Programación Orientada a Objetos
Ingeniería en Mecatrónica

Proyecto final

Simulador robot

Profesor: Esp. Ing. César Omar Aranda
Integrantes: Avanzini, Gino Hernán (11355)
Cabrino, Emiliano Martín (11220)

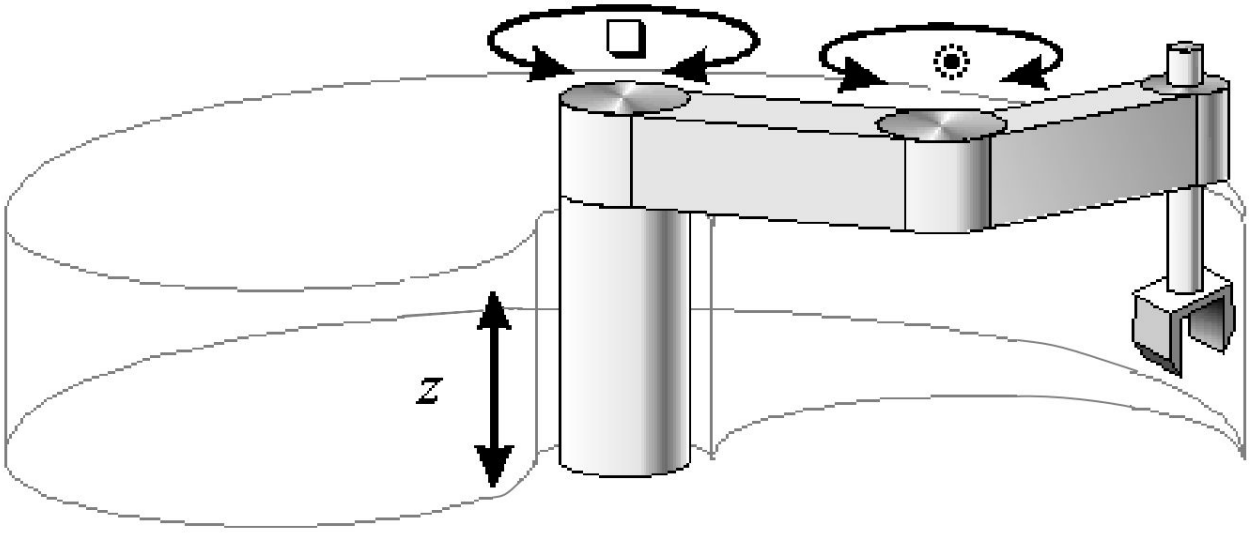
Universidad Nacional de Cuyo
Facultad de Ingeniería
Mendoza - Argentina
2018

INTRODUCCIÓN:

En el presente informe se desarrollará el trabajo final integrador de la cátedra Programación Orientada a Objetos de la carrera de Ingeniería en Mecatrónica de la Universidad Nacional de Cuyo. Se mostrarán los aspectos teóricos y de diseño involucrados en la realización del Proyecto Final Integrador a modo de incorporar aspectos avanzados del lenguaje C++, el trabajo consiste en la programación de una interfaz gráfica para la implementación virtual de un simulador de un robot de 3 grados de libertad tipo PRR Skara, es decir de configuración cilíndrica, en el cual la primera articulación es de tipo prismática, y las otras dos de tipo rotacional. El término de configuración cilíndrica se debe al hecho de que son justamente las coordenadas cilíndricas, las que mejor definen la posición del efector terminal de este tipo de robots, con respecto a un sistema de referencia. Se utilizan en trabajos de carga, desplazamiento y descarga de materiales, en aplicaciones de sellado, ensamblaje, y manejo de máquinas-herramientas.

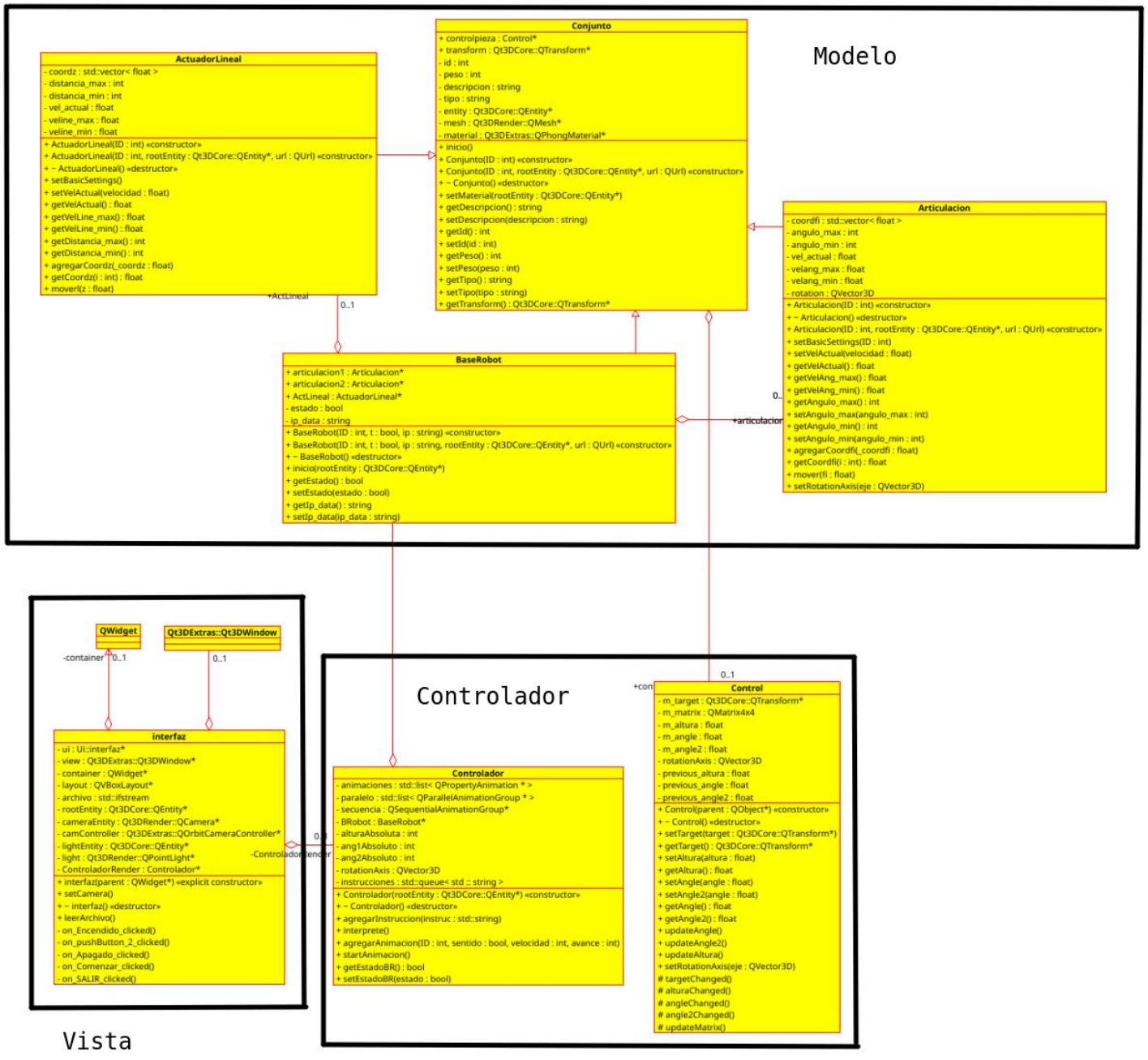
Para poder realizar esta interfaz se generan las distintas clases del robot para poder controlarlo de la manera más simple pero a la vez que sea los más aplicable a la realidad posible. La interfaz tiene elementos de control del robot así como entrada de órdenes en código G o posibilidad de leer las tareas desde un archivo .txt.

