UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

CONTROL DE ROBOT CON PLATAFORMA DE SIMULACIÓN VIRTUAL CON PLUGIN INTEGRADO

AUTOR: JOSÉ TOMÁS MORA

TUTOR: LEOPOLDO ARMESTO ÁNGEL

Curso Académico: 2013-14

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Contexto histórico y antecedentes

El término robótica es un término moderno, pues no fue acuñado hasta entrado el siglo XX. Sin embargo la robótica viene desarrollándose, de manera conceptual desde el siglo III a.C. y algunos historiadores incluso se remontan hasta el siglo X a.C., también en la antigua China.

No obstante, dichos diseños e invenciones no estaban centrados en la funcionalidad y sólo desarrollaban el concepto de autómata como tal a nivel básico. No fue hasta el siglo XX cuando se empezaron a centrar en la funcionalidad de los mismos y aumentaron exponencialmente su complejidad, ayudados por el gran progreso tecnológico vivido desde el siglo XIX.

El primer robot industrial instalado en una cadena de montaje fue Unimate, creado por George Devol. General Motors fue la compañía que dispuso de este tipo de autómata en su cadena de montaje por primera vez en el año 1961. Unimate transportaba las piezas fundidas en molde hasta la cadena de montaje, donde soldaba dichas piezas sobre los chasis de los vehículos. Gracias a ello, se evitaba a los trabajadores el riesgo de inhalar gases nocivos provenientes de las soldaduras y el peligro de cercenamiento.

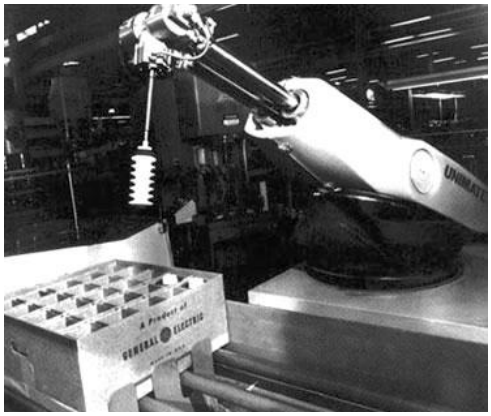


Fig.1. Foto de Unimate

Desde entonces y llegados a la época actual, hemos visto como los robots que trabajan en cadenas de montaje no son simplemente comunes, si no que son la parte fundamental de la mayoría de cadenas de montaje de las grandes empresas. La precisión y rapidez con la que realizan sus tareas es superior a la de cualquier humano, unido a que pueden realizar cualquier trabajo peligroso sin arriesgar ninguna vida humana explica que sean tan importantes hoy en día.

La gran evolución y difusión de la robótica no sólo ha afectado a las empresas que disponen de ellos, también a cualquier persona y estudiante que se quiera interesar por ello. Hoy en día casi cualquier persona puede acceder a la robótica si así lo desea, y con unos conocimientos mínimos podría ensamblar su propio robot. Sin embargo, va más allá, pues otra persona con unos conocimientos superiores podría hacerlo también pero aumentando su grado de complejidad todo lo que quisiera.

De manera que con todo ello, los avances en la robótica no se limitan al desarrollo que las grandes compañías hagan de ello, si no que cualquier persona interesada puede ser partícipe del desarrollo de la propia robótica.

1.2. Objetivos

I. ***Control:*** Se necesita, en primer lugar, un elemento que sirva como controlador. Este elemento de control tiene que ser capaz de ordenar al robot que se mueve de una determinada manera, en general, que realice una tarea concreta a voluntad del usuario que lo maneje.

II. ***Comunicación:*** Se debe establecer una comunicación entre el elemento de control y robot a controlar, que haga las funciones de puente entre las órdenes del usuario enviadas por el controlador, hasta llegar al robot.

III. ***Movimiento:*** La fase final, debe ser aquella que después del control y tras el proceso de comunicación, pueda verse reflejado en una orden real. Dicha orden real debe ser exactamente la que el usuario ha ordenado y debe ser ejecutada de tal manera.

1.3. Trabajo previo

Para poder enmarcar correctamente este trabajo se necesita comprender el proyecto dentro del cual se incluye en su totalidad. El trabajo previo que se nombra ha sido realizado por Oihan Elesgaray Susierra, estudiante de GITI en la ETSII, en su Trabajo Final de Grado, y se hará referencia a él siempre que se crea necesario a lo largo del trabajo.

Dicho trabajo comprende desde la fase inicial de diseño y montaje del brazo robot y su base móvil, hasta el desarrollo de una PCB (*Printed Circuit Board*) para facilitar la conexión de los servomotores y el USB del Arduino, al propio Arduino, para trabajar con su microprocesador.

2. Análisis de los objetivos

El objetivo principal de este trabajo, de ámbito académico, es conseguir controlar un robot real mediante su homólogo virtual simulado en un entorno virtual, replicando el primero los movimientos que se le ordenan al segundo.

Partimos de una simulación de una representación virtual de nuestro robot. En dicha simulación se dispondrá de unos *sliders* donde se ajustará la posición de cada uno de los servos del brazo, y otro par de *sliders* para ajustar las velocidades de los motores que mueven las ruedas de la base.

La simulación reproducirá el movimiento y posiciones que se le piden al robot a través de la interfaz, pudiendo ver su comportamiento gracias a que se está ejecutando la misma en tiempo real.

El siguiente paso sería desarrollar un *plugin* con el que podamos tratar la información de parte del código que se está ejecutando durante la simulación y poder enviarla.

Una vez definido esto, se necesita establecer una comunicación entre el software encargado de la simulación y el robot real. Para ello se define un protocolo de comunicaciones, tanto el camino que seguirá la información como los cambios que sufrirá la misma para poder obtener una intercomunicación óptima entre las partes.

La última parte del trabajo sería trasladar la información recibida al brazo, procesarla e interpretarla para poder ordenarle un movimiento y posicionamiento correcto. De manera que al final de su trayecto, la información final que ejecuta el microprocesador que controla el brazo sea equivalente a la que se le pide al robot virtual.

El propósito último del trabajo es obtener un método de control fácil e intuitivo para un robot (en este caso un brazo robot con una base móvil) y además poder aprovechar su integración en un entorno de simulación virtual. Con unos simples cambios también podríamos realizar la comunicación haciendo el camino inverso, partiendo de un objeto

físico real que podemos controlar para simular un movimiento en el entorno virtual al que podríamos añadir todos aquellos objetos que quisiéramos.

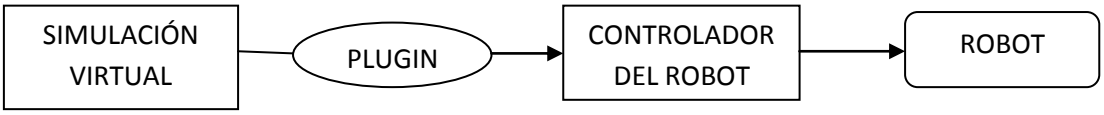


Fig. 2. Representación esquemática del trabajo

3. Soluciones propuestas

Para realizar la simulación se propone usar el software Coppelia Robotics V-REP (*Virtual Robot Experimentation Platform*), en su versión de licencia de estudiante, V-REP Pro Edu de la que disponen los estudiantes para su libre descarga en la UPV.

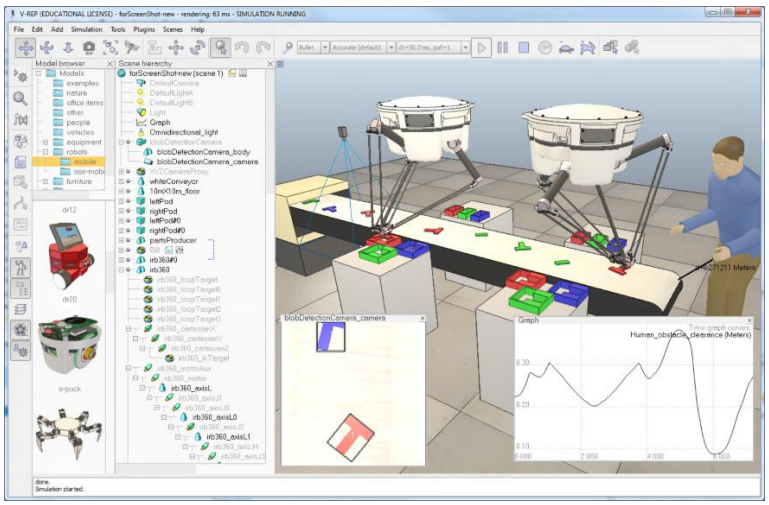


Fig. 3. Software de simulación robótica V-REP

V-REP es el software ideal para lo que se pretende, que es realizar una simulación en tiempo real del robot. Contiene una gran cantidad de opciones a la hora de establecer propiedades físicas del robot, así como para su control (extensa capacidad de APIs) y visualización de información.

Entre otras cosas, permite la simulación de sistemas de fabricación automatizados, monitorización remota, control de hardware, monitorización de seguridad, presentación de productos, etc.

Además, cualquiera de las maneras que se elijan para controlar los objetos de la simulación, es compatible con los lenguajes de programación más extendidos actualmente, como por ejemplo C/C++, Python, Java, Matlab, etc.

Cuenta además con otra ventaja importante, que es la gran cantidad de información que aporta el propio programa mediante ejemplos, tutoriales y archivos de los cuales uno mismo se puede servir para configurar lo que se desee. En V-REP podemos consultar una extensa guía de usuario de los cuales obtener todos los comandos para desarrollar el *plugin* que se necesita y obtener toda la información que se necesite.

Para el desarrollo del *plugin* que se integre en V-REP, se ha optado por usar la plataforma Microsoft Visual Studio. Visual Studio es de pago, pero dispone de versiones de prueba durante un periodo de tiempo, que es por lo que se opta en este trabajo. El *plugin* será escrito en lenguaje de programación C++ dentro del IDE Visual C++.

V-REP contiene dentro de sus archivos un archivo que sirve como esquema de un *plugin* desarrollado en Visual C++, el cual hay que configurar para que obtengamos de él lo que queramos, pero que ya está configurado desde un principio para que funcione como tal dentro del propio V-REP.

Elegimos el protocolo de comunicación puerto serie dentro del sistema operativo Windows. Dentro del propio *plugin* se configurará la comunicación que se realizará mediante este tipo de protocolo, y de ello también nos serviremos en la recepción de datos pues está implementado este tipo de comunicación en el software Arduino que usaremos. Con lo cual es el camino probablemente más sencillo, o simplemente directo, para realizar este intercomunicación entre programas.

Se decide elegir Arduino Mini como microcontrolador del robot. Ello se debe a que cuenta con un tamaño reducido que facilita el diseño e integración del mismo en el robot, cuenta con un precio reducido pudiendo realizar todo aquello que se desea en el robot, es decir que no es necesario un microcontrolador mayor y con mayores prestaciones. También se tienen en cuenta las facilidades para programar la placa, que cuenta con un software propio que lleva el mismo nombre, Arduino.

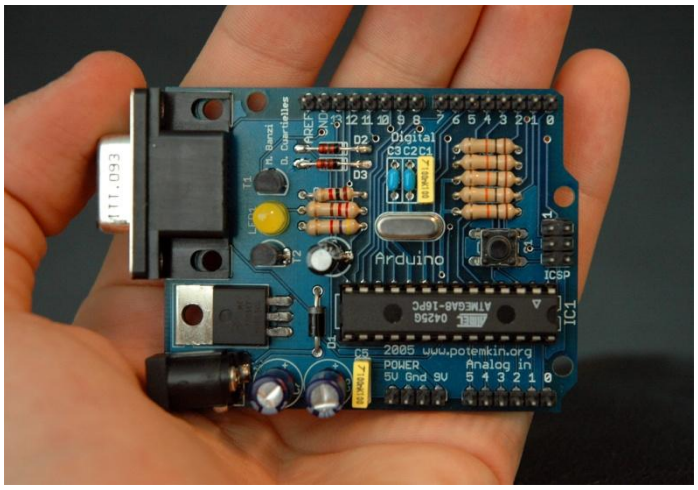


Fig. 4. Microcontrolador Arduino

Partiendo de la información enviada por el puerto serie, usaremos el software de Arduino, con las librerías oportunas, para tomar esa información, descifrarla y usarla. La plataforma software de Arduino es gratuito, de código abierto, y está originalmente escrito en Java. Gracias a este software enviaremos a los servos y motores la información recibida (y posteriormente tratada) desde la simulación, y el robot realizará esos mismos movimientos.

Cabe destacar también las facilidades que supone usar el software Arduino para el control de los servos, para los que dispone de su librería propia y también para el control de potencia, pues también dispone de comunicación I^2C gracias a otra de sus librerías. Es decir que es una solución ideal para el problema que nos ocupa.

