Programación Orientada a Objetos

Proyecto Final:
Manipulación remota de un robot

<u>INTEGRANTES:</u>

ESPERLAZZA LUCIANO 12277

LLANO JAVIER 11711

24/02/2023

INDICE

1) INTRODUCCION

En este proyecto se realizó el desarrollo de una aplicación del tipo cliente-servidor para controlar un robot de 3 grados de libertad con efector final, utilizando Programación Orientada a Objetos y modelos de capas. El robot específico utilizado es del tipo 3DF, con configuración RRR y efector final simple, en este caso una pinza, y tiene ciertas limitaciones físicas y velocidades máximas establecidas. El controlador del robot recibe órdenes y devuelve mensajes con información de estado.

La aplicación se controla mediante un panel de control a través de una consola, utilizando interfaz de línea de comandos o CLI. Esta permite realizar diversas acciones, tales como la activación/desactivación del robot, el listado de los comandos de control, la ayuda al operador con la sintaxis requerida para los comandos, y diversas funcionalidades de movimiento manual y automático del robot que incluyen el manejo de archivos de forma local y remota. Además, se implementa un servidor XML-RPC, donde del lado del cliente la aplicación ofrece una interfaz principal que incluye un panel de control del robot con funcionalidades similares a las de la interfaz de la consola del lado servidor.

En general, este proyecto permite un control completo y eficiente del robot de 3DF con efector final, con una interfaz de usuario intuitiva y fácil de usar, y con diversas funcionalidades opcionales para mejorar la experiencia del usuario, como lo son el aprendizaje de rutinas y su posterior ejecución para funcionamiento automático.

2) DESARROLLO

Fundamentos

El paradigma de programación orientada a objetos (POO) es un modelo de programación que se basa en el concepto de objetos. Los objetos son instancias de una clase, y cada clase define el comportamiento y las propiedades de los objetos que se crean a partir de ella.

Basados en esto desarrollamos nuestra aplicación, respetando el concepto de capas, el cual permite tener un código más fácil de entender, modificar y escalar. Además, permite la utilización de la aplicación independientemente de si usamos una interfaz por línea de comandos (CLI) o una interfaz gráfica (GUI).

Los principales conceptos de POO aplicados fueron:

- Herencia: se utilizó la herencia para la implementación del intérprete de comandos. La consola de comandos hereda de la clase **Cmd** la cuál contiene los métodos necesarios para construir este tipo de interfaces. También se aplicó el concepto de herencia para la creación del servidor, heredando de **SimpleXMLRPCServer**. Se utilizó de esta forma para poder agregar funciones propias al servidor, así como un manejo de respuestas o RequestHandler propio.
- Encapsulamiento: en el lado del Cliente, realizado en C++ se utilizaron los conceptos de encapsulamiento que nos permitió ocultar el estado interno de los objetos y solo exponer lo que es necesario para su uso externo.

Se realizó una clase para validar los comandos ingresados, y un manejo propio de excepciones y errores que controlan los límites físicos del robot, para evitar ingresar datos no válidos y consecuentemente la detención prematura de la aplicación.

Estructura

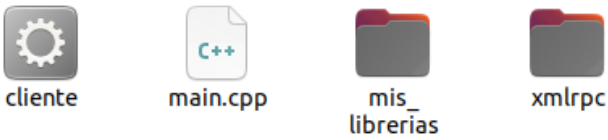
Como se mencionó en el punto anterior, se utilizó el concepto de capas para mantener un orden en la aplicación y a su vez facilitar su comprensión.

Contamos con dos carpetas contenedoras para cada parte del proyecto:

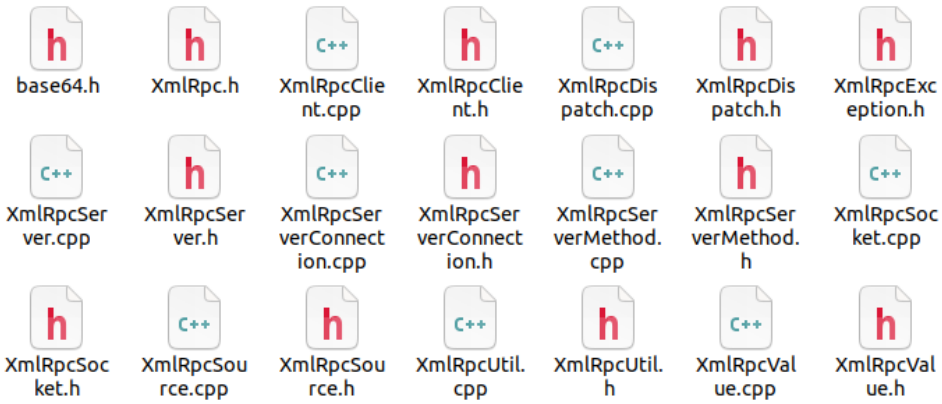
CLIENTE

Contamos con un archivo principal llamado main.cpp que es quien se encarga de crear las instancias de los objetos y enlazarlos entre sí, el archivo "cliente" es quien se encarga de ejecutar la aplicación, es decir, es el código ya compilado.