Para realizar esta tarea se utilizó el framework multiplataforma orientado a objetos Qt y el lenguaje de programación C++, junto a herramientas de soporte para control de versiones y avance del proyecto como Git con GitHub o el editor de texto como Atom con su función Teletype para el desarrollo del código de forma colaborativa remota. Es valioso remarcar que el proyecto se desarrolló íntegramente con herramientas libres.



DISEÑO DEL SISTEMA:

Para el desarrollo del programa, primero se realizó una etapa de diseño, en la cual se discutieron grupalmente las distintas ideas para abordar el problema, obteniendo un esquema temporal de las clases, métodos y atributos a usar.
A medida que se fueron programando las distintas clases, surgieron numerosas modificaciones, resultando finalmente el siguiente diagrama de clases UML:



Se implementó una arquitectura de diseño del tipo Modelo-Vista-Controlador, con tres grupos de clases bien diferenciadas entre sí implementando dicha arquitectura. Por el lado del modelo,

contamos con las clases que heredan de Conjunto, la clase base de toda esta capa, estas son: BaseRobot, Articulacion, ActuadorLineal y EfectorFinal. Con respecto a la Vista, tenemos una clase única llamada convenientemente Interfaz la cual es lanzada desde el main del programa y contiene todos los elementos necesarios para contener la escena en 3D y proveerle al usuario algunas herramientas de control. Por último, en la capa de control contamos con las clases Controlador y Control. La primera se encarga puramente del control de la interfaz y de la generación de las animaciones de acuerdo a lo que se lee del archivo. La clase control provee los métodos para realizar las transformaciones a las QMesh de las piezas para de esa forma generar las animaciones.

El diagrama consiste en varias clases. La mayoría de ellas fueron reutilizadas a partir de los trabajos prácticos 2 A y B de la cátedra haciendo modificaciones. Además se implementaron otras clases con haciendo uso de herramientas y clases específicas del framework Qt5 para poder desarrollar las consignas del trabajo integrador, en especial el renderizado y el control de los elementos 3D del modelo.

Las clases son, en orden alfabético:

Articulación

Clase que representa los brazos que tienen movimientos angulares en el robot. Si bien el nombre puede parecer poco intuitivo, se decidió conservarlo para reforzar la idea de que esta clase posee la QMesh y la QTransform del brazo que rota respecto a la articulación. De esta forma es más sencillo realizar las transformaciones para generar las animaciones.

BaseRobot

Como contiene la electrónica del robot, tiene conocimiento total de las posiciones espaciales de todos los elementos; los cuales están agregados a esta clase. Cuando se construye esta clase se instancian objetos de las clases agregadas y se generan todos los objetos para el correcto renderizado de los elementos del robot. Debido a que el diseño utilizado no fue muy bueno se tuvo que recurrir a colocar las clases agregadas como públicas. Si bien se podrían haber escrito métodos públicos para acceder a los atributos de las articulaciones y el actuador lineal, se decidió colocar estas clases agregadas como públicas por la cantidad de métodos que habría que haber escrito. Esta es una de las tareas que se pueden implementar a futuro para mejorar el diseño y profundizar el uso del paradigma Orientado a Objetos.
Contiene el estado general del robot. Si no se enciende explícitamente el robot mediante el uso del botón en la interfaz principal, no se ejecutará el programa precargado. Contiene las clases de las articulaciones y el actuador lineal y las inicializa. De esta forma, crea y renderiza dichos objetos aprovechando que ellos heredan de conjunto.

Actuador Lineal

Clase que representa la columna sobre la cual se realiza el movimiento traslacional de la altura en el robot. Al igual que la clase Articulación, contiene los mismos atributos de QMesh y la QTransform para poder generar animaciones.