Para más información del software utilizado consultar ***anexo 8.1***.

4. Elementos constituyentes del robot

4.1. Brazo robot y base móvil

4.1.1. *Diseño y software*

Los diseños del brazo robot corresponden al Proyecto Fin de Carrera de Antonio Castro Gómez y no se pretende hacer pasar por diseño propio. Se trabaja sobre dicho diseño, al cual se le añaden dos ruedas en su base con sus correspondientes motores y una rueda loca para obtener una configuración diferencial. De esta manera pasamos de tener un brazo robótico fijo a uno sobre una base móvil.

Una vez se tienen los diseños en el programa SolidWorks en formato STL se utiliza el software Slic3r para convertir dicho formato en un Gcode, con el que el software que usa la impresora 3D, Repetier-Host, pueda trabajar.

4.1.2. *Impresión, montaje y hardware*

La impresora 3D extruye un polímero termoplástico a alta temperatura mientras la superficie de trabajo, que es una base móvil, se mueve y el cabezal permanece fijo (en algunos casos es el cabezal el que se mueve sobre la superficie de trabajo de la impresora), generando capas de este polímero.

El polímero termoplástico utilizado es el acrilonitrilo butadieno estireno (ABS), que una vez que se enfría adquiere rigidez y dureza, proporcionan piezas solidas con las que poder ensamblar el robot. Cabe destacar además la facilidad de mecanizado de las piezas impresas en este material.

El último paso, una vez impresas todas las piezas, sería el montaje y mecanizado necesario para garantizar la correcta unión de las piezas.

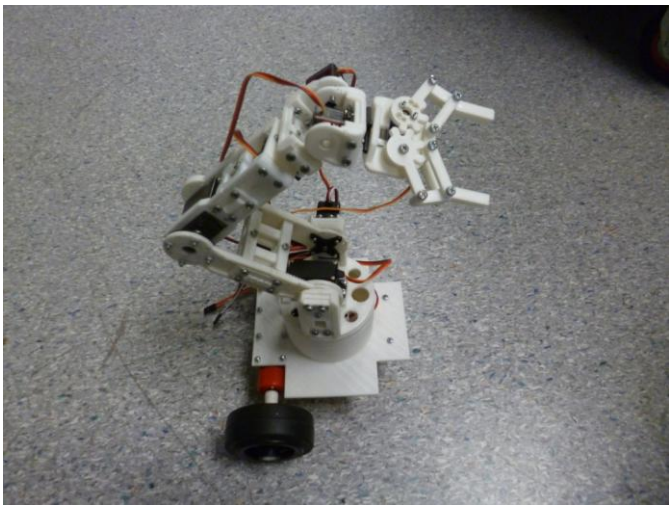


Fig. 5. Brazo robot y base móvil

La impresión de las piezas así como su mecanizado y montaje se realizó en el Instituto de Diseño y Fabricación (IDF) de la UPV, situado en la Ciudad Politécnica de la Innovación, utilizando la impresora 3D Prusa i3 de la que allí se disponía.

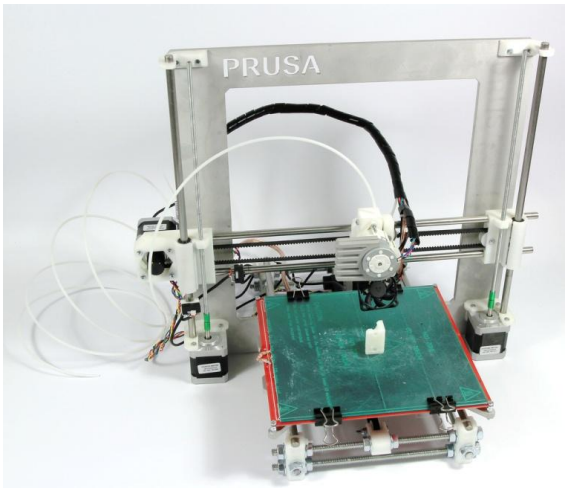


Fig. 6. Impresora 3D Prusa i3

Para más información relativa al diseño y montaje, consultar Trabajo Fin de Grado de Oihan Elesgaray Susierra, en el que se describe este montaje con más detalle.

4.2. Servomotores

4.2.1. *Características de los servomotores*

Los componentes encargados de dotar al brazo de movimiento son los servomotores, también llamados servos. La característica fundamental de los servomotores es su capacidad de ubicarse y mantenerse en cualquier posición dentro de su rango de operación.

En este caso se han utilizado 8 de ellos: Los 2 primeros de ellos los encontramos en la base del brazo, uno en la parte superior y otro en la parte inferior de la misma. El siguiente lo encontramos en la unión de la base con el brazo en su primera articulación. El cuarto sigue, de igual manera, uniendo la anterior parte del brazo con la siguiente articulación. El quinto servo proporciona movimiento de rotación al cabezal donde está incluida la pinza. El sexto, rota en un eje perpendicular al anterior, situado también en el cabezal donde está insertada la pinza. El séptimo proporciona un movimiento similar de rotación al quinto, pero afectando de manera independiente a la pinza. Y el último de los servos se encarga de abrir y cerrar la pinza para agarrar objetos.

Los servos tienen tres terminales de conexión. Uno para la alimentación positiva del servo, otro para Tierra (GND) y el último es el de la señal de control del servo. Generalmente el terminal de alimentación es de color rojo, sin embargo el código de colores para el resto suele oscilar entre fabricantes. Así por ejemplo el terminal de GND puede ser marrón o negro y el de la señal de control, blanco, naranja o amarillo.

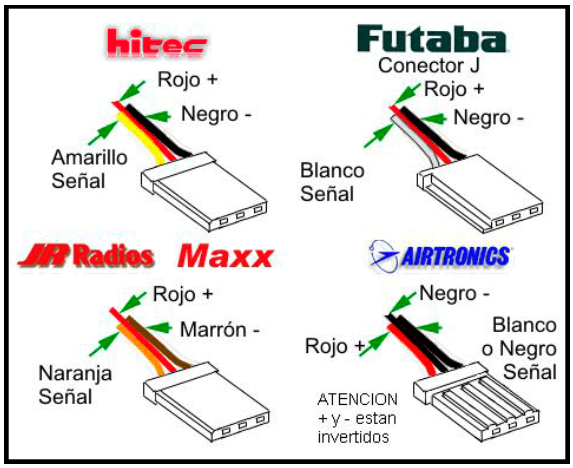


Fig. 7. Diferentes terminales de conexión de servomotores

A continuación se detallan las características de los servomotores utilizados:

CARACTERÍSTICAS	FUTABA S3003	TOWER PRO MG90S	TOWER PRO SG90	TOWER PRO MG995
Par (4.8V) [kg·cm]	3.17	1.8	1.8	10.0
Par (6.0V) [kg·cm]	4.10	2.2	-	-
Velocidad (4.8V) [s/60º]	0.23	0.1	0.1	0.2
Velocidad (6.0V) [s/60º]	0.19	0.08	-	-
Rango rotacional [º]	60	180	180	180
Peso [g]	37.0	13.4	9.0	55.0
Dimensiones [mm]	39.9x20.1x36.1	22.5x12x35.5	23.1x12.2x29.0	40.6x19.8x42.9

Tabla 1. Características de los servomotores empleados

4.2.2. *Justificación de la elección de los servomotores*

Los servos correspondientes a la parte inferior del brazo que otorgan movimiento de rotación a su base, son el modelo TowerPro MG995, que se puede observar es el que un mayor par otorga. Lógicamente, las partes más alejadas de la pinza son las que sufrirán un mayor par cuando esté cargada la pinza.

Fig. 8. Servomotor Tower Pro MG995

Siguiendo desde la parte inferior a la superior, las siguientes dos articulaciones del brazo serán las correspondientes al modelo Futaba S3003. Se obtiene un par aceptable de 3.17 kg·cm (o bien 4.10 kg·cm si se alimenta con 6V), superior al de los servos de la pinza y la cabeza de la pinza e inferior al que soporta al brazo entero en su base.

Fig. 9. Servomotor Futaba S3003

En las articulaciones de la parte superior del brazo, se usarán indistintamente los modelos TowerPro MG90S y TowerPro SG90. Son muy similares y la única diferencia es que el modelo MG90S acepta también alimentación a 6V, ofreciendo unas prestaciones ligeramente superiores. El motivo de la utilización de ambos modelos, en lugar de uno solo, fue la disponibilidad inmediata de dichos servos. Los servos los alimentaremos a 5 V (características a 4.8 V).

Fig.10. Servomotor Tower Pro MG90S

Fig.11. Servomotor Tower Pro SG90

4.3. Motores eléctricos

Se utilizan un par de motores eléctricos de corriente continua que trabajan con tensiones de 12 hasta 24 V, para mover, de manera independiente ambas ruedas motrices de la base del brazo robot. En su punto máximo puede girar a 6300 rpm, entregando al eje un par de 25 g·cm. Sin carga el eje podría girar hasta a 8400 rpm.

Fig.12. Motor eléctrico 12-24V DC ref. 413-0622

Para 24 V, lógicamente las revoluciones por minuto (rpm) a las que girarán el eje del motor, serán máximas. Si se alimentara el motor cambiando la polaridad de las conexiones, supongamos -24 V, las rpm que proporcionaría el eje también serían máximas, pero en el sentido totalmente opuesto al caso anterior.

Con ello podríamos conseguir que la base rotara sobre un punto fijo sin desplazarse si lo quisiéramos, alimentando un motor a 24 V y el otro a -24 V. Si consideráramos tan sólo la alimentación con una única polaridad también podríamos conseguir un movimiento rotacional de la base con una diferencia de velocidades de motores izquierdo y derecho, sin embargo sí se desplazaría la base.

En cualquier caso, gracias a la rotación que también puede conseguir el brazo en su base gracias a los servos, no hay ningún problema a la hora de que el robot y el brazo se situaran para conseguir un objeto en cualquier posición del suelo.

4.4. Microcontrolador Arduino Mini

4.4.1. *Características de Arduino Mini*

Para controlar los servos y los motores se utiliza la placa Arduino Mini. Esta placa tiene como microprocesador el ATmega168, y cuenta como principal ventaja respecto a otros productos de la marca Arduino, con su reducido tamaño y su bajo precio.

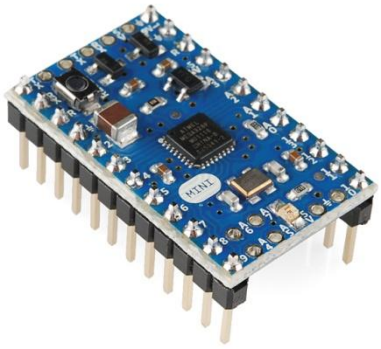


Fig.13.Arduino Mini 05

La alimentación y la comunicación de la placa Arduino con el PC se realiza a través del adaptador mini USB, o en este caso, un conector USB genérico que cumple la misma función. Se usa un adaptador puerto USB serie JY-MCU.

Fig.14.Adaptador puerto USB serie JY-MCU

Para programar el Arduino Mini utilizaremos el propio software de Arduino. El software es de código abierto y funciona en Windows, Mac OS X y Linux.

Microcontrolador Arduino Mini 05	
Microprocesador	ATmega168
Voltaje de funcionamiento	5 V
Voltaje de entrada	7-9 V
Pines E/S Digital	14
Pines entrada Analógica	8
DC Corriente Continua pin E/S	40 mA
Memoria Flash	16 KB
SRAM	1 KB
EEPROM	512 bytes
Velocidad de reloj	16 MHz

Tabla 2. Características de Arduino Mini

Notas:

- No se puede alimentar el Arduino Mini a más de 9 V ni alimentarlo por error con la polaridad invertida (-9 V) pues destruiría el Arduino.
- De las 14 E/S digitales, 6 pueden ser usadas como salidas PWM (Pulse Width Modulation).
- La memoria flash es de 16 KB, sin embargo 2 KB son usados por el bootloader.
- De todos los pines de entrada/salida de los que dispone el Arduino usaremos 8 para los servos y 2 para los motores.

4.4.2. *Salidas utilizadas*

Para los servos usaremos se usarán los pines digitales. Aunque los servos utilicen PWM como señal de control y sólo dispongamos de 6 pines digitales con PWM, se pueden generar más salidas digitales como si fueran tales gracias al software interno y librerías propias de Arduino.

Para los motores usaremos 2 pines analógicos. Disponemos de una placa de control de potencia para los motores para la cual necesitamos usar una comunicación mediante I^2C en su entrada. Para conseguirlo usaremos las entradas analógicas y una librería de Arduino que se encarga de ese tipo de comunicación.

