Universidad
Carlos III de Madrid

GRADO EN INGENIERÍA ELECTRÓNICA Y
AUTOMÁTICA INDUSTRIAL

TRABAJO FIN DE GRADO

Modelo cinemático y control de un brazo robótico imprimible

Tutor: Dr. Alberto Valero Gómez

Autor: D. Juan Carlos Rodríguez Zambrana

Agradecimientos

En primer lugar, quiero expresar mi agradecimiento al Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid y en especial a mi tutor, D. Alberto Valero Gómez y a mi profesor D. Juan González Gómez. Ellos me brindaron la oportunidad de embarcarme en este proyecto. Además me enseñaron el maravilloso mundo de la robótica y la impresión 3D.

También quiero agradecer a mis compañeros Ana De Prado Navarrete, María Ramos Montero y Jose Manzano Fraile por compartir conmigo sus conocimientos y darme todo su apoyo. Gracias también a mi compañero Antonio Castro Gómez por su colaboración y por cederme su diseño del brazo robótico imprimible.

Por último, agradecer a mi familia todo el esfuerzo que han hecho por mí. Mención especial para mi padre que ha revisado exhaustivamente esta memoria.

Resumen

El objetivo de este proyecto es el desarrollo del modelo cinemático, el control y la simulación mediante un ordenador de un brazo robótico imprimible. Dicho robot es el objeto del proyecto fin de carrera de Antonio Castro Gómez. Así, se intenta dar una continuidad al proyecto realizando una interfaz gráfica para su control.

Para ello, se ha diseñado e implementado un programa en C++ y OpenGL. Este programa incorpora una interfaz gráfica que permite la representación en tres dimensiones de la posición y orientación del brazo robótico. Además se indican por pantalla los valores de los ángulos de los ejes y la posición del extremo del robot.

El funcionamiento del programa consta de los siguientes pasos: el usuario manda una orden a través del gamepad, se recibe y se analiza el mensaje, se simula los movimientos en la pantalla y, finalmente, se ejecuta la orden en el robot real.

Se ha dotado al simulador de tres modos de funcionamiento. En primer lugar, tenemos el modo de teleoperación en el que el usuario mueve el robot a través de los botones del gamepad. En segundo lugar, el modo de grabación permite guardar una sucesión de movimientos realizado por el usuario y posteriormente reproducirlos en el robot real. En tercer y último lugar, con el modo de posicionamiento el usuario puede introducir las coordenadas de un punto del espacio donde se situará el extremo del robot.

Se ha elaborado el modelo cinemático completo del brazo robótico. Por un lado, el problema de la cinemática directa ha sido solucionado a través del algoritmo de Denavit-Hartenberg. Por otro lado, se ha resuelto la cinemática inversa a través de métodos geométricos.

Para la comunicación entre el robot y el punto de control (un ordenador) se contemplan dos posibilidades. La primera es la utilización de un cable USB que otorga mayor fiabilidad a la comunicación. La segunda posibilidad utiliza comunicación inalámbrica mediante Bluetooth, permitiendo un movimiento más libre de hasta 9 metros de distancia al punto de control.

El propósito final es la obtención de un brazo robótico imprimible, de bajo coste y con un método de control y simulación de fácil manejo. Tanto las piezas del robot como el software que lo controla están en código abierto. De esta forma, se posibilita la modificación, ampliación o mejora del proyecto por parte de la comunidad.

Abstract

The objective of this project is to develop the kinematic model, the control and simulation using a computer of a printable robotic arm. This robot is the subject of the final project of Antonio Castro Gómez. The intention is to give continuity to the project by making a GUI for its control.

To do this, we have designed and implemented a program in C + + and OpenGL. This program has a graphical interface that displays a three-dimensional representation of the position and orientation of the robot arm. Furthermore the values of the angles of the axes and the position of the end of the robot are indicated on the screen.

The operation of the program consists of the following steps: the user sends a command through the gamepad, the message is received and analyzed, it simulates the movements on the screen and, finally, the command is executed by the real robot.

The simulator has been provided with three operating modes. First, we teleoperation mode in which the user moves the robot with the buttons on the gamepad. Second, the recording mode to save a sequence of movements performed by the user and then reproduces them on the real robot. Third and finally, with the positioning mode the user can enter the coordinates of a point in space where you will place the end of the robot.

We have developed complete kinematic model of the robotic arm. On one hand, the direct kinematic problem has been solved through Denavit-Hartenberg algorithm. On the other hand, inverse kinematics resolved through geometric methods.

For communication between the robot and the control point (a computer) there are two possibilities. The first is the use of a USB for added reliability in communication. The second possibility uses Bluetooth wireless communication, allowing freer movement of up to 9 meters away from the checkpoint.

The final purpose is to obtain a low cost printable robotic arm with an easy to use control and simulation method. Both, the robot parts and the control software are open source. Thus, it allows the modification, extension or improvement of the project by the community.

Índice

Índice de Ilustraciones

Índice de tablas

Índice de ecuaciones

1. INTRODUCCIÓN

1.1. OBJETIVOS

El objetivo principal de este proyecto es el desarrollo del modelo cinemático, el control y la simulación de un brazo robótico imprimible. A continuación se expone en detalle, los diferentes puntos que abarca el proyecto:

- **Comunicación**

El usuario podrá comunicar órdenes de actuación al robot a través del teclado o del gamepad. El programa interpreta las indicaciones, muestra la nueva posición del robot en el simulador y se comunica con el microcontrolador. Éste interpreta el mensaje y manda las señales a los servomotores para que adopten la posición correspondiente.

- **Simulación en 3D del brazo robótico**

Para la simulación en 3D del brazo robótico, se implementará una interfaz gráfica que muestra una representación del robot con un posicionamiento y orientación determinados. Además se muestran por pantalla los valores de los ejes y la posición del extremo del robot.

- **Cinemática directa e inversa del brazo robótico**

Se quiere dar solución al problema cinemático directo e inverso del brazo robótico objeto de este trabajo. Así, se obtiene la posición del extremo del robot a partir de los valores de los ángulos del mismo y viceversa.

- **Teleoperación remota**

La operación remota del brazo robótico se posibilita gracias al simulador y al gamepad. El usuario recibe la información gráfica del posicionamiento y la orientación del robot a través del monitor del ordenador. Esto posibilita teleoperar el robot gracias a la representación gráfica del simulador.

- **Grabación de movimientos**

La grabación de movimientos será posible gracias al programa de simulación. Así, el usuario podrá realizar la grabación de movimientos simulados y reproducirlos en el robot real.

- **Posicionamiento**

El usuario podrá especificar un punto del espacio y ordenar al robot que adopte dicha posición. Esto será posible gracias a la implementación de un modelo cinemático inverso completo.

1.1. ANÁLISIS DEL PROBLEMA

Para realizar el control y la simulación, de un brazo robótico imprimible se nos plantean los siguientes problemas:

1. Comunicar el gamepad con el ordenador y éste con el microcontrolador para controlar los movimientos del robot.
2. Realizar un simulador del robot con una representación 3D de la posición y la orientación del mismo.
3. Mostrar los valores numéricos de la posición del extremo del robot (x, y, z) por pantalla. Para ello será necesario solucionar el problema de la cinemática directa del robot.
4. Conseguir que el robot alcance la posición del espacio (x, y, z) indicada por el usuario. Para ello será necesario solucionar el problema de la cinemática inversa.
5. Realizar la grabación de los movimientos seleccionados por el usuario y reproducirlos posteriormente.

1.3. SOLUCIONES EXISTENTES

Entre las soluciones existentes para el control y la simulación de brazos robóticos podemos diferenciar dos ámbitos:

- Software privado de fabricantes de robots
- Diferentes proyectos OpenSource como puede ser OpenRAVE.

Software privado de fabricantes de robots

Los fabricantes de robots poseen sus propios programas para el control y la simulación de sus productos. Se trata de software privado, muy elaborado y que conlleva un alto coste monetario. Éste software permite:

- El control de los ejes del robot
- La grabación de puntos y trayectorias
- La programación offline de los robots
- La elaboración de simulaciones complejas

Gracias a estos programas, antes de iniciar la puesta en servicio, se puede comprobar los procesos y, si procede, optimizarlos y validarlos.

Existen diversas empresas dedicadas a la fabricación de robots pudiendo mencionar fabricantes como ABB, KUKA, Toshiba, Honda, Epson, etc. Elegiremos los dos primeros fabricantes para analizar sus programas de simulación.

El fabricante ABB es propietario de un software llamado RobotStudio. Además, cuenta con un lenguaje de programación propio llamado Rapid. El entorno gráfico de RobotStudio

está compuesto por menús con una extensa variedad de herramienta. A continuación podemos ver una captura de pantalla del programa.

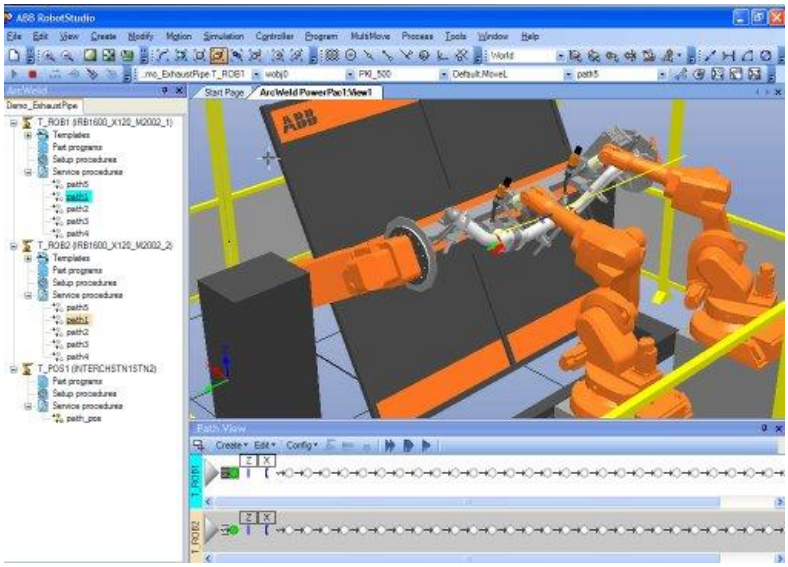


Ilustración 1. Software RobotStudio de la empresa ABB

El fabricante KUKA posee un software llamado KUKASim. Ofrece prácticamente las mismas posibilidades que el programa de ABB. El entorno gráfico también es similar como se puede apreciar en la siguiente ilustración.

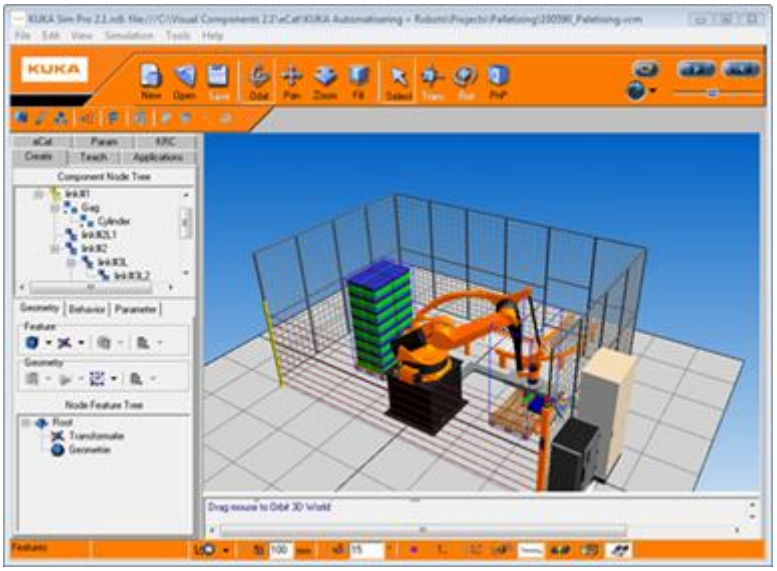


Ilustración 2. Software KUKASim de la empresa KUKA

OpenRAVE

OpenRave es un entorno libre para la simulación, modelado y planificación de robots, desarrollado por la comunidad científica. Es utilizado por investigadores de todo el mundo para realizar experimentos con sus últimos robots y/o algoritmos.

La atención se centra en la simulación y análisis de la información geométrica y cinemática relacionados con la planificación de movimiento. Permite una integración sencilla en los sistemas existentes de robótica. Las características principales de OpenRAVE son:

- Libre: cualquiera puede modificarlo para adaptarlo a sus necesidades.
- Multiplataforma: Linux/Mac/Windows.
- Creado por investigadores para investigadores.
- Utiliza el motor físico ODE (*Open Dynamics Engine*).
- Se pueden modelar piezas con mucho detalle.
- El visualizador de OpenRave permite interactuación con los objetos físicos de la simulación.
- Facilidad para generar vídeos de las simulaciones.

Existen diversos proyectos que, tomando OpenRAVE como base, realizan adaptaciones como es el caso de OpenGRASP. Se trata un conjunto de herramientas de código fuente abierto para la simulación del agarre y manipulación.

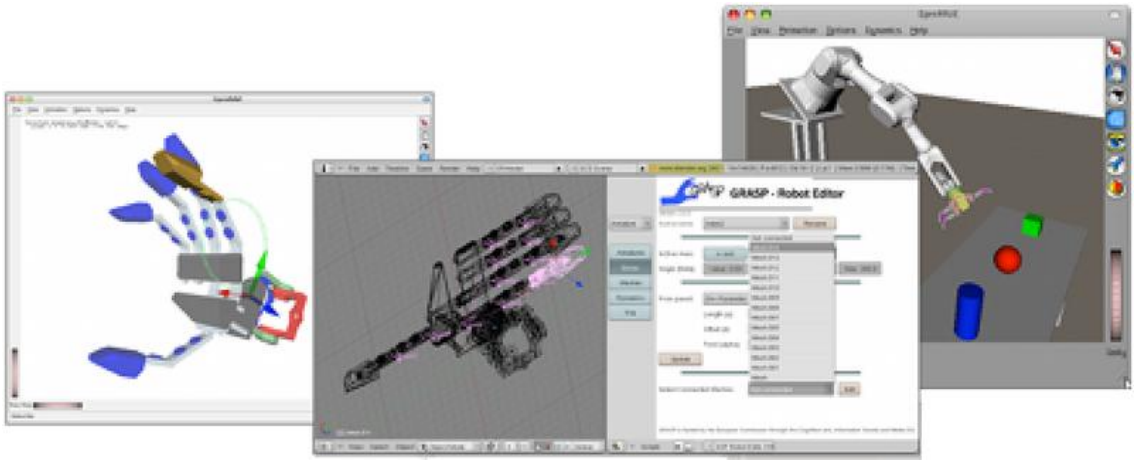


Ilustración 3. Simulador OpenGRASP adaptación de OpenRAVE

Otro ejemplo de adaptación de OpenRave, realizada por Juan González Gómez, es la denominada OpenMR. Ésta se centra en la simulación de la locomoción de los robots modulares. A continuación se muestra una captura de pantalla de dicho simulador.

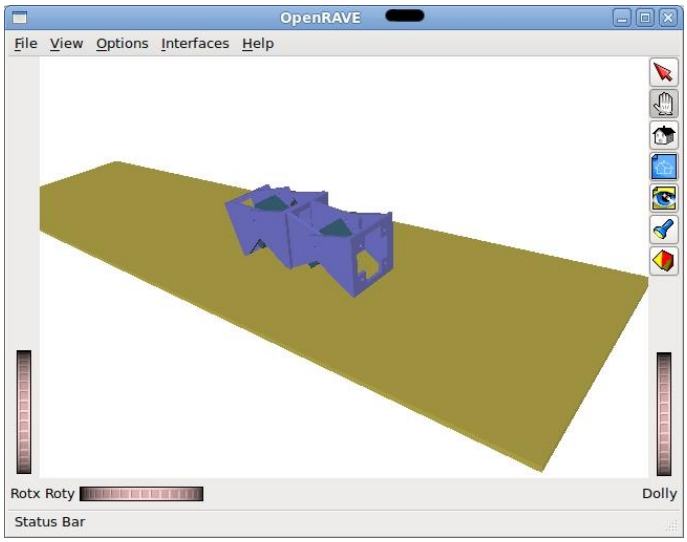


Ilustración 4. Simulador OpenMR adaptación de OpenRAVE

2. ELEMENTOS Y MEDIOS UTILIZADOS

En este apartado se describe los elementos utilizados en el proyecto. Principalmente cabe destacar la impresora 3D utilizada para la fabricación de las piezas y los servomotores que dan movimiento al brazo robótico.

Mencionaremos el microcontrolador que regirá el comportamiento de los anteriormente mencionados servomotores. Por último describiremos el mando o gamepad utilizado que será la herramienta mediante la cual el usuario controlará los movimientos del brazo robótico.

2.1. IMPRESORA 3D

Para la construcción de las piezas que forman el brazo robótico se ha utilizado una impresora 3D. El funcionamiento de ésta consiste en la extrusión, a alta temperatura, de un polímero termoplástico. Éste es extruido en finas líneas creando formas por capas.

Las dos impresoras utilizadas pertenecen a la Universidad Carlos III de Madrid. Son del modelo Thing-o-matic fabricadas por la empresa Makerbot Industries. En la ilustración siguiente, podemos ver una imagen frontal de una de las impresoras 3D utilizadas.

Ilustración 5. Impresora 3D modelo Thig-o-matic de Makerbot Industries

En concreto, el plástico utilizado es el acrilonitrilo butadieno estireno (ABS). Las dos impresoras utilizadas requieren un diámetro distinto de plástico. Éste puede ser de 3.00 mm o 1.75 mm según utilicemos una y otra impresora.

2.2. SERVOMOTORES

Para dar movimiento al brazo robótico imprimible se han utilizado tres servomotores modelo S3003 y uno modelo S3305 todos ellos de la marca Futaba. Éstos se ocupan del movimiento de la base y los dos primeros eslabones. Además el brazo robótico cuenta con cuatro micro-servomotores cuyo modelo es el SG90 de TowerPro. Éste modelo, de reducido peso y tamaño, se empleado en el último eslabón y en la muñeca del robot.

A continuación se muestran las imágenes correspondientes a los diferentes servomotores empleados. Además de un esquemático mostrando las dimensiones de éstos.

Ilustración 6. Servomotor Futaba S3305

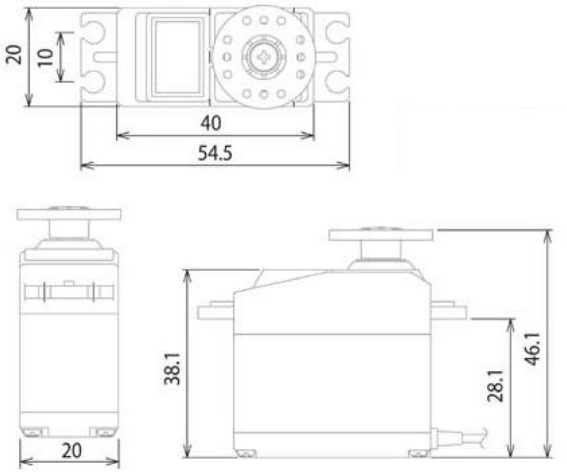


Ilustración 7. Dimensiones de los servomotores Futaba.

Ilustración 8. Micro-servomotor Tower Pro modelo SG90

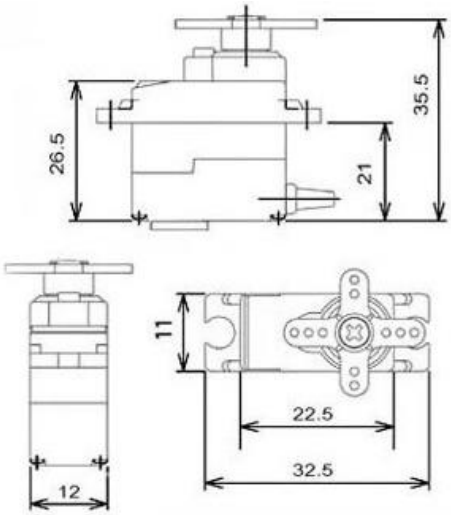


Ilustración 9. Dimensiones de los micro-servomotores

Los modelos S3003 y S3305 son análogos con excepción de su torque y su peso. En la tabla siguiente podemos ver una comparación de las diferentes características de los servomotores utilizados.