Conjunto

Es la clase base de todos los objetos a ser renderizados y por eso agrega varias clases necesarias para eso. Entre dichas clases se encuentra la QMesh del modelo en 3D, la QTransform que permite realizar rotaciones y traslaciones al modelo y la QEntity del objeto. Esta es contenedora del material, la mesh del modelo y la QTransform. Además se agrega y se instancia un objeto de la clase Control.

Control

Esta clase es agregada a conjunto y, como consecuencia, a cada uno de los objetos con modelos en 3D. Esta clase se encarga de realizar las transformaciones necesarias a las QMesh del robot para generar las animaciones que se ven al cargar el programa. Contiene métodos específicos de acuerdo al tipo de transformación que sufre el modelo; esto es, cambiar altura o rotar cierto ángulo. Para realizar dichas transformaciones se utilizaron Q_PROPERTYs de acuerdo a la característica que se quiere cambiar.

Controlador

La clase controlador se encarga propiamente del control de los movimientos en base al archivo con el GCode. Una vez presionado el pulsador que se presenta en la pantalla, esta clase se encarga de cargar el archivo e interpretar las líneas de GCode. Luego, por cada instrucción se generan las animaciones correspondientes. La estructura de las animaciones se realizó con algunas de las clases provistas por el framework, notablemente: QPropertyAnimation, QParallelAnimationGroup y QSequentialAnimationGroup.
Además Controlador tiene una relación HAS-A con la clase BaseRobot. Cuando se instancia un objeto Controlador se construye además la BaseRobot, la cual a su vez creará los modelos con las piezas del robot. Es preciso recordar que tanto BaseRobot como las clases de las piezas heredan de Conjunto.
La estrategia para crear las animaciones fue crear un control centralizado, del cual se encarga, justamente, la clase Controlador. De acuerdo a qué elemento del robot estamos manipulando agregaremos 1, 2 o 3 animaciones en paralelo. Estas animaciones están almacenadas en una std::list conteniendo las QPropertyAnimation. Luego, todas las animaciones en paralelo (debido a que las piezas se encuentran acopladas solidariamente) se almacenan en otra std::list llamada paralelo. Cada uno de los elementos de paralelo contiene un grupo de QPropertyAnimation que se ejecutarán en paralelo, dando la sensación de que las piezas separadas conforman una sola pieza rígida solidaria. Luego, cada uno de estos grupos de animaciones en paralelo se agregan al último

contenedor de animaciones. Este es QSequentialAnimationGroup. A este objeto se le van agregando los grupos de animaciones en paralelo que van a sucederse secuencialmente. Esto es justamente cada una de las instrucciones que se encuentran en el GCode.