4.5. Placa de circuito impreso (PCB)

4.5.1. *Descripción de funciones de la PCB*

Se trata de una placa de circuito impreso (PCB) hecha a medida para este proyecto. Tiene como funciones servir como extensión de los pines que se van a usar para facilitar el conexionado y no realizarlo sobre el propio Arduino; realizar el conexionado del USB de Arduino en la propia placa que de otra manera habría que realizar sobre una placa de conexiones; adición de dos resistencias *pull-up* en serie con los pines que usaremos para la comunicación I^2C; añadir un condensador en el pin RESET de la comunicación USB para mejorarla y la posibilidad de alimentar tanto a Arduino como a los pines y motores, usando una única alimentación en la placa con un reductor de voltaje.

En la placa conectaremos el Arduino Mini y ya no necesitaremos trabajar directamente sobre Arduino y lo haremos sobre la propia placa y sus salidas correspondientes. Debido a

su reducido tamaño y la cantidad de conexiones que hay que realizar, sin la placa resultaría mucho más confuso para realizar las conexiones.

Así pues, esta placa resulta fundamental pues en ella se realizan todas las conexiones. Desde las conexiones de la alimentación de cada uno de los componentes, como las de los servos y la etapa de potencia de los motores y también el USB para cargar programas en Arduino Mini.

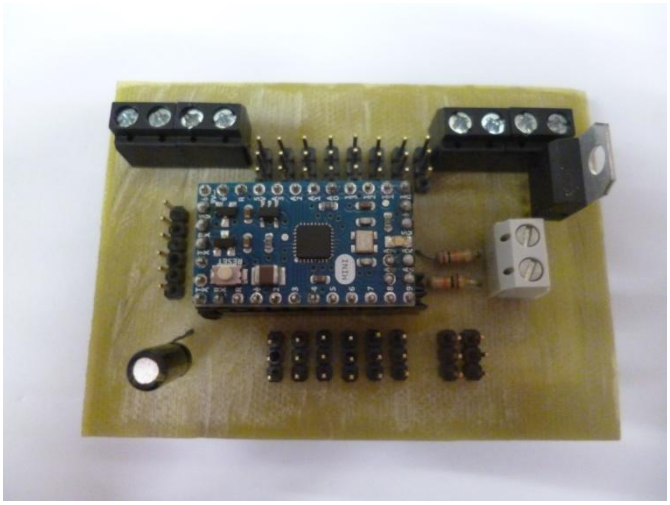


Fig.15. PCB diseñada con Arduino Mini insertado

4.5.2. *Diseño e impresión*

La impresión de la placa se realizó en el laboratorio de fabricación digital (Fab Lab) situado en la Ciudad Politécnica de la Innovación en la UPV. Se utilizó la fresadora digital Roland iModela iM-01.

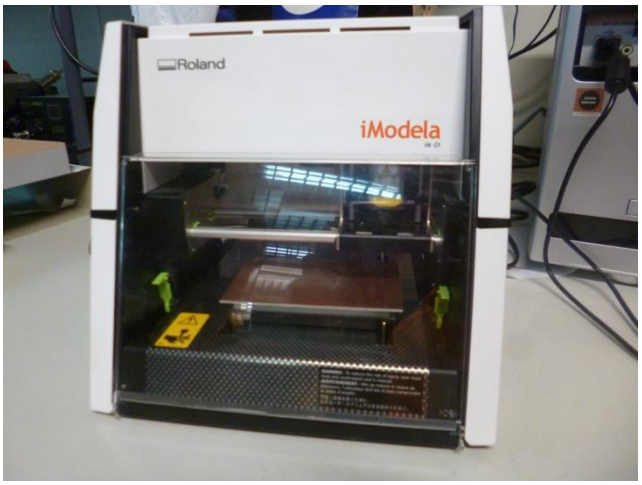


Fig.16. Fresadora digital Roland iModela iM-01

El diseño de la placa corresponde al Trabajo de Fin de Grado de Oihan Elesgaray Susierra, estudiante de GITI de la ETSII. La información de esta parte se ha extraído de su trabajo y para más detalles debería consultarse.

4.6. Placa de etapa de potencia

4.6.1. *Descripción de la placa de potencia*

Para la etapa de potencia se utiliza una placa modelo MD22, capaz de controlar dos motores de corriente continua de mediana potencia. La placa se alimenta con 5V (a 50 mA) con lo que se alimenta el circuito.

Para obtener los 15V de tensión de control del MOSFET, la propia placa contiene una bomba de carga con los que es capaz de sacar esa tensión.

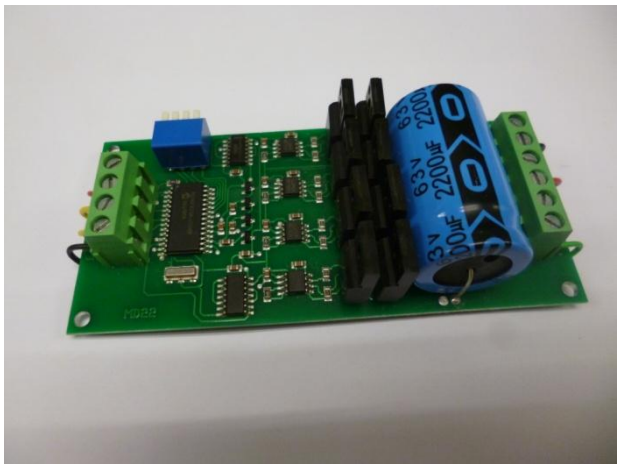


Fig.17. Placa de potencia MD22

4.6.2. *Modos de funcionamiento y registros*

Podemos trabajar con la placa en 5 modos de funcionamiento diferentes. Para el caso que nos ocupa, buscamos el modo de funcionamiento de comunicación I^2C. Para elegir entre los diferentes modos de funcionamiento, la placa cuenta con 4 interruptores y con sus diferentes combinaciones ON/OFF accederemos a todos ellos.

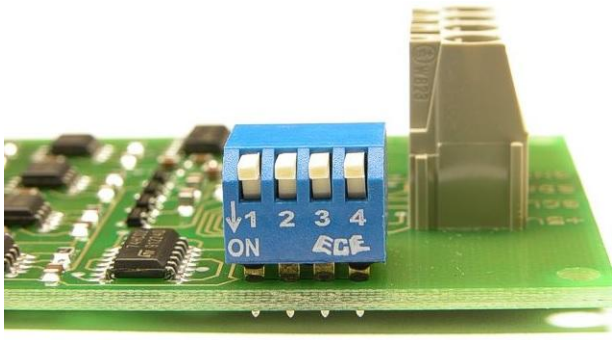


Fig.18. Switches de placa de potencia MD22

MODO DE FUNCIONAMIENTO	SWITCH1	SWITCH2	SWITCH3	SWITCH4
I^2C. Dirección Bus 0xB0	ON	ON	ON	ON
I^2C. Dirección Bus 0xB2	OFF	ON	ON	ON
I^2C. Dirección Bus 0xB4	ON	OFF	ON	ON
I^2C. Dirección Bus 0xB6	OFF	OFF	ON	ON
I^2C. Dirección Bus 0xB8	ON	ON	OFF	ON
I^2C. Dirección Bus 0xBA	OFF	ON	OFF	ON
I^2C. Dirección Bus 0xBC	ON	OFF	OFF	ON
I^2C. Dirección Bus 0xBE	OFF	OFF	OFF	ON
0V / 2.5V / 5V Analógico	ON	ON	ON	OFF
0V / 2.5V / 5V Analógico + Dirección	OFF	ON	ON	OFF
Radiocontrol	ON	OFF	ON	OFF
Radiocontrol + Dirección	OFF	OFF	ON	OFF

Tabla 3. Modos de funcionamiento de placa de potencia MD22

Se utiliza I^2C con el bus 0xB0, es decir que habrá que poner todos los interruptores en posición ON.

La placa MD22 usa los registros (en total 8 registros, del 0 al 7), especificados a continuación.

DIRECCIÓN	NOMBRE	LECTURA R/ ESCRITURA W	DESCRIPCIÓN
0	Modo	R / W	Registro de modo
1	Velocidad 1	R / W	V motor izquierdo o V lineal
2	Velocidad 2 / Giro	R / W	V motor derecho o Giro
3	Aceleración	R / W	Aceleración I^2C
4	Sin especificar	R	Devuelve 0
5	Sin especificar	R	Devuelve 0
6	Sin especificar	R	Devuelve 0
7	Versión	R	Nº software

Tabla 4. Propiedades de los registros de placa de potencia MD22

El registro de modo tiene por defecto un valor 0. El valor puede oscilar entre 0 y 3 para distintos modos de funcionamiento. Por ejemplo, cuando el registro tiene un valor 0, si los registros de velocidad, 1 y 2, tienen un valor de 0 significa que el motor girará al máximo de su velocidad pero invertida. Para 128 el motor estaría parado y el 255 sería el máximo en sentido positivo. Si eligiéramos el modo 1, para establecer una velocidad positiva máxima sería con el valor 128, 0 sería parado y -128 sería el valor máximo en sentido contrario. En este caso usaremos el modo 0.

4.7. Alimentación

En el diseño preliminar, se pretendía alimentar con 2 baterías de 12 V o 1 batería de 24 V, los motores y aquellos elementos que se pudieran alimentar a dicha tensión. Por motivos de tiempo y disponibilidad, finalmente los ensayos se realizaron con una fuente de alimentación regulada. Además también se aprovechó la alimentación por USB del Arduino para conectarlo, cargarle programas y testearlo.

5. Desarrollo de la solución propuesta

5.1. Diseño del entorno de simulación virtual

5.1.1. *Guía de configuración de objetos de la simulación*

A partir de los ficheros STL usados para imprimir el robot, se puede empezar a construir la simulación. Lo primero sería importar dichos ficheros STL en V-REP para empezar a trabajar con ellos (se han añadido los objetos servo respecto a los archivos que se imprimen).

La siguiente parte del proceso sería orientar y colocar los objetos de manera que formen el robot tal y como debería ser, por comodidad y por facilidad en los pasos siguientes, usando los botones "Object/item shift" y "Object/item rotate" situados en la barra de herramientas superior.

A continuación se definiría si los objetos importados son estáticos (*Static*) o no. Los objetos estáticos son aquellos que no pueden ser directamente actuados, si no que su posición y orientación dependen directamente de otros objetos. Los objetos que no sean estáticos, serán dinámicos y son los objetos que el motor de simulación dinámica de V-REP tratará. Para estos objetos hay que añadir su masa una vez se establezca que no son estáticos.

Para distinguir entre objetos estáticos y dinámicos lo que hay que hacer es ir al cuadro "Scene hierarchy", donde, entre otras cosas, están todos los archivos que se han importado anteriormente. Para los eslabones y el cabezal de la pinza se establece que no son estáticos y se añaden sus masas (que puede ser de manera aproximada).

Para poder centrar las articulaciones del robot se tienen que crear unas nuevas geometrías, que corresponderían a los ejes de los servos. Para extraer dichas geometrías, primero habría que entrar en el modo "Shape edit", al que podemos acceder apretando click derecho encima de la ventana de trabajo y luego "Edit > Edit modes > Enter shape

edit mode" (se debe estar seleccionando el servo o eslabón adecuado mientras realizamos esta operación).

Con ello se generaría un mallado en la pieza seleccionada, de la cual se seleccionaría la parte de interés de su eje, manteniendo la tecla "shift" y clicando con el botón izquierdo las partes que queramos. Una vez seleccionadas se clica en la opción "Extract shape" y con ello se habrá generado una nueva geometría.

Y no sólo se realizará este proceso con los ejes de los servos, también en la pinza en cada una de las inserciones mecanizadas donde se unen los dedos y barras de la pinza.

