

TRABAJO PRÁCTICO INTEGRADOR DE POO

Emanuel Agüero, Carlos Bustillo, Agustín Lezcano, Anatole Trouffaut y Romain Houé

Programación Orientada a Objetos, Facultad de Ingeniería, Universidad Nacional de Cuyo

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, frezado, soldadura a gas, soldadura al arco, y pintura con spray.

1.1. Ejemplo 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

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. Stringstream

Es una clase usada para operar sobre cadenas. Los objetos de esta clase usan un búfer de cadena que contiene una secuencia de caracteres. Se puede acceder a esta secuencia de caracteres directamente como un objeto de cadenas, usando `str`. Los caracteres se pueden insertar y / o extraer de la secuencia utilizando cualquier operación permitida en las secuencias de entrada y salida. La librería que se debe utilizar es `<sstream>` para usarlo en C++. En Python no se hace uso de este concepto.

2.3. 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()`.

2.4. Cmd

Se usó en el servidor. Esta clase provee un *framework* simple para escribir interpretes 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 mediante un cliente, el cual envía la consigna al servidor y este desarrolla el control. Posteriormente se hace uso de la gestión de archivos, que consiste en el envío de la consigna codificada (la cual se denomina trama) que se puede interpretar de la siguiente manera: El cliente debe iniciar el mensaje con la letra E (de empezar), que indica el comienzo de la operación, y finalizarlo con la letra P (parar). Luego el cliente debe especificar, considerando signos, el ángulo de rotación de cada articulación. La base u hombro está denominado con la letra A, la segunda articulación o codo con la letra B y la tercera articulación con la letra C. El mensaje debe contener ángulos de rotación menores a 360. Así por ejemplo, si se requiere la rotación de la articulación A un ángulo de 50, B 170 en sentido antihorario y C 40, el mensaje se verá así: EA+50B-170C+40P Análogamente se requiere especificar el movimiento de la articulación, especificando la velocidad (V), rotación del efector final pinza (P) y la operación de abrir o cerrar las pinzas (A). El control del robot se puede realizar de dos formas distintas: el modo automático y el modo manual. Para el modo

manual se requiere ingresar las consignas una por una, y las mismas serán guardadas en un archivo de texto. En cambio en el modo automático se ingresa un archivo de texto con todas las instrucciones. Por parte del servidor se realiza una verificación de la orden y luego se procede al envío del mensaje mediante el protocolo XML-RPC. Por parte del servidor, se puede realizar la conexión/desconexión del robot, de manera tal que se verifica si hay conexión con el servidor, y luego si se desea conectar el robot. Del lado del servidor se realizan las mismas acciones, tanto la decodificación del mensaje como el manejo de archivos siguiente, como guardar la instrucción, guardar el historial del espacio de trabajo y el tiempo usado para cada tarea. Se partió de los siguientes modelos de diagrama de clases (Figura 2) y de casos de uso (Figura 3), los cuales se modificaron debido a consideraciones de implementación.

