



UNIVERSIDAD de VALLADOLID



ESCUELA de INGENIERÍAS INDUSTRIALES

**INGENIERÍA TÉCNICA DE TELECOMUNICACIONES  
ESPECIALIDAD SISTEMAS ELECTRÓNICOS**

**PROYECTO FIN DE CARRERA**

# **CONTROL AUTOMÁTICO DE UN BRAZO ROBOT DE 5 GRADOS DE LIBERTAD CON ARDUINO**

**Autor:**

**Yagüe Niño, Juan José**

**Tutor:**

**García Ruiz, Francisco Javier**

**Ingeniería de Sistemas y Automática**

**JULIO – 2013**





## 1. RESUMEN

La robótica nace en décadas recientes con el fin de complementarse con la automatización, aportándole como elemento innovador cierto grado de inteligencia. La automatización está relacionada con el empleo de sistemas mecánicos, electrónicos e informáticos. Entre los sistemas mecánicos se encuentra el uso de robots.

En el contexto industrial, el uso de robots ha permitido sustituir la actividad física del hombre en tareas repetitivas, monótonas o peligrosas, suponiendo un importante paso en la mejora de la producción y el uso de los recursos.

En el contexto de la formación, han surgido sistemas como Arduino, enfocados al diseño electrónico de prototipos de bajo coste. Se trata de una herramienta para hacer que los ordenadores puedan sentir y controlar el mundo físico sobre una plataforma de código abierto, basada en una placa con un sencillo microcontrolador.

Poder usar Arduino para crear objetos interactivos, leyendo datos de una gran variedad de interruptores y sensores, controlando luces, motores y actuadores físicos le han convertido en una herramienta muy versátil y su uso aumenta día a día. E incluso grandes empresas, como Google, participan en proyectos de desarrollo de aplicaciones para acceder a módulos de control de Smartphones con sistema operativo Android.

Por este motivo, intentando profundizar en un sistema tan lleno de posibilidades como Arduino, y aplicándolo a la robótica, imprescindible en la empresa industrial moderna, he decidido desarrollar éste proyecto.

## 2. OBJETIVOS

En el presente proyecto hemos tratado de simular el control de un brazo robótico industrial automatizado. Para ello disponemos de un brazo robot Velleman KSR10 de 5 grados de libertad controlado únicamente de forma manual mediante un mando con botones y una placa Arduino MEGA 2560.

El robot Velleman KSR10 es un brazo con cinco motores y cinco articulaciones, manipulado por una unidad de control; no posee ningún tipo de control de posicionamiento. Se pretende realizar un control automático del mismo para dotarle de cierta inteligencia artificial así como poder desarrollar un sistema completo de automatizado extrapolable a cualquier otro robot. Para ello se dota al mismo de un sistema para poder leer la posición de todos los motores y se crea un programa de control y generación de trayectorias.

El interfaz de control se diseñará a partir de un controlador Arduino Mega 2560. El software de control y comunicación con Arduino se desarrollará a partir de la plataforma .NET de Microsoft.

De este modo, podemos profundizar en varios campos tratados teóricamente a lo largo de la carrera y en otros interesantes de cara al futuro profesional en la industria:

- Sistemas electrónicos de control de procesos.

- Desarrollo de un sistema de trayectorias para el brazo robot.
- Implementación de un sistema de cinemática inversa para el cálculo de posiciones.
- Implementación de un sistema de comunicación entre arduino y .NET.
- Desarrollo de una aplicación en entorno Windows para el control del mismo.
- Diseño electrónico de comunicación y control Arduino vs Brazo robot.
- Obtención del modelo matemático del sistema de sensores de posición para el cálculo de su posición.



Figura 1: Brazo robot Velleman KSR10 y su unidad de control remoto

### 3. ESTUDIO PREVIO

El objetivo final de nuestro proyecto es desarrollar un software capaz de controlar el brazo robot Velleman KSR10 tanto de forma manual como automática.

Inicialmente, el robot era controlable a través de un mando remoto de contactos que únicamente activaban o desactivaban los diferentes motores del mismo, sin tener en cuenta la llegada al final del recorrido. Los motores del brazo robot tienen una corona sin fin, de tal modo que, llegado al final del recorrido del eje, en vez de parar, dan saltos, pudiendo llegar a deteriorar los engranajes.

Uno de los puntos a tener en cuenta es que cada uno de los motores debe parar al llegar al final del recorrido de su eje, salvaguardando así la estructura del brazo robot y sus componentes.

Para la interacción con el brazo robot se decide diseñar una aplicación para PC desde la que controlar ambos modos (manual y automático) así como la posibilidad de programar trayectorias.

Tras el estudio de las diferentes posibilidades de control, se determina que:

- Arduino Mega 2560 será el controlador utilizado para mantener la comunicación con el dispositivo de control, en nuestro caso el PC, y el brazo robot. Los cálculos

- de posicionamiento de control automático serán gestionados por Arduino, reduciendo las atribuciones de la aplicación PC.
- Cada eje del brazo robot tendrá instalada una resistencia para el cálculo de su posición.
  - Los motores quedan controlados a partir de puertos en H L293D y variando la velocidad a partir de las salidas PWM de Arduino.
  - Cada motor tendrá configurado un PID, buscando la mayor rapidez y precisión en su posicionamiento.
  - No es posible aplicar la cinemática inversa y el algoritmo de Bresenham al brazo robot por lo que se especifican tres tipos de movimientos para el control total del brazo con movimientos naturales.
  - Al descartar la aplicación de la cinemática inversa, no es necesario el uso del modelado matemático que relaciona el valor de la resistencia con el ángulo del eje, si no que se mantendrá el uso del parámetro recibido en Arduino con una resolución entre 0 y 1024, de tal forma que se eviten errores de precisión.
  - El software Arduino controlará el movimiento automatizado, los PID de cada motor, los límites de movimiento de cada eje para evitar deteriorar los engranajes de la reductora y almacenará la información de programación de movimientos.
  - El software .NET será el encargado de interactuar con el usuario, permitiendo el control manual, el control automático, la programación de trayectorias así como controles de parada automática y resto de funcionalidades típicas de control.
  - Arduino estará conectado al PC mediante cable USB, utilizando la alimentación de éste último para alimentar la placa y la circuitería lógica restante.
  - Los motores serán alimentados mediante una fuente de alimentación externa de 6V y 1000mA.

### Sensores de posición

Para el control de posicionamiento del extremo final del brazo robot Velleman KSR10 sobre el plano XYZ es preciso implementar un sistema de sensores. Los motores del brazo robot quedan encapsulados dentro de unas cajas junto con su correspondiente reductora.

Inicialmente se estudia la posibilidad de incluir un encoder en la parte exterior de cada una de las cajas, pero por la estructura del mismo resultaba imposible en algunos casos. La opción de incluir el encoder dentro del encapsulado se descartó finalmente debido a la falta de espacio.

La solución final consistió en colocar resistencias variables en cada uno de los ejes del robot, con el rotor unido a una varilla metálica anclada a la parte fija contigua, de tal forma que una variación del valor de la resistencia suponga una variación en grados del brazo robot.

Para tal fin se escoge la resistencia CA9 con encapsulado CA9 H2.5 y rotor STD con relación lineal.

Cada una de las resistencias quedará conectada a las entradas analógicas de Arduino.

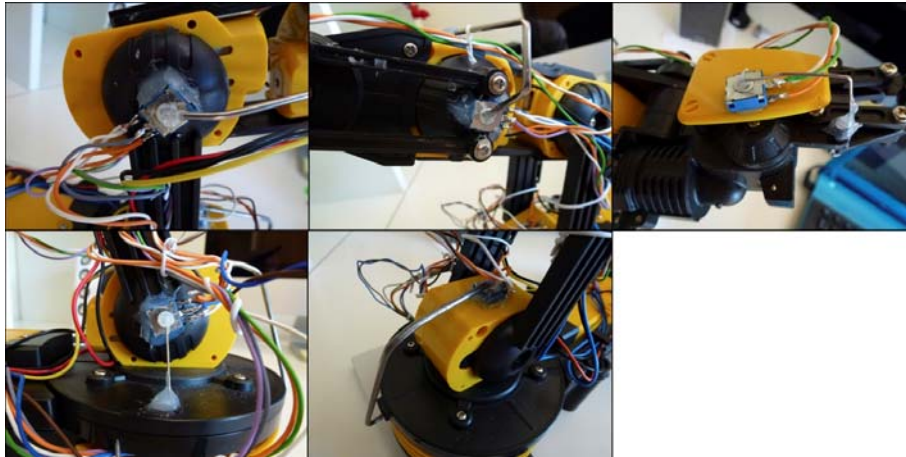


Figura 2: sensores de posición instalador en el brazo robot

### Transmisión de datos Arduino - .NET

Una de las “limitaciones” de Arduino en relación a otros lenguajes de programación, es la inexistencia del tipo de datos “String”. Por tanto, al recibir una cadena de caracteres habrá que procesarla carácter a carácter almacenándolo, por ejemplo, en un vector de caracteres. Esto supone un inconveniente a la hora de recibir los parámetros de configuración de trayectorias.

Como solución se ha ideado un sistema de transmisión de tramas, de tal modo que el software .NET creará una trama de información con la configuración completa de todas las instrucciones así como control de errores de envío.

## 4. SOFTWARE ARDUINO

Para el desarrollo del software del controlador se han utilizado la librería PID de Arduino. Cada motor será controlado por un PID configurado acorde a sus limitaciones y necesidades.

Se han definido dos modos de funcionamiento: manual y automático, los cuales serán controlados a través de la aplicación diseñada en .NET

Para el modo de control manual, según el parámetro recibido accionará el movimiento de los motores en uno u otro sentido, siempre limitados por el máximo valor de giro de los sensores de posición.

Para el control automático, se han definido un máximo de 25 instrucciones, las cuales son definidas por el usuario a través de la aplicación .NET. Una vez programados, son enviados a Arduino en una trama, tal y como se especifica en el punto 3, para su programación y control automatizado. El brazo robot seguirá la trayectoria predefinida hasta su finalización o hasta la acción del botón de Parada de Emergencia.

## 5. SOFTWARE .NET

La aplicación .NET es el entorno gráfico de interacción con el usuario. Corresponde con un archivo ejecutable .exe que funciona bajo sistemas operativos Windows.



Figura 3: estado aplicación comunicada con Arduino

Desde el panel de control es posible realizar tareas de configuración de Arduino, conocer su estado así como la consola de comunicación serie entre Arduino y la aplicación.

Además, se ha insertado un panel de control de posicionamiento de los sensores. Puede resultar útil su uso en caso de realizar tareas de reconfiguración del algoritmo de Arduino.

En la consola se muestra la información relativa a la comunicación entre Arduino y la aplicación de control. En caso de producirse errores de transmisión en las tramas de localización de posición o configuración de trayectorias, se mostrará la información de tal modo que sea el usuario quien decida el modo de actuar.

### **MODO MANUAL**

En el modo manual, es posible controlar cualquier movimiento del brazo robot haciendo clic sobre el icono de desplazamiento correspondiente. Mientras mantenga pulsado el botón, el brazo robot se desplazará, parándose al llegar al límite de giro o cuando el usuario deje de pulsar.

La configuración de trayectorias se lleva a cabo en modo manual, tal y como se describe en el siguiente apartado. No será posible alternar entre modo manual y automático si no se ha enviado una trayectoria de configuración a Arduino.

### **CONFIGURACIÓN DE TRAYECTORIAS**

Es posible automatizar una serie de movimientos del Brazo Robot Velleman KSR10, de tal modo que sea capaz de desarrollar un trabajo de automatización.

Una vez creada la trayectoria en la aplicación, se envía a través de puerto serie a Arduino en una trama de configuración. Arduino recibirá la información por partida doble, de tal



modo que se eviten errores de transmisión. Si se produce algún error, quedará a la espera de una nueva transmisión.

Además, es posible guardar trayectorias predefinidas en archivos de configuración, eliminar configuraciones, editarlas y otra serie de funcionalidades diversas.

## **MODO AUTOMÁTICO**

Una vez cargada la configuración de trayectorias a Arduino, es posible alternar entre modo manual y automático.

Se ha insertado un botón de parada de emergencia en la aplicación .NET así como instalado un botón de Reset de Arduino en el propio robot.

Arduino almacena la información de trayectorias hasta que sean sobrescritas o este sea reseteado o apagado.

## **6. CONCLUSIONES**

En el Mundo industrial que vivimos actualmente, la automatización es base fundamental en la industria, no sólo para suprimir tareas peligrosas o monótonas, sino por la eficiencia, la supresión de errores y la optimización de recursos.

El conocer los objetivos del proyecto, mezclando Arduino + robótica + programación, me convencieron para lograr el resultado que ahora podemos contemplar.

Haber tenido que estudiar a fondo el brazo robótico Velleman KSR10 y su automatización me ha introducido en el mundo de la robótica, un campo nunca tocado a lo largo de la carrera.

Por el contrario, otros campos sí tocados en la carrera han sido reforzados, mejorados e incluso me he percatado de la utilidad de metodologías, cálculos, algoritmos, etc. a los que nunca había dado un uso profesional, principalmente relacionados con los sistemas de control, la programación y la transmisión de datos.

Todos los objetivos marcados en un primer momento se han cumplido, si bien es cierto que me habría gustado poder perfeccionar algunos otros, como la aplicación de la cinemática inversa y el Algoritmo de Bresenham pero que, debido a las limitaciones de Brazo Robot Velleman KSR10 no ha sido posible. Sin embargo, el comportamiento del brazo es muy natural y la repetitividad del sistema de sensores instalado fiable.

Haber estudiado Arduino con tanto detenimiento me ha llevado a comprender la importancia que tiene en el mundo de la formación y el diseño de prototipos. Creo que el desarrollo de Arduino no ha hecho más que empezar y que la evolución tecnológica lo hará más dependiente en la tecnología del futuro a corto plazo.

En general, este proyecto ha cumplido mis expectativas y mis conocimientos se han visto reforzados en los campos que considero más importantes para mi carrera: control y automatización de sistemas electrónicos, programación y transmisión de datos.