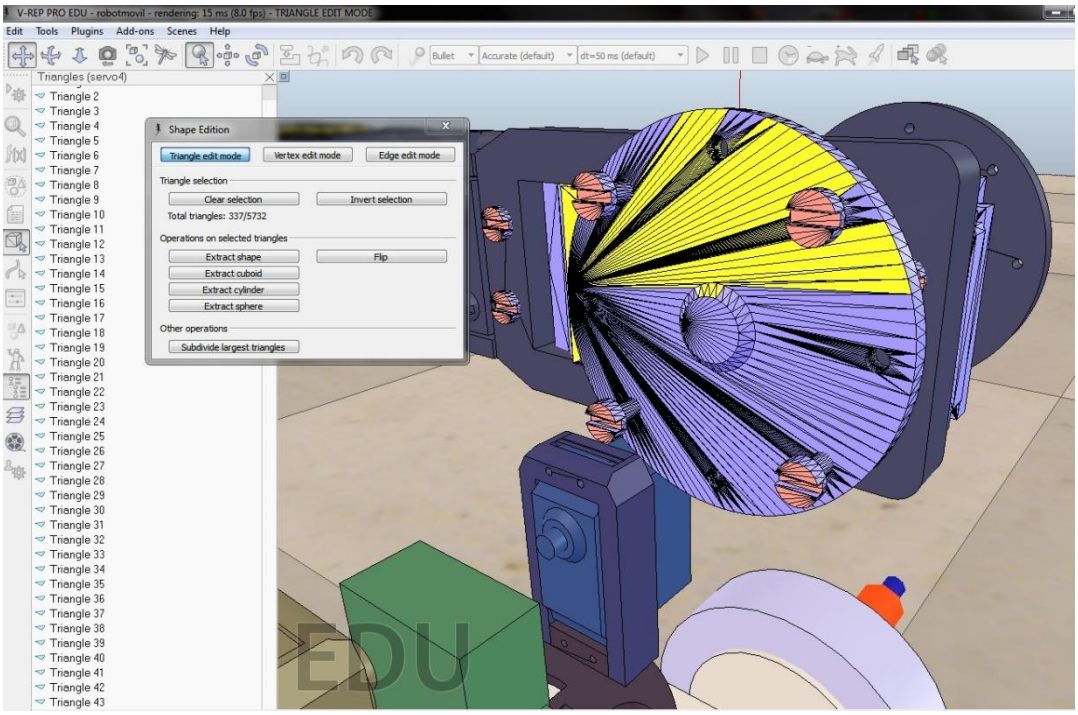


Fig.19. Vista en V-REP de modo Shape edit

El siguiente paso sería crear las articulaciones, en nuestro caso serían de revolución. Para ello hacemos click derecho encima de la ventana de trabajo, luego "Add > Joint > Revolute".

A continuación habría que posicionar correctamente las articulaciones. Para ello seleccionamos la articulación, y apretando "shift" se hará click izquierdo sobre el objeto al que van emparentadas, en este caso los ejes anteriormente creados. Apretando el botón "Object/item rotate" abriremos el cuadro de diálogo donde clicaremos la opción "Apply to selection". Una vez posicionadas todas, se puede prescindir de las geometrías de los ejes y mantener tan sólo las articulaciones.

Por defecto, el programa crea las articulaciones en modo Par/Fuerza, y así estarán todas, exceptuando las de la pinza que no hayamos asociado a ningún eje de servo. Así que, la articulación simétrica a la que sí contiene un servo le asociaremos una relación de dependencia con ésta, no obstante están engranadas y se moverá de manera simétrica a ella.

Para asociarle una relación de dependencia se clica en la lista de la izquierda, "Scene hierarchy", y sobre el icono de la articulación correspondiente se hace doble click para abrir el menú "Scene object properties". Una vez en ese menú en el subapartado "Joint mode" hay un desplegable donde se selecciona el modo adecuado, en este caso "joint is in dependent mode". Habrá que establecer una ecuación de dependencia, como en este caso es un movimiento simétrico del que depende de la otra articulación será "Joint position=+0.0 -1.0x" y se selecciona en el desplegable la articulación simétrica de la que depende.

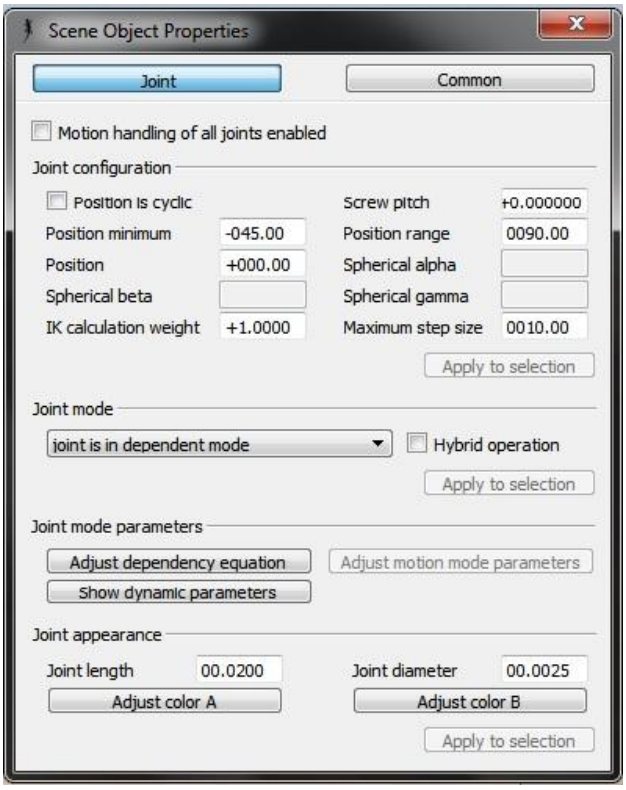


Fig.20. Vista en V-REP de cuadro Scene Object Properties

Para el resto de articulaciones de la pinza que no lleven asociadas ningún servo, ni sea la anteriormente mencionada, accediendo al mismo menú se selecciona "joint is inverse kinematics mode", para que sea el propio V-REP el encargado de calcular su posicionamiento.

Otra característica importante a añadir en las articulaciones es su rango de posicionamiento, que en las que estén unidas a un servo será el rango del propio servo. Para acceder al menú para insertar dichas propiedades, simplemente realizamos el mismo procedimiento que hemos utilizado antes, accediendo al menú "Scene object properties". Una vez allí, tenemos que asegurarnos que "Position is cyclic" está desmarcado y junto a dicha opción encontramos "Position range" donde incluimos el rango del servo unido a dicha articulación. Como suponemos que los servos están centrados de manera simétrica, en "Position minimum" se pondrá la mitad del valor de "Position range".

Para las articulaciones que no contienen ningún servo, se deja la opción cíclica marcada al no estar limitadas por ningún mecanismo y será el propio programa el que gestione su movimiento. Para el resto de articulaciones habrá que establecer también el máximo par o fuerza, que en cada caso sea capaz de proporcionar el servo, teniendo en cuenta las tablas de los servos incluidas en el apartado correspondiente, y teniendo también en cuenta que no se planea alimentarlos con más de 5 V.

Para fijar los valores de los pares y fuerzas, se entra al mismo menú que las anteriores veces, clicando esta vez en "Show dynamic parameters". En la siguiente ventana se marcará "Motor enabled" y a la derecha se añaden los valores de fabricante de los servos requeridos.

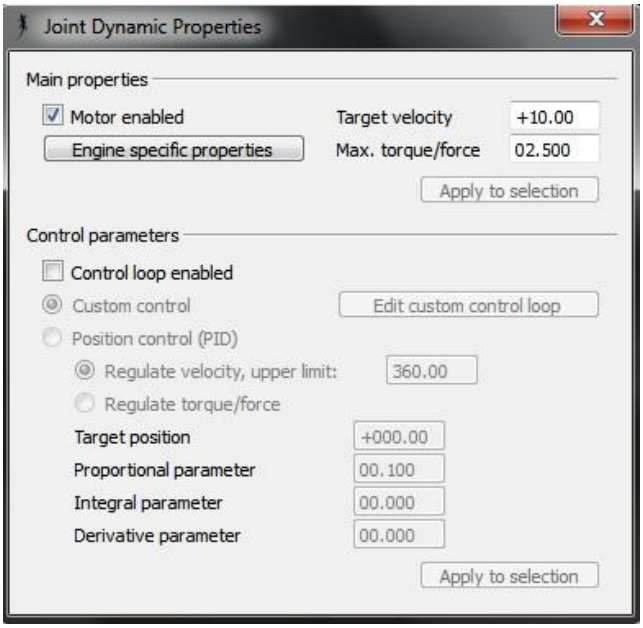


Fig.21. Vista en V-REP de cuadro Joint Dynamic Properties

Las ruedas son objetos que pueden o no ser impresos, en este caso se cogieron dos ruedas de modelismo de caucho y se utilizaron en el diseño. Como no se imprimieron, hay que crear las ruedas en V-REP porque no han sido importadas. Para ello se hace click derecho "Add > Primitive shape > Cylinder", con lo que haremos un objeto similar a una rueda.

Habrá que definir el objeto creado como uno dinámico, es decir, no estático. Es importante hacerlo, porque si se deja en estático puede haber problemas a la hora de la simulación, pudiendo incluso inutilizar toda la simulación.

Para dotar el movimiento de las ruedas se sigue el mismo procedimiento que para dotar de movimiento a las articulaciones. Se crea una articulación por rueda, centradas en el centro de las mismas y se añaden propiedades de movimiento marcando la casilla "Motor enabled" e introduciendo el valor máximo del par producido por el motor. También hay que marcar la casilla de "Position is cyclic" en "Joint properties", pues el eje del motor puede girar tantas veces como quiera sin verse limitado en su rango de giro.

La base del robot se define también como objeto dinámico. Y se debe marcar en "Object special properties" del menú "Scene Object Properties / Common" las casillas "Collidable", "Renderable", "Detectable" y "Measurable". Es un paso importante, si no se marcaran el objeto no podría correr en la simulación pues al hacerlo desaparecería del espacio de trabajo al no considerarse un "objeto físico". Es recomendable también hacerlo con las ruedas, pero puede funcionar sin incluirlo.

Para la tercera rueda de la base, se usa un tipo de especial de objeto llamado "Force sensor", se añade como cualquier otro objeto. Dentro de este objeto se crea otro, con "Primitive shape" y con forma de esfera, y al que se define como "Collidable", "Renderable", etc. justo como en el resto de los objetos.

El paso más importante para el correcto funcionamiento de la simulación, viene dado por las relaciones de parentesco que se establezcan entre los distintos objetos que componen el robot. Si no se hace correctamente este paso, se provocarían anomalías en el funcionamiento de la simulación puesto que no se movería como si fuera un solo ente.

Las relaciones de parentesco pues, implican una relación de dependencia de unos componentes a otros. Por ejemplo, dentro del cuerpo principal del robot, el centro de su base, incluiríamos otro nivel de parentesco con las articulaciones de los motores, la tercera rueda y el brazo robot completo.

Dentro de la articulación de cada rueda, estaría incluido en un nivel de parentesco inferior, la rueda a la que dota de movimiento. Y en el caso de la tercera rueda, estaría en el nivel inferior el objeto esfera que representa al objeto real.

En cuanto al brazo robot, dentro de la articulación inferior que permite al brazo robot rotar, estaría incluido el eslabón al que mueve con dicha articulación y también al resto de articulaciones en subniveles de parentesco inferiores. De igual manera funciona para los objetos que definen a los servos, que estaría incluidos dentro de los eslabones correspondientes, en un nivel inferior.

Para explicarlo de otra manera, la relación de parentesco influye en la dependencia e independencia de los movimientos de unas articulaciones o eslabones con respecto a otras. Por ejemplo, que la cabeza de la pinza se mueva no afecta para nada al resto de articulaciones. Sin embargo, si la base rota, el brazo entero rotará al estar éste incrustado en la base.

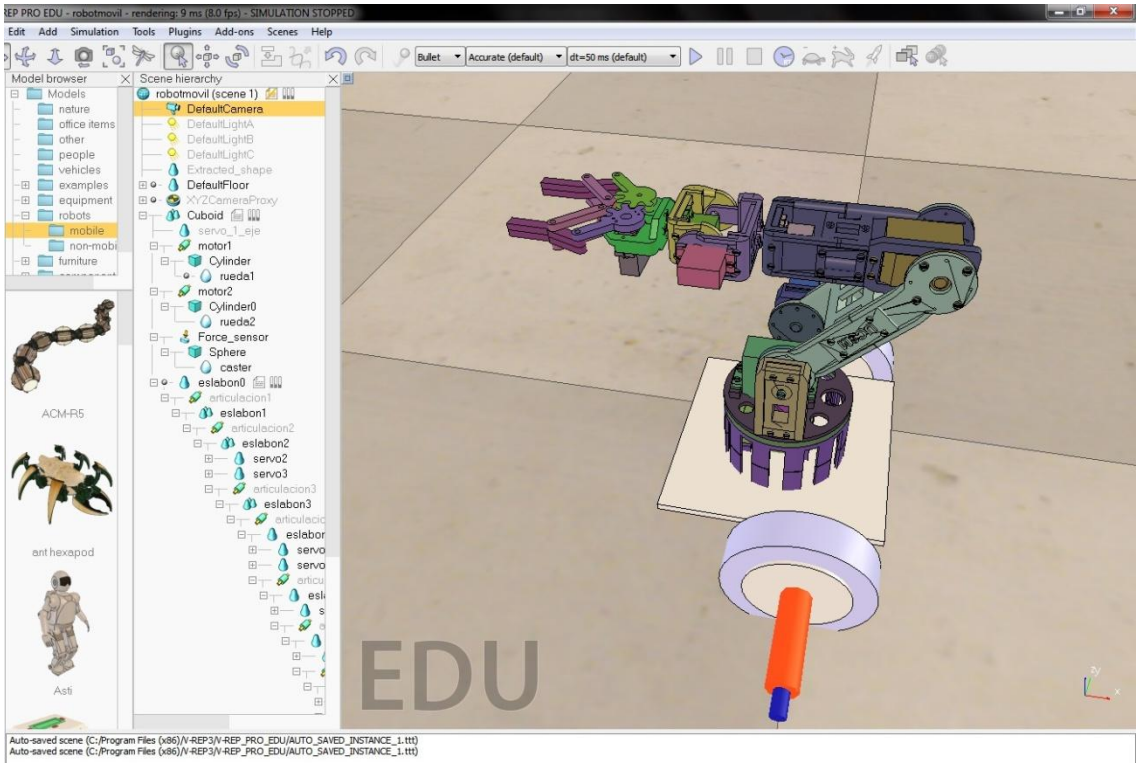


Fig.22. Vista en V-REP con configuración de objetos realizada

5.1.2. *Creación de interfaces de usuario y de sliders*

Una vez configurado con todo detalle el robot dentro de V-REP, faltaría añadir unos *sliders* con los cuales modificar los valores de posición de los servos y velocidad de los motores mientras se está ejecutando la simulación.