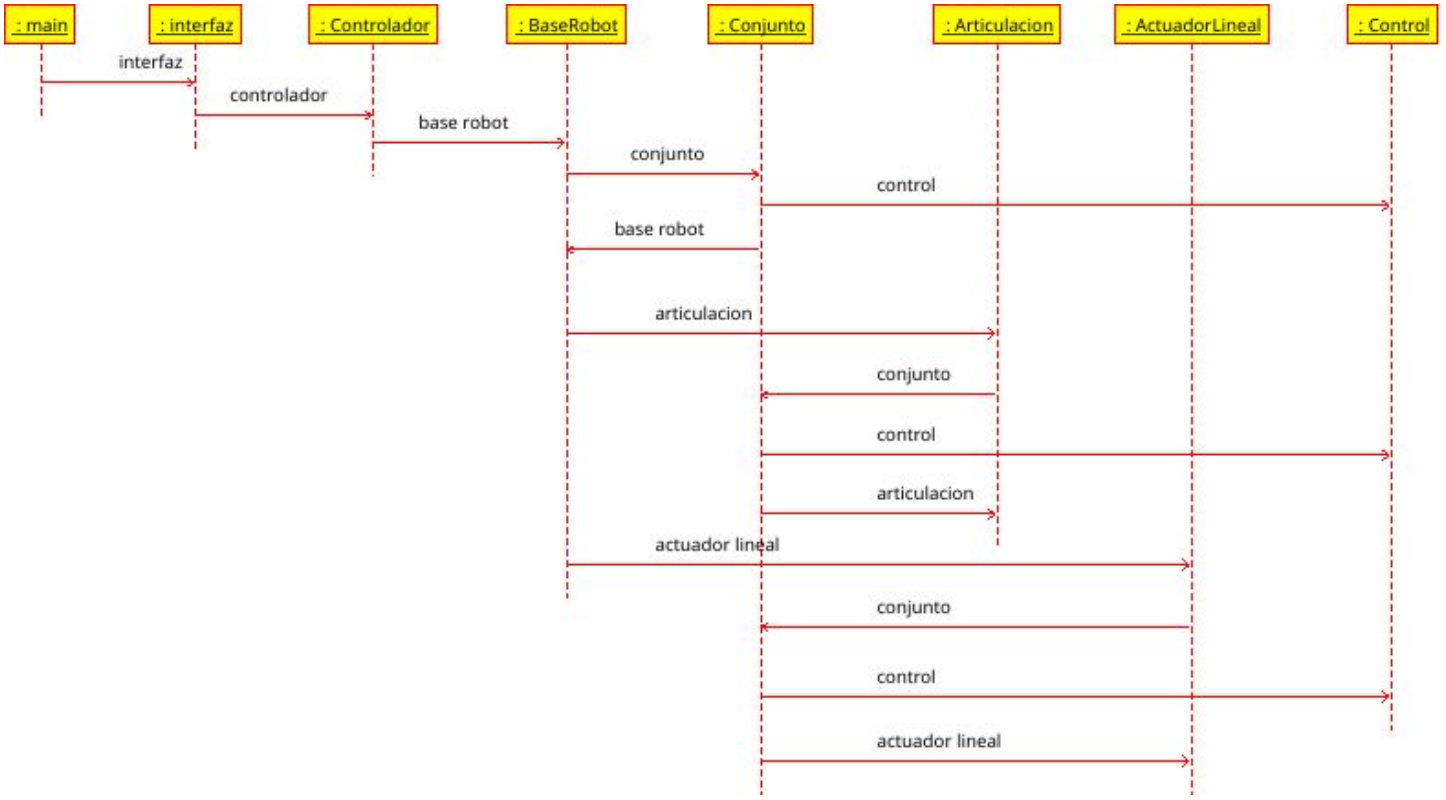
Efector Final

La clase *Efector Final* es una clase que hereda de *Conjunto* y está asociada al efector final del robot. Las posibles tareas que puede ejecutar son: *pintar*, *sostener*, *soltar* y *rotar*. Cada una de esas tareas cuenta con un ID separado, los cuales son los *int* 1, 2, 3 y 4 respectivamente. Debido a que no el efector no realiza transformaciones a ninguna QMesh, se decidió simular la acción del mismo mediante la implementación de pausas en las animaciones.
En el GCode se ingresa la línea correspondiente y seguido a ella, la cantidad de ciclos a ejecutar dicha acción. El tiempo de pausa que toma cada realizar cada acción ha sido simplemente agregado al código, en el constructor del EfectorFinal; contando con la posibilidad en el futuro de cambiarlos. La pausa total que se notará en la animación es nada más que el tiempo que toma cada acción multiplicado por la cantidad de ciclos que se indicón en el GCode.

Interfaz

Esta clase se instancia en el main de nuestro programa. Se encarga de proveer la interfaz gráfica al usuario, la cual cuenta con la Qt3DWindow en la cual se reproduce la animación en 3D, un QWidget que *contenga* a dicha Qt3DWindow y numerosas botones y Layouts. Además agrega e instancia las clases necesarias para visualizar correctamente la animación. Estas son las clases que generan la cámara, el controlador de la cámara y la luz de la escena.
Además, contiene una clase agregada muy importante, Controlador. Cuando se construye la clase Interfaz se construye además un objeto de la clase Controlador. Esta clase se encarga de realizar las transformaciones necesarias a las piezas del modelo para realizar correctamente la secuencia de acciones descritas por el archivo con el GCode.

Diagrama de secuencia



Identificadores internos para comunicación entre elementos y clases

Nº	Identificador y tipo de pieza
1	Id: 10 Tipo: primer articulación tipo lineal
2	Id: 11 Tipo: segunda articulación tipo rotacional
3	Id: 12 Tipo: tercera articulación tipo rotacional
4	Id:50 Tipo: efector final

Instrucciones G-Code

Debido a la dificultad que se nos presentó para usar el G-Code estándar de Rep-Rap, diseñamos nuestro propio G-Code a modo simple, en el G-Code estándar las órdenes están referidas a la posición global del efector final, en cambio aquí lo que se hace son todos movimientos relativos, uno a la vez de cada articulación-actuador lineal, la manera de cargar las instrucciones es mediante un archivo de texto que el programa cuando se inicia lo carga.

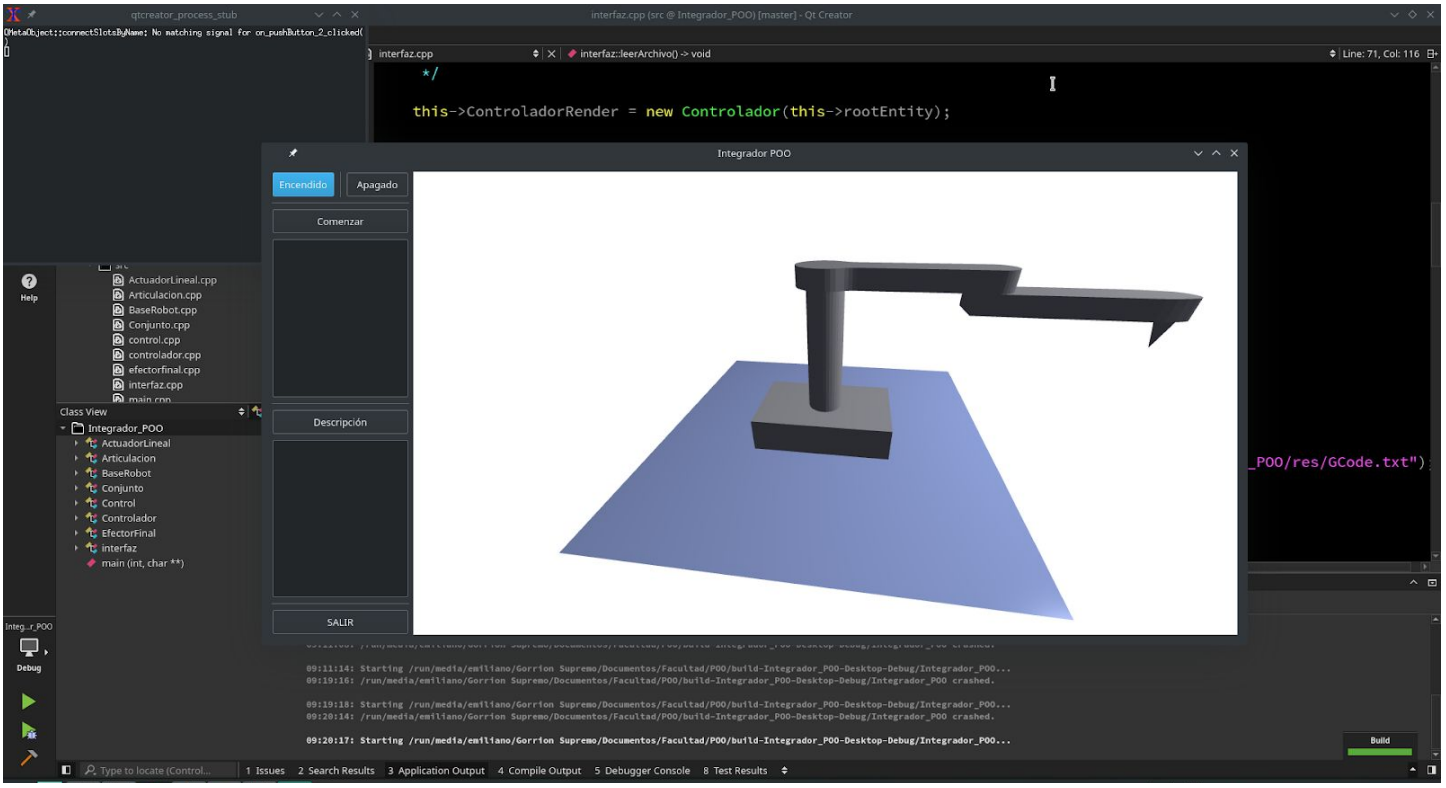
Instrucción	Descripción
GXY **Vel** **Pos**	Siendo **X** 1 correspondiente al actuador lineal, 2 a la primera articulación y 3 a la articulación final e **Y** al sentido de movimiento, siendo 1 arriba/giro horario y 2 abajo/giro antihorario. Después de declarar dicha instrucción en las 2 líneas de abajo se debe declarar primero la velocidad de movimiento deseada y en la 2ª línea la posición relativa a la cual se quiere que llegue dicha articulación que está en centímetros o en grados, dependiendo la articulación que se maneje.
HT **Ciclos**	Siendo **T** igual a 1 para pintar, 2 para rotar, 3 sostener y 4 soltar. Después de declarar dicha instrucción en la línea de abajo se debe declarar la cantidad de ciclos en la cual se mantiene la tarea a realizar.
C00	Instrucción para terminar el ingreso de todas las instrucciones y lanzar la animación.