UNIVERSIDAD de VALLADOLID



ESCUELA de INGENIERÍAS INDUSTRIALES

**INGENIERÍA TÉCNICA DE TELECOMUNICACIONES  
ESPECIALIDAD SISTEMAS ELECTRÓNICOS**

**PROYECTO FIN DE CARRERA**

# **CONTROL AUTOMÁTICO DE UN BRAZO ROBOT DE 5 GRADOS DE LIBERTAD CON ARDUINO**

**Autor:**

**Yagüe Niño, Juan José**

**Tutor:**

**García Ruiz, Francisco Javier**

**Ingeniería de Sistemas y Automática**

**JULIO – 2013**





# Contenido

<b>1. INTRODUCCIÓN</b> .....	5
1.1 Resumen .....	5
1.2 Objetivos.....	5
1.3 Planteamiento del sistema de ejecución.....	6
1.4 Descripción de la planta e interconexión del sistema.....	7
<b>2. FUNDAMENTOS TEÓRICOS</b> .....	9
2.1 REGULACIÓN AUTOMÁTICA.....	9
2.1.1 <i>Sistemas en lazo abierto</i> .....	9
2.1.2 <i>Sistemas en lazo cerrado</i> .....	9
2.2 PID.....	10
2.2.1 <i>Introducción</i> .....	10
2.2.2 <i>Descripción</i> .....	10
2.2.3 <i>Acción Proporcional</i> .....	10
2.2.4 <i>Acción Integral</i> .....	11
2.2.5 <i>Acción Derivativa</i> .....	12
2.2.6 <i>Modelo matemático del PID</i> .....	12
2.2.7 <i>Método de sintonización de controladores PID de Ziegler-Nichols</i> .....	13
2.3 CINEMÁTICA DEL ROBOT .....	15
2.3.1 <i>Introducción</i> .....	15
2.3.2 <i>Cinemática Inversa</i> .....	15
2.3.3 <i>Método geométrico de Cálculo de cinemática inversa</i> .....	16
2.4 PLANIFICACIÓN DE TRAYECTORIAS .....	19
2.4.1 <i>Introducción</i> .....	19
2.4.2 <i>Tipos de trayectorias</i> .....	19
2.4.3 <i>Algoritmo de Bresenham</i> .....	20
2.4.4 <i>Planificación de trayectorias y cinemática inversa del robot</i> .....	22
<b>3. DESCRIPCIÓN DEL HARDWARE</b> .....	25
3.1 ARDUINO .....	25
3.1.1 <i>Introducción</i> .....	25
3.1.2 <i>Arduino Mega 2560</i> .....	26
3.1.3 <i>Pulse Width Modulation (PWM)</i> .....	29
3.1.4 <i>Licencia</i> .....	30
3.2 BRAZO ROBOT VELLEMAN KSR10 .....	31
3.2.1 <i>Introducción</i> .....	31
3.2.2 <i>Especificaciones</i> .....	32



<b>3.3</b>	<b>PUENTE EN H</b> .....	32
<b>3.3.1</b>	<b>Introducción</b> .....	32
<b>3.3.2</b>	<b>Puente en H L293D</b> .....	33
<b>3.4</b>	<b>REGULADOR DE TENSIÓN LM7805</b> .....	35
<b>3.5</b>	<b>RESISTENCIAS VARIABLES</b> .....	35
<b>3.5.1</b>	<b>Introducción</b> .....	35
<b>3.5.2</b>	<b>Tipos de resistencias variables</b> .....	35
<b>3.5.3</b>	<b>Resistencia variable lineal CA9</b> .....	36
<b>3.6</b>	<b>USB</b> .....	36
<b>4.</b>	<b>DESCRIPCIÓN DEL SOFTWARE</b> .....	37
<b>4.1</b>	<b>ARDUINO</b> .....	37
<b>4.1.1</b>	<b>Entorno de Desarrollo para Arduino</b> .....	37
<b>4.1.2</b>	<b>Lenguaje de programación de Arduino</b> .....	38
<b>4.1.3</b>	<b>Estructura de un programa Arduino</b> .....	38
<b>4.1.4</b>	<b>Comunicación Serie a través de Arduino</b> .....	39
<b>4.1.5</b>	<b>Librería PID Arduino</b> .....	42
<b>4.2</b>	<b>VISUAL STUDIO 2012</b> .....	43
<b>4.2.1</b>	<b>Introducción</b> .....	43
<b>4.2.2</b>	<b>Versiones de Visual Studio 2012 y alternativas</b> .....	44
<b>4.2.3</b>	<b>VB .NET</b> .....	44
<b>4.2.4</b>	<b>Framework .NET</b> .....	45
<b>4.2.5</b>	<b>Estructura de una aplicación .NET</b> .....	45
<b>5.</b>	<b>ESTUDIO PREVIO</b> .....	47
<b>5.1</b>	<b>OBJETIVO DEL PROYECTO</b> .....	47
<b>5.2</b>	<b>PLACA ARDUINO</b> .....	47
<b>5.3</b>	<b>PUENTE EN H L293D</b> .....	48
<b>5.4</b>	<b>SENSORES DE POSICIÓN</b> .....	49
<b>5.5</b>	<b>CINEMÁTICA INVERSA Y ALGORITMO DE BRESENHAM</b> .....	53
<b>5.6</b>	<b>PID</b> .....	54
<b>5.7</b>	<b>PROGRAMACIÓN DE DISPOSITIVOS</b> .....	55
<b>5.8</b>	<b>CONCLUSIONES</b> .....	56
<b>6.</b>	<b>INTERCONEXIÓN DEL SISTEMA</b> .....	57
<b>6.1</b>	<b>DETALLE CONEXIONADO</b> .....	57
<b>6.2</b>	<b>ESQUEMA ELÉCTRICO</b> .....	60
<b>7.</b>	<b>DESARROLLO DE SOFTWARE</b> .....	69
<b>7.1</b>	<b>INTRODUCCIÓN</b> .....	69
<b>7.2</b>	<b>REQUISITOS NO FUNCIONALES</b> .....	70
<b>7.3</b>	<b>REQUISITOS FUNCIONALES</b> .....	71
<b>7.4</b>	<b>TRANSMISIÓN DE CONFIGURACIÓN DE TRAYECTORIAS</b> .....	73



<b>7.5</b>	<b>SOFTWARE ARDUINO</b> .....	75
7.5.1	<i>Librerías</i> .....	75
7.5.2	<i>Variables y Constantes</i> .....	75
7.5.3	<b>SETUP</b> .....	77
7.5.4	<b>LOOP</b> .....	78
7.5.5	<i>Control Manual</i> .....	78
7.5.6	<i>Programación de Trayectorias</i> .....	79
7.5.7	<i>Envío información sensores</i> .....	82
7.5.8	<i>Control Automático</i> .....	83
7.5.9	<i>Función Parada de Emergencia</i> .....	83
7.5.10	<i>Función control Mano – Modo Automático</i> .....	84
7.5.11	<i>Función control posición – Modo Automático</i> .....	85
<b>7.6</b>	<b>SOFTWARE .NET</b> .....	87
7.6.1	<i>Introducción</i> .....	88
7.6.2	<i>Librerías</i> .....	88
7.6.3	<i>Variables y Constantes</i> .....	89
7.6.4	<i>Inicialización Formulario</i> .....	89
7.6.5	<i>Conexión con Arduino</i> .....	90
7.6.6	<i>Lectura puerto serie</i> .....	92
7.6.7	<i>Control manual de motores</i> .....	95
7.6.8	<i>Definición de posiciones y eventos en trayectorias</i> .....	95
7.6.9	<i>Envío configuración trayectorias</i> .....	97
7.6.10	<i>Almacenamiento información trayectorias, carga y borrado de eventos</i> .....	98
7.6.11	<i>Cambio Modo Manual / Automático</i> .....	101
7.6.12	<i>Parada de Emergencia en modo Automático</i> .....	101
7.6.13	<i>Consola de transmisión y detección de errores</i> .....	102
<b>8.</b>	<b>GUIA DE USUARIO</b> .....	103
8.1	<b>INTRODUCCIÓN</b> .....	103
8.2	<b>DESCRIPCIÓN DEL SISTEMA</b> .....	103
8.3	<b>PUESTA EN MARCHA</b> .....	105
8.4	<b>PANEL DE CONTROL</b> .....	107
8.5	<b>MODO MANUAL</b> .....	108
8.6	<b>PROGRAMACIÓN DE TRAYECTORIAS</b> .....	110
8.7	<b>MODO MANUAL</b> .....	111
8.8	<b>MANTENIMIENTO PREVENTIVO</b> .....	111
<b>9.</b>	<b>CONCLUSIONES</b> .....	113
<b>10.</b>	<b>ANEXO</b> .....	115
10.1	<b>GUÍA INSTALACIÓN IDE ARDUINO</b> .....	115
10.2	<b>GUÍA INSTALACIÓN VISUAL STUDIO 2012</b> .....	117



---

10.3	CÓDIGO FUENTE ARDUINO .....	120
10.4	CÓDIGO FUENTE VISUAL BASIC .NET .....	136
10.5	DATASHEET L293D.....	151
<b>11.</b>	<b>BIBLIOGRAFÍA.....</b>	<b>157</b>



# 1. INTRODUCCIÓN

## 1.1 Resumen

La robótica nace en décadas recientes con el fin de complementarse con la automatización, aportándole como elemento innovador cierto grado de inteligencia. La automatización está relacionada con el empleo de sistemas mecánicos, electrónicos e informáticos. Entre los sistemas mecánicos se encuentra el uso de robots.

En el contexto industrial, el uso de robots ha permitido sustituir la actividad física del hombre en tareas repetitivas, monótonas o peligrosas, suponiendo un importante paso en la mejora de la producción y el uso de los recursos.

En el contexto de la formación, han surgido sistemas como Arduino, enfocados al diseño electrónico de prototipos de bajo coste. Se trata de una herramienta para hacer que los ordenadores puedan sentir y controlar el mundo físico sobre una plataforma de código abierto, basada en una placa con un sencillo microcontrolador-

Poder usar Arduino para crear objetos interactivos, leyendo datos de una gran variedad de interruptores y sensores, controlando luces, motores y actuadores físicos le han convertido en una herramienta muy versátil y su uso aumenta día a día. E incluso grandes empresas, como Google, participan en proyectos de desarrollo de aplicaciones para acceder a módulos de control de Smartphones con sistema operativo Android.

Por este motivo, intentando profundizar en un sistema tan lleno de posibilidades como Arduino, y aplicándolo a la robótica, imprescindible en la empresa industrial moderna, he decidido desarrollar éste proyecto.

## 1.2 Objetivos

En el presente proyecto hemos tratado de simular el control de un brazo robótico industrial automatizado. Para ello disponemos de un brazo robot Velleman KSR10 de 5 grados de libertad controlado únicamente de forma manual mediante un mando con botones y una placa Arduino MEGA 2560.

El robot Velleman KSR10 es un brazo con cinco motores y cinco articulaciones, manipulado por una unidad de control; no posee ningún tipo de control de posicionamiento. Se pretende realizar un control automático del mismo para dotarle de cierta inteligencia artificial así como poder desarrollar un sistema completo de automatizado extrapolable a cualquier otro robot. Para ello se dota al mismo de un sistema para poder leer la posición de todos los motores y se crea un programa de control y generación de trayectorias.

El interfaz de control se diseñará a partir de un controlador Arduino Mega 2560. El software de control y comunicación con arduino se desarrollará a partir de la plataforma .NET de Microsoft.

De este modo, podemos profundizar en varios campos tratados teóricamente a lo largo de la carrera y en otros interesantes de cara al futuro profesional en la industria:

- Sistemas electrónicos de control de procesos.
- Desarrollo de un sistema de trayectorias para el brazo robot.
- Implementación de un sistema de cinemática inversa para el cálculo de posiciones.
- Implementación de un sistema de comunicación entre arduino y .NET.
- Desarrollo de una aplicación en entorno Windows para el control del mismo.
- Diseño electrónico de comunicación y control Arduino vs Brazo robot.
- Obtención del modelo matemático del sistema de sensores de posición para el cálculo de su posición.



Figura 1: Brazo robot Velleman KSR10 y su unidad de control remoto

### 1.3 Planteamiento del sistema de ejecución

El siguiente objetivo será plantear el funcionamiento final de la planta:

- El brazo robot debe ser controlado de forma manual a través de la aplicación .NET.
- El brazo robot debe trabajar de forma automatizada previa programación a través de la aplicación .NET



## 1.4 Descripción de la planta e interconexión del sistema

La planta consta de un brazo robótico Velleman KSR10, al que se le han instalado unas resistencias variables en cada uno de los ejes de rotación, conectados a las entradas de lectura analógica de Arduino, con una resolución de movimiento entre 0 y 1023. La parte móvil de las resistencias está anclada a la parte fija de la parte adyacente, de tal forma que una variación de posición en cualquiera de los ejes se convierte en una variación de su sensor de posición.

Tras diferentes cálculos empíricos se determina una relación lineal entre la variación en grados y la resolución del parámetro recibido. En apartados posteriores se detalla el cálculo de dicha relación.

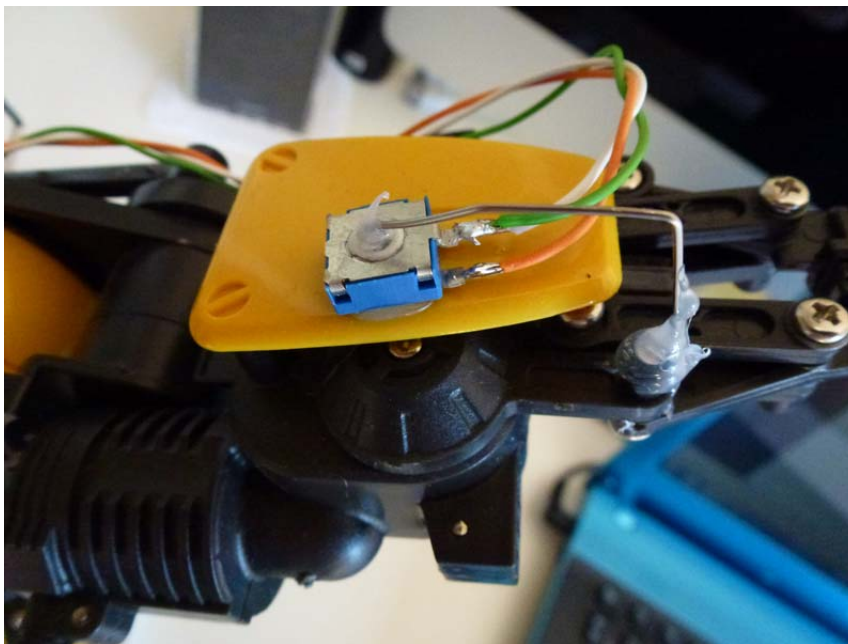


Figura 2: Sensores de movimiento del brazo robot

El controlador del brazo será Arduino Mega. Como el software de control será una aplicación Windows, el dispositivo quedará conectado mediante USB al PC y emulando un puerto de transmisión serie.

La alimentación de Arduino, así como la circuitería lógica de control, proviene de la alimentación USB del PC.

La alimentación de los motores proviene de un alimentador de corriente continua de 6V 1000mA.

Los motores de los ejes del brazo robot son DC. A partir del software de control, Arduino determina el movimiento de los mismos, siendo unos puentes en H L293D los que hacen girar los motores hacia uno u otro lado.



La placa diseñada incluye un regulador de tensión LM7805 que puede ser utilizado para alimentar la circuitería lógica, arduino y un módulo bluetooth de tal forma que podría controlarse el brazo robot mediante el bluetooth y sin estar conectado por USB.

El software .NET es el encargado del control manual del robot y de la programación de trayectorias pero es arduino quien determina el comportamiento de los motores.

## 2. FUNDAMENTOS TEÓRICOS

### 2.1 REGULACIÓN AUTOMÁTICA

La regulación automática o teoría de control es una rama de la ingeniería encargada el control de procesos. Por ejemplo, el regulador de velocidad de un automóvil es un sistema de control.

El estudio de estos sistemas dinámicos se lleva a cabo tratándolos como bloques con una entrada y una salida. Entre los bloques se encuentran los elementos que afectan a la señal. La función matemática del bloque se denomina función de transferencia. La salida del sistema se llama referencia y corresponde con el valor de la señal tras la actuación de la función de transferencia.

#### 2.1.1 Sistemas en lazo abierto

Los sistemas en lazo abierto son aquellos en los que la salida depende únicamente de la entrada del circuito y de la función de transferencia del sistema. Su valor nunca depende del estado actual ni de estados anteriores, es decir, para una misma entrada la salida siempre será la misma:



Figura 3: sistema en lazo abierto

#### 2.1.2. Sistemas en lazo cerrado

Los sistemas en lazo cerrado son aquellos en los que la salida depende de la señal de entrada así como de la salida. De éste modo, se comparan entrada y salida, de tal forma que una misma entrada no siempre tendrá la misma salida si no que dependerá de la salida actual. Por ejemplo, un sistema controlador de temperatura no activa la calefacción indefinidamente si no que la salida depende de la entrada y de la temperatura actual.