Tabla 1. Características principales de los servomotores utilizados

Características	Tipo de servomotor		
	FUTABA S3305	FUTABA S3003	Tower Pro SG90
Sistema de control	PWM	PWM	PWM
Tensión de funcionamiento [V]	4.8-6.0	4.8-6.0	4.0-7.2
Velocidad (a 6.0V) [s/°]	0.2s/60°	0.19s/60°	0.12s/60°
Torque (a 6.0V) [kg·cm]	8.9	4.1	1.2
Tamaño [mm]	40x20x38.1	40x20x38.1	22.5x12.0x26.5
Peso [g]	46.5	37.2	9.0

2.3. MICROCONTROLADOR

El microcontrolador empleado es el Arduino UNO. Utiliza un microprocesador ATMEGA328 del fabricante Atmel. Se han utilizado las salidas digitales y PWM para el control de los servomotores. A continuación se muestra una imagen de dicho microcontrolador.

Ilustración 10. Microcontrolador Arduino UNO.

La alimentación y la transmisión de datos desde el ordenador hacia el microcontrolador se realizan mediante un cable USB. Para el control de los servos el microcontrolador dispone de librerías específicas.

Los servomotores necesitan para su funcionamiento una señal de control de tipo PWM. El microcontrolador dispone de 14 salidas digitales de las cuales 6 proporcionan dicha señal PWM. No obstante, se podrán generar más salidas de ese tipo mediante el software interno.

Las características principales del microcontrolador están recogidas en la tabla expuesta a continuación.

Tabla 2. Características principales del microcontrolador utilizado

Microprocesador	ATMEGA328
Tensión de funcionamiento	5V
Voltaje de entrada (recomendada)	7-12V
Voltaje de entrada (límites)	6-20V
Entradas/salidas digitales	14
Entradas analógicas	6
Memoria Flash	32KB (ATMEGA328)
SRAM	2KB (ATMEGA328)
EPROM	1KB (ATMEGA328)
Velocidad de reloj	16MHz

2.5. GAMEPAD

Para el control de los ejes del robot se ha optado por un Gamepad. De esta forma, se consigue un control más sencillo del brazo robótico. El mando utilizado es un Dual Shock del fabricante Maxwise. Se conecta al ordenador por medio de un cable USB. En la imagen siguiente se muestra el tipo de mando de control utilizado.

Ilustración 11. Gamepad (mando de control)

Para describir las funciones asignadas al Gamepad describiremos en primer lugar los botones y en segundo lugar los 2 joysticks frontales. Por último, se describirá la programación realizada para el mando.

- **Botones**

Según la configuración del mando utilizado, los botones reciben la numeración que se especifica en la ilustración 12. Se utilizarán los dos Joystick para el movimiento de los 4 primeros ejes (q_1, q_2, q_3 y q_4) y la cruceta para los dos siguientes pertenecientes a la muñeca (q_5 y q_6).

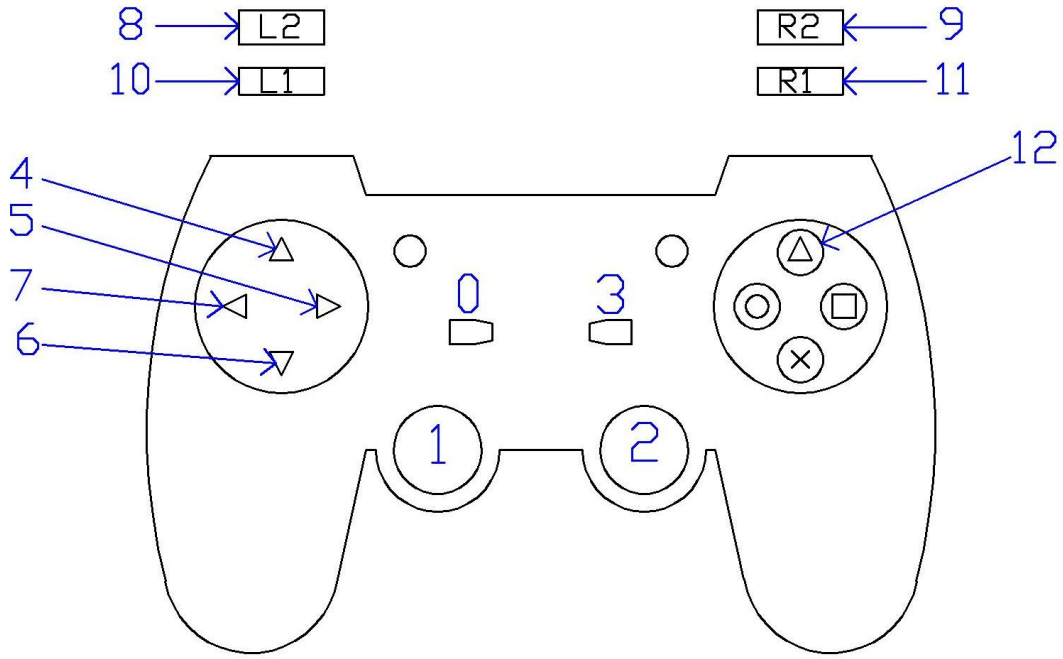


Ilustración 12. Gamepad vista frontal. Numeración de los botones.

- **Joysticks**

Los dos Joystick del Gamepad poseen movimiento en el eje X e Y tal y como se muestra en la ilustración siguiente. Adoptan valores de -180 a 180 según la dirección y el sentido del movimiento.

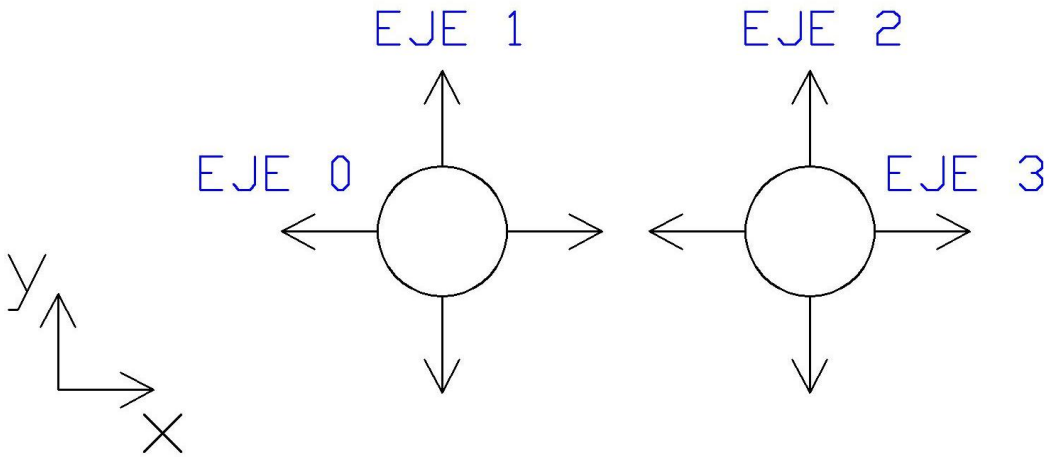


Ilustración 13. Ejes de los Joysticks frontales del Gamepad.

2.6. BATERÍA

Para la alimentación tanto de los servomotores como del microcontrolador se utilizará una batería de tipo polímero de litio (LiPo). Son una variación de las baterías de iones de litio (Li-ion). Poseen un tamaño y peso más reducido que las baterías de ión litio además de una mayor densidad de energía.

Concretamente se ha utilizado la batería del fabricante DragonRed cuyas características se detallan en la tabla siguiente.

Tabla 3. Características principales de la batería utilizada

Tipo de batería	Polímero de Litio
Tensión	7.4V
Densidad de energía	5000mAh
Número de celdas	2
Intensidad máxima	150A
Dimensiones	132x44x21 mm
Peso	235g

Se ha elegido la utilización de una batería de tipo LiPo por las siguientes razones:

- Alta densidad de energía, casi el doble que las basadas en Níquel.
- Alto voltaje por célula, lo que permite llegar a 7,4 V en dos celdas sin ocupar excesivo volumen y con un reducido peso.
- Poca resistencia interna que permite aprovechar casi el 100% de la energía almacenada.

Ilustración 14. Imagen de la batería de tipo LiPo

2.7. UBEC: REGULADOR DE VOLTAJE

Un "UBEC" es un regulador de voltaje de tipo conmutado ('switching'). Soporta una intensidad máxima de entrada de 5A y es adaptable a voltajes de entrada entre 5,5V y 23V. Esto nos permite acoplarlo a la batería LiPo descrita anteriormente ya que suministra 7,4V.

El modelo concreto elegido para este proyecto es el UBEC de HobbyWing con voltaje de salida seleccionable (5 o 6 voltios). Los valores de salida coinciden con los necesarios para alimentar los servomotores sin dañarlos.

Tabla 4. Características del UBEC

Tensión de salida	5V ó 6V
Intensidad de salida	3A
Tensión de entrada	5.5V-26V
Dimensiones	43 x 17 x 7 mm
Peso	11g

Ilustración 15. UBEC regulador de voltaje

2.8. CONEXIÓN INALÁMBRICA: MÓDULO BLUETOOTH

Para conseguir una conexión inalámbrica entre el ordenador y el microcontrolador se ha optado por incorporar a este último un módulo bluetooth. Se trata del módulo *JY-MCU Arduino Bluetooth Wireless Serial Port*. Éste se conecta a través del puerto serie al microcontrolador Arduino UNO y se comunica a su vez con el ordenador de forma inalámbrica gracias a la tecnología Bluetooth. Las características se especifican en la tabla siguiente.

Tabla 5. Características del módulo Bluetooth.

Tensión de entrada	5V
Dimensiones	44 x 16 x 7 mm
Peso	7g

Ilustración 16. Módulo JY-MCU Arduino Bluetooth Wireless Serial Port.

3. ANÁLISIS DEL ROBOT

El brazo robótico, objeto de este Trabajo Fin de Grado, se ajusta a la definición de robot dada por la *Organización Internacional de Estándares* (ISO). Dicha definición enuncia lo siguiente: "Un manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas".

Además podemos añadir, para completar dicha definición, que se trata de un robot servocontrolado, polivalente, capaz de posicionar y orientar piezas, útiles o dispositivos especiales. Tiene la forma de un brazo terminado en una muñeca. Su unidad de control incluye un dispositivo de memoria.

Según la Asociación Francesa de Robótica Industrial (AFRI) podemos clasificar el robot como Tipo A, manipulador con control manual o telemando, cuando es teleoperado con el Gamepad. Y gracias a la grabación de movimientos, estaríamos frente a un robot de Tipo C, robot programable con trayectoria continua o punto a punto. Carece de conocimientos sobre su entorno.

Tabla 6. Clasificación de los robots según la AFRI

Tipo A	Manipulador con control manual o telemando.
Tipo B	Manipulador automático con ciclos preajustados; regulación mediante fines de carrera o topes; control por PLC; accionamiento neumático, eléctrico o hidráulico.
Tipo C	Robot programable con trayectoria continua o punto a punto. Carece de conocimientos sobre su entorno.
Tipo D	Robot capaz de adquirir datos de su entorno, readaptando su tarea en función de éstos.

El brazo robótico posee 6 grados de libertad. A continuación se muestra un esquema con los seis ejes de giro que posee el brazo (*ilustración 17*). Se puede observar que los ejes q1, q4 y q6 son giros respecto a la vertical mientras que q2, q3 y q5 son giros respecto de la horizontal.

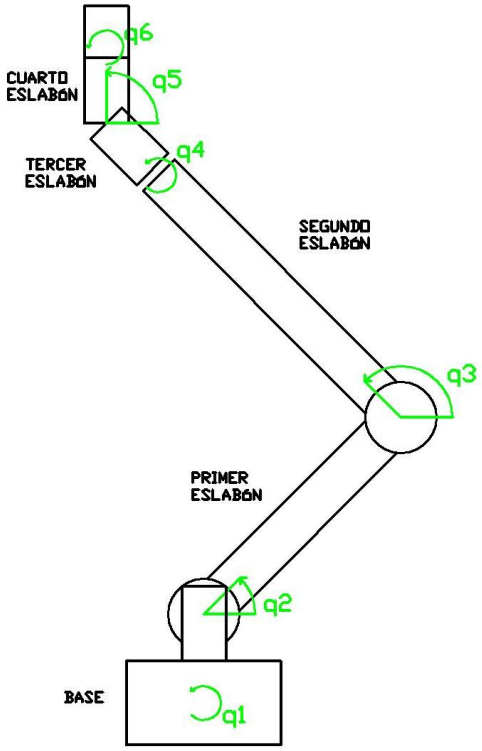


Ilustración 17. Esquema con la ubicación de los ejes del robot.

Una vez indicados los ejes de giro del robot podemos describir el área de trabajo del mismo. Ésta queda contenida en un plano delimitado por el ángulo de giro de la base (q1). Se puede observar el área de trabajo del robot en la siguiente ilustración.

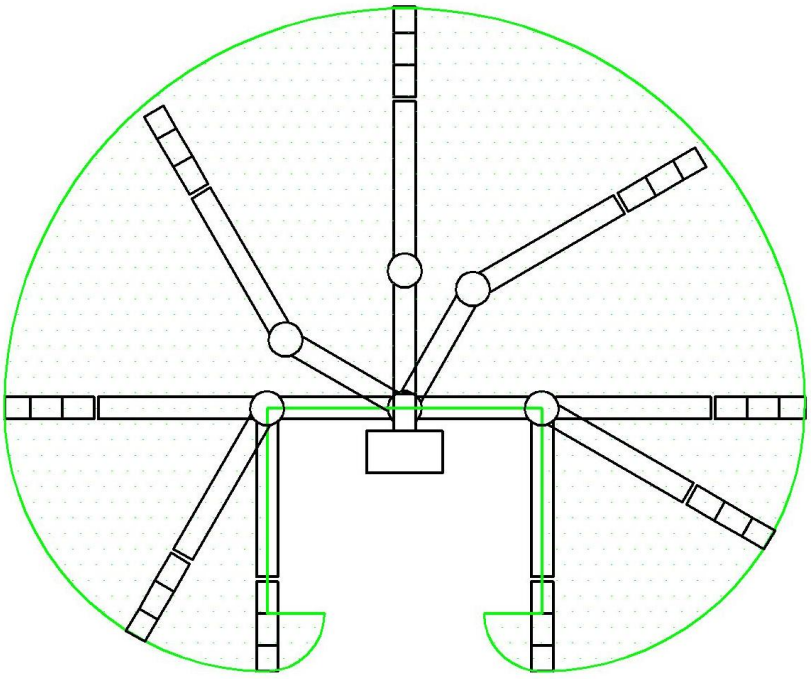


Ilustración 18. Área de trabajo del robot en un plano

3.1. DISEÑO

El diseño del brazo robótico forma parte del Proyecto Fin de Carrera de Antonio Castro Gómez. En este Trabajo Fin de Grado que nos ocupa, se ha utilizado dicho diseño con el consentimiento expreso del autor. Se ha intentado dar una continuidad a ese proyecto realizando el control y la simulación de dicho robot.

Se han tomado los archivos de los diseños en formato del programa OpenSCAD (software libre de diseño 3D), que han sido convertidos a formato STL e introducidos en las impresoras 3D. Así, mediante el software ReplicatorG de dichas impresoras se ha procedido a la impresión de las piezas con plástico ABS.

A continuación se muestran dos imágenes del robot completo y un detalle de la muñeca.

Ilustración 19. Fotografía del brazo robótico completo

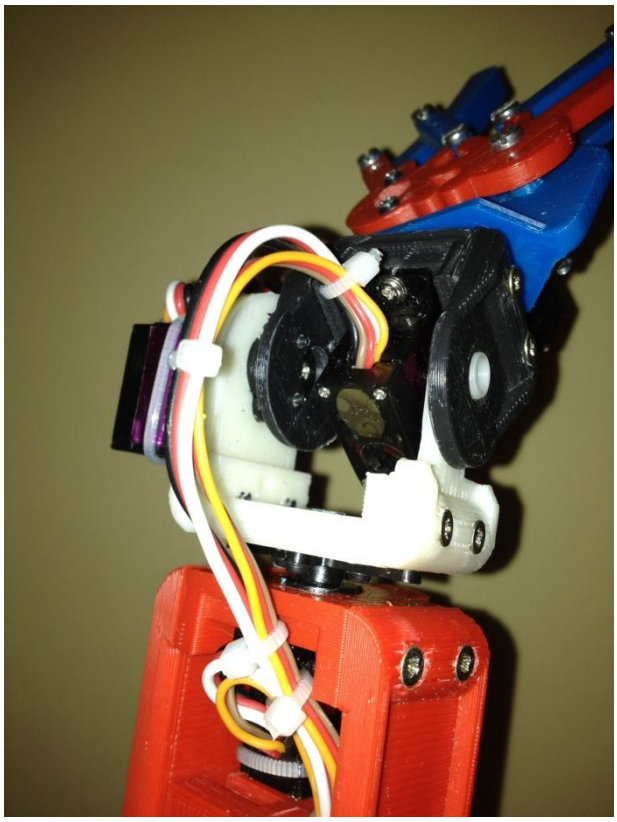


Ilustración 20. Fotografía de la muñeca del brazo robótico

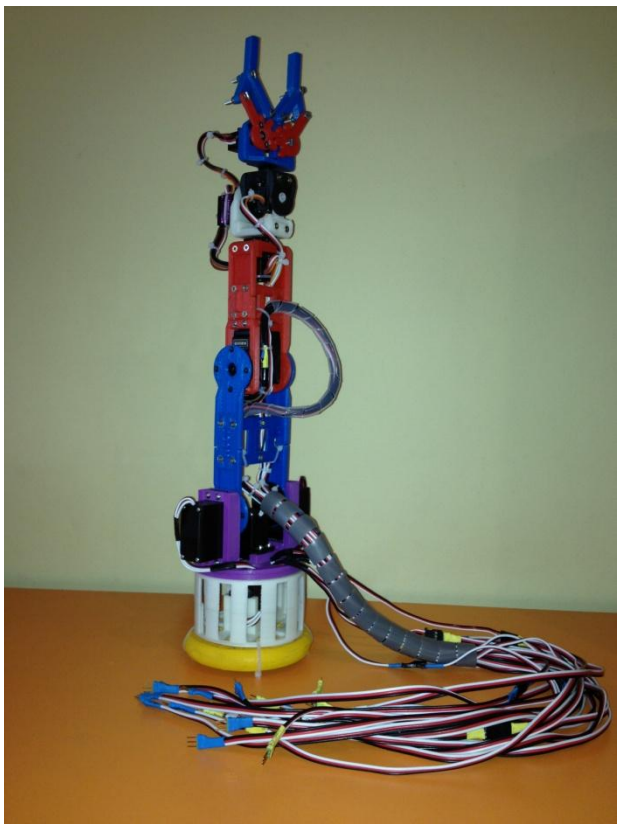


Ilustración 21. Fotografía del brazo robótico completo en posición vertical

3.2. MORFOLOGÍA

El robot está formado por los siguientes elementos: estructura mecánica, sistema de accionamiento, sistema de control y elementos terminales.

- **Estructura mecánica**

Está formado por una serie de eslabones unidos mediante articulaciones que permiten un movimiento relatico entre cada dos eslabones consecutivos. Dichos eslabones, de plástico ABS, están fabricados utilizando una impresora 3D.

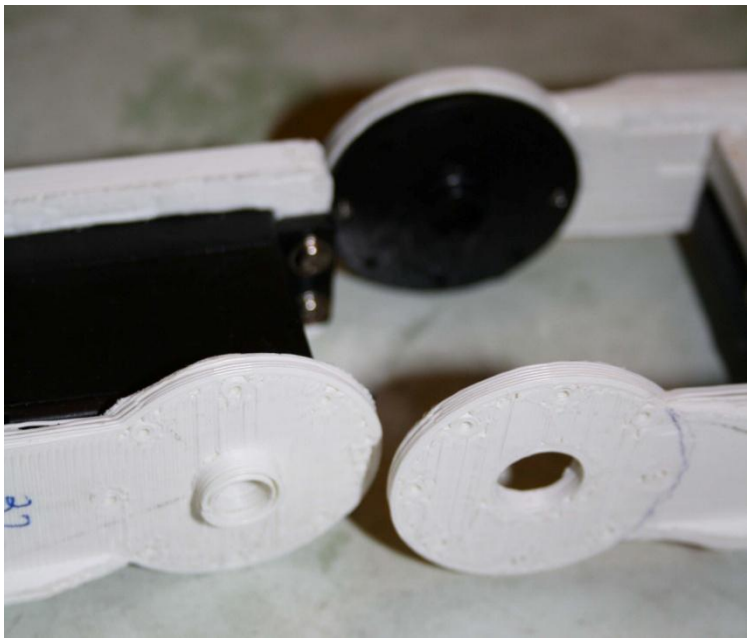


Ilustración 22. Imagen del sistema de unión entre dos de los eslabones.

