



UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO



FACULTAD DE
INGENIERÍA

TRABAJO PRÁCTICO INTEGRADOR:

XmlRpc, Python y C++

Programación Orientada a Objetos

Integrantes:

Bruno Peralta

Mario Bustillo

Esteban Vila

Docente:

Mg. Ing. César Omar Aranda

Carrera:

Ingeniería en Mecatrónica

1. Introducción

El objetivo principal del trabajo es el control de un Robot tipo RRR (robot de tipo rotacional, con tres articulaciones de tipo rotacional) de tres grados de libertad, también denominado antropomórfico debido a las similitudes entre su estructura y el brazo humano. Se emplean en operaciones de ensamblaje, vaciado de metales, fresado, soldadura a gas, soldadura al arco, y pintura con spray.

1.1 Descripción breve del problema

Para este trabajo se busca, dentro del Dominio de la Programación Orientada a Objetos, la Manipulación Remota de un Robot tipo RRR de 3 grados de libertad con efector final, mediante el desarrollo de un aplicativo para controlarlo.

El objetivo general del presente proyecto consiste en el diseño e implementación del sistema de control remoto del robot mencionado que posee un efector final a través del protocolo XML-RPC, haciendo uso de conocimientos de C++ y Python adquiridos previamente y en clase

1.2 Ejemplos de uso

Un ejemplo posible para este tipo de robot es un robot de ensamblado de piezas mecánicas o de pintura en una automotriz (ver Figura 1)



Figura 1. Ejemplo de brazo RRR de la marca ABB

2. Marco Teórico

Para el desarrollo del programa Cliente-Servidor se aplicaron conceptos de Clases, Objetos, Herencia, Colecciones de Objetos, Streams de E/S, Control de errores mediante Manejo de Excepciones.

Además se aplicaron algunos conceptos que se tuvieron que reforzar, tales como: Vectores, Manejo de archivos, Cmd y Comunicación Serial;

2.1. Vectores

Un vector es una zona de almacenamiento contiguo que forma parte de una matriz y contiene elementos del mismo tipo. Un vector con una cantidad fija de elementos se denomina vector estático, en cambio un vector con una cantidad variable de elementos se denomina vector dinámico. Otra característica importante es la ubicación de los elementos dentro de ellos, las posiciones de los vectores ocupan espacios de memoria contiguos. Para implementar vectores en C++ se incluye la librería `<vector>`, en cambio para Python, que es un lenguaje dinámicamente tipado, no se debe incluir librería alguna para su uso.

2.2. Manejo de archivos

Se realizó tanto en el cliente como en el servidor, con el objetivo de poder guardar un historial de los movimientos pedidos y realizados por el robot. También se utilizó en el modo automático para poder leer cada consigna en un archivo en formato `.txt` y realizar los movimientos. Para leer los archivos en C++ se usó la clase `ifstream`. Para el manejo de archivos en el servidor se hizo uso del módulo `os`, con el cual se accede al directorio; también para abrir archivos con el tipo de permisos que se requieran (por ejemplo de lectura o escritura) con el método `open()`.

2.3. Cmd

Se usó en el servidor. Esta clase provee un framework simple para escribir intérpretes de comandos. Se suele usar la instancia o subclase `cmd`, que es útil para heredar los métodos `Cmd` y encapsularlos.

3. DESARROLLO

Para la realización del trabajo se requiere el control del mismo que puede ser de forma local en el servidor o de forma remota a través de un cliente el cual envía la consigna al servidor el cual se comunica con el robot y la devuelve al cliente la salida del mismo. Posteriormente se hace uso de la gestión de archivos para un manejo automático del robot. El cliente debe guiarse con los comandos que se le muestra en pantalla y cuando se trabaja de forma local en el

servidor se puede consultar la secuencia correcta de uso y la documentación de cada función.