Para añadir interfaces de usuario, hay que ir al menú de edición y en modos de edición buscar "Enter user interface mode". También se puede asociar dicha interfaz con un objeto, de manera que cuando se copie dicho objeto en otra escena diferente, también lo haga la interfaz. Para hacerlo, en el menú desplegable "UI is associated with" seleccionamos el objeto al que queramos asociarlo.

Para crear títulos y *sliders* dentro de la interfaz, se selecciona un grupo de cuadrículas de la parrilla de la interfaz y se selecciona la opción "Insert merged button" de manera que el ocupe toda una fila en lugar de tan sólo un cuadrado.

Ahora se elegiría el tipo de botón, bien sea o un propio botón, o una etiqueta de texto o un slider. En nuestro caso se genera una etiqueta en la parte superior de cada slider, indicando el servo que mueve. Es importante ver el "Button handle" asociado a cada slider, pues luego será de utilidad.

Se puede configurar en profundidad la apariencia de interfaz y *sliders*, en tamaño, colores, fondos, texturas, rellenos, etc. Sin embargo no se considera de importancia para el trabajo y por ello se omite.

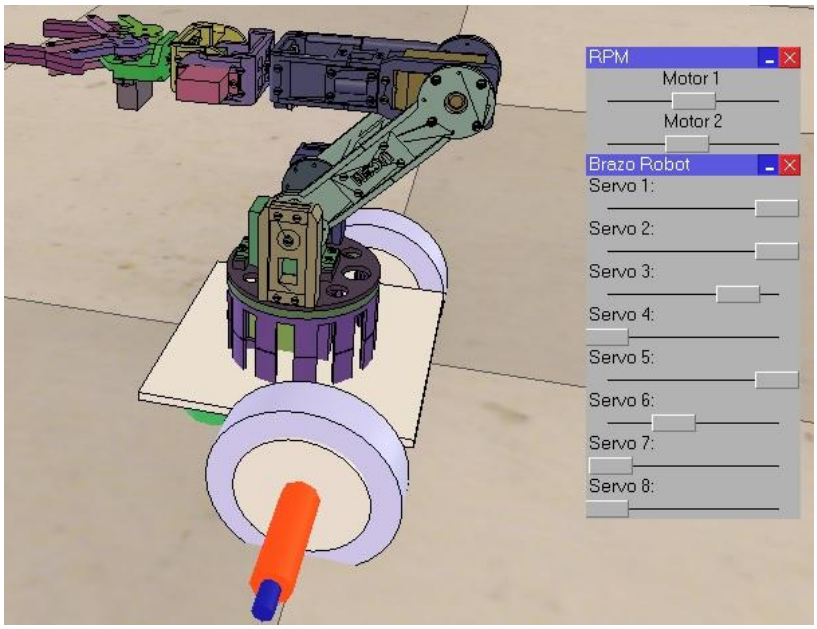


Fig.23. Vista en V-REP con interfaces de usuario con sliders añadidos

5.1.3. *Scripts asociados a interfaces y sliders*

Una vez configurado visualmente la apariencia de la interfaz y definido el número de *sliders* que se usarán, falta asociar dichos *sliders* a un movimiento dentro del robot. Para conseguirlo hay que usar los *scripts* de los que disponemos en V-Rep.

El botón para consultar los *scripts* aparece en la barra de herramientas izquierda del programa, con un icono similar a una hoja de papel. Una vez se accede al menú se puede consultar la información de los *scripts* existentes y se pueden crear nuevos.

Existen 2 tipos de *scripts* los "Main Scripts" y los "Child Scripts", dentro de estos últimos existe una diferenciación, "Child Scripts (threaded)" y "Child Scripts (non-threaded)". El *Main Script* se recomienda no utilizarlo (ni modificar el ya existente por defecto) porque define los valores por defecto de la simulación y el arranque de los *Child Script*.

Para el caso que nos ocupa, usaremos un *Child Script (non-threaded)* y se asocia al objeto con la relación de parentesco más alta. Una vez creado el *script* hay que configurarlo para que al insertar un valor en el *slider*, se traduzca en un cambio de velocidad o posición en el servo o motor correspondiente.

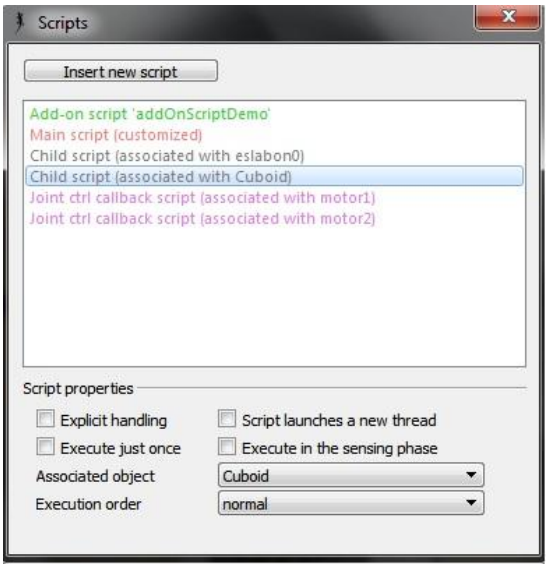


Fig.24. Vista en V-REP del cuadro Scripts

Haciendo doble click en cualquier *script* se accede a su código. Generalmente, cuando se crea un nuevo *script*, el propio programa ofrece un pequeño ejemplo a modo de guía para comprender el funcionamiento del *script* y cómo se puede interactuar con la interfaz anteriormente generada.

Los comandos fundamentales para usar en el *script* son:

- ***simGetUIHandle("nombre_interfaz"):*** Devuelve el *handle* (identificador) de la interfaz.
- ***simGetObjectHandle("nombre_objeto"):*** Devuelve el *handle* correspondiente a un determinado objeto.

- ***simGetUISlider(handle_interfaz, handle_slider):*** Devuelve el valor del *slider* de una interfaz, que generalmente varía entre 0 y 1000. El valor del handle_slider puede conocerse desde el menú edición de la interfaz de usuario.
- ***simSetJointTargetVelocity(handle_objeto, velocidad):*** Hace que el objeto cuyo *handle* se especifica, adquiera una determinada velocidad. Utilizado para las articulaciones que representan los motores.
- ***simSetJointPosition(handle_objeto, posicion):*** Se modifica la posición de un determinado objeto. Se usa para las articulaciones que representan el posicionamiento de los servos.

Primero se obtendrían el *handle* de la interfaz con la que estemos trabajando. Se podría hacer todo en una única interfaz, sin embargo, en este trabajo se han usado dos interfaces diferentes con sus dos *scripts* correspondientes.

A continuación se almacenan los *handles* de los objetos que se van a modificar, en este caso las articulaciones que están con los servos, y se almacenan en variables para después usarlas (***Anexo 8.2***).

Deberíamos conocer tanto mínimos y máximos como de las velocidades de los motores, como de las posiciones de las articulaciones. Con ello se operará según un modelo lineal (***Anexo 8.3***), en el que si se introduce el valor mínimo del *slider*, la salida será el valor mínimo. Si por el contrario introducimos el máximo en el *slider*, obtendremos en la salida el máximo, sea para posición o para velocidad.

Finalmente, con los datos operados, se procede a utilizar el comando que modificaría la velocidad o posición del objeto correspondiente, a un valor ya calculado dependiendo del valor del *slider*.

Después de todos estos pasos, ya estaría el robot definido al completo para ser simulado bajo cualquier condición que se deseara de posición de articulaciones (servos) y

velocidad de las ruedas (motores), pudiendo modificar los valores incluso mientras se ejecuta la simulación.

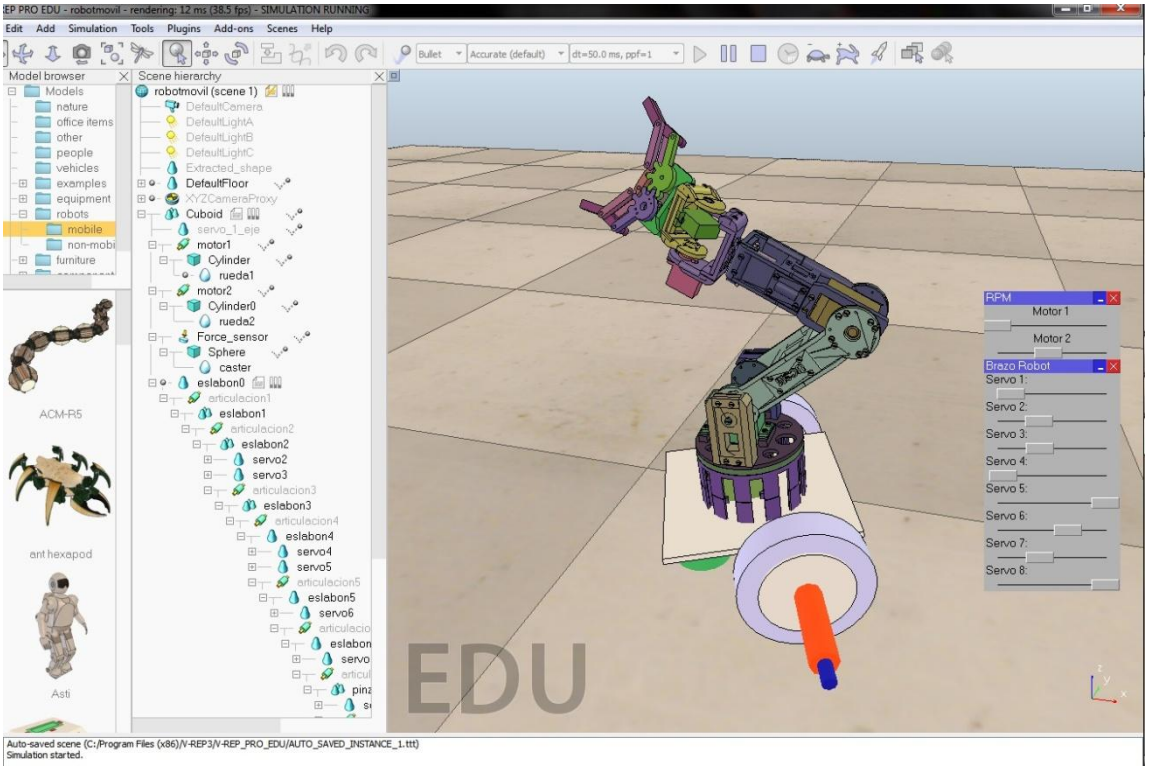


Fig.25. Vista en V-REP durante la simulación después de toda la configuración

5.2. Desarrollo del plugin

5.2.1. *Planteamiento inicial*

Para su creación se parte del esqueleto del *plugin* que encontramos dentro de los ficheros de la instalación del propio V-REP, que el Dr. Marc Andreas Freese de Coppelia Robotics deja para su libre distribución.

Para encontrarlo hay que buscar dentro del directorio de instalación del programa la subcarpeta "programming", y en su interior, "v_repExtPluginSkeleton". En esa carpeta se encuentran los distintos archivos necesarios para la integración del *plugin*, incluido el fichero proyecto en C++ con el que se trabaja.