APLICACIÓN

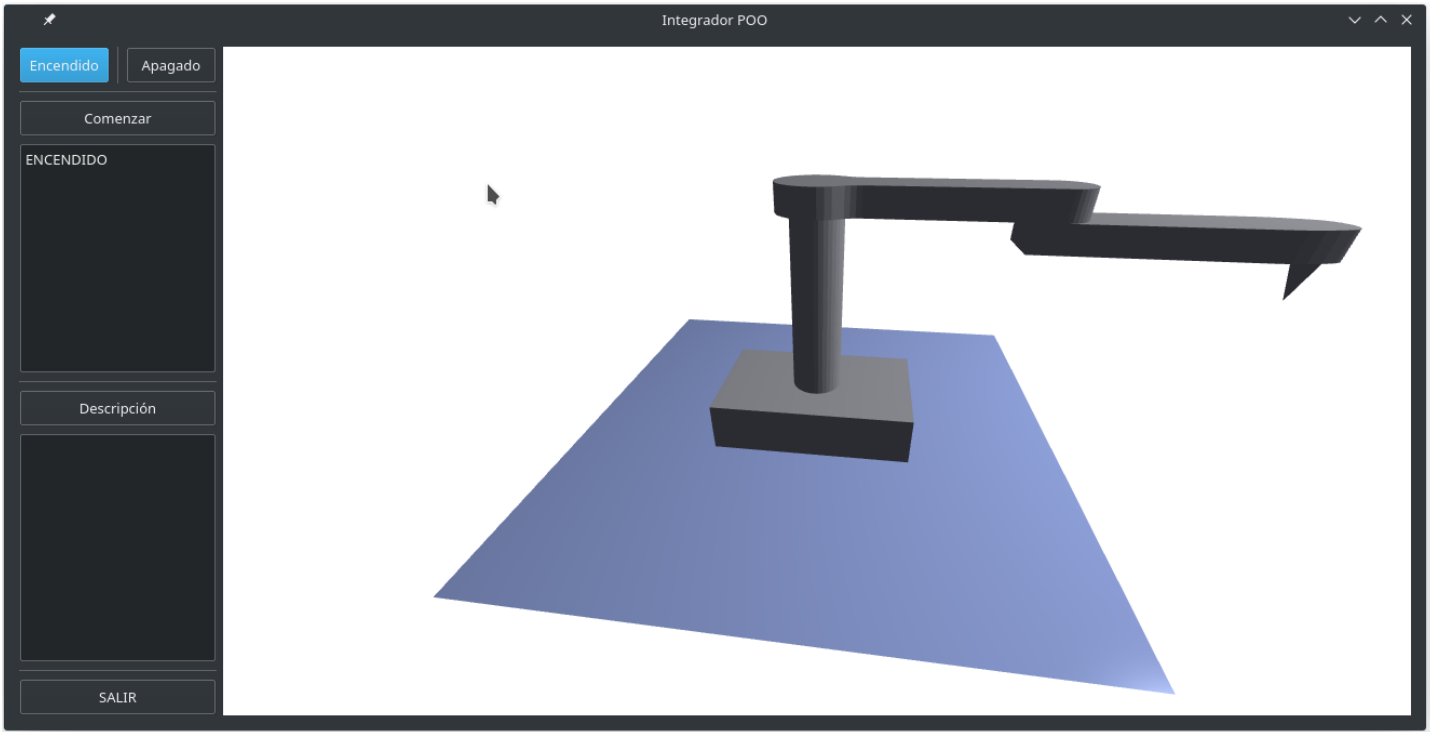
Para ejecutar la aplicación se deben tener en cuenta ciertas cosas:
1º - Se debe cambiar el path (/home/gino/Dropbox/FING/Programación_Orientada_a_Objetos/TP-Integrador_POO/res/GCode.txt) del archivo gcode.txt que aparece en el archivo interfaz.cpp en la linea 71 por la ubicación global donde se encuentre el proyecto y el archivo /res/GCode.txt

2º Para ejecutar la aplicacion abrir el archivo “Integrador_POO.pro” mediante QT Creator 5.11, configurar el proyecto en “Projects” de QT.