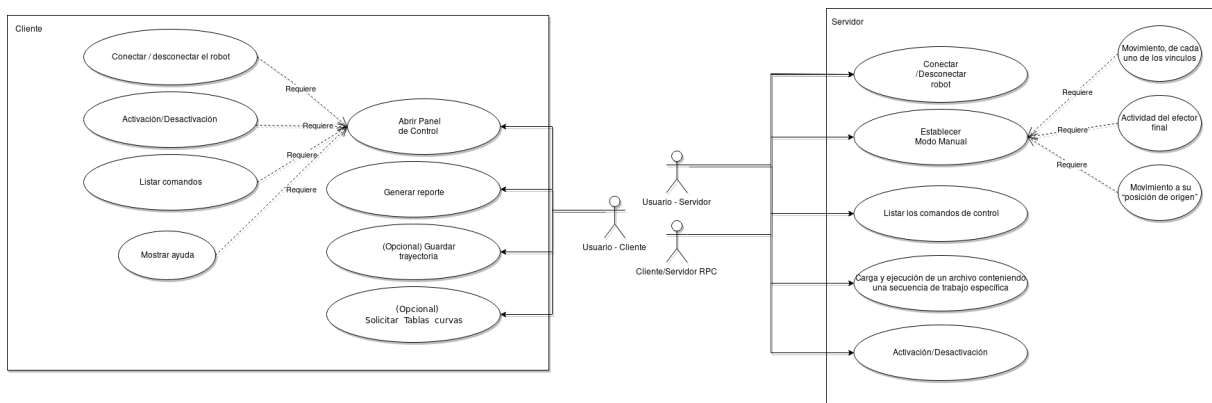


Figura 2: Diagrama de clases

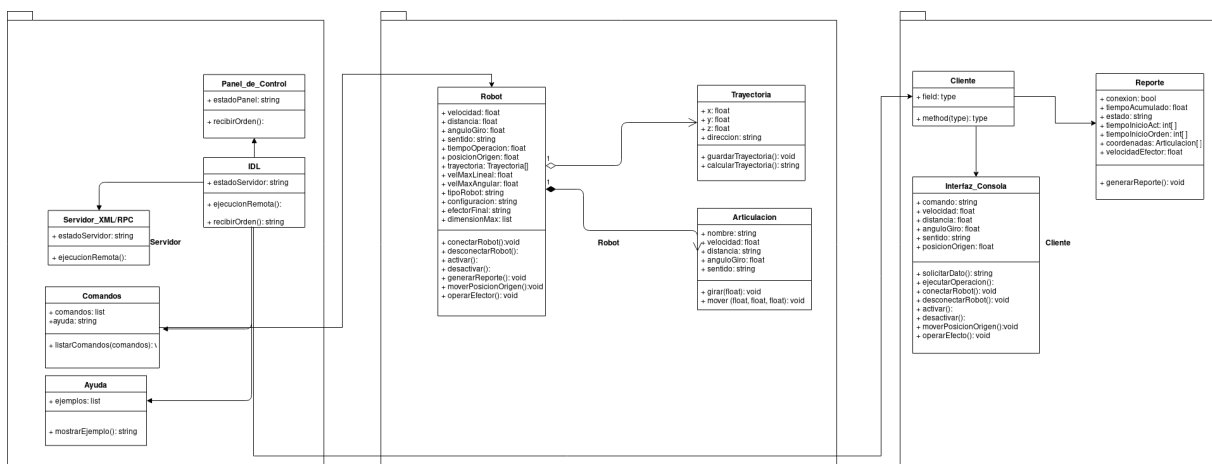


Figura 3: Diagrama de casos de uso

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 hicieron uso del paquete 'xmlrpc' de Python, que incluye módulos para cliente y para servidor. El modulo '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 0.7, 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 8080.

5. EJECUCIÓN DE LA APLICACIÓN

Del lado del cliente, el menú se presenta de la siguiente manera:

```
[agustin@agustin-pc C++]$ ./a.out
=====
ORDENES
1 - Control articulaciones
2 - Control efector final
3 - Generar reporte
0 - Salir
=====
Ingrese el número de orden a realizar: 1

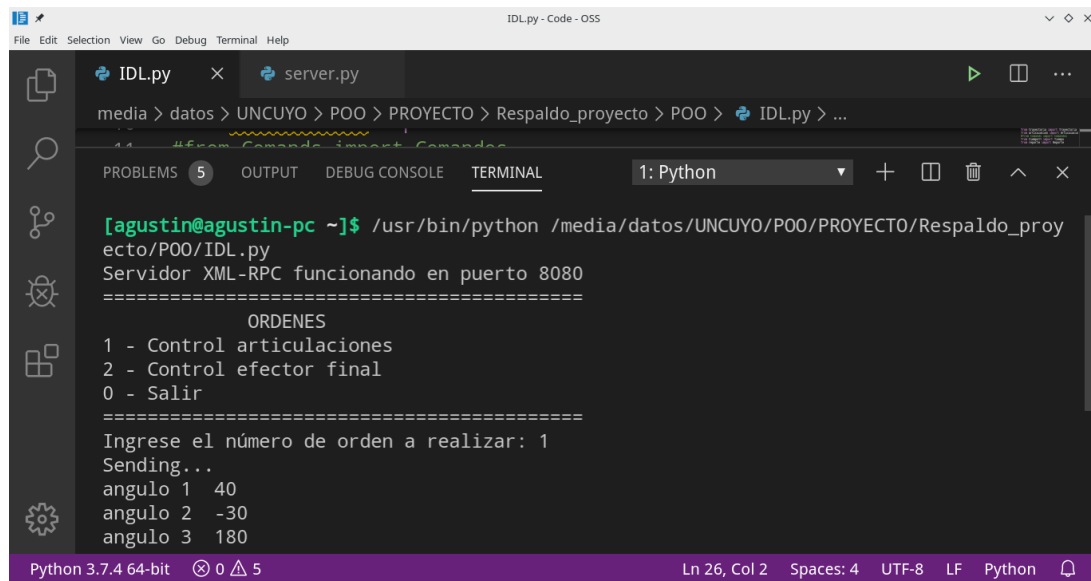
Ingrese modo:
A - Modo Automático
M - Modo Manual
M
=====
MOVIMIENTO ARTICULACIONES
=====
E - Empezar
P - Parar
A - Articulacion A
B - Articulacion B
C - Articulacion C
T para terminar

Ingrese el número de la acción:
EA+40B-30C+180P
EA+40B-30C+180P
Orden válida

=====
ORDENES
1 - Control articulaciones
2 - Control efector final
3 - Generar reporte
0 - Salir
=====
Ingrese el número de orden a realizar: █
```