4. CONEXIÓN XMLRPC

Se procedió a realizar la conexión entre el servidor y el cliente, haciendo uso para ello del protocolo XMLRPC (Remote procedure call), que estructura los datos mediante XML y para su transmisión utiliza el protocolo HTTP. Para poder usar este protocolo en el servidor, cuyo código está escrito en Python, se hizo uso del paquete 'xmlrpc' de Python, que incluye módulos para cliente y para servidor. El módulo 'xmlrpc.server' provee un framework para servidores básicos. Se usó la clase SimpleXMLRPCServer provista por la librería. Para la comunicación por parte del cliente, desarrollado en C++, se usó la librería 'xmlrpc++' versión 1.54.06, XmlRpc++ es una implementación en C++ del protocolo XMLRPC. El IP utilizado es el *de la computadora respecto a la red doméstica* y se especifica para su uso el puerto 8093.

5. Instrucciones G-Code

El robot está configurado para recibir comandos estándar G-Code. En el cuadro 1 se enlistan las instrucciones disponibles a la fecha de entrega.

Instrucción	Descripción
G1 X0 Y0 Z0 Fv	posición (a,b,c) y velocidad de movimiento f
M3	Activa gripper
M5	Desactiva gripper
G28	Homing
M114	Posición actual
M119	Estado de los endstops
M17	Activa los motores
M18	Desactiva los motores
M999	Reanuda el movimiento

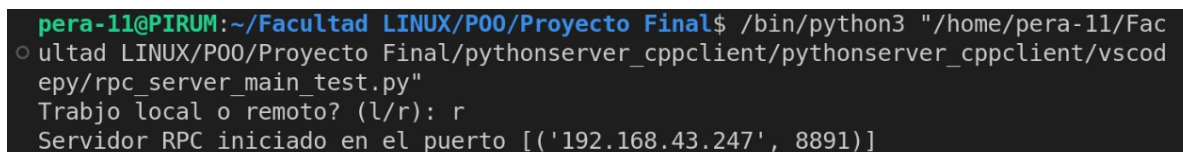
G90	Movimiento absoluto
G91	Movimiento relativo
G92	Establece la posición actual como (0,0,
G4 sb	Espera b segundos

Tabla 1. Tabla de instrucciones G-Code disponibles a la fecha de entrega

6. Secuencia de Inicialización

Para ejecutar el servicio cliente-servidor mediante conexión XmlRpc y conexión serial Arduino - Servidor - Cliente, se debe seguir la siguiente secuencia de pasos:

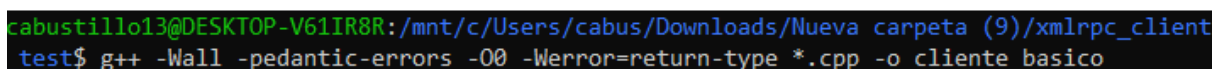
1. Desde el servidor se debe ejecutar el archivo "rpc_server_test.py", en el cual debemos seleccionar por consola si se trabajará de manera local (l) o con conexión remota desde un cliente (r).
2. Por consola se mostrará el IP y el puerto con el que se podrá hacer la conexión.



```
pera-11@PIRUM:~/Facultad LINUX/P00/Proyecto Final$ /bin/python3 "/home/pera-11/Fac
ultad LINUX/P00/Proyecto Final/pythonserver_cppclient/pythonserver_cppclient/vscod
epy/rpc_server_main_test.py"
Trabjo local o remoto? (l/r): r
Servidor RPC iniciado en el puerto [('192.168.43.247', 8891)]
```

Imagen 1. Captura de pantalla de la inicialización por consola del servidor.

3. Los parámetros de IP y puerto se deben actualizar en el archivo "main_cliente.cpp"
4. Una vez definido la IP y puerto de trabajo, compilamos el archivo "main_cliente.cpp" por consola. No debe mostrar ningún error en el programa.



```
cabustillo13@DESKTOP-V61IR8R:/mnt/c/Users/cabus/Downloads/Nueva carpeta (9)/xmlrpc_client
_test$ g++ -Wall -pedantic-errors -O0 -Werror=return-type *.cpp -o cliente_basico
```