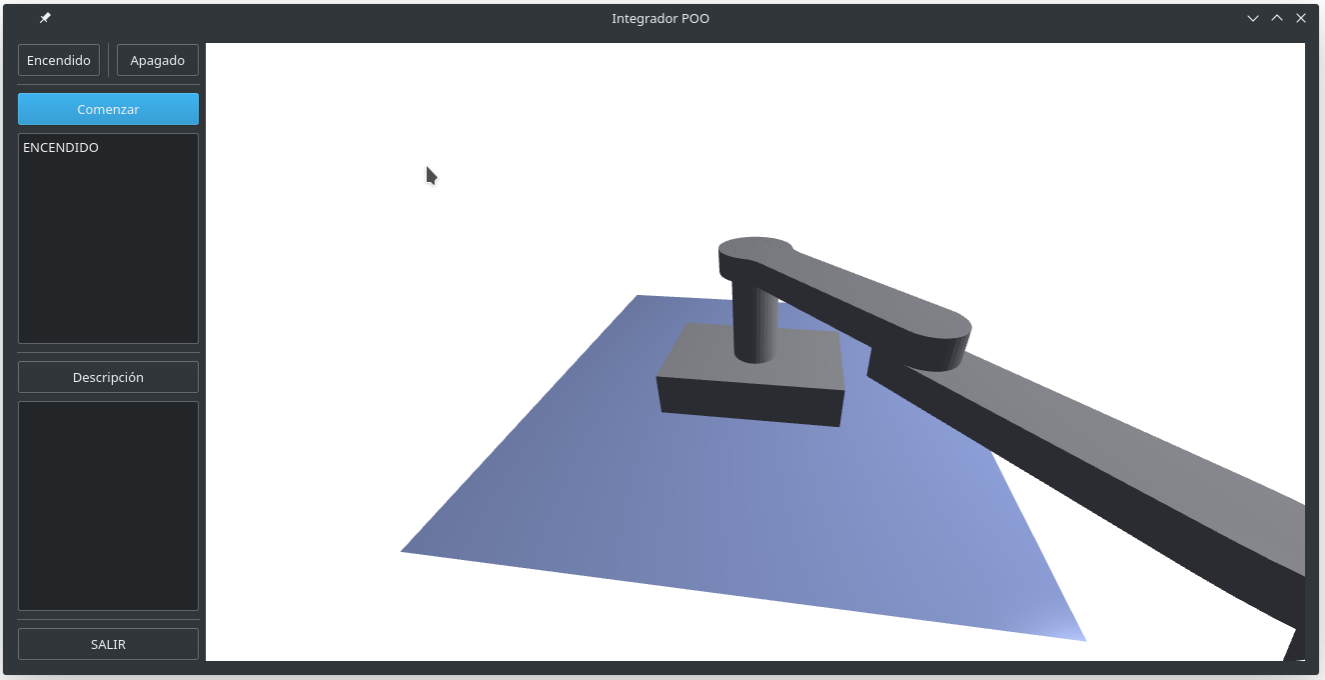
Una vez hecho esto se puede lanzar el programa y se verá algo como esto:



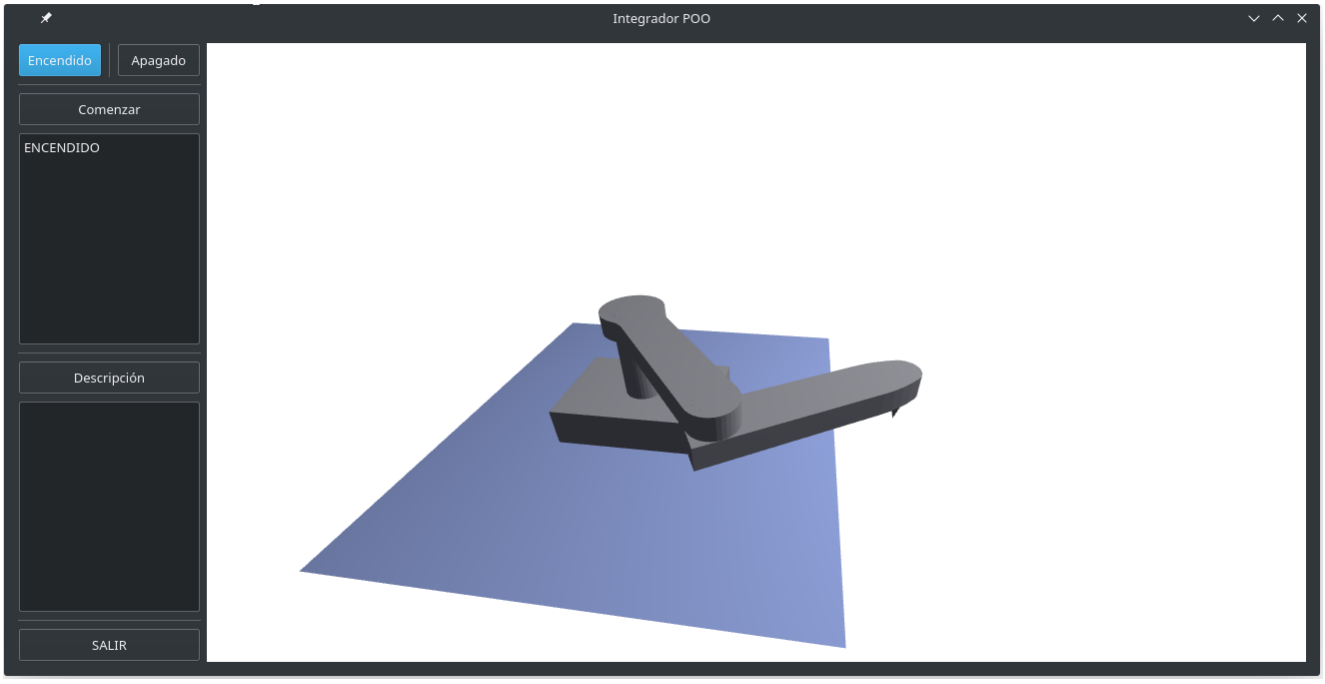
Se pueden ver 2 ventanas, la aplicación principal y otra del proceso.
Si nos centramos en la principal, vemos que hay diferentes botones.