Figura 4: Menú de movimiento del lado del cliente

Del lado del servidor, se recibe la consigna y se transmite el mensaje:



```
media > datos > UNCUYO > POO > PROYECTO > Respaldo_proyecto > POO > IDL.py > ...
11 #from Commands import Commands

[agustin@agustin-pc ~]$ /usr/bin/python /media/datos/UNCUYO/POO/PROYECTO/Respaldo_proyecto/POO/IDL.py
Servidor XML-RPC funcionando en puerto 8080
=====
ORDENES
1 - Control articulaciones
2 - Control efector final
0 - Salir
=====
Ingrese el número de orden a realizar: 1
Sending...
angulo 1 40
angulo 2 -30
angulo 3 180
```

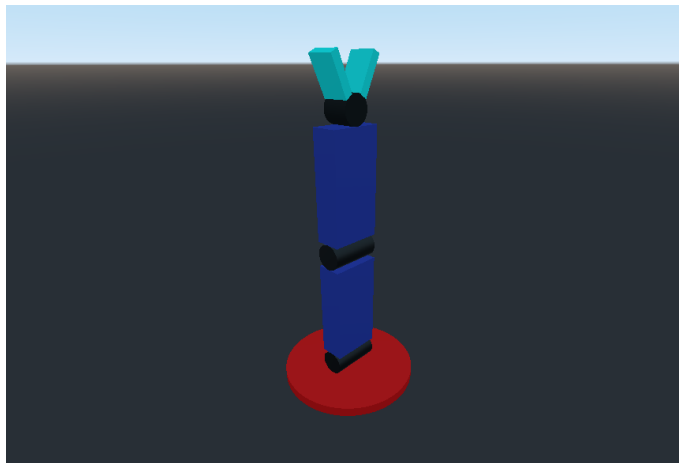
Figura 5: Servidor

6. SIMULACIÓN

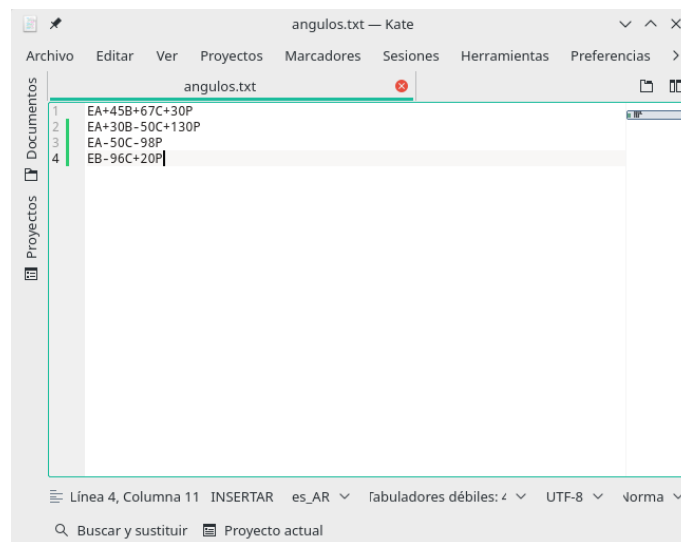
Adicionalmente, se realizó una simulación del movimiento del brazo robótico con el motor de videojuegos *Godot*, con el cual se pueden observar perspectivas del brazo rotando y los movimientos del efector (Ver Figura 6a).

El control los movimientos de las articulaciones del robot se realizó instanciando cada articulación a su articulación predecesora, y ellas a la base, mediante señales. Para realizar la simulación se hizo uso del lenguaje GDScript, propio del motor de búsqueda.

El programa recibe un archivo *.txt* (ver Figura 6b) con las instrucciones a seguir para el control del brazo y las ejecuta .



(a) Simulación en Godot



(b) Bloque de órdenes para el comando automático

7. CONCLUSIÓN

Uno de los principales problemas que se tuvieron durante la realización del proyecto fue el de la representación de los movimientos de las articulaciones, el cual se resolvió mediante el uso de arreglos trigonométricos. Para el diseño en 3d hubiese necesitado contar conocimientos de cinemática de robots que hasta la fecha no han sido adquiridos. Otro de los inconvenientes fue el uso del protocolo XMLRPC, ya que no se encontraban bibliotecas de XMLRPC para C++ que se adecuen a nuestras necesidades, eligiendo finalmente la librería xmlrpc++. Un tercer y último inconveniente se presentó al incluir los módulos escritos en Python al servidor y entre ellos.

Una vez superados esos inconvenientes, se logró una buena modularidad y un buen funcionamiento de la aplicación. En cuanto al proyecto, el grupo encontró un buen desafío y sirvió para afianzar los conocimientos adquiridos durante el cursado de la materia.

Como posibles mejoras a futuro se puede pensar en una optimización de las tareas realizadas en el servidor, lograr una mayor modularización en el mismo y realizar el control y visualizador 3D.

8. BIBLIOGRAFÍA

<https://docs.python.org/3/>
<http://www.cplusplus.com/>
<http://xmlrpc-c.sourceforge.net/>
<https://docs.python.org/3/library/xmlrpc.html>