Imagen 2. Captura de pantalla de compilación por consola del cliente

5. Posteriormente ejecutamos el archivo

```
cabustillo13@DESKTOP-V61IR8R:/mnt/c/Users/cabus/Downloads/Nueva carpeta (9)/xmlrpc_client_test$ ./cliente_basico 192.168.43.247 8891
```

Imagen 3. Captura de pantalla de inicialización por consola del cliente, dado un IP y puerto,

6. Por consola deberá aparecer el siguiente menú

```
=====
                                ORDENES
1 - Control Automatico
2 - Control Manual
3 - Generar reporte
4 - Mostrar Comandos
5 - Verificar Comando
6 - Conexión/Desconexión Arduino
0 - Salir
=====
Ingrese el número de orden a realizar:
```

Imagen 4. Captura de pantalla del menú del cliente mostrado por consola

7. El primer paso es hacer la conexión remota al Arduino. Esta se hace ingresando el número de opción correspondiente de la secuencia "Conexión/Desconexión Arduino"(1)>"Conectarse al Arduino"(1). Si se ha logrado realizar la conexión, por consola se verá el mensaje "Arduino Conectado".

```
=====
                                ORDENES
1 - Control Automatico
2 - Control Manual
3 - Generar reporte
4 - Mostrar Comandos
5 - Verificar Comando
6 - Conexión/Desconexión Arduino
0 - Salir
=====
Ingrese el número de orden a realizar: 6
=====
                                Movimiento del Efector
1 - Conectarse al Arduino
2 - Desconectarse del Arduino
0 - Volver al Menu de Inicio
=====
Ingrese el número de orden a realizar: 1
Arduino conectado
```

Imagen 5. Secuencia de pasos para inicializar la conexión Cliente - Servidor - Arduino

7. Secuencia en modo Manual de control de Robot desde el Cliente:

Se tiene la posibilidad de ejecutar comandos de manera manual para controlar el robot desde la conexión del cliente. Al seleccionar la opción de Control Manual (2) desde el menú de órdenes, se podrá acceder al siguiente menú con comandos predefinidos.

```
=====
                                MODO MANUAL
1 - Movimiento Circular
2 - Movimiento Lineal
3 - Accionar Efector
4 - Homing
5 - Modo Aprendizaje
6 - Mostrar Comandos
0 - Volver al Menu de Inicio
=====
Ingrese el número de orden a realizar:
```

Imagen 6. Menú de secuencias de comandos manual para el cliente

Si se desea ejecutar uno de los comandos, se debe ingresar el número de la opción correspondiente. Por ejemplo, para ejecutar Homing se ingresa por consola el número 4.

```
=====
                        MODO MANUAL
1 - Movimiento Circular
2 - Movimiento Lineal
3 - Accionar Efecto
4 - Homing
5 - Modo Aprendizaje
6 - Mostrar Comandos
0 - Volver al Menu de Inicio
=====
Ingrese el número de orden a realizar: 4
```

Imagen 7. Menú de Modo manual con el ingreso por consola de la opción 4 (Homing).

El servidor devolverá el mensaje que otorga la placa de Arduino para este comando G-Code.

```
Respuesta del Servidor:
INFO: HOMING
INFO: HOMING COMPLETE t=4.50s
```

Imagen 8. Respuesta del servidor bajo el comando "Homing"

De la misma manera se ejecutan los otros comandos manuales.

8. EJECUCIÓN DE LA APLICACIÓN

A continuación se muestra una serie de capturas de pantalla del funcionamiento del lado del cliente, el menú se presenta de la siguiente manera:

```
Ingrese el número de orden a realizar: 1
ControlAutomatico();

=====
                        MODO AUTOMATICO
1 - Cargar rutina
0 - Volver al Menu de Inicio
=====
Ingrese el número de orden a realizar: 1
CargarRutina();

Ingrese archivo para leer:
ma.txt
```

Imagen 9. Menú de Control Automático para ingresar un archivo.txt para la ejecución automática de comandos


```
Ingrese archivo para leer:
ma.txt
Ejecutando G28

Respuesta del Servidor:

INFO: HOMING
INFO: HOMING COMPLETE t=2.00s

Ejecutando M5

Respuesta del Servidor:

INFO: GRIPPER OFF

Ejecutando M3

Respuesta del Servidor:

INFO: GRIPPER ON
```