La mecánica guarda cierta similitud con la anatomía del brazo humano. El movimiento entre las articulaciones es de rotación. Éste movimiento está limitado a los 180° de rotación de los servomotores utilizados.

Ilustración 23. Movimiento de rotación de las articulaciones

El movimiento de rotación de 180° era demasiado limitado para la base restringiendo la zona de acción del robot considerablemente. Por este motico, gracias a la utilización de dos servomotores conectados, se ha conseguido que la base pueda rotar 360°.

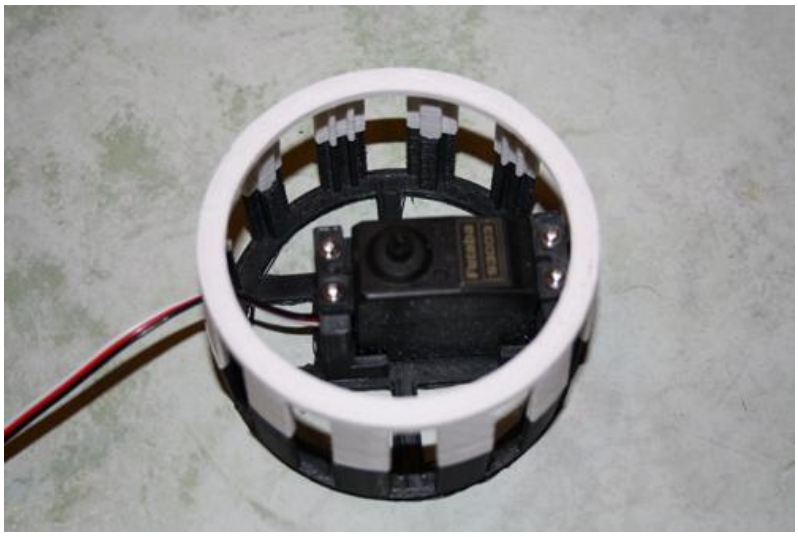


Ilustración 24. Servomotor inferior de la base del robot.

Ilustración 25. Servomotor superior de la base del robot.

Posee seis grados de libertad (GDL). Dentro de las configuraciones más frecuentes en robots industriales podemos clasificarlo como robot angular o antropomórfico.

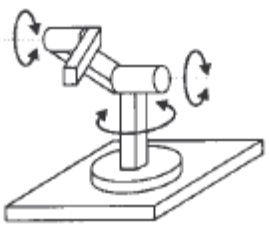


Ilustración 26. Configuración angular o antropomórfica

- **Sistema de accionamiento**

El accionamiento es directo (*Direct Drive* DD), ya que el eje del actuador se conecta directamente a la articulación, sin la utilización de un reductor intermedio. Gracias a esto conseguimos un posicionamiento rápido y preciso además de una simplificación del sistema mecánico al eliminarse el reductor.

Los actuadores son servomotores que emplean energía eléctrica (*Véase el apartado 2.2. Servomotores*).

- **Sistema de control**

El sistema de control está formado por un microcontrolador Arduino Uno (*Véase el apartado 2.3. Microcontrolador*). Éste, recibe a través del puerto serie el mensaje enviado por el programa de simulación y establece el comportamiento correspondiente de los servomotores.

- **Elementos terminales**

Como elemento terminal se ha montado una pinza accionada por un mini-servomotor. El diseño ha sido realizado por Edgar Simo. Se han obtenido los archivos de dicho diseño en formato STL a través del portal de internet Thingiverse (www.thingiverse.com). Posteriormente se imprimieron las piezas y se montó la pinza. A continuación mostramos una imagen de la pinza montada.

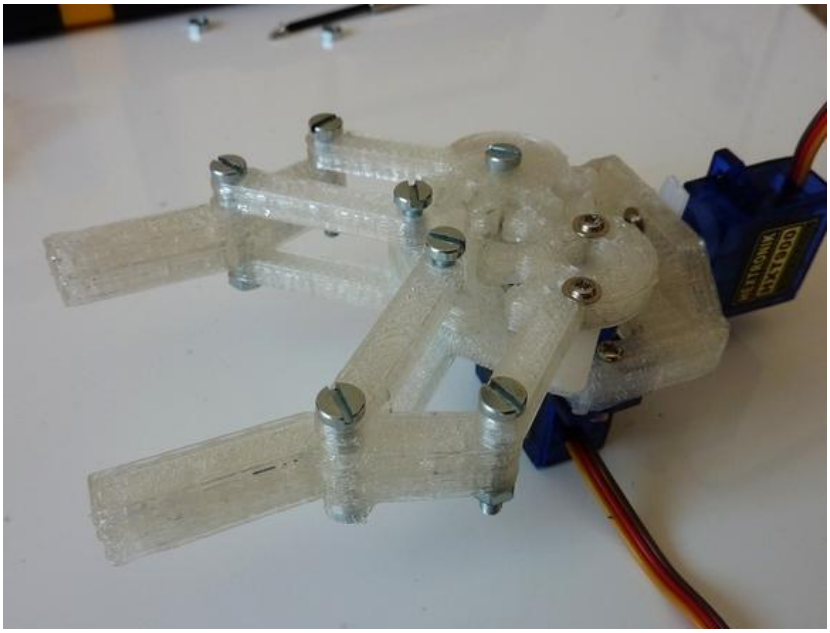


Ilustración 27. Pinza imprimible utilizada como herramienta del robot.

3.3. CINEMÁTICA

En este apartado analizaremos el movimiento del robot con respecto a un sistema de referencia situado en la base. Obtendremos una descripción analítica del movimiento espacial y, en particular, de la posición y orientación del extremo final del robot.

Tenemos dos problemas a resolver en cuanto a la cinemática del brazo robótico:

- **Cinemática directa:** determinar la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas de referencia, conocidos los valores de las articulaciones.
- **Cinemática inversa:** determinar la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas.

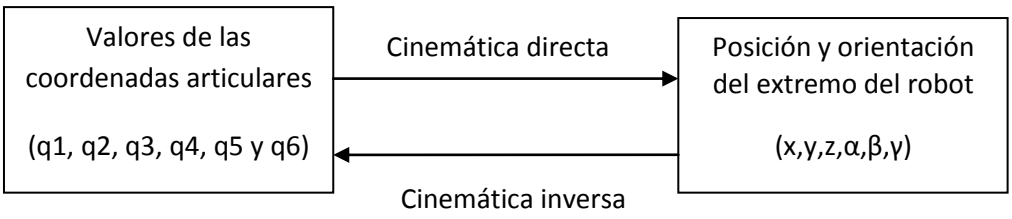


Ilustración 28. Diagrama de relación entre cinemática inversa y directa.

Para solucionar el primer problema se utilizará el Algoritmo de Denavit-Hartenberg. De esta forma, se obtiene la posición del extremo del robot a partir de los valores de los ángulos del mismo.

Para solucionar el problema de la cinemática inversa se ha optado por el método de la matriz de transformación homogénea. Así, se puede determinar los diferentes valores de los ángulos de los ejes del robot para conseguir posicionar su extremo en un punto del espacio establecido por el usuario.

- ## Cinemática directa

Como se ha explicado anteriormente, la cinemática directa consiste en obtener la posición del robot conociendo los valores de los diferentes ángulos de los ejes del mismo. Para conseguir dicho objetivo se ha utilizado el algoritmo de Denavit-Hartenberg (D-H).

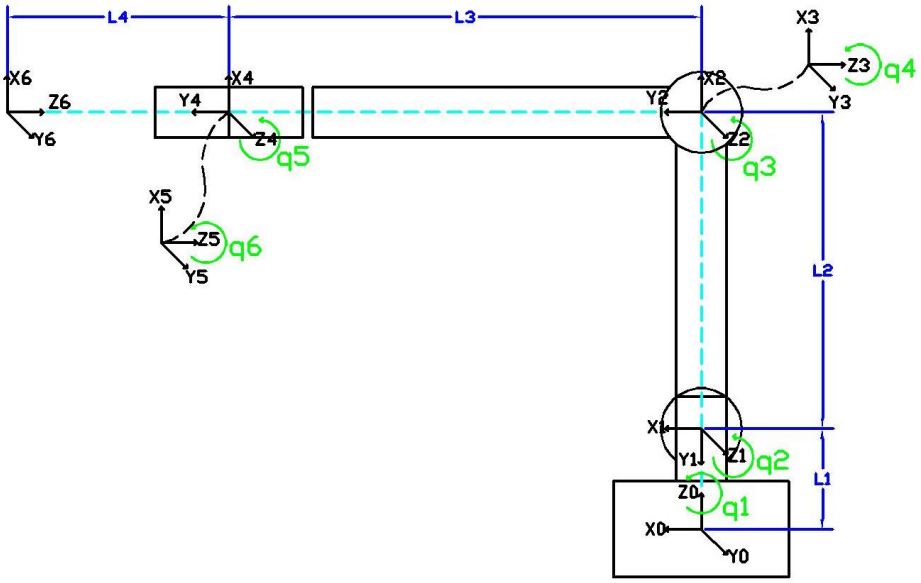


Ilustración 29. Diagrama con los sistemas de referencia, eslabones y ejes del robot

A continuación se muestra un diagrama esquemático de los diferentes sistemas de referencia y los ejes del brazo robótico además de la distancia entre eslabones.

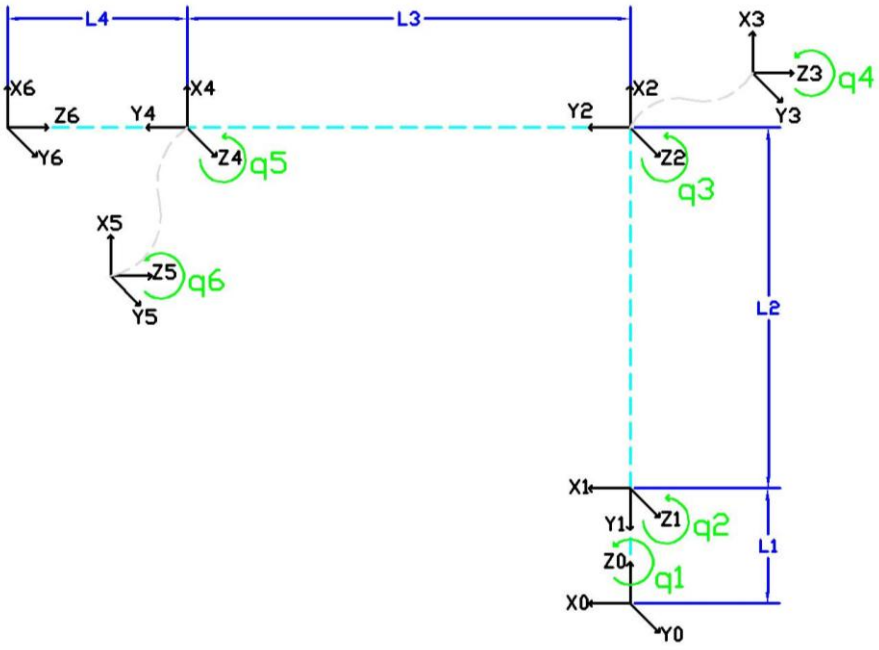


Ilustración 30. Diagrama con los sistemas de referencia y los ejes del robot

Siguiendo los pasos marcados por el Algoritmo de Denavit-Hartenberg se ha calculado los parámetros correspondientes y se disponen en la tabla siguiente.

Tabla 7. Parámetros Denavit-Hartenberg

Articulación	θ	d	a	α
1	q1	L1	0	90
2	q2-90	0	L2	0
3	q3	0	0	-90
4	q4	-L3	0	90
5	q5	0	0	-90
6	q6	-L4	0	0

Para la obtención de los parámetros se ha tenido en cuenta que:

- El parámetro θ_i es el ángulo que hay que girar sobre el eje z_{i-1} para que x_{i-1} y x_i queden paralelos.
- El parámetro d_i es la distancia sobre el eje z_{i-1} que hay que desplazar el sistema i-1 para que x_{i-1} y x_i queden alineados.
- El parámetro a_i es la distancia sobre el eje x_i que hay que desplazar el sistema i-1 para que su origen coincida con el sistema i.
- El parámetro α_i es el ángulo que hay que girar sobre x_i para que el sistema i-1 coincida con el sistema i.

Una vez hallados los parámetros se obtienen las matrices de transformación como expresa la siguiente ecuación:

Ecuación 1. Obtención de la Matriz de transformación D-H

$$^{i-1}A_{i-1} = RotZ(\theta_i) \cdot Tras([0,0,d_i]) \cdot Tras([a_i,0,0]) \cdot RotX(\alpha_i)$$

Para mayor simplicidad en la notación, el cos(θi) está representado por C_i y el sen(θi) está representado por S_i. La expresión anterior (ecuación 1) se puede convertir en una única matriz quedando representada de la siguiente forma:

Ecuación 2. Matriz de transformación D-H

$$^{i-1}A_{i-1} = \begin{pmatrix} C\theta_i & -C\propto_i S\theta_i & S\propto_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\propto_i C\theta_i & -S\propto_i C\theta_i & a_i S\theta_i \\ 0 & S\propto_i & C\propto_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ahora, una vez calculador los parámetros de Denavit-Hartenberg, podemos calcular las matrices de transformación de un sistema a otro. A continuación se muestran los resultados obtenidos:

Ecuación 3. Resultados obtenidos de la Matrices de transformación ($^{i-1}A_i$).

$$^0A_1 = \begin{pmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad ^1A_2 = \begin{pmatrix} S_2 & -C_2 & 0 & L_2C_2 \\ C_2 & S_2 & 0 & L_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^2A_3 = \begin{pmatrix} C_3 & 0 & -S_3 & 0 \\ S_3 & 0 & C_3 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad ^3A_4 = \begin{pmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & -L_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^4A_5 = \begin{pmatrix} C_5 & 0 & -S_5 & 0 \\ S_5 & 0 & C_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad ^5A_6 = \begin{pmatrix} C_6 & -S_6 & 0 & -L_4C_6 \\ S_6 & C_6 & 0 & -L_4S_6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para obtener la matriz de transformación (T) entre la base y el extremo del robot hay que multiplicar por las diferentes matrices de transformación entre el sistema 0 y el sistema 6. Se procedería de la siguiente forma:

Ecuación 4. Producto para la obtención de la matriz de transformación T.

$$T = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6 = {}^0A_6$$

Como se puede observar en la ecuación 5, la matriz T resultante, está formada por una submatriz 3x3 dedicada a la orientación ($\vec{n}$, $\vec{o}$, $\vec{a}$) y un vector 3x1 dedicado al posicionamiento ($\vec{p}$).

Ecuación 5. Matriz de Transformación T

$$T = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & w \end{pmatrix}$$

Para la obtención de los valores de la posición (x, y, z) del extremo del robot nos interesa el exclusivamente la última columna de la matriz de transformación (T). Así, para obtenerlo, realizaremos la siguiente operación:

$$^0A_6 \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = (x, y, z, w) \rightarrow \vec{p} = (x, y, z)$$

Cinemática Inversa

El objetivo del problema cinemático inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del robot q = $[q_1, q_2, q_3, q_4, q_5, q_6]^T$ para que su extremo se posicione y oriente según una determinada localización espacial.

Para resolver este problema encontraremos una solución cerrada que tendrá la siguiente forma:

Ecuación 6. Forma de las soluciones de la cinemática inversa.

$$q_k = f_k(x, y, z, \propto, \beta, \gamma) \quad para \quad k = [0,6]$$

Para el caso que nos ocupa, la solución del problema cinemático inverso no es única, existiendo diferentes soluciones que posicionan y orientan el extremo del robot del mismo modo. No obstante, podemos establecer restricciones para la solución obtenida.

A pesar de las dificultades que se nos plantean se da la circunstancia de que los tres últimos grados de libertad, dedicados fundamentalmente a orientar el extremo del robot, corresponden a giros sobre ejes que se cortan en un punto. De esta forma, plantearemos el problema cinemático inverso para el posicionamiento:

- **Posicionamiento:** involucra a los valores de los tres primeros ejes (q_1, q_2, q_3) y depende del punto del espacio objetivo (x, y, z).

Ecuación 7. Forma de las soluciones de posicionamiento.

$$q_k = f_k(x, y, z) \quad para \quad k = [0,3]$$

Cinemática inversa para el posicionamiento

Realizamos la resolución del problema cinemático inverso por métodos geométricos. El procedimiento en sí se basa en encontrar suficiente número de relaciones geométricas en las que intervendrán las coordenadas del extremo del robot (x, y, z), sus coordenadas articulares (q1, q2, q3) y las dimensiones físicas de los eslabones (L1, L2, L3).

Aplicamos este método a los primeros 3 GDL de rotación de nuestro robot. Los datos de partida son las coordenadas (px,py,pz) referidas al sistema de referencia en las que se quiere posicionar su extremo.

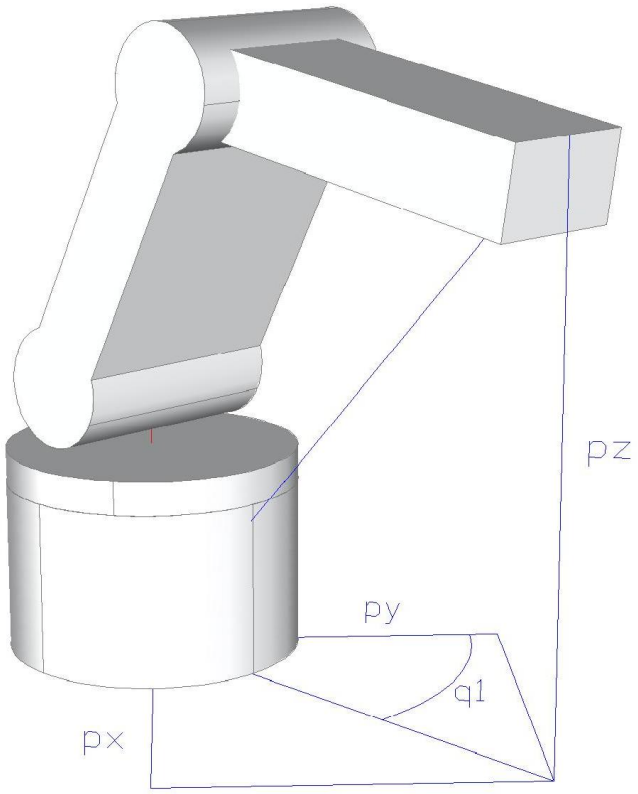


Ilustración 31. Esquemático de los 3 primeros GDL del robot.

El valor de q1 se obtiene de manera inmediata:

$$q_1 = arctg\left(\frac{p_y}{p_x}\right)$$

Considerando ahora únicamente los eslabones 2 y 3 que están situados en un plano y utilizando el teorema del coseno, se tendrá:

$$\left.\begin{aligned} r^2 &= {p_x}^2 + {p_y}^2 \\ r^2 + {p_z}^2 &= {l_2}^2 + {l_3}^2 + 2\, l_2\, l_3 \cos(q_2) \end{aligned}\right\}$$

$$\cos(q_3) = \frac{{p_x}^2 + {p_y}^2 + {p_z}^2 - {l_2}^2 - {l_3}^2}{2\, l_2\, l_3}$$

Esta expresión permite obtener q3 en función del vector posición del extremo ($\vec{p}$). No obstante por motivos de ventajas computacionales, es más conveniente utilizar la expresión de la arcotangente en lugar del ascoseno.

Puesto que

$$\cos(q_3) = \pm\sqrt{1 - cos^2(q_3)}$$

Se tendrá que

$$q_3 = arctg\left(\frac{\pm\sqrt{1-cos^2(q_3)}}{\cos(q_3)}\right)$$

Con

$$\cos(q_3) = \frac{{p_x}^2 + {p_y}^2 + {p_z}^2 - {l_2}^2 - {l_3}^2}{2\, l_2\, l_3}$$

Como se ve, existen dos posibles soluciones par q3 según tome el signo positivo o negativo en la raíz. Éstas corresponden a las configuraciones de codo arriba y codo abajo del robot.