Hay dos subcarpetas, una llamada xmlrpc que contiene todos los archivos necesarios para utilizar y compilar esta librería:

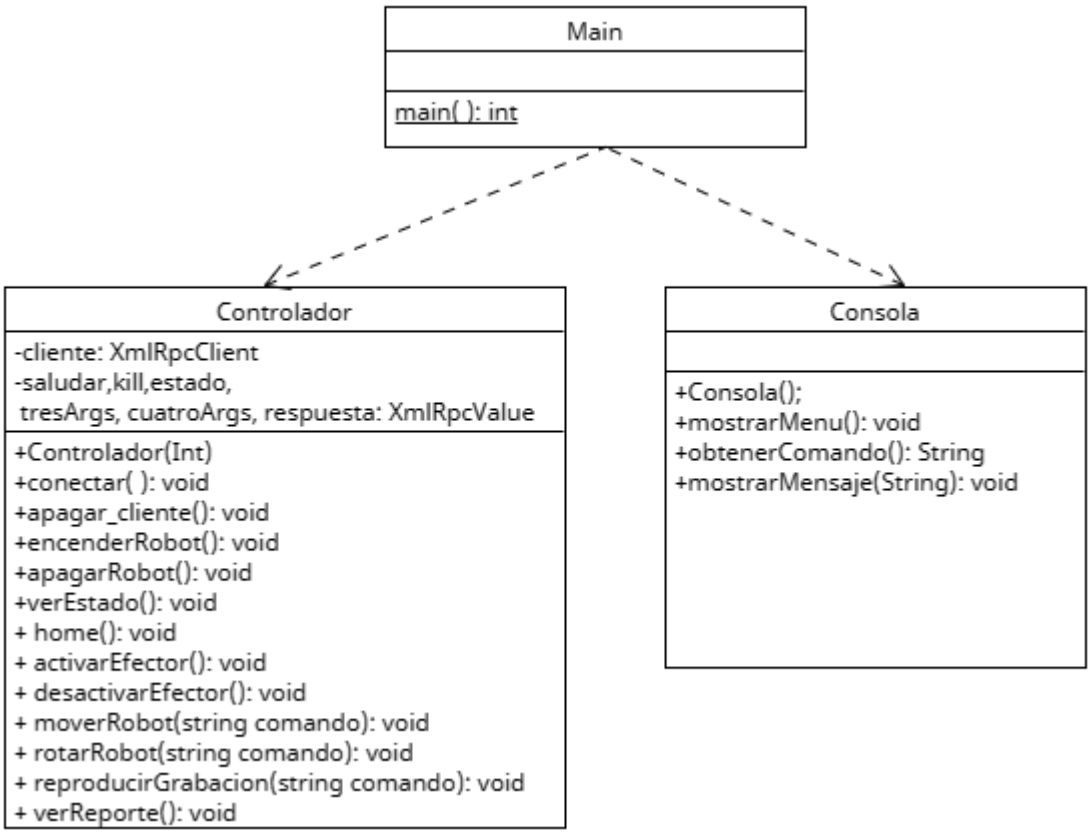


Xmlrpc:



Y mis_librerias que contiene los archivos de cabecera y de código propios de nuestras clases:

DIAGRAMA DE CLASES



SERVIDOR

Del lado servidor contamos con siete archivos, siendo main.py el encargado de enlazar todas las clases, crear instancias de las mismas y ejecutar el código.

En esta carpeta también se almacenarán los archivos de texto relacionados a las rutinas para el modo de control automático, además de los reportes en formato XML para su posterior lectura.