Imagen 10. Control Automático de Resultados por consola de la ejecución de archivo con comandos G-Code para la ejecución en automático.

```
=====
Ingrese el número de orden a realizar: 3
GenerarReporte();

=====
                        REPORTE
Coordenadas Actuales: X:0.00Y:170.00Z:120.00
Cantidad de Comandos Enviados: 1
Cantidad de Comandos Exitosos: 1
Cantidad de Comandos con Errores: 0
Ultimo comando enviado: G28
Estado de la última operación: Comando exitoso
```

Imagen 11. Obtener un reporte del estado actual del robot que describe la cantidad de comandos exitosos, la cantidad de comandos erróneos y la última operación ejecutada.

```

=====
                                ORDENES
1 - Control Automatico
2 - Control Manual
3 - Generar reporte
4 - Mostrar Comandos
5 - Verificar Comando
6 - Conexión/Desconexión Arduino
0 - Salir
=====
Ingrese el número de orden a realizar: 0

Desconexión Arduino
Arduino desconectado

Desconexión con el Servidor

Cliente desconectado
Fin del programa - Funcional!

```

Imagen 12. La desconexión del cliente respecto a la conexión serial con el Arduino y la conexión XmlRpc con el Servidor.

9. Diagramas UML

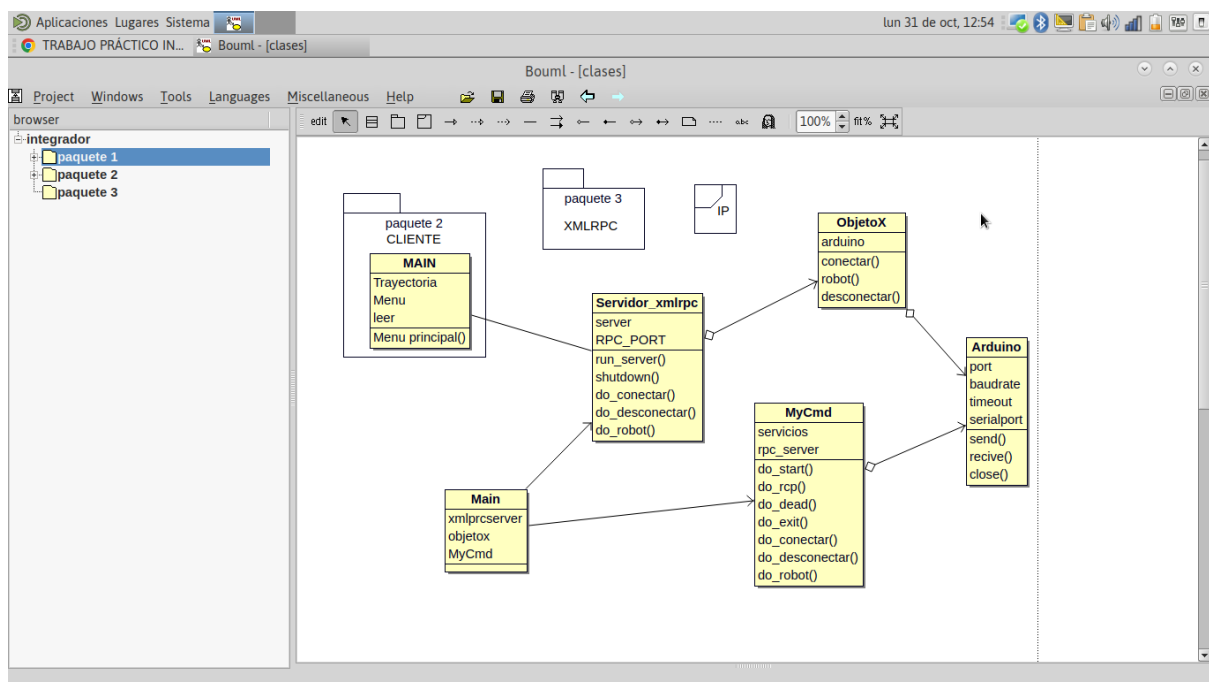