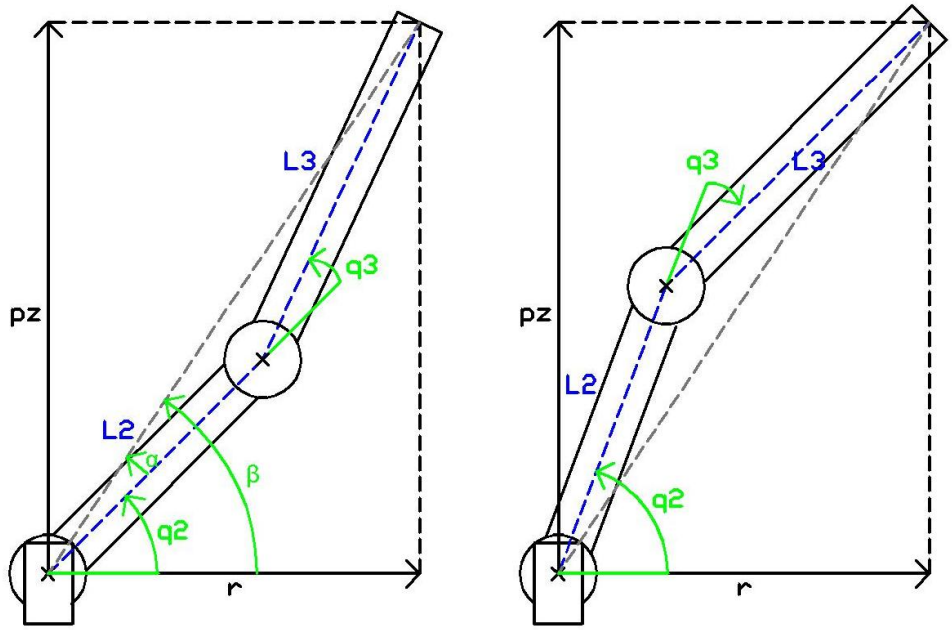


Ilustración 32. Configuraciones codo arriba y abajo

El cálculo de q2 se hace a partir de la diferencia entre β y α:

$$q_2 = \beta - \alpha$$

Siendo:

$$\beta = arctg\left(\frac{p_z}{r}\right) = arctg\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right)$$

$$\propto = arctg\left(\frac{l_3\, sen(q_3)}{l_2 + l_3 cos(q_3)}\right)$$

Luego, finalmente

$$q_2 = arctg\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right) - arctg\left(\frac{l_3\, sen(q_3)}{l_2 + l_3 cos(q_3)}\right)$$

De nuevo los dos posibles valores según la elección del signo dan lugar a dos valores diferentes de q2 correspondientes a las configuraciones codo arriba y abajo.

Las expresiones que resuelven el problema cinemático inverso para los tres primeros grados de libertada del robot son:

$$q_1 = arctg\left(\frac{p_y}{p_x}\right)$$

$$q_2 = arctg\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right) - arctg\left(\frac{l_3\, sen(q_3)}{l_2 + l_3 cos(q_3)}\right)$$

$$q_3 = arctg\left(\frac{\pm\sqrt{1 - cos^2(q_3)}}{\cos(q_3)}\right)$$

4. CONTROL Y SIMULACIÓN

Para el control y la simulación del robot se ha implementado un programa en lenguaje C++ y OpenGL. En este apartado describiremos el control realizado a través de un gamepad y la simulación que representa el robot en la pantalla del ordenador.

Podemos describir el sistema como un bucle cerrado en el que el usuario ejecuta unas acciones de control sobre el gamepad, el programa las recibe, representa la nueva posición y orientación del robot y la muestra al usuario.

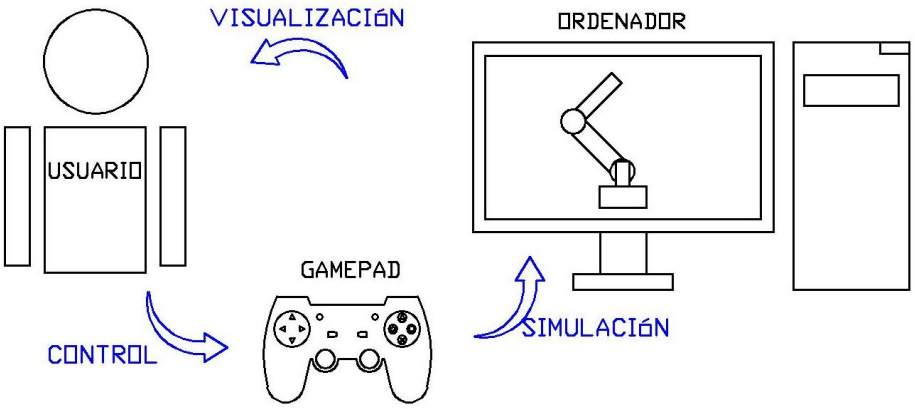


Ilustración 33. Esquema de interacción entre el usuario y el simulador

El esquema completo de interacción entre dispositivos está formado por: el gamepad, el ordenador, el microcontrolador y, finalmente el robot. A continuación podemos ver de forma esquemática dicha interacción.

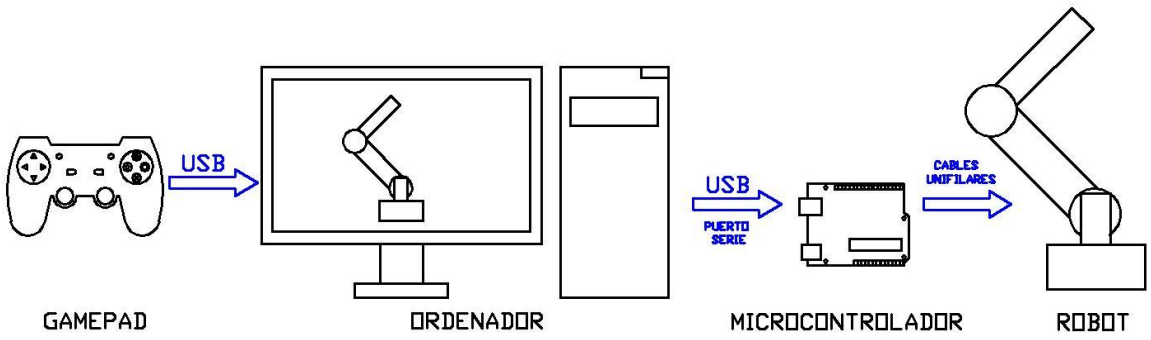


Ilustración 34. Esquema de interacción (Usuario-Ordenador-Robot)

Una vez descrito el sistema podemos definir los pasos que sigue el proceso de control y simulación completo:

1. El usuario da una orden a través del gamepad.
2. El ordenador recibe la orden.
3. Se modifican los valores de los ángulos de los ejes correspondientes.
4. Se muestra la nueva posición y orientación por pantalla en el simulador.
5. Se mandan los nuevos valores de los ejes al microcontrolador.
6. El microcontrolador manda las señales correspondientes a los servomotores del robot.
7. El robot adopta la posición y orientación indicados por el usuario.

Existen tres modos de funcionamiento del programa de simulación y control:

1. **Teleoperación:** en este modo, el robot y el simulador reproducirán exactamente las instrucciones del usuario transmitidas a través del gamepad.
2. **Grabación de movimientos:** en este modo, el usuario puede realizar la grabación de movimientos simulados y reproducirlos en el robot real. Así, el procedimiento para este modo es el siguiente:
   - El usuario indica el inicio de la grabación pulsando un botón.
   - Una vez iniciada la grabación el usuario realiza los movimientos que desee visualizándolos en el simulador. Éstos movimientos sólo se visualizarán en el simulador no se enviarán al robot.
   - El usuario indica la detención de la grabación cuando crea conveniente.
   - Por último, el usuario podrá iniciar la reproducción de los movimientos grabados pulsando un botón. De esta forma, tanto el robot como el simulador reproducen los movimientos preestablecidos.
3. **Posicionamiento:** en este modo el usuario introduce un punto del espacio y el robot adopta dicha posición. Para llevar a cabo este proceso se calcula la cinemática inversa del robot (*ver apartado 3.3. Cinemática*).

Seguidamente nos centraremos de forma más detallada en el control y la simulación.

4.1. CONTROL

El control se realiza mediante un gamepad (*ver apartado 2.5. Gamepad*). Éste proporcionará el control al usuario sobre los movimientos de los diferentes ejes del robot. A continuación recopilamos en una tabla con las funciones asociadas a los diferentes botones del gamepad.

Tabla 8. Funcionalidad de los botones del Gamepad

Numero del Botón	Nombre del botón	Función
0	Start	Parar la grabación de movimientos.
1	Pulsación Joystick 1	Posición inicial de los ejes q1 y q2.
2	Pulsación Joystick 2	Posición inicial de los ejes q3 y q4.
3	Select	Selección de la memoria de grabación.
4	Cruceta arriba	Aumento del ángulo q4.
5	Cruceta derecha	Aumento del ángulo q5.
6	Cruceta abajo	Decremento del ángulo q4.
7	Cruceta izquierda	Decremento del ángulo q6.
8	L2	Reproducción de la grabación 1
9	R2	Reproducción de la grabación 2
10	L1	Grabación de movimientos en memoria 1
11	R1	Grabación de movimientos en memoria 2
12	Triángulo	Mientras está pulsado abre la pinza del extremo.

Además de los botones, utilizaremos los dos joysticks frontales del gamepad para el manejo de los ejes principales (q1, q2, q3 y q4). A continuación mostramos una tabla con la asignación de los ejes.

Tabla 9. Funcionalidad de los josysticks del gamepad

Joystick	Eje	Función
Izquierdo	EJE 0	Aumenta o disminuye el ángulo q1
	EJE 1	Aumenta o disminuye el ángulo q2
Derecho	EJE 2	Aumenta o disminuye el ángulo q3
	EJE 3	Aumenta o disminuye el ángulo q4

4.2. SIMULACIÓN

Se ha diseñado e implementado un simulador en 3D en OpenGL y C++. Éste posibilita la visualización de la posición y orientación del brazo robótico a través de la pantalla de un ordenador. Así, el usuario puede operar el robot visualizándolo en el simulador sin necesidad de estar viendo el robot real.

El simulador consta de tres partes:

- La representación gráfica en tres dimensiones del posicionamiento y orientación del robot.
- Los datos correspondientes a los ángulos del robot. Estos son representados en la parte superior derecha del simulador.
- Los datos de la posición del extremo del robot (x, y, z). Para obtener dicha posición el simulador resuelve el problema cinemático directo de la forma comentada anteriormente (*ver apartado "3.3. Cinemática"*). Los datos son representados en la parte derecha del simulador a continuación de los valores de los ejes.

En la ilustración siguiente podemos ver las tres partes mencionadas. La representación del robot se encuentra en el centro y los datos se disponen en la parte derecha de la pantalla.

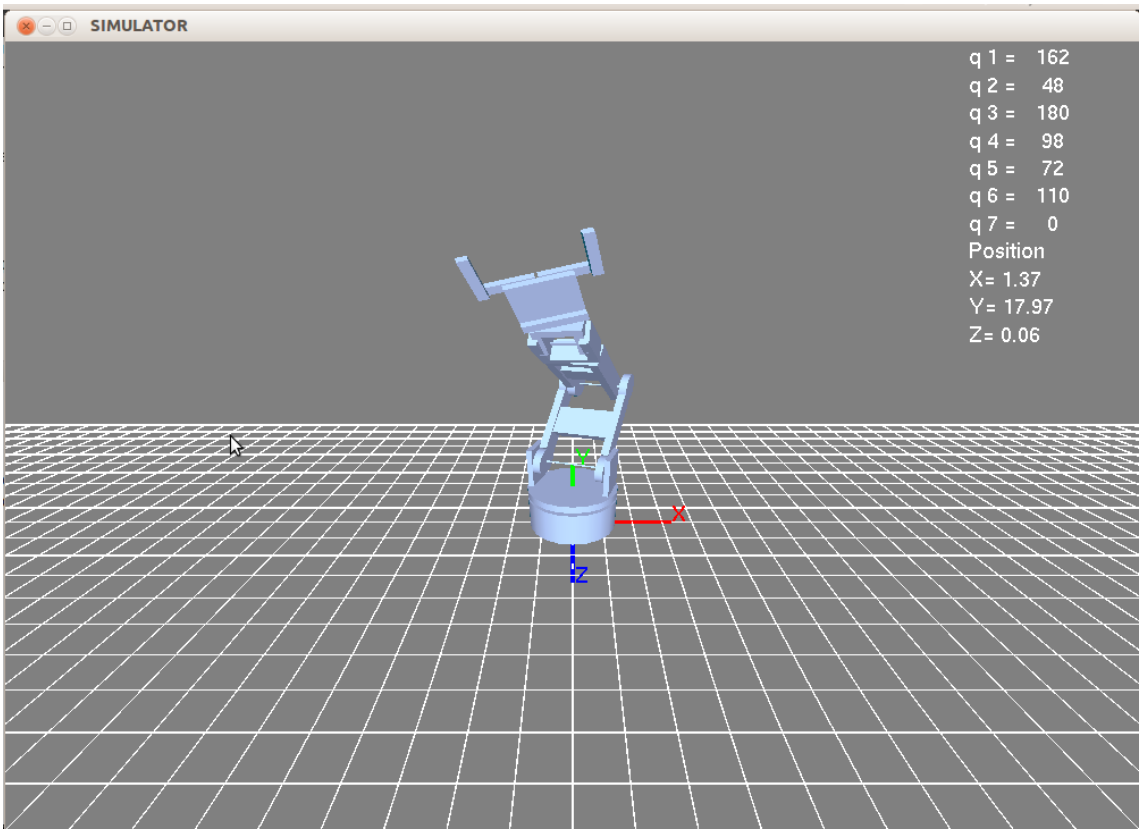


Ilustración 35. Captura de pantalla del simulador.

4.3. IMPLEMENTACIÓN DEL PROGRAMA

Para la implementación del software del simulador se ha utilizado el lenguaje C++ y las librerías de OpenGL. El entorno de desarrollo utilizado ha sido el QTCreator. Es un programa gratuito distribuido por la empresa Nokia. A continuación mostramos una captura de pantalla en la que se puede apreciar el entorno gráfico que utiliza dicho programa.

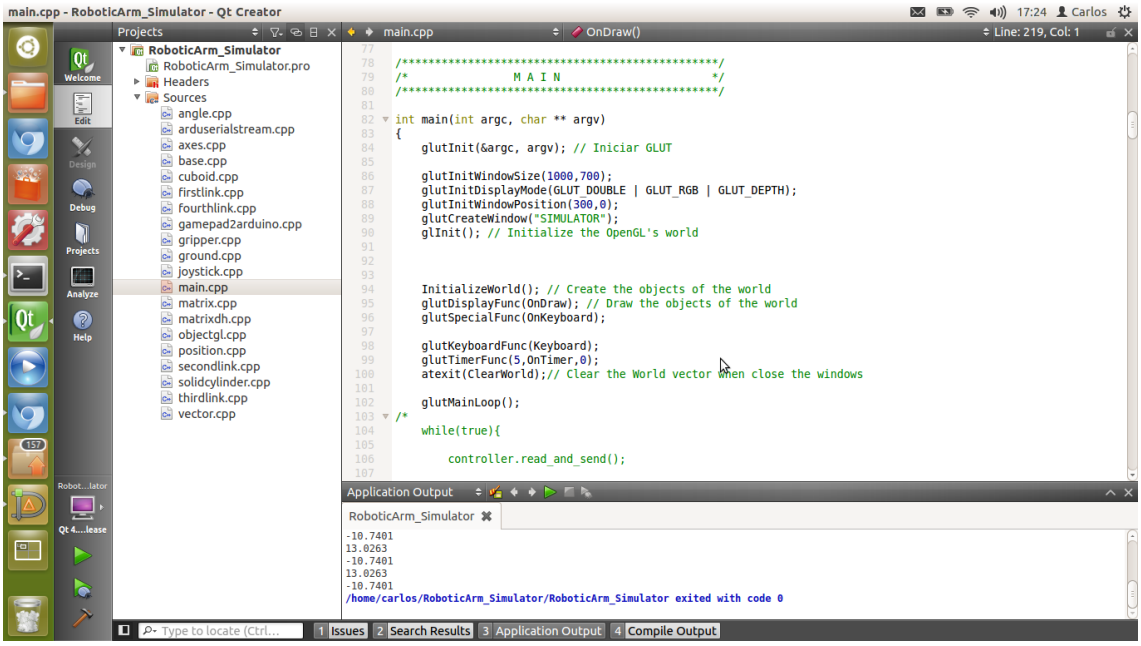


Ilustración 36. Entorno de desarrollo QTCreator.

Para definir el proceso de implementación del programa describiremos brevemente las características del lenguaje C++ y de la librerías de OpenGL. Además, mostraremos las clases implementadas y los procesos que realiza el programa.

▪ LENGUAJE DE PROGRAMACIÓN C++

Para la implementación del simulador se ha utilizado el lenguaje C++. Es una ampliación del Lenguaje C con la posibilidad de manipular objetos. Con éste lenguaje se ha creado un programa orientado a objetos. Esto ha permitido obtener un alto nivel de encapsulamiento gracias a las diferentes clases utilizadas.

▪ OPENGL (GLU, GLUT Y GLUI)

Para el entorno gráfico en tres dimensiones, se han utilizado las librerías de OpenGL (Open Graphics Library). Se trata una especificación estándar que define una interfaz programación de aplicaciones (API) multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos.

Entre las variadas funciones de OpenGL cabe destacar las que se han utilizado para dotar de movimiento a la simulación el robot. Se trata de las dos instrucciones de transformación básicas:

- *glRotatefv (a,x,y,z):* Realiza una rotación determinada por el valor de "a" alrededor del vector x, y, z.
- *glTranslatefv (x,y,z):* realiza una traslación determinada por los valores x, y, z.

Para la programación se han utilizado varias bibliotecas internas y externas que añaden características no disponibles en el propio OpenGL. Algunas de ellas son:

- **GL (Graphics Library):** En ella se encuentran las funciones básicas de OpenGL. Sus funciones empiezan por el prefijo gl.
- **GLU (Graphics Library Utility):** Ofrece funciones de dibujo de alto nivel basadas en primitivas de OpenGL. Las funciones de GLU se reconocen fácilmente pues todas empiezan con el prefijo glu.
- **GLUT (Graphics Library Utility Toolkit):** Es la encargada de proveer un interface común de programación para las ventanas, el uso del ratón, los eventos de teclado, etc. Gracias a ella cuando se desarrolla una aplicación OpenGL no hay que tener en cuenta detalles sobre el sistema utilizado.
- **GLUI (Graphics Library User Interface):** Interfaz de usuario basada en GLUT; proporciona elementos de control tales como botones, cajas de selección y spinners. Es independiente del sistema operativo, sustentándose en GLUT para manejar los elementos dependientes del sistema.

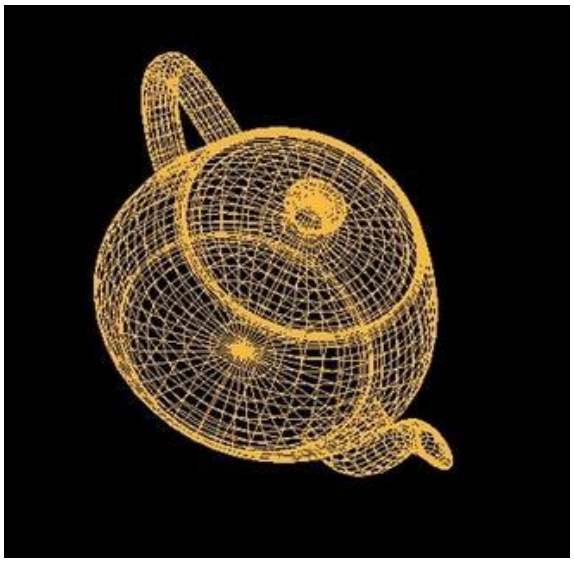


Ilustración 37. Ejemplo de una escena desarrollada con GLU.

