Índice

Resumen

El presente trabajo se responde a lo solicitado como trabajo práctico integrador de la cátedra de *Programación Orientada a Objetos* de la /Facultad de Ingeniería/ de la /Universidad Nacional de Cuyo/, Mendoza, Argentina. Año 2018. Prof. Ing. Esp. César Aranda.
El objetivo del trabajo consiste en controlar un robot de 3 grados de libertad con efector final, a modo de simulación.
Dado que la cátedra es de programación orientada a objetos, la resolución del problema planteado está representada e implementada usando un **paradigma orientado a objetos**. Está diseñado utilizando un **modelo de capas**, de manera que la solución puede usarse independientemente de que haya interfaz gráfica o de consola. Debido a que el lenguaje de programación aprendido y utilizado durante el cursado de la cátedra es **C++**, lenguaje que cumple los requerimientos que tiene la solución del problema, siendo uno de los más indicados en problemas de éste tipo, se lo adoptó como lenguaje principal del proyecto. Todo el proyecto es desarrollado bajo **sistema operativo Linux**. Para la implementación de la interfaz de usuario, es requerido que sea gráfica, con interacción estándar mediante mouse y teclado. En este caso se utiliza las **librerías del framework Qt5**.

1. Sobre el brazo robótico simulado

El robot que se simula es un **modelo RRR**, lo que quiere decir que sus tres grados de libertad corresponden a rotaciones. El tipo propuesto es uno similar al modelo **Mitsubishi RV-2AJ**.

1.1. Modelo 3D del robot

La solución propuesta para la vista del robot en la interfaz, es que ésta sea tridimensional.
Buscando en la web, se encontró en la página GrabCad Community un gran aporte de Filipe Barbosa, de un robot Mitsubishi RV-6S Industrial MELFA Series, figura 1, que responde al modelo RRR.

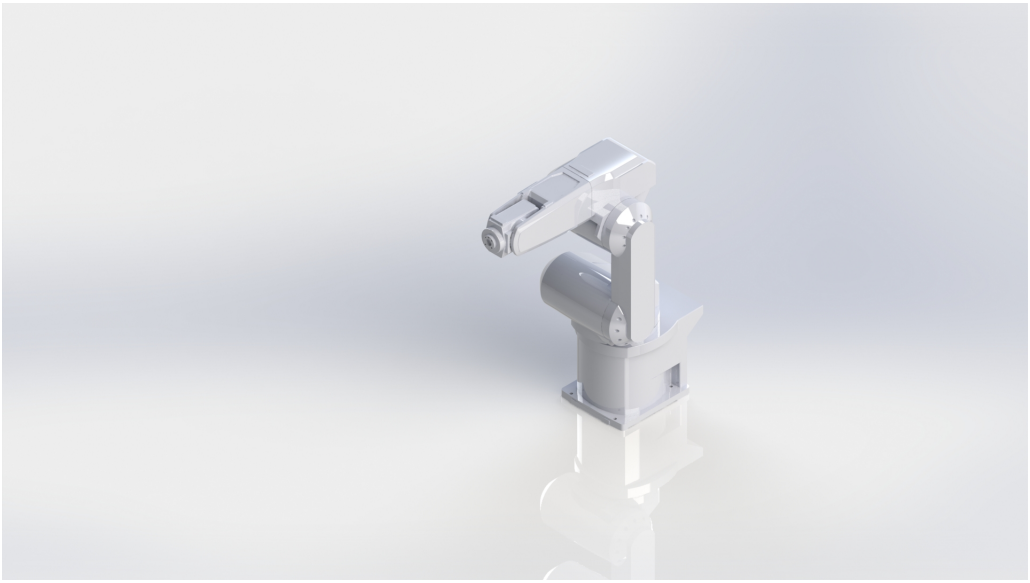

Figura 1: Render de modelo 3D Mitsubishi RV-6S Industrial MELFA Series.