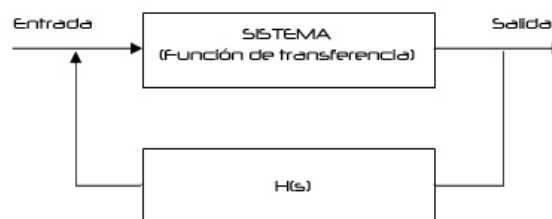


Figura 4: sistema en lazo cerrado

## 2.2 PID

### 2.2.1 Introducción

Un PID es un mecanismo de control de un sistema o planta basado en la realimentación de información para el cálculo de la desviación o error entre el valor deseado o un valor medido y el que realmente se desea obtener, de tal forma que dicho PID aplica una acción de ajuste correctora.

### 2.2.2 Descripción

El algoritmo de cálculo del mecanismo de control consta de tres parámetros:

- Proporcional: proporcional al error actual, de forma que decrece en estado estacionario.
- Integral: proporcional a la integral del error, de forma que disminuirá el error estacionario.
- Derivativo: proporcional a la derivada del error, de forma que considera la tendencia de error y actúa para eliminarlo.

El objetivo de éstas acciones de control es que el comportamiento de la planta se adapte a un setpoint deseado en la menor tardanza posible y con la mayor precisión posible.

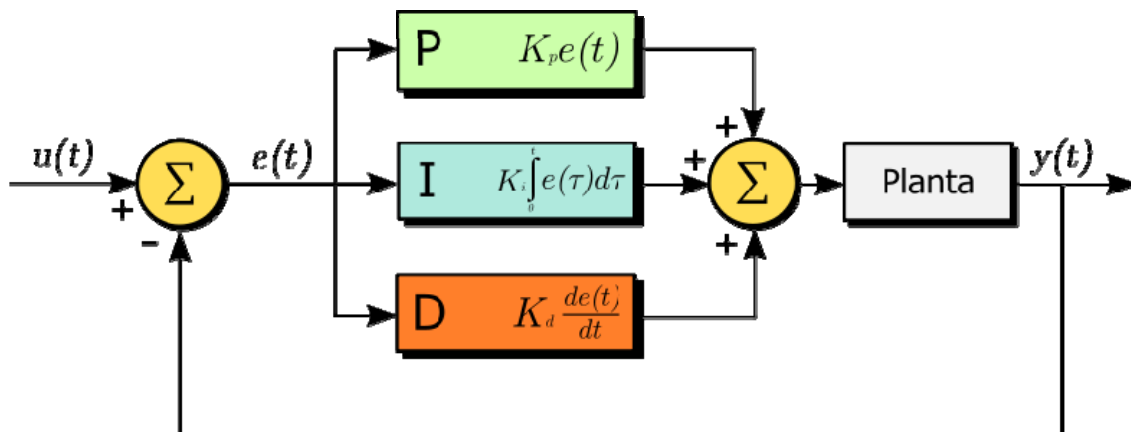


Figura 5: diagrama de bloques de un PID

### 2.2.3 Acción Proporcional

El componente proporcional consiste en el producto entre la señal de error y una constante proporcional  $K_p$  con el objetivo de que el error en estado estacionario se aproxime a cero. Es, por tanto, una relación lineal entre el valor deseado y el actual.

Estos valores sólo serán óptimos en un determinado rango, de tal forma que existe un valor límite a partir del cual se produce un fenómeno denominado sobreoscilación.

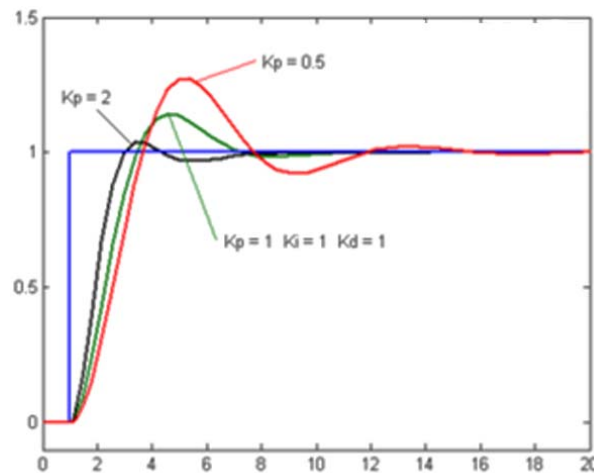


Figura 6: respuesta proporcional del sistema ante una señal de referencia

Como la parte proporcional no tiene en cuenta el tiempo, no mejorará el error en estado estacionario aunque sí responde de forma rápida ante el error de referencia. En este punto entran en acción las componentes integral y derivativa, de tal forma que la variación esté relacionada con el tiempo de variación.

Matemáticamente puede expresarse como:

$$Cp(t) = Kp \cdot e(t)$$

#### 2.2.4 Acción Integral

El componente integral consiste en el producto entre la integral de la señal de error y una constante proporcional  $K_i$  con el objetivo de disminuir y eliminar el error en estado estacionario, provocado por la componente proporcional.

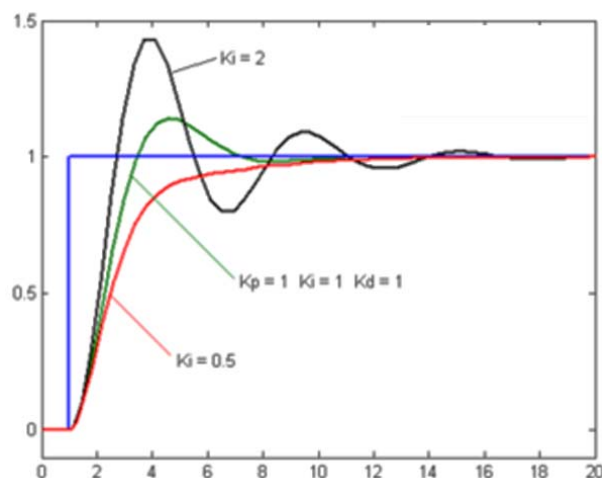


Figura 7: respuesta integral del sistema ante una señal de referencia

La componente integral se adapta al tiempo de variación, siendo más precisa cuanto más lenta sea ( $K_i=1/T_i$ ), de tal modo que existe un valor máximo a partir del cual pueden aumentar las amortiguaciones.

Matemáticamente puede expresarse como:

$$C_i(t) = K_i \cdot \int_0^t e(t) dt$$

### 2.2.5 Acción Derivativa

El componente derivativo consiste en el producto entre la derivada de la señal de error y una constante proporcional  $K_d$ . Únicamente actúa cuando se produce un cambio en el valor absoluto del error; es decir, si el error es constante, sólo actúan los modos proporcional e integral).

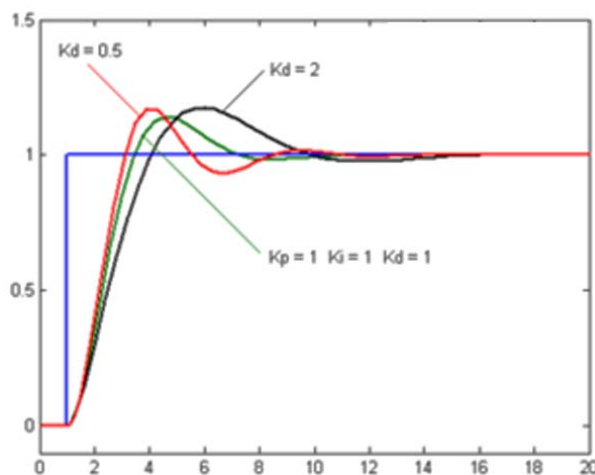


Figura 8: respuesta derivativa del sistema ante una señal de referencia

La idea es que ésta función corrija el error proporcionalmente a la velocidad en que se produce, de tal forma que no se incremente. Ésta acción es adecuada en sistemas con retardo puesto que actúa de forma rápida a las perturbaciones.

La desventaja de dicha acción es su excesiva sensibilidad al ruido por lo que suele ser poco utilizada.

Matemáticamente puede expresarse como:

$$C_d(t) = K_d \cdot \frac{de}{dt}$$

### 2.2.6 Modelo matemático del PID

La ecuación final de todas las acciones supone:

$$y(t) = Kp \cdot e(t) + Ki \cdot \int_0^t e(t)dt + Kd \cdot \frac{de}{dt}$$

### 2.2.7 Método de sintonización de controladores PID de Ziegler-Nichols

El método de Ziegler-Nichols permite ajustar un controlador PID de forma empírica, sin necesidad de conocer las ecuaciones del sistema a controlar.

Los valores propuestos por éste método intentan conseguir en el sistema realimentado una respuesta al escalón con un sobrepaso máximo del 25%, que es un valor con buenas características de rapidez y estabilidad para la mayoría de los sistemas.

Existen dos métodos de sintonización:

#### Método I: Sintonización por la respuesta al escalón

Este método se adapta bien a los sistemas estables en lazo abierto y que presentan un tiempo de retardo desde que reciben la señal de control hasta comenzar a actuar.

De forma experimental, se obtiene la respuesta del sistema ante una entrada escalón unitario. La curva de respuesta al escalón debe obtenerse de forma experimental o a partir de una simulación. Como condición para el uso de éste primer método, la curva obtenida debe ser similar a la de la figura:

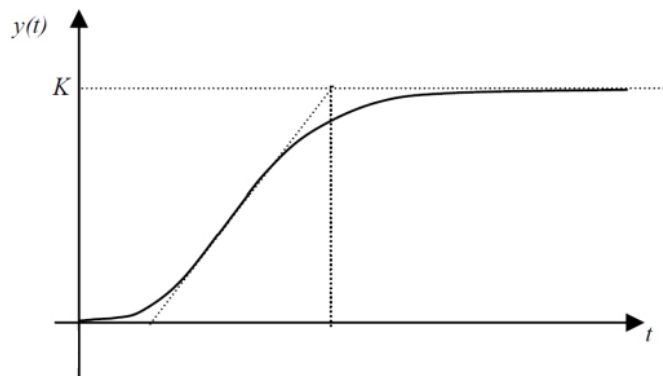


Figura 9: respuesta al escalón

La respuesta obtenida se caracteriza por el tiempo de retardo  $L$  y la constante de tiempo  $T$ . El tiempo de retardo y la constante de tiempo se determinan trazando la tangente a la curva en forma de S en el punto de inflexión y se determina su inserción con el eje de tiempo y con la línea  $c(t)=K$ , según se indica en la figura 8:

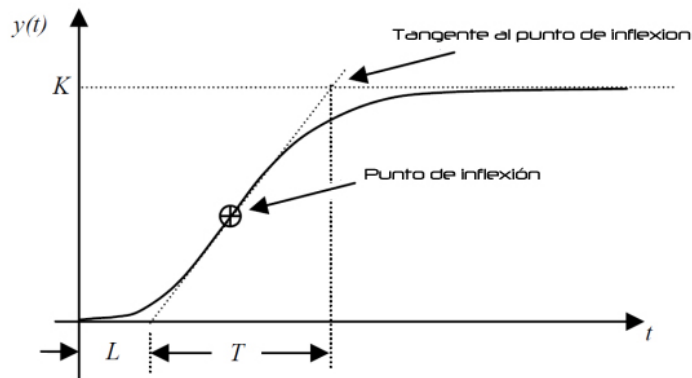


Figura 10: respuesta al escalón

El modelo del sistema sería:

$$\frac{C(s)}{U(s)} = \frac{Ke^{-s \cdot t_0}}{Ts + 1}$$

A partir de ésta información, se establecen los valores  $K_p$ ,  $T_i$  y  $T_d$ . Debido a que se han hecho los cálculos de forma empírica, estos valores siempre serán susceptibles de mejora por tanteo:

Tipo Controlador	$K_p$	$T_i$	$T_d$
P	$\frac{T}{KL}$	$\infty$	0
PI	$0.9 \frac{T}{KL}$	$\frac{T}{0.3}$	0
PID	$1.2 \frac{T}{KL}$	$2L$	$0.5L$

### Método I: Método de la ganancia en lazo cerrado

Este método no requiere retirar el controlador PID del lazo cerrado. Se trata de reducir al mínimo la acción derivativa y la acción integral ( $T_i = \infty$ ,  $T_d = 0$ ) del regulador PID. Usando sólo la acción del controlador proporcional, se incrementa  $K_p$  hasta que el sistema oscile de forma mantenida ante cualquier perturbación. Esta oscilación debe ser línea, sin saturaciones. Si la salida no presenta oscilaciones sostenidas para cualquier valor de  $K_p$ , no se aplica este método.

Teniendo en cuenta la ganancia proporcional crítica  $K_C$  y el periodo de oscilación  $T_C$  en segundos:



Tipo Controlador	$K_p$	$T_i$	$T_d$
P	$0.5K_C$	$\infty$	0
PI	$0.45K_C$	$\frac{1}{1.2}P_C$	0
PID	$0.6K_C$	$0.5P_C$	$0.125P_C$

## 2.3 CINEMÁTICA DEL ROBOT

### 2.3.1 Introducción

La cinemática, en el ámbito de la robótica, estudia el movimiento del mismo con respecto a un sistema de referencia, relacionando la posición y la orientación del extremo final del robot con los valores que toman sus coordenadas articulares.

Los dos problemas a los que hay que enfrentarse para resolver la cinemática del robot son:

- Cinemática directa: determinar la posición y orientación del extremo final del robot con respecto a un sistema de coordenadas, conocidos los valores de las articulaciones y los parámetros de los elementos geométricos del robot.
- Cinemática inversa: determinar los valores de las articulaciones para posicionar el extremo final del robot en un sistema de coordenadas conociendo los parámetros geométricos del robot.

Dadas las necesidades del robot y con las especificaciones del objetivo del proyecto, centraremos nuestra atención en la cinemática inversa, es decir, desplazar el robot a una posición XYZ sobre el plano, actuando para ello sobre los valores de las articulaciones.

### 2.3.2 Cinemática Inversa

La cinemática inversa trata de determinar los valores que deben adoptar las coordenadas articulares del robot para posicionar el extremo final del mismo en un sistema de coordenadas. Su cálculo supone la resolución de una serie de ecuaciones cuya solución no tiene por qué ser única.

A la hora de resolver el problema cinemático inverso, lo más adecuado es encontrar una relación matemática cerrada entre las posiciones  $x$ ,  $y$ ,  $z$ ,  $\alpha$ ,  $\beta$ ... y los valores articulares  $j$ ,  $k$ ,  $l$ ,  $m$ ...

Existen diferentes procedimientos genéricos para que, a partir del conocimiento de la cinemática del robot, se puedan obtener los diferentes valores articulares que posicionan el extremo final del mismo:

- Los métodos geométricos permiten obtener los valores articulares que consiguen posicionar el robot, prescindiendo de la orientación del extremo. Para ello utilizan relaciones trigonométricas y geométricas sobre las diferentes articulaciones del robot y el plano en que se encuentra. Pero existen limitaciones en este método cuando alguno de los últimos grados de libertad corresponde a giros sobre ejes que se cortan en un punto. En éste caso, debemos utilizar el siguiente método.
- La manipulación de las ecuaciones correspondientes al problema cinemático directo.