CLASES

Las clases que se han implementado se irán describiendo a continuación según su funcionalidad. Se muestran además los diagramas de clases en Lenguaje Unificado de Modelado (UML) y las tablas con los nombres de cada clase acompañadas de una breve descripción.

Para un mejor análisis, hemos definido las funcionalidades de las clases. Así se establecen 3 grupos distintos según la funcionalidad:

1. Dibujado
2. Cálculo y visualización de parámetros
3. Comunicación entre el Gamepad y el microcontrolador

1. CLASES ENCARGADAS DEL DIBUJADO

Las clases que se centran fundamentalmente en el dibujado son:

Tabla 10. Descripción de las clases cuya función es el dibujado.

Nombre de la clase	Descripción
ObjectGL	Clase abstracta correspondiente a todos los objetos dibujables en OpenGL. Es la clase padre de otros objetos más específicos.
Cuboid	Clase que, dados los parámetros correspondientes, dibuja un tetraedro.
SolidCylinder	Clase que, dados los parámetros correspondientes, dibuja un Cilindro.
Base	Clase que, a parir de los parámetros correspondientes, dibuja la base cilíndrica del brazo robótico. Utiliza objetos de las clases Cuboid y SolidCylinder.
FirstLink	Clase que, a parir de los parámetros correspondientes, dibuja el primer eslabón del brazo robótico. Utiliza objetos de las clases Cuboid y SolidCylinder.
SecondLink	Clase que, a parir de los parámetros correspondientes, dibuja el segundo eslabón del brazo robótico. Utiliza objetos de las clases Cuboid y SolidCylinder.
ThirdLink	Clase que, a parir de los parámetros correspondientes, dibuja el tercer eslabón del brazo robótico. Utiliza objetos de las clases Cuboid y SolidCylinder.
FourthLink	Clase que, a parir de los parámetros correspondientes, dibuja el tercer eslabón del brazo robótico. Utiliza objetos de las clases Cuboid y SolidCylinder.
Gripper	Clase que, a parir de los parámetros correspondientes, dibuja la pinza situada en el extremo del brazo robótico. Utiliza objetos de las clases Cuboid y SolidCylinder.

En el diagrama UML siguiente podemos observar que todas las clases de dibujado heredan de la clase ObjectGL. Así mismo, existe también una relación de composición entre algunas clases. Esto es debido a que las más complejas, como FirstLink, SecondLink, ThirdLink, FourthLink o Gripper, están formadas por objetos de las más básicas que son Cuboid y SolidCylinder.

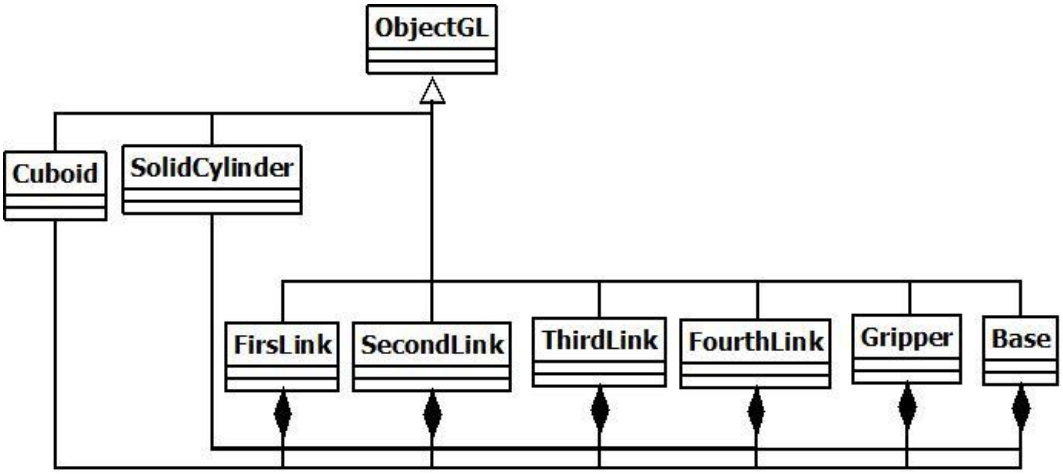


Ilustración 38. Diagrama UML de las relaciones entre las clases encargadas del dibujado.

Además de representar las relaciones que guardan las clases encargadas del dibujado, hemos realizado el diagrama UML detallado de las más relevantes. Así podemos observar en los diagramas siguientes los atributos y funciones que poseen las clases ObjectGL, Cuboid , SolidCylinder y FirstLink.

En primer lugar, mostramos el diagrama correspondiente a la clase ObjectGL. Cabe destacar que ésta es la clase padre de los objetos dibujables. Así posee los atributos que luego heredarán el resto de clases hijas como la posición (x, y, z), el nombre (name), y las dimensiones (x_size, y_size, z_size).

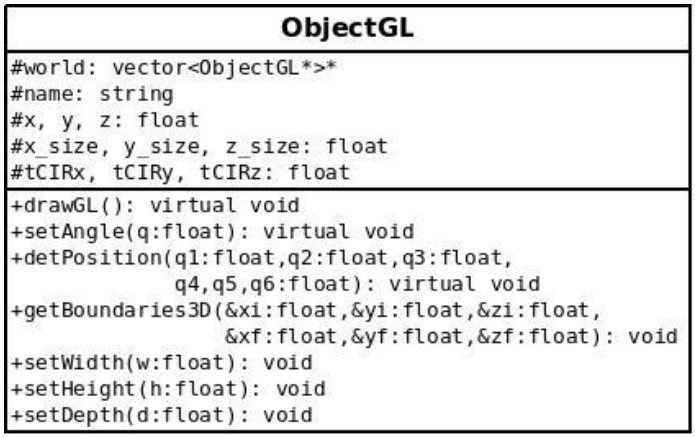


Ilustración 39. Diagrama UML de la clase ObjectGL.

Un ejemplo de una clase hija de ObjectGL es Cuboid. Esta clase posee un constructor en el que hay que indicar la posición (x, y, z), las dimensiones del ortoedro (width, height, depth) y la posible traslación del CIR si es necesaria.

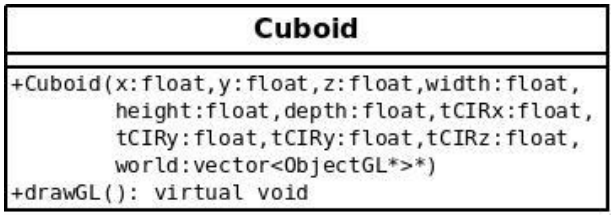


Ilustración 40. Diagrama UML de la clase Cuboid.

Para el dibujado de un cilindro sólido hemos implementado la clase SolidCylinder. Ésta dibuja dos discos a modo de caras superior e inferior y un cilindro hueco para la cara lateral.

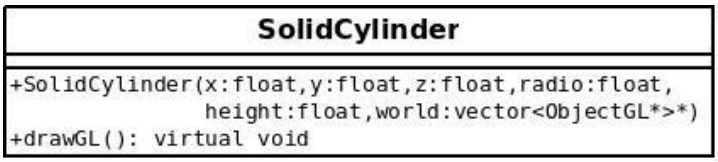


Ilustración 41. Diagrama UML de la clase SolidCylinder.

FirstLink se dedica al dibujado del primer eslabón del robot. Podemos observar, en el diagrama siguiente, los atributos que posee destinados a establecer las dimensiones: ancho (width), alto (height) , profundidad (depth), radio y el hueco entre piezas (gap).

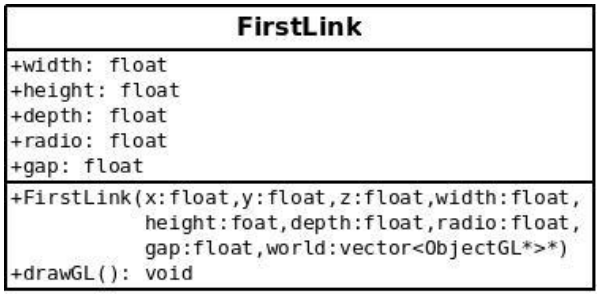


Ilustración 42. Diagrama UML de la clase FirstLink.

Se han omitido los diagramas correspondientes a las clases “Base”, “SecondLink”, “ThirdLink”, “FourthLink” y “Gripper” por ser análogas a “FirstLink”. Todas ellas se dedican al dibujado de los diferentes eslabones del robot.

2. CLASES ENCARGADAS DEL CÁLCULO Y LA VISUALIZACIÓN DE PARÁMETROS

Las clases que se centran en el cálculo y la visualización de parámetros del robot son:

Tabla 11. Definición de las clases cuya función es el cálculo y la visualización de parámetros.

Nombre de la clase	Descripción
Angle	Clase que muestra por pantalla el nombre y el valor del eje del robot.
Position	Clase que calcula y muestra por pantalla la posición (x,y,z) del extremo del robot.
Matrix	Clase que genera matrices de las dimensiones previamente indicadas. Posee funciones para introducir y mostrar los valores de la matriz. Cuenta con sobrecarga de operadores que realizan las operaciones suma y multiplicación. También se ha implementado una función que realiza la inversa de matrices de transformación homogéneas.
MatrixDH	Clase que hereda de la clase Matrix. Genera una matriz, según el algoritmo de Denavit-Hartenberg, que transforma de un sistema a otro.

Hay que mencionar que las clases Angle y Position son hijas de la clase OpenGL ya que sus valores son representados de forma gráfica en la pantalla. Mostramos a continuación los diagramas UML de las clases mencionadas:

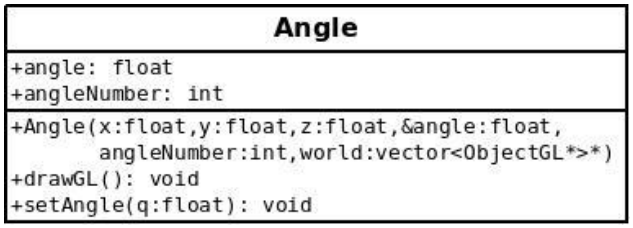


Ilustración 43. Diagrama UML de la clase Angle.

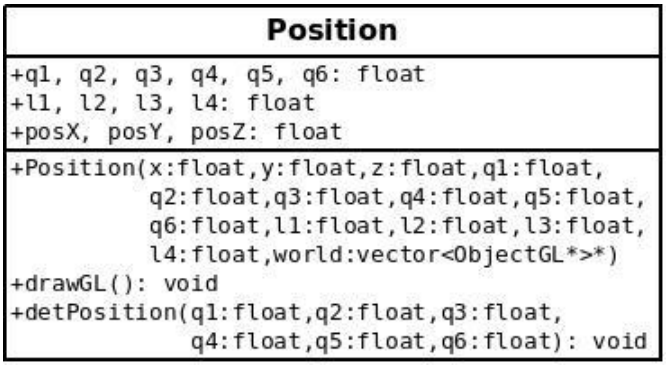


Ilustración 44. Diagrama UML de la clase Position.

Como se describe en la tabla, la clase MatrixDH es hija de Matrix. Así, además de heredar sus atributos y funciones posee su constructor propio. Éste recibe los parámetros de Denavit-Hartenberg (θ, d, a, α) y construye una matriz de la siguiente forma:

$$ {}^{i-1}A_{i-1} = \begin{pmatrix} C\theta_i & -C\propto_i S\theta_i & S\propto_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\propto_i C\theta_i & -S\propto_i C\theta_i & a_i S\theta_i \\ 0 & S\propto_i & C\propto_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

De esta manera, el cálculo de la cinemática directa, se realiza a través de los objetos de la clase MatrixDH y facilita considerablemente la resolución del problema. A continuación mostramos los diagramas UML detallados de las clases Matrix y MatrixDH:

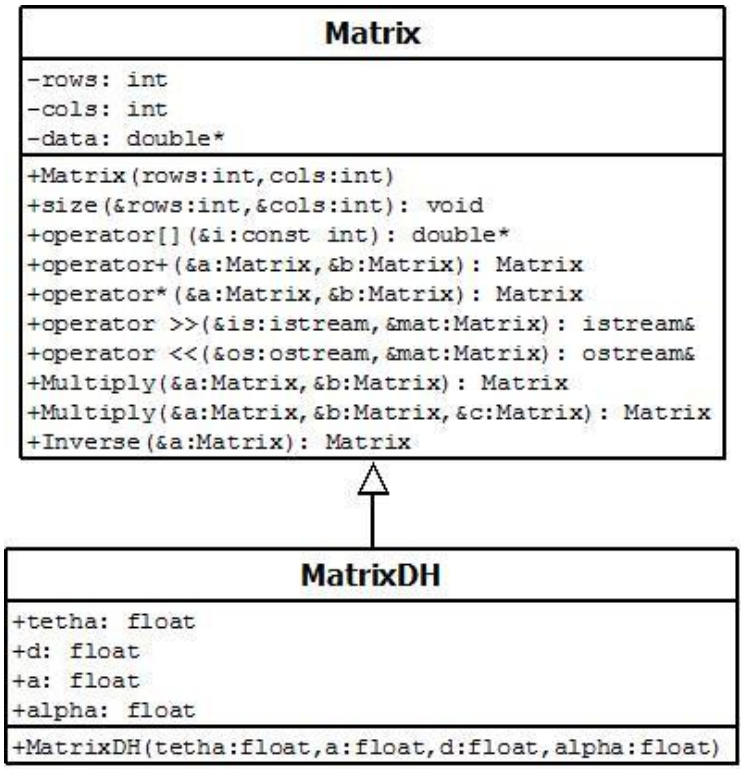


Ilustración 45. Diagrama UML de la relación entre las clases Matrix y MatrixDH.

3. CLASES ENCARGADAS DE LA COMUNICACIÓN ENTRE EL GAMEPAD Y EL ROBOT

Es necesario establecer una comunicación entre el gamepad, el ordenador y el microcontrolador que rige los movimientos del robot. Así, las clases que se centran en la comunicación son: Joystick, GamePad2Arduino y ArduSerialStream.

Describiremos las tres clases involucradas en la comunicación a través de la siguiente tabla:

Tabla 12. Descripción de las clases dedicadas a la comunicación.

Nombre de la clase	Descripción
GamePad2Arduino	Clase formada por objetos de las clases Joystick y ArduSerialStream. Comunica el gamepad con el simulador y con el microcontrolador.
Joystick	Clase que posee las funciones de comunicación con el Gamepad.
ArduSerialStream	Clase que posee las funciones de comunicación con el microcontrolador Arduino.

La relación entre estas tres clases es de agregación como se muestra en el siguiente diagrama:

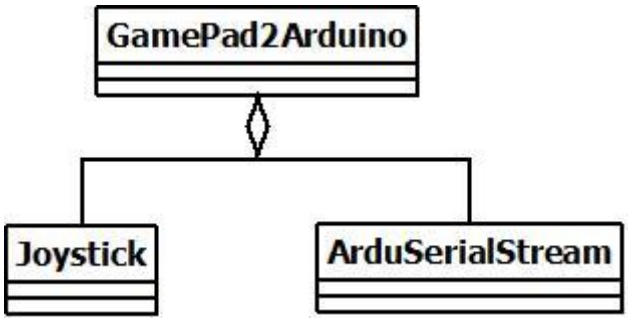


Ilustración 46. Relación entre las clases Joystick, ArduSerialStream y GamePad2Arduino

Seguidamente, exponemos los diagramas UML detallados de las clases involucradas en la comunicación.

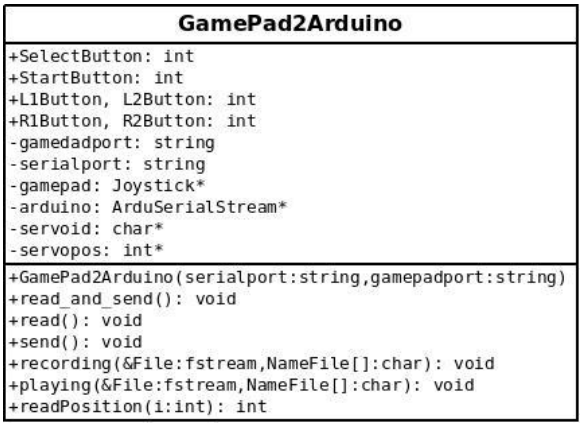


Ilustración 47. Diagrama UML de la clase GamePad2Arduino.

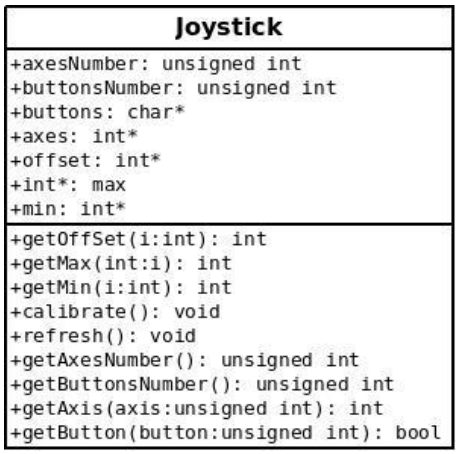


Ilustración 48. Diagrama UML de la clase Joystick.

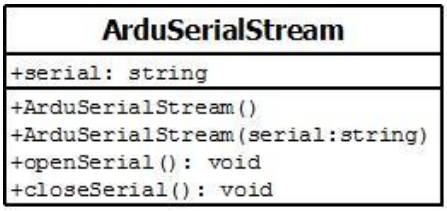


Ilustración 49. Diagrama ULM de la clase ArduSerialStream

Por un lado, la clase GamePad2Arduino está compuesta por objetos de las clases Joystick y ArduSerialStream. Así, la función GamePad2Arduino realiza diferentes funciones que comunican el Gamepad con el microcontrolador. Entre esas funciones están:

- Definir el puerto de conexión del Gamepad.
- Definir el puerto de conexión del Microcontrolador.
- Realiza una calibración previa de los joystick delanteros del mando.
- Grabar los movimientos dados por el usuario en el modo grabación.
- Reproducir los movimientos guardados en el modo reproducción.

Por otro lado, la clase Joystick establece funciones para la obtención de los valores de los ejes (*getAxis(i)*) y de los botones (*getButton(i)*) del gamepad. En definitiva, es la encargada de comunicarse con el gamepad.

La clase ArduSerialStream permite una comunicación sencilla con el microcontrolador a través del puerto serie. Establece sobrecarga de operadores para realizar una escritura fácil en el puerto serie.

Finalmente cabe señalar cómo se realiza el proceso de calibración de los joysticks. La calibración establece los valores de parada, los máximos y los mínimos de los ejes de los joystick. Dicho proceso de calibración consta de dos partes:

- Valores iniciales: en esta primera parte el usuario no debe mover los controles, así se establece los valores iniciales.
- Valores máximos y mínimos: en esta segunda parte el usuario debe mover los joysticks en los dos ejes para que el programa obtenga los valores máximos y mínimos.