1.2. Incorporación del modelo al proyecto

El modelo se encontraba en formato con extensión *.stp*. Para poder utilizarlo correctamente en el framework Qt5, se convirtió el modelo a formato *.obj*. El proceso de conversión fue de *.stp* a un ensamble de piezas en formato *.stl*, y con ayuda del software *Blender*, se exportó este ensamble en una versión simplificada de 5 piezas en formato *.obje* que se encuentra en la carpeta assets.

2. Instrucciones tipo G-Code

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

Instrucción	Descripción
G0 AX PY	Consigna de posición, donde X es la articulación a rotar e Y los grados.
G1 AX VY	Consigna de velocidad, donde X es la articulación a setear velocidad e Y el valor de la velocidad en radianes por segundo [rad/s]
G28	Homing y/o encender robot.
M0	Apagar robot.
TX	Cambio de actividad del efector final. Donde X es: - 0: Pintar - 1: Sostener - 2: Soltar - 3: Rotar
DXX	Duración de la actividad del efector final, dada en segundos [s]
EF	Inicio de actividad del efector final.

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

3. Interfaz gráfica de usuario

Como ya se mencionó anteriormente, la interfaz gráfica de usuario se diseñó utilizando librerías proveídas por Qt, en este caso las principales utilizadas son *QWidget*, *QMainWindow* y la ventana 3D utilizando la librería *Qt3DCore*.

3.1. Controles de cámara en ventana 3D

Los controles configurados para la interfaz fueron tipo *Orbit Controllers*, donde se hace zoom con el scroll del mouse, se traslada con el click izquierdo y se rota con el click derecho. En la carpeta vids pueden observarse animaciones de dicha ventana.

3.2. Panel de control

3.2.1. Ventana principal

El panel de control consiste en una ventana principal (clase *Widget*), con botones que permiten el encendido y apagado del robot. El encendido procede automáticamente con un secuencia de movimientos de homing. La acción de apagado frenará el robot en la posición en que esté, dandole un estado inactivo. Esta ventana principal también posee un cuadro de texto indicador del estado global del robot (encendido o apagado) y otros dos cuadros de texto que proveen la información del ángulo en que se encuentra cada articulación, su velocidad relativa, e información de la configuración del efector final. Los botones "Establecer características iniciales", "Cargar archivo de comandos" y "Comenzar movimiento" abren las otras respectivas ventanas. En el inferior de la ventana se encuentra el botón salir para cerrar ésta.

3.2.2. Características

Desde la ventana principal, si se presiona el botón "Establecer características iniciales", se abre la ventana de "Características". En esta ventana el usuario puede introducir los parámetros que desee del robot: Ángulos de las articulaciones y sus respectivas velocidades. Una vez introducidos estos parámetros, cuando el usuario presione "Aceptar" estos datos serán cargados y si son diferentes a los previos el robot procederá a ejecutar los movimientos correspondientes.

3.2.3. Operación

Si desde la ventana principal el usuario presiona la ventana principal, se abre la ventana de operación. Básicamente desde esta ventana el usuario tiene completo control sobre el robot, y puede moverlo ya sea introduciendo linea a linea comandos GCode en el cuadro de texto correspondiente o controlando ángulo y velocidad de cada articulación a través de los correspondientes sliders de la ventana. En el inferior de la ventana se encuentra un botón para cargar un archivo de comandos GCode y otro para realizar homing del robot. El botón "Terminar" cierra la ventana.

3.2.4. Carga de archivo de comandos

Si desde la ventana principal o desde la ventana de operación, el usuario presiona el botón para cargar archivo de comandos, se abrirá una ventana con dos cuadros de texto y una pestaña superior. La pestaña superior "File" puede presionarse, y se abrirá el gestor de archivos correspondiente para que el usuario cargué su archivo de texto con el programa de instrucciones GCode que desee ejecutar. Una vez seleccionado, en un cuadro de texto de la ventana se pude observar una vista preliminar del archivo y en el otro la ruta de éste. En la parte inferior de la ventana se encuentran dos botones, "Aceptar" y "Cancelar". El botón Aceptar cargará el programa seleccionado y lo ejecutará, El botón Cancelar cerrará la ventana sin que ninguna acción suceda.