Recibe el nombre de "esqueleto" porque si se compilara el proyecto tal y se cargara, funcionaría pero no realizaría ninguna función. Falta, por tanto, realizar el resto de la programación para que desempeñe la función para la cual se está desarrollando. El autor incluso añade algunas anotaciones en su código para ayudar a su comprensión y a integrar las partes donde correspondan.

5.2.2. *Programación del plugin*

El primer paso sería la inclusión de librerías que se vayan a utilizar. La librería <*iostream*> añade operaciones de entrada/salida, <*sstream*> para operar *streams* con *strings*, <*windows.h*> y <*windows.lib*> contiene declaraciones de todas las funciones en el API de Windows.

A continuación se incluyen algunas funciones que no son totalmente necesarias pero sí son útiles, como el aviso por pantalla de si existe algún error y qué error es concretamente. Bien que no se ha podido cargar porque no se dispone de todas las funciones requeridas para iniciar el *plugin* o si la versión del V-REP no está actualizada y produce ese problema de incompatibilidad.

El siguiente paso sería inicializar algunas de las variables que se van a usar, que podría realizarse más tarde pero se realiza al principio para evitar olvidar alguna importante. Seguido por la obtención de los *handle* de las interfaces para luego poder utilizar

comandos que necesiten de ellos. Generalmente serán aquellos comandos para averiguar los valores de los *sliders*.

El último paso de la inicialización será la apertura del puerto serie, realizado con la función *CreateFile*, en el cual abriremos el puerto COM1 del PC, que posteriormente utilizaremos también en la programación de Arduino.

Después de la inicialización, aparece la rutina de para el *plugin*, en la cual se incluye el cierre del puerto COM1 abierto anteriormente, mediante la función *CloseHandle*.

En la versión original aparecen numerosas funciones a las que se hacen llamadas en distintos momentos de su ejecución. Por ejemplo, justo el momento anterior de iniciarse la simulación, justo el momento antes de finalizarla, así como una vez finalizado y una vez acabado. Sin embargo, se puede prescindir de la mayoría de ellas para el caso que nos ocupa.

El bucle principal será el que se ejecuta mientras dura la simulación y en él se realizará el proceso más importante del *plugin* que es obtener los valores de los *sliders*, crear el *stringstream* que incluirá esos valores y mandarlo por el puerto serie creado en la inicialización.

Al final del proceso, se manda una cadena con los valores de los *sliders*, en el que también están incluidos los valores identificadores de cada servo o motor precediendo dicho valor. Y al final un valor del tiempo de espera recomendado entre lecturas que haya que hacer, para evitar solapamientos.

```
for (i = 0; i < 8; i++){
    WF[i] = simGetUISlider(UI_id, (2 * i + 4));
}

for (i = 8; i < 10; i++){
    WF[i] = simGetUISlider(motor_id, (2 * i + 8));
}

for (i = 0; i < 10; i++){
    SS << "#" << i << "P" << WF[i];
}

SS << "T" << 100 << "\r\n";


WriteFile(ScomPort, pSS, (sizeof(SS)), NULL, NULL);
```

Fig.26. Código en Visual Studio de creación y envío de cadena

Para explicar el formato de la cadena enviada, se supone por ejemplo que se ha enviado la siguiente cadena:

#1P600#2P800#3P500#4P1000#5P400#6P200#7P0#8P250#9P600#10P100T100\r\n

El valor que sigue a "#" indica el valor identificador del componente dentro de la cadena, como se ve en la tabla inferior en la columna "ID de cadena". Se escribe porque después se querrá descifrar la cadena para enviarle órdenes al servo o motor, y necesitamos saber de qué parte se ha obtenido dicha lectura.

NOMBRE	INTERFAZ	ID DE CADENA	HANDLE SLIDER
Servo 1	UI	1	4
Servo 2	UI	2	6
Servo 3	UI	3	8
Servo 4	UI	4	10
Servo 5	UI	5	12
Servo 6	UI	6	14
Servo 7	UI	7	16
Servo 8	UI	8	18
Motor Derecho	motor	9	24
Motor Izquierdo	motor	10	26

Tabla 5. Identificación de componentes para programación

A continuación "P" separa el valor del identificador del componente en la cadena, del valor obtenido del *slider* de ese componente. Cabe recordar que los *sliders* tienen un valor mínimo de 0 y un valor máximo de 1000.

Sucesivamente de esa manera se van enviando todos los valores en la cadena, hasta que llega al final, donde distinguimos el final gracias al "T". Justo detrás aparece el valor, por lo general 100, que indica el tiempo en milisegundos de espera entre las distintas lecturas que se realizarán, de la cadena.

Así por ejemplo "#2P800" indica que el valor del *slider* del Servo 2, obtenido de la interfaz UI, es de 800. Será después de la recepción de datos donde se operará para convertir el valor de 800 del *slider*, en un valor real de posición del servo. De igual manera si fuera el valor del *slider* de un motor, la diferencia sería que no indicaría posición sino velocidad.

En el ***anexo 8.4*** se puede encontrar el código completo del *plugin* para su consulta.

5.3. Programación en Arduino

5.3.1. *Inicialización*

Con el software de Arduino debemos conseguir, en primer lugar, recibir e interpretar la información que llega por la cadena a través del puerto serie. Una vez descifrada la cadena, se puede operar con sus datos para ordenar a servos y motores que se coloquen en cierta posición o vayan a una determinada velocidad.

Para empezar se incluyen las librerías que se utilizarán y se definen unos valores constantes que servirán como direcciones.

El primer paso dentro de la función *setup* será definir las variables (variables enteras, variables servo y vectores *float*) e inicializar algunas de ellas. Para el caso de los servos, también se unirá cada servo a su salida para que al darle la orden a un servo de escribir cierto valor a su salida, Arduino sepa cuál es esa salida. La numeración es simple, pues coincide el número del servo con su salida.

El último paso de la función *setup* será iniciar la comunicación I^2C con la que nos comunicaremos con los motores, y además iniciar la comunicación con el puerto serie del que recibiremos datos. Es importante comprobar que Arduino Mini esté conectado al mismo puerto serie por el que se envían los datos, en el caso presente, el puerto COM1.

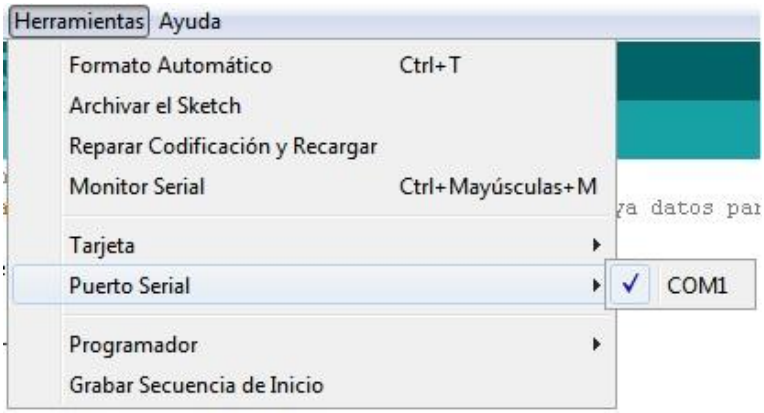


Fig.27. Comprobación de puerto serie en Arduino

5.3.2. *Bucle de lectura*

La siguiente función, la principal del programa se ejecutará constantemente como un bucle a diferencia de la función *setup* que se ejecuta una única vez.

El primer paso de esta función es leer la cadena para obtener de ella los datos que interesan, es decir, las posiciones (o velocidades) de los servos (o motores), siempre teniendo en cuenta que el valor estará comprendido entre 0 y 1000, pues estamos leyendo valores de *sliders*.

Para entender cómo se realiza el bucle de lectura, se ilustrará con un ejemplo. Supongamos la cadena recibida:

#1P600#2P800#3P500#4P1000#5P400#6P200#7P0#8P250#9P600#10P100T100\r\n

Primero, se leería el *string* hasta encontrar un "#", el cual no leeríamos y descartaríamos. Por lo tanto en la primera iteración lo que hacemos es descartar el caracter "#" que no nos aporta nada útil, aunque sirve para situarnos dentro de la cadena.

La cadena resultante (que no la guardada), quedaría como la recibida a diferencia del primer "#" que ya no aparecería. El segundo paso sería leer hasta encontrar el caracter "P", y este trozo quedaría almacenado en la variable *in_string*. La variable *in_string* estaría en este momento almacenando "1".

La cadena no entiende "1" como valor entero, con lo cual, antes de guardar el valor del identificador del componente tendremos que usar la función *.toInt( )* para transformar la cadena almacenada a un valor entero, y éste sí lo almacenaremos en un vector.

La cadena que quedaría por leer en estos momentos sería:

600#2P800#3P500#4P1000#5P400#6P200#7P0#8P250#9P600#10P100T100\r\n

A continuación se leería de nuevo hasta encontrar "#", con lo cual se obtendría la cadena "600". Realizando el mismo proceso que anteriormente para convertirlo a entero, finalmente almacenaríamos el valor en otro vector.

Gracias a que antes se ha leído el valor del identificador, se conoce el componente del que se obtiene la información y además se guarda dentro del vector en una posición determinada.

La cadena después de esta última lectura quedaría:

2P800#3P500#4P1000#5P400#6P200#7P0#8P250#9P600#10P100T100\r\n

Debido a que sólo en la primera iteración buscamos primero hasta "#", no se haría en esta parte. Sí leeríamos de la cadena hasta "P", y seguiría el mismo procedimiento que anteriormente se ha explicado hasta que acabemos de leer toda.

Es una caso especial si identificamos el valor "10", correspondiente a la lectura del motor izquierdo. En este caso para averiguar el valor de su *slider* no se puede seguir leyendo de la cadena hasta encontrar "#", pues el valor que lo delimita es el caracter "T".

Una vez almacenados los valores de los *sliders* en la posición que le corresponde a cada uno dentro del vector dependiendo del componente al que se refieran, habrá que obtener el valor real que necesitamos enviar mediante las funciones, a servos y motores.

5.3.3. *Bucle de escritura*

Para obtener los valores reales que habrá que introducir en las funciones de escritura basta con entender el funcionamiento de dichas funciones además de conocer el rango de valores en el que trabaja cada uno de los elementos.

Por ejemplo, la función que se utiliza para posicionar los servos acepta valores entre 0 y 180, siendo estos los valores reales en ángulos. El valor 0 será girar 90º en sentido antihorario, partiendo desde la posición intermedia, y el valor 180 será girar 90º en sentido horario. Lógicamente el valor 90 será la posición intermedia, y si el servo está centrado, significa que no girará.

Además, no todos los servos trabajan con el mismo rango de posicionamiento. Los servos Futaba S3003 disponen de un rango de 60º mientras que el resto dispone del rango de 180º, detalle que hay que tener en cuenta.

Para los motores, debido al modo de funcionamiento que se configuró en la placa de potencia, se aceptarán valores entre 0 y 255. Dichos valores corresponden a velocidad máxima en sentido negativo (0) y velocidad máxima en sentido positivo (255). Como se puede suponer el valor intermedio (128) corresponde al estado de motor detenido.

No obstante, para que funcione de acuerdo a la simulación se ha optado por elegir 128 como la posición de reposo, 255 como velocidad máxima (sentido positivo) pero se han obviado las velocidades entre 0 y 128 que supondrían el giro en sentido inverso del motor. Esto es debido a que en la simulación, ambos motores funcionan tan sólo en sentido positivo.

El último paso sería utilizar las funciones de los servos y motores para enviarles los últimos valores reales obtenidos, con los que finalmente se podrían replicar el movimiento generado en la simulación virtual.

Cerrando el programa estaría el *delay*, marcado por el valor que tuviera dentro de la cadena recibida, en milisegundos.

Se puede consultar el código completo desarrollado para Arduino en el ***anexo 8.5***.

5.4. Mejoras propuestas

Por motivos principalmente de tiempo, no se pudo hacer todo aquello que se quería haber incluido en el trabajo y se relatan en este apartado, como posible continuación de este trabajo. O bien completando el trabajo pensado inicialmente, o siendo una propuesta alternativa de desarrollo del mismo.

- ***Control del robot mediante periférico externo:*** Se pensaba añadir como complemento al control mediante la interfaz virtual, un periférico externo de tipo joystick con el que se pudiera controlar el robot.

Fig.28. Joystick Thrustmaster USB como posible periférico de control

- ***Módulo wireless o bluetooth:*** Para completar el trabajo, habría que añadir la comunicación mediante uno de estos protocolos de comunicación remota, pues la idea es que el robot sea móvil y se pueda manejar desde el propio ordenador personal y no sea necesario estar cerca.