Figura 2. Diagrama del lado del servidor

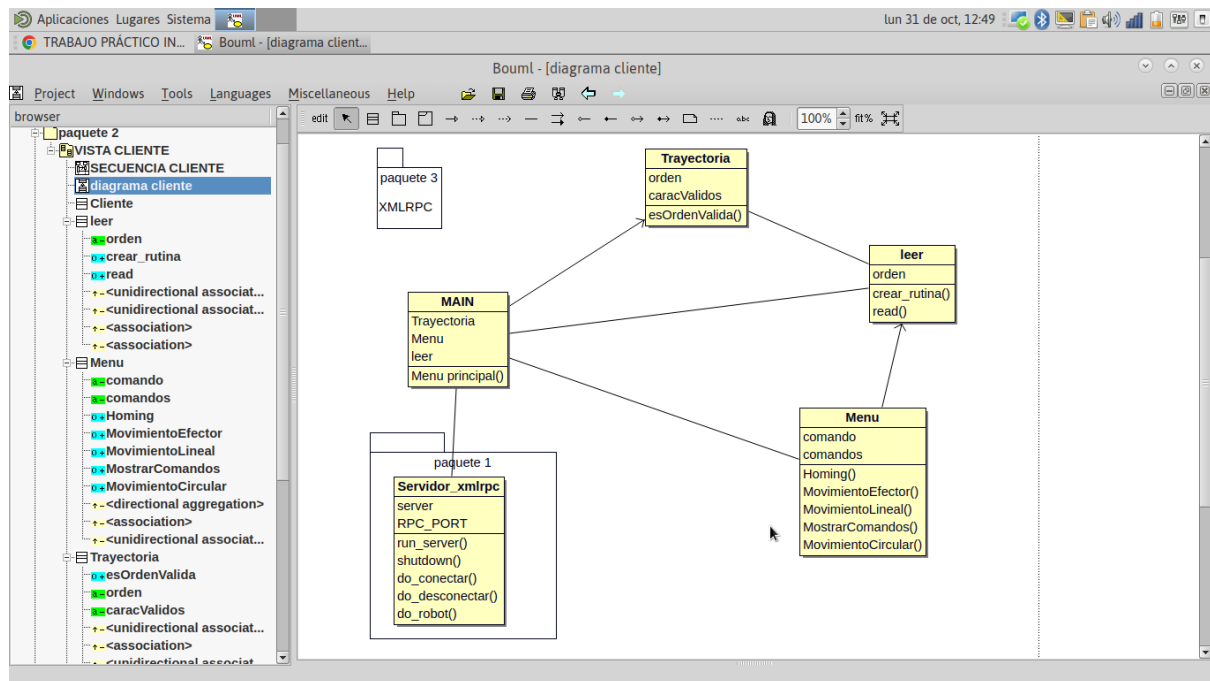


Figura 3. Diagrama del lado del cliente.

10. Comentarios y Conclusiones

Analizando el proceso seguido para llegar a una aplicación funcional vimos que las mayores dificultades se presentaron en aprender a utilizar la librería XML-RCP, ya que contenía muchos errores que tuvimos que solucionar para su correcto funcionamiento y tuvimos bastantes problemas de conexión con la misma ya que no todos los router wifi permiten realizar este tipo de conexiones y los proveedores no nos dieron respuesta, aunque una vez solucionado esto, enlazar el código de C++ realizado en VSCode fue relativamente sencillo compilado desde una consola de linux. Estamos muy conformes con la modularidad que logramos en la implementación. Como posibles mejoras nos gustaría implementar un visualizador 3D para simular los movimientos del robot, pulir el diseño de la interfaz.

11 Referencias

<https://docs.python.org/3/>
<http://www.cplusplus.com/>
<http://xmlrpc-c.sourceforge.net/>
<https://docs.python.org/3/library/xmlrpc.html>
<https://wiki.qt.io>
<https://Qcustomplot.com>
<https://es.stackoverflow.com>