Para el caso del brazo robot Velleman KSR10, al no disponer de una mano giratoria y estar limitados sus últimos grados de libertad, utilizaremos métodos geométricos para el cálculo su posición.

### 2.3.3 Método geométrico de Cálculo de cinemática inversa

Partiendo de un brazo robot de dos grados de libertad, del cual conocemos la longitud de su brazo y antebrazo, podemos calcular el ángulo de los mismos para situarlo en una posición X,Y.

Más en concreto:

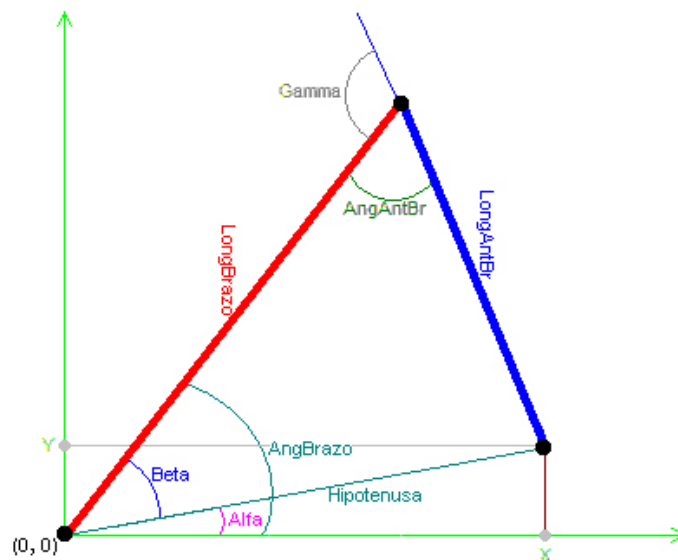


Figura 11: modelo geométrico para 2GL

$$Hipotenusa = \sqrt{x^2 + y^2}$$

$$Alfa = atan2(Y, X)$$

$$\text{Beta} = \text{Acos} \left( \frac{\text{longbrazo}^2 - \text{longantebrazo}^2 + \text{hipotenusa}^2}{2 \cdot \text{longbrazo} \cdot \text{hipotenusa}} \right)$$

$$\text{Angulo brazo} = \text{Alfa} + \text{Beta}$$

$$\text{Gamma} = \text{Acos} \left( \frac{\text{longbrazo}^2 + \text{longantebrazo}^2 - \text{hipotenusa}^2}{2 \cdot \text{longbrazo} \cdot \text{longantebrazo}} \right)$$

$$\text{Angulo antebrazo} = \text{Gamma} - 180$$

Obsérvese el uso de la función trigonométrica “atan2”. La diferencia entre “atan” y “atan2” es que, dependiendo de los signos de X e Y, se encontrará en uno de los cuatro cuadrante. Sin embargo, la función “atan” sólo se encontrará en los cuadrantes 1 y 4.

Partiendo de las premisas halladas para 2GL, el cálculo de la cinemática inversa para un robot de 5GL similar al brazo robot Velleman KSR10:

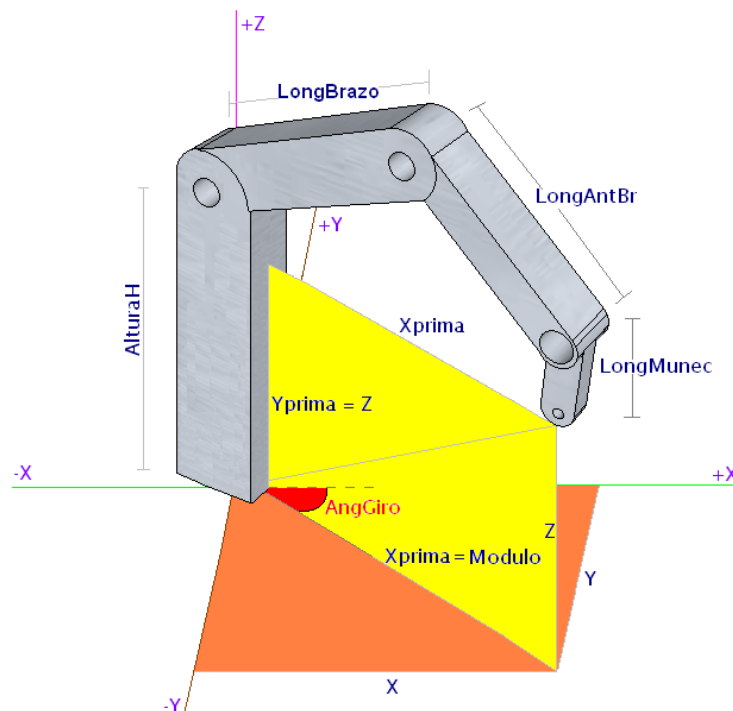


Figura 12: modelo geométrico para 5GL

Para llegar a una posición XYZ dentro del plano, y conociendo la longitud de las articulaciones, es posible calcular los valores articulares:

$$\text{Angulo giro} = \text{atan2}(Y, X)$$

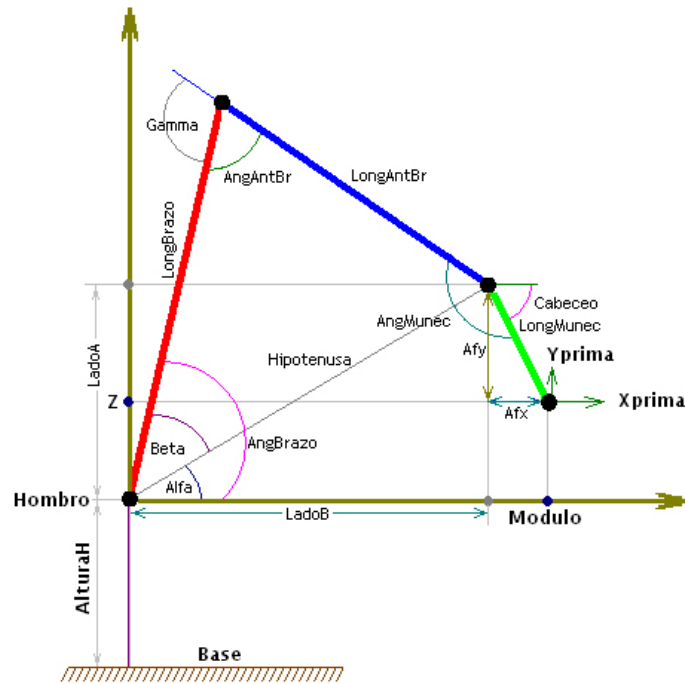


Figura 13: modelo geométrico para 5GL

$$\text{Modulo} = \sqrt{x^2 + y^2}$$

$$Yprima = Z$$

$$Afx = \cos(\text{cabeceo}) \cdot \text{LongMunec}$$

$$\text{LadoB} = Xprima - Afx$$

$$Afy = \sin(\text{cabeceo}) \cdot \text{LongMunec}$$

$$\text{LadoA} = Yprima - Afy - \text{AlturaH}$$

$$\text{Hipotenusa} = \sqrt{\text{ladoA}^2 + \text{ladoB}^2}$$

$$\text{Alfa} = \text{atan2}(\text{LadoA}, \text{LadoB})$$

$$\text{Beta} = \text{Acos} \left( \frac{\text{longbrazo}^2 - \text{longantbr}^2 + \text{hipotenusa}^2}{2 \cdot \text{longbrazo} \cdot \text{hipotenusa}} \right)$$

$$\text{Angulo brazo} = \text{Alfa} + \text{Beta}$$

$$\text{Gamma} = \text{Acos} \left( \frac{\text{longbrazo}^2 + \text{longantbr}^2 - \text{hipotenusa}^2}{2 \cdot \text{longbrazo} \cdot \text{longantbr}} \right)$$

$$\text{Angulo antebrazo} = -(180 - \text{Gamma})$$

$$\text{Angulo Muneca} = \text{Cabeceo} - \text{AngBrazo} - \text{AngAntbr}$$



## 2.4 PLANIFICACIÓN DE TRAYECTORIAS

### 2.4.1 Introducción

Una vez obtenidos los modelos cinemáticos o dinámicos del robot, se puede abordar el problema del control de los mismos. Definir el movimiento del robot implica controlar dicho robot de manera que siga un camino planificado. El objetivo es, por tanto, establecer cuales son las trayectorias que debe seguir cada articulación a lo largo del tiempo para conseguir los objetivos fijados, cumpliendo con una serie de restricciones físicas impuestas por los actuadores y de calidad de la trayectoria, suavidad, precisión, etc.

La realidad del problema de planificación de trayectorias exige, sin embargo, tener en consideración las prestaciones reales de los actuadores, de tal manera que el movimiento del robot sea suave y coordinado.

El cálculo de un sistema de planificación óptimo para el robot supone un estudio de las necesidades específicas del usuario, evitando colisiones con el entorno, etc.

### 2.4.2 Tipos de trayectorias

La mejora tecnológica ha permitido que los robots puedan realizar trayectorias cada vez más complejas. A continuación se citan algunos tipos de trayectorias:

#### **Trayectorias punto a punto**

En este tipo de trayectorias, cada articulación se mueve independientemente, sin considerar el efecto de las articulaciones. Dentro de este tipo se encuentran las trayectorias con movimientos eje a eje y las de movimiento simultáneo de los ejes.

#### **Trayectorias coordinadas**

En este tipo de trayectorias se pretende lograr un movimiento coordinado de todas las articulaciones. Esto quiere decir que la articulación más lenta en alcanzar su posición ralentiza al resto, de manera que ningún movimiento finaliza antes que el resto. El inconveniente de éste tipo de trayectorias es que la trayectoria del extremo final es desconocida a priori.

#### **Trayectorias continuas**

En este tipo de trayectorias se pretende que el camino seguido por el extremo del robot sea conocido. Para ello, cada articulación se mueve por separado describiendo una trayectoria prevista y manteniendo el extremo final del robot en un punto planificado previamente.

### 2.4.3 Algoritmo de Bresenham

El algoritmo de Bresenham fue desarrollado por Jack Elton Bresenham en 1962 para dibujar líneas rectas en dispositivos gráficos rasterizados, de forma que determina qué píxeles se rellenarán, en función de la inclinación del ángulo de la recta a dibujar.

Éste procedimiento puede ser aplicado como planificación de trayectorias en robótica, de tal modo que el extremo final del robot describa una trayectoria lineal entre dos puntos.

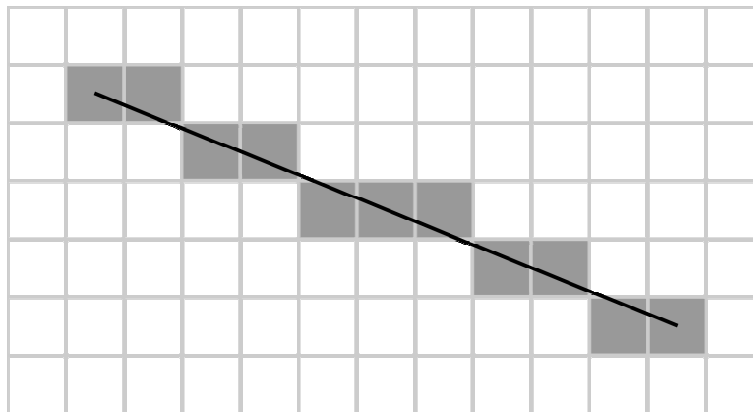


Figura 14: resolución algoritmo de Bresenham para 2D

Este algoritmo puede adaptarse para generar trayectorias curvadas, elipses o circunferencias.

### Algoritmo de Bresenham para 3 dimensiones XYZ

Es posible adaptar el algoritmo de Bresenham a una planificación de trayectorias en 3D con coordenadas XYZ de tal forma que el desplazamiento del robot de un punto  $X_1Y_1Z_1$  a otro  $X_2Y_2Z_2$  se lleve a cabo de forma líneas, interactuando entre sí las diferentes articulaciones.

El código fuente de simulación del algoritmo de Bresenham es el siguiente, donde  $X_{new}$ ,  $Y_{new}$  y  $Z_{new}$  corresponden con los valores finales deseados y  $X_{old}$ ,  $Y_{old}$  y  $Z_{old}$  con los valores de posición actuales.

```
Do
    xxx=Xold
    yyy=Yold
    zzz=Zold

    dx = xnew - Xold
    dy = ynew - Yold
    dz = znew - Zold

    If (dx < 0) Then
        x_inc = -1
```



```
Else
  x_inc = 1
EndIf

If (dy < 0) Then
  y_inc = -1
Else
  y_inc = 1
EndIf

If (dz < 0) Then
  z_inc = -1
Else
  z_inc = 1
EndIf

Adx = Abs(dx)
Ady = Abs(dy)
Adz = Abs(dz)

dx2 = Adx*2
dy2 = Ady*2
dz2 = Adz*2

If ((Adx>= Ady) And (Adx>= Adz)) Then

  err_1 = dy2 - Adx
  err_2 = dz2 - Adx

  For Cont = 0 To Adx-1

    If (err_1 > 0) Then
      yyy+= y_inc
      err_1 -= dx2
    EndIf

    If (err_2 > 0) Then
      zzz+= z_inc
      err_2 -= dx2
    EndIf

    err_1 += dy2
    err_2 += dz2
    xxx+= x_inc

    cinematica_inversa(xxx,yyy,zzz)

  Next

EndIf

If ((Ady> Adx) And (Ady>= Adz)) Then

  err_1 = dx2 - Ady
  err_2 = dz2 - Ady

  For Cont = 0 To Ady-1

    If (err_1 > 0) Then
      xxx+= x_inc
```



```
        err_1 -= dy2
    EndIf

    If (err_2 > 0) Then
        zzz+= z_inc
        err_2 -= dy2
    EndIf

    err_1 += dx2
    err_2 += dz2
    yyy+= y_inc

    cinematica_inversa(xxx,yyy,zzz)

Next

EndIf

If ((Adz> Adx) And (Adz> Ady)) Then

    err_1 = dy2 - Adz
    err_2 = dx2 - Adz

    For Cont = 0 To Adz-1

        If (err_1 > 0) Then
            yyy+= y_inc
            err_1 -= dz2
        EndIf

        If (err_2 > 0) Then
            xxx+= x_inc
            err_2 -= dz2
        EndIf

        err_1 += dy2
        err_2 += dx2
        zzz+= z_inc

        cinematica_inversa(xxx,yyy,zzz)

    Next

EndIf

Xold=xnew
Yold=ynew
Zold=znew

While (xxx<>xnew OR yyy<>ynew OR zzz <>znew)
```

#### **2.4.4 Planificación de trayectorias y cinemática inversa del robot**

Una vez desarrollado el algoritmo de planificación de trayectorias deseado, es preciso que interactúe de forma conjunta con la cinemática inversa del brazo robot, de tal forma que, conociendo un punto  $X_1Y_1Z_1$  del espacio y el  $X_2Y_2Z_2$  deseado, el algoritmo de planificación calculará los puntos  $X_{TMP}Y_{TMP}Z_{TMP}$  temporales que deberá recorrer el



extremo final del robot hasta llegar al punto final y la cinemática inversa transformará esos puntos en valores de las coordenadas articulares.

