFO: HOMING
INFO: HOMING COMPLETE t=7.50s

Comentarios finales

Al comienzo nos costó mucho trabajo entender cómo afrontar el problema debido a que no habíamos realizado nada semejante previamente. Del lado servidor al principio tuvimos problemas de adaptación entre el Arduino y el servidor, el cual se resolvio agregando tiempos de espera y lineas de vaciado del buffer para evitar inconvenientes. Del lado cliente tuvimos problemas para conectarnos ya que las librerías están en su mayoría desarrolladas para Linux, lo cual generaba confusión y errores complejos, obligándonos a usar Debian.

Utilizamos el framework de Qt Creator, el cual tiene un tiempo de prueba muy corto lo que nos llevó a tener que crear muchas cuentas para poder realizar la implementación de la GUI.

Algunas cuestiones de extensión para mejor el código puede ser la implementación de hilos paralelos para el manejo de las alarmas sonoras. Estas solo funcionan cuando la ejecución de la orden que las activa son llamadas por el cliente.

Referencias

- Implementar un Servidor RPC (Remote Procedure Call) con XML-RPC
- Código G-Code para rotaciones
- XML-RPC: llamadas a procedimientos remotos en formato XML
- Tutorial Qt Creator - QMediaPlayer (Reproductor de audio en C++)
- Reproducir Sonido en C++
- Como Programar C++ - Deitel
- Diseño Orientado a Objetos - Alarcón