DIAGRAMA DE CLASES

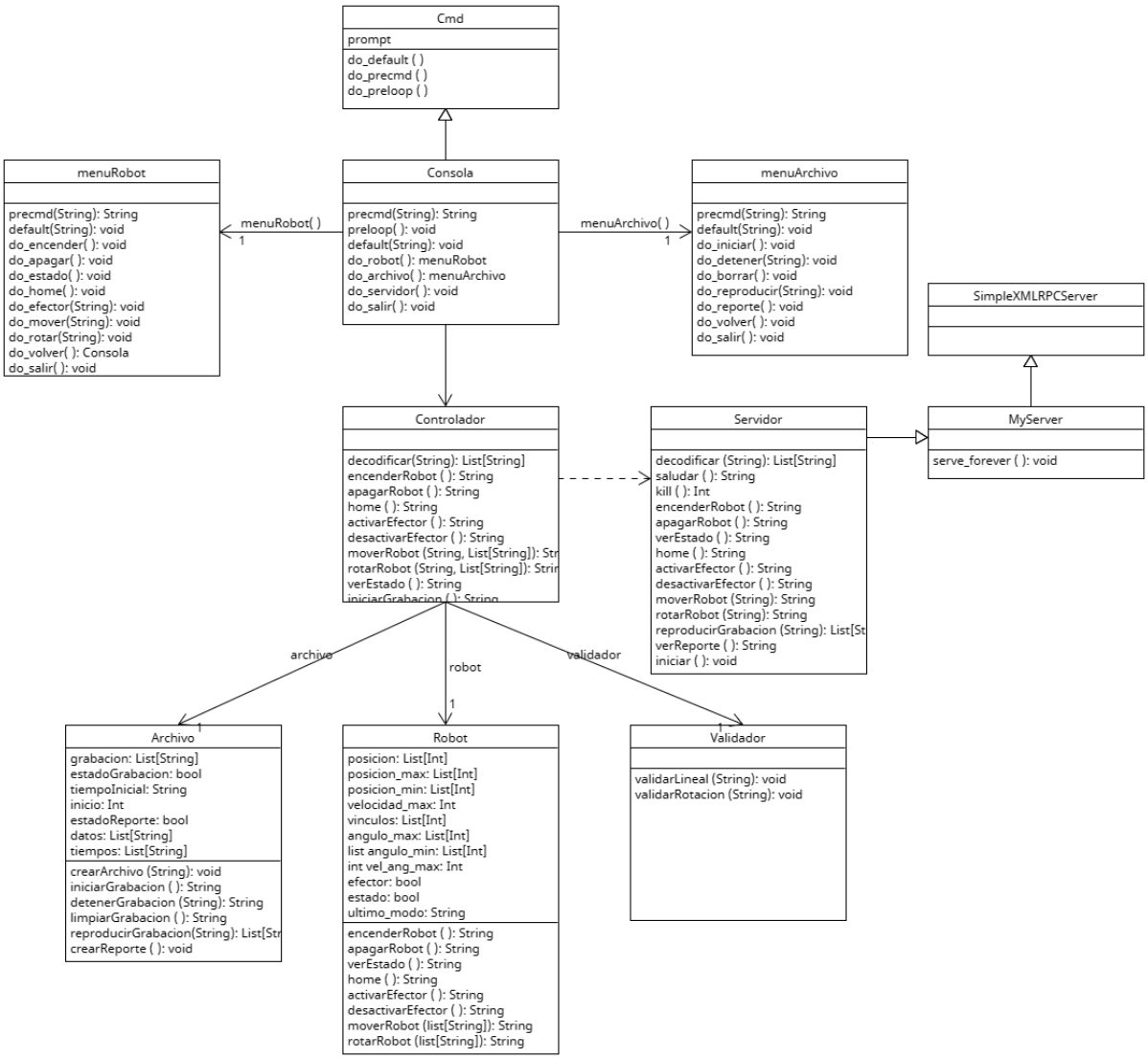


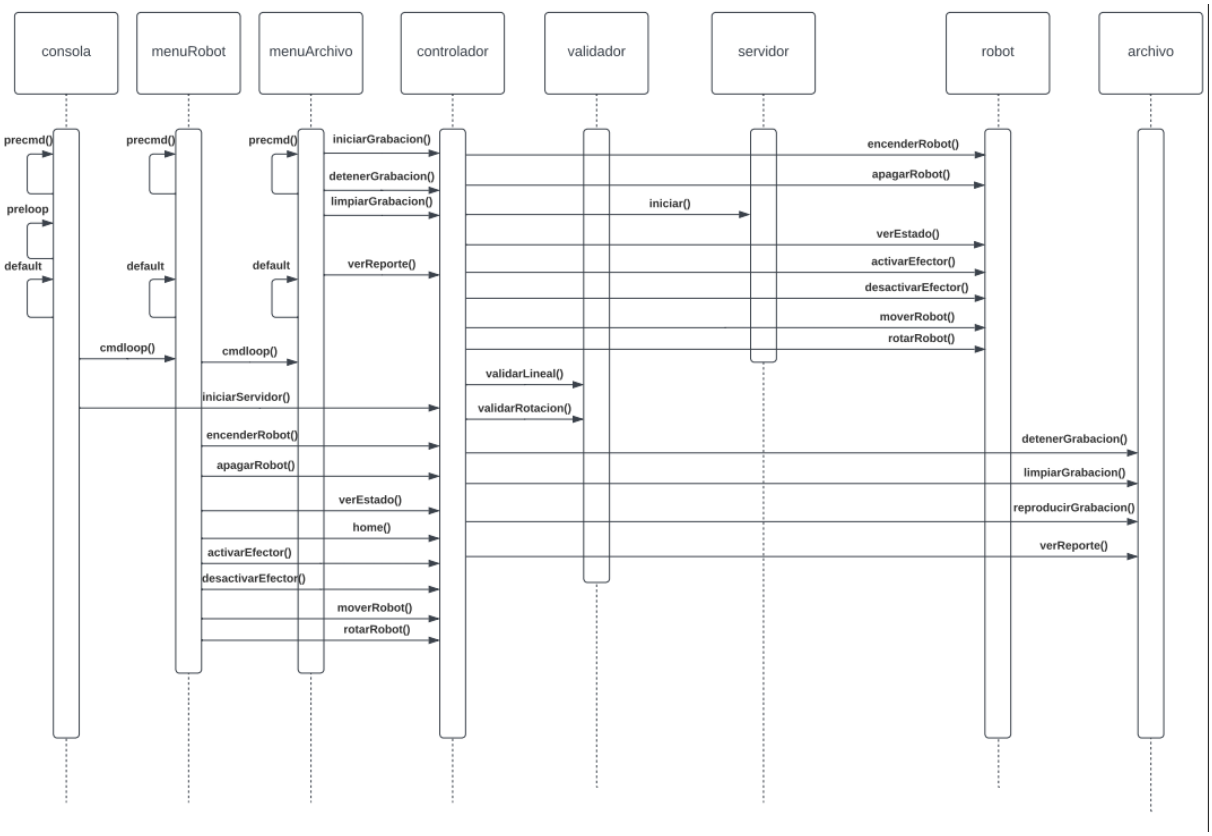
DIAGRAMA DE SECUENCIA

El diagrama de secuencia modela cómo interactúan los objetos, representa el intercambio de mensajes entre los distintos objetos del sistema para cumplir con una funcionalidad. En este caso podemos representar el comportamiento dinámico del sistema de información mediante objetos que contienen instancias de sus respectivas clases.

Estos objetos mediante métodos proporcionados por las instancias se logran comunicar con los otros.

También ciertos objetos interactúan con ellos mismos (precmd( ) por ejemplo).

Aquí podemos ver el recorrido entre objetos que tiene una orden emitida por el usuario. Vemos que varios métodos se envían desde los menús hacia el controlador y este los envía a robot y archivo, esto se debe a que el objeto controlador actúa de intermediario con entre el modelo y la vista.



Software y librerías

La realización del proyecto fue llevada a cabo con el IDE de programación Visual Studio Code, y se utilizó una máquina virtual creada en Oracle VM VirtualBox con un sistema operativo Linux, específicamente Ubuntu 22.04.1.

La versión de Python utilizada para el lado del servidor fue la 3.10.6 de 64-bits.

Fue necesario el uso de esta maquina virtual debido a incompatibilidades entre la librería XML-RPC y el sistema operativo Windows en el lado del cliente (programado en C++).

XML-RPC

XML-RPC es una librería que permite la comunicación remota de procedimientos en lenguaje C++ y Python. XML-RPC se basa en la tecnología de protocolo HTTP y utiliza el formato XML para codificar los datos transmitidos.

La librería XML-RPC permite la creación de clientes y servidores que pueden comunicarse entre sí a través de Internet o de una red local. Un cliente XML-RPC envía una solicitud HTTP POST al servidor XML-RPC, que a su vez devuelve una respuesta XML que contiene los datos solicitados. El servidor XML-RPC procesa la solicitud del cliente y ejecuta el procedimiento solicitado, devolviendo el resultado a través de la respuesta HTTP.

Como se mencionó anteriormente, se hizo uso de esta librería en Linux para el lado cliente, debido a que debía realizarse en el lenguaje C++ y no era posible utilizarla en Windows debido incompatibilidades y falta de librerías base.

Para la creación del cliente se hizo uso del método XmlRpcCliente, y para las respuestas y mensajes XmlRpcValue. Los mensajes son enviados al servidor mediante el método **excecute** que permite en llamar a un procedimiento remoto en un servidor y obtener su resultado o respuesta.

Por otro lado, para el servidor se hizo uso de xmlrpc.server y se utilizó la clase SimpleXMLRPCServer para crear nuestro objeto de tipo servidor. Se utilizaron algunos métodos heredados como pueden ser el método de iniciación o constructor __init__ o el método serve_forever ().

DATETIME y TIME

Se utilizaron las librerías **datetime** y **time** para obtener los instantes de tiempo y calcular los tiempos de ejecución del programa. En el caso de Datetime se utilizó el método **datetime.datetime.time ()** que devuelve la fecha y la hora en formato AA-MM-DD HH:MM:SS. Por otro lado, la librería time permite obtener un tiempo en segundos y mediante una diferencia entre el tiempo de inicio y el tiempo final es posible obtener el tiempo total de ejecución en formato HH:MM:SS.

RE y OS

Se utilizó la librería **re** en el validador, usando el método **re.match(patrón, arreglo)** que permite establecer un patrón y compararlo con un arreglo, si este arreglo cumple con el formato del patrón devuelve True, caso contrario devuelve False, esto permite realizar la validación de los comandos ingresados, de modo que cumplan con la simbología y formato que corresponda, además del signo (+ o -).

Por otro lado, la librería **os** se encarga del manejo de archivos y comprobaciones de existencia de los mismos.

CMD

La librería **cmd** fue fundamental para la construcción de nuestro intérprete de comandos. Se utilizó la clase **Cmd** la cual fue heredada por nuestra consola permitiendo comprender y ejecutar comandos trabajando con argumentos, permitió el uso de ayudas al usuario mediante el comando **help,** y la creación de los menús utilizando los métodos **def do_algo**. Además, fuimos capaces de utilizar prompts personalizados tales como “>>”, “ROBOT>>” o “ARCHIVO>>” para facilitar la interacción con el usuario.

Detalles de aplicación

COMPILACION

Para el caso del servidor, la ejecución del programa se realiza de forma sencilla. Debemos ubicarnos en la carpeta contenedora del mismo con un terminal y ejecutar el comando "python3 main.py", esto dará como resultado:

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/SERVIDOR$ python3 main.py
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>>
```

En el caso del cliente, es necesario compilar el programa junto con todas sus librerías. Para realizar esta tarea es necesario nuevamente ubicarse con un terminal en la carpeta contenedora y ejecutar el comando "g++ *.cpp ./mis_librerias/*.cpp ./xmlrpc/*.cpp -o cliente" que se encargará de compilar todos loas archivos .cpp en la carpeta principal y en las carpetas mis_librerias y xmlrpc. Una vez compilado, genera un archivo binario ejecutable llamado cliente.

Para utilizarlo es necesario, con un terminal ubicado en la carpeta contenedora, ejecutar el comando "./cliente", obteniendo el siguiente resultado:

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/CLIENTE$ g++ *.cpp ./mis_librerias/*.
cpp ./xmlrpc/*.cpp -o cliente
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/CLIENTE$ ./cliente
Bienvenido al sistema de control de robot
Escriba help para ver los comandos disponibles
>>
```

La principal diferencia entre C++ y Python es el proceso de ejecución del código.

En C++, el código se compila en un archivo binario ejecutable antes de ser ejecutado. Durante el proceso de compilación, el código fuente es traducido a lenguaje de máquina, que es el lenguaje que el procesador de la computadora puede entender y ejecutar directamente. El archivo binario generado por el compilador es lo que se ejecuta directamente en la computadora, y no requiere la presencia del código fuente.

Por otro lado, en Python, el código fuente se interpreta línea por línea en tiempo de ejecución. El código fuente es procesado por el intérprete de Python, que lee cada línea de código y la convierte en lenguaje de máquina a medida que se ejecuta. Esto significa que el código fuente en sí mismo es necesario para ejecutar el programa.

La ventaja de la compilación es que el archivo binario ejecutable generado es más rápido y eficiente en términos de recursos de la computadora, ya que no se necesita un proceso de interpretación en tiempo de ejecución. Sin embargo, el proceso de compilación puede ser más complejo y requiere más tiempo y esfuerzo para configurar un entorno de compilación adecuado.

La ventaja de la interpretación es que el código fuente puede ser ejecutado de manera más flexible y conveniente, ya que no es necesario compilar el código antes de ejecutarlo. Esto permite una mayor interactividad y facilidad para el desarrollo y la depuración de código. Sin embargo, el proceso de interpretación en tiempo de ejecución puede ser más lento y requiere más recursos de la computadora en comparación con la ejecución de un archivo binario compilado.

USO

SERVIDOR

Modo local

Al ejecutar el servidor nos encontramos la siguiente pantalla:

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/SERVIDOR$ python3 main.py
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>>
```

Con el comando **help** podemos ver las opciones disponibles y utilizando **help comando** podemos obtener ayuda específica de un comando.

Las opciones disponibles son:

```
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>> help

Ayuda de comandos (escriba 'help <comando>' para obtener mas información)
========================================================================
archivo  help  robot  salir  servidor

>> help robot

Accede al menu de control del robot
```

Donde robot y archivo son submenús

Dentro del submenú robot encontramos las opciones:

```
>> robot
ROBOT>> help

Documented commands (type help <topic>):
========================================
apagar  efector  encender  estado  help  home  mover  rotar  salir  volver
```

Y dentro del submenú archivo nos encontramos con:

```
>> archivo
ARCHIVO>> help

Documented commands (type help <topic>):
========================================
borrar  detener  help  iniciar  reporte  reproducir  salir  volver
```

Procedemos a ejecutar una posible secuencia de uso real:

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/SERVIDOR$ python3 main.py
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>> robot
ROBOT>> encender
Robot encendido
ROBOT>> mover 10,20,30,40
El robot se movio a la posicion: X:10 Y:20 Z:30 con velocidad: 40
ROBOT>> efector on
Efector activado
ROBOT>> mover -10,-20,0,30
El robot se movio a la posicion: X:-10 Y:-20 Z:0 con velocidad: 30
ROBOT>> efector off
Efector desactivado
ROBOT>> rotar 1,35,a,15
El vinculo 1 roto 35° en sentido antihorario y con velocudad angular 15
ROBOT>> rotar 2,20,h,10
El vinculo 2 roto 20° en sentido horario y con velocudad angular 10
ROBOT>> rotar 3,75,h,5
El vinculo 3 roto 75° en sentido horario y con velocudad angular 5
ROBOT>> home
Robot en home
ROBOT>> apagar
Robot apagado
ROBOT>> volver
>> archivo
ARCHIVO>> reporte
<Reporte>
```

```
  <Inicio>2023-02-24 18:28:06.374756</Inicio>

  <Robot>

<Estado>Encendido</Estado>

  </Robot>

  <Tiempo>2023-02-24 18:28:11.528911</Tiempo>

  <Movimiento>

<X>10</X>

<Y>20</Y>

<Z>30</Z>

<Velocidad>40</Velocidad>

  </Movimiento>

  <Tiempo>2023-02-24 18:28:17.009505</Tiempo>

  <Efector>

<Estado>Activado</Estado>

  </Efector>

  <Tiempo>2023-02-24 18:28:20.207069</Tiempo>

  <Movimiento>
```

.
.
.

```
  <Tiempo>2023-02-24 18:28:28.752263</Tiempo>
  <Efector>
<Estado>Desactivado</Estado>
  </Efector>
  <Tiempo>2023-02-24 18:28:35.253920</Tiempo>
  <Rotacion>
<Vinculo>1</Vinculo>
<Angulo>35</Angulo>
<Sentido>a</Sentido>
<Velocidad>15</Velocidad>
  </Rotacion>
```

.
.
.

```
  </Rotacion>
  <Tiempo>2023-02-24 18:29:04.938711</Tiempo>
  <Home>
<X>0</X>
<Y>0</Y>
<Z>0</Z>
  </Home>
  <Tiempo>2023-02-24 18:29:16.827226</Tiempo>
  <Robot>
<Estado>Apagado</Estado>
  </Robot>
  <Tiempo>2023-02-24 18:29:25.289311</Tiempo>
  <Fin>2023-02-24 18:29:30.651129</Fin>
  <Tiempo total>0 horas 01 minutos 24 segundos</Tiempo total>
</Reporte>
ARCHIVO>> volver
>> salir
```

En esta secuencia se encendió el robot, se movió a la posición x=10 y=20 z=30 con velocidad 40, luego se activó el efector, se movió a la posición x=-10 y=-20 z=0 con velocidad 30 y se desactivo el efector, luego se realizó una rotación de los tres vínculos, se ejecutó el comando home para llevar el robot a su estado de reposo o inicial y finalmente se apagó el robot. Para completar la secuencia se pide el reporte para observar todas las tareas realizadas.

Esta puede ser una secuencia típica en que el robot se mueve a una posición, agarra una caja, vuelve a moverse y suelta la caja.

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/SERVIDOR$ python3 main.py
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>> archivo
ARCHIVO>> iniciar
Grabacion iniciada
ARCHIVO>> volver
>> robot
ROBOT>> encender
Robot encendido
ROBOT>> mover 40,30,20,10
El robot se movio a la posicion: X:40 Y:30 Z:20 con velocidad: 10
ROBOT>> efector on
Efector activado
ROBOT>> mover 10,20,30,40
El robot se movio a la posicion: X:10 Y:20 Z:30 con velocidad: 40
ROBOT>> efector off
Efector desactivado
ROBOT>> rotar 1,45,h,15
El vinculo 1 roto 45° en sentido h y con velocudad angular 15
ROBOT>> rotar 3,30,h,10
El vinculo 3 roto 30° en sentido h y con velocudad angular 10
ROBOT>> apagar
Robot apagado
ROBOT>> volver
>> archivo
ARCHIVO>> detener secuencia
Grabacion detenida
ARCHIVO>> reproducir secuencia
A 1
M 40 30 20 10
E 1
M 10 20 30 40
E 0
R 1 45 h 15
R 3 30 h 10
A 0
Repoduciendo secuencia
ARCHIVO>> salir
```

SERVIDOR > secuencia.txt

```
1  A 1
2  M 40 30 20 10
3  E 1
4  M 10 20 30 40
5  E 0
6  R 1 45 h 15
7  R 3 30 h 10
8  A 0
9
```

En esta secuencia se muestra el uso de la función de aprendizaje. Comenzamos ingresando al menú archivo, y desde allí iniciamos la grabación.

Luego en el menú de robot realizamos una secuencia de movimientos similar a la anterior, y finalmente regresamos al menú de archivo para detener la grabación indicando el nombre del archivo que se guardará de forma local.

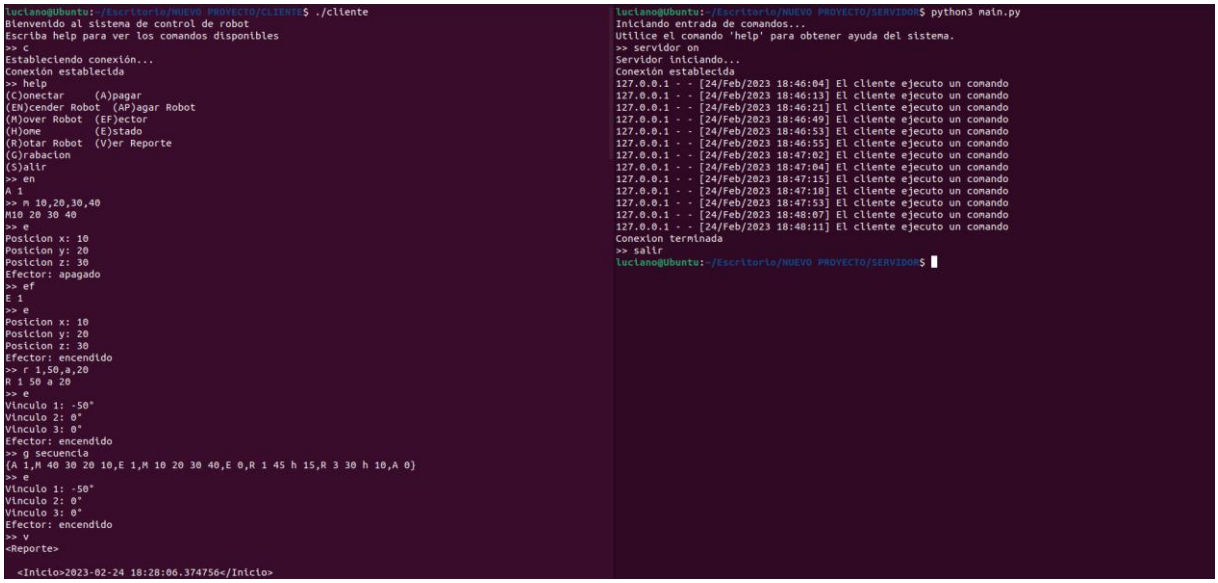
Luego esta rutina puede ser ejecutada desde el menú archivo con el comando reproducir seguido del nombre del archivo local, este es el modo de funcionamiento automático del robot.

Ayudas

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/SERVIDOR$ python3 main.py
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>> help

Ayuda de comandos (escriba 'help <comando>' para obtener mas información)
==========================================================================
archivo  help  robot  salir  servidor

>> help servidor

Inicia o apaga el servidor, utilice 'servidor [on/off]'

>> robot
ROBOT>> help

Documented commands (type help <topic>):
========================================
apagar  efector  encender  estado  help  home  mover  rotar  salir  volver

ROBOT>> help mover

Mueve el robot a la posicion indicada, utilice mover [X,Y,Z,velocidad]

ROBOT>> help rotar

Rota un vinculo del robot un angulo indicado, utilice rotar [vinculo,angulo,sentido,velocidad]

ROBOT>> volver
>> archivo
ARCHIVO>> help

Documented commands (type help <topic>):
========================================
borrar  detener  help  iniciar  reporte  reproducir  salir  volver

ARCHIVO>> help detener

Detiene la grabacion de una rutina y guarda el archivo, utilice detener [nombre_archivo]

ARCHIVO>> help reproducir

Reproduce un archivo, utilice reproducir [nombre_archivo]

ARCHIVO>> salir
```

Podemos observar el tipo de ayuda que recibimos para cada tipo de comando, siendo extremadamente útil para las funciones que necesitan argumentos extra para funcionar.

Modo en línea



.
.
.

En este caso se realizó la simulación de conexión entre el cliente y el servidor. Comenzamos ejecutando ambas aplicaciones para luego proceder a abrir las conexiones desde el lado servidor. Este queda a la espera de una conexión remota de parte de un cliente. El cliente procede a ejecutar el comando de conexión y se establece la misma entre ambos lados de la aplicación. En este momento el cliente toma el control y el servidor solo escucha las peticiones del cliente, pero no puede realizar ningún tipo de acción, esto es por motivos de seguridad (y algunas limitaciones de la librería xml-rpc). El cliente procede a realizar una secuencia típica de movimiento, finalizando con un reporte. Luego el cliente ejecuta una rutina automática de forma remota.

Errores

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/CLIENTE$ ./cliente
Bienvenido al sistema de control de robot
Escriba help para ver los comandos disponibles
>> c
Estableciendo conexión...
Error in XmlRpcClient::writeRequest: write error (error 111).

>> c
Estableciendo conexión...
Conexión establecida
>> m
[faultCode:1,faultString:<class 'IndexError'>:list index out of range]
>> m 1 2 3 4
[faultCode:1,faultString:<class 'IndexError'>:list index out of range]
>> m 80,20,30,40
[faultCode:1,faultString:<class 'Exception'>:Posicion x fuera de rango]
>> m 50,50,0,70
[faultCode:1,faultString:<class 'Exception'>:Velocidad fuera de rango]
```

CLIENTE

```
luciano@Ubuntu:~/Escritorio/NUEVO PROYECTO/SERVIDOR$ python3 main.py
Iniciando entrada de comandos...
Utilice el comando 'help' para obtener ayuda del sistema.
>> robot
ROBOT>> help

Documented commands (type help <topic>):
========================================
apagar  efector  encender  estado  help  home  mover  rotar  salir  volver

ROBOT>> efector
Argumento invalido
ROBOT>> mover 100,200,300,400
Debe ingresar cuatro valores numericos (dos digitos)
ROBOT>> rotar 1,2,3,4
El comando no es valido
ROBOT>> mover 100,20,30,40
Debe ingresar cuatro valores numericos (dos digitos)
ROBOT>> mover 60,50,50,50
El robot esta apagado
ROBOT>> encender
Robot encendido
ROBOT>> mover 60,50,50,50
Posicion x fuera de rango
ROBOT>> mover 50,60,50,50
Posicion y fuera de rango
ROBOT>> mover 50,50,60,50
Posicion z fuera de rango
ROBOT>> mover 50,50,50,90
Velocidad fuera de rango
ROBOT>> volver
>> archivo
ARCHIVO>> detener
Grabacion detenida
ARCHIVO>> reproducir
No existe el archivo
```

SERVIDOR

En estas imágenes se observan distintos problemas o errores que pueden surgir en el uso de la aplicación, y como el programa lidió con estos. Esto es posible gracias al uso de los condicionales try: except: y raise Exception, los cuales permiten hacer un tratamiento de errores evitando la detención del programa.

3) COMENTARIOS Y CONCLUSIONES

Fue muy interesante la realización de este proyecto, ya que el mismo representó un gran desafío para nuestras capacidades en el mundo de la programación, y además nos incentivó a investigar y querer conocer a fondo el funcionamiento de los lenguajes utilizados, así como los conceptos aplicados y el funcionamiento de las librerías.

Por otro lado, tuvimos que utilizar el sistema operativo Linux, el cuál fue una experiencia nueva, pero positiva. El uso de este sistema fue necesario debido a incompatibilidades entre la librería fundamental de este proyecto que es Xml-Rpc y Windows.

Además de esta dificultad, nos encontramos con la tarea de esquematizar el proyecto según el patrón de arquitectura de software MVC (Modelo-Vista-Controlador), ideal para el desarrollo de aplicaciones. Este patrón nos permitió reutilizar código y separar de conceptos para aumentar la legibilidad y facilitar su posterior mantenimiento.

Creemos que la realización de este proyecto fue la mejor forma de poner en práctica nuestros conocimientos y nuestro ingenio para resolver los problemas que se presentaron, así como aprender nuevos conceptos que nos serán útiles para proyectos futuros.

Destacamos el trabajo en equipo y la coordinación, de otro modo no habría sido posible lograr este proyecto con la misma calidad y presentación.

Por último, creemos que es posible extender la funcionalidad de este proyecto aplicando algunas consignas que eran opcionales para el mismo. Algunas de ellas serian un sistema de cinemática inversa para el control de los vínculos al realizar un movimiento lineal, o el uso de una interfaz gráfica para facilitar la interacción con el usuario y poder mostrar gráficos más avanzados.

4) RESUMEN

El proyecto desarrollado consistió en crear una aplicación cliente-servidor utilizando la librería xml-rpc. Se crearon las dos partes en lenguajes de programación diferentes (cliente C++, servidor Python) respetando siempre el paradigma de objetos o POO. Se diseñó el sistema en capas, para permitir mayor facilidad de comprensión y escalado del mismo, y permitir el uso de interfaces de comandos o interfaces gráficas.

Se dividieron las partes del programa de forma conveniente en carpetas contenedoras para mantener el orden y facilitar el uso.

Se realizaron múltiples iteraciones y test de la aplicación en su conjunto para asegurar su correcto funcionamiento e intentar controlar la mayor cantidad de errores posibles.

Finalmente, obtuvimos una aplicación del tipo cliente-servidor completamente funcional con capacidad para controlar un robot de forma local o remota, con múltiples funcionalidades y siempre haciendo uso de los conceptos clave de la Programación Orientada a Objetos aprendidos en esta asignatura, el cual era el principal objetivo de este proyecto.

5) BIBLIOGRAFIA

FANUC Palletizing Robot

https://www.youtube.com/watch?v=ppbsrAEEA8Q&feature=youtu.be

MVC example

https://stackoverflow.com/questions/38042632/mvc-the-simplest-example

Análisis Orientado a Objetos

https://pankrdo.wordpress.com/analisis-orientado-a-objetos-aoo

Programación orientada a objetos - Wikipedia

https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

Bibliografía provista por la cátedra