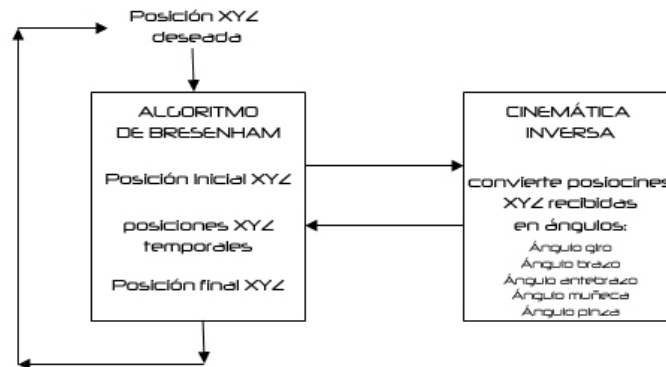


Figura 15: interacción algoritmo bresenham vs cinemática inversa





## 3. DESCRIPCIÓN DEL HARDWARE

### 3.1 ARDUINO

#### 3.1.1 Introducción

Arduino es una plataforma electrónica abierta para la creación de prototipos, basada en el software y hardware libre, flexible y fácil de implementar. Su filosofía de trabajo es DIY (Do It Yourself; hazlo tú mismo) y existen en el mercado multitud de sensores, controladores y placas para desarrollar proyectos de forma rápida y sencilla.

Nació en 2005 como un proyecto educativo y con la intención de crear un hardware libre basado en un microcontrolador y un entorno de desarrollo para facilitar el uso en la electrónica de proyectos multidisciplinarios: desde cubos led, sistemas de automatización de domótica, displays de twitter o analizadores de ADN.

Sus primeras 300 unidades fueron vendidas por 1 dólar a los alumnos del Instituto IVRAE, con el fin de que las probasen y diseñasen sus primeros prototipos.

A día de hoy se han vendido más de 300.000 placas en todo el mundo, sin contar con versiones clones y compatibles, y la comunidad de arduino es una de las más grandes y activas del mundo.

Algunas de las empresas tecnológicas más importantes, como Google, se han interesado en el proyecto Arduino y desarrollan ADK (Accessory Development Kit) para comunicarse directamente con smartphones android, accediendo a las funcionalidades del teléfono (GPS, GSM, acelerómetros) y viceversa.

El hardware de arduino consiste en una placa con un microprocesador Atmel y diferentes puertos de entrada y salida. Los microprocesadores más usados son el Atmega168, Atmega328, Atmega1280, Atmega8 que, por su sencillez y bajo coste, permiten el desarrollo de proyectos y prototipos a un coste relativamente bajo.

El entorno de desarrollo de Arduino está basado en Processing y este, a su vez, en java. El lenguaje de programación de Arduino está basado en Wiring, el cual proviene de C#.

La placa de Arduino puede ser alimentada a través del USB o de una fuente de alimentación externa, de tal forma que puede trabajar de modo independiente sin necesidad de un PC.

Las diferentes entradas pueden conectarse a una gran variedad de sensores existentes y puede interactuar con prácticamente cualquier sistema electrónico.

La comunicación via USB con el ordenador emula un puerto serie aunque también es posible la comunicación GSM, Wifi o Bluetooth. En algunos casos, estos módulos adicionales emulan un puerto serie y son conectados directamente a los puertos RX y TX de Arduino.

Según las especificaciones y necesidades del usuario, existen diferentes placas Arduino.

Para el presente proyecto, debido al número de entradas y salidas necesarias, utilizaremos Arduino MEGA 2560.

### 3.1.2 **Arduino Mega 2560**

Arduino Mega 2560 es una actualización de Arduino Mega, basado en el microcontrolador ATmega2560. Tiene 54 entradas/salidas digitales, de las cuales 14 proporcionan salidas PWM, 16 entradas digitales, 4 UARTS (puertos serie por hardware). El cristal oscilador es de 16MHz. Posee conexión USB, entrada externa de alimentación, conector ICSP y botón reset integrado (también posee pin reset).

Microcontrolador	<i>ATmega2560</i>
Voltaje funcionamiento	<i>5V</i>
Voltaje entrada recomendado	<i>7-12V</i>
Voltaje entrada límite	<i>6-20V</i>
Pines E/S digitales	<i>54 (14 son PWM)</i>
Pines entrada analógica	<i>16</i>
Intensidad por pin	<i>40mA</i>
Intensidad pin 3.3V	<i>50mA</i>
Memoria flash	<i>256KB (8KB usados para el arranque)</i>
SRAM	<i>8KB</i>
EEPROM	<i>4KB</i>
Velocidad reloj	<i>16MHz</i>



Figura 15: Placa de Arduino Mega 2560



## Alimentación

Arduino Mega puede ser alimentado via USB o mediante una fuente de alimentación externa AC/DC o baterías. La fuente de alimentación se selecciona de forma automática, de tal manera que no es necesaria ninguna configuración adicional.

Arduino posee un powerjack macho de 2.1mm. El rango de alimentación está entre 7 y 12V. Si el rango es superior a 12V, el regulador de tensión de Arduino podría dañarse. Si la tensión de alimentación es inferior a los 7V, el pin 5V de arduino podría proporcionar menos de 5V y producirse inestabilidad.

Los pines de alimentación integrados en la placa son:

- **VIN:** es la entrada de alimentación de arduino, equivalente a la conexión del powerjack 2.1mm.
- **5V:** los 5v proporcionados provienen del voltaje estabilizado por el regulador de tensión integrado y es el mismo que se utiliza para alimentar el microcontrolador.
- **3.3V:** una fuente de voltaje de 3.3v generada en el chip FTDI de la placa. La corriente máxima aportada son 50mA.
- **GND:** pines de tierra.

## Memoria

El ATMEGA2560 posee 256KB de memoria flash, de los cuales 8KB son utilizados para el proceso de arranque, 8KB de SRAM y 4KB de EEPROM. La memoria SRAM es volátil y la EEPROM no volátil.

## Entradas y Salidas

Cada uno de los 54 pines digitales de Arduino MEGA pueden utilizarse como entradas o salidas. Todas ellas operan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40mA y tiene una resistencia interna (desconectada por defecto) de 20-50KΩ.

- **Pines para transmisión serie RX/TX:** puede transmitir y recibir información a través de los puertos serie TTL de Arduino. Los pines serie 0 (RX) y 1(TX) están conectados a los pines correspondientes del chip FTDI USB-to-TTL, de tal modo que el puerto USB se comporta como serie a través de éstos pines.

El resto de pines serie son:

Serie: 0 (RX), 1 (TX);

Serie1: 19 (RX), 18 (TX);

Serie2: 17 (RX), 16 (TX);

Serie3: 15 (RX), 14 (TX);



- **Interrupciones externas:** algunos pines se pueden configurar para lanzar una interrupción en valor LOW (0v), en flancos de subida o bajada o en cambios de valor:
  - Interrupción 0: 2.
  - Interrupción 1: 3.
  - Interrupción 2: 21.
  - Interrupción 3: 20.
  - Interrupción 5: 18.
  - Interrupción 4: 19.
- **PWM:** pines 0 a 13: algunos pines pueden proporcionar una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución para obtener valores de 0 a 255 mediante la función `analogwrite()`.
- **LED 13:** en la placa Arduino se ha integrado un led conectado al pin digital 13. Cuando éste tiene un valor alto, se enciende dicho LED.
- **Entradas Analógicas:** Arduino MEGA posee 16 entradas analógicas; cada una de éstas entradas proporciona una resolución de 10 bits (1024 valores). Por defecto se mide de tierra a +5V aunque es posible cambiar la cota usando el pin AREF y la función `analogreference()`.
- **Reset:** suministrando un valor de 0V se puede reiniciar el microcontrolador.

## Comunicaciones

A pesar de que existen 4 pares de pines destinados a la comunicación serie, es posible utilizar la librería `SoftwareSerial` de Arduino para implementar un puerto serie a través de cualquier otro puerto.

La comunicación con el PC a través del puerto USB se canaliza a través del chip FTDI 232RL USB-to-TTL. Arduino posee leds RX y TX que parpadean cuando se detecta comunicación a través del chip FTDI y la conexión USB. En caso de utilizar los pines 0 y 1 de forma independiente, dichos leds no parpadearán.

## Protección contra sobretensiones en USB

Arduino MEGA posee un multifusible reinicializable que protege la conexión USB del ordenador de posibles cortocircuitos y sobretensiones en caso de detectar más de 500mA en el puerto USB.

Para nuestro proyecto utilizaremos la conexión serie de los pines 0 y 1 a través del puerto USB del PC.

Arduino y el resto de circuitería lógica quedará alimentado por la tensión proporcionada por el puerto USB, mientras que los motores serán alimentados con una fuente de alimentación externa.



La placa diseñada posee un regulador de tensión que podría ser utilizado en caso de que se desee suprimir el cable USB y conectar el PC a Arduino mediante bluetooth. El puerto Bluetooth quedaría alimentado a través de los 5V de la placa y conectado al puerto de transmisión 0 y 1, de tal forma que el cambio sea transparente tanto para el software de Arduino como .NET.

### **3.1.3 Pulse Width Modulation (PWM)**

La modulación por ancho de pulsos o PWM de una señal es una técnica que modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

El ciclo de trabajo de una señal periódica es el ancho relativo a su parte positiva en relación con su periodo. Matemáticamente puede expresarse como:

$$D = \frac{\tau}{T}$$

Donde D es el ciclo de trabajo,  $\tau$  es el tiempo en que la función es positiva (ancho del pulso) y T el periodo de la función.

La modulación por ancho de pulsos es una técnica utilizada en diversos campos: desde la regulación de velocidad de giro de motores eléctricos, a reguladores de tensión o convertidores ADC que permiten comunicarse de forma analógica con sistemas digitales, generando señales analógicas a partir de digitales.

La modulación PWM en Arduino nos permite variar la tensión de salida de los pines 0 a 13 en un valor proporcional a la escala de 0 a 255, correspondiendo 0 con 0V y 255 con 5V. Esto se consigue creando una onda cuadrada que conmuta constantemente entre 0 y 5V para generar una tensión proporcional a la escala.

Para generar ésta señal se utiliza la instrucción `analogwrite()` de Arduino, tal y como se puede comprobar en la siguiente imagen:

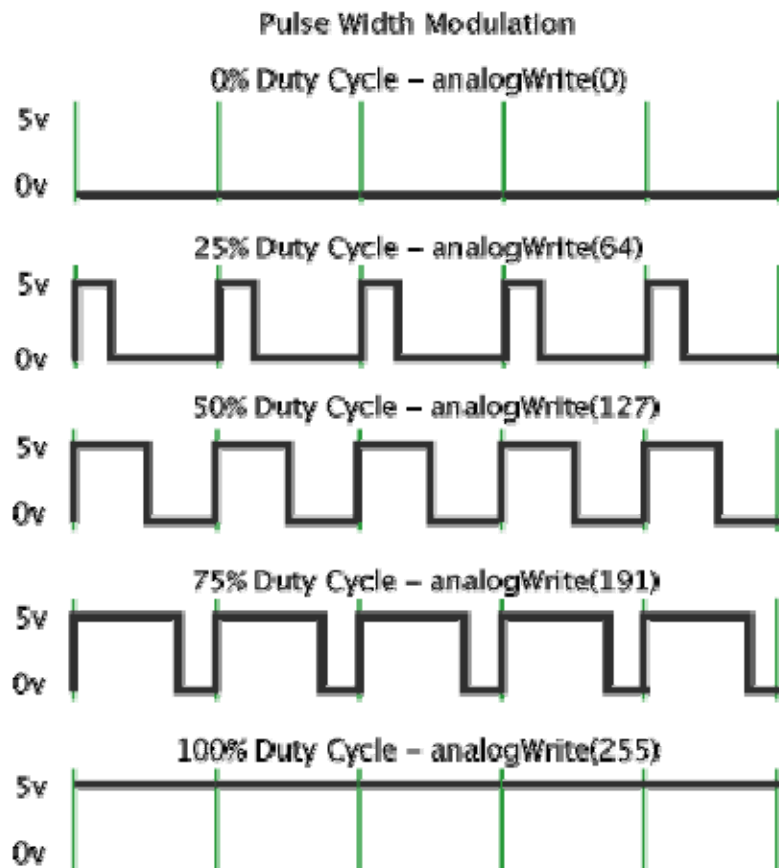


Figura 16: Modulación de pulsos en Arduino

### 3.1.4 Licencia

Es posible fabricar productos comerciales basados en Arduino, con las siguientes condiciones:

- Utilizar una placa Arduino dentro de un producto comercial no requiere que se revele ninguna parte de su diseño.
- Derivar el diseño de un producto comercial a partir de los ficheros Eagle de la placa Arduino requiere que los ficheros modificados sean publicados bajo la misma licencia Creative Commons Attribution Share-Alike. No obstante, el producto resultante puede ser comercializado.
- Utilizar el núcleo y las bibliotecas de Arduino para el firmware de un producto comercial no requiere la publicación del código fuente. La LGPL requiere que se liberen los ficheros objeto que permiten el re-enlace al firmware para versiones actualizadas de las bibliotecas y núcleo de Arduino. Cualquier modificación realizada debe ser publicada bajo la LGPL. La LGPL es la Licencia Pública General para Bibliotecas de GNU y pretende garantizar la libertad de compartir y modificar el software cubierto por ella, asegurando que el mismo es libre para todos los usuarios.
- El código fuente del entorno IDE de Arduino está cubierto bajo la GPL, que requiere que cualquier modificación sea de código libre bajo la misma licencia.



## 3.2 BRAZO ROBOT VELLEMAN KSR10

### 3.2.1 Introducción

El robot Velleman KSR10 es un brazo de configuración circular con cinco motores y cinco articulaciones, por tanto, 5 grados de libertad. El brazo es manipulado por una unidad de control y está equipado con una base giratoria, codo y muñeca móviles y una mano funcional.



Figura 17: Brazo robot Velleman KSR10 y su unidad de control remoto

Los cinco motores que posee el robot Velleman son de corriente continua y soportan una tensión de alimentación máxima de 6V. Acoplado a cada motor se encuentra un condensador cerámico de  $0.1\mu\text{F}$  que protege a los mismos ante posibles picos de tensión.



Figura 18: Motor DC del Brazo Robot Velleman KSR10

Existen ampliaciones de control para el robot Velleman KSR10, como un módulo de control a través del PC mediante USB, el cual está limitado por la falta de sensores de posición que controlen la posición, limitando su uso al control manual via PC o a la programación de accionamiento de los motores utilizando como parámetro de control tiempo en segundos.

### 3.2.2 Especificaciones

La capacidad elevadora máxima del robot son 100g. Está alimentado por 4 pilas tipo LR20C, que suministran un total de 6V. La altura máxima del brazo extendido son 38cm y sus dimensiones 22.85cm de largo x 16cm de ancho. El peso 660g.