4. Características del lenguaje y framework utilizado

Como se mencionó antes, el proyecto está desarrollado en C++ basado en el framework Qt5, específicamente la versión 5.11. Se aplicó la mayor parte de los conceptos aprendidos en clases: *abstracción, modularidad, encapsulamiento, otros más específicos como herencia, agregación y composición, constructores y destructores, polimorfismo, sobrecarga de métodos, etc.* Por extensión del documento, se decide exponer a continuación una característica del framework que no se dió en clases y por lo tanto puede resultar más enriquecedor. Todo los conocimientos adquiridos del framework Qt fueron obtenidos gracias a su documentación. El framework posee librerías que brindan las herramientas para conectar diferentes clases mediante la emisión y recepción de señales, esto se denomins **SIGNALS** y **SLOTS**. El gran potencial de ésto es que hay señales que son emitidas y están implícitas en la librería del framework (y no explícitas en la implementación), que se pueden utilizar de la dorma que se expone a continuación: Dando como ejemplo un caso utilizado. La clase BaseRobot es de la forma:

```
//baserobot.h
#include <QObject>

class BaseRobot : public QObject {
public:
  ...
public slots:
  void endReceiver();
}
```

El método endReceiver implementato en ./baserobot.cpp, recibe la señal *finished()* perteneciente a la clase *QPropertyAnimation* propia del framework. Esta señal se emite cada vez que un clip de animación asociado (en este caso la animación de un movimiento del robot) finaliza. En el programa se utiliza para saber cuando ejecutar la siguiente instrucción en la cola. Para conectar la animación con el *SLOT endReceiver()*, se utiliza la función connect() de la siguiente forma:

```
// baserobot.cpp
#include "baserobot.h"

void BaseRobot::externalGdl1(int value){
  if (this->estado == ACTIVE){
          QParallelAnimationGroup *motion =
                      new QParallelAnimationGroup();
          this->gdl1Changed(value);

          motion->addAnimation(this->p2->animate(
                      this->p2->getPrevious_angle(0),
                      this->p2->getAngle(0), this->p2->getDuration(),
                      0));
          ...
```

```
            connect(motion, &QParallelAnimationGroup::finished,
                    this, &BaseRobot::endReceiver);
            motion->start();
            this->estado = RUNNING;
        }
}
```

El objeto *motion* es de la clase *QParallelAnimation*, y contiene el conjunto de animaciones a ejecutar al cambiar el primer grado de libertad. Como puede observarse la función *connect()* recibe 4 argumentos: el objeto que envía la señal (motion), la dirección de la declaración de la señal a conectar (*finished*), el objeto que recibe la señal (*BaserRobot*, es decir, *this*), y por último la dirección de la declaración del *slot* a conectar (*endReceiver()*). Una vez configurado ésto, cada vez que se emita la señal *finished()* de parte de motion, se ejecutará la función *endReceiver()*. Para una explicación más clara y profunda, ir a la documentación oficial.

5. Diagrama UML

El diagrama UML de la figura 2 fue realizado con el software *Umbrello UML Modeller*. Es provisorio ya que todavía está sujeto a cambios, pero estos cambios no provocarías grandes modificaciones en la estructura del proyecto y, por lo tanto, en el plano, sino más bien son cambios que modificarían algunos métodos de la clase *BaseRobot, Elemento y Controller*.