6. Soluciones alternativas

Se obvian aquellas soluciones que incluyan la misma solución que la propuesta pero con un software distinto, pues podríamos estar hablando de multitud de soluciones diferentes debido al amplio catálogo de software.

Nada más en posibles lenguajes de programación se podría hablar de Java, Python, Ruby, Matlab y Octave, por nombrar algunos. También en simulación de robots hay múltiples programas que se podrían haber usado debido a que muchas empresas desarrollan el suyo propio. Por ejemplo RobotExpert de Siemens y KUKA.sim de KUKA Robotics.

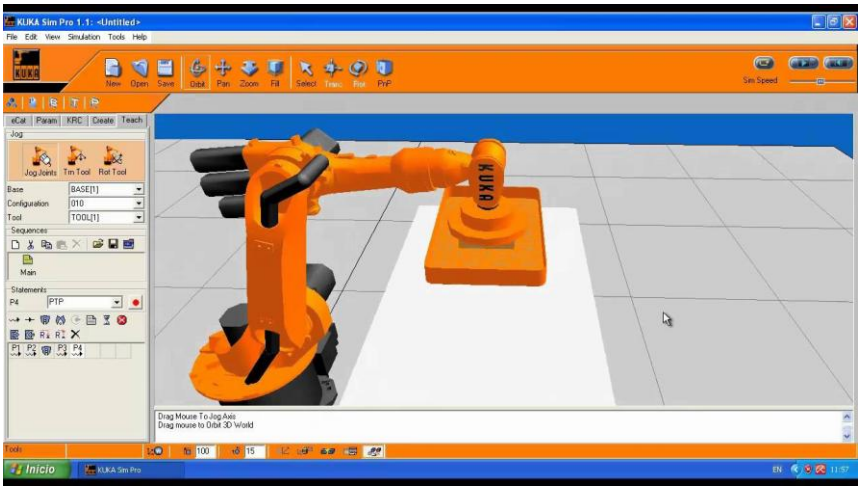


Fig.29. Software KUKA.sim de KUKA Robotics

Al igual también, que se podría hablar de sustituir Arduino por una plataforma que ofrezca lo mismo como Raspberry Pi.

Fig.30. Ordenador de placa reducida Raspberry Pi

No obstante, es preferible nombrar también algunas soluciones que difieran en planteamiento de la solución propuesta, para aportar un enfoque diferente a la resolución del problema.

Por ejemplo, en lugar de un *plugin* se podría haber implementado directamente uno de los "Child script" de los que dispone V-REP. Es la manera más directa puesto que los objetos de la simulación pueden estar directamente unidos al *script*. Por contra, no se puede elegir el lenguaje de programación que se desee, puede que el código no esté optimizado para obtener la mayor velocidad y que no se puede acceder directamente a librerías externas.

Otra posible opción sería controlar el robot mediante una aplicación cliente externa basada en la API (*Application Programming Interface*) remota. O bien vía un nodo ROS (*Robot Operating System*) que permite conectar virtualmente cualquier número de procesos entre ellos y dispone de una gran cantidad de librerías.

Podría controlarse también con una aplicación externa que se comunique vía V-REP *script*. Las principales ventajas es que se podría elegir cualquier lenguaje de programación que se desee puesto que es una aplicación totalmente externa al propio programa, pero siendo ésta una manera más complicada de conseguir el objetivo.

7. Bibliografía

1. ***Coppelia Robotics:*** Página de la empresa desarrolladora de V-REP donde se puede consultar cualquier información relativa a su programa. http://www.coppeliarobotics.com/

2. ***Microsoft Developer Network:*** Página web de Microsoft en la que se puede consultar información relativa a herramientas de Microsoft, como Visual Studio, y programación.
   http://msdn.microsoft.com/

3. ***Cplusplus:*** Página web dedicada a la programación en C++. Se puede encontrar todo tipo de tutoriales, artículos y comandos de este lenguaje de programación.
   http://www.cplusplus.com/

4. ***C++Reference:*** Página web dedicada a la programación en C++. Se pueden encontrar las librerías más comunes en C++ y las funcionalidades que ofecen. http://en.cppreference.com/

5. ***Arduino:*** Página web oficial donde podemos encontrar toda la información relativa a placas Arduino así como al respecto del software (programación y librerías).
   http://www.arduino.cc/

6. ***Servo Database:*** Página web que ofrece especificaciones detalladas, análisis y comparativas de la mayoría de servos a la venta en el mercado. http://www.servodatabase.com/

7. ***RSAmidata:*** Página web de distribuidor de componentes electrónicos, eléctricos e industriales, donde se puede consultar información detallada de un gran número de ellos.
   http://es.rs-online.com/web/

8. Anexos

8.1. Guía de software

- **V-REP (Virtual Robot Experimentation Platform)**
  - **Desarrollador:** Coppelia Robotics
  - **3 licencias:** Comercial, de estudiante y versión gratuita
  - **2 licencias gratuitas:** de estudiante y versión gratuita
  - **Sistemas operativos:** Windows, Mac OS X y Linux
  - **Requisitos del sistema:** No especificados
  - **Descarga:** http://www.coppeliarobotics.com/downloads.html
  - **Utilizado en este trabajo:** Licencia de estudiante de UPV en Windows

- **Microsoft Visual Studio 2013**
  - **Desarrollador:** Microsoft
  - Dispone de **versión de prueba gratuita durante 90 días**
  - Versión Visual Studio Professional sin coste* para estudiantes (*consultar disponibilidad en página web)
  - **Requisito imprescindible:** Windows OS
  - **Requisitos del sistema:** No especificados
  - **Directorio de descargas:** http://www.visualstudio.com/es-es/downloads
  - **Utilizado en este trabajo:** Visual Studio Express 2013 (versión prueba gratuita)

- **Arduino**
  - **Desarrollador:** Arduino
  - Totalmente **gratuito**
  - De **código abierto**
  - **Sistemas operativos:** Windows, Mac OS X y Linux
  - **Requisitos del sistema:** No especificados
  - **Descargas:** http://arduino.cc/en/Main/Software
  - **Utilizado en este trabajo:** Arduino 1.0.5-r2

8.2. Código de los scripts

8.2.1. *Código del script asociado a la interfaz de los servomotores*

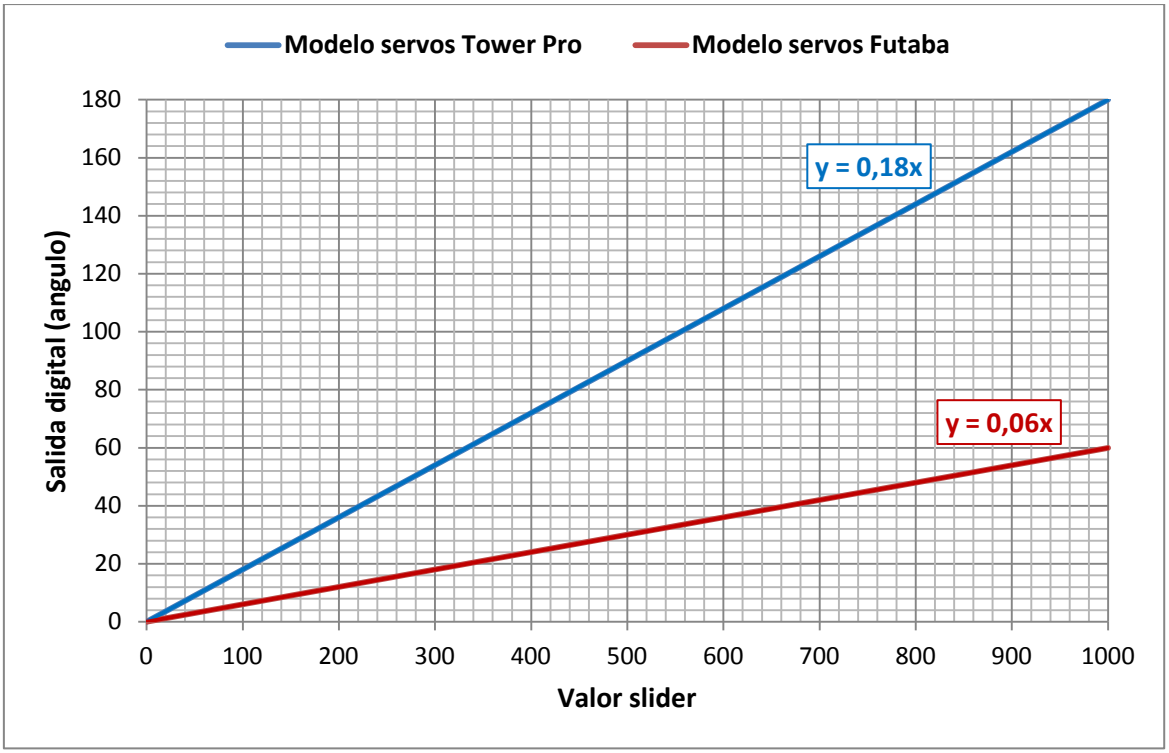
```
if (simGetScriptExecutionCount()==0) then

    sliders={4,6,8,10,12,14,16,18}
    posID={}
    pos={}
    pos_max={}
    pos_min={}
    int_hand=simGetUIHandle("UI")

    for i=1,8,1 do
      posID[i]=simGetObjectHandle('articulacion' .. i)
    end

    for i=1,8,1 do
      cyclic,interval=simGetJointInterval(posID[i])
      pos_min[i]=interval[1]
      pos_max[i]=interval[2]+interval[1]
    end
end

simHandleChildScript(sim_handle_all_except_explicit)

   for i=1,8,1 do
      pos[i]=simGetUISlider(int_hand,sliders[i])/1000*(pos_max[i]-pos_min[i])+pos_min[i]
   end

   for i=1,8,1 do
     simSetJointPosition(posID[i],pos[i])
   end
```

8.2.2. *Código del script asociado a la interfaz de los motores*

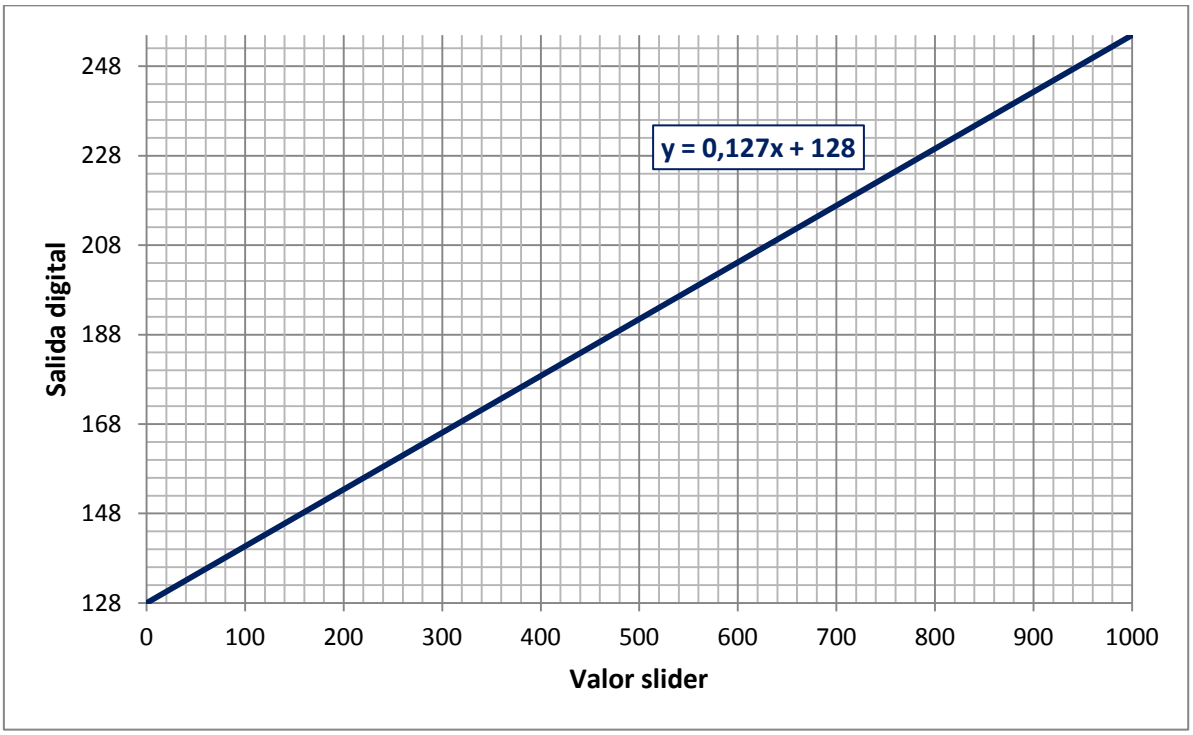
```
if (simGetScriptExecutionCount()==0) then

    Cuboid=simGetObjectAssociatedWithScript(sim_handle_self)
    int_hand=simGetUIHandle("motor")
    motor1=simGetObjectHandle("motor1")
    motor2=simGetObjectHandle("motor2")
    minmaxSpeed={0,6300*math.pi/180}

end

simHandleChildScript(sim_handle_all_except_explicit)

speed1=minmaxSpeed[1]+(minmaxSpeed[2]-minmaxSpeed[1])*simGetUISlider(int_hand,24)/1000
speed2=minmaxSpeed[1]+(minmaxSpeed[2]-minmaxSpeed[1])*simGetUISlider(int_hand,26)/1000

simSetJointTargetVelocity(motor1,speed1)
simSetJointTargetVelocity(motor2,speed2)
```

8.3. Modelo lineal de correspondencia entre sliders y salidas digitales

8.3.1. *Modelo para las salidas de los servomotores*



8.3.2. *Modelo para las salidas de los motores*



8.4. Código completo del plugin

```
#include "plugin_tfg.h"
#include "v_repLib.h"
#include <iostream>
#include <sstream>
#include <windows.h>
#include <windows.lib>


#ifdef _WIN32
    #include <shlwapi.h>
    #pragma comment(lib, "Shlwapi.lib")
#endif

#define PLUGIN_VERSION 1

LIBRARY vrepLib;

VREP_DLLEXPORT unsigned char v_repStart(void* reservedPointer,int reservedInt)
{
    char curDirAndFile[1024];

#ifdef _WIN32
    GetModuleFileName(NULL,curDirAndFile,1023);
    PathRemoveFileSpec(curDirAndFile);
#endif

    std::string currentDirAndPath(curDirAndFile);
    std::string temp(currentDirAndPath);

#ifdef _WIN32
    temp+="\\v_rep.dll";
#endif

    vrepLib=loadVrepLibrary(temp.c_str());
    if (vrepLib==NULL)
    {
        std::cout << "Error, fallo al cargar la librería de V-REP.\n";
        return(0);
    }
    if (getVrepProcAddresses(vrepLib)==0)
    {
        std::cout << "Error, faltan funciones requeridas en la librería de V-REP.\n";
        unloadVrepLibrary(vrepLib);
        return(0);

    int vrepVer;
    simGetIntegerParameter(sim_intparam_program_version,&vrepVer);
    if (vrepVer<20604)
    {
        std::cout << "Error, tu versión de V-REP está anticuada.\n";
        unloadVrepLibrary(vrepLib);
        return(0);
    }
```

```
    std::ostringstream SS;
    const char* UI = "UI";
    const char* motor = "motor";
    int i, UI_id, motor_id, WF[10];
    LPCVOID pSS;
    HANDLE ScomPort;

    UI_id = simGetUIHandle(UI);
    motor_id = simGetUIHandle(motor);
    pSS = &SS;

    ScomPort = CreateFile(_T("COM1:"), GENERIC_READ | GENERIC_WRITE, 0, 0,
                          OPEN_EXISTING, FILE_FLAG_OVERLAPPED, 0);


    return(PLUGIN_VERSION);
}

VREP_DLLEXPORT void v_repEnd()
{
    CloseHandle(ScomPort);

    unloadVrepLibrary(vrepLib);
}


VREP_DLLEXPORT void* v_repMessage(int message,int* auxiliaryData,void* customData,int* replyData)
{
    static bool refreshDlgFlag=true;
    int errorModeSaved;
    simGetIntegerParameter(sim_intparam_error_report_mode,&errorModeSaved);
    simSetIntegerParameter(sim_intparam_error_report_mode,sim_api_errormessage_ignore);
    void* retVal=NULL;

    if (message==sim_message_eventcallback_refreshdialogs)
        refreshDlgFlag=true;

    if (message==sim_message_eventcallback_instancepass)
    {

        int flags=auxiliaryData[0];
        bool sceneContentChanged=((flags&(1+2+4+8+16+32+64+256))!=0);
        bool instanceSwitched=((flags&64)!=0);

        if (sceneContentChanged)
        {
            refreshDlgFlag=true;
        }
    }
```

```
    if (message==sim_message_eventcallback_modulehandle)
    {
        if ( (customData==NULL)||(_stricmp("PluginTFG",(char*)customData)==0) )
        {
            for (i = 0; i < 8; i++){
                WF[i] = simGetUISlider(UI_id, (2 * i + 4));
            }
            for (i = 8; i < 10; i++){
                WF[i] = simGetUISlider(motor_id, (2 * i + 8));
            }
            for (i = 0; i < 10; i++){
                SS << "#" << i << "P" << WF[i];
            }
            SS << "T" << 100 << "\r\n";

            WriteFile(ScomPort, pSS, (sizeof(SS)), NULL, NULL);
        }
    }

    if ((message==sim_message_eventcallback_guipass)&&refreshDlgFlag)
    {
        refreshDlgFlag=false;
    }

    simSetIntegerParameter(sim_intparam_error_report_mode,errorModeSaved);
    return(retVal);
}
```

8.5. Código completo de Arduino

```
#include <String.h> //Librería para usar Strings
#include <Servo.h>  //Librería para usar servos
#include <Wire.h> //Librería para trabajar I2C
#define DIR_DISP 0xB0 //Dirección dispositivo hoja catálogo
#define motorD 0x01 //Dirección motor 1
#define motorI 0x02  //Dirección motor2


void setup() {
  Servo servo1, servo2, servo3, servo4; //Definimos los servos
  Servo servo5, servo6, servo7, servo8;

  servo1.attach(1); //Unimos cada servo a su salida
  servo2.attach(2);
  servo3.attach(3);
  servo4.attach(4);
  servo5.attach(5);
  servo6.attach(6);
  servo7.attach(7);
  servo8.attach(8);

  int pos[10], id, i=0, j=0, s=0, Twait; //Inicialización algunas variables
  float ang[8], mot[2], rang[8];

  Wire.begin(); //Se inicia la comunicacion I2C para motores
  Serial.begin(9600); //Se inicia la comunicación con el puerto serie
}
void loop(){ //Función principal bucle que se ejecutará
  while(Serial.available()>0){ //Se ejecutará siempre y cuando haya datos para leer

    for(j=0; j<10; j++){ //Leeremos 10 veces para obtener 10 valores
      if (i=0){
        String in_string = Serial.readStringUntil('#');
        i=1;
      } //Para la primera vez que leemos la cadena

      in_string = Serial.readStringUntil('P');
      id = in_string.toInt(); //Leer hasta P para obtener la id del servo/motor

      if(id==10){ //Si hemos llegado al final procedimiento distinto
        in_string = Serial.readStringUntil('T');
        pos[(id-1)] = in_string.toInt(); //Leer hasta T en lugar de # para obtener la posición
        in_string = Serial.readStringUntil('\r');
        Twait = in_string.toInt(); //Valor de Twait sólo incluido al final
        in_string = Serial.readStringUntil('\n');
        i=0; //Para dejar la cadena como si fuera primera iteración
      }
```

```
      else{
        in_string = Serial.readStringUntil('#');
        pos[(id-1)] = in_string.toInt(); //Obtenemos el valor de la posición
      }
    }

    j=0; //Para al volver al bucle for que empiece desde el principio
    i=0;

    //Tenemos las posiciones de servos y motores según sliders. Tenemos que conseguir su
    //valor en ángulos para servo e I2C(tiempo) para motores para poder usar las funciones
    for(s=0; s<8; s++){
      rang[s]=180.0;
    }s=0

    rang[2] = 60.0;
    rang[3] = rang[2]; //Todos los servos tienen un rango límite de 180 menos el 3 y 4

    for(s=0; s<8; s++){
      ang[s] = (rang[s]/1000.0) * pos[s];
    }s=0;

    for(s=0; s<2; s++){
      mot[s] = ((255-128)/1000.0) * pos[(s+8)];
    }s=0;

    //Valor para enviar a los servos
    servo1.write(ang[0]);
    servo2.write(ang[1]);
    servo3.write(ang[2]);
    servo4.write(ang[3]);
    servo5.write(ang[4]);
    servo6.write(ang[5]);
    servo7.write(ang[6]);
    servo8.write(ang[7]);

    //Valores para controlar los motores
    Wire.beginTransmission(DIR_DISP);
    Wire.send(motorD);
    Wire.send(mot[0]);
    Wire.send(motorI);
    Wire.send(mot[1]);
    Wire.endTransmission();

    delay(Twait); //Tiempo de espera entre lecturas del bucle
  }
}
```

8.6. Presupuesto

Código	Descripción	Unidad de medida	Precio unitario	Cantidad	Importe
1	**BRAZO ROBOT Y BASE**				
1.1	***IMPRESIÓN PIEZAS***				
	Impresión piezas en impresora 3D	h	2	20	40
	Filamento ABS	kg	10	0,45	4,5
					44,5
	Costes complementarios	%	44,5	2	0,89
1.1	***Coste total***				***45,39***
1.2	***DISEÑO Y FABRICACIÓN PCB***				
	Placa cobre 100x160x1.6 mm	ud	3,73	1	3,73
	Poste macho 40C P2,54 Recto	ud	0,39	2	0,78
	Poste hembra 40C P2,54	ud	1	1	1
	Regulador tensión780CV TO220 5V 1,5A	ud	0,2	1	0,2
	Resistencia 1/4W 10kOhm	ud	0,01	2	0,02
	Condensador multicapa 10 pF	ud	0,2	1	0,2
	Regleta PCB tornillo 2C P5MM	ud	1,71	5	8,55
	Cable de estaño para soldar 0,5 mm	g	0,08	5	0,4
	Uso de fresadora digital	h	1,5	3	4,5
	Ingeniero industrial	h	40	70	2800
					2819,38
	Costes complementarios	%	2819,38	2	56,39
1.2	***Coste total***				***2875,77***

1.3	***DISEÑO Y MONTAJE ROBOT***				
	Tornillo M3x16	ud	0,02	75	1,5
	Tornillo M2x10	ud	0,05	15	0,75
	Tuerca M3	ud	0,02	40	0,8
	Tuerca M2	ud	0,05	5	0,25
	Arandela M3	ud	0,03	40	1,2
	Arandela M2	ud	0,08	5	0,4
	Motor eléctrico DC 12-24V	ud	16,17	2	32,34
	Placa potencia MD22	ud	66,05	1	66,05
	Rueda modelismo	ud	8,68	2	17,36
	Servo Futaba S3003	ud	8,75	2	17,5
	Servo TowerPro MG90S	ud	7,1	2	14,2
	Servo TowerPro SG90	ud	4,38	2	8,76
	Servo TowerPro MG995	ud	8,75	2	17,5
	Adaptor USB serie JY-MCU	ud	5,48	1	5,48
	Arduino Mini 05	ud	16	1	16
	PCB diseñada	ud	0	1	0
	Ingeniero industrial	h	40	60	2400
					2600,09
	Costes complementarios	%	2	2600,09	52
1.3	***Coste total***				***2652,09***

Código	Descripción	Unidad de medida	Precio unitario	Cantidad	Importe
2	**DESARROLLO SOFTWARE**				
2.1	***SOFTWARE CONTROL Y PLUGIN***				
	V-REP	%	0	0	0
	Visual Studio Professional 2013	%	387	2,5	9,68
	Ingeniero industrial	h	45	150	6750
					6759,68
	Costes complementarios	%	6759,68	2	135,19
2.1	***Coste total***				***6894,87***
2.2	***PROGRAMACIÓN DE ARDUINO***				
	Arduino	%	0	0	0
	Ingeniero industrial	h	45	50	2250
					2250
	Costes complementarios	%	2250	2	45
2.2	***Coste total***				***2295***

1	**BRAZO ROBOT Y BASE**	
1.1	IMPRESIÓN PIEZAS	45,39
1.2	DISEÑO Y FABRICACIÓN PCB	2875,77
1.3	DISEÑO Y MONTAJE ROBOT	2652,09
1	***Coste Total***	***5573,25***

2	**DESARROLLO SOFTWARE**	
2.1	SOFTWARE CONTROL Y PLUGIN	6894,87
2.2	PROGRAMACIÓN DE ARDUINO	2295
2	***Coste Total***	***9189,87***

1	BRAZO ROBOT Y BASE	5573,25
2	DESARROLLO SOFTWARE	9189,87
	Presupuesto ejecución material	**14763,12**
	13% Gastos generales	1919,21
	6% Beneficio industrial	885,79
	Presupuesto contrata	**17568,12**
	21% I.V.A.	3689,31
	Presupuesto total	**21257,43**