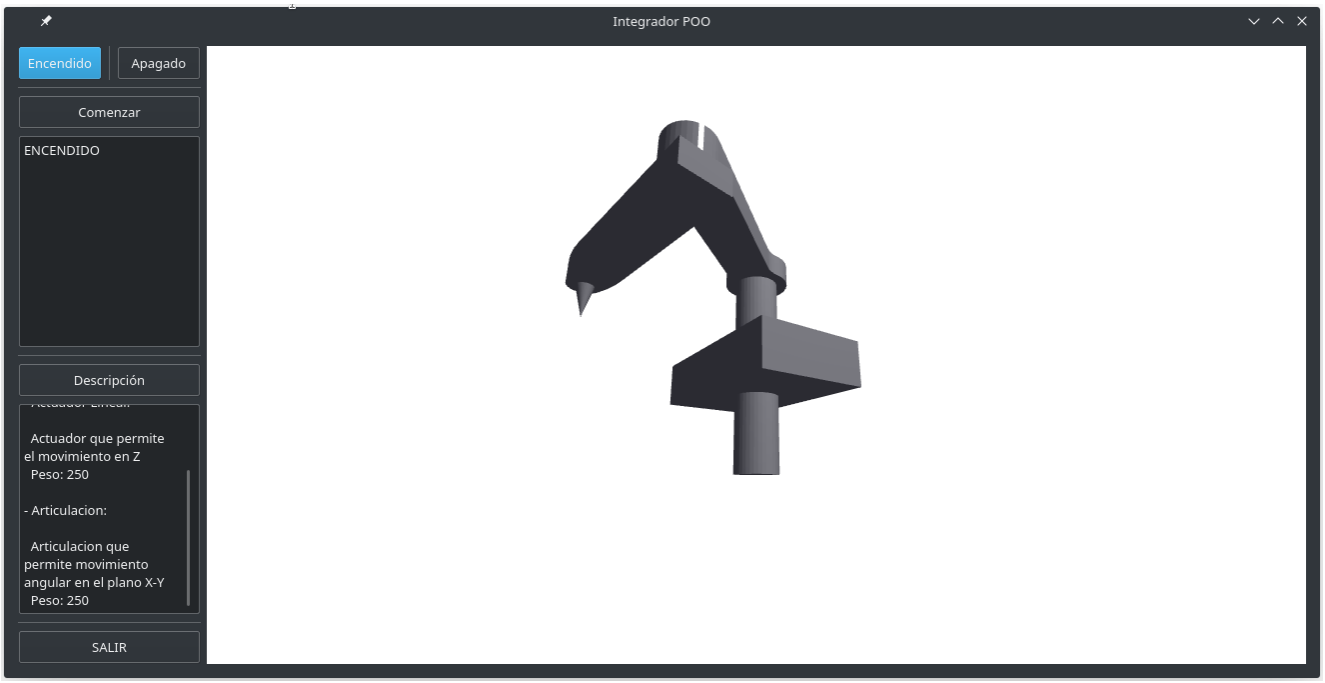
Para poder Comenzar primero hay que apretar el boton de Encendido y luego el de Comenzar.



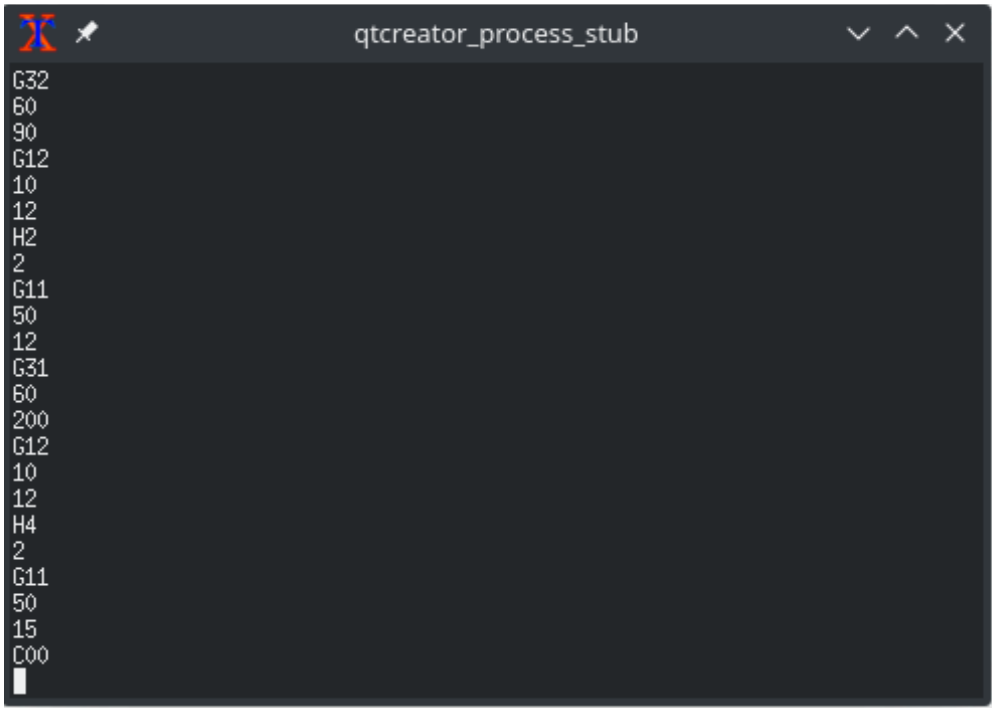
Vemos que empieza la simulación con el archivo G-Code previamente cargado y con el mouse podemos mover y rotar la cámara con botón derecho e izquierdo, y con la ruedita del mouse podemos aumentar o alejar el zoom.



Si apretamos el boton de Descripción se despliega dicha descripción en la ventana de abajo



De manera secundaria se lanza una ventana de procesos donde muestra el archivo g-code que se ejecuta.



LIMITACIONES Y MEJORAS A FUTURO

Debido a que en la consigna del trabajo integrador pedía que se vieran 3 vistas del robot hechas con una esquematización sencilla o una sola vista en 3d, decidimos por complejidad y por posibilidad de aprender una herramienta más enriquecedora aplicar una sola vista en 3d, haciendo esto que no se llegará al 100% de lo esperado, a continuación se detallan algunas limitaciones, las cuales no se llegaron a resolver:

-Las órdenes del G-Code solo se cargan por archivo de texto, esto es debido al uso de los métodos qpropertyanimation y qsequentialanimationgroup.

-Hay bugs en ciertas combinaciones de movimientos, debido al cambio de ejes de rotación, sobre todo en el eje de rotación de la última articulación, podríamos decir que se desacoplan los ejes y no hemos encontrado forma de arreglarlo.

-No se encontró forma sencilla de informar el estado del efector final en sincronismo con la animación. Esto es porque el efector final al no realizar ninguna qtransform, la única forma de visualizarlo en la animación es mediante una pausa en la ejecución.
qsecuentialanimationgroup->addpause(tiempo);

-Por sencillez se “hardcodeo” el path del archivo a abrir conteniendo el G-Code. En una etapa temprana del desarrollo se decidió hacer eso para avanzar rápidamente en la implementación del control y las transformaciones de las QMesh. Nos dimos cuenta de eso cuando urgía terminar apropiadamente la implementación de las animaciones por lo que se tuvo que dejar así al no encontrar fácilmente una solución satisfactoria.

DIFICULTADES

El framework de QT está muy documentado para el uso de su propio lenguaje QML, tiene varios ejemplos, de los cuales su mayoría y en el caso de ejemplos Qt3D, que eran de nuestro interés están en QML, encontrando solo 2 ejemplos dados en C++ los cuales no están muy bien documentados. Además la información, ejemplos, tutoriales por parte de la comunidad en la red están más orientados al diseño de aplicaciones en 2d.
Algo a mencionar es que usamos la última versión de Qt 5.11, y mucho del material encontrado se encuentra para Qt version 4, increíblemente la compatibilidad de una versión a otra cambia mucho, ya que no se respetan del todo los métodos de sus librerías propias. sin embargo hay que destacar que una vez comprendido cómo encontrar información se hizo más amena la lectura de la documentación oficial de qt.
La nula experiencia previa en diseño de interfaces gráficas nos puso en una situación en la que tuvimos que aprender el framework y aprender a hacer un aceptable diseño de clases a la vez; con la consecuencia de no haber hecho la mejor implementación posible. si se tuviera que comenzar todo de nuevo lo haríamos de otra forma. por el alto grado de avance que teníamos al momento de

darnos cuenta de nuestros errores decidimos convivir con ellos y tratar de subsanarlos sobre la marcha en lugar de reescribir totalmente el proyecto.
Otra de las dificultades a comentar es que debido al tamaño que fue adquiriendo el proyecto y a la inexperiencia con el desarrollo de proyectos tan grandes, nos fue complicado podernos ubicar en el codigo en las ultimas partes de desarrollo.
Por todo esto cabe decir que la curva de aprendizaje de Qt es muy empinada, siendo una herramienta muy potente y usada en la industria del software de interfaces gráficas, creemos que el hecho de aprender a usar bien este framework proporciona cierto tipo de habilidades para poder empezar laboralmente en el campo de las interfaces en caso de que se quisiera profundizar.

CONCLUSIONES:

Con la realización de este proyecto final, hemos podido agrupar todos los conocimientos aprendidos a lo largo del cursado de la materia como por ejemplo el diseño mediante diagramas UML, y aplicarlos a la resolución de un trabajo específico.
Además, ante la necesidad de incorporar el uso de nuevas herramientas, aprendimos la metodología que se debe seguir para poder aplicar nuevos conceptos, la incorporación de nuevas librerías a un trabajo, así como también los grandes beneficios y facilidades que esto trae.
En este caso particular, logramos aprender sobre herramientas para el desarrollo de interfaces gráficas con visualizaciones en 3D usando el framework de QT.
También, al tratarse de un ejercicio complejo y aplicando el concepto de modularidad, se pudo dividir el trabajo total, de manera que cada integrante pudo programar desde su casa en el momento de trabajo que le fuere más cómodo sin necesidad de juntarse en grupo, mejorando de esta manera la eficiencia y nuestras capacidades de trabajo en equipo.

BIBLIOGRAFÍA

Dudas con C++: Cpp Reference, StackOverflow, CPlusPlus
Documentacion oficial de Qt