La calibración del gamepad se realiza al iniciar el programa. Se muestra un diálogo por pantalla para indicar al usuario qué debe hacer. A continuación se muestra una captura de pantalla del proceso de calibración:

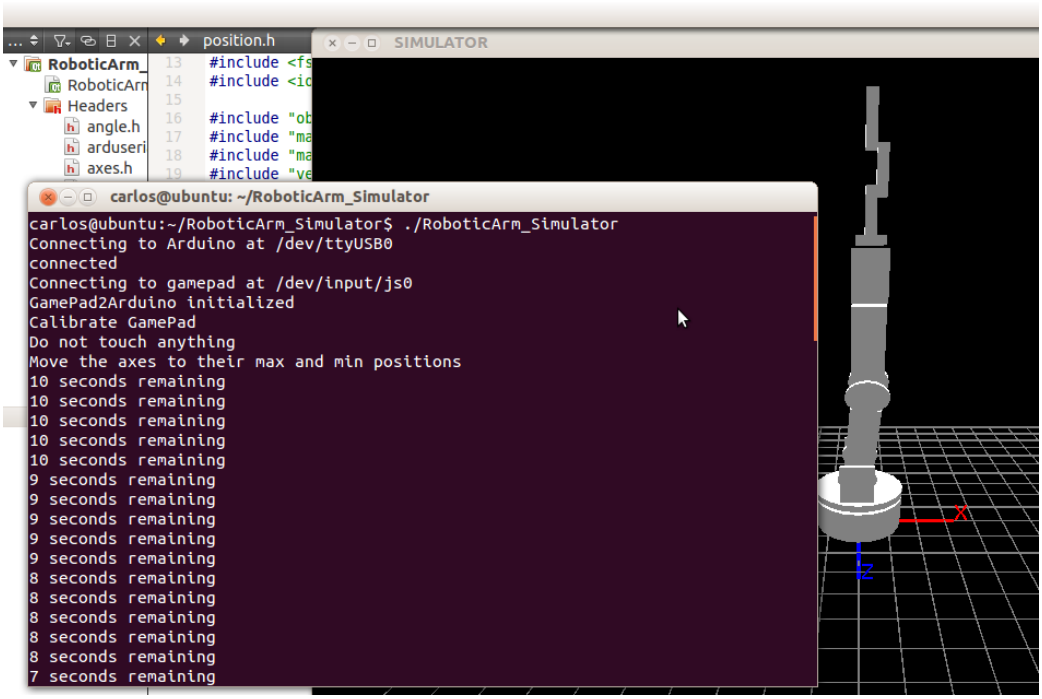


Ilustración 50. Captura de pantalla del proceso de calibración del gamepad

PROCESOS

En este apartado describiremos los diferentes procesos seguidos por el programa implementado. Podemos dividir estos procesos en diferentes partes:

- Flujo principal del programa.
- Bucle principal (función glutMainLoop)
- Función OnDraw

En la ilustración 51 mostramos el diagrama de flujo correspondiente al programa principal. Como se puede observar en él, el programa comienza inicializando el Gamepad, las librerías de OpenGL, la ventana donde se visualiza el robot, establece las cámaras y las luces de la escena. Después crea los objetos que forman el entorno gráfico y los guarda en un vector. Éste vector, llamado World, es importante ya que guardará todos los objetos dibujables del simulador.

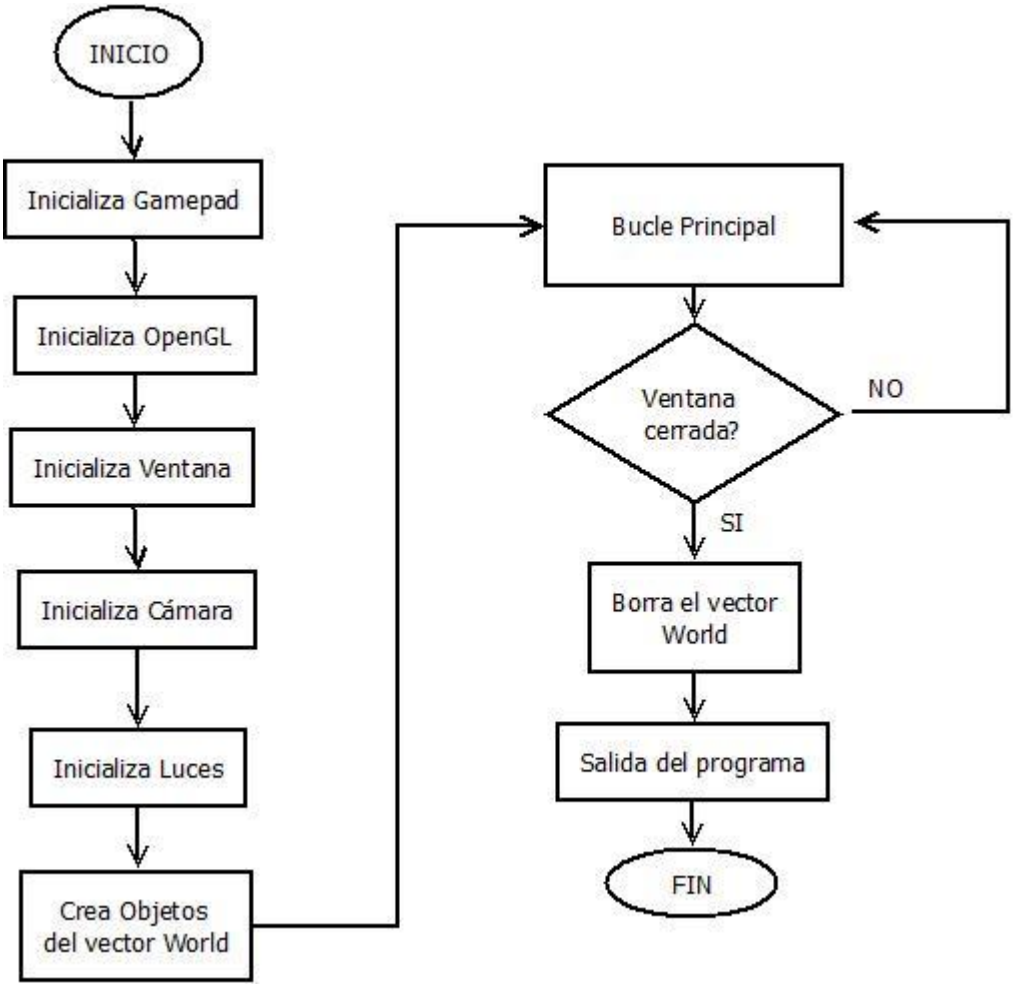


Ilustración 51. Diagrama de flujo principal del programa.

Una vez ejecutados los pasos anteriores entramos en el bucle principal del programa. Este se realiza gracias a la función “glutMainLoop” que incorpora la librería GLUT. Se saldrá del bucle cuando el usuario cierre la ventana principal del simulador o detenga el proceso de ejecución.

Durante el bucle principal se realiza la comprobación del teclado, la lectura de los botones del Gamepad, el cálculo de la posición y el dibujado de la representación gráfica en 3D del brazo robótico. Además muestra los parámetros del robot por pantalla. Se puede observar el proceso seguido por el bucle en el diagrama de flujo siguiente.

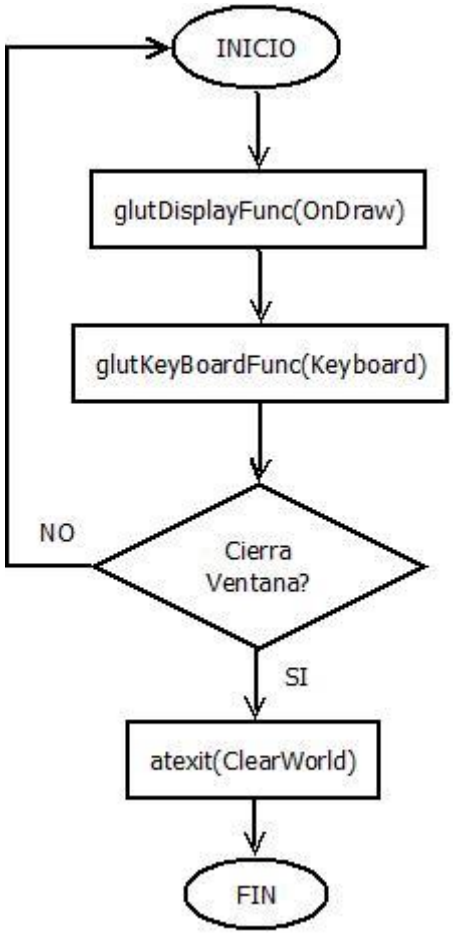


Ilustración 52. Diagrama de flujo del bucle principal (glutMainLoop)

Del diagrama anterior cabe destacar que las funciones *glutDisplayFunc*, *glutKeyboardFunc* y *atexit* son callbacks o retrollamadas. De esta forma, dichas funciones serán ejecutadas cuando se produzca el evento correspondiente.

Finalmente, dentro de las funciones realizadas por el programa nos encontramos con la función OnDraw. Como hemos podido ver en la ilustración 52, ésta función está dentro de un callback (glutDisplayFunc). Así se ejecutará cuando se produzca un evento determinado. En este caso el evento es cualquier mensaje enviado por el usuario a través del teclado o el gamepad.

La función OnDraw se centraba inicialmente en el dibujado de los elementos del simulador. No obstante, según se iba desarrollando el programa, se han ido incorporando más tareas a esta función. Dichas tareas son las siguientes:

- Recepción de datos del Gamepad
- Grabación de movimientos o posición
- Reproducción de movimientos o posición
- Envío de datos al robot

El proceso que sigue esta función está reflejado a través de su diagrama de flujo que se expone a continuación. Cabe destacar que en él aparecen los tres modos de funcionamiento del simulador, es decir, la teleoperación, la grabación y el posicionamiento.

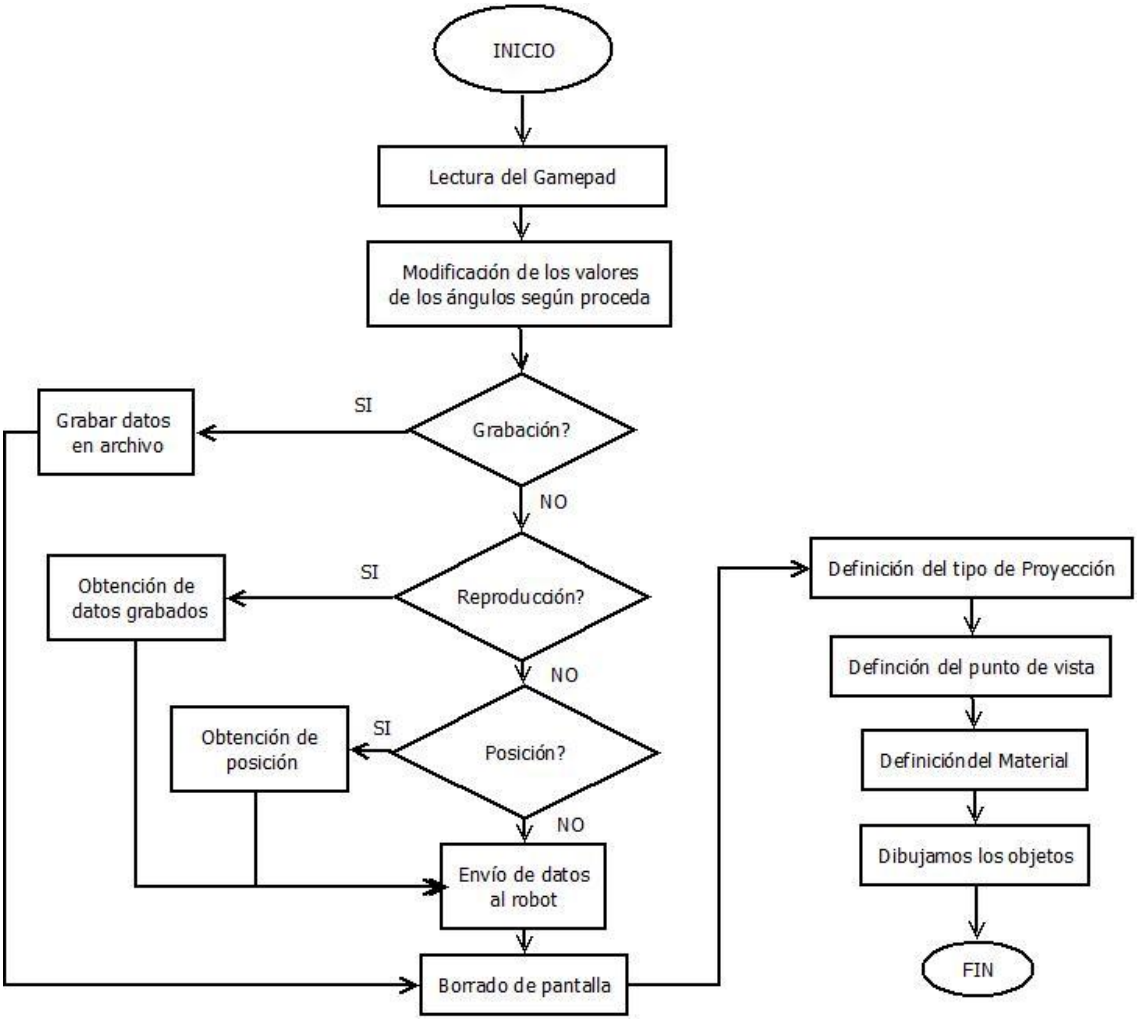


Ilustración 53. Diagrama de flujo de la función OnDraw

4.4. DISEÑO DEL SIMULADOR

Para dibujar las piezas del robot se ha utilizado las librerías de OpenGL (*ver apartado 4.2. OpenGL*). Como se ha descrito anteriormente, se han creado diferentes clases para, entre otras funciones, facilitar el proceso de dibujado del simulador.

A pesar de haber desarrollado la descripción de las clases encargadas del dibujado anteriormente (*ver apartado 4.3. Implementación del programa sección clases*), las retomaremos desde el punto de vista del diseño del simulador.

En primer lugar, se han implementado la clase "Cuboid" que se corresponde con la figura geométrica de un ortoedro. El constructor de la clase establece los atributos como el ancho, el largo y el alto así como la posición (x,y,z) donde se dibuja el objeto. Además la clase cuenta con la función "DrawGL" que se encarga de dibujar el ortoedro con los parámetros dados.

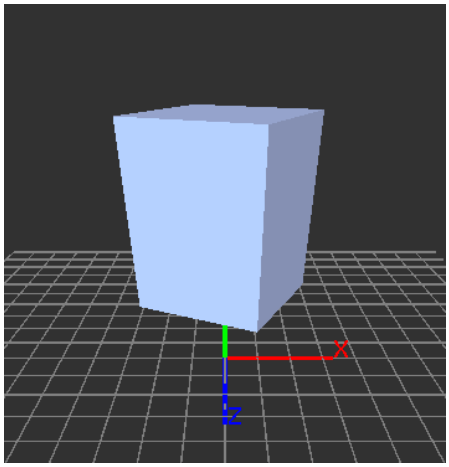


Ilustración 54. Representación de un objeto de la clase Cuboid

Las traslaciones y rotaciones en OpenGL se realizan a partir del sistema de referencia desde el que se dibuja el objeto. Así, este sistema se convierte en el centro instantáneo de rotación (CIR). Según la implementación realizada, para establecer el CIR que nosotros deseemos se han tenido que añadir unos parámetros de traslación de este punto (tCIRx, tCIRy, tCIRz).

En segundo lugar, se ha implementado la clase "SolidCylinder". Ésta se corresponde con un cilindro incluyendo la cara superior e inferior. Para construir un objeto de esta clase sólo es necesario definir su posición (x, y, z), el radio y la altura del cilindro.

Al igual que con la clase "Cuboid", la clase "SolidCylinder" también cuenta con la función "DrawGL" que se encarga de dibujar el cilindro con los parámetros dados. La librería de OpenGL posee las funciones que dibujan cilindros huecos y discos. Así, para la implementación de el cilindro completo hemos utilizado un cilindro hueco y, situándolos correctamente, dos discos que hacen de cara inferior y superior.

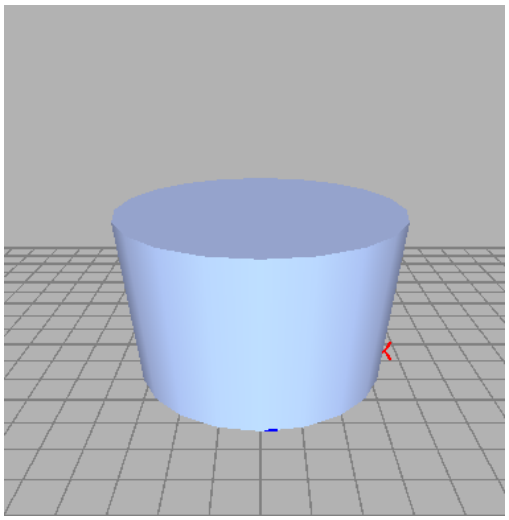


Ilustración 55. Representación de un objeto de la clase SolidCylinder

A partir de las dos clases básicas anteriores, Cuboid y SolidCylinder, se han implementado las clases: Base, FisrtLink, SecondLink, ThirdLink y FourthLink. Éstas clases se han implementadas con la finalidad de dibujar los diferentes eslabones del robot. De ésta forma conseguimos mayor simplicidad y encapsulamiento de código.

A continuación mostramos la implementación paso a paso de los diferentes objetos que conforman la representación 3D del robot. En la ilustración siguiente podemos ver la secuencia de dibujado de la base, los soportes laterales y el primer eslabón.

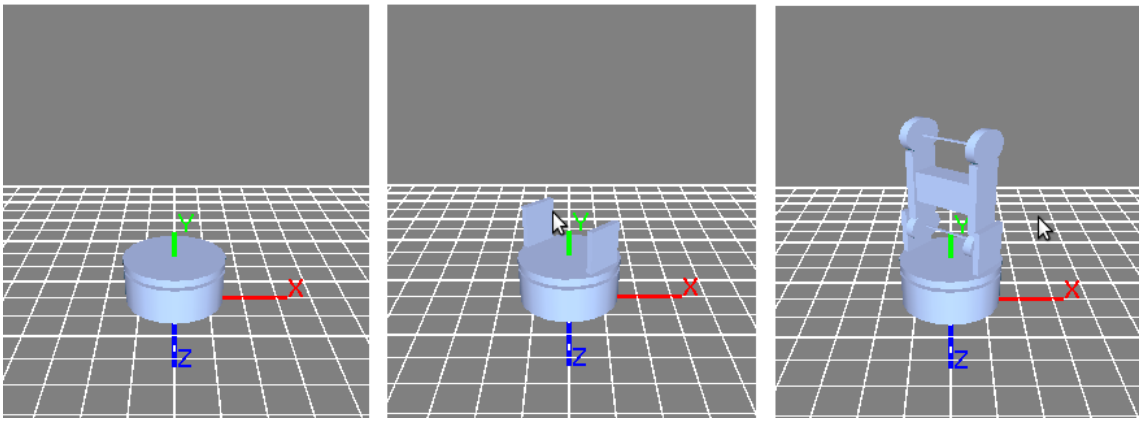


Ilustración 56. Secuencia de dibujado de la Base, los soportes y el primer eslabón

Seguidamente se muestra la continuación de la secuenciad de dibujado añadiendo el segundo eslabón y el tercero y cuarto que forma la muñeca del robot.

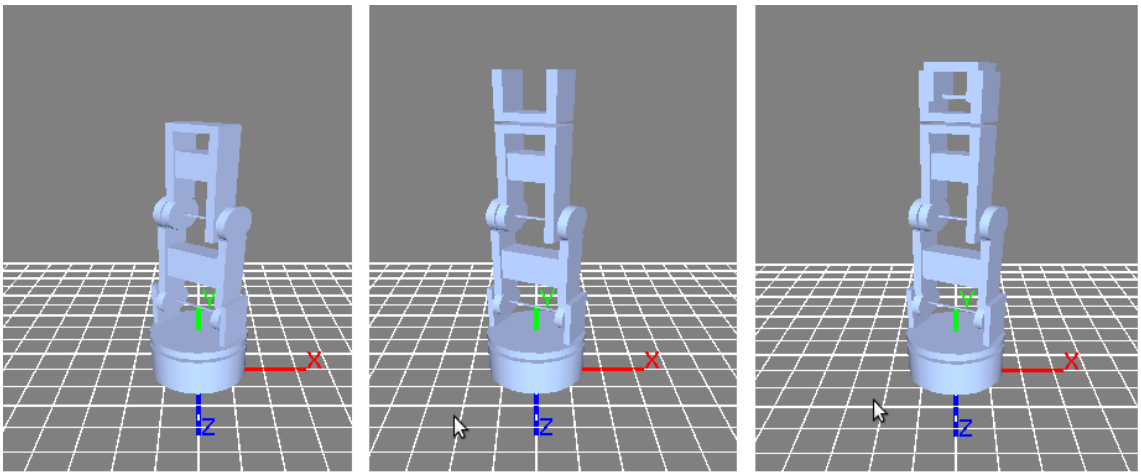


Ilustración 57. Secuencia de dibujado de los eslabones segundo, tercero y cuarto.

Para el tercer y cuarto eslabón, pertenecientes a la muñeca, se ha realizado una simplificación para facilitar el dibujado. Para ello no se ha respetado la forma original y se ha optado por una aproximación basada en ortoedros. No obstante, se ha mantenido los ejes de movimiento y el funcionamiento de éstos.

Por último, tenemos la clase "Gripper" que se encarga de dibujar la pinza que lleva incorporada el brazo robótico en su extremo. Esta clase también ha sido simplificada con respecto a la pinza real. Se ha eliminado el accionamiento de la pinza y se ha representado únicamente los eslabones que participan en la retención de objetos. No obstante se ha respetado el principio de funcionamiento y los movimientos.

A continuación podemos ver una imagen de la representación en tres dimensiones del brazo robótico. En ella podemos diferenciar la base, los eslabones y la pinza abierta en el extremo del robot.

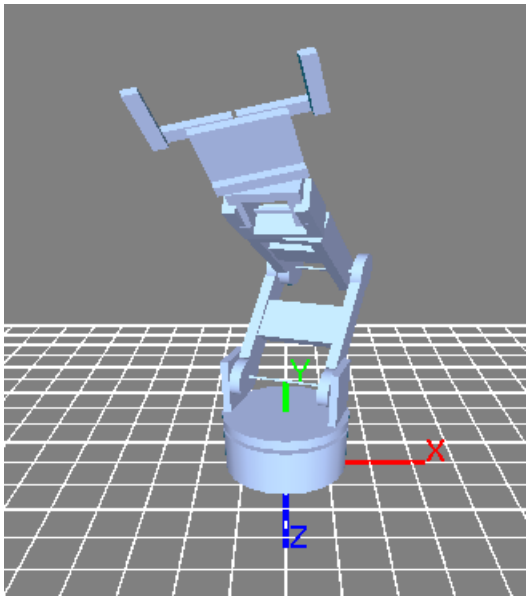


Ilustración 58. Representación del robot completo con la pinza abierta en su extremo.

4.5. ENVÍO Y GRABACIÓN DE DATOS

Se ha establecido un protocolo de envío basado en datos separados por puntos. Los datos están compuestos por una letra y un número. La letra indica a qué servomotor o eje del robot nos dirigimos. El valor numérico que la sucede indica el ángulo en grados que debe adoptar el eje. Así, un ejemplo de línea de mensaje podrías ser el siguiente:

A100.B90.C120.D10.E0.F90.G90.H180

Del ejemplo se obtendría que el primer eje, representado por la letra A, tendría que adoptar la posición de 100°; el segundo eje 90°, el tercero 120° y así sucesivamente hasta completar los 8 datos que se envían.

Este protocolo de mensajes se ha utilizado también para realizar el guardado de datos en el modo de grabación del simulador. Así cuando el usuario activa dicho modo, los datos generados por los diferentes ejes son guardados en un fichero. Para la reproducción de la grabación simplemente se accede al fichero y se van leyendo los datos. Así se reproducen los movimientos en el simulador y, simultáneamente, son enviados al robot.

4.6. FIRMWARE MICROCONTROLADOR

El microcontrolador es el Arduino UNO. Se programa en lenguaje C mediante un entorno de desarrollo llamado Arduino IDE (Integrated Devolopment Environment) que suministra de forma gratuita el propio fabricante. El firmware tiene por objetivo traducir los datos que recibe por el puerto serie y transmitir las correspondientes órdenes a los servomotores.

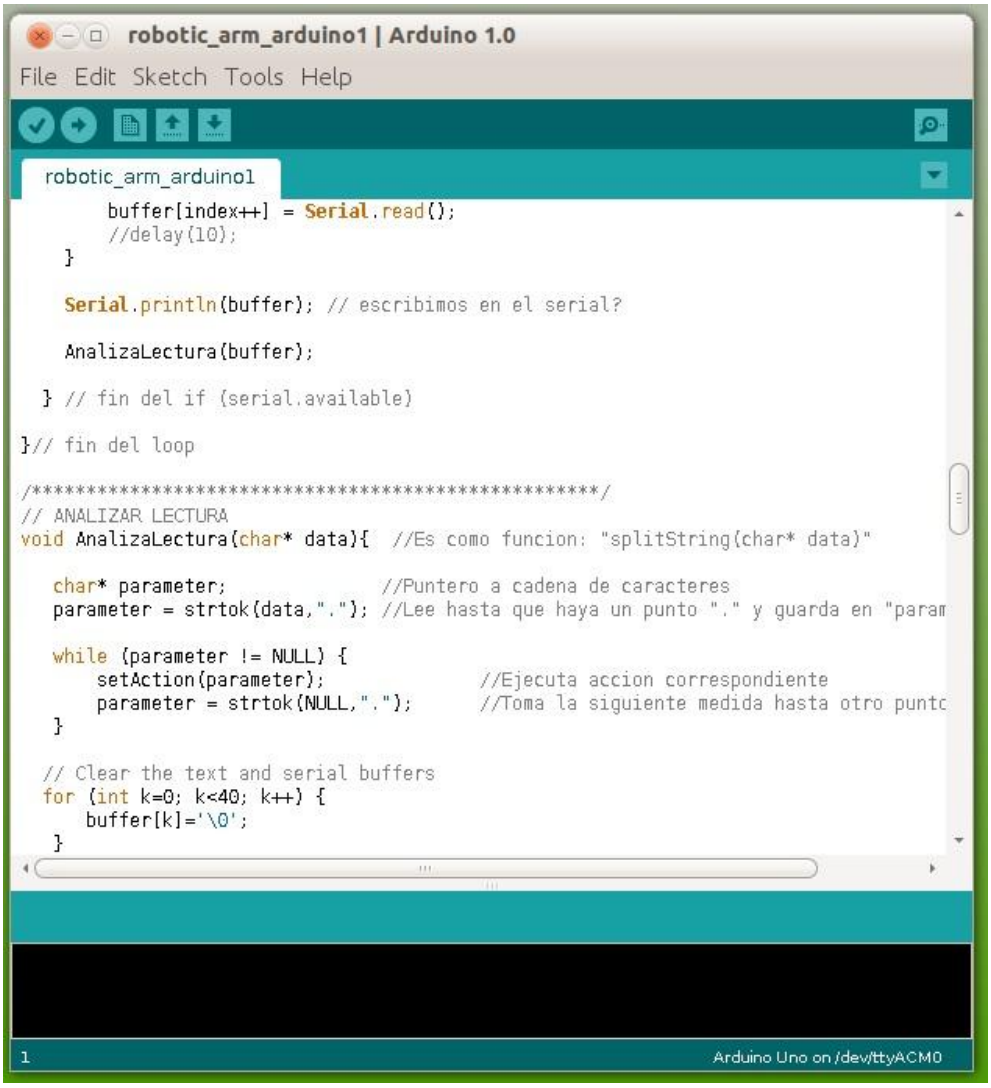


Ilustración 59. Entorno de desarrollo Arduino IDE

Para la programación se ha utilizado la librería servo.h. Esta proporciona funciones sencillas para inicializar los servomotores, asignarles el pin correspondiente y enviar la posición que deseemos adopte.

Gracias al compilador, una vez realizado el programa, lo enviamos al microcontrolador y éste lo almacena en memoria. De ésta forma, interpretará los datos que reciba por el puerto serie y le comunicará las instrucciones precisas al robot.

5. PUESTA EN FUNCIONAMIENTO

En este apartado describiremos los pasos necesarios para la puesta en funcionamiento del robot y su programa de control y simulación. Diferenciaremos entre dos partes, el hardware y el software.

HARDWARE

En cuanto al hardware nos encontramos con las piezas de plástico impresas, los servomotores, el microcontrolador y el cableado. Los pasos a seguir con estos materiales son:

1) **Adecuación de las piezas impresas:** en muchas ocasiones el acabado de las piezas no es siempre el óptimo, por ese motivo es necesario revisar y en su caso adecuar las piezas impresas. Se tendrán que lijar las superficies irregulares y perforar los agujeros que no hayan sido definidos correctamente en la impresión.
2) **Realización de pruebas previas:** es necesario comprobar previamente el correcto funcionamiento de los servomotores.
3) **Montaje de las piezas del robot y los servomotores:** se montarán las piezas del robot uniéndolas mediante tornillos y, si es necesario con pegamento. Los servomotores deberán estar bien fijados a la estructura del robot.
4) **Conexionado de los cables de control:** los cables de control comunicarán al microcontrolador con los diferentes servos. En la siguiente ilustración se muestra un esquema de conexionado a este respecto.

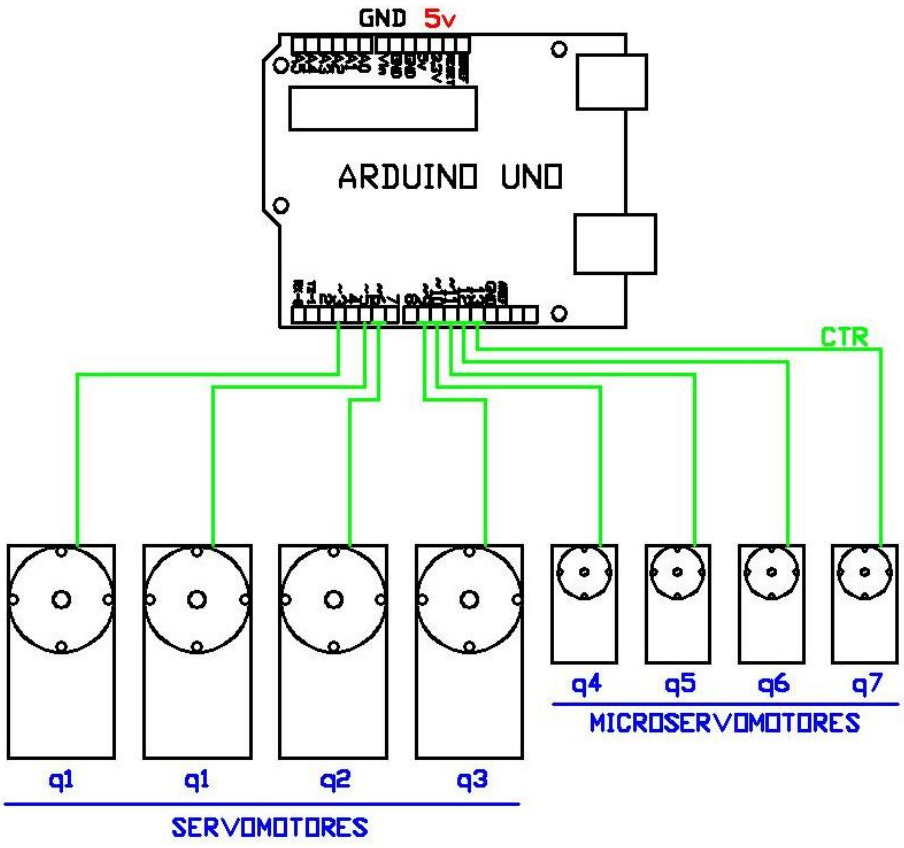


Ilustración 60. Esquema de conexionado de los cables de control de los servos

5) **Conexionado del microcontrolador al ordenador:** Para el conexionado entre el microcontrolador y el ordenador hemos planteado dos opciones:
   a. **Conexionado USB:** utilizaremos un cable USB e interconectaremos el microcontrolador y el ordenador.

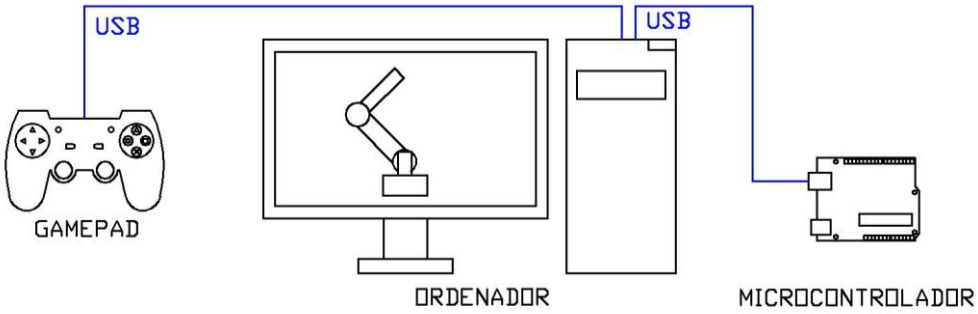


Ilustración 61. Esquema de conexionado USB

   b. **Conexionado inalámbrico:** utilizaremos el módulo bluetooth para comunicar el microcontrolador y el ordenador. De ésta forma dotaremos al robot de mayor movilidad (hasta 9m del punto de control) eliminando los cables.

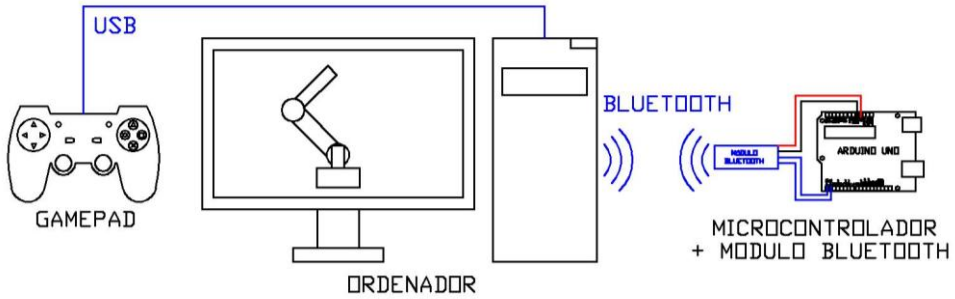


Ilustración 62. Esquema de conexionado USB y Bluetooth

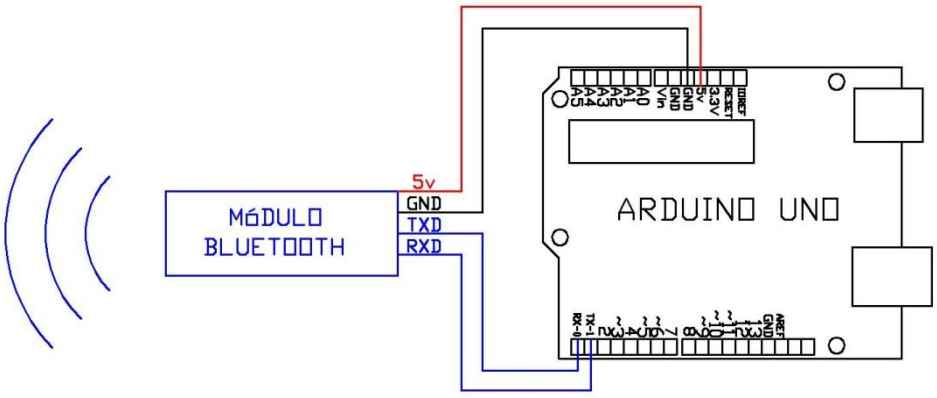


Ilustración 63. Esquema de conexionado del módulo bluetooth con el microcontrolador

6) **Conexionado de la alimentación:** en primer lugar se conectará la batería directamente a la entrada de alimentación del microcontrolador. Luego en paralelo a la batería se conectará la entrada del regulador de tensión (UBEC). Así, se utilizará la salida del UBEC para alimentar los servomotores. A continuación mostramos un esquema aclaratorio.

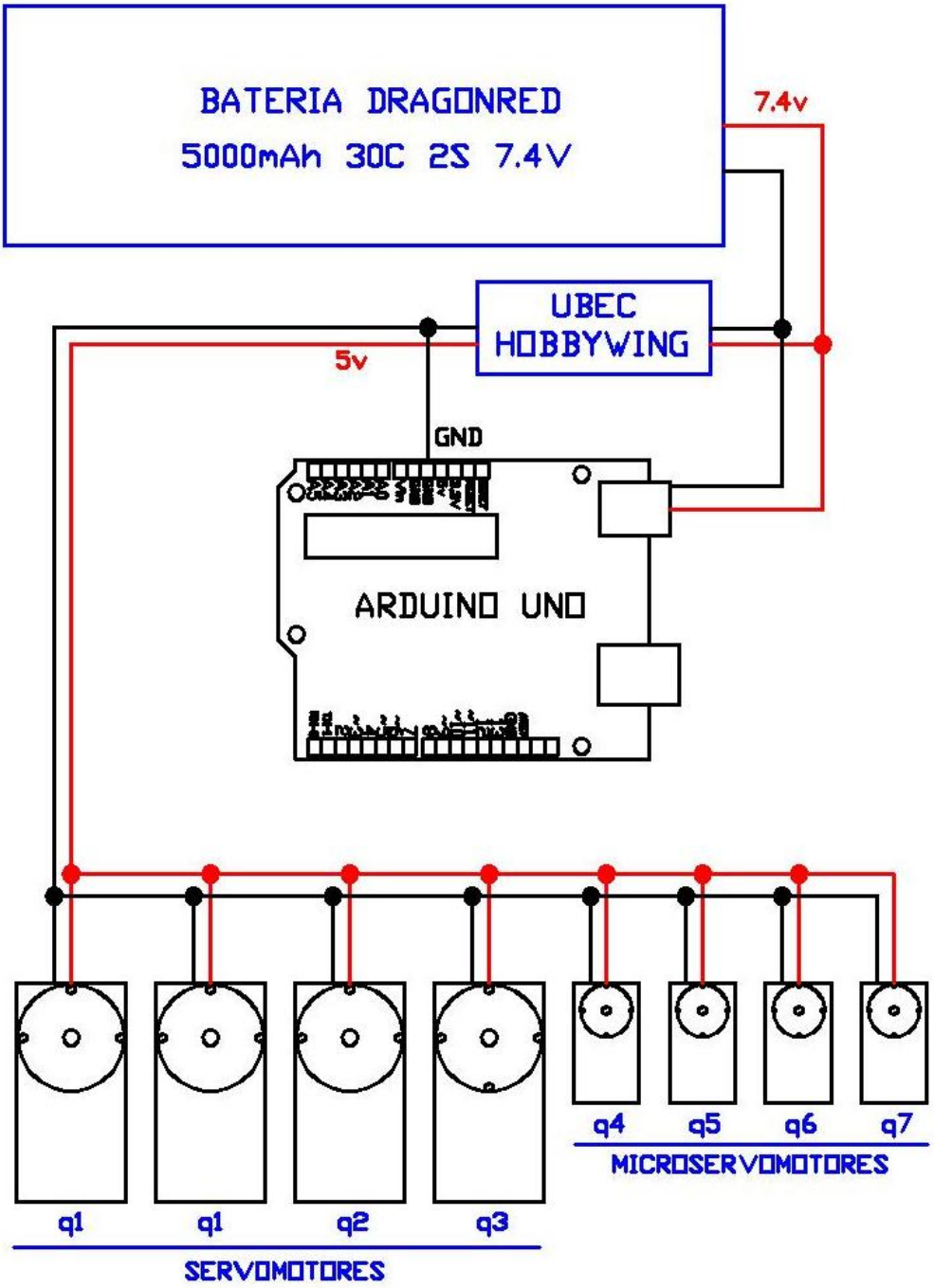


Ilustración 64. Esquema de conexionado de alimentación

SOFTWARE

En este apartado describiremos los pasos para poder ejecutar correctamente el programa de simulación. Cabe señalar que el software en su totalidad ha sido implementado en el sistema operativo Ubuntu 12. No obstante, las librerías utilizadas son multiplataforma de manera que la compatibilidad con otros sistemas operativos está garantizada.

1. **Instalación de las librerías necesarias:** el programa utiliza librerías que pueden no estar instaladas en el ordenador donde se desee ejecutar el programa. Así será necesario proceder a la instalación de éstas previa ejecución del programa. Las librerías necesarias son GL, GLUT, GLUI (*para más información ver anexo 8.3. Instalación de librerías*).
2. **Comprobaciones previas:** será necesaria la comprobación de las conexiones y la asignación de los puertos de los periféricos. Si no se realizan todas las conexiones el programa no funcionará correctamente o incluso detendrá su ejecución. La asignación de los puertos del ordenador a los periféricos utilizados figura en la siguiente tabla.

Tabla 13. Asignación de puertos a los periféricos

Periférico	Puerto
Gamepad	/dev/input/js0
Microcontrolador	/dev/ttyUSB0

3. **Ejecución del programa:** podemos ejecutar el programa a través de un entorno de desarrollo de software o directamente desde el terminal. Para ello nos situaremos en la carpeta del programa “RoboticArm_Simulator” y lo ejecutaremos.

Si se han realizado correctamente todos los pasos anteriores el programa se ejecutará satisfactoriamente. Así, el usuario podrá controlar el brazo robótico a través del gamepad y visualizar la simulación.

6. CONCLUSIONES

Se ha alcanzado el objetivo principal de este proyecto que era el desarrollo del modelo cinemático, el control y la simulación de un brazo robótico imprimible. A continuación iremos detallando los diferentes puntos que ha abarcado el proyecto:

✓ Comunicación

El usuario puede mandar órdenes de actuación al robot a través del teclado o del gamepad. El programa interpreta las indicaciones, muestra la nueva posición del robot en el simulador y se comunica con el microcontrolador. Éste interpreta el mensaje y manda las señales a los servomotores para que adopten la posición correspondiente.

Esto se ha conseguido gracias a un protocolo predeterminado de mensajes, una programación de envío a través del puerto serie y un firmware en el microcontrolador que interpreta el mensaje

✓ Simulación en 3D del brazo robótico

Para la simulación en 3D del brazo robótico, se ha implementado una interfaz gráfica que muestre una representación del robot con un posicionamiento y orientación determinados. Además se muestra por pantalla los valores de los ejes y la posición del extremo del robot.

✓ Cinemática directa e inversa del brazo robótico

Se ha dado solución al problema cinemático directo por medio del algoritmo de Denavit-Hartenberg. El problema cinemático inverso se ha solucionado por medios geométricos. Así, se ha conseguido obtener la posición del extremo del robot a partir de los valores de los ángulos del mismo y viceversa.

✓ Teleoperación remota

La operación remota del brazo robótico es posible gracias al simulador y al gamepad. El usuario recibe la información gráfica del posicionamiento y la orientación del robot a través del monitor del ordenador. Así, puede teleoperar el robot gracias a la representación gráfica del simulador.

✓ Grabación de movimientos

La grabación de movimientos es posible gracias al programa de simulación. Así, el usuario puede realizar la grabación de movimientos simulados y reproducirlos en el robot real. Para conseguir esto el programa genera un archivo donde guarda todos los movimientos.

✓ Posicionamiento

Gracias al modelo cinemático el usuario podrá introducir un punto del espacio y el robot adoptará dicha posición. Esto se ha conseguido gracias a la solución del problema de la cinemática inversa del robot.

PROYECTOS FUTUROS

En primer lugar se diseñó el brazo robótico. En segundo lugar se implementó el control y la simulación del mismo. Ahora quedan aún muchas cosas por hacer que pueden ser objeto de proyectos futuros.

Por ejemplo se podría crear un sistema de control con realimentación para este brazo robótico. Esto sería posible incorporando un cable más a los servomotores de forma que puedan transmitir la posición en la que se encuentran.

Otro posible proyecto consistiría en desarrollar un sistema de sensado. Así se podría instalar un sensor óptico entre las pinzas para detectar si existe un objeto que agarrar. También, utilizando detectores de presión en los laterales de las pinzas, podríamos detectar si cogemos o no correctamente un objeto.

Se podrían diseñar diferentes herramientas imprimibles para colocarlas en el extremo del brazo robótico como por ejemplo una pinza más precisa, un pequeño taladro o destornillador eléctrico.

Por útlimo, quiero mencionar un proyecto de futuro muy cercano. Se trata de la participación en el concurso “Mars Challenge” organizado por el Departamento de Ingeniería de Sistemas y Automática. El objetivo es crear un robot teleoperado que sea capaz de moverse por un entorno marciano, recoger un objeto del entorno y transportarlo.

Para cumplir los requisitos del concurso, el brazo robótico será incorporado en un vehículo imprimible objeto del proyecto fin de grado de María Ramos Montero. De esta forma, conseguiremos que el vehículo y el brazo robótico se desplacen por el entorno marciano y pueda recoger y transportar el objeto.

7. BIBLIOGRAFÍA

Recursos en formato físico

1. Antonio Barrientos, Luis Felipe Peñín, Carlos Balaguer y Rafael Arecil: "**Fundamentos de Robótica Industrial**" Ed. McGraw-Hill 1997.

Recursos en formato electrónico

1. **OpenGL:** "The Industry's Fundation for High Performance Graphics". Página web con información relativa a OpenGL. Dirección: http://www.opengl.org (Última consulta: 30 de agosto 2012).

2. **Cplusplus**: página web con información general a cerca de la programación en lenguaje C++. Dirección web: http://www.cplusplus.com (Última consulta: 30 de agosto 2012.)

3. **C con clase**: página web con información general a cerca de la programación en C y C++. Dirección web: http://c.conclase.net (Última consulta: 30 de agosto 2012).

4. **Iearobotics:** "Ingeniería, Electrónica, Aplicaciones frikis y Robótica". En ésta página se encuentra información relativa a las impresoras 3D y diversos proyectos de robótica. Dirección web: http://www.iearobotics.com (Última consulta: 30 de agosto 2012.)

5. **Curso de introducción a OpenGL (v1.0)**, Jorge García (Bardok) 2003. Disponible en el enlace web: http://iearobotics.com/alberto/lib/exe/fetch.php?media=teaching:manual-opengl.pdf (Última consulta: 30 de agosto 2012).

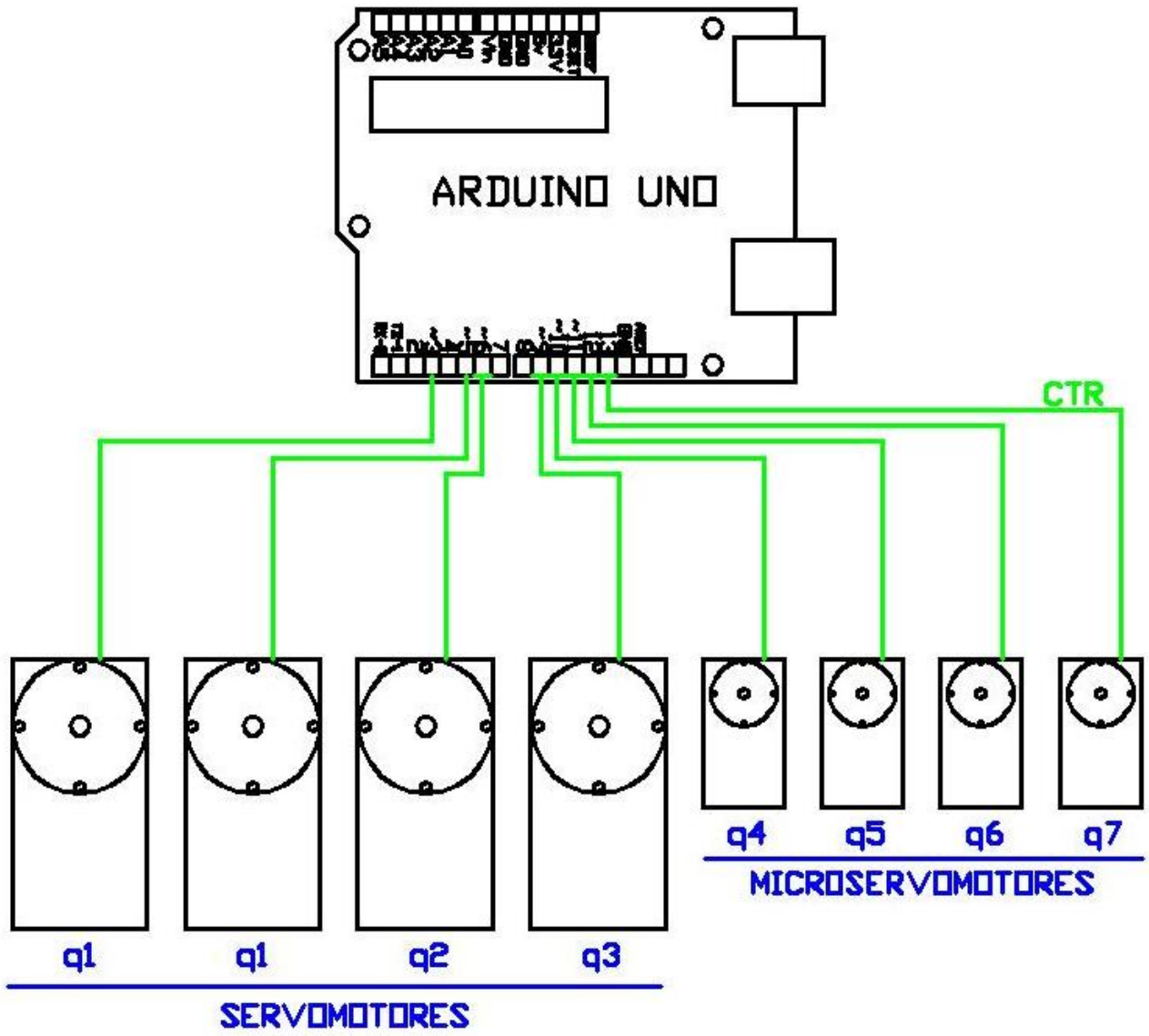
6. **Thingiverse:** portal de la empresa MakerBot de donde se pueden obtener diseños de piezas imprimibles compartidos por usuarios de forma gratuita. Dirección web: http://www.thingiverse.com (Última consulta: 30 de agosto 2012).

7. Librería "servo.h" de Arduino. Esta librería incorpora funciones para el manejo de servomotores desde el microcontrolador Arduino. Dirección web: http://arduino.cc/es/Reference/Servo (Última consulta: 30 de agosto 2012).

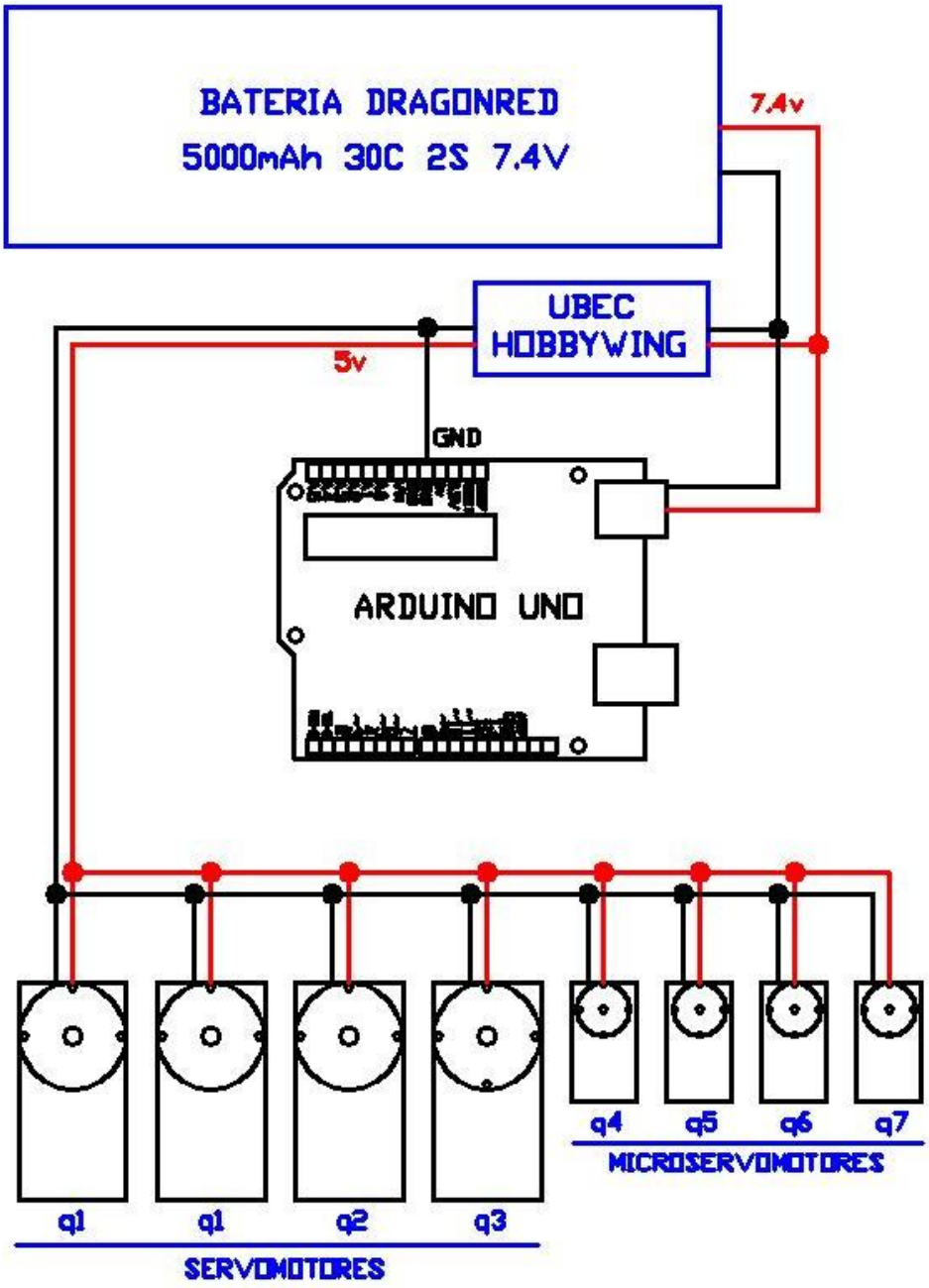
8. ANEXOS

8.1. PLANOS

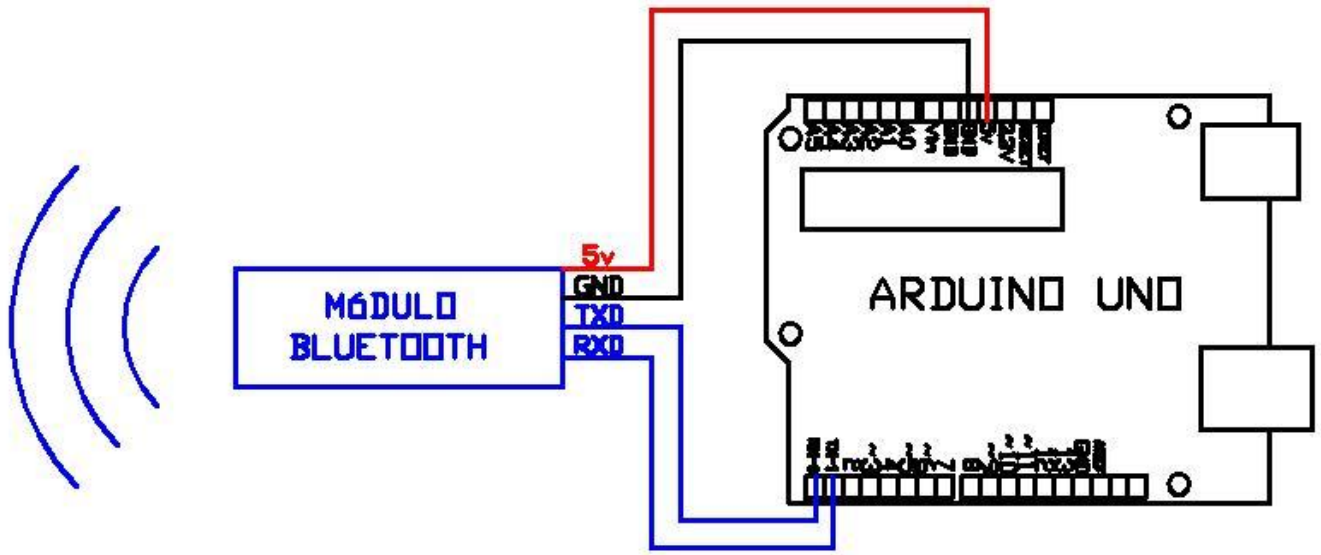
- PLANO 1 - CONEXIONADO DE CABLES DE CONTROL
- PLANO 2 – CONESIONADO DE CABLES DE ALIMENTACIÓN
- PLANO 3 - CONEXIONADO DEL MÓDULO BLUETOOTH



TÍTULO	MODELO CINEMÁTICO Y CONTROL DE UN ROBOT IMPRIMIBLE	
PLANO	CONEXIONADO DE CABLES DE CONTROL	
INGENIEROS	Don Juan Carlos Rodríguez Zambrana	FIRMA
FECHA 01/09/12	SITUACIÓN Madrid	PLANO Nº 01
ESCALA 1:1	PROPIEDAD Universidad Carlos III de Madrid	



TÍTULO	MODELO CINEMÁTICO Y CONTROL DE UN ROBOT IMPRIMIBLE
PLANO	CONEXIONADO DE CABLES DE ALIMENTACIÓN
INGENIEROS	Don Juan Carlos Rodríguez Zambrana
FIRMA	
FECHA	01/09/12
SITUACIÓN	Madrid
PLANO Nº	02
ESCALA	3:4
PROPIEDAD	Universidad Carlos III de Madrid



TÍTULO	MODELO CINEMÁTICO Y CONTROL DE UN ROBOT IMPRIMIBLE	
PLANO	CONEXIONADO DEL MODULO BLUETOOTH	
INGENIEROS	Don Juan Carlos Rodríguez Zambrana	FIRMA
FECHA 01/09/12	SITUACIÓN Madrid	PLANO Nº 03
ESCALA 1:1	PROPIEDAD Universidad Carlos III de Madrid	

8.2. PRESUPUESTO

Código	Unidad medida	Descripción	Cantidad	Precio Unitario	Precio Total
01.		**Capítulo 1 - Software**			
01.02	und	*OpenSCAD*			
		Software OpenSCAD para la creación de objetos sólidos en 3D. Es gratuito y multiplataforma.	1	0,00 €	0,00 €
01.03	und	*ReplicatorG*			
		Software ReplicatorG de manejo de las impresoras 3D. Es gratuito y multiplataforma.	1	0,00 €	0,00 €
01.03	und	*QTCreator*			
		Entorno de desarrollo de software de la empresa Nokia. Es gratuito y multiplataforma	1	0,00 €	0,00 €
01.04					
		TOTAL CAPITULO 1			***0,00 €***
02.		**Capítulo 2 - Componentes del robot**			
02.01	g	*Plástico ABS*			
		Filamento de plástico Acrilonitrilo Butadieno Estireno (ABS) para impresora 3D. Sección de 3mm.	300	0,01 €	3,00 €
02.02		*Servomotor Futaba S3305*			
	und	Servomotor del fabricante Futaba modelo S3305. Tensión de entrada de 4.8-6.0v. Torque: 8.9kg·cm. Dimensiones: 40x20x38.1mm. Peso: 46.5g	1	40,00 €	40,00 €
02.03	ud	*Servomotor Futaba S3003*			
		Servomotor del fabricante Futaba modelo S3303. Tensión de entrada de 4.8-6.0v. Torque: 4.1kg·cm. Dimensiones: 40x20x38.1mm. Peso: 37.2g	3	10,00 €	30,00 €
02.04	ud	*Micro-servomotor Tower Pro SG90*			
		Micro-servomotor del fabricante TowerProa modelo SG90. Tensión de entrada de 4.0-7.2v. Torque: 1.2kg·cm. Dimensiones: 22.5 x 12.0 x 26.5mm. Peso: 9.0g	4	4,00 €	16,00 €
02.04	ud	*Microcontrolador - Arduino UNO*			
		Microcontrolador Arduino uno con microprocesadro ATMEGA328.	1	21,90 €	21,90 €
02.04	m	*Cable unifilar*			
		Cable unifilar de 0.5mm	5	0,22 €	1,12 €
		TOTAL CAPITULO 2			***112,02 €***
		TOTAL			***112,02 €***

8.3. INSTALACIÓN DE LIBRERÍAS

Para la ejecución del programa de control y simulación es necesaria la instalación de diferentes librerías que éste utiliza. En este apartado describiremos el proceso seguido desde terminal para el sistema operativo Ubuntu o Debian.

1. El primer paso es instalar las librerías de desarrollo para OpenGL/Glut. Para ellos introduciremos la siguiente línea en el terminal:

```
$sudo apt-get install freeglut3 freeglut3-dev
```

2. Para versiones modernas de Ubuntu (>= 11.10) es necesario instalar un paquete extra debido a que el linker no enlaza correctamente:

```
$sudo apt-get install binutils-gold
```

3. Por último para utilizar las librerías de GLUI necesitamos instalar su correspondiente librería:

```
$sudo apt-get intall libglui-dev
```

Para el compilador utilizado en este proyecto (QTCreator) es necesario establecer una relación con las librerías que vamos a utilizar. Para ello introduciremos el siguiente código en el archivo de extensión ".pro" del proyecto:

```
LIBS  += -lGL \
         -lGLU \
         -lglut \
         -lglui \
         -lserial \
```

Finalmente, dentro del código del programa deberemos incluir las librerías necesarias. En el caso concreto de OpenGL, glut, glui y la comunicación puerto serie es necesario el siguiente código:

```
#include <GL\glut.h>
#include <GL\glui.h>
#include <SerialPort.h>
```