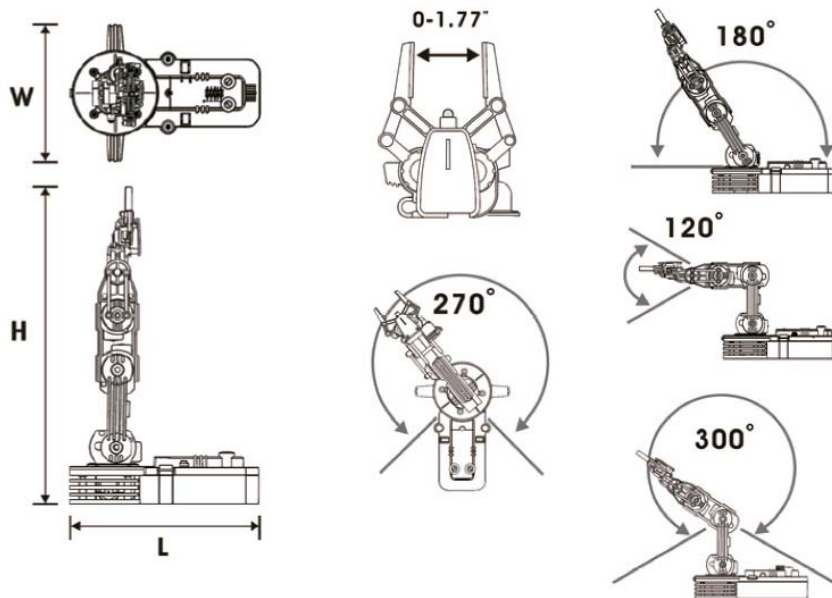


Figura 19: especificaciones Brazo robot Velleman KSR10

## 3.3 PUENTE EN H

### 3.3.1 Introducción

Un puente en H es un circuito electrónico que permite a un motor eléctrico DC girar en ambos sentidos, avance y retroceso. Los puentes en H están disponibles como circuitos electrónicos, como el L293D que utilizaremos en el proyecto, aunque también pueden construirse a partir de componentes discretos.

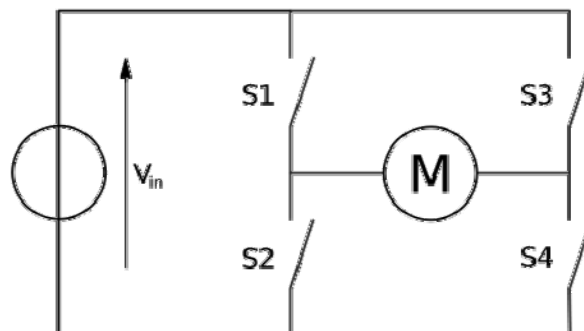


Figura 20: puente en H

El término “puente en H” proviene de la típica representación gráfica del circuito: está construido con 4 interruptores mecánicos o transistores. Cuando los interruptores S1 y S4 están cerrados, se aplica tensión positiva en el motor, haciéndolo girar en un sentido. Abriendo los interruptores S1 y S4 y cerrando S2 y S3, el voltaje se invierte, permitiendo el giro en sentido inverso del motor.

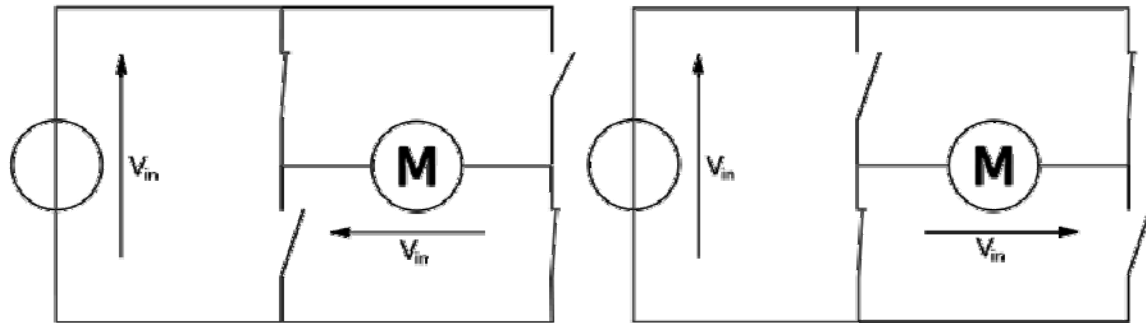


Figura 21: tensiones en puente en H

### 3.3.2 Puente en H L293D

El L293D es un puente en H cuádruple integrado. Está diseñado para proporcionar hasta 1000mA en dos canales con corrientes entre 4.5V y 36V y permite accionar dos motores DC de forma simultánea en cualquiera de sus sentidos.

El L293D posee un total de 16 pines:

- 1- 1,2 Enable: se utilizan para habilitar las salidas 1 y 2, es decir el motor 1, conectado entre Output1 y Output2. La tensión de activación o desactivación será Vcc o Gnd.
- 2- Input 1: controla la dirección de salida del Motor 1, conectado entre Output1 y Output 2.
- 3- Output 1: pin de conexión del motor 1.
- 4- GND: conectado a tierra.
- 5- GND: conectado a tierra.
- 6- Output 2: pin de conexión del motor 1.
- 7- Input 2: controla la dirección de salida del Motor 1, conectado entre Output1 y Output 2.
- 8- Vcc2: tensión de alimentación de los motores.
- 9- 3,4 Enable: se utilizan para habilitar las salidas 3 y 4, es decir el motor 2, conectado entre Output1 y Output2. La tensión de activación o desactivación será Vcc o Gnd.
- 10- Input 3: controla la dirección de salida del Motor 2, conectado entre Output3 y Output 4.
- 11- Output 3: pin de conexión del motor 2.
- 12- GND: conectado a tierra.
- 13- GND: conectado a tierra.
- 14- Output 4: pin de conexión del motor 2.

- 15- Input 4: controla la dirección de salida del Motor 2, conectado entre Output3 y Output 4.
- 16- Vcc1: tensión de alimentación entre 4.5 y 7V.

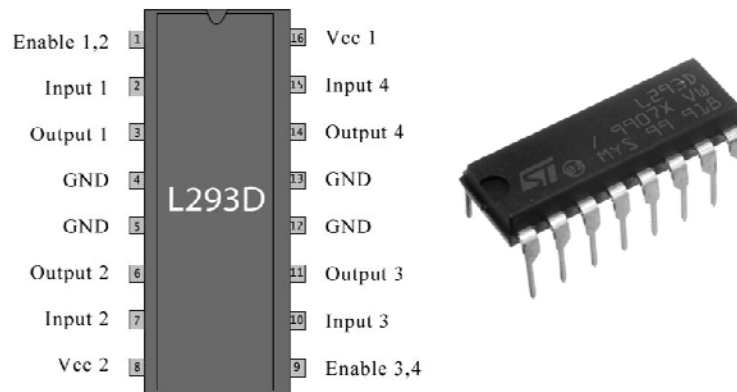


Figura 22: encapsulado L293D y diagrama de pines

Variando la tensión de Enable del motor 1 o motor 2 con respecto a la tensión de alimentación Vcc1, es posible variar la velocidad del motor, sin necesidad de variar la tensión de alimentación de los mismos Vcc2. Éste punto es clave a la hora de aplicar un controlador PID a cada uno de los motores del sistema.

Para controlar el brazo robot Velleman KSR10 debemos utilizar tres L293D con la siguiente configuración:

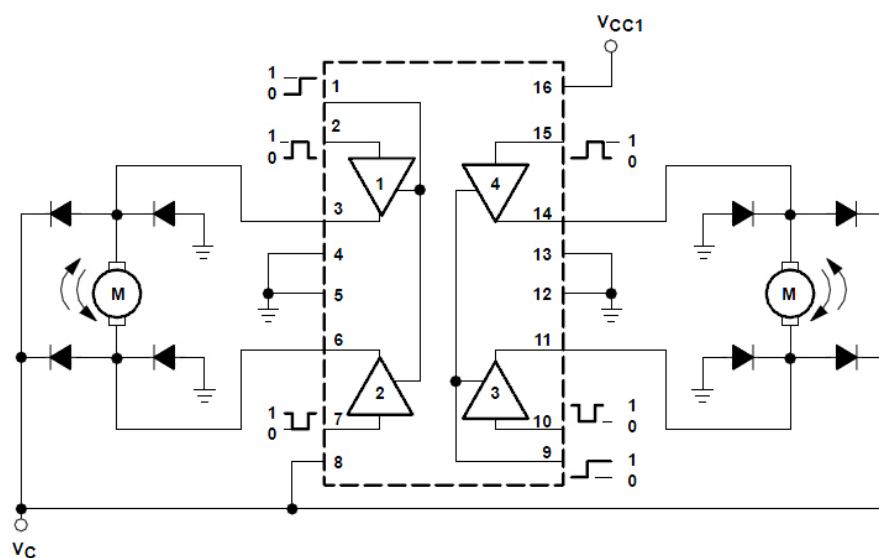


Figura 23: conexiones L293D

### 3.4 REGULADOR DE TENSIÓN LM7805

El regulador de tensión LM7805 es un circuito electrónico capaz de generar una alimentación regulada de 5V y 1000mA en relación a GND, a partir de una tensión de entrada de hasta 35V.

Posee tres terminales, y suele encontrarse en el encapsulado TO220.



Figura 24: encapsulado LM7805

Éste regulador será incluido en la placa del controlador de nuestro proyecto para futuras actualizaciones en las que no se utilice el puerto usb como medio de comunicación/alimentación.

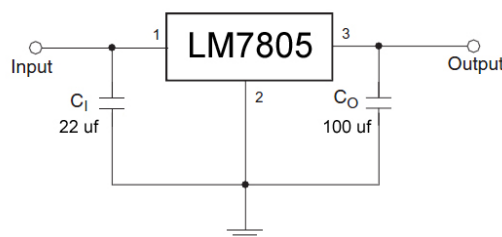


Figura 25: conexiones LM7805

### 3.5 RESISTENCIAS VARIABLES

#### 3.5.1 Introducción

Un potenciómetro o resistencia variable es un resistor cuyo valor de resistencia es variable. De esta manera, indirectamente, se puede controlar la intensidad de corriente que fluye por un circuito o la diferencia de potencial al conectarlo en serie.

Normalmente, los potenciómetros se utilizan en circuitos de poca corriente. Para circuitos de corrientes mayores, se utilizan los reóstatos, que pueden disipar mayor potencia.

#### 3.5.2 Tipos de resistencias variables

Según la relación variación angular / variación resistiva, existen diferentes tipos de potenciómetros:

- **Lineales:** la resistencia es proporcional al ángulo de giro.
- **Logarítmicos:** la resistencia depende logarítmicamente del ángulo de giro.
- **Senoidales:** la resistencia es proporcional al seno del ángulo de giro.

### 3.5.3 Resistencia variable lineal CA9

La resistencia variable CA9 es una resistencia rotatoria lineal cuyo encapsulado se adapta a las necesidades de control de los ejes del brazo robot Velleman KSR10.



Figura 26: CA9 con encapsulado CA9 H2.5 y rotor STD

Existen diversos encapsulador y formas del rotor para éste tipo de resistencia. Basándonos en las necesidades de ubicación de los sensores sobre el brazo robot, se escoge el encapsulado CA9 H2.5 por su reducido tamaño y ubicación de los pines así como el rotor STD, que se adaptará más adecuadamente a las varillas instaladas.

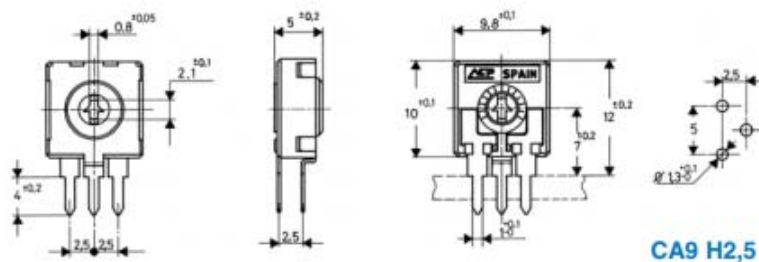


Figura 27: CA9 con encapsulado CA9 H2.5 y rotor STD

## 3.6 USB

El bus serie universal (USB) es un estándar industrial desarrollado en los años 90 que define cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica ordenadores y periféricos.

Su campo de aplicación actual se extiende en la actualidad a cualquier dispositivo electrónico o con componentes. Se calcula que, desde 2004, aproximadamente 6 mil millones de dispositivos disponen de conexión usb.

El puerto USB es el que se utilizará en nuestro proyecto para comunicar, mediante puerto serie, Arduino y el Software de control desarrollado.

## 4. DESCRIPCIÓN DEL SOFTWARE

### 4.1 ARDUINO

#### 4.1.1 Entorno de Desarrollo para Arduino

El entorno de Desarrollo Arduino hace fácil escribir código y cargarlo a la placa. Se denomina IDE (Integrated Development Environment) y está escrito en plataforma java y basado en Processing, avr-gcc y otros programas de código abierto.

Es posible descargar el IDE de Arduino para las diversas plataformas del mercado: Windows, MacOSX o Linux, si bien es cierto que, en caso de intercambiar de equipo durante el uso de nuestra placa Arduino, conviene revisar las diferencias en su funcionamiento.

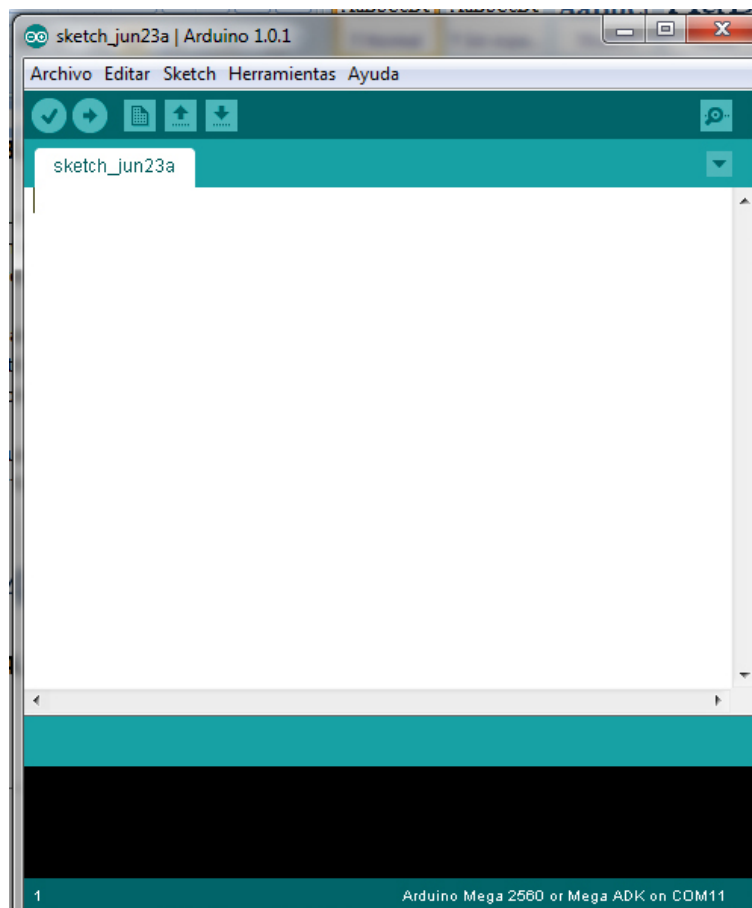









Figura 28: Entorno IDE de programación de Arduino

El entorno está constituido por un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas y un menú. Este software permite la conexión directa con el hardware de Arduino para cargar los programas e incluso es posible comunicarse con ellos a través de la consola serie.



Los programas de Arduino se denominan “sketch”. En el área de mensajes se muestra información mientras se cargan los programas y los errores de depurado del código. La consola muestra el texto de salida para el entorno Arduino. La barra de herramientas permite verificar el proceso de carga, creación, apertura, guardado de programas, etc.

Algunos de los controles básicos del IDE de Arduino:

-  NUEVO: crea un nuevo sketch.
-  ABRIR: permite abrir un sketch almacenado previamente.
-  GRABAR: permite grabar un sketch en una ruta seleccionada.
-  CARGAR EN ARDUINO: compila el programa y lo carga en Arduino.
-  COMPILAR: verifica el código en búsqueda de errores.
-  STOP: finaliza la monitorización serie.
-  SERIE: inicia la consola de monitorización serie.

En el menú encontramos las acciones más comunes tales como copiar, pegar, abrir, etc. En el submenú “Tools” se encuentran algunas acciones destacables:

- Board: es preciso seleccionar la placa Arduino a utilizar.
- Serial Port: selección de los dispositivos serie (reales o virtuales) del equipo. Es preciso seleccionar el puerto COM al que ha sido conectado Arduino.

Cabe destacar que, en caso de conectar un módulo de comunicación, por ejemplo bluetooth, a los puertos 0 y 1 de Arduino, será preciso que sea desconectado previamente a la carga de una nueva versión del software.

Adjuntas al anexo se encuentran las instrucciones de instalación del IDE de arduino, las librerías externas y la configuración de la tarjeta Arduino en Windows.

#### **4.1.2 Lenguaje de programación de Arduino**

El lenguaje de programación de Arduino es meramente un grupo de funciones de C/C++. Todas las instrucciones de C/C++ soportadas por el compilador avr-g++ funcionan con Arduino.

#### **4.1.3 Estructura de un programa Arduino**

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone al menos de dos partes bien diferenciadas. Estas dos partes o funciones necesarias, encierran bloques que contienen declaraciones, estamentos o instrucciones:





```
Void setup(){  
  
}  
Void loop(){  
  
}
```

Setup() es la parte encargada de recoger la configuración y Loop() es la que contiene el programa que se ejecutará cíclicamente. Ambas son estrictamente necesarias.

La función de configuración debe contener la declaración de variables, el modo de trabajo de las E/S, configuración de comunicación serie y cualquier otro tipo de información relativa a la configuración. Se invoca únicamente al inicio de la carga de Arduino y antes de la función loop.

La función loop contiene el código que se ejecutará continuamente. Este es el núcleo de todos los programas Arduino y quien realiza la mayor parte del trabajo.

Es preciso tener en cuenta que la función loop se ejecuta continuamente salvo el caso en que se programe una lectura del puerto serie; en éste último caso, la ejecución esperará en cada ciclo a la entrada de datos a través de dicho puerto a no ser que se realicen las modificaciones de código oportunas.

#### **4.1.4 Comunicación Serie a través de Arduino**

Arduino puede transmitir y recibir información a través de los puertos serie TTL. Los pines serie 0 (RX) y 1(TX) están conectados a los pines correspondientes del chip FTDI USB-to-TTL, de tal modo que el puerto USB se comporta como serie a través de éstos pines. El resto de pines serie son:

```
Serie: 0 (RX), 1 (TX);  
Serie1: 19 (RX), 18 (TX);  
Serie2: 17 (RX), 16 (TX);  
Serie3: 15 (RX), 14 (TX);
```

Es importante tener en cuenta que para enviar un nuevo código de programación desde IDE a Arduino no puede haber ningún dispositivo de comunicaciones conectado a los pines Serie 0 (RX) y 1 (TX). En caso contrario, la carga fallará.

En nuestro caso, al utilizar el puerto USB para comunicar la aplicación .NET con Arduino, no es posible conectar dispositivos a los pines 0 y 1. En caso de utilizar un puerto bluetooth, de tal modo como se especifica en puntos posteriores, será conectado a los puertos 0 (RX) y 1 (TX), debiendo ser desconectado para futuras actualizaciones del software controlador del robot.



### Particularidades de la comunicación serie con Arduino

En Arduino no existe el tipo de datos String, de tal forma que no es posible recibir una cadena de caracteres y ser asignada a una variable. Para ello, es preciso declarar un vector de caracteres con un puntero que indicará al algoritmo cual es la siguiente posición a insertar. Idealmente puede diseñarse de forma cíclica, de tal forma que, al llegar a la última posición, comience a escribir de nuevo en la posición 0 del mismo.

En nuestro caso, comunicaremos una aplicación .NET con Arduino a través del puerto Serie. Teniendo en cuenta ésta particularidad, y como se explicará en apartados posteriores, se ha desarrollado un método de comunicación y transmisión de grandes volúmenes de información a través de dicho puerto.

El formato de envío de información a través de puerto serie es ASCII; no obstante, es posible definir otro tipo de codificaciones de envío. Para nuestro caso, la transmisión ASCII es válida, teniendo en cuenta únicamente que en el método implementado para la comunicación entre la aplicación .NET y Arduino no se podrán utilizar caracteres especiales.

Caracteres ASCII de control			Caracteres ASCII imprimibles				ASCII extendido (Página de código 437)									
00	NULL	(carácter nulo)	32	espacio	64	@	96	`	128	Ç	160	á	192	Ł	224	Ó
01	SOH	(inicio encabezado)	33	!	65	A	97	a	129	ú	161	í	193	ł	225	ó
02	STX	(inicio texto)	34	"	66	B	98	b	130	é	162	ó	194	ł	226	ô
03	ETX	(fin de texto)	35	#	67	C	99	c	131	â	163	ú	195	ł	227	ö
04	EOT	(fin transmisión)	36	\$	68	D	100	d	132	ã	164	ñ	196	—	228	ø
05	ENQ	(consulta)	37	%	69	E	101	e	133	ä	165	Ñ	197	ł	229	õ
06	ACK	(reconocimiento)	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
07	BEL	(timbre)	39	'	71	G	103	g	135	ç	167	º	199	Ä	231	þ
08	BS	(retroceso)	40	(	72	H	104	h	136	ê	168	¿	200	Ł	232	ƒ
09	HT	(tab horizontal)	41	)	73	I	105	i	137	ë	169	©	201	ł	233	ú
10	LF	(nueva línea)	42	*	74	J	106	j	138	è	170	¬	202	ł	234	Û
11	VT	(tab vertical)	43	+	75	K	107	k	139	í	171	½	203	ł	235	Ü
12	FF	(nueva página)	44	,	76	L	108	l	140	î	172	¾	204	ł	236	ý
13	CR	(retorno de carro)	45	-	77	M	109	m	141	ï	173	ı	205	=	237	ÿ
14	SO	(desplaza afuera)	46	.	78	N	110	n	142	Ā	174	«	206	ł	238	—
15	SI	(desplaza adentro)	47	/	79	O	111	o	143	Ā	175	»	207	ł	239	·
16	DLE	(esc.vínculo datos)	48	0	80	P	112	p	144	É	176	⋮	208	ø	240	≡
17	DC1	(control disp. 1)	49	1	81	Q	113	q	145	æ	177	⋮	209	ø	241	±
18	DC2	(control disp. 2)	50	2	82	R	114	r	146	Æ	178	⋮	210	È	242	—
19	DC3	(control disp. 3)	51	3	83	S	115	s	147	ó	179		211	É	243	¾
20	DC4	(control disp. 4)	52	4	84	T	116	t	148	ö	180	ı	212	È	244	Œ
21	NAK	(conf. negativa)	53	5	85	U	117	u	149	õ	181	Ā	213	ı	245	Œ
22	SYN	(inactividad sinc)	54	6	86	V	118	v	150	ù	182	Ā	214	ı	246	÷
23	ETB	(fin bloque trans)	55	7	87	W	119	w	151	û	183	Ā	215	ı	247	°
24	CAN	(cancelar)	56	8	88	X	120	x	152	ÿ	184	©	216	ı	248	°
25	EM	(fin del medio)	57	9	89	Y	121	y	153	Œ	185	ł	217	ı	249	°
26	SUB	(sustitución)	58	:	90	Z	122	z	154	Ū	186	ł	218	ı	250	·
27	ESC	(escape)	59	;	91	[	123	{	155	ø	187	ł	219	ı	251	·
28	FS	(sep. archivos)	60	<	92	\	124		156	€	188	ł	220	ı	252	·
29	GS	(sep. grupos)	61	=	93	]	125	}	157	ø	189	ł	221	ı	253	·
30	RS	(sep. registros)	62	>	94	^	126	~	158	×	190	¥	222	ı	254	·
31	US	(sep. unidades)	63	?	95	_			159	f	191	ł	223	ı	255	nbsp
127	DEL	(suprimir)														

Figura 29: Tabla de caracteres ASCII

### Código fuente básico de inicialización de la comunicación serie

Un ejemplo básico de sketch de Arduino para la transmisión/recepción serie es el siguiente:



```
Int input;  
void setup(){  
    Serial.begin(9600);  
}  
void loop(){  
    If(Serial.available()>0{  
        Input=Serial.read();  
        Serial.println("Valor recibido: "+String(Input));  
    }  
}
```

En la función Setup() se inicializa el puerto Serie especificando los baudios de transmisión, y que corresponderán con el mismo baudrate en el otro punto comunicador.

En la función loop() se lee un dato del puerto serie y se envía de nuevo junto con una cadena de caracteres.

Arduino puede enviar largas cadenas de caracteres sin error. Sin embargo, vamos a afrontar el problema de la recepción de cadenas de caracteres. Para ello, hemos desarrollado la siguiente función:

```
//se define el tamaño máximo de caracteres susceptibles de  
//ser recibidos  
#define BUFFSIZ 250  
  
char buffer[BUFFSIZ];  
  
void readString() {  
    char c;  
    int buffSize = 0;  
    int buc=0;  
    //inicialización del buffer de almacenamiento con el valor 0  
    memset(buffer, 0, sizeof(buffer));  
    //vaciado del buffer de lectura serial de Arduino  
    Serial.flush();  
    do{  
        while (Serial.available()>0){  
            buffer[buffSize]=Serial.read();  
            //Utilizamos como fin de cadena el caracter #  
            if(buffer[buffSize]=='#'){control_prog=0;break;}  
            buffSize++;  
        }  
    }while(control_prog==1);  
    for(buc=0;buc<=buffSize;buc++){  
        Serial.print(buffer[buc]);  
    }  
    control_prog=0;  
}
```

#### 4.1.5 Librería PID Arduino

En el apartado 2.2 se describen los fundamentos teóricos de los controladores PID. En éste punto se va a abordar la aplicación de un PID en un sistema real, utilizando un microcontrolador programado en lenguaje de alto nivel como es Arduino.

El control proporcional consiste en el producto de la señal de error y la constante proporcional, siendo casi nulo en estado estacionario.

El control integral tiene como propósito disminuir y eliminar el error estacionario provocado por el modo proporcional.

El control derivativo considera la tendencia del error y permite una reacción rápida después de presentarse una perturbación en el proceso.

No se va a profundizar en el código fuente de la librería PID de Arduino, pero sí en su uso y sintonización de los parámetros de ajuste.

La librería PID puede ser descargada a través de la página oficial de Arduino, apuntando a la url: <https://code.google.com/p/arduino-pid-library/downloads/list?can=1&q=>

Un ejemplo básico de utilización de la misma es el siguiente:

```
#include <PID_v1.h>
double Setpoint, Input, Output;

double aggKp1=6, aggKi1=0.1, aggKd1=0;

PID myPID(&Input, &Output, &Setpoint,aggKp1,aggKi1,aggKd1,
DIRECT);

void setup()
{
  Input = analogRead(0);
  Setpoint = 100;
  myPID.SetMode(AUTOMATIC);
}

void loop()
{
  Input = analogRead(0);
  myPID.Compute();
  analogWrite(3,Output);
}
```

A la hora de adaptar el PDI al caso particular de nuestro control, es preciso el uso de algunas otras funciones de la librería PID:

**SetControllerDirection(param):** donde param corresponde con "DIRECT" o "REVERSE", según el sentido rotacional del motor en cada caso.



SetOutputLimits(min,max): donde min y max corresponden con los valores máximo y mínimo de salida del PID. Esto puede producir un efecto lineal en la salida ante algunas variaciones en el setpoint pero ayuda a regular la tensión de alimentación, y por tanto, la velocidad de los motores del brazo robot.

Una particularidad aplicable es la variación de los parámetros de sintonización teniendo en cuenta el valor actual de lectura del sensor en relación al setpoint deseado. Por ejemplo:

```
double aggKp1=6, aggKil=0.1, aggKd1=0;
double consKp1=0.2, consKil=0.05, consKd1=0;

Input1 = analogRead(id_sensor);
//distancia entre el punto actual y el definitivo
double gap = abs(Setpoint-Input1);
//si estamos cerca del punto, reducimos parametro PID
if(gap<reduccion){
    myPID.SetTunings(consKp1, consKil, consKd1);
}else{
    //si estamos alejados, parametros mas agresivos
    myPID.SetTunings(aggKp1, aggKil, aggKd1);
}
```

De este modo, y calculando teórica o empíricamente un valor límite de reducción a partir del cual cambiar los parámetros de sintonización del PID, es posible variar la sintonización, de tal forma que el robot se comporte de una manera más natural.

Éste es el caso del robot Velleman KSR10. Debido a la limitación de sus motores, que únicamente actuarán a partir de una tensión mínima que puede provocar movimientos bruscos y a la inercia en algunos otros movimientos, unas articulaciones tendrán configurados parámetros más agresivos.

## 4.2 VISUAL STUDIO 2012

### 4.2.1 Introducción

Microsoft Visual Studio es un entorno de desarrollo para sistemas operativos Windows. Soporta varios lenguajes de programación tales como C#, C++, ASP.NET o Visual Basic .NET.

El software controlador del robot Velleman KSR10 ha sido desarrollado a partir de ésta aplicación en su última versión 2012 y en lenguaje Visual Basic .NET

Visual Studio 2012 funciona únicamente en sistemas Windows, más en concreto para las versiones 7 SP1 (x86 y x64), 8 (x86 y x64), Server 2008 R2 SP1 (x64) y Server 2012 (x64).

Adjuntas al anexo se encuentran las instrucciones de instalación de Visual Studio 2012.

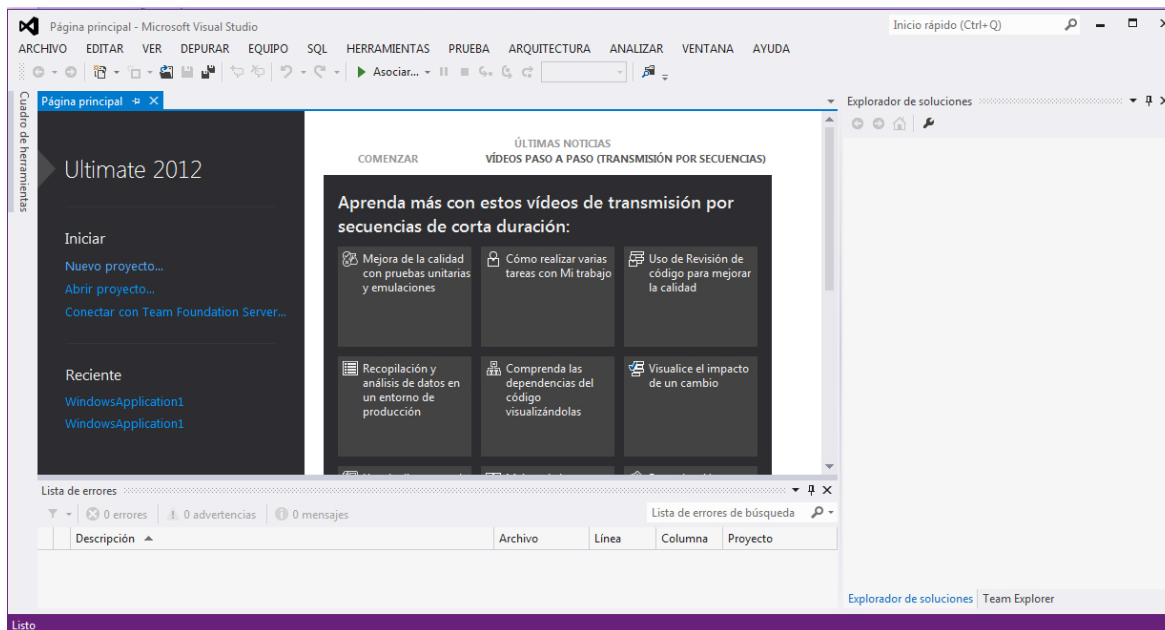


Figura 30: Entorno de desarrollo de .NET

#### 4.2.2 Versiones de Visual Studio 2012 y alternativas

Existen varias versiones de Visual Studio 2012 según las necesidades del usuario final, desde la versión Ultimate, la más completa, a la versión reducida Express.

La versión Express está desarrollada para el ámbito de la formación, es gratuita aunque limitada. Sin embargo, las limitaciones de dicha versión no suponen un problema para el software desarrollado, puesto que todas las funcionalidades utilizadas son de libre acceso.

Existen otras alternativas a la programación en .NET a través de Visual Studio 2012, entre las que cabe destacar, por ejemplo, SharpDevelop o MonoDevelop.

Sin embargo, para el desarrollo del software de este proyecto se ha utilizado el software oficial de Microsoft.

#### 4.2.3 VB .NET

Visual Basic .NET es un lenguaje de programación orientado a objetos, implementado sobre el framework .NET, que puede considerarse una evolución del Visual Basic.

Para que una aplicación .NET funcione en un sistema Windows, es preciso instalar Framework .NET, el cual puede ser descargado de la página [www.microsoft.com](http://www.microsoft.com).

La elección de éste lenguaje de programación, entre otros muchos posibles para el desarrollo del software, se debe a la posibilidad de acceder de forma eficiente a los



diferentes recursos del sistema tales como puerto serie, a su seguridad e interoperabilidad con el Sistema Operativo Windows, a sus posibilidades gráficas para crear un entorno agradable, a sus aplicaciones de depuración de errores, tales como IntelliTrace y otras ayudas para crear una estructura de aplicación y código fuente robusto, escalable y sin errores.

#### **4.2.4 Framework .NET**

Framework .NET es un componente de software que puede ser añadido al Sistema operativo Windows. Provee un extenso conjunto de soluciones predefinidas para necesidades de programación de aplicación, en nuestro caso Visual Basic .NET.

Puede decirse que .NET es una biblioteca de clases base que además administra el código a partir de los distintos subsistemas del Common Language Runtime, actúa como motor de seguridad, administra excepciones o coordina la comunicación con los componentes del Sistema Operativo.

#### **4.2.5 Estructura de una aplicación .NET**

Para el caso particular de nuestro proyecto, la estructura de la aplicación estará formada por una ventana (formulario) y su correspondiente clase, que engloba las variables y funciones propias de la aplicación diseñada.

Microsoft ha dispuesto la librería de referencia completa de Visual Basic .NET 2012 en la página <http://msdn.microsoft.com/>.







## 5. ESTUDIO PREVIO

En éste capítulo veremos las diferentes posibilidades estudiadas para automatizar el brazo robot Velleman KSR10, desde las soluciones teóricas hasta las tecnológicas, que serán desarrolladas en capítulos posteriores.

### 5.1 OBJETIVO DEL PROYECTO

El objetivo final de nuestro proyecto es desarrollar un software capaz de controlar el brazo robot Velleman KSR10 tanto de forma manual como automática.

Inicialmente, el robot era controlable a través de un mando remoto de contactos que únicamente activaban o desactivaban los diferentes motores del mismo, sin tener en cuenta la llegada al final del recorrido. Los motores del brazo robot tienen una corona sin fin, de tal modo que, llegado al final del recorrido del eje, en vez de parar, dan saltos, pudiendo llegar a deteriorar los engranajes.

Uno de los puntos a tener en cuenta es que cada uno de los motores debe parar al llegar al final del recorrido de su eje, salvaguardando así la estructura del brazo robot y sus componentes.

Para la interacción con el brazo robot se decide diseñar una aplicación para PC desde la que controlar ambos modos (manual y automático) así como la posibilidad de programar trayectorias.

Para la conexión y control del brazo robot se utilizará una Placa Arduino.

### 5.2 PLACA ARDUINO

El control del brazo robot y la comunicación con el PC se lleva a cabo a través de una placa Arduino. Inicialmente se plantea la opción de utilizar la versión UNO R3. Ésta versión posee únicamente 14 entradas y salidas, de las cuales dos corresponden con el puerto serie y tan sólo 6 permiten PWM. Además, posee 6 entradas de lectura analógica.

Tras el estudio de componentes, se determina que el número de entradas y salidas necesario es superior. Como solución, se plantea el uso de un multiplexor/demultiplexor que active las salidas de los puentes en H L293D. Como se ha visto en el capítulo anterior, el puente L293D activará el motor en uno u otro sentido según la diferencia de potencial de sus dos pines de control. Si, por defecto, dejamos activos todos los motores y únicamente variamos la diferencia de potencial en los pines de control, podemos controlar los 5 motores utilizando únicamente 4 pines de arduino.

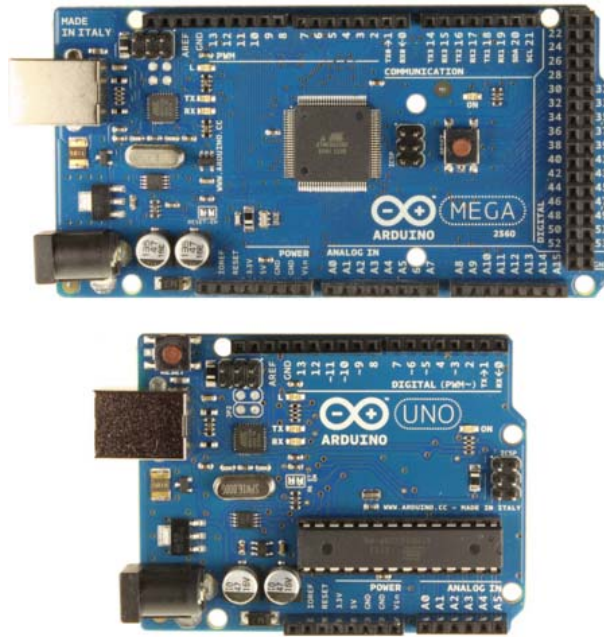


Figura 31: Arduino Mega 2560 vs Arduino Uno

El multiplexor/demultiplexor escogido inicialmente es el CD4067B, el cual consta de 24 pines, de los 4 corresponden con el selector.

Sin embargo, ésta opción es desechada debido a los tiempos de retardo y la imposibilidad de que dos motores actúen de forma simultánea. Se podría buscar uno con un tiempo de retardo menor pero el problema de simultaneidad de movimientos no quedaría solventado.

Finalmente se escoge como placa Arduino la versión MEGA 2560, el cual posee 54 pines de entrada y salida, de los cuales 14 permiten PWM. Además, posee 16 pines de entrada analógica. De éste modo, Arduino será quien controle los puentes en H L293D.

### 5.3 PUENTE EN H L293D

Se ha escogido el puente en H L293D ya que su funcionamiento corresponde exactamente con las necesidades de nuestro proyecto así como por cumplir con las especificaciones técnicas necesarias.

Al tener 5 motores, se utilizarán tres integrados L293D, siendo el tercero el que tiene un canal sin uso.

La alimentación  $V_{cc1}$  proviene de la propia alimentación de Arduino, mientras que la alimentación para los motores  $V_{cc2}$  proviene de una fuente de alimentación externa de 6V y 1000mA.

Las entradas Enable 1,2 y Enable 3,4 quedarán conectadas a una salida digital de Arduino que permita PWM, de tal forma que podamos variar la velocidad de los motores. Para el sentido de giro, se utilizarán dos salidas diferentes de Arduino por cada motor, si bien es cierto que se podría utilizar una única salida de Arduino y una puerta NOT para el otro bit, puesto que nunca podrán estar activas las dos entradas.

En los pines 3 y 6, 11 y 14, donde son conectados los motores, se recomienda conectar un condensador cerámico de 0.1uF. Sin embargo, el robot Velleman KSR10 ya ha integrado dichos condensadores junto a los motores con el fin de evitar posibles daños con picos de tensión.

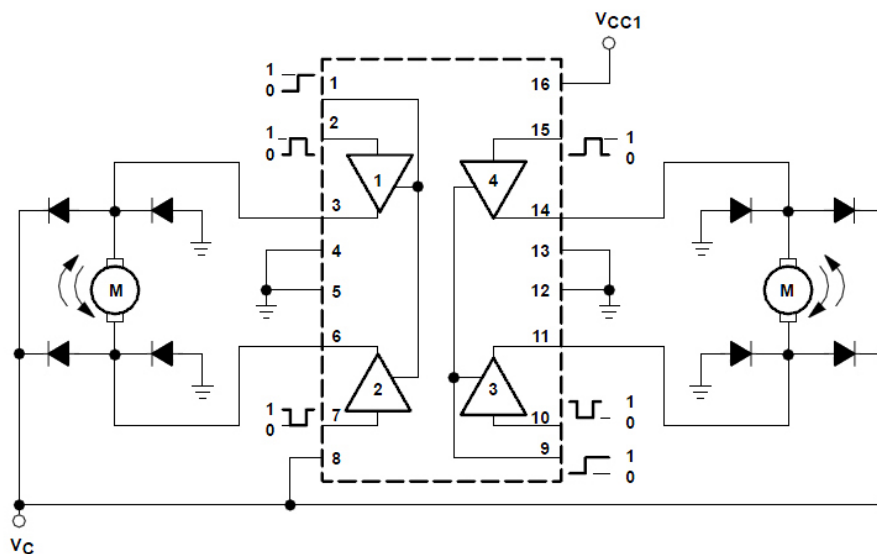


Figura 32: conexiones L293D

## 5.4 SENSORES DE POSICIÓN

Para el control de posicionamiento del extremo final del brazo robot Velleman KSR10 sobre el plano XYZ es preciso implementar un sistema de sensores. Los motores del brazo robot quedan encapsulados dentro de unas cajas junto con su correspondiente reductora:

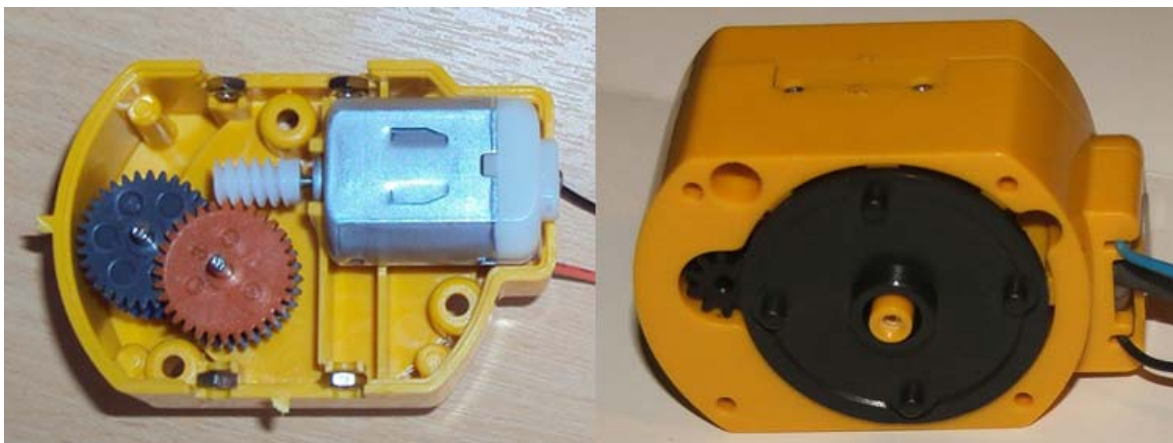


Figura 33: conexiones L293D

Inicialmente se estudia la posibilidad de incluir un encoder en la parte exterior de cada una de las cajas, pero por la estructura del mismo resultaba imposible en algunos casos. La opción de incluir el encoder dentro del encapsulado se descartó finalmente debido a la falta de espacio.

La solución final consistió en colocar resistencias variables en cada uno de los ejes del robot, con el rotor unido a una varilla metálica anclada a la parte fija contigua, de tal forma que una variación del valor de la resistencia suponga una variación en grados del brazo robot.

Para tal fin se escoge la resistencia CA9 con encapsulado CA9 H2.5 y rotor STD con relación lineal.

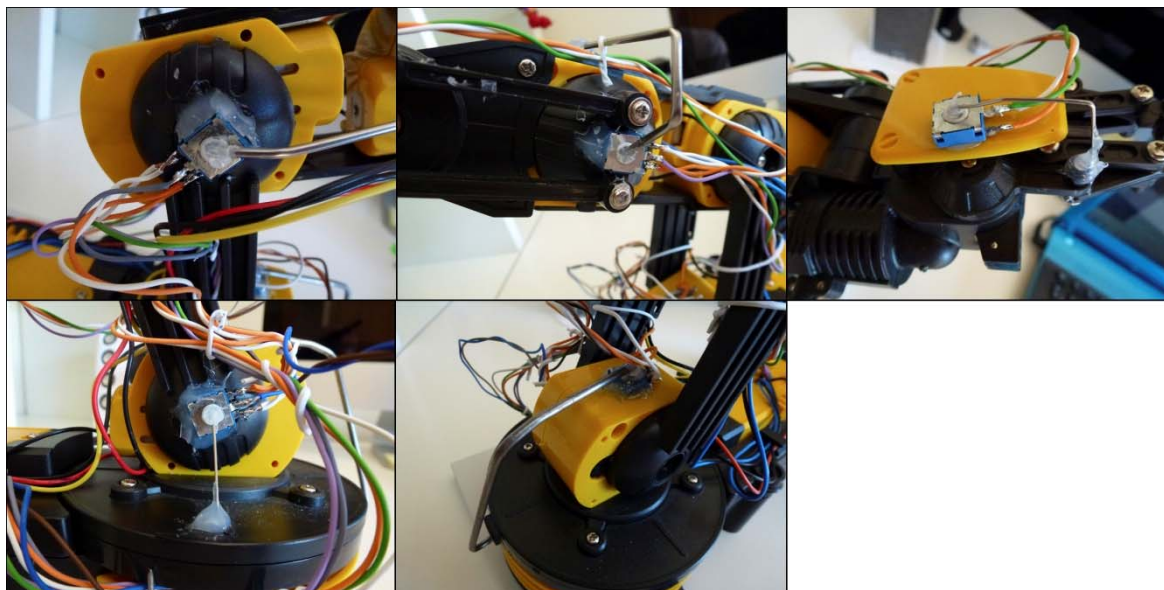


Figura 34: sensores de posición instalados en el brazo robot

Cada una de las resistencias quedará conectada a las entradas analógicas de Arduino.

### **Conversión Ángulos – Voltios**

Es necesario convertir el valor en voltios recogido a través de la entrada analógica de arduino en grados, de tal modo que, mediante cinemática inversa, podamos conocer la posición exacta del extremo final del brazo.

El valor leído por la entrada analógica de Arduino tiene un rango entre 0 y 1023. Las resistencias se han colocado de tal modo que se aproveche su recorrido adaptado al recorrido del propio eje evitando así que dichas resistencias limiten el movimiento del robot.

Tomando datos de para cada uno de los ejes se puede hallar el modelo matemático de cada una de ellas, observándose además un comportamiento prácticamente lineal en todos los casos:

