industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Adaptación de un robot SCORBOT-ER III para su control usando Arduino

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Rubén Jiménez Andreu**
Director: Jorger Feliu Batlle

Cartagena, 12 de septiembre de 2015

Universidad Politécnica de Cartagena

Índice de contenido

Índice de figuras

Índice de tablas

Índice de códigos

1.Introducción

1.1. Antecedentes históricos[1]

Desde la antigua Grecia se han construido máquinas con el propósito de imitar el movimiento y las funciones de los seres vivos, tanto de los humanos como de animales. A estas máquinas se las denominaba *automatos,* de donde deriva la actual palabra **autómata**: máquina que imita la figura y movimientos de un ser animado. Durante siglos dichos autómatas se han utilizado para fines puramente lúdicos siendo su uso en aplicaciones prácticas contado. Destacan *el Gallo de Estraburgo* (1352), el más antiguo que se conserva; el *León mecánico*, ideado por Leonardo Da Vinci para el rey Luis XII de Francia; y en España el conocido con *Hombre de palo,* construido por Juanelo Turriano en el siglo XVI para emperador Carlos V y que imitaba a un monje.

A finales del siglo XVIII y principios del XIX se desarrollan alguna máquinas, principal destinados a la industria textil, tales como la hiladora mecánica de Hargreaves, la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785) y el telar de Jacquard (1801). Este último utilizaba cintas de papel perforado como programa para sus patrones. Desde este momento se empiezan a usar los autómatas en la producción industrial.

El origen de la palabra **robot** proviene de la obra de teatro rusa *Rossum's Universal Robot*, de Karel Capek(1890-1938) y estrenada en 1921. Deriva de la palabra eslava *robota*, que se refiere al trabajo realizado de manera forzada. En ella, los androides fabricados a partir de la fórmula del científico Rossum se rebelan contra los humanos hasta su destrucción, dejando vivo solamente uno de sus creadores con la esperanza de que los ayude a reproducirse. Esta palabra ha sido usada desde entonces por los escritores de ciencia ficción, pero fue sin duda es escritor Isaac Asimov (1920-1992) quien la hizo famosa y el que creo el concepto de *robótica* y las consabidas 3 leyes de la robótica:

1. Un robot no puede perjudicar a un ser humano, ni por inacción, permitir que un ser humano sufra daño.
2. Un robot ha de obedecer siempre las órdenes de un ser humano, excepto si tales ordenes entran en conflicto con la primera ley.
3. Un robot debe proteger su propia existencia siempre que esto no entre en conflicto con primera o la segunda ley.

1.2. Origen y desarrollo de la robótica

Tras los primeros autómatas, con el principal objetivo de manipular material radioactivo con seguridad, en 19848 R.C. Goertz empezaro a desarrollar los **telemanipuladores**. Consistían en dispositivos mecánicos maestro-esclavo donde el manipulador era controlado directamente por un operario y sus movimientos eran reproducidos a distancia por el esclavo. Posteriormente se introdujo la tecnología electrónica y del servocontrol y estos aparatos encontraron un gran desarrollo en la industria nuclear, militar, espacial, etc. Sin embargo, el estar recluidos a un mercado muy restringido y la necesidad de un control continuo por parte de un operador explica que no hayan tenido un gran desarrollo.

La sustitución del operador humano por un programa informático dio origen al concepto de robot. La primera patente de un dispositivo robótico fue solicitada por el inglés C.W. Kenward en 1954, pero fue el norteamericano George C. Devol quien estableció las bases del robot industrial moderno, descrito como un *dispositivo de transferencia de artículos programa* en 1954. En 1956, junto con Joseph F. Engelberg, director de ingeniería de la división aeroespacial de la empresa Manning Maxwell y Moore en Stanford, Conneticut, empiezan a trabajar en la utilización industrial de sus maquinas. Fundaron la Consolidated Controls Corporation, que más tarde se convierte en Unimation, e instalaron su primera máquina Unimate (1960) en la fabrica de General Motors de Trenton, Nueva Jersey.

Los primeros robots tenían configuración esférica y antropomórfica, de uso especialmente válido para la manipulación. En 1982, el profesor Makino de la Univesidad Yamahasi en Japón desarrolla en concepto el robot SCARA, como un robot con un número de grados de liberta reducido (3 o 4), un coste limitado y una configuración orientada al ensamblado de piezas.

1.3. Clasificación del robot industrial

La definición de robot adoptada por la *Organización Internacional de Estándares* (ISO) define al robot industrial de la siguiente manera:

> 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.

La *Federación Internacional de Robótica* (IFR) clasifica los robots en 4 tipos:

- Robot secuencial.
- Robot de trayectoria controlable.
- Robot adaptativo.
- Robot telemanipulado.

1.4. Morfología del robot*[2]*

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

1.4.1.Estructura mecánica del robot

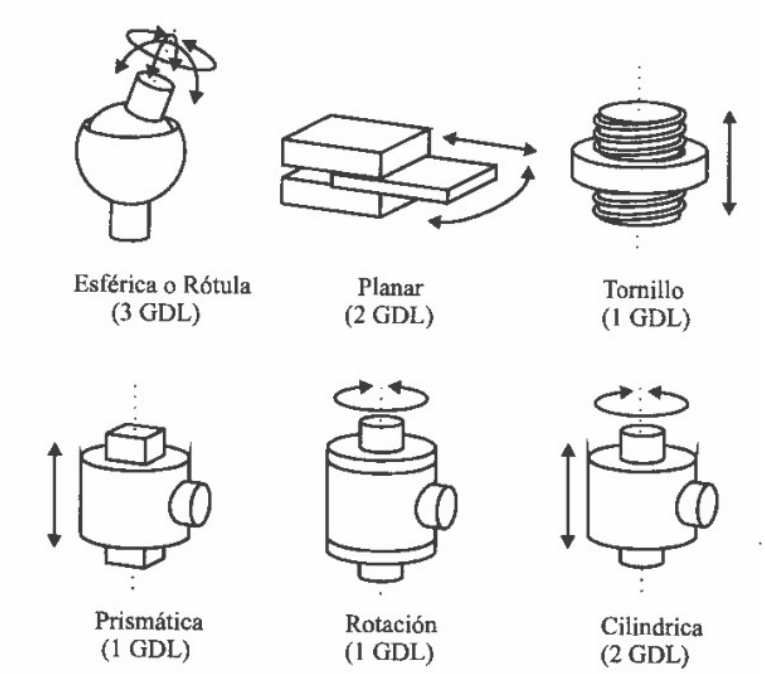

Figura 1.2: Distintos tipos de articulaciones para robots.

Mecánicamente, un robot está formado por una serie de elementos o eslabones unidos mediante articulaciones que permiten un movimiento relativo entre cada dos eslabones consecutivos. La constitución física de la mayor parte de los robots industriales guarda cierta similitud con la anatomía que componen el robot, se usan términos como cuerpo, brazo, codo y muñeca. El movimiento de cada articulación puede ser de desplazamiento, giro o una combinación de ambos. Pueden verse las principales en la Figura 1.2. En la práctica, en los robots solo se emplean la articulación de rotación y la prismática.

La suma de los movimientos que puede realizar una articulación se llama **grado de libertad** (GDL). Los grados de libertad del robot del robot son la suma de los GDL de todas sus articulaciones. Como las articulaciones son unicamente las de rotación y prismática, cada una con un solo GDL, los grados de libertad del robot coincide con el número de articulaciones.

La combinación de articulaciones da lugar a diferentes configuraciones, con características a tener en cuenta tanto en el diseño y construcción del robot como a su aplicación. Algunas de las más frecuentes

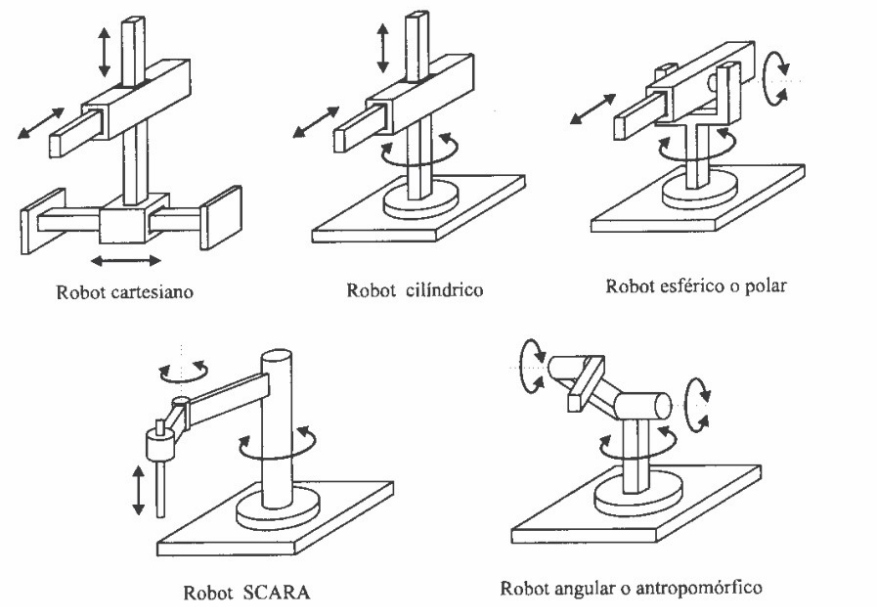

Figura 1.1: Configuraciones más comunes en los robots industriales.

pueden verse en la Error: no se encontró el origen de la referencia, donde se atiende unicamente a las tres primeras articulaciones, las más importantes a la hora de posicionar el robot.

Los grados de libertad necesarios para posicionar la herramienta en cualquier posición y en cualquier orientación en el espacio son al menos seis GDL, aunque algunos cuentan solamente con cuatro o cinco por ser estos suficientes para las tarea encomendadas; o bien más de seis para evitar obstáculos.

1.4.2. Transmisiones y reductores

Las transmisiones son los elementos encargados de transmitir el movimiento desde los actuadores hasta las articulaciones. Los reductores son los encargados de adaptar el par y la velocidad de la salida de los actuadores al movimiento de los elementos del robot.

Dado que un robot mueve su extremo con aceleraciones elevadas, para reducir su momento de inercia se procura colocar los actuadores, por lo general pesados, lo más cerca posible de la base. Esta circunstancia obliga a usar sistemas de transmisión que trasladen el movimiento hasta las articulaciones. Así mismo, también pueden usar para transformar un movimiento circular en uno lineal (tornillo sin fin, cremallera) y viceversa (paralelogramo articulador, cremallera), o simplemente para transmitir movimiento circular (engranajes, correas dentadas, cadenas). Un buen sistema de reducción debe tener un peso y un tamaño reducido, se ha de evitar que presente juegos y holguras y se deben buscar trasmisiones con gran rendimiento.

En cuanto a los reductores, sí que existen determinados sistemas usados de manera preferente en los robots industriales debido a que se les exigen condiciones muy restrictivas, motivadas por la necesidad de gran precisión y velocidad de posicionamiento. El par de salida de un reductor es el siguiente:

$$T_2 = \eta T_1 \frac{\omega_1}{\omega_2} \tag{1.1}$$

donde el rendimiento (η) puede llegar a ser cerca del 100% y la relación de velocidades (ω_1 = velocidad de entrada; ω_2 = velocidad de salida) varía entre 50 y 300.

1.4.3.Actuadores

Los actuadores tiene por misión generar el movimiento de los elementos del robot según las órdenes dadas por la unidad de control. Los actuadores usados en robótica pueden emplear energía neumática, hidráulica o eléctrica.

En los actuadores neumáticos la fuente de energía es aire a presión entre 5 y 10. Existen de dos tipos: cilindros neumáticos y motores neumáticos. En general no consiguen mucha precisión de posicionamiento, pero su robustez y sencillez los hacen adecuados en aquellos casos en los que sea suficiente un posicionamiento en dos situaciones diferentes.

Los actuadores hidráulicos se asemejan a los neumáticos, usando en este caso aceites minerales a una presión entre 50 y 100 bar. Se consigue mayor precisión, estabilidad frente a cargas estáticas y desarrollan elevadas cargas y pares. Por contra, por las elevadas presiones pueden producirse fugas de aceite y son instalaciones complicadas.

En cuanto a los actuadores eléctricos su control, sencillez y precisión han hecho que sean los más usados en los robots industriales actuales. Los motores de corriente continua (DC) son los más usados por

su facilidad de control. Si el motor trabaja a tensión constante se puede variar su velocidad disminuyendo el flujo d excitación. Por otro lado, en el caso de control por inducido, la intensidad del inductor se mantiene constante mientras que la tensión inducida controla la velocidad de giro. Presentan el inconveniente del obligado mantenimiento de las escobillas del motor.

En cuanto a los motores de corriente alterna (AC), no han tenido aplicación en campo de la robótica debido fundamentalmente a su dificultad de control. Sin embargo, se han introducido mejoras técnicas que los han hecho con los motores de continua, entre ellas, la construcción de rotores síncronos sin escobillas, uso de convertidores estáticos para regular la frecuencia (y así la velocidad) con facilidad y precisión, y el empleo de la microelectrónica, que permite una gran capacidad de control.

1.4.4. Sensores internos

Los sensores internos proporcionan información sobre el estado del robot, fundamentalmente de la posicione de la posición de sus articulaciones para conseguir una adecuada precisión, velocidad e inteligencia.

Para el control de la posición angular se emplean fundamentalmente los denominados **encoders** y **resolvers**.

Los codificadores ópticos constan, en su forma más simple, de un disco con partes que dejan pasar la luz y otras que no colocadas radialmente y equidistantes entre sí, de un sistema emisor de luz y de un elemento fotoreceptor. A medida que el eje gira se generan pulsos a medida que el eje gira y la luz atraviese cada marca. Llevando la cuenta de esos impulsos es posible conocer la posición del eje. Si además queremos conocer en qué sentido giro, es preciso añadir otro dispositivo emisor y receptor para obtener una señal adicional desplazada 90º que indique si el contador asciende o desciende. Su precisión depende del número de marcas del disco. Una forma de aumentarla es contar tantos los pulsos de bajada como de subida, multiplicando por cuatro la precisión.

Otra alternativa a los encoders son los resolveres. Se trata de captadores analógicos de resolución teóricamente infinita. El funcionamiento de los resolveres se basa en la utilización de una bobina solidaria al eje excitada por una portadora, generalmente con 400Hz, y por dos bobinas fijas situadas a su alrededor. El giro de la bobina móvil hace que el acoplamiento con las bobinas fijas varíe, consiguiendo que la señal resultante en estas dependa del seno del angulo de giro. El funcionamiento de los sincroresolveres es análogo salvo que las bobinas fijas forman un sistema trifásico en estrella.

1.4.5. Elementos terminales

Los elementos terminales son los encargados de interaccionar directamente con el entorno del robot. Pueden ser tanto de aprehensión como herramientas.

Se puede establecer una clasificación de los elementos terminales atendiendo a si se trata de un elemento de sujeción o de una herramienta. Los primeros se pueden clasificar según el sistema de sujeción empleado.

Los elementos de sujeción se utilizan para agarrar y sostener los objetos y se suelen denominar pinzas. Se puede distinguir entre las que utilizan dispositivos de agarre mecánico y las que utilizan otro tipo de dispositivo, como ventosas, pinzas magnéticas, etc.

La elección de un pinza tiene en cuenta varios factores, entre ellos el tipo de objeto y de manipulación que se desea realizar, así como el peso, la capacidad de control y el tipo de accionamiento de la pinza.

En muchas aplicaciones el robot ha de realizar operaciones que no consisten en manipular objetos, sino que implica el uso de una herramienta. El tipo de herramienta con que puede dotarse un robot es muy amplio, entre las que se encuentran pinzas de soldadura, sopletes, pistolas de pintura, fresa-lija, etc. La herramienta suele estar fija al extremo del robot, aunque en ocasiones se dota a este de un cambio automático que permite el uso de varias herramientas.

2.SCORBOT-ER III

2.1. Introducción

En el Departamento de Automatización y Sistemas de la Universidad Politécnica de Cartagena se encontraba este robot sin uso durante años.

Se trata de un robot de tipo antropomórfico destinado a la educación con cinco grados de libertad. Como actuadores emplea cinco motores de corriente continua de $12V_{DC}$ cada uno y para su posicionamiento emplea encoders ópticos así como *microswitches* o finales de carrera. Una pinza también controlada con un motor DC hace de elemento terminal del mismo.

En un principio, junto al robot venia un controlador con el se manejaba conectando a un ordenador y programando con su *software* correspondiente. Sin embargo, este controlador no se encuentra disponible y el robot se encuentra en desuso.

Es, por tanto, objeto de este **proyecto fin de grado** realizar una puesta en funcionamiento de este robot. En primer lugar se comprobará el buen funcionamiento de todas sus partes, después se realizará su electrónica de control en un circuito impreso y por último se procederá a su programación usando las plataformas *Arduino Mega* y *Matlab.*

A continuación se presentarán las características técnicas más importantes del *SCORBOT-ER III* [3].

2.2. El brazo robot

El **SCORBOT-ER III** es un robot articulado vertical. Consta de una articulación en la base que hace que el robot rote en el plano horizontal, tres articulaciones que hacen que los eslabones del robot giren el plano vertical y una articulación en la muñeca que permite rotar la pinza.

El cuerpo del robot es la parte principal, el cual contiene cinco de los seis motores. Los eslabones (brazo y antebrazo) y las articulaciones permiten el movimiento deseado de la pinza.

La pinza es el elemento terminal del brazo robot. Para que la pinza alcance la posición deseada, algunos o todos los ejes deben moverse. Los dedos de la pinza se abren y cierran en paralelo. Varios tipos de almohadillas y elementos finales, como un cepillo de aire o una pinza aspiradora, pueden unirse a la

pinza por medio de los agujeros de Ø 4 mm (0,16") en los dedos de la pinza. La tabla siguiente ofrece las especificaciones técnicas del brazo robot **SCORBOT-ER III**

Tabla 2.1: Especificaciones brazo robot SCORBOT-ER III

Elemento	Especificaciones
Estructura mecánica	Articulado verticalmente 5 ejes más pinza Control de 8 ejes simultáneamente
Espacio de trabajo:	
Eje 1: Rotación de la base	310º
Eje 2: Rotación del hombro	+ 130º / -35º
Eje 3: Rotación del codo	± 130º
Eje 4: Rotación de la muñeca	±130º
Eje 5: Giro de la muñeca	Ilimitado
Radio máximo de trabajo	610 mm (24,4")
Apertura de la pinza	75 mm (3") sin almohadillas 65 mm (2,56") con almohadillas
Transmisiones	Engranajes, correas dentadas y tornillo sin fin
Actuadores	6 motores DC con control de lazo cerrado
Realimentación	Encoders ópticos en todos los ejes
Hard Home	Posición de referencia fija en todos los ejes
Repetibilidad	± 0,5 mm (±0,02")
Velocidad máxima	330 mm/s (13"/s)
Peso	
Brazo robot	11 kg (24 lb.)
Controlador	5 kg (11 lb.)

2.3. Motores

Los cinco ejes y la pinza están operados mediante servo motores de corriente continua. La dirección de revolución del motor es determinada por la polaridad del voltaje operativo: un voltaje positivo hace que el motor se mueva en una dirección mientras que uno negativo hace que se mueva en la contraria. Cada motor tiene un lazo de control cerrado, esto es, un encoder suministra realimentación sobre el movimiento del motor.

2.4. Encoders

Un encoder óptico montado en cada motor moniterea continuamente la posición, dirección y velocidad en cada movimiento. El encoder produce un impulso eléctrico de acuerdo a la rotación del eje del motor en el que está montado. El número y la frecuencia de impulsos son medidos por el controlador de manera que compara la posición actual con la posición deseada, y hace los ajuste necesarios.

El SCORBOT-ER III usa dos tipos de encoders. Los encoder en los motores 1 al 5 difieren del 6, los cuales pueden verse en Figura 2.1. El disco de encoder con 6 huecos (izquierda) es usado en los motores del brazo robot, mientras que el disco con 3 huecos (derecha) se usa en la pinza. En la Figura 2.2 puede verse el circuito de los encoders, que constan de los leds infrarrojos y dos fototransistores. En la Figura 2.3 se muestra como encaja el circuito dentro de su carcasa.

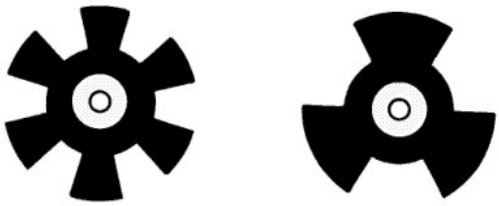

Figura 2.1: Discos de los encoders

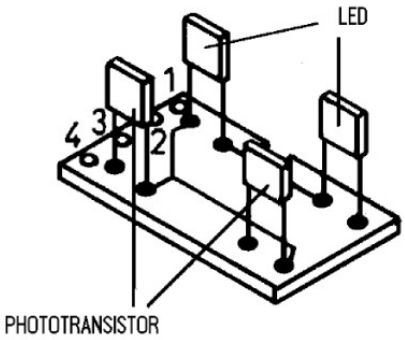

Figura 2.2: Circuito del encoder

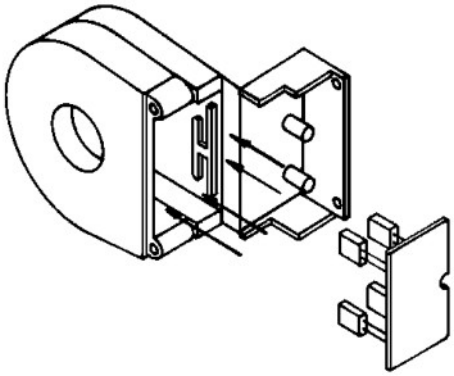

Figura 2.3: Ensamblaje del encoder

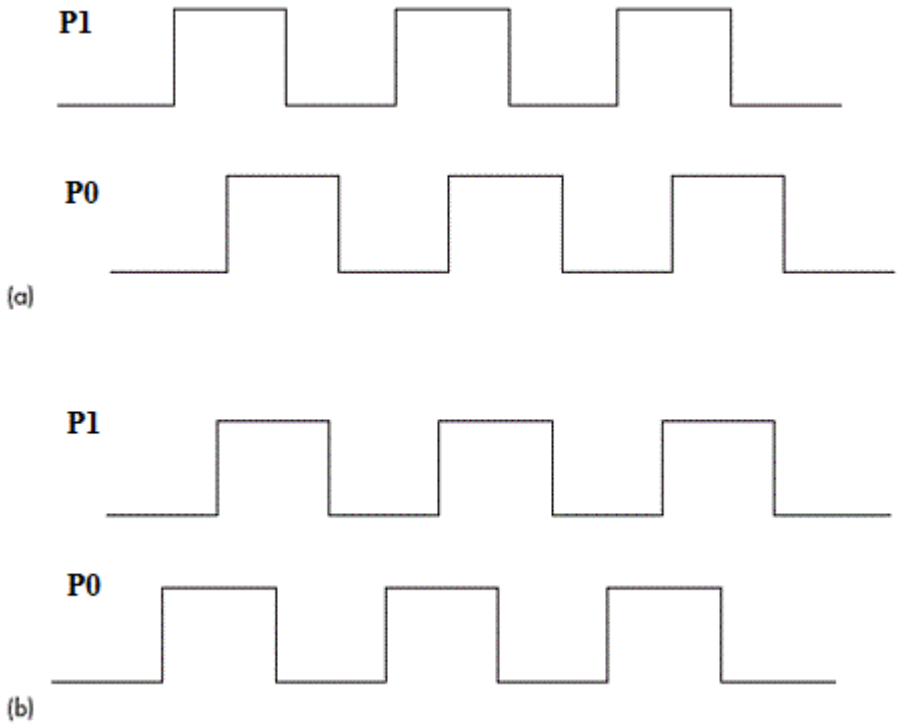

Figura 2.4: Señales de los encoders.

Tenemos entonces por cada encoder dos señales, que llamaremos P_1 y P_0. Una de ellas, P_1, se usará para contabilizar los pulsos del encoder y conocer cuánto ha girado. Con la otra, P_0, conoceremos el sentido de giro y si hay que sumar los pulsos o girarlos.

En la Figura 2.4 pueden verse unas gráficas con las señales de los encoders. En la gráfica (a) el encoder lleva sentido negativo y los pulsos se restan, en la gráfica (b) se tiene sentido positivo y los pulsos se suman. Cada pulso se contabiliza cuando la señal P_1 cambia (pasa de 0 a 1 o de 1 a 0) y según si P_0 se encuentra en 0 o 1 en ese momento se restará o sumará. En la Tabla 2.2 se puede ver la relación entre los estados de los encoders en el momento del cambio de P_1 y el sentido del encoder.

Tabla 2.2: Señales y sentido de los encoders.

P1	P0	Sentido
0	0	Positivo
0	1	Negativo
1	0	Negativo
1	1	Positivo

2.5. Microswitches

Cinco *microswitches* están instalados en el brazo mecánico. Cuando el robot alcanza la posición en la cual el microswitch está pulsado en cada articulación, esta posición predeterminada es conocida como *hard home*. Este es el punto de referencia para las operación del robot. Cualquiera que sea la posición en la que el sistema sea conectado, el robot debe ser resteado a esta posición de inicio o *hard home*.

2.6. Transmisiones

Distintos tipos de transmisiones son usadas para operar los eslabones del brazo mecánico. Se emplean engranajes para el movimiento de la base y el hombro, engranajes y correas dentadas para codo y correas dentadas y un sistema diferencial con engranajes cónicos al final del brazo permiten el movimiento de la muñeca. Un tornillo sin fin acoplado directamente a un servo motor DC hace que la pinza se abra y se cierre.

2.7. Cableado del robot

El cable principal del robot contiene 50 cables divididos en seis grupos, uno para cada motor. Cada grupo contiene ocho cables:

- 2 cables suministran voltaje al motor.
- 2 cables reciben pulsos del encoder óptico (canal 0 y canal 1).
- 1 cable lleva la señal al microswitch.
- 1 cable suministra voltaje al encoder (V_{LED}).
- 1 cable hace de masa para el encoder.
- 1 cable hace de masa para el microswitch.

Todos los comandos, tanto operacionales como de control, son transmitidos a través de este cable, el cual es la única conexión entre el brazo robot y el controlador. El cable parte de la base del robot y tiene en su extremo un conector D50. La conexiones del conector D50 pueden verse en el Apéndice B.

2.8. LMD18200

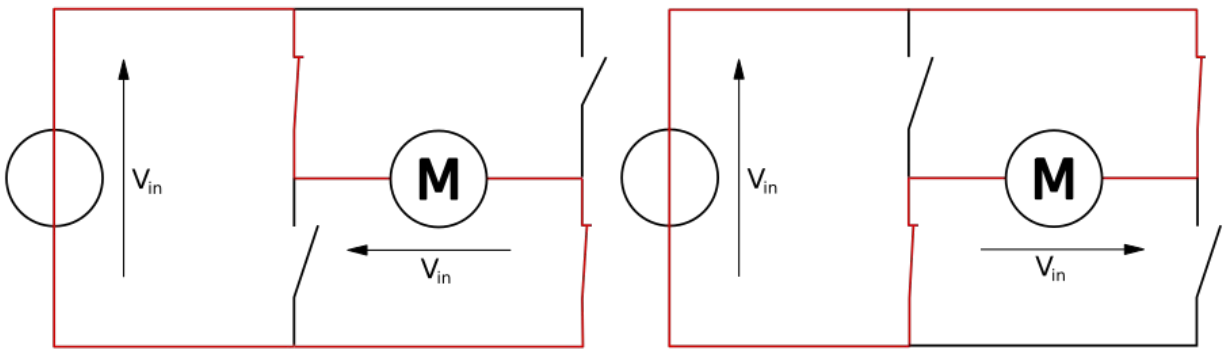

Figura 2.5: «Puente en H operando» de Cyril BUTTAY

Para el control de los motores, esto es, hacer que giren en un sentido o en otro, se utilizará un puente en H. La forma mas sencilla de hacer este tipo de circuitos es con cuatro interruptores los cuales se conectan o desconectan en parejas para cambiar la polaridad del motor o detenerlos, como se muestra en la Figura 2.5. Esto puede hacerse más complejo. En lugar de interruptores pueden emplearse transistores de manera que los motores puedan controlarse con la señal de disparo de los mismos. Usando además un tren de impulsos, o PWM, puede regularse la frecuencia con la que los transistores se conectan y desconectan. Por ejemplo, con un PWM al 60% los transistores están conduciendo un 60% del tiempo y desconectados y 40%. Los motores detectan esta frecuencia como una variación del voltaje que se les suministra, tomando la media en el tiempo. Siguiendo con el ejemplo anterior, con una señal PWM de 60% y un voltaje de 12VDC los motores detectarían una tensión de 9,8VDC. De esta manera la velocidad de giro de los motores es fácilmente controlable.

Existen circuitos integrados que contienen todos los elementos necesarios de un puente en H, con algunas características añadidas. Tal es el caso del circuito integrado **LMD18200**, de *Texas Instruments*, el cual se utilizará para el control de los motores.

El LMD18200 es un puente en H de 3A diseñado para aplicaciones de control de movimiento. El elemento ha sido construido usando procesos multitecnológicos que combinan circuitos de control

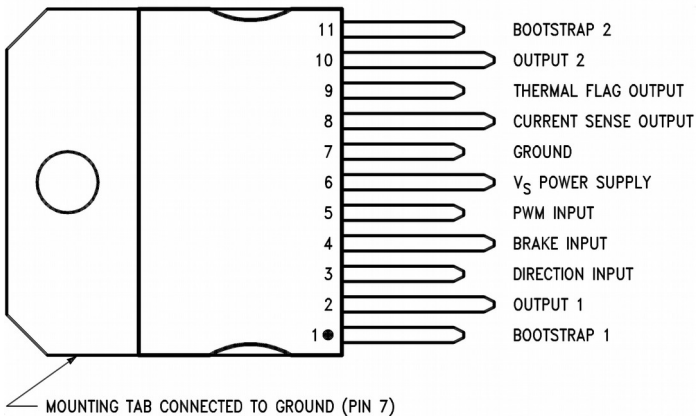

Figura 2.6: Encapsulado del LMD18200 (vista superior)

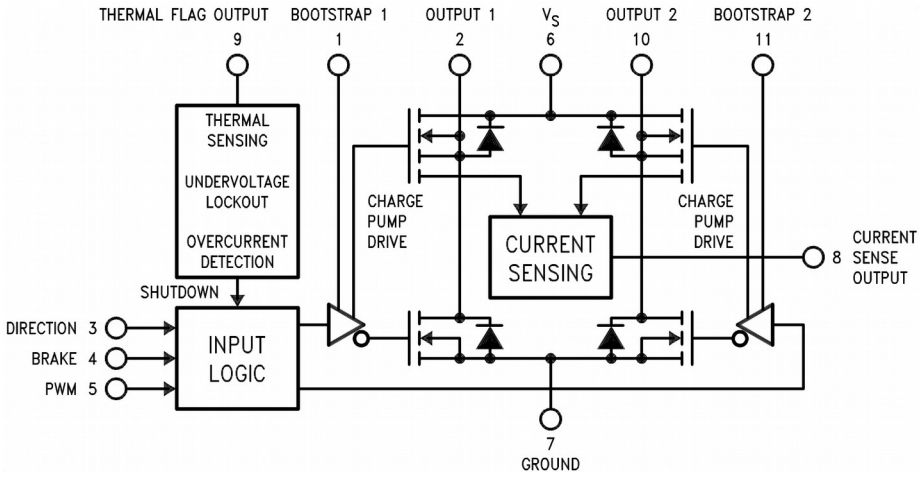

Figura 2.7: Diagrama de bloques funcional del LMD18200

bipolares y CMOS con compones de energía DMOS en la misma estructura monolítica. Ideal para motores de corriente continua y paso a paso, el LMD18200 soporta picos de corriente de hasta 6A. Ha sido añadido un innovador circuito que proporciona bajas pérdidas de sensibilidad en la corriente de salida [4]. Cada LMD18200 sirve permite el control de un motor, por lo que serán necesarios seis de ellos para todo el control del brazo robótico.

En las Figuras 2.6 y 2.7 pueden verse algunos detalles del LMD18200. Se adjunta su su hoja de especificaciones.

3. Electrónica

3.1. Comprobación del SCORBOT-ER III

El primer paso en la puesta en funcionamiento del robot fue comprobar que todos los motores funcionasen correctamente y que no le faltara ningún componente y estuvieran en buen estado. A continuación, se estudiaría la manera de hacer el circuito electrónico para que su funcionamiento fuese el adecuado.

Para los motores, se conectaron unos por uno a una fuente de tensión continua con una tensión de $12V_{DC}$, la recomendada para estos motores. Todos funcionaban correctamente en ambos sentidos y producían el movimiento del brazo mecánico para el que estaban diseñados, es decir, que los sistemas reductores también funcionaban correctamente. Así mismo, se comprobó la corriente que consumía cada motor, que no era superior a los 0,6A en ningún caso, la cual puede se soportado por el LMD18200, que aguanta hasta 3A.

Durante la comprobación de los encoders del robot se detectaron algunos fallos:

1. La rueda del encoder 5 estaba rota y la del encoder 4 no estaba. Para sustituirlas se hizo un modelo de las mismas con *SolidWorks* y con ayuda de los compañeros de *UPCTMakers* se imprimieron en una impresora 3D.
2. Un LED infrarrojo también del encoder número 4 no funcionaba. Se retiro el LED defectuoso y se sustituyo por otro semejante con las mismas características y encapsulado. Gracias a que existen orificios extra en el circuito impreso de los encoder la instalación fue sencilla.
3. Un fototransistor del encoder de la pinza no funciona. Sin embargo esto no resulta un problema en este caso y la pinza puede controlarse con una sola entrada de encoder.

3.2. Circuitos electrónicos

Como se ha explicado anteriormente, por cada motor tenemos 8 conexiones que nos permiten tanto el control del mismo como obtener realimentación sobre su posición. Todo esto será, a su vez, controlado por un Arduino Mega, por ser el único modelo de Arduino que nos proporciona el número suficiente de conexiones. A continuación se explicará los circuitos electrónicos utilizados para cada elemento. El circuito final consistirá en ponerlos juntos y copiarlos para cada uno de los seis motores. A continuación se imprimirá dicho circuito en una placa de circuito impreso (PCB) con el uso del software **Eagle**.

3.2.1. Motores

Cada motor tiene dos entradas en el conector D50, llamado así por tener 50 pines, que servirán para su control mediante el circuito integrado LMD18200 antes descrito.

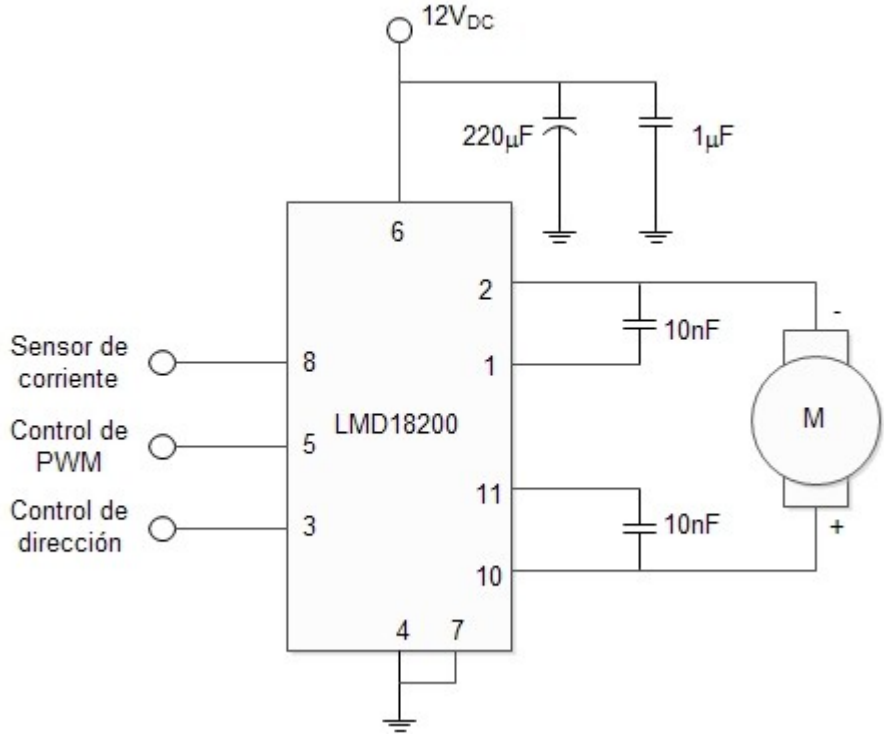

Figura 3.1: Circuito para el LMD18200

El LMD18200 tiene varias salidas y entradas de las cuales solo se utilizaran aquellas necesarias y se conectará tal como se indica en su hoja de especificaciones.

- Los pines 2 y 11, correspondientes a *output* 1 y 2 respectivamente, se conectarán cada uno a un borne del motor (- y +).
- Los pines 1 y 12 (*bootstrap* 1 y 2), tienen que conectarse a un condensador de 10nF y este al *output* correspondiente en los pines 2 u 11.
- Los pines 3 y 5 son respectivamente *direction input* y *PWM input*. Estos servirán para el control del motor y se conectan directamente al los pines del Arduino, teniendo en cuenta que hay que conectar el pin 5 a unas de las salidas PWM del Arduino.
- El pin 6 es V_S *power supply.* Por aquí se le suministra la energía al LMD18200 y se conecta a la fuente de tensión. Para evitar cambios bruscos de tensión se añaden dos condesares, uno de 22µF y otro de 1µF.
- El pin 7 es *ground* y se conecta directamente a masa.
- El pin 4 es *break* y, como su nombre indica, sirve para frenar el motor. Como no es necesario para esta aplicación se conecta a masa.
- El pin 8 es *current sense output* y a través de él podemos medir la tensión la tensión que pasa por el LMD18200. Puede ser útil para futuros trabajos o prácticas con el brazo mecánico, por lo que se añadirá una salida para este pin.

3.2.2. LEDs infrarrojos

Por cada motor tenemos un V_{LED} y un *GND*. Se conecta V_{LED} a una resistencia de 39Ω según se indica en las especificaciones del SCORBOT-ER III y esta a la salida de $5V_{DC}$ del Arduino. Conectamos *GND* a masa.

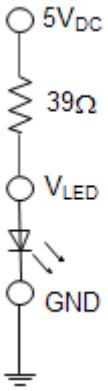

Figura 3.2: Conexión de los LEDs infrarrojos.

3.2.3. Microswitches

Los microswitches son interruptores que pueden estar en dos estados: conduciendo o no conduciendo. La forma adecuada de conectar un interruptor es usando un divisor de tensión, como el que puede verse en la Figura 3.3. Se puede considerar el interruptor como una resistencia que toma valor nulo cuando está conduciendo y valor infinito cuando no lo está. En nuestro caso, por las conexiones propias del robot, el microswitch será la resistencia R_2. La formula para calcular el valor de la tensión de salida es:

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \tag{3.1}$$

Donde R_1 es una resistencia tipo *pull-up* de 20kΩ que nosotros añadiremos, V_{in} es la alimentación de $5VD_C$ que tomaremos directamente del Arduino y V_{out} es la señal de entrada en el Arduino.

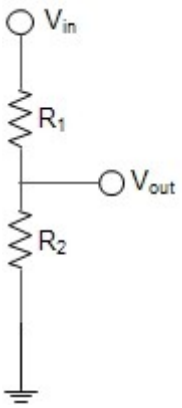

Figura 3.3: Divisor de tensión.

De esta manera, cuando el microswitch esté pulsado R_2 será cero, V_{out} será también cero y el Arduino detectará un *LOW* en la entrada. Por el contrario, cuando no esté pulsado R_2 será podemos considerarlo como un valor muy grande, V_{out} será igual que V_{in}, $5V_{DC}$, y el Arduino detectará un *HIGH* en la entrada.

El circuito final quedará de la siguiente manera:

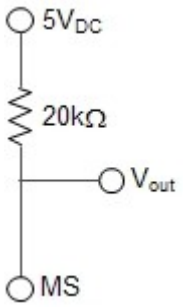

Figura 3.4: Divisor de tensión para los microswitches.

Donde MS es la entrada del conector D50 del SCORBOT-ER III.

3.2.4. Fototransistores

Para realizar las mediciones del encoder y conocer la posición de nuestro brazo robot usamos, junto con los leds infrarrojos, dos fototransistores. Por cada motor tenemos dos entradas en el conector D50: P1 y P0.

Los fototransistores funcionan de forma parecida a los microswitches: cuando reciben la luz infrarroja tienen una resistencia muy alta actuando como interruptores abiertos mientras que cuando no la reciben presentan resistencia nula y funcionan como interruptores cerrados. Por tanto, también se usarán divisores de tensión para saber si estamos en un aspa o un hueco y medir los impulsos recibidos.

El esquema es el siguiente:

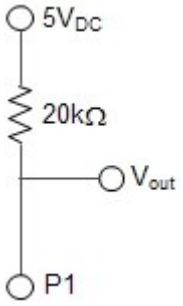

Figura 3.5: Divisor de tensión para los fototransistores.

3.3. Diseño de la PCB

Una vez vistos todos los circuitos y estructuras que formarán la electrónica del robot podemos desarrollar nuestro circuito impreso con Eagle.

Primero hacemos el esquema en donde pondremos todos los componentes y las conexiones entre ellos. Además de los descritos arriba se añade:

- Un conector D50 hembra para conectar el brazo robot.
- Tres leds rojos con tres resistencias de 330Ω con común a masa que servirán como pilotos para distintos estados del robot.

- Más conectores donde, por medio de cables, se conectarán las salidas y entradas al Ardunio así como la fuente de tensión para alimentar los motores.

Una vez hecho esto se pasa al diseño del circuito impreso. Se acomodan dentro del espacio de la placa de 15 x 20cm todos los componentes de la manera más cómoda y eficiente posible. Dado el número de componentes y la complejidad del circuito se usa una placa de dos caras.

Para hacer las conexiones en la PCB, también por la complejidad de este, usamos la función *autoruter* que incorpora el programa Eagle. Esta función diseña automáticamente las rutas y las vías por ambas caras del placa con las opciones que se le indican. Después de múltiples pruebas se llego al resultado final que puede verse en el Apéndice C.

Una vez se tiene el circuito impreso se imprime en papel para transparencias con una impresora láser. Se imprimen dos circuitos: el de la cara superior y el de la inferior. Como PCB se usará una de tamaño 15x20cm fotosensible por las dos caras. Se retiran los papeles protectores de ambas caras y colocamos los circuitos que hemos imprimido de forma que casen perfectamente. A continuación se mete en una insoladora durante un minuto y medio por cada cada. Esta insoladora ilumina la placa con luz ultravioleta de forma que el circuito situado encima quede grabado.

Pasado este tiempo, sacamos la placa de la insoladora, retiramos el papel para transparencias con los circuitos e introducimos la placa en una solución de agua con sosa caustica hasta que aparezca el circuito sobre ella. Se retira y se limpia con agua y papel para después introducirla en otra solución de agua con percloruro de hierro, que se comerá el cobre sobrante de la placa.

Para terminar, hacemos los agujeros en la placa con una taladradora y brocas de distinto grosor (0.8, 1.0 y 1.2mm) y soldamos los componentes con estaño.

3.3.1. Correcciones sobre la PCB

Terminada la placa con la ayuda de un polímetro se revisan que todas las conexiones sean correctas y se comprueba que no se hayan formado cortocircuitos, corrigiendo aquellos que se hayan formado durante la soldadura.

Además, con el ataque con el ácido, algunas partes no se borraron correctamente mientras que algunas rutas se borraron accidentalmente. También esto tuvo que ser corregido, raspando las partes sobrantes y uniendo con cables soldados las rutas borradas.

Cometí un error en el esquemático del circuito. Como he dicho anteriormente, los pines 2 y 10 del LMD18200 son los *outputs* que se conectan a los motores. Por equivocación, en lugar de estos conecté los pines 1 y 11, que se corresponden con los *bootstrap*, salvo en el motor número 4. Una vez hecho la PCB tuve que borrar las pistas erróneas y mediante cable soldado realizar las conexiones adecuadas.

Por último, las conexiones a masa de los microswitches, según la tabla de conexiones del SCORBOT-ER III, eran en un principio las correspondientes a los pines número 33, 32, 31, 29 y 28 del conector D50, las mismas que para los leds, y como tal hice las conexiones en el esquemático. Sin embargo, posiblemente en algún proyecto anterior con este mismo robot, alguien cambio estas conexiones a las de los pines que se muestran el la tabla Tabla B.1. Una vez dado cuenta de este hecho la solución fue sencilla, conectar estos pines masa directamente en la PCB.

3.4. Conexiones

En la PCB tenemos cuatro *PinHD,* tres de 13 pines para las salidas y las entradas de los encoders, que se recogen en la Tabla 3.1, microswitches y LMD18200 y uno de 7 pines para la entrada de voltaje y de los LEDs, que se puede ven la Tabla 3.2. En el anexo B están las conexiones del conecto D50 del SCORBOT.

Tabla 3.1: Pines del circuito impreso y Arduino Mega

Señales del brazo robot				**Pines Arduino**	**PinesHD**	
Motor #	**Encoder**	**MS #**	**Sensor intensidad**	**Pin #**	**PinHD #**	**Pin #**
1 Dir. PWM				2 3	1	8 9
2 Dir. PWM				4 5	2	1 2
3 Dir. PWM.				7 6	2	8 9
4 Dir. PWM				8 9	3	1 2
5 Dir. PWM				12 10	1	1 2
Grp PWM Dir.				13 11	3	8 9
	1 P_1 P_0			22 23	1	11 12
	2 P_1 P_0			24 25	2	4 5
	3 P_1 P_0			26 27	2	11 12
	4 P_1 P_0			28 29	3	4 5
	5 P_1 P_0			30 31	1	1 2
	Grp P_1 P_0			32 33	3	11 12
		1		14	1	13
		2		15	2	6
		3		16	2	13
		4		17	3	6
		5		18	1	6
		Grp		-	3	13
			1		1	10
			2		2	3
			3		2	10
			4		3	3
			5		1	3
			Grp		3	10

Tabla 3.2: Conexiones PinHD 4

Señal	PinHD	Pin Arduino
12 VDC fuente tensión	1	
GND fuente tensión	2	
5VDC Arduino	3	5VDC
GND Arduino	4	GND
LED 1	5	49
LED 2	6	48
LED 3	7	50

4.Cinemática

4.1. Introducción

En este capítulo se estudiarán las ecuaciones y las matrices de transformación necesarias para el posicionamiento del brazo robot. Contamos para ello con dos métodos: por cinemática directa y por cinemática inversa.

La cinemática directa permite conocer la posición y la orientación del extremo de nuestra herramienta a partir de los ángulos de los distintos eslabones. Por el contrario, con la cinemática inversa a partir de la posición y la orientación que deseemos en nuestra herramienta se calculan los ángulos de giro del robot. Es, por tanto, esté último el de mayor interés.

Se usará el algoritmo de Denavit-Hatenberg para obtener el modelo cinemático directo del SCORBOT-ER III y a partir de él y con ecuaciones trigonométricas se calculará su modelo cinemático inverso.

4.2. Problema cinemático directo [5]

Se utiliza fundamentalmente el álgebra vectorial y matricial para representar y describir la localización de un objeto en el espacio tridimensional con respecto a un sistema de referencia fijo. Dado que un robot se puede considerar como una cadena cinemática formada por objetos rígidos o eslabones unidos entre sí mediante articulaciones, se puede establecer un sistema de referencia fijo situado en la base del robot y describir la localización de cada uno de los eslabones con respecto a dicho sistema de referencia. De esta forma, el problema cinemático directo se reduce a encontrar una matriz homogénea de transformación **T** que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo de la base del mismo. Esta matriz **T** será función de las coordenadas articulares.

La resolución del problema cinemático directo consiste en encontrar las relaciones que permiten conocer la localización espacial del extremo del robot a partir de los valores de sus coordenadas articulares.

Así, si se han escogido coordenadas cartesianas y ángulos de Euler para representar la posición y orientación del extremo de un robot de 5 grados de libertar, la solución al problema cinemático directo vendrá dada por las relaciones:

$$\begin{aligned} x &= f_x(q_1, q_2, q_3, q_4, q_5) \\ y &= f_y(q_1, q_2, q_3, q_4, q_5) \\ z &= f_z(q_1, q_2, q_3, q_4, q_5) \\ \alpha &= f_\alpha(q_1, q_2, q_3, q_4, q_5) \\ \beta &= f_\beta(q_1, q_2, q_3, q_4, q_5) \end{aligned} \tag{4.1}$$

Un robot de n grados de libertad está formado por n eslabones unidos por n articulaciones, de forma que cada par articulación-eslabón constituyen un grado de libertad. A cada eslabón se le puede asociar un sistema de referencia solidario a él y, utilizando las transformaciones homogéneas, es posible representar las rotaciones y traslaciones relativas entre los distintos eslabones que componen el robot. Normalmente la matriz de transformación homogénea que representa la posición y orientación relativa entre los sistemas asociados a dos eslabones consecutivos del robot se suele denominar matriz $^{i-1}\mathbf{A}_i$. Así pues, $^{0}\mathbf{A}_1$ describe la posición y orientación del sistema de referencia solidario al primer eslabón con respecto al sistema de referencia de la base y así sucesivamente. Del mismo modo, denominando $^{0}\mathbf{A}_k$ a las matrices resultantes del producto de las matrices $^{i-1}\mathbf{A}_i$ con i desde 1 hasta k, se puede representar de forma total o parcial la cadena cinemática que forma el robot. Así, por ejemplo, la posición y orientación del sistema solidario con el segundo eslabón del robot con respecto al sistema de coordenadas de la base se puede expresar mediante la matriz $^{0}\mathbf{A}_2$:

$$^{0}\mathbf{A}_2 = {}^{0}\mathbf{A}_1 {}^{1}\mathbf{A}_2 \tag{4.2}$$

Cuando se consideran todos los grados de libertad, la matriz $^{0}\mathbf{A}_n$ se le suele denominar **T**. Así, dado un robot de cinco grados de libertad, se tiene que la posición y orientación del eslabón final vendrá dada por la matriz T:

$$\mathbf{T} = {}^{0}\mathbf{A}_5 = {}^{0}\mathbf{A}_1 {}^{1}\mathbf{A}_2 {}^{2}\mathbf{A}_3 {}^{3}\mathbf{A}_4 {}^{4}\mathbf{A}_5 \tag{4.3}$$

Aunque para describir la relación que existe entre dos elementos contiguos se puede hacer uso de cualquier sistema de referencia ligado a cada elemento, la forma habitual que se suele utilizar en robótica es la representación de Denavit-Hatenberg (D-H). Denavit y Hatenberg propusieron en 1955 un método matricial que permite establecer de manera sistemática un sistema de coordenadas $\{S_i\}$ ligado a cada eslabón i de una cadena articulada, pudiéndose determinar a continuación las ecuaciones cinemáticas de la cadena completa.

Según la representación de D-H, escogiendo adecuadamente los sistemas de coordenadas asociados a cada eslabón, será posible pasar de una al siguiente mediante cuatro transformaciones básicas que dependen exclusivamente de las características de cada eslabón.

Estás transformaciones sucesivas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento i con el sistema del elemento i-1. Las transformaciones en cuestión son las siguientes:

1. Rotación alrededor del eje $\mathbf{z}_{i-1}$ un ángulo θ_i.
2. Traslación a lo largo de $\mathbf{z}_{i-1}$ una distancia d_i ; vector $\mathbf{d}_i(0,0,d_i)$.
3. Traslación a lo largo de $\mathbf{x}_i$ una distancia a_i ; vetor $\mathbf{a}_i(0,0,a_i)$.
4. Rotación alrededor del eje $\mathbf{x}_i$ un ángulo α_i.

Dado que el producto de matrices no es conmutativo, las transformaciones se han de realizar en el orden indicado. De este modo se tiene que:

$$\mathbf{^{i-1}A_i} = \mathbf{T}(\mathbf{z}, \theta_i)\mathbf{T}(0,0,d_i)\mathbf{T}(a_i,0,0)\mathbf{T}(\mathbf{x}, \alpha_i) \tag{4.4}$$

y realizando el producto entre matrices:

$$\mathbf{^{i-1}A_i} = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} C_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

donde θ_i, a_i, d_i, α_i son los parámetros D-H del eslabón i. De este modo, basta con identificar dichos parámetros para obtener las matrices **A** y relacionar así todos y cada uno de los eslabones del robot.

La información que nos ofrece la matriz $^{i-1}\mathbf{A}_i$ es la siguiente:

- La primera columna es el vector ortonormal de la dirección x_i respecto al sistema S_{i-1}.
- La segunda columna es el vector ortonormal de la dirección y_i respecto al sistema S_{i-1}.
- La tercera columna es el vector ortonormal de la dirección z_i respecto al sistema S_{i-1}.
- La cuarta columna indicada el origen del sistema de coordenadas S_i respecto de S_{i-1}.

De esta forma es como la matriz $^0\mathbf{A}_n$ nos da la posición y la orientación de la herramienta respecto al origen en la base.

4.2.1.Algoritmo de Denavit-Hartenberg para la obtención del modelo cinemático directo

D-H 1. Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerará como eslabón 0 a la base fija del robot.

D-H 2. Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad) y acabando en n.

D-H 3. Localizar el eje de cada articulación. Si ésta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

D-H 4. Para i de 0 a n-1, situar el eje z_i sobre el eje de la articulación i+1.

D-H 5. Situar el origen del sistema de la base $\{S_0\}$ en cualquier pinto del eje z_0. Los ejes x_0 e y_0 se situarán de modo que formen un sistema dextrógiro con z_0.

D-H 6. Para i de 1 n-1, situar el sistema $\{S_i\}$ (solidario al eslabón i) en la intersección del eje z_i con la línea normal común a z_{i-1} y z_i. Si ambos ejes se cortasen se situaría {Si} en el punto de corte. Si fuesen paralelos $\{S_i\}$ se situaría en la articulación i+1.

D-H 7. Situar x_i en la linea normal común a z_{i-1} y z_i.

D-H 8. Situar y_i de modo que forme un sistema dextrógiro x_i y z_i.

D-H 9. Situar el sistema $\{S_n\}$ en el extremo del robot de modo que z_n coincida con la dirección de z_{n-1} y x_n sea norma a z_{n-1} y z_n.

D-H 10. Obtener θ_i como el ángulo que hay que girar en torno a z_{i-1} para que x_{i-1} y x_i queden paralelos.

D-H 11. Obtener d_i como la distancia, medida a lo largo del eje z_{i-1}, que habría que desplazar $\{S_{i-1}\}$ para que x_i y x_{i-1} quedasen alineados.

D-H 12. Obtener a_i como la distancia medida a lo largo de x_i (que ahora coincidiría con x_{i-1}) que habría que desplazar el nuevo $\{S_{i-1}\}$ para que su origen coincidiese con $\{S_i\}$.

D-H 13. Obtener α_i como el ángulo que habría que girar en torno a x_i (que ahora coincidiría con x_{i-1}) para que el nuevo $\{S_{i-1}\}$ coincidiese totalmente con {Si}.

D-H 14. Obtener las matrices de transformación ${}^{i-1}\mathbf{A}_i$ definidas en (4.5).

D-H 15. Obtener la matriz de transformación que relaciona el sistema de la base con el extremo del robot $\mathbf{T} = {}^{0}\mathbf{A}_1\ {}^{1}\mathbf{A}_2 \ldots {}^{n-1}\mathbf{A}_n$.

D-H 16. La matriz **T** define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de las n coordenadas articulares.

Siguiendo estos pasos obtenemos las coordenadas articulares para SCORBOT-ER III que pueden verse en la Figura 4.1. Los parámetro de Denavit-Hatenberg se encuentran en la Tabla 4.1. El problema cinemático directo se resolverá posteriormente con MATLAB.

Tabla 4.1: Parámetros D-H del SCORBOT-ER III.

i	**θ_i**	**d_i**	**a_i**	**α_i**
1	q_1	l_0+l_1	a_1	$-\pi/2$
2	$q_2-\pi/2$	0	l_2	0
3	$-q_2+q_3+\pi/2$	0	l_3	0
4	$-q_4-q_3$	0	0	$-\pi/2$
5	q_5	l_4+l_5	0	0

donde los parámetros l_0, l_1, l_2, l_3, l_4, l_5, y a_1 son las distancias entre los orígenes de los ejes articulares del SCORBOT-ER III que pueden verse en la Figura 4.2 y la Tabla 4.2.

Tabla 4.2: Longitudes del SCORBOT-ER III.

Longitud	**Tamaño (cm)**
l_0	21
l_1	14
l_2	22,5
l_3	22,5
l_4	7,5
l_5	6,5

Longitud	Tamaño (cm)
a_1	2,5

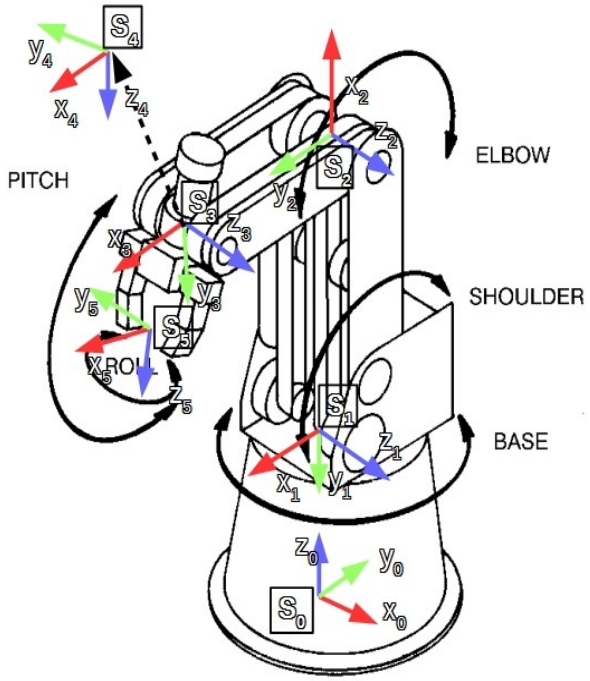

Figura 4.1: Coordenadas articulares SCORBOT-ER III

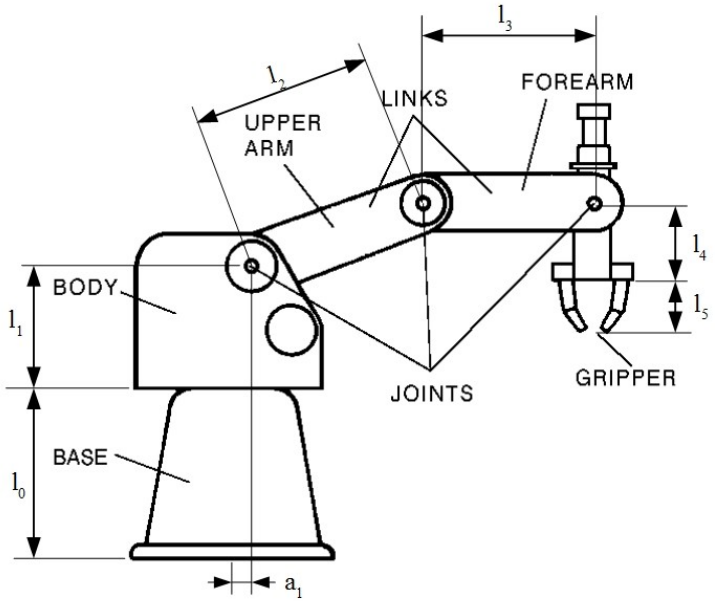

Figura 4.2: Longitudes SCORBOT-ER III.

4.3. Problema cinemático inverso [5], [6]

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

Así como es posible abordar el problema cinemático directo de una manera sistemática a partir de la utilización de matrices de transformación homogéneas, e independientemente de la configuración del robot, no ocurre los mismo con el problema cinemático inverso, siendo el procedimiento de obtención de las ecuaciones fuertemente dependiente de la configuración del robot.

A la hora de resolver el problema cinemático inverso es mucho más adecuado encontrar una solución cerrada. Esto es, encontrar una relación matemática explicita de la forma:

$$\begin{gathered} q_k = f_k(x, y, z, \alpha, \beta) \\ k = 1, ..., 5(GDL) \end{gathered} \tag{4.6}$$

Este tipo de solución presenta, entre otras cosas, las siguientes ventajas:

1. En muchas ocasiones, el problema cinemático inverso ha de resolverse en tiempo real. Una solución de tipo iterativo no garantiza tener la solución en el momento adecuado.
2. Con frecuencia la solución al problema cinemático inverso no es única, existiendo diferentes duplas que posicionan y orientan el extremo del mismo modo. En estos casos una solución cerrada permite incluir determinadas reglas o restricciones que aseguren que la solución obtenida sea la más adecuada.

En el caso de robots antropomórficos como SCORBOT-ER III tenemos dos formas de posicionar el extremo: con el codo hacia arriba y con el codo hacia abajo. Por las características de nuestro robot usaremos la configuración de **codo hacia arriba.**

Para posicionar nuestro brazo robot de 5 GDL necesitaremos, a su vez, 5 parámetros: sus coordenadas en el espacio (p_x, p_y, p_z) y su orientación que vendrá dada por los dos últimos grados de rotación q_5 y q_6 . Introducidos estos parámetros se calcularán q_1, q_2 y q_3 por métodos geométricos.

El primer ángulo es sencillo de obtener con las coordenadas p_x y p_y. Se usará el comando arcotangente por ser más adecuado a la hora de la computación.

$$q_1 = atan\left(\frac{p_y}{p_x}\right) \tag{4.7}$$

Para los ángulos q_2 y q_3 en primer lugar calculamos la distancia que hay desde el origen del sistema S_1 hasta el origen de la herramienta S_3, la altura y su proyección en base, que se corresponderán con los valores de r, z_3 y R.

$$\begin{gathered} R = \sqrt{p_x^2 + p_y^2} - (l_5 + l_6)\sin q_4 - a_1 \\ z_1 3 = p_z + (l_5 + l_6)\cos q_4 - (l_1 - l_2) \\ r = \sqrt{R^2 + z_3^2} \end{gathered} \tag{4.8}$$

Calculamos el segundo ángulo:

En primer lugar calculamos el ángulo α:

$$\cos\alpha = \frac{z_{13}}{r}$$

$$\sin\alpha = \frac{R}{r} \tag{4.9}$$

$$\cos\alpha = atan\frac{\sin\alpha}{\cos\alpha}$$

A continuación el ángulo β:

$$(r - l_2\cos\beta)^2 + (l_2\sin\beta)^2 = l_3^2 \rightarrow l_3^2 = r^2 + l_2^2 - 2rl_2\cos\beta \rightarrow$$

$$\rightarrow \cos\beta = \frac{r^2+l_2^2-l_3^2}{2rl_2}$$

$$\sin\beta = \sqrt{1-\cos\beta^2} \tag{4.10}$$

$$\beta = atan\frac{\sin\beta}{\cos\beta}$$

El ángulo q2 no es es más que la diferencia de α y β:

$$q_2 = \alpha - \beta \tag{4.11}$$

Ya solo queda por calcular el ángulo q3:

$$\cos q_3 = \frac{R-l_2\sin q_2}{l_3}$$

$$\sin q_3 = \frac{l_2\sin q_2 - z_13}{l_3} \tag{4.12}$$

$$q_3 = atan\frac{\sin q_3}{\cos q_3}$$

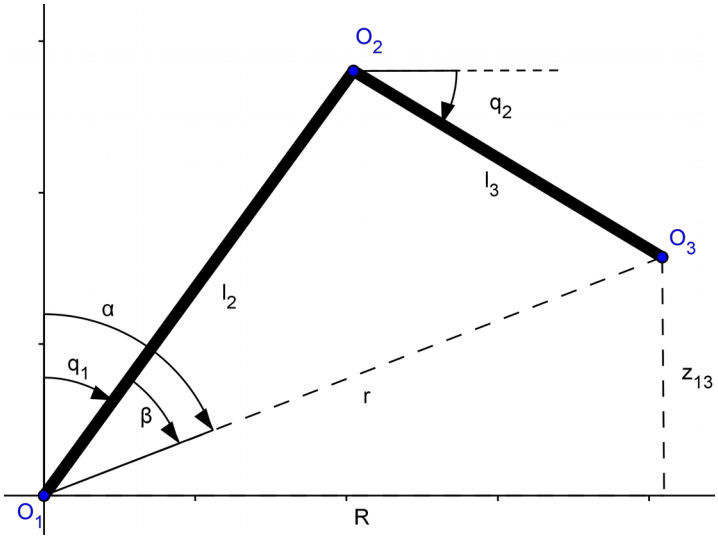

Figura 4.3: Esquema para el cálculo de q_1 y q_2.

Teniendo todas las ecuaciones necesarias para resolver el problema cinemático inverso posteriormente se introducirán en MATLAB y desde allí se resolverán.

5. Programación

5.1. Introducción

En los capítulos anteriores hemos visto el funcionamiento de nuestro robot y hemos hecho el circuito electrónico adecuado para su control. También hemos indicado las distintas conexiones con Arduino Mega y hemos resuelto su problema cinemático tanto directo como inverso para posicionarlo correctamente.

En este capítulo abordaremos la parte de programación del SCORBOT-ER III, donde se hara uso de lo expuesto anteriormente para dejar el brazo mecánico en un estado funcional.

Para la programación utilizaremos por un lado el microprocesador de código abierto **Arduino Mega 2560** y por otro el software matemática **MATLAB**, cuya facilidad para trabajar con matrices y los paquetes que permiten la comunicación con Arduino nos ayudaran a la hora de la programación. Además, MATLAB nos ofrece la posibilidad de controlar Arduino en tiempo real.

5.2. Arduino Mega 2560

La compañía *Mathworks*, desarrolla de MATLAB, ha creado un *support package* o paquete de soporte de Arduino para MATLAB llamado **ArduinoIO**. Este paquete permite la comunicación a través de puerto serie de 115200 baudios entre el software y el microprocesador.

Este paquete contiene los archivos necesarios para su instalación en MATLAB y un archivo llamado *adioes.ino* para cargar en Arduino. Cargado este código en el Arduino ya es posible su manejo con MATLAB, sin embargo surge el problema en esta aplicación de que el tiempo de adquisición de datos por parte de MATLAB a través del puerto serie es superior a la velocidad con que los encoders cambian de estado. Esto hace que no se puedan medir los pulsos de los encoders ni, por tanto, calcular la posición del robot con MATLAB.

La solución a este problema consiste en modificar el archivo *adioes.ino* de forma que los pulsos de los encoders se contabilicen directamente con Arduino y este conteo se envíe por puerto serie a MATLAB cuando este lo solicite.

El archivo *adioes_modificado.ino* se adjunta a este documento y las modificaciones realizadas en el mismo son las siguientes:

- Arrays para almacenar las señales de los seis encoders para las entradas P_1 y P_0, sus estados previos, contabilizar los pulsos y almacenar los pines correspondientes de cada uno. Lo añadimos al inicio, en la declaración de variables, antes de *void setup()*:

Código 5.1: Modificación adios.ino, arrays de los encoders.

```
// Array para almacenar el estado de los encoders {P1,P0} (0 ó 1)
int EncoderState[6][2] = {{0,0},{0,0},{0,0},{0,0},{0,0},{0,0}};
// Array para almacenar el estado previo de los encoders P1
int prevEncoderState[6] = {0,0,0,0,0,0};
// Array para contabilizar los pulsos de los encoders
long EncoderCounter[6] = {0,0,0,0,0,0};
// Pines de los encoders {P1,P0}
int pinEncoder[6][2] = {{22,23},{24,25},{26,27},{28,29},{30,31},{32,33}};
```

- Dentro del *switch(s)* que se encuentra dentro del *void loop()* se programa el envío de información de los encoder a MATLAB y la función para devolver la cuenta a 0. Cuando por puerto serie Arduino lee un 100 seguido de un número entre 1 y 6, que hace referencia a la información del encoder que se necesita, se envía el valor del contador de dicho encoder. Si en su lugar llega un 110 seguido de un número entre 1 y 6 lo que se hace es resetear el contador del encoder que se indica.

Código 5.2: Modificación adioes.ino, comunicación por puerto serie.

```
/*--------------MODIFICACIÓN PARA EL ENVIO DE LOS ENCODERS-------------*/

/* s=100 obtiene conteo almacenado pulsos encoder      */
      case 100:
          if(val>0 && val<7){
             Serial.println(EncoderCounter[val-1]);/* send value via serial*/
          }
          s=-1;  /* we are done with this so next state is -1    */
      break; /* s=100 taken care of                              */

    /* s=110 inicia contador                                     */
      case 110:
        if(val>0 && val<7){
            EncoderCounter[val-1]=0;
        }
        s=-1;  /* we are done with this so next state is -1    */
  break;/* s=100 taken care of                                  */
```

- Al final del *void loop()* tenemos el código para contabilizar los pulsos de los encoder. Como su el encoder número 6 o *Grp* de la pinza solo tiene una entrada disponible se programa aparte y siempre incrementa su valor.

Código 5.3: Modificación adioes.ino, conteo de los encoders.

```
 /*------------------CONTEO DE LOS ENCODERS--------------------*/

  //Para los encoders del 1 al 5
  for(int i=0; i<5; i++){
    //Almacenar en prevEncoder el estado del Encoder P0
    prevEncoderState[i]=EncoderState[i][0];

    //Leer el estado de los encoders y almacenarlos en EncoderState
    EncoderState[i][0]=digitalRead(pinEncoder[i][0]);
    EncoderState[i][1]=digitalRead(pinEncoder[i][1]);

    /*
     * Aqui contbilizamos los pulsos de los encoders del 1 al 5:
     * Se activa cuando el estado del encoder P1 es distinto que su estado
      *anterior.
     * Si su estado el estado de P0 es 1 y P1 es 0 o si P1 es 0 y P0 es 1 se
     * resta 1 en EncoderCountere. En caso contradrio se suma 1.
     */
    if(EncoderState[i][0]!= prevEncoderState[i]){
      if((EncoderState[i][0]==1 && EncoderState[i][1]==0)||(EncoderState[i]
[0]==0 && EncoderState[i][1]==1)){
        EncoderCounter[i] -=1;
      }else{
        EncoderCounter[i] +=1;
      }
    }
  }

  //Encoder Pinza
  /*En el caso de la Pinza cada vez que su estado es mayor que el
   * anterior se suma 1 a su EncoderCounter.
   */
    prevEncoderState[5]=EncoderState[5][0];
    EncoderState[5][0]=digitalRead(pinEncoder[5][0]);

    if(EncoderState[5][0]!= prevEncoderState[5]){
      EncoderCounter[5] +=1;
    }

}/* end loop statement   */
```

Con esto tenemos todas las modificaciones necesarias en el código de Arduino. Pasamos a programas con MATLAB.

5.3. MATLAB

MATLAB es el lenguaje de alto nivel y el entorno interactivo utilizado por millones de ingenieros y científicos en todo el mundo. Le permite explorar y visualizar ideas, así como colaborar interdisciplinariamente en procesamiento de señales e imagen, comunicaciones, sistemas de control y finanzas computaciones. [7]

Existen para MATLAB multitud de *support packages* para multitud de hardware y una gran cantidad de herramientas. El que se usará en este caso es ArduinoIO, como ya se ha comentado. Dentro de la carpeta que contiene este paquete encontramos dos archivos con extensión *.m,* la usada por MATLAB:

install_arduino.m y *arduino.m*. Ejecutando el primero con MATLAB se instala el paquete y el segundo contiene las funciones para la comunicación con Arduino.

La programación de MATLAB se hará a través de funciones. Las funciones son programas que pueden tener entradas y salidas y que son llamados a través de la consola de MATLAB. También se pueden llamar usos de funciones a través de otras funciones.

Se empezará con funciones básicas, para inicializar el Arduino, leer los contadores de los encoders o activar y desactivar los motes; y se irán haciendo otras cada vez más complejas construidas a partir de las anteriores hasta poder controlar la posición de la herramienta. También, se programará el calculo de la cinemática directa e inversa. Por último se hará una interfaz gráfica o *GUIDE* para un control más cómodo y simple del robot.

5.3.1. Modificación de "arduino.m"

Antes de empezar hay que hacer una pequeña modificación en el código *arduino.m* antes descrito. En la línea 7 se cambia las acceso de propiedades de *private* a *public*. Así podemos acceder a propiedades de Arduino que posteriormente necesitaremos desde cualquier otra función.

```
1   classdef arduino < handle
2
3       % This class defines an "arduino" object
4       % Giampiero Campa, Aug 2013, Copyright 2013 The MathWorks, Inc.
5
6       % Se cambia SetAcces a "public"
7       properties (SetAccess=public,GetAccess=public)
8           aser    % Serial Connection
9           pins    % Pin Status Vector
10          srvs    % Servo Status Vector
11          mspd    % DC Motors Speed Status
12          sspd    % Stepper Motors Speed Status
13          encs    % Encoders Status
14          sktc    % Motor Server Running on the Arduino Board
15      end
```

Figura 5.1: Modificación "arduino.m".

Las principales funciones que se usarán de *arduino.m* son las siguientes:

- `a=arduino('PUERTO')`: con esta función establecemos la conexión con Arduino a través de la variable a, donde se almacenarán las propiedades y las funciones de arduino y a la que llamaremos siempre que queramos controlarlo.
- `pinMode(a,pin,'output')`: se establece el pin indicado como salida digital.
- `digitalWrite(a,pin,estado)`: se escribe un pin de salida digital. El estado puede ser 0 o 1.
- `digitalRead(a,pin)`: se hace la lectuara de un pin de entrada digital. Devuelve 0 o 1.
- `analogWrite(a,pin,PWM)`: se hace la escritura de un pin de salida analógico. El valor de PWM es un número entero entre 0 y 255, siendo 0 una salida de 0VDC y 255 una salida de 5VDC por el pin correspondiente.

5.3.2. IniciarArduino.m

Llamando a esta función estableceremos la conexión entre Arduino y MATLAB y estableceremos los pines que servirán como salidas digitales (para la dirección) y digitales (PWM del LMD18200).

En primer se crea una variable global *"a"* que contendrá todo lo que contendrá todas las propiedes y funciones de *"arduino.m"*. Hay que indicar el puerto serie por el que se conectará Arduino, el COM5 en este caso, aunque podría ser distinto, en cuyo caso habría que poner el que corresponda.

Después de crear un *array* o vector con los pines de las conexiones PWM y la dirección estableces que los pines de la dirección serán digitales e iniciamos a 0 tanto dirección como PWM para que el brazo no se mueva.

Se añade también un juego de luces con los leds y un mensaje de que se indica que se ha establecido la conexión en la consola. El led 1 se deja encendido como piloto de que Arduino y MATLAB están conectados.

Es a partir de que se establezca la conexión cuando **puede conectarse la fuente de tensión**, de lo contrario los PWM de los LMD18200 pueden encontrarse es estado de alta impedancia y el robot se movería sin control.

A continuación puede verse el código:

Código 5.4: Funció iniciarArduino.m

```
function iniciarArduino

    % Crear la variable global a como
    % un objeto arduino. Puerto COM5.
    global a;
    a=arduino('COM5');

    %Pines de PWM, dirección y los LEDs rojos
    pinPWM=[3 5 6 9 10 11];
    pinDireccion=[2 4 7 8 12 13];
    pinLed=[50 51 52];

    %Establecer los pines como salidas analogicas y digitales

    %Para los motores del 1 al 6
    for i=1:6
        %Establecer los pines como salidas digitales
        pinMode(a,pinDireccion(i),'output');
        %Poner las direcciones y los PWM a 0
        digitalWrite(a,pinDireccion(i),0);
        analogWrite(a,pinPWM(i),0);
    end

    %Secuencia de luces con los LEDs
    for i=1:3
        % Se establecen los pines de los LEDs
        % como salidas digitales
        pinMode(a,pinLed(i),'output');
```

```
        %Se encienden secuencialmente durante
        %1/4 de segundo y se apagan
        digitalWrite(a,pinLed(i),1);
        pause(0.25);
        digitalWrite(a,pinLed(i),0);
    end

    % Mantenmos el LED 1 encendido indicando que
    % que el Arduino está conectado.
     digitalWrite(a,pinLed(1),1);

     %Imprimimos por consola que el Arduino está conectado.
     fprintf('Arduino conectado\n');
end
```

5.3.3. desconectarArduino.m

Igual que hay una función para conectar Arduino hay una para desconectarlo. Sencillamente se trata de eliminar la variable global *a* antes creada.

La fuente de tensión debe desconectarse antes de llamar a esta función, porque como pasaba antes de iniciar Arduino no podemos saber el estado de los PWM y el robot puede moverse erráticamente.

Código 5.5: Función *desconectarArduino.m*

```
function desconectarArduino
    %Imprimimos el mensaje de que estamos
    %desconectando arduino
    fprintf('Desconectando arduino...');
    global a; %Se llama a la variable global a

    %con delete y clearvars la eliminamos
    delete(a);
    clearvars -global a;

    % Imprimimos por consola que se ha desconectado
    %exitosamente.
    fprintf('\nArduino desconectado\n');
end
```

5.3.4. motor.m

Con la función *motor.m* se activan y desactivan fácilmente los motores además de su dirección. Para que los motores se muevan debemos poner el pin correspondiente a la dirección de su LMD18200 un 0 o 1 para que se mueva en un sentido u en otro y la señal PWM también por el pin de Arduino correspondiente según tensión que queremos que llegue a los motores y también su velocidad, como se explicó anteriormente.

Esta función tiene tres entradas:

- *num*: indicamos el número del motor que queremos mover o parar. Debe ser un número entero entre 1 y 6.
- *direccion*: dirección del motor. Puede ser 0 o 1.

- *PWM:* porcentaje de PWM que enviamos al LMD18200 en tanto por uno. Valores entre 0,0 y 1,0 , siendo 0,0 que la tensión sea 0 y el motor esté parado; y 1,1 que esta sea 12V y el motor se mueva a la máxima velocidad posible. Posteriormente se multiplica por 255 y se redondea al entero.

La dirección de movimiento de cada motor y el comportamiento de su encoder se pueden ver en la siguiente tabla:

Tabla 5.1: Características de los motores.

Motor #	Reducción[1]	Límites	Dirección	Movimiento	Contador encoder
1	1:600	-180º, 140º	0 1	Sentido antihorario Sentido horario[2]	Decrece Crece
2	1:523	-30º, 120º	0 1	Baja Sube	Decrece Crece
3	1:523	-15º, ver (5.2)	0 1	Sube Baja	Decrece Crece
4 + 5	1:120	0º, 180º Ilimitado	0 0 1 1 0 1 1 0	Sube Baja Giro antihorario Giro horario	Decrece – Crece Crece – Decrece Decrece – Decrece Crece – Crece
Grp o 6	-	Abierto/ Cerrado	0 1	Cierra pinza Abre pinza	

Código 5.6: Función *motor.m*

```
function motor( num, direccion, PWM )
    %Con la función motor se controla el movimiento de cada motor
    %Tenemos tres entradas:
    % "num": numero del motor que se quiere controlar (1,2,...,6)
    % "direccion": direccion del motor, 0 o 1.
    % "PWM": porcentaje de PWM, entre 0.0 y 1.0

    %Llamar a la variable global a
    global a;

    % Pines de PWM y Dirección de cada motor
    pinPWM=[3 5 6 9 10 11];
    pinDireccion=[2 4 7 8 12 13];

    % Se establece la dirección del motor correspondiente (0 o 1)
    digitalWrite(a,pinDireccion(num),direccion);

    % Se estable el PWM del motor correspondiente (entre 0 y 255)
    analogWrite(a,pinPWM(num),round(PWM*255));

end
```

1 Vueltas del eslabón : vueltas del encoder
2 Sentido horario o antihorario visto desde arriba.

5.3.5. encoder.m

Con esta función enviamos el número del encoder que queremos leer y recibimos el valor del contador de dicho encoder.

Tenemos que enviar dos valores por puerto serie: el primero es el *mode* o la acción que queremos que haga Arduino, el segundo es el encoder que queremos leer.

Como se programó en la sección de Arduino el *mode* correspondiente para la enviar la información del contador de los encoder es 100, pero al valor que recibe le resta 48 y lo multiplica por 10, por lo que se tiene que enviar en primer lugar 48+10. El segundo valor es un número natural entre 1 y 6 para indicar el motor.

Para poder enviar y recibir mensajes por puerto serie se necesita hacer uso de la propiedad *aser* de arduino, por lo cual hicimos públicas sus propiedades al principio.

Código 5.7: Función *encoder.m*

```
% Con esta función leemos el contador de los encoders.
% Tenemos una entrada, num, con el numero del encoder
% y una salida, val, con el valor del contador.
function val=encoder(num)

    %Llamos a la variable a
    global a;

    % envimas el modo y el encoder que queremos leer
    % para ellos debemos hacer uso de a.aser, para lo
    % que se hizo public las propiedades de arduino.m
    fwrite(a.aser,[48+10 num], 'uchar');

    % Leemos el valor del contador que nos llega por
    % el puerto serie y lo sacamos.
    val =fscanf(a.aser,'%d');
end
```

5.3.6. resetEncoder.m

Hay ocasiones en los que necesitamos poner el contador de los encoders a 0, por ejemplo, al iniciar el robot y posicionarlo en su posición inicial o *hard home*. Para ellos haremos uso de esta función, que igual que la anterior recibe el numero del encoder a resetear y lo envia la instrucción correspondiente por puerto seria al Arduino.

En este caso el *mode* que programamos era el el 110, por lo que hay que enviar primero 48+11 para indicar que queremos poner el contador a 0 y después el numero del encoder.

Código 5.8: Función *resetEncoder.m*

```
%% Función para resetear los encoders
function resetEncoder(num)
    %Llmamos a la var global a
    global a;
```

```
    % Se envia por puerto serie la instrucción a Arduino
    fwrite(a.aser,[48+11 num], 'uchar');
end % fin de la función
```

5.3.7.pinza.m

Función para abrir y cerrar la pinza. Para la pinza solo contamos con un encoder, pero es suficiente.

Como entrada se introduce un carácter que puede ser "c" para cerrar o "a" para abrir. Según la opción se activa el motor de la pinza en una dirección o en la otra. Cada 0,1 segundos se mide se comprueba el encoder. Si el contador no aumenta es que ha llegado a su máximo, la pinza está abierta o cerrado y el motor se para.

Se activa el 2º LED rojo para indicar que el brazo está en movimiento.

Código 5.9: Función *pinza.m*

```
% Función para abrir o cerrar la pinza
function pinza(str)
    % str: 'a' para abrir, 'c' para cerrar

    % Encendemos el 2° LED rojo
    digitalWrite(a,51,1);

    % Se inicia el encoder 6 de la pinza a 0.
    resetEncoder(6);

    % Según la opción introducida se mueve el
    % motor en una dirección o en la otra.
    if str(1)=='a'
            motor(6,1,1)
       end
     if str(1)=='c'
            motor(6,0,1)
     end


     enc = encoder(6); % Se mide el contador del encoder
     pause(0.1); % Espera de 0.1 segundos
     enc_old = enc; % Se almacena la antigua lectura del encoder en enc_old
     enc = encoder(6); % Se vuelve a medir el encoder

     % Mientras que el contador del encoder vaya aumentando se repite
     % el proceso, tomando medidas cada 0.1 segundos.
     while enc >enc_old
        enc_old = enc;
        enc = encoder(6);
        pause(0.1);
     end

     % Cuadno el contador ya no aumenta más se ha llegado al maximo
     % y se para el motor.
     motor(6,0,0);

    % Se apaga el LED
    digitalWrite(a,51,0);
```

```
end
```

5.3.8.posInicial.m

Función para posicionar el robot en la posición inicial o *hard home* independientemente del lugar en que se encuentre. Esta posición sirve de referencia, será el punto 0 de los ángulos.

La operación que se realiza consiste en mover el motor en una dirección hasta que se pulse el microswitch correspondiente. A su vez, cada 0,01 segundos se comprueba el contador de los encoders.

Puede darse el caso de que se llegue al extremo sin haberse pulsado el microswitch. Para se comprueba que los contadores se incrementen o disminuyan. Si no cambia es que se ha llegado al final y se comienza a mover el motor en la dirección contraria.

Los motores 4 y 5 controlan a la vez ambos movimientos de la muñeca. Primero se mueven en la misma dirección para posicionar la pinza en el plano vertical y después en direcciones opuestas para posicionarla en el plano horizontal.

Para los casos 2, 3 y 4 el abultamiento que sirve para pulsar el microswitch no es un solo punto, sino que tiene cierta longitud, y no se alcanzará la misma disposición desde una dirección que desde la otra. Por ello, cuando el motor venga desde una de las direcciones y se pulse el microswitch no se detendrá, sino que seguirá moviéndose hasta que deje de estarlo y después girará de nuevo en la dirección inversa hasta que vuelva a estar pulsado. Por la otra dirección no habrá problemas.

Por otro la lado, la posición de la pinza en el plano horizontal no queda exactamente recta por lo que después de llegar a la posición de mantener el microswitch pulsado hay que ajustar su posición.

También se coloca la pinza en posición abierta.

Al finalizar se resetean los encoders a 0 y se crean dos variables gloabales, *pos4* y *pos5*, para guardar las orientaciones de la muñeca, también iniciadas a 0.

Durante el proceso se mantiene encendido el 3er LED rojo.

Código 5.10: Función *posInicial.m*

```
% Función para colocar el brazo en la posción inicial o "hard home"
% Cuando los microswitches están pulsados recibimos un 0
% Cuando no lo está, recibimos un 1

function posInicial()
    global a; % Cargar variable global a
    % Crear las variables globales pos4 y pos5 para la muñeca
    global pos4;
    global pos5;

    pinMS = [14,15,16,17,18]; % Pines de los microwitches
    digitalWrite(a,52,1); % Se enciende el tercer led rojo

    % Se imprime un mensaje por pantalla
    fprintf('Realizando el posicionamiento inicial');

    %%
```

```
%Posicionar Base
dir=0; % Mover primero en la dirección 0

% Mientras no se pulse el microswitch:
while digitalRead(a,pinMS(1))==1
    enc = encoder(1); % Se lee el contador del encoder
    motor(1,dir,1); % Se mueve el motor en la dirección inicial
    pause(0.01); % Pausa de 0.01 segundos
    % Si el contador no se ha incrementado hemos llegado al extremo
    % y se cambia la dirección del motor
    if enc == encoder(1)
        dir=1;
    end
end
% Cuando se pulsa el microswitch se para el motor.
motor(1,0,0)
fprintf('.');

%%
%Posicionar Antebrazo
dir2=0; % Mover primero en la dirección 0

% Mientras que la dirección del motor sea 0:
while dir2 == 0

    % Mientras se mantenga pulsado el boton el motor se moverá
    % en esa dirección
    while digitalRead(a,pinMS(2))==0
        motor(2,dir2,1);
    end

    dir2=1; % Se cambia la dirección a 1
    % Mientras el MS esté sin pulsar:
    while digitalRead(a,pinMS(2))==1
        enc = encoder(2);% Almacenar el valor del contador
        motor(2,dir2,1); % Mover el motor
        pause(0.01); % Pausar 0.01 segundos
        % Si el contador no ha cambiado cambiar la dirección
        if enc == encoder(2)
            dir2=0;
        end
    end
end
motor(2,0,0); %Parar el motor
fprintf('.');

%%
%Posicionar Brazo
dir3=0;% Mover primero en la dirección 0

% Mientras que la dirección del motor sea 0:
while dir3 == 0

    % Mientras se mantenga pulsado el boton el motor se moverá
    % en esa dirección
    while digitalRead(a,pinMS(3))==0
        motor(3,dir3,1);
    end
```

```
    dir3=1;  % Se cambia la dirección a 1

    % Mientras el MS esté sin pulsar:
    while digitalRead(a,pinMS(3))==1
        enc = encoder(3); % Almacenar el valor del contador
        motor(3,dir3,1); % Mover el motor
        pause(0.01); % Pausar 0.01 segundos
        % Si el contador no ha cambiado cambiar la dirección
        if enc == encoder(3)
            dir3=0;
        end
    end
end
motor(3,0,0); % Parar el motor
fprintf('.');
%%
%Posicionar herramienta (Plano vertical)
dir4=1; % Mover primero en la dirección 1

% Mientras que la dirección del motor sea 1:
while dir4 == 1

    % Mientras el MS esté pulsado los motores 4 y 5 se moverán
    % en esa dirección
    while digitalRead(a,pinMS(4))==0
        motor(4,dir4,1);
        motor(5,dir4,1);
    end

    dir4=0; % Se cambia la dirección a 0

    % Igual que en los casos anteriores, el motor se mueve en una
    % dirección. Si se llega al maximo antes de pulsar el MS, se
    % invierte.
    while digitalRead(a,pinMS(4))==1
        enc = encoder(5);
        motor(4,dir4,1);
        motor(5,dir4,1);
        pause(0.01);
        if enc == encoder(5)
            dir4=1;
        end
    end
end
motor(4,0,0); %Se para el motor 4
motor(5,0,0); %Se para el motor 5
fprintf('.');
%%
%Posicionar herramiente (plano horizontal)
    % Mientras no se pulse el MS los motores 4 y 5 se mueven en
    % dirección opuestas. Como no hay limite en el giro no es
    % necesario invertir las direcciones.
    while digitalRead(a,pinMS(5))==1
        motor(4,1,1);
        motor(5,0,1);
        pause(0.01);
    end
```

```
    motor(4,0,0); % Se para el motor 4
    motor(5,0,0); % Se para el motor 5

    %%
    %Abrir la pinza
    pinza('abrir');
    fprintf('.');
    %%
    %Resetear los encoders y poner a 0 los contadores pos4 y pos5
    for i=1:5
        resetEncoder(i)
    end
    fprintf('.');

    pos4 = 0;
    pos5 = 0;

    %%
    % Se ajusta la posición de la herramienta en el plano horizontal
    % y se vuelven a poner a 0 los contadores.
    Mover(5,-20,1);
    for i=1:5
        resetEncoder(i)
    end
    fprintf('.');

    pos4 = 0;
    pos5 = 0;
    %%
    digitalWrite(a,52,0); % Se apaga el LED

    % Se imprime un mensaje por pantalla
    fprintf('\nPosicionamiento realizado con exito!\n');
end
```

5.3.9. mover.m

La función mover se utilizará para que un determinado segmento se mueva hasta el ángulo en que se quiere posicionar, los ángulos q_i de los que se habló en la parte de cinemática. La función tienes tres entradas:

- *num*: número del eslabón que se quiere mover. Nótese que no hablamos del número de motor. Los motores 1, 2 y 3 se corresponden con el eslabón que se quiere mover (base, brazo o antebrazo) mientras que los motores 4 y 5 controlan conjuntamente la rotación (num = 4) y el giro (num = 5) de muñeca.
- *val*: ángulo al que se quiere girar (q_i), en grados.
- *PWM:* señal PWM del LMD18200 correspondiente en tanto por uno.

Para cada caso, excepto para el 4 y 5, el algoritmo es el mismo:

1. Se multiplica el ángulo introducido por la reducción correspondiente para conocer el número de pulsos del contador para dicho ángulo. El valor obtenido se almacena en la variable *pulsos*. No

existe ningún indica de esta reducción ni en el manual ni en trabajos anteriores con este robot. Por tanto, ha tenido que ser medida experimentalmente y de forma aproximada haciendo girar cada elemento 90º y contando los pulsos. Se indican en la Tabla 5.1.

La reducción está expresada en vueltas del eslabón por vueltas del encoder, es decir, el número de vueltas que tiene que dar el encoder por cada vuelta del eslabón. Dada la composición de la rueda del encoder y la lectura que se hace de él, por cada vuelta se detectan 12. La fórmula para calcular los pulsos es la siguiente.

$$\text{pulsos} = \frac{\text{val}}{360} \cdot \text{reducción} \cdot 12 \tag{5.1}$$

2. Se comprueba que el ángulo introducido esté dentro de los límites. Es cierto que en la Tabla 2.1 aparecen los ángulos máximos y mínimos que puede girar cada motor, sin embargo se usarán otros límites más restrictivos para evitar que se produzcan choques entre las partes con sus delimitadores físicos. Además, si el eslabón 3 girá demasiado hacia arriba la estructura mecánica del robot hace que la muñeca y el elemento terminal también se vean afectados, ocasionando luego problemas.

   El límite superior de giro de la articulación 3 también viene limitado por el de la articulación 2. Según sea q_2 el rango de q_3 podrá ser mayor o menor sin que el eslabón choque con la estructura del robot. Tomando los valores máximos de q_3 para varios q_2 que se muestran en la Tabla 5.2 se ha trazado la gráfica de la Figura 5.8 que se puede ver al final del capítulo y por interpolación polinómica de grado 2 se ha llegado a la siguiente expresión para $q_{3,máx}$:

$$q_{3,\text{máx}} = -0,00432864q_2^2 + 1,00469484q_2 + 64,2739366 \tag{5.2}$$

   con q_2 y $q_{3,máx}$ en grados.

Tabla 5.2: Medidas de $q_{3,máx}$ en función de q_2.

q_2 (º)	$q_{3,máx}$ (º)
-30	30
0	65
30	90
45	60
60	110
75	115

3. Puede darse el caso que la articulación se encuentre en el ángulo requerido, en tal caso no es necesario hacer nada. En caso contrario existen dos opciones:

   a) El ángulo que se desea es menor que aquel en el cual se encuentra, o lo que es lo mismo, que el valor de la variable *pulsos* sea menor que el del contador del encoder. En tal caso se hace girar el motor en la dirección para que el contador decrezca.

b) Lo contrario: el ángulo deseado es mayor que aquel en el que se encuentra, el valor de *pulsos* es ahora mayor que el del contador y el motor se tendrá que hacer girar de manera que este aumente.

4. Una vez que se alcanza la posición deseada el motor se desactiva poniendo el PWM a 0.

El caso de la muñeca es algo más complicado, porque se tiene que hacer cada movimiento a la vez con el motor 4 y 5. La opción 4 es para hacer rotar la muñeca, que se oriente hacia arriba o hacia abajo. La opción 5 es para que la muñeca gire, que la pinza rote sobre sí misma en sentido horario o antihorario. En estos dos casos se hace el algoritmo es semejante al anterior, salvo que la información de cada posición se almacena en dos variables globales, *pos4* y *pos5*, y se utilizan dos variables bandera *m4* y *m5* para conocer si se ha alcanzado la posición deseada. Mientras las variables sean 1 el motor correspondiente se seguirá moviendo, si es 0 se detiene.

Código 5.11: Función *mover.m*

```
% Esta función servirá para posicionar los eslabones del robot
% a partir de los ángulos introducidos en grados.
function mover(num, val,PWM)
    % num : número del eslabón que se quiere mover
    %       además,4 para rotación de la muñeca, 5 para giro
    % val : ángulo en grados
    % PWM : señal PWM del LMD18200 en tanto por uno


    % Llmamos a la variable global a
    global a

    % Encendemos el 2° LED rojo
    digitalWrite(a,51,1);

    % Iniciamos las dos variables globales pos4 y pos5
    % Estas variables contendran las dos posiciones de
    % la muñeca en cada momento
    global pos4;
    global pos5;

    % Array con las reducciones de cada motor
    reduccion = [600 523.068 523.068 120 120];

    % Se calculan los pulsos correspondientes para el caso requerido
    pulsos = round(val/360*reduccion(num)*12);

    % Se selección el motor o motores a mover
    switch num

        % Rotación de la base
        case 1

            % Se comprueba que el ángulo este dentro de los límites
            if val>=-180 && val<=140
                %Se comprueba que el motor no este ya en
                %la posición deseada
                if pulsos ~= encoder(1)
                    % En caso de no estarlo sel motor se mueve en la
                    % dirección necesaria.
```

```
                if pulsos < encoder(1)
                    while pulsos < encoder(1)
                        motor(1,0,PWM);
                    end
                else
                    while pulsos > encoder(1)
                        motor(1,1,PWM);
                    end
                end
            end
        else
            % Si el angulo no esta dentro de los límite se imprime
            % en mensaje por pantalla.
            fprintf('Ángulo no válido\n');
        end
        % Se detiene el motor
        motor(1,0,0);


    %Giro del hombro
    case 2
        % Se necesita cambiar el signo de los pulsos
        pulsos = (-1)*pulsos;
        if val>=-30 && val<=120
            if pulsos ~= encoder(2)
                if pulsos < encoder(2)
                    while pulsos < encoder(2)
                        motor(2,0,PWM);
                    end
                else
                    while pulsos > encoder(2)
                        motor(2,1,PWM);
                    end
                end
            end
        else
            fprintf('Ángulo no válido\n');
        end
        motor(2,0,0);

    % Giro del codo
    case 3

        % Angulo máximo que puede girar
        q2 = encoder(2)*360/reduccion(2)/12;
        max = -0.00432864*q2^2+1.00469484*q2+64.2739366;
        if val>=-15 && val<=max
            if pulsos ~= encoder(3)
                if pulsos < encoder(3)
                    while pulsos < encoder(3)
                        motor(3,0,1);
                    end
                else
                    while pulsos > encoder(3)
                        motor(3,1,PWM);
                    end
                end
            end
```

```
        else
            fprintf('Angulo no válido\n');
        end
        motor(3,0,0);


    %Giro de la muñeca
    case 4
        % Se almacena el valor de los encoders en el momento inicial
        enc4 = (-1)*encoder(4);
        enc5 = encoder(5);

        % Se crean unas variables bandera con valor 1
        m4= 1;
        m5 = 1;

        % Ángulo restringido entre 0 y 180°
        if ang>=0 && ang<=180
        % Si se encuentra en una posición distinta a la requerida
        if pulsos ~= pos4

            % Si el ángulo es menor
            if pulsos < pos4

                %Motores 4 y 5 con dirección 1
                motor(4,1,PWM);
                motor(5,1,PWM);

                % Mientras alguna de las variable bandera sea 1
                while m4 || m5
                   % Se almacena el incremento de los encoders
                   dif4 = (-1)*encoder(4) - enc4;
                   dif5 = encoder(5) - enc5;

                   % Cuando un motor alcanza la posición que se desea
                   %(posicion al inicio + la difencia) se para y se
                   %pone la bandera a 0.
                   if pulsos >= pos4 + dif4
                       motor(4,0,0);
                       m4 = 0;
                   end
                   if pulsos >= pos4 + dif5
                       motor(5,0,0);
                       m5 = 0;
                   end
                end % fin del while

            %en caso contrario repetimos la misma secuencia
            %pero con los motores 4 y 5 con dirección 0.
            else
                motor(4,0,PWM);
                motor(5,0,PWM);

                while m4 || m5
                   dif4 = (-1)*encoder(4) - enc4;
                   dif5 = encoder(5) - enc5;
                   if pulsos <= pos4 + dif4
                       motor(4,0,0);
```

```
                        m4 = 0;
                    end
                    if pulsos <= pos4 + dif5
                        motor(5,0,0);
                        m5 = 0;
                    end

                end % fin del while
            end %fin del else
        end %fin del if
        else
            fprintf('Ángulo no válido\n');
        end

        %Se actualiza la posición
        pos4 = pulsos;

    %Rotación de la muñeca.
    case 5
        %En el case 5 se sigue el mismo procedimiento que en el 4
        %Salvo que la dirección de los motores 4 y 5 será 0 y 1
        %1 y 0 respectivamente que la muñeca gire sin rotación.
        enc4 = encoder(4);
        enc5 = encoder(5);

        m4= 1;
        m5 = 1;

        if pulsos ~= pos5

            if pulsos < pos5
                motor(4,0,PWM);
                motor(5,1,PWM);


                while m4 || m5
                    dif4 = encoder(4) - enc4;

                    dif5 = encoder(5) - enc5;

                    if pulsos >= pos5 + dif4
                        motor(4,0,0);
                        m4 = 0;
                    end
                    if pulsos >= pos5 + dif5
                        motor(5,0,0);
                        m5 = 0;
                    end
                end % fin del while

            else
                motor(4,1,PWM);
                motor(5,0,PWM);

                while m4 || m5
                    dif4 = encoder(4) - enc4;
                    dif5 = encoder(5) - enc5;
                    if pulsos <= pos5 + dif4
```

```
                            motor(4,0,0);
                            m4 = 0;
                        end
                        if pulsos <= pos5 + dif5
                            motor(5,0,0);
                            m5 = 0;
                        end

                    end % fin del while
                end %fin del else
            end %fin del if
            pos5 = pulsos;


    end % fin del switch

    % Se apaga el LED
    digitalWrite(a,51,0);
end
```

5.4. Cinemática[6]

En el capítulo anterior se obtuvieron por un lado los parámetros de Denavit-Hatenberg del robot para las matrices de transformación para el problema cinemático directo, y por otro las fórmulas geométricas para resolver el problema cinemático inverso.

En esta apartado se resolverán dichos problemas con MATLAB y se usarán los resultados para la posicionar el robot.

5.4.1. denavit.m

Esta función devuelve la matriz de transformación ${}^{i-1}\mathbf{A}_i$ a partir de los parametros θ, d, a y α que se le introducen según la formulación de (4.5).

Código 5.12: Función *denavit.m*

```
function dh = denavit( teta, d, a, alfa )

    dh = [cos(teta), -cos(alfa)*sin(teta), sin(alfa)*sin(teta),  a*cos(teta);
          sin(teta), cos(alfa)*cos(teta),  -sin(alfa)*cos(teta), a*sin(teta);
          0,         sin(alfa),            cos(alfa),            d;
          0,         0,                    0,                    1];

end
```

5.4.2. cinematicaDirecta.m

Como su nombre indica esta función sirve para calcular la matriz $\mathbf{T}={}^{0}\mathbf{A}_5$ como solución del problema cinemático directo para conocer la orientación y la posición del extremo de la herramienta. Como entrada se le pasa un array de cinco elementos con los ángulos en grados de cada articulación.

Además se calcula la posición de cada articulación y se dibuja el robot en una gráfica.

Código 5.13: Función *cinematicaDirecta.m*

```
function T = cinematicaDirecta(q)

    % Pasar q de rad a grados
    q = q * pi/180;

    % Parámetros Denavit-Hartenberg del robot
    teta = [q(1) q(2)-pi/2 -q(2)+q(3)+pi/2 -q(4)-q(3) q(5)];
    d = [35 0 0 0 14];
    alfa = [-pi/2 0 0 -pi/2 0];
    a = [2.5 22.5 22.5 0 0];

    %Matrices de transformación homogénea
    A01 = denavit(teta(1),d(1),a(1),alfa(1));
    A12 = denavit(teta(2),d(2),a(2),alfa(2));
    A23 = denavit(teta(3),d(3),a(3),alfa(3));
    A34 = denavit(teta(4),d(4),a(4),alfa(4));
    A45 = denavit(teta(5),d(5),a(5),alfa(5));

    %Matriz de transformación del primer al ultimo
    %sistema de coordenadas
    A02 = A01 * A12;
    A03 = A02 * A23;
    A04 = A03 * A34;
    A05 = A04 * A45;

    T = A05;

    % Vector de posicion (x,y,z) de cada sistema de coordenadas
    x0 = 0;         y0 = 0;         z0 = 0;
    x1 = A01(1,4); y1 = A01(2,4); z1 = A01(3,4);
    x2 = A02(1,4); y2 = A02(2,4); z2 = A02(3,4);
    x3 = A03(1,4); y3 = A03(2,4); z3 = A03(3,4);
    x4 = A04(1,4); y4 = A04(2,4); z4 = A04(3,4);
    x5 = A05(1,4); y5 = A05(2,4); z5 = A05(3,4);

    % Se dibuja el robot
    x = [x0 x1 x2 x3 x4 x5];
    y = [y0 y1 y2 y3 y4 y5];
    z = [z0 z1 z2 z3 z4 z5];
    plot3(x,y,z);
    grid;
    axis([-80 80 -80 80 0 80]);
    xlabel('x'); ylabel('y'); zlabel('z');

end
```

5.4.3.mover_a_angulo

Función para mover el robot a la posición especificada por los ángulos que se le introducen en grados. Como entrada se le introduce un array de cinco elementos con los ángulos en grados y la velocidad a de movimiento en tanto por uno. Después de mover las articulaciones se calcula la cinemática directa y se devuelve la matriz **T** como salida.

Código 5.14: Función *mover_a_posicion.m*

```
function T = mover_a_angulo(q,velocidad)
```

```
    % Se mueve cada articulación al ángulo especificado
    for i=1:5
        mover(i,q(i),velocidad);
    end

    % Se calcula la cinematica directa
    T = cinematicaDirecta.m;

end
```

5.4.4.cinematicaInversa.m

Con la función cinemática inversa se resuelve el problema cinemático inverso según las ecuaciones expuestas en el apartado 4.3.

Como entrada se introduce un array de cinco elementos: los tres primeros son las coordenadas cartesianas del extremo de la herramienta desde la base del robot, las dos últimas son los ángulos de la muñeca en grados.

La salida es un array de cinco elementos con los ángulos de cada articulación en grados.

Código 5.15: Función *cinematicaInversa.m*

```
function q=cinematicaInversa(p)

    x = p(1);
    y = p(2);
    z = p(3);

    % Se pasan los angulos de grados a rad
    q4 = p(4)*pi/180;

    % Parametros DH
    d = [35 0 0 0 14];
    alfa = [-pi/2 0 0 -pi/2 0];
    a = [2.5 22.5 22.5 0 0];

    % Cálculo de R, z13 y r
    R = sqrt(x^2+y^2)-d(5)*sin(q4)-a(1);
    z13 = z + d(5)*cos(q4)-d(1);
    r = sqrt(R^2+z13^2);

    % Cálculo de q1
    q(1)=atan2(y, x);

    % Cálculo de q2
    cbeta = (r^2+a(2)^2-a(3)^2)/(2*r*a(2));
    sbeta = sqrt(1-cbeta^2);
    calfa = (z13)/r;
    salfa = R/r;
    beta = atan2(sbeta,cbeta);
    alfa = atan2(salfa,calfa);
    q(2) = alfa - beta;

    % Cálculo de q3
```

```
    cq3 = (R-a(2)*sin(q(2)))/a(3);
    sq3 = (a(3)*cos(q(2))-z13)/a(3);
    q(3) = atan2(sq3,cq3);

    % Se pasan los ángulos de radines a grados
    q = q*180/pi;

    % q4 y q5 coinciden con p(4) y p(5)
    q(4) = p(4);
    q(5) = p(5);
end
```

5.4.5.mover_a_coordenadas.m

Con esta función se mueve el robot a la posición necesaria para que la herramienta se sitúe en las coordenadas y la orientación deseada.

Como entrada se introducen los mismo parámetros que en la cinemática inversa además de la velocidad y se obtiene como salida los ángulos de cada articulación.

Primero se resuelve el problema cinemático inverso y a continuación se mueve cada articulación a la posición calculada.

Código 5.16: Función *mover_a_coordendas.m*

```
function q=mover_a_coordenadas(p,velocidad)
    % Se resuelve el problema cinematico inverso
    q = cinematicaInversa(p);

    % Se mueve el robot a la posición deseada con los ángulos obtenidos
    for i=1:5
        mover(i,q(i),velocidad);
    end
end
```

5.5. Interfaz gráfica

Las funciones anteriores sirven para controlar el SCORBOT-ER III, sin embarga para ello hay que introducirlas una a una en la ventana de comandos de MATLAB.

Las GUI (también conocidas como interfaces gráficas de usuario o interfaces de usuario) permiten un control sencillo (con uso de ratón) de las aplicaciones de software, lo cual elimina la necesidad de aprender un lenguaje y escribir comandos a fin de ejecutar una aplicación.

Las apps de MATLAB son programas autónomos de MATLAB con un frontal gráfico de usuario GUI que automatizan una tarea o un cálculo. Por lo general, la GUI incluye controles tales como menús, barras de herramientas, botones y controles deslizantes.. También es posible crear apps personalizadas propias, incluidas las interfaces de usuario correspondientes, para que otras personas las utilicen.

GUIDE (entorno de desarrollo de GUI) proporciona herramientas para diseñar interfaces de usuario para Apps personalizadas. Mediante el editor de diseño de GUIDE, es posible diseñar gráficamente la interfaz de usuario. GUIDE genera entonces de manera automática el código de MATLAB para construir la interfaz, el cual se puede modificar para programar el comportamiento de la app.

A fin de ejercer un mayor control sobre el diseño y el desarrollo, también se puede crear código de MATLAB que defina las propiedades y los comportamientos de todos los componentes. MATLAB contiene funcionalidad integrada que le ayudará a crear la GUI para su app de forma programática. Cabe la posibilidad de agregar cuadros de diálogo, controles de interfaz de usuario (como botones y controles deslizantes) y contenedores (como paneles y grupos de botones). [8]

Con la interfaz gráfica se crean tres archivos:

- *SCORBOT_cinematica.fig*: es el archivo que contiene el display de la GUI. Abriendolo a través de guide se pueden cambiar sus elementos.
- *SCORBOT_cinematica.m:*este archivo contiene toda la programación de los distintos elementos de la interfaz.
- *SCORBOT_cinematica.asv*: es un *backup* del archivo .m para restaurarlo en caso de que este dañado o sea eliminado.

La interfaz tiene distintas partes:

- **Menú Arduino:** en este menú tenemos una opción para conectar Arduino y otra para desconectarlo.
  - **Conectar Arduino:** se establece la conexión con Arduino. Una vez conectado un mensaje avisa de cuando es necesario conectar la fuente de tensión. Por último se coloca el robot en su posición inicial y se actualiza la interfaz.
  - **Desconectar Arduino:** después de posicionar el brazo robot en la posición de reposo un mensaje avisa de que es necesario desconectar la fuente de tensión. Una vez desconectada y cerrado el mensaje el Arduino se desconecta.
- **Panel de cinemática directa:** Aquí se introducen los ángulos de cada articulación en grados. Al pulsar el botón *Mover* el brazo mecánico se mueve a la posición introducida con una trayectoria eje a eje, se resuelve la cinemática inversa y se actualiza la interfaz.
- **Panel de cinemática inversa:** Se introducen los datos de la cinemática inversa: las coordenadas del extremo de la herramienta en centímetros y los ángulo de giro de la muñeca en grados. Al pulsar *Mover* se resuelve la cinemática inversa y el robot se mueve a la posición calculada con un movimiento eje a eje. También se resuelve la cinemática directa y se actualiza la interfaz.
- **Matriz T:** panel donde se muestran la información más relevante de la matriz **T** calculada con la cinemática inversa. Los paneles *x5,y5,z5* muestra los vectores ortonormales que forman el sistema de coordenadas de $<S_5>$ en el extremo de la herramienta y el panel *O5* muestra origen de dicho sistema, o lo que es lo mismo, la posición del extremo de la pinza. Todo referido al sistema de la base del robot.
- **Panel Pinza:** tenemos dos opciones, abrir y cerrar. Cada una abre o cierra la pinza.
- **Boton Hard** Home: situa el robot en la posición inicial.
- **Velocidad:** se introduce la velocidad de giro del robot en tanto por uno. Por defecto es 1.
- **Gráfica:** en esta gráfica se muestra una representación del robot.

A continuación se comentan los aspectos más importantes y el código utilizado. Al final del capítulo se pueden encontrar algunas capturas de la GUI.

5.5.1. Función actualizarMatriz(T)

Con esta función, a la que se le introduce una matriz T como entrada, se actualizan los valores de la matriz T que se muestra por la GUI. La entrada *handles* permite acceder a los elementos de la interfaz.

Código 5.17: Función actualizarMatriz(T)

```
function actualizarMatriz(T,handles)
    set(handles.text18,'String', T(1,1));
    set(handles.text19,'String', T(2,1));
    set(handles.text20,'String', T(3,1));

    set(handles.text21,'String', T(1,2));
    set(handles.text22,'String', T(2,2));
    set(handles.text23,'String', T(3,2));

    set(handles.text24,'String', T(1,3));
    set(handles.text25,'String', T(2,3));
    set(handles.text26,'String', T(3,3));

    set(handles.text29,'String', T(1,4));
    set(handles.text28,'String', T(2,4));
    set(handles.text27,'String', T(3,4));
```

5.5.2. Botón Mover (cinemática directa)

Al pulsar este botón en primer se guardan los valores introducidos en el panel de cinemática directa en un vector *q*, pasándolos de texto a número. Con este vector se hace uso de la función *mover_a_angulo(q)* ya descrita para posicionar el robot. A continuación, se actualizan los valores de la matriz T y del panel de la cinemática inversa.

Código 5.18: Función *pushbutton_cd_Callback*

```
function pushbutton_cd_Callback(hObject, eventdata, handles)
    %Se crea un vector nulo q de 5 elementos
    q = [0 0 0 0 0];

    % Se guardan los valores introducidos en el vector q
    q(1) = str2double(get(handles.edit_q1, 'String'));
    q(2) = str2double(get(handles.edit_q2, 'String'));
    q(3) = str2double(get(handles.edit_q3, 'String'));
    q(4) = str2double(get(handles.edit_q4, 'String'));
    q(5) = str2double(get(handles.edit_q5, 'String'));

    % Se lee la velocidad introducida en tanto por uno
    velocidad = str2double(get(handles.edit_velocidad, 'String'));

    % Se resuelve el problema cinematico directo del robot
    T = mover_a_angulo(q,velocidad);

    %Se actualiza la matriz
```

```
    actualizarMatriz(T,handles);

    % Se actualizan los valores de la cinematica inversa
    set(handles.edit_p1,'String', T(1,4));
    set(handles.edit_p2,'String', T(2,4));
    set(handles.edit_p3,'String', T(3,4));
    set(handles.edit_p4,'String', q(4));
    set(handles.edit_p5,'String', q(5));
```

5.5.3. Función mover (cinemática inversa)

Semejante a la anterior: se almacenan los valores introducidos en el panel de cinemática inversa y se resuelve el problema cinemático inverso posicionando el robot en la disposición calculada. También se actualizan los valores de la cinemática directa, se resuelve el problema cinemático directo y se actualizan los valores de la matriz T.

Código 5.19: Función *pushbutton_ci_Callback*

```
function pushbutton_ci_Callback(hObject, eventdata, handles)
    % Se crea un vector nulo de 5 elementos
    p=[0 0 0 0 0];

    % Se guardan los datos introducidos en el vector
    p(1) = str2double(get(handles.edit_p1, 'String'));
    p(2) = str2double(get(handles.edit_p2, 'String'));
    p(3) = str2double(get(handles.edit_p3, 'String'));
    p(4) = str2double(get(handles.edit_p4, 'String'));
    p(5) = str2double(get(handles.edit_p5, 'String'));

    % Se lee la velocidad introducida en tanto por uno
    velocidad = str2double(get(handles.edit_velocidad, 'String'));
    % Se resuelve el problema cinemático inverso y se mueve
    % el robot
    q = mover_a_coordenadas(p, velocidad);


    % Se resuelve el problema cinematico directo con los resultados
    % obtenidos y se muestra la matriz
    T = cinematicaDirecta(q);

    %Se actualiza la matriz
    actualizarMatriz(T,handles);

        % Se actualizan los datos de la cinemática directa
    set(handles.edit_q1,'String', q(1));
    set(handles.edit_q2,'String', q(2));
    set(handles.edit_q3,'String', q(3));
    set(handles.edit_q4,'String', q(4));
    set(handles.edit_q5,'String', q(5));
```

5.5.4. Botón Hard Home

Primero se coloca el robot en la posición con ángulos igual a 0. Después se usa la función *posInicial* para colocarlo en la posición de *hard home*.

Código 5.20: Función *pushbutton_hh_Callback*

```
function pushbutton_hh_Callback(hObject, eventdata, handles)

    % Se coloca el robot en la posición 0
    q = [0 0 0 0 0];
    T = mover_a_angulo(q,1);

    %Se actualiza la matriz
    actualizarMatriz(T,handles);

    % Se actualizan los valores de la cinematica inversa
    set(handles.edit_p1,'String', T(1,4));
    set(handles.edit_p2,'String', T(2,4));
    set(handles.edit_p3,'String', T(3,4));
    set(handles.edit_p4,'String', q(4));
    set(handles.edit_p5,'String', q(5));

    % Se coloca el robot  en la posición inicial.
    posInicial;

    % Se cambia el radio button de la pinza a abierto
    set(handles.radiobutton_abierta,'value',1);
    set(handles.radiobutton_cerrada,'value',0);
```

5.5.5. Opción del menú Conectar Arduino

Con está opción conectamos el Arduino Mega a MATLAB y una vez hecho posiciona el robot en su posición inicial. Además hace saltar una ventana emergente avisando de que necesitamos conectar la fuente de tensión.

Código 5.21: Función *conArduino_Callback*

```
function conArduino_Callback(hObject, eventdata, handles)

    % Se conecta el Arduino
    iniciarArduino;

    % Se lanza un mensaje avisando de que ya se puede conectar
    % la fuente de tensión.
    msgbox('Conecta la fuente de tensión a 12V','Atención','warn');
    uiwait; % La función se detiene hasta que se le da a aceptar

    % Se coloca en la posición inicial y se actualiza la interfaz con
    % los ángulos a 0.
    posInicial; posInicial;

    q = [0 0 0 0 0];
    T = cinematicaDirecta(q);

    %Se actualiza la matriz
    actualizarMatriz(T,handles);

    % Se cambia el radio button de la pinza a abierto
    set(handles.radiobutton_abierta,'value',1);
    set(handles.radiobutton_cerrada,'value',0);
```

```
    % Se actualizan los valores de la cinematica inversa
    set(handles.edit_p1,'String', T(1,4));
    set(handles.edit_p2,'String', T(2,4));
    set(handles.edit_p3,'String', T(3,4));
    set(handles.edit_p4,'String', q(4));
    set(handles.edit_p5,'String', q(5));
```

5.5.6. Opción del menú Desconectar Arduino

Esta función en primer lugar coloca el robot en su posición inicial y actualiza la interfaz. A continuación lanza una mensaje avisando de que es necesario desconectar la fuente de tensión. Por último desconecta Arduino Mega de MATLAB.

Código 5.22: Función *desconArduino_Callback*

```
function desconArduino_Callback(hObject, eventdata, handles)

    % Se coloca el robot en la posición de reposo y se actualiza
    % la interfaz
    pinza('c');
    q = [0 -30 30 0 0];
    T = mover_a_angulo(q,1);


    set(handles.radiobutton_abierta,'value',0);
    set(handles.radiobutton_cerrada,'value',1);

    actualizarMatriz(T,handles);

    set(handles.edit_p1,'String', T(1,4));
    set(handles.edit_p2,'String', T(2,4));
    set(handles.edit_p3,'String', T(3,4));
    set(handles.edit_p4,'String', q(4));
    set(handles.edit_p5,'String', q(5));


    % Se lanza un mensaje avisando de que se debe conectar
    % la fuente de tensión.
    msgbox('Desconecta la fuente de tensión','Atención','warn');
    uiwait; % La función se detiene hasta que se le da a aceptar

    % Se desconecta Arduino
    desconectarArduino;
```

5.5.7. Abrir y cerrar la pinza

Código 5.23: Función *Pinza_SelectionChangeFcn*

```
function Pinza_SelectionChangeFcn(hObject, eventdata, handles)
    % Se lee la opción seleccionada
    A = get(hObject,'String');
    % En función esta se abre o cierra la pinza
    pinza(A);
```

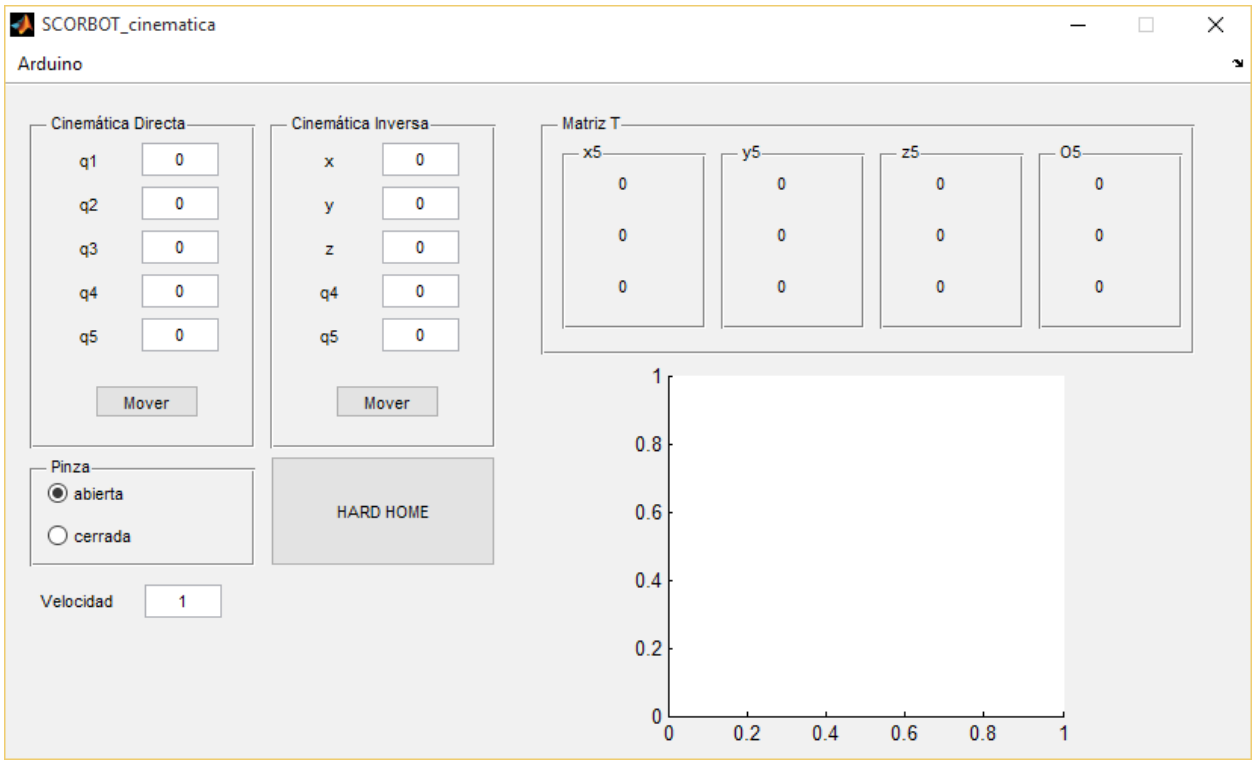

Figura 5.2: GUI al iniciar la aplicación.

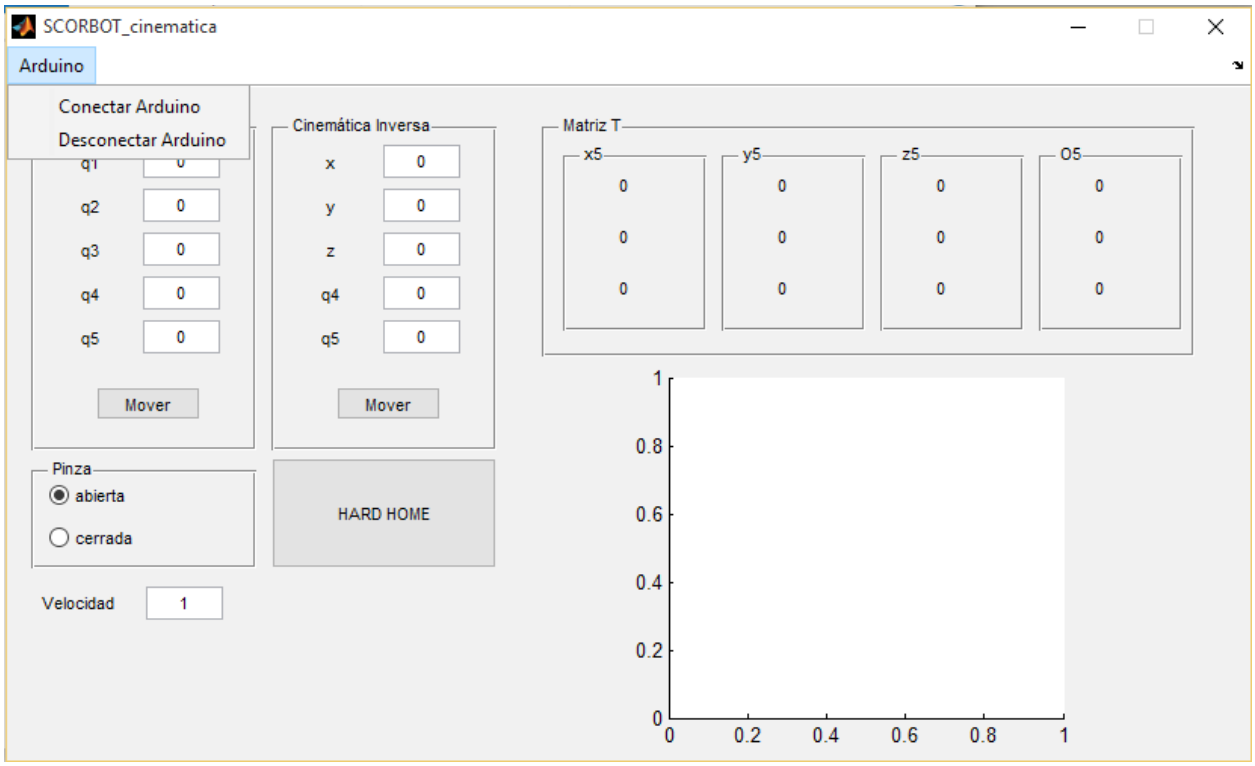

Figura 5.3: Menú Arduino

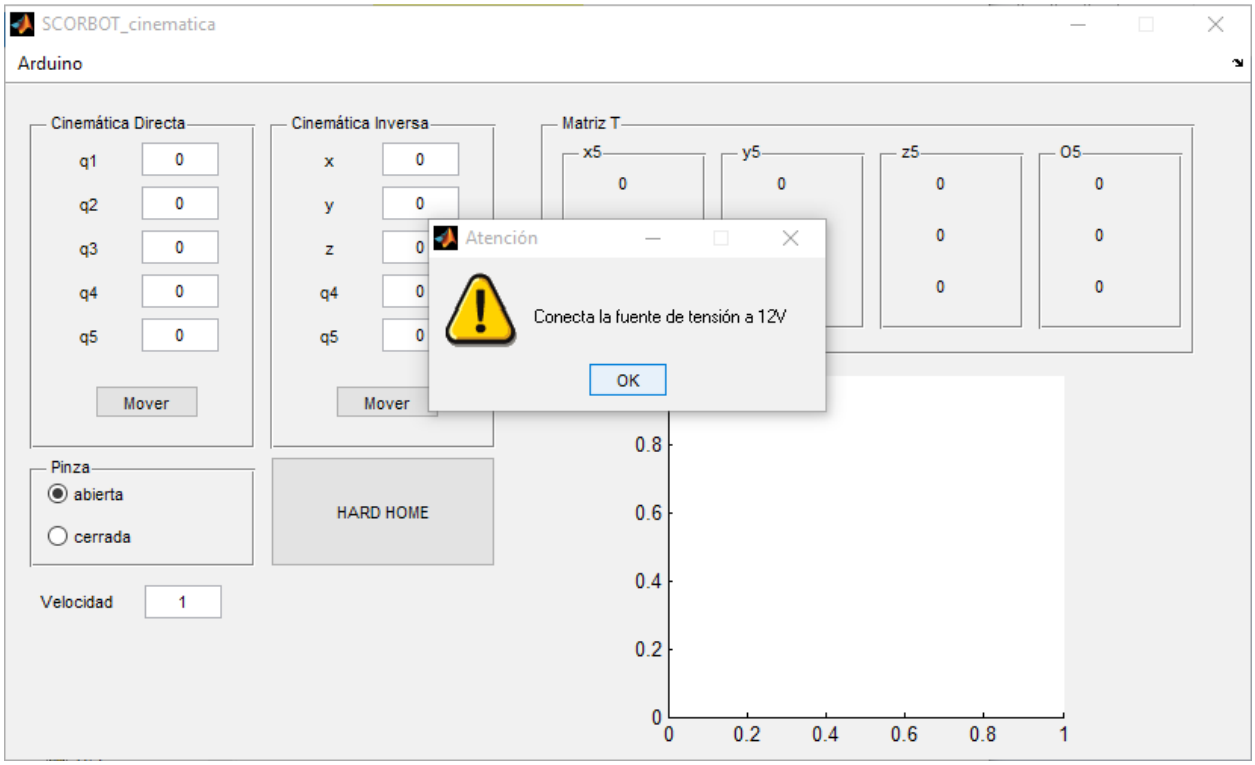

Figura 5.4: Ventana emergente al conectar Arduino.

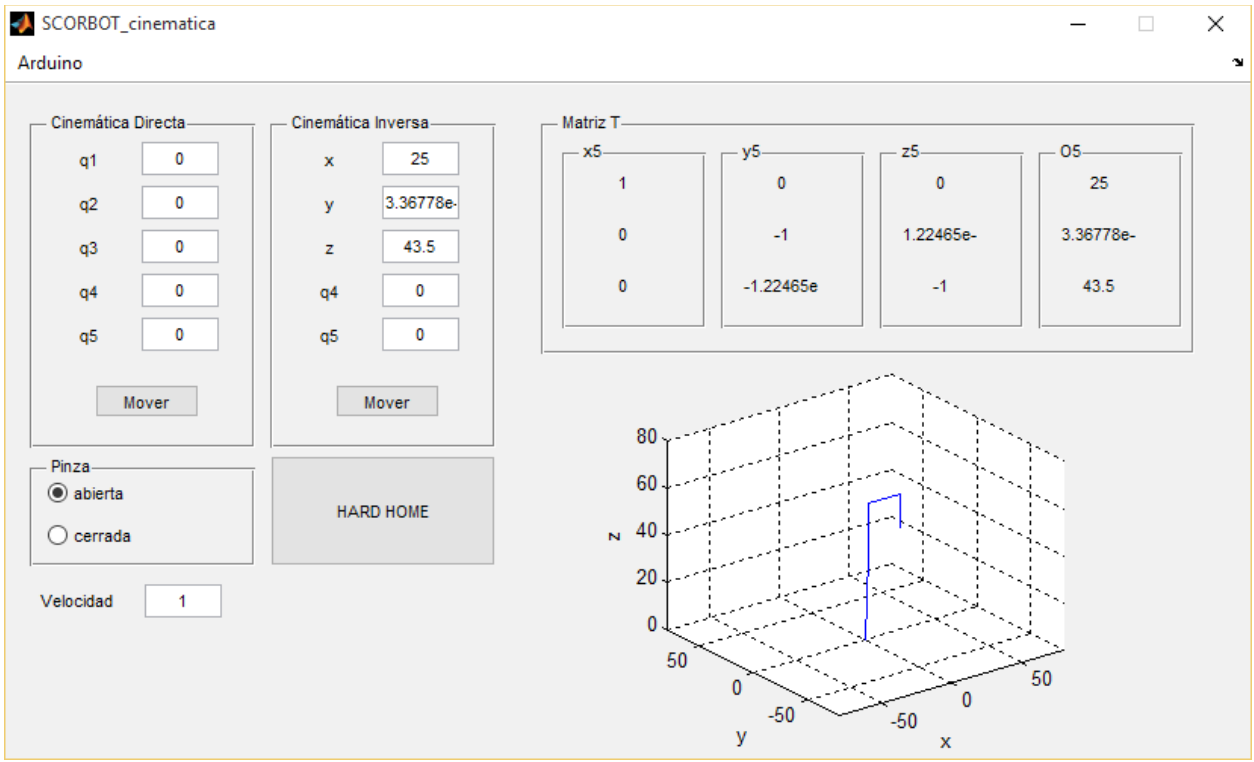

Figura 5.5: GUI una vez conectado Arduino.

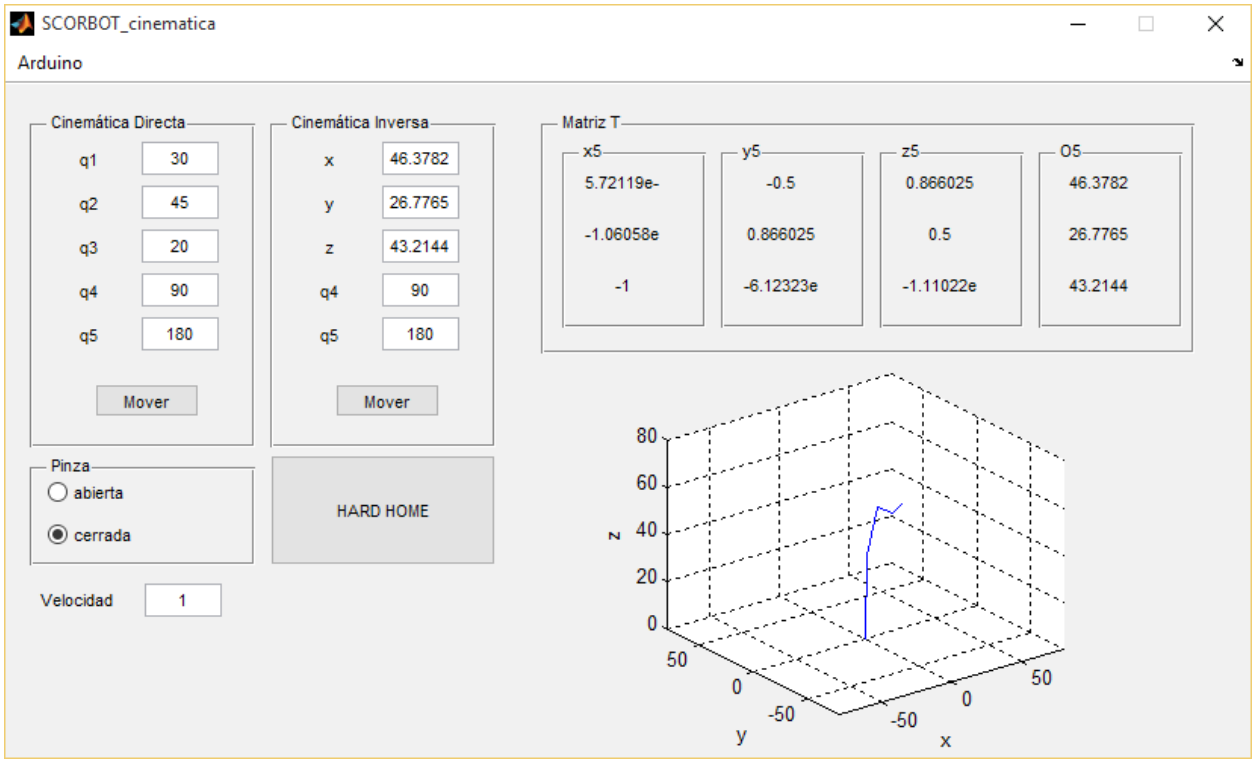

Figura 5.6: GUI al introducir una posición cualquiera.

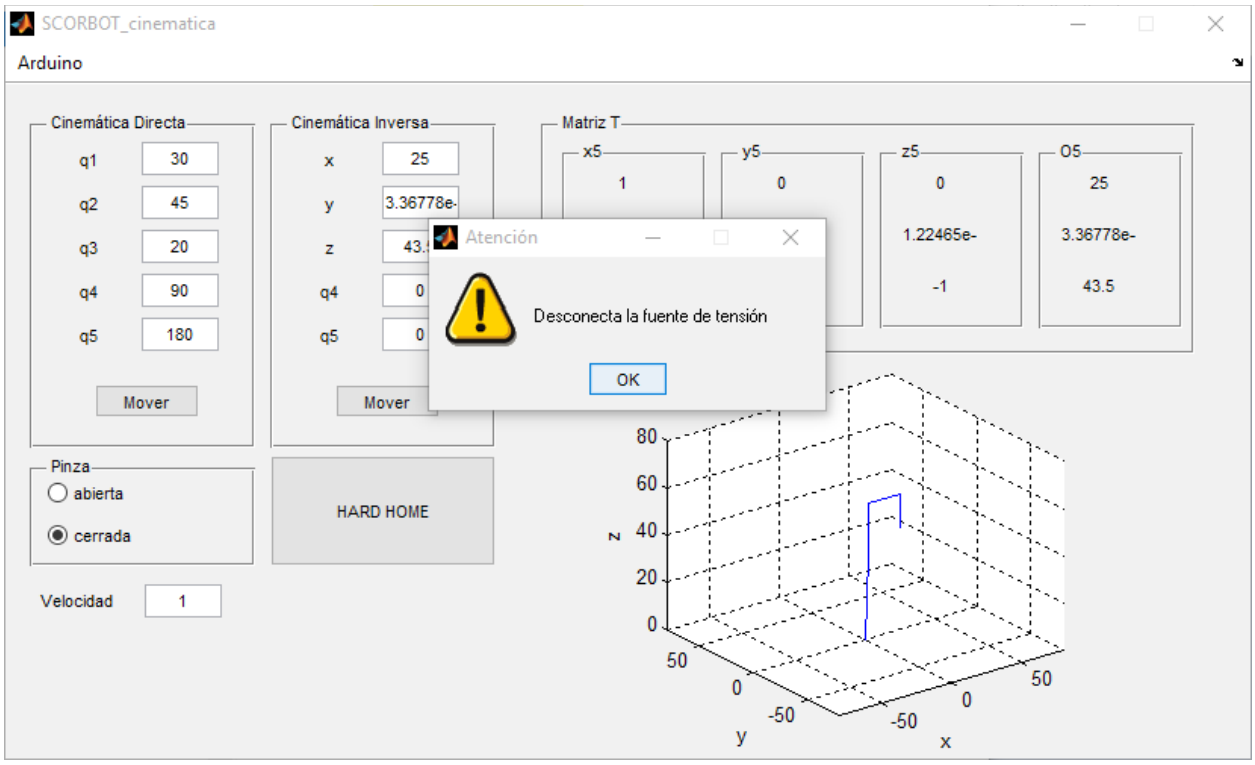

Figura 5.7: Ventana emergente al desconectar Arduino.

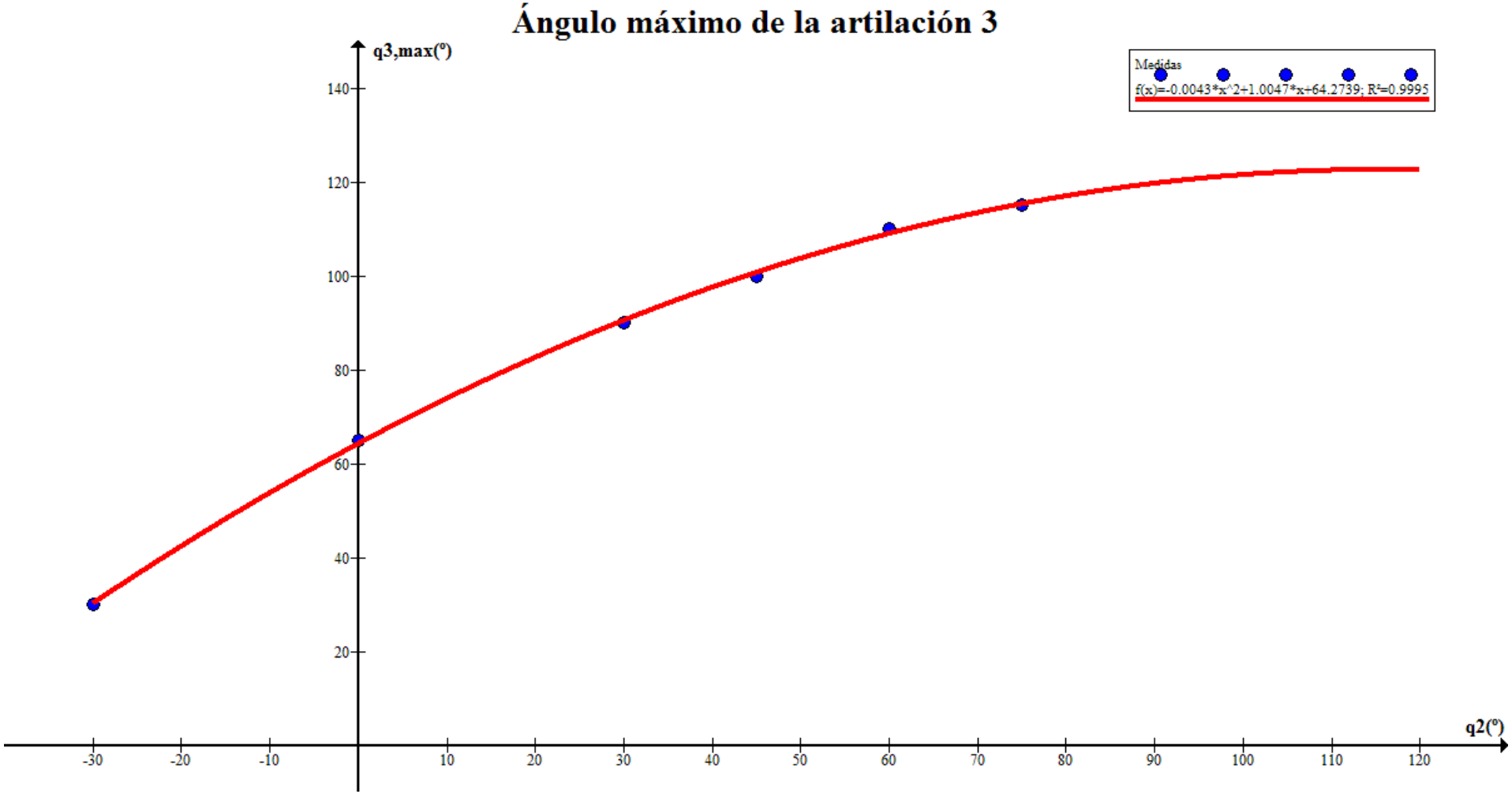

Figura 5.8: Gráfico de interpolación del ángulo q_3 máximo.

6.Gráficas

6.1. Introducción

A continuación se expondrán una serie de gráficas en la que ésta representado el movimiento de cada motor. Para tomarlas se ha hecho girar cada motor un ángulo de 90º en cada dirección (salvo el motor 3, que por motivos mecánicos solo se ha hecho girar 45º) y con los encoders se ha medido cada 0,1 segundos el ángulo en el que se encontraba.

Además, con los datos obtenidos se ha calculado la velocidad de cada motor, tanto en una dirección como en la contraria.

Por último se ha obtenido también el espacio de trabajo de nuestro robot.

6.2. Trayectorias articulares

En las gráficas de las Figuras 6.1 a la 6.6 se han representado la evolución de cada articulación en el tiempo. Como puede apreciarse cada articulación gira de manera lineal y con velocidad constante, siendo esta la máxima posible (PWM del 100%). Las velocidades que alcanza cada articulación se muestran en la Tabla 6.1.

Sin embargo cada articulación no gira a la misma velocidad en una dirección que en otra. Esto se debe a dos motivos:

- En primer lugar por los sistemas de transmisión. Esto es más apreciable en el giro del motor 1. Se ha echado aceite a los los engranajes y a las correas para lubricarlos, pero se trata de un robot viejo y que está mucho tiempo sin utilizar, por lo que es posible que las transmisiones no funciones al 100% ni igual en ambas direcciones.
- En segundo lugar y más importante, subir una articulación que bajarla. Como se aprecia en el resto de articulaciones la velocidad a la hora de hacer bajar una articulación es más alta que al hacerla subir debido al peso. Esto afecta sobre todo al motor 2, que tiene que aguantar mayor peso.

Tabla 6.1: Velocidades de los motores.

Motor #	Dirección	Velocidad (rad/s)
1	0 1	0,4538 0,4284
2	0 1	0,4976 0,3988
3	0 1	0,4110 0,5650
4	0 1	0,7438 1,0104
5	0 1	0,8582 0,9832
Grp	Abrir Cerrar	1 segundo 1 segundo

Además, los motores 4 y 5 que deben girar a la vez lo hacen a velocidades distintas. Aunque la diferencia no es muy grande esto da problemas de precisión y repetibilidad a la larga.

La pinza tarda 1 segundo tanto en abrirse como en abrirse.

6.3. Espacio de trabajo

Con la ayuda de la cinemática directa de nuestro robot se han representado los puntos en los que el extremo de este puede posicionarse. Se han tenido en cuenta las limitaciones expuestas en la Tabla 5.1 y se ha calculado la posición de la pinza variando los ángulos entre los intervalos (de uno en uno para q_2 y q_3 de cinco en cinco para q_4).

El resultado es la Figura 6.7. Se trata del espacio de trabajo para el plano y=0 (q_1=0), pero se repite para todo el intervalo de giro de la base del robot.

Naturalmente los puntos situado por debajo de z=0 no son accesibles en principio a no ser que el robot se encuentre en un voladizo.

A continuación se exponen los puntos máximos y mínimos en cada coordenada x y z:

Coordenada	x máximo	x mínimo	z máximo	z mínimo
x	61,5 cm	10,0624 cm	24,2333 cm	21,9856
z	35 cm	-9,3311 cm	77,3234 cm	-12,75
q1	90º	120º	0	120º
q2	0º	122º	-15º	90º
q3	90º	0º	-15	0
Color en la gráfica	Rojo	Magenta	Azul	Cian

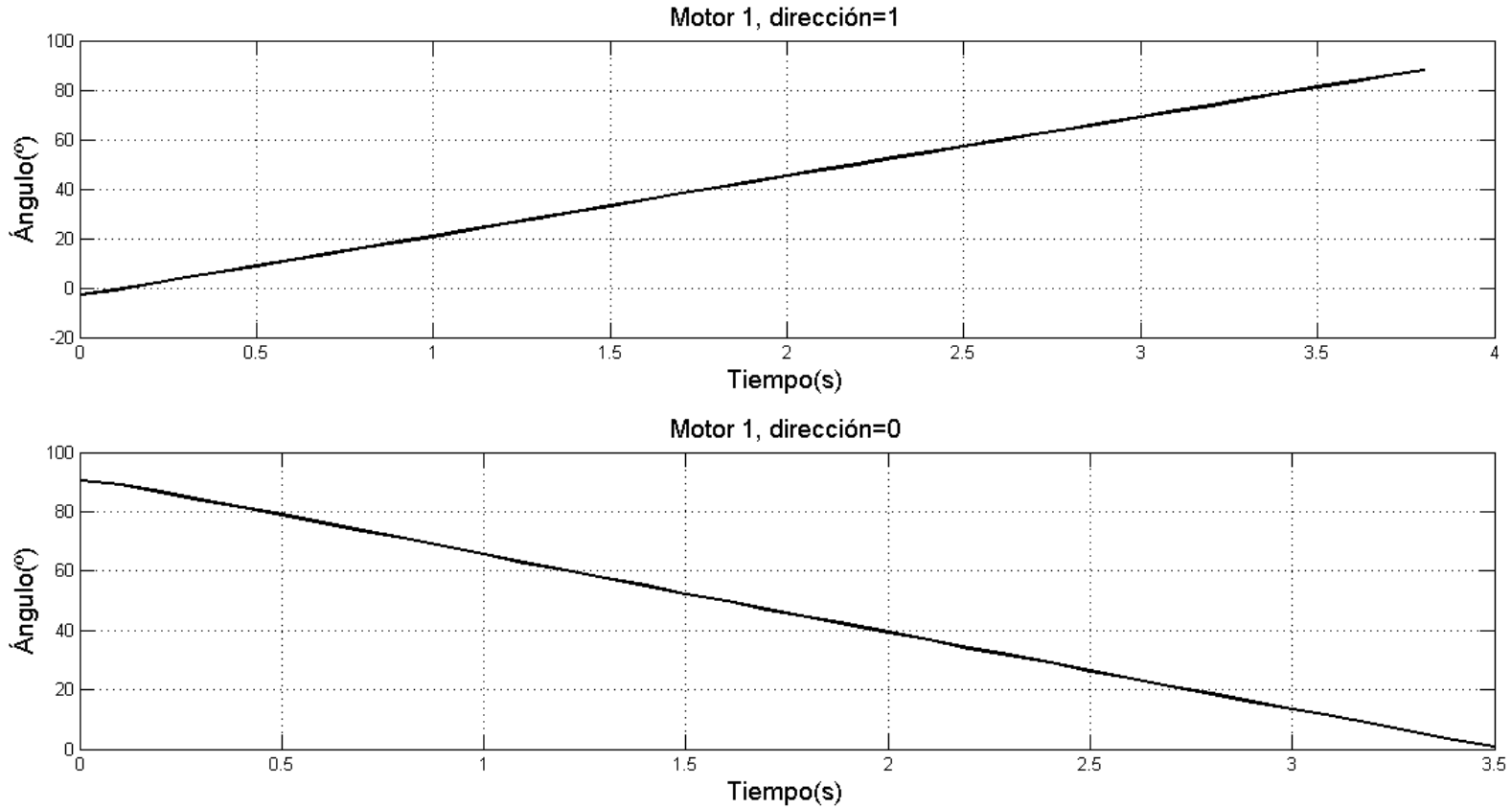

Figura 6.1: Gráfica de la trayectoria articular del motor 1 de 0 a 90º.

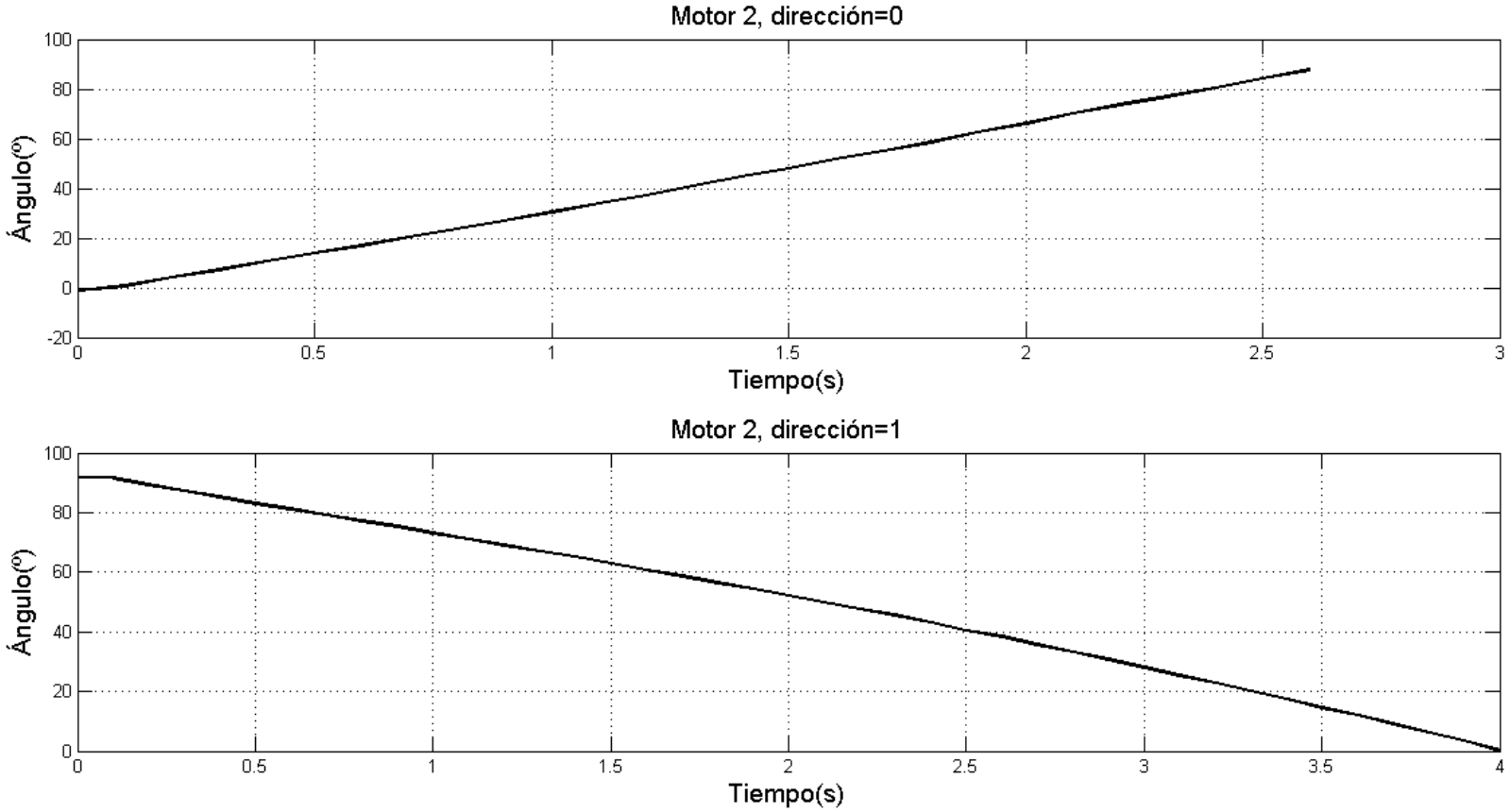

Figura 6.2: Gráfica de la trayectoria articular del motor 2 de 0 a 90º.

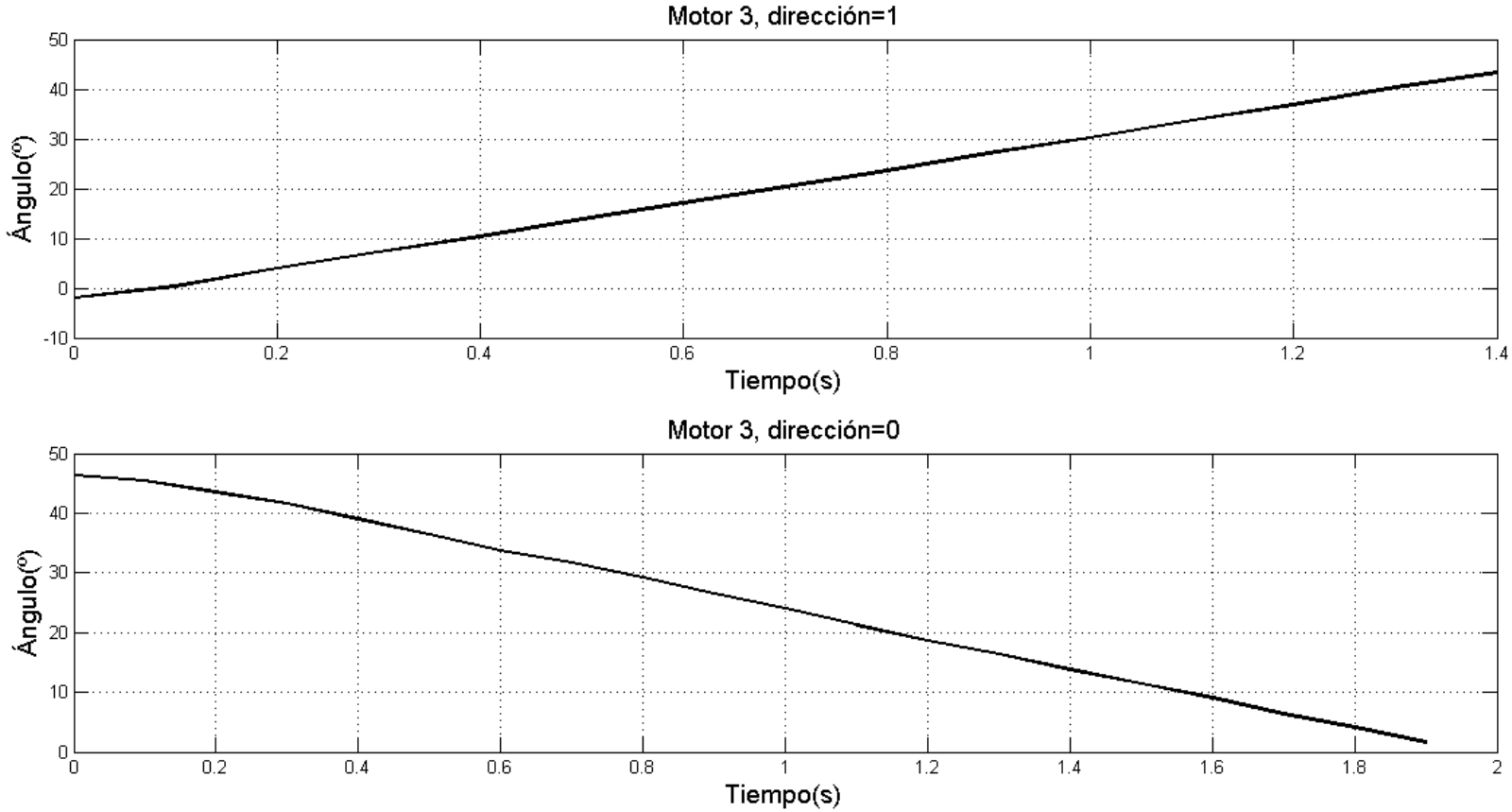

Figura 6.3: Gráfica de la trayectoria articular del motor 2 de 0 a 45º.

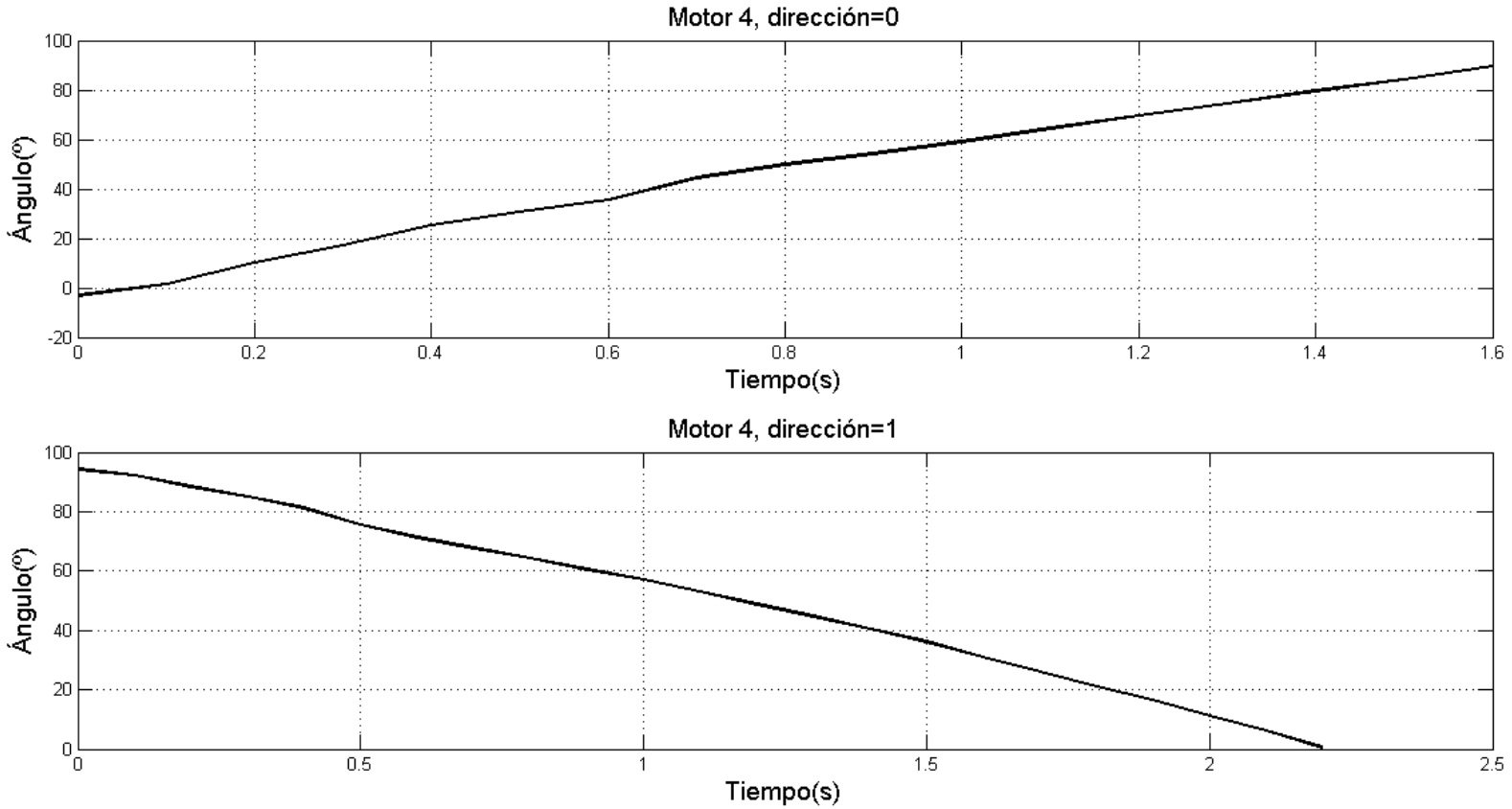

Figura 6.4: Gráfica de la trayectoria articular del motor 4 de 0 a 90º.

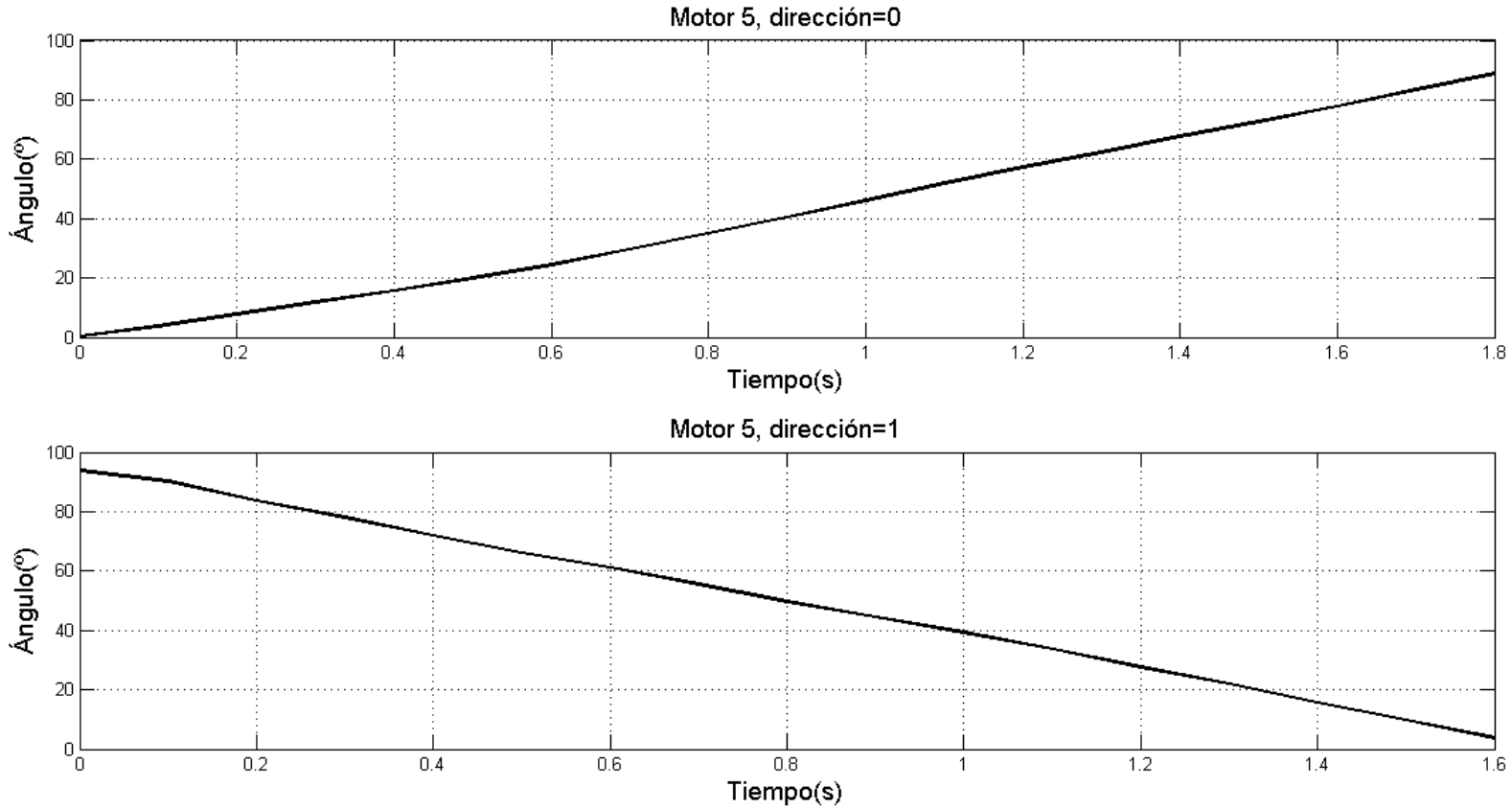

Figura 6.5: Gráfica de la trayectoria articular del motor 5 de 0 a 90º.

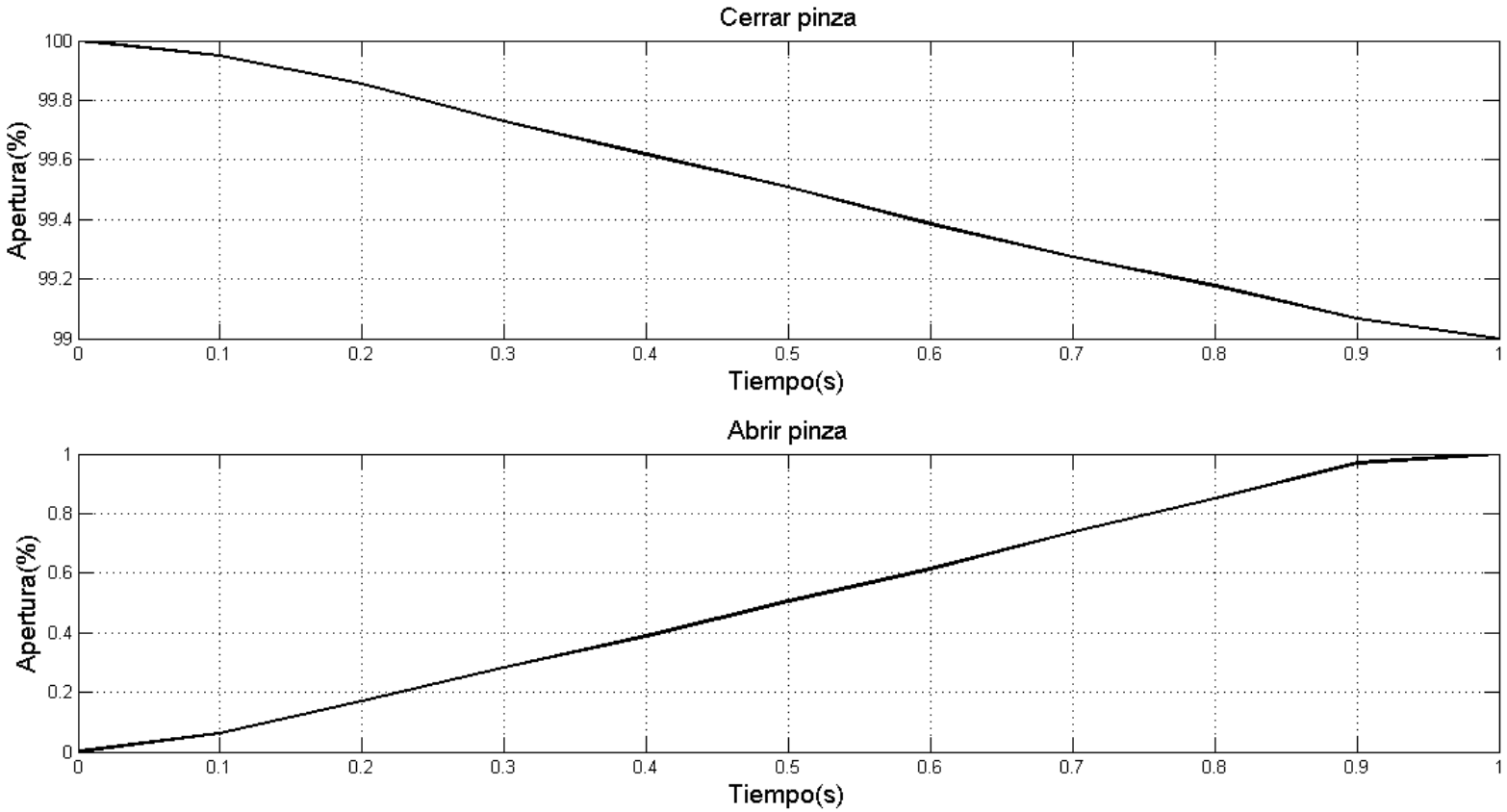

Figura 6.6: Cierre y apertura de la pinza.

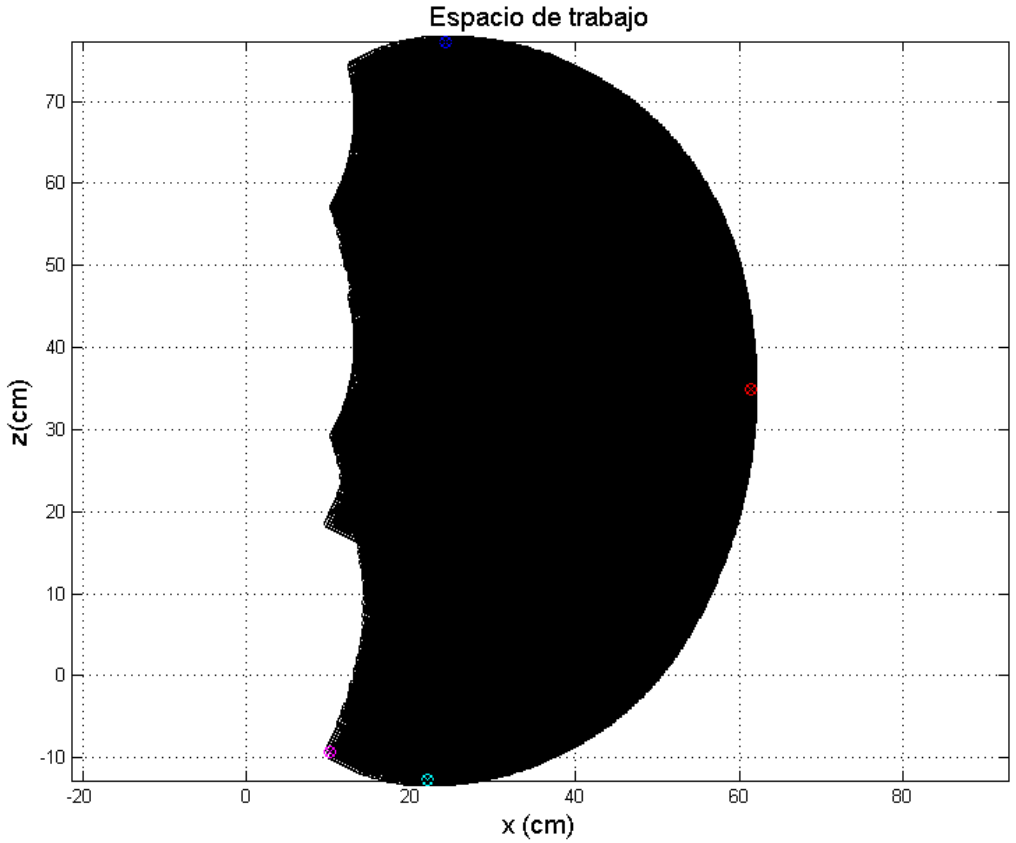

Figura 6.7: Espacio de trabajo de SCORBOT-ER III

7.Conclusión y mejoras

7.1. Conclusiones

Se ha cumplido el objetivo de devolver este antiguo SCORBOT-ER III de nuevo a la vida. Un breve resumen de los distintos pasos seguidos es el siguiente:

En primer lugar se ha estudiado la mecánica y los componentes del robot con su hoja de especificaciones y se ha comprobado que todo funcionara correctamente, sustituyendo aquellas partes que no servían o que faltaban o bien buscando una solución alternativa.

A continuación se han analizado todos los circuitos eléctricos que se usarán para controlar el robot, alimentarlo de energía y recibir las entradas correctamente con Arduino. Hecho esto, con el software *Eagle* se han unido todos estos circuitos en uno solo para todo el robot y se ha diseñado el circuito impreso o PCB, para a continuación hacerla físicamente según los pasos indicados.

Por último, una vez comprobado que la PCB funciona correctamente y reparándola en algunos puntos, se ha pasado a la programación. Para ella se ha usado el harware libre Arduino Mega 2560 y el software MATLAB, este último por ofrecer una gran potencia de cálculo y dar la opción de poder controlar Arduino (y por lo tanto el robot) en tiempo real. Primero se modifico el código *adioes.m* de Arduino para leer correctamente los encoders y después se paso a programar las distintas funciones de MATLAB, empezando por las más básicas para leer los encoders y activar los motores y terminando por la creación de una interfaz gráfica que ofrece facilidad de uso.

Además, se ha resuelto el problema de cinemática directo del robot usando matrices de transformación y el método de Denavit-Hatenberg y el problema cinemático inverso basándose en la geometría del robot. También se han creado gráficas para representar el movimiento articular de cada motor y el espacio de trabajo del robot.

Tras todos estos pasos hemos hecho el SCORBOT-ER III completamente operativo que puede servir tanto para prácticas sobre robótica como para base para futuros proyectos, con la ventaja de que se está usando software y hardware completamente modificable según convenga.

7.2. Posibles mejoras

A pesar de haber conseguido los objetivos propuestos con este trabajo existen hay modos de mejorarlo y hacerlo más eficiente. Algunas de ellas son las siguientes:

- Mejorar la precisión y la repetibilidad. Como se ha comentado, las reducciones de los motores se han obtenido de forma aproximada y los motores 4 y 5 que deben moverse coordinadamente a la larga acaban desfasándose. Como resultado se tiene un desajuste a a la hora de posicionar la herramienta de unos pocos centímetros en algunos casos, lo cual es muy grande para un robot de este tamaño. Sería necesario hacer una análisis más a fondo de la mecánica del robot y realizan un mejor control para solucionar el problema con los motores.

- Hacer un análisis dinámico del robot para mejorar su movimiento y eficiencia.
- Añadir un controlador de tipo PID para reducir el error y tener un mejor control y mejores respuestas.
- Añadir más funciones al robot como podría ser el cálculo de trayectorias para pasar por distintos puntos o añadir sensores externos.

A. Imágenes

A continuación se muestran unas imágenes, tanto del SCORBOT-ER III tanto de la PCB creada.

Figura A.1: SCORBOT-ER III en posición inicial.

Figura A.2: SCORBOT-ER III en posicón de reposo.

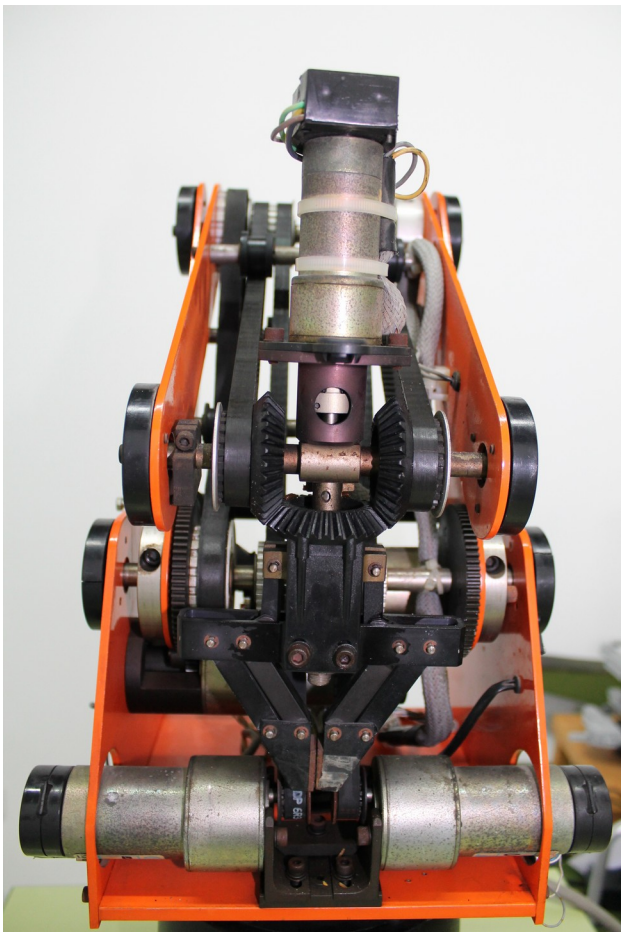

Figura A.3: Vista frontal del SCORBOT-ER III.

Figura A.4: Vista trasera del SCORBOT-ER III.

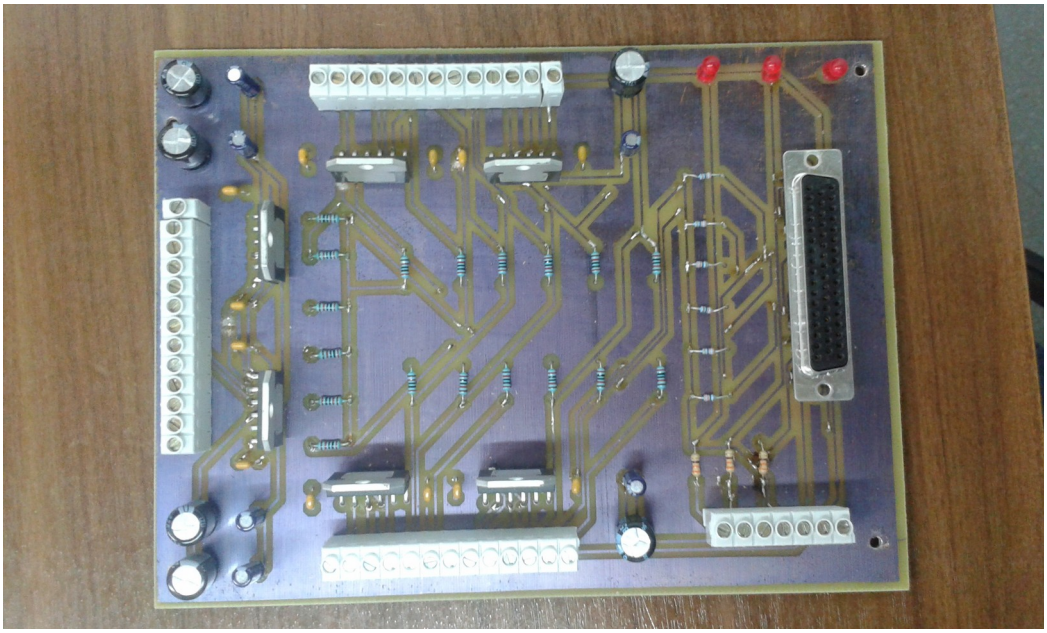

Figura A.5: Circuito impreso, parte superior.

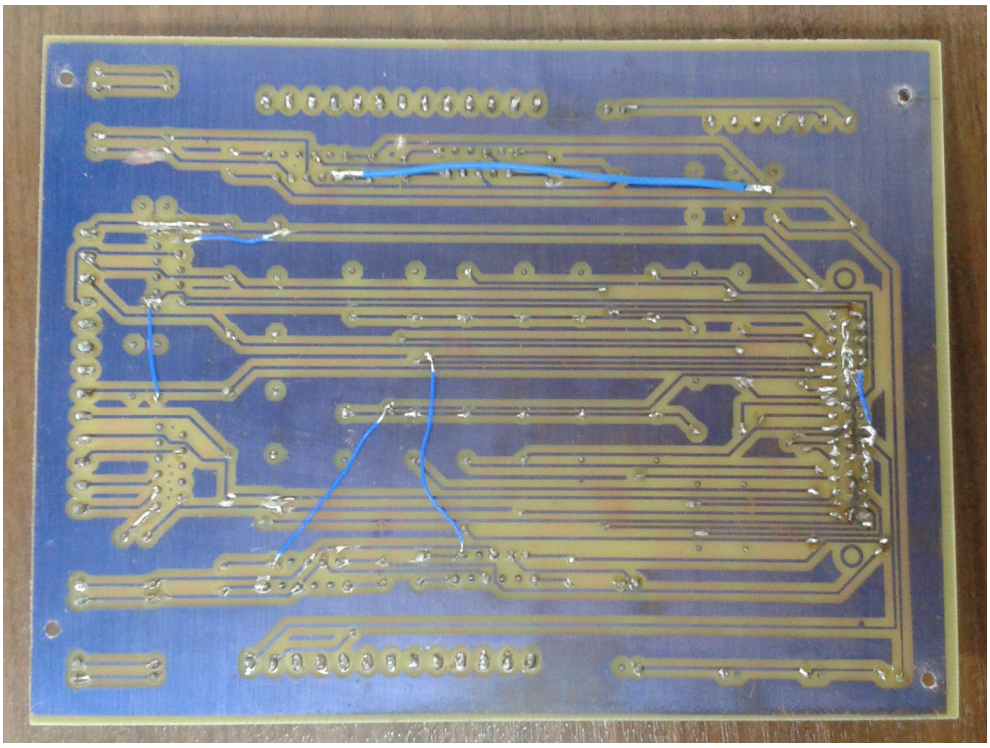

Figura A.6: Circuito impreso, parte inferior.

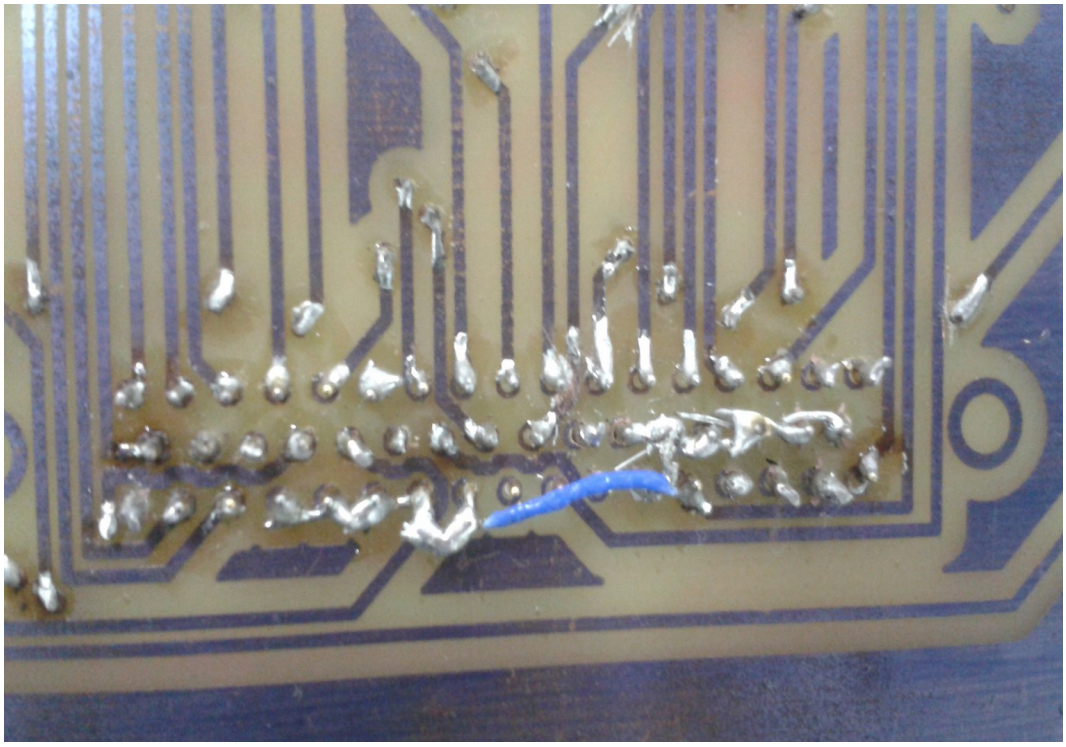

Figura A.7: Detalle de la PCB, conector D50.

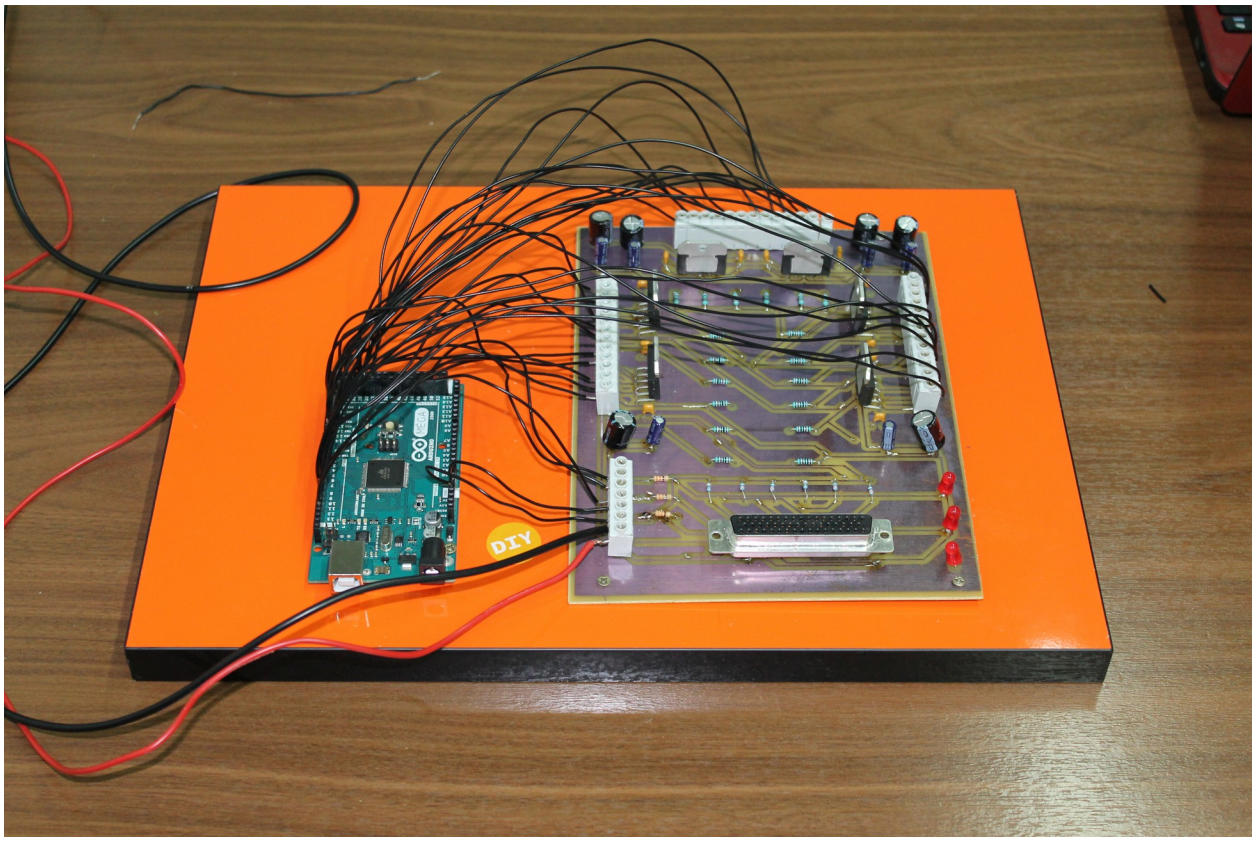

Figura A.8: PCB motada en la base y conectada a Arduino Mega

B. Conector D50

Tabla B.1: Cableado de motores, encoders y microswitches.

Señales del brazo robot					**Conector D50**	
Motor #		**Encoder**		**MS #**	**Color**	**Pin #**
1	Dir.				Blanco	50
	PWM				Gris/Verde	17
2	Dir.				Blanco	49
	PWM				Blanco/Verde	16
3	Dir.				Blanco	48
	PWM.				Naranja/Marrón	15
4	Dir.				Blanco	47
	PWM				Naranja/Verde	14
5	Dir.				Blanco	46
	PWM				Naranja/Azul	13
Grp	PWM				Blanco	45
	Dir.				Naranja/Azul	12
		1	GND		Blanco	33
			P_1		Blanco/Gris	5
		1	V_{LED}		Amarillo	11
			P_0		Blanco/Naranja	2
		2	GND		Blanco	32
			P_1		Blanco/Naranja	21
		2	V_{LED}		Amarillo	27
			P_0		Gris	1
		3	GND		Blanco	31
			P_1		Marrán/Azul	4

Señales del brazo robot			**Conector D50**	
Motor #	**Encoder**	**MS #**	**Color**	**Pin #**
	3 V_{LED}		Amarillo	10
	P_0		Verde	36
	4 GND		Blanco	30
	P_1		Verde/Marrón	20
	4 V_{LED}		Amarillo	26
	P_0		Naranja	35
	5 GND		Blanco	29
	P_1		Verde/Azul	3
	5 V_{LED}		Blanco	9
	P_0		Azul	18
	Grp GND		Blanco	28
	P_1		Gris/Azul	19
	Grp V_{LED}		Blanco	25
	P_0		Blanco/Azul	34
		1 GND	Blanco	46
		MS	Marrón	23
		2 GND	Blanco	45
		MS	Gris	7
		3 GND	Blanco	44
		MS	Naranja	24
		4 GND	Blanco	43
		MS	Verde	8
		5 GND	Blanco	42
		MS	Azul	6
		Grp GND	Blanco	28
		MS	Marrón/Gris	22

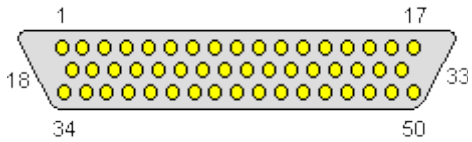

Figura B.1: Conector D50.

C. Diseño de la PCB

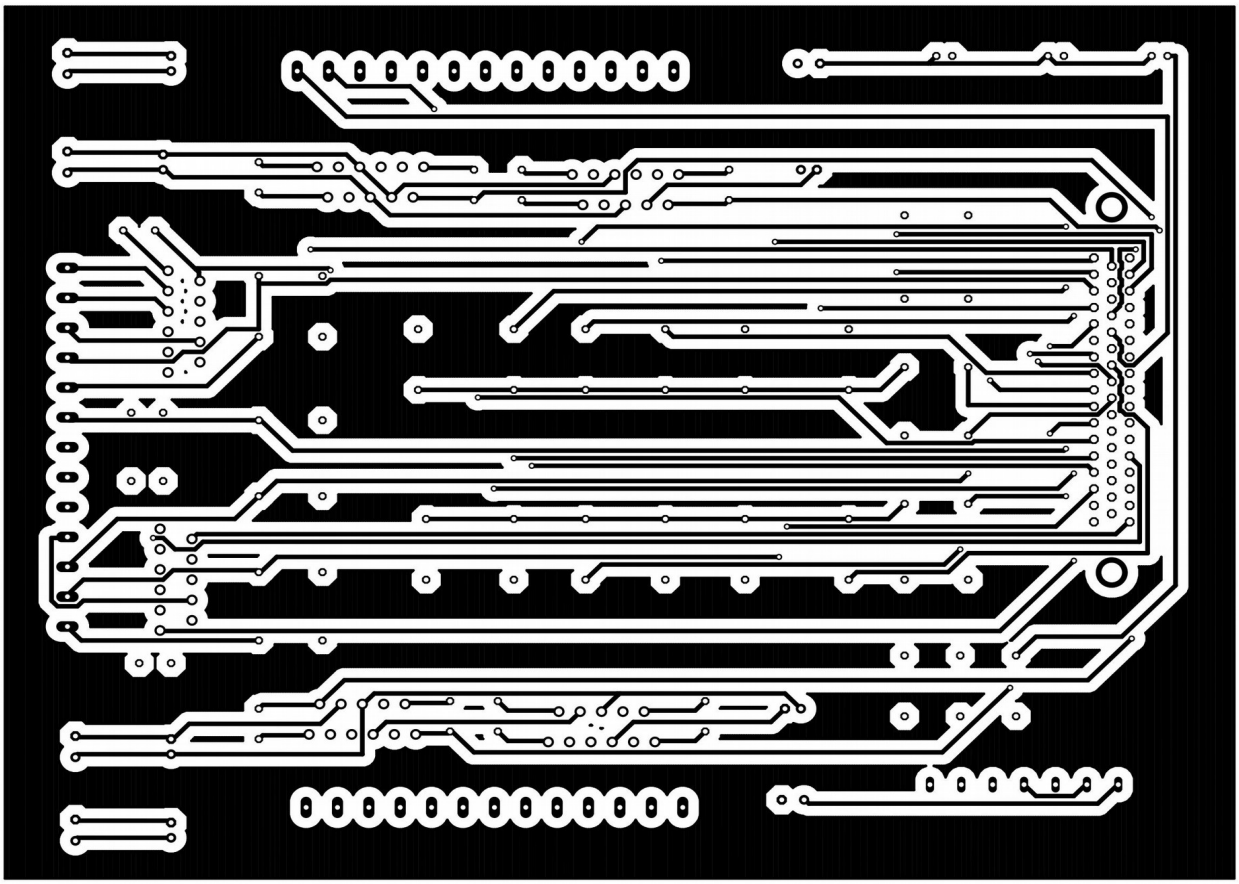

Figura C.1: Parte inferior de la PCB.

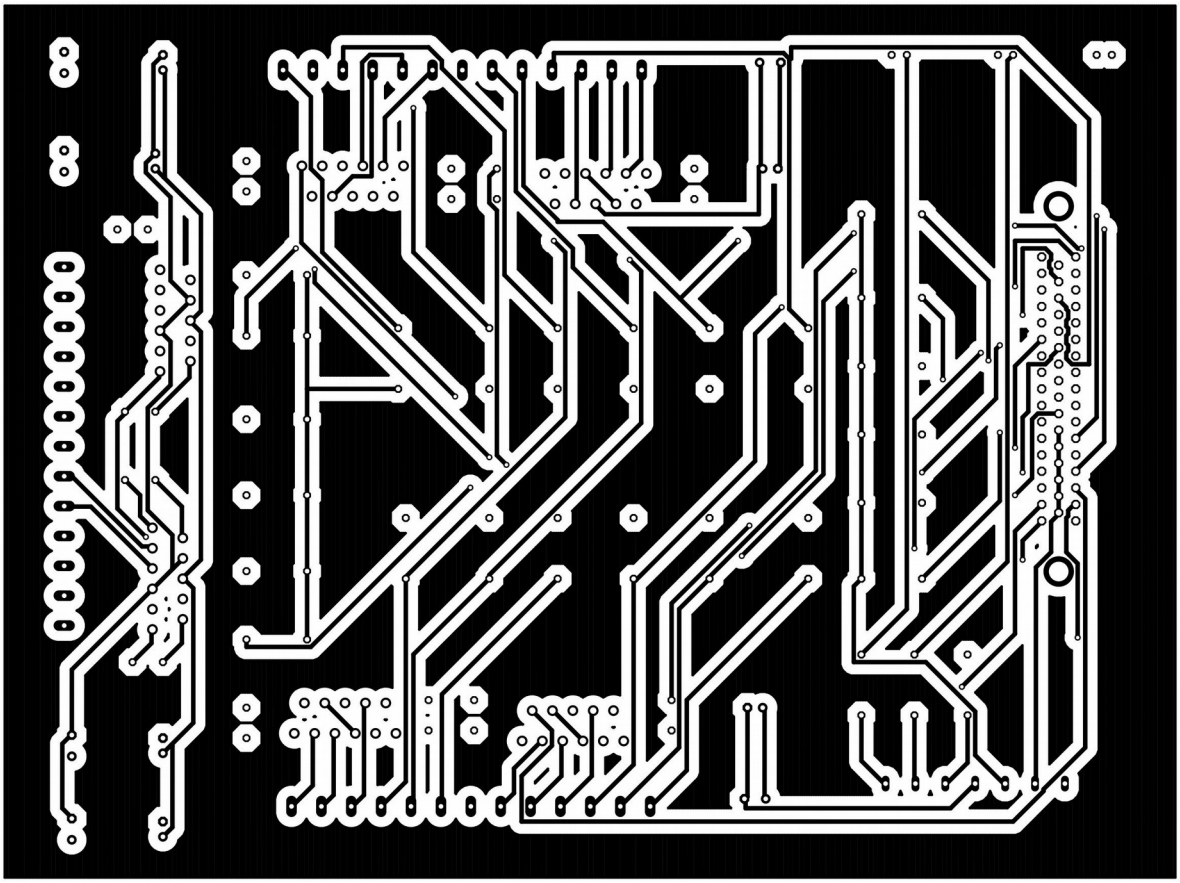

Figura C.2: Parte inferior de la PCB.

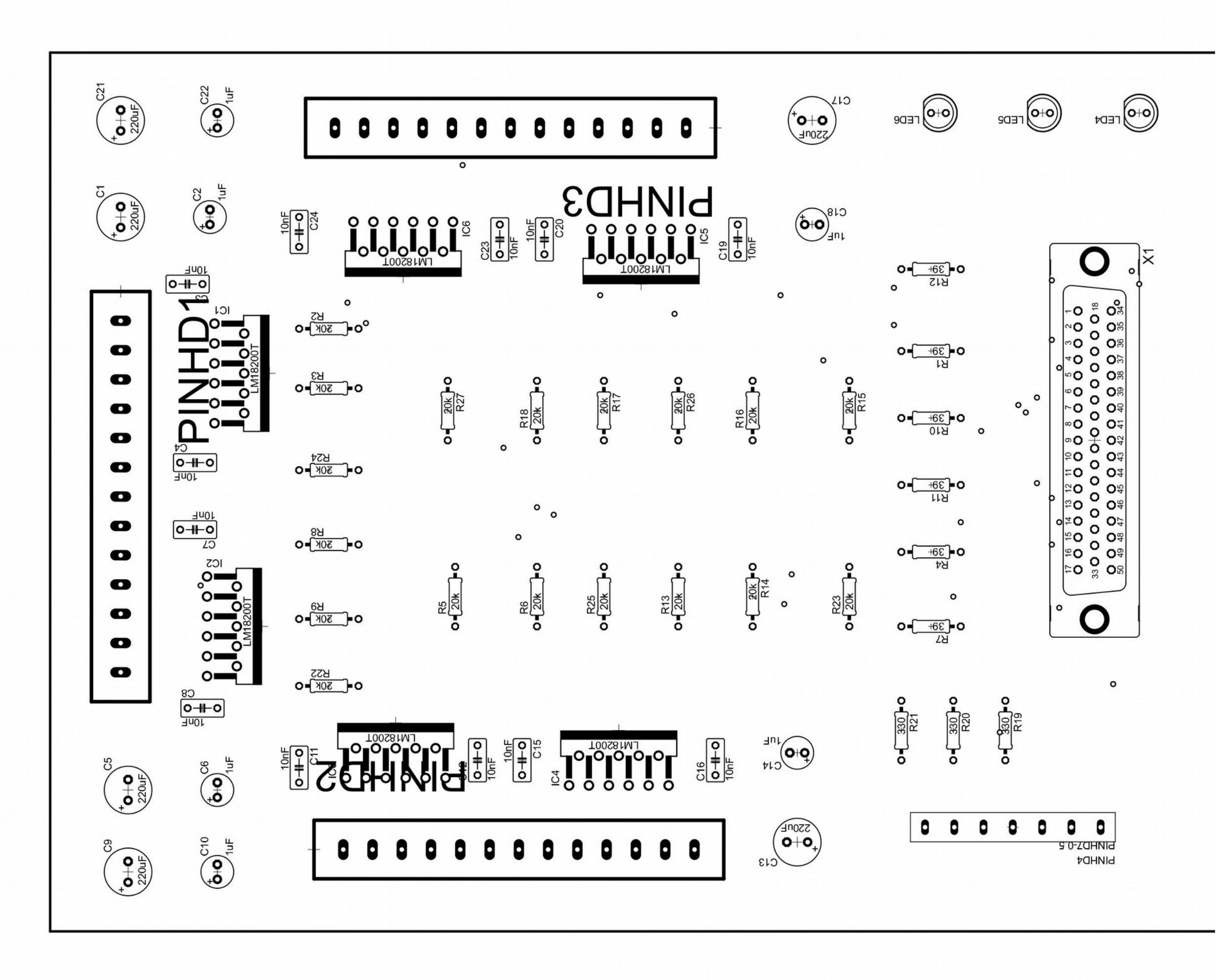

Figura C.3: Componentes de la PCB.

Bibliografía

[1] A. Barrientos, «Capítulo 1. Introducción.», en *Fundamentos de robótica*, Madrid: McGraw-Hill, Interamericana de España, 2007.

[2] A. Barrientos, «Capítulo 2. Morforlogía del robot.», en *Fundamentos de robótica*, Madrid: McGraw-Hill, Interamericana de España, 2007.

[3] Eshed robotec, «SCORBOT-ER III User's Manual.» .

[4] Texas Instrument, «LMD18200 User's manual.» .

[5] A. Barrientos, «Capítulo 4. Cinemática del Robot.», en *Fundamentos de robótica*, Madrid: McGraw-Hill, Interamericana de España, 2007.

[6] Roque J. Saltarén Pazmiño, José Mª Azorín Poveda, Miguel Almonacid Kroeger, y José Mª Sabater Navarro, «Práctica 2.Cinemática del Robot», en *Prácticas de robótica utilizando Matlab*, Escuela Politécnica Superior de Elche.

[7] «MATLAB - El lenguaje de cálculo técnico - MathWorks España.» [En línea]. Disponible en: http://es.mathworks.com/products/matlab/.

[8] MathWorks, «GUI de MATLAB - MATLAB - MathWorks España.» [En línea]. Disponible en: http://es.mathworks.com/discovery/matlab-gui.html.