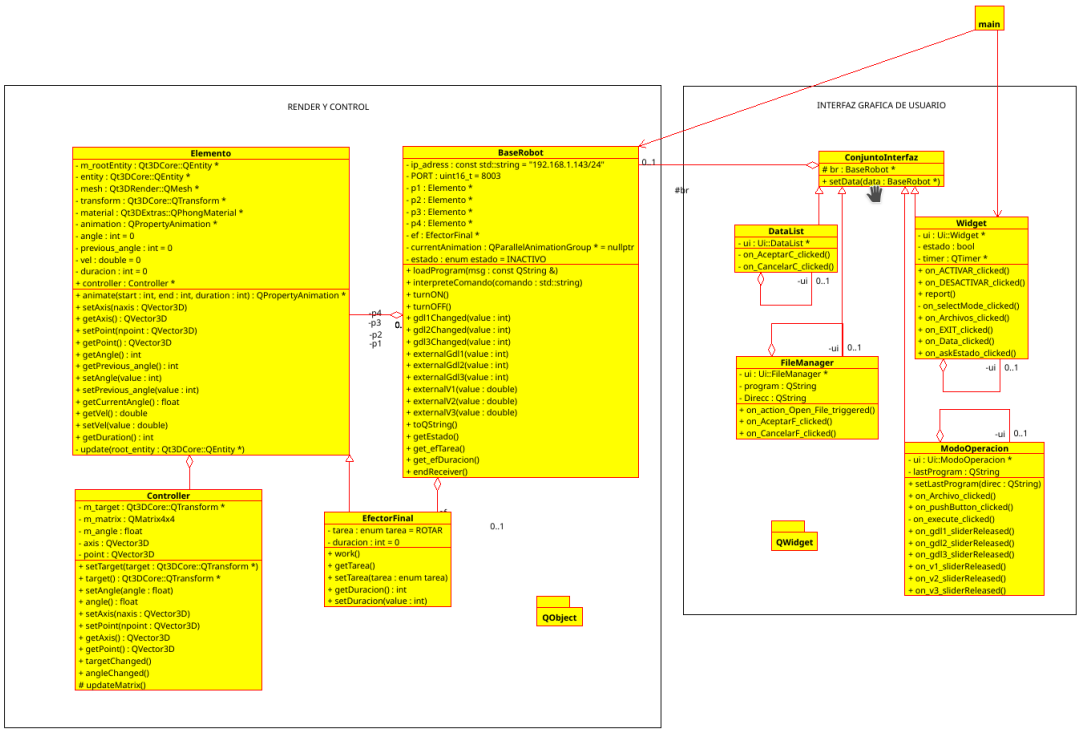

Figura 2: Diagrama UML provisorio del proyecto.

6. Descripción de clases

6.1. BaseRobot

Como puede observarse en el diagrama UML, la clase principal del proyecto es la denominada BaseRobot, está clase es la encargada del manejo de datos y comunicación con la interfaz gráfica de usuario. Es por esto que la instanciación de BaseRobot en todo el proyecto es solo una, y se ve agregada en todas las diferentes clases relacionadas con la GUI. Esta clase, como muchas otras en el proyecto, hereda de QObject librería perteneciente al framework utilizado. En rasgos generales, tiene métodos para cargar archivo de texto con el programa G-Code a realizar, método para la ejecución de dicho programa gracias

a otros de sus métodos más importantes que es el interprete de comando que como su nombre indica, interpreta cada línea G-Code y la agraga a la cola de instrucciones (atributo de la clase). BaseRobot es la clase que posee métodos para la variación de los datos correspondientes a la configuración del robot: sus ángulos y velocidades. También posee información del estado de éste: activo, inactivo y en movimiento, también información necesaria para la comunicación con el robot: dirección IP y puerto de comunicación (información no representativa para la simulación). Otros métodos importantes en esta clase permiten tanto el apagado y encendido del robot, como el comienzo de la ejecución de su cola de instrucciones. Respecto a la cola de instrucciones, como los movimientos del robot están dados por clips de animación, cuándo cada uno de los clips finaliza o, más general, cuándo una instrucción finaliza, se emite una señal que es recibida por la clase para eliminar la instrucción de la cola y continuar con la siguiente. También posee un método para limpiar dicha cola, ésto sucede cuándo se realiza una maniobra de homing o cuándo se apaga la máuina. Ésta clase agrega 4 objetos clase Elemento y uno clase Efector Final que contienen la información relacionada a cada pieza "física" del brazo.

6.2. Elemento

La clase Elemento es la clase encargada de albergar toda la información correspondiente a la pieza del brazo de su instanciación. Su constructor recibe como parámetro el rootEntity de la entidad robot que se crea en BaseRobot. También hereda la librería QObject al igual que BaseRobot. Los parámetros de esta clase están muy relacionados al render de la pieza, del modelo .obj: un mesh del modelo 3D en sí, un material que provee el framework, una animación con la que se trabajará para parametrizar los movimientos, una transformación, herramienta con la que se ejecutan las traslaciones y rotaciones del mesh y una entidad propia a la que se le agregas los componentes anteriores. Además de toda la información anterior, los métodos de la clase permiten el get y set de los puntos, ejes y ángulos necesarios para definir la configuración de la pieza. Otros métodos dan la configuración de la animación, como la duración de ésta en base a la velocidad seteada. Cada elemento agrega un objeto clase Controller que se describe más adelante.

6.3. EfectorFinal

La clase EfectorFinal es una clase que hereda de Elemento y está asociada al efector final del robot. En ella se encuentran métodos y atributos para el control de la tarea que debe ejecutar y su duración. Las posibles tareas son pintar, sostener, soltar y rotar.

6.4. Controller

La clase Controller instancia los controladores de cada Elemento. Estos controladores tienen como target los atributos QTransform de cada elemento, es decir, manejan el aspecto de transformación de los mesh de cada parte del robot. Esta manipulación de la transformación es para dar los valores de la configuración de la pieza en cada frame de la animación. Por lo tanto, hay métodos encadenados con métodos de la clase Elemento correspondiente como los get y set de los puntos, ejes y ángulos de la pieza. La función más importante de la clase Controller es la que actualiza la matriz 4x4 característica de la transformación del elemento, donde está la información correspondiente a traslaciones y rotaciones ejecutadas sobre la pieza. Existen señales internas en la clase que informan el cambio en el ángulo o target seteado en la instanciación del controlador.

6.5. ConjuntoInterfaz

La clase ConjuntoInterfaz es una superclase de todas aquellas relacionadas con la interfaz gráfica de usuario. Agrega a BaseRobot y tiene como método una función que permite setear éste atributo con la instancia de BaseRobot correspondiente. Por lo que permite vincular a todas las clases relacionadas con la GUI a las clases orientadas al control de la simulación.

6.6. Otras clases

Las clases Widget, DataList, ModoOperacion y FileManager son las relacionadas con la GUI. Para no extender la descripción del código, no se realiza un análisis detallado de sus métodos y atributos, éstos están muy relacionados con el diseño de las diferentes ventanas asociadas que se describen en la sección de ésta temática. Sin embargo, si vale resaltar que se implementan utilizando signals, slots y objetos ui;

donde gran parte del código es generado automáticamente por el IDE de Qt QtCreator. Otro aspecto a resaltar, es que estas clases tienen herencia múltiple: de la clase ConjuntoInterfaz como ya se mencionó, y de QWidget, librería propia del framework como QObject. La librería QWidget permite dentro del IDE diseñar las ventanas de una forma más didáctica y facilita la tarea con la herramienta Design.

7. Diagrama de secuencia temporal

Los diagramas de secuencia expuestos a continuación, fueron realizados al igual que el diagrama UML, con el software *Umbrello UML Modeller*. El orden de instanciación de objetos al lanzar o ejecutar la aplicación es como se describe en la figura 3.

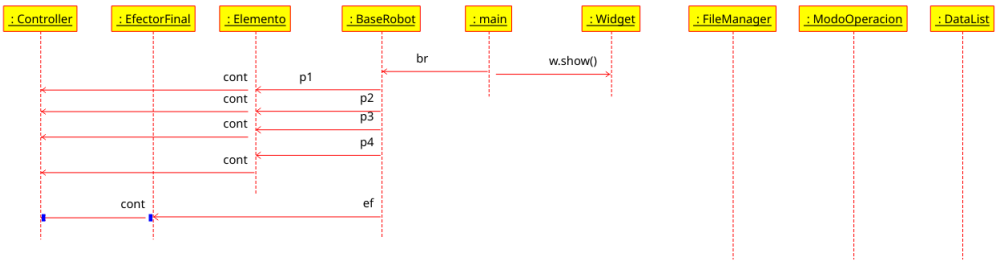

Figura 3: Diagrama de secuencia al lanzar la aplicación.

En la figura 4, un diagrama de secuencia a modo de ejemplo, de un caso de uso de la aplicación en la que se ejecutan la mayoría de las señales implementadas en el proyecto. Este caso es la carga de un archivo de comandos G-Code desde la ventana de modo de operación.

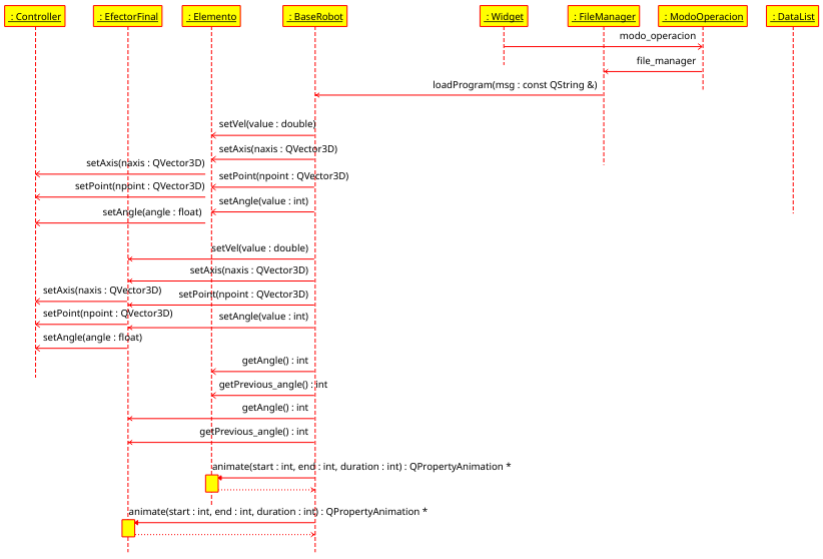

Figura 4: Diagrama de secuencia para la carga de un archivo de comandos G-Code desde la ventana de modo de operación.

8. Diagrama de actividad

El diagrama de actividad también se obtiene del mismo archivo en el que se realizó el diagrama UML y de secuencia. En la figura 5, se plantea para el mismo caso de la ejecución de un archivo de instrucciones G-Code desde la ventana de modo de operación o de uso.

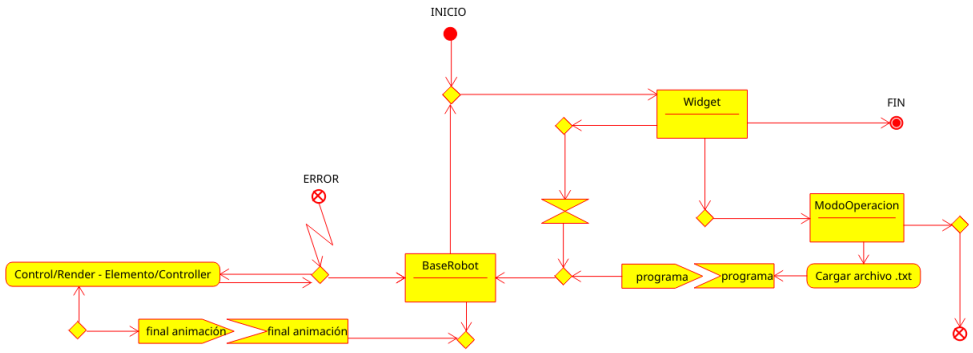

Figura 5: Diagrama de actividad para la selección de un archivo de instrucciones G-Code desde la ventana de modo de operación o de uso.

9. Sonido

La simulación realiza una señal sonora cada vez que el robot inicia un movimiento de rotación en alguno de sus grados de libertad. Esta alarma sonora puede encontrarse en ./assets/sound.wav. Este sonido se puede implementar haciendo uso de la clase *QSound* que provee el framework.

10. Comentarios y conclusiones

10.1. Versión para navegador web. HTML5 y Javascript

En el inicio del proyecto se presentó gran dificultad para aprender y dominar el framework Qt. Dada esta dificultad, se decidió comenzar el proyecto con otras herramientas diferentes, y una vez avanzadas y terminadas ciertas etapas de diseño básico de la aplicación y esquemas generales del programa, proceder a mudar el proyecto dentro del framework con la esperanza de que partiendo sobre una base lo suficientemente sólida, adaptarse al framework sea de una manera menos agresiva que arrancando desde cero. Se fue consciente de que esta decisión costaría mayor cantidad de tiempo de trabajo, pero con la ventaja de que si se venciera el plazo de entrega, el grupo dispondría de una solución que a pesar de no estar diseñada con las herramientas planteadas, cumpliría los requisitos básicos. El framework Qt ofrece entre sus tantas plantillas de tipos de proyectos (canvas 3D, aplicación de consola, aplicación mobile, etc.), la posibilidad de realizar la parte gráfica de renderizado y animación 3D con una librería de **javascript** muy conocida denominada three.js. Como uno de los integrantes del grupo posee conocimientos básicos en *HTML5* y *javascript*, y éste último también puede utilizarse con paradigma de objetos, se decidió utilizar esta librería en un entorno web,, suponiendo que mudarlo a Qt seía una opción viable. La experiencia con ésta librería fue muy buena, y hubo grandes avances en un corto periodo de tiempo: se pudo importar el modelo 3D, animar todas sus articulaciones, controlar éstas mediante sliders, colocar indicadores luminosos del estado del robot, y crear una interfaz de usuario muy básica en *HTML5*.

10.1.1. C++ addons para Node.js

Por más que va más alla del alcance del proyecto, se investigó como realizar el control y cálculos matemáticos de la simulación del brazo robótico implementado con *C++* en un entorno web donde los lenguajes de desarrollo, como ya se mencionó, son *HTML5* y *Javascript*. Con palabras más claras, surgió la pregunta de cómo desarrolar una página web para la simulación, donde el frontend esté implementado en HTML5 y Javascript, y el backend con C++. Inmediatamente surgió que la respuesta es trabajar el proyecto con Node.js.
"**Node.js** es un entorno en tiempo de ejecución multiplataforma, open-source, para la capa del servidor, basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google" - Wikipedia, La enciclopedia libre.
Es posible la creación de "addons" implementados en C++, que por medio de herramientas facilitadas por node y otras librerías de javascript permiten el llamado de estas funciones en C++ desde código escrito en Javascript. Esta solución era de una complejidad mayor a la esperada, así que se opto por

comenzar el proceso de mudar el proyecto de su implementación con three.js en un entorno web al framework Qt.

10.2. Versión con framework Qt, aplicación 3D. C++, QML y Javascript.

El proceso de mudar el proyecto desde un entorno web basado en HTML5 y Javascript al framework Qt resultó ser relativamente sencillo, eligiendo la plantilla de proyecto "aplicación 3D basada en librería three.js. El modo de trabajo es muy similar en aspectos generales a lo que se deseaba hacer con los addons de node. En este caso el intermediario entre lenguajes C++ y Javascript es QML.
"**QML** (del inglés, Qt Meta Language) es un lenguaje basado en JavaScript creado para diseñar aplicaciones enfocadas a la interfaz de usuario. Es parte de Qt Quick, el kit de Interfaz de usuario creado por Digia junto al framework Qt." - Wikipedia, la enciclopedia libre.
El IDE de Qt, QtCreator, ofrece su herramienta Design que sirve para facilitar el diseño gráfico generando automáticamente código QML o C++ según sea el caso. A partir de este código en QML se puede ejecutar como funciones en javascript, las implementadas para la versión anterior del proyecto, y obtener información de lo que sucede en los scripts de javascript en lenguaje C++. Gracias a éstas herramientas, fue posible diseñar una aplicación de escritorio con una interfaz adecuada, pero aún utilizando el código escrito previamente. Los resultados de esta versión del proyecto se encuentran en la carpeta ./etc/. Se puede pbservar animaciones en de esta versión en la carpeta ./vids/.

10.3. Versión con framework Qt. C++ puro.

La versión del proyecto previamente expuesta tuvo un visto bueno de parte del profesor previa a la entrega. Sin embargo, el grupo era consciente de que el lenguaje que se aprendió en la catedra Programación Orientada a Objetos fue C++, y era altamente recomendado realizar la implementación del trabajo en éste lenguaje. Además, uno de los inconvenientes en la versión con QML del proyecto es que la animación, realizada por medio de la librería three.js en javascript, con el control y GUI, implementado en C++, estaban realmente disociados. Esta disociación no era una ventaja como uno esperaría de un diseño de capas, sino que realmente estaban desconectados llegando al punto en que una de las soluciones era actualizar toda la información en intervalos discretos de tiempo. Puede ser una opción en el planteo de una página web donde el procesamiento se hace en el backend, pero el rendimiento de la aplicación podía incrementarse mucho si la aplicación fuera nativa e implementada completamente en C++. Es por eso que se llegó a la versión expuesta al principio de éste informe, donde a pesar de poseer muchos bugs en la animación del robot, es notable la diferencia de desempeño.

10.4. Dificultades presentadas

10.4.1. Modelo 3D

Que el diseño del robot sea de un tercero, y el poco conocimiento en software para modelado 3D dificultó el manejo de dimensiones del robot, de modo que se tuvo que "hardcodear" la posición de los puntos y ejes por los que debe realizarse la rotación de los diferentes grados de libertad.

10.4.2. Framework

La experiencia con el framework fue diferente a otras. El inicio es muy duro, la dificultad para arrancar desde cero es notable. Sin embargo, una vez que se aprenden y dominan los aspectos fundamentales, es realmente impresionante el potencial del framework. De hecho, animar el robot tuvo un grado de dificultad mucho mayor al de las otras opciones, y es por eso que a la fecha de entrega el proyecto tiene muchos errores en este aspecto. Pero, leyendo la documentación y comentándolo en el grupo, se reconoció el potencial, por ejemplo, de la clase QTransform. Teniendo conocimientos de la cátedra Robótica I es posible realizar las operaciones necesarias con la clase de modo de obtener una simulación muy fina y precisa, respetando toda la física implicada.

11. A futuro

- Corregir errores.
- Planear un diseño aún más enfocado en el modelo de capas.

- Realizar una versión web y publicarla.
- Construir robot físico y vincular a cátedra Microcontroladores y Electrónica de Potencia, de modo de tener un entorno virtual y físico del robot.
- Realizar diseño 3D propio.
- Continuar, ahora con mayor dominio del framework utilizado.

12. Licencia

GNU General Public License v3.0.

13. Repositorio GitHub

RobotArm-Simulation

14. Bibliografía

- Modelo 3D del robot: https://grabcad.com/library/mitsubishi-rv-6s-1
- Dudas con C++: Cpp Reference, StackOverflow
- A fines prácticos, y no mencionar cadda clase utilizada, toda la información para poder realizar el proyecto en el framework Qt, se obtuvo de la documentación oficial: Qt
- Librería para versión web: three.js
- Recursos para *backend* de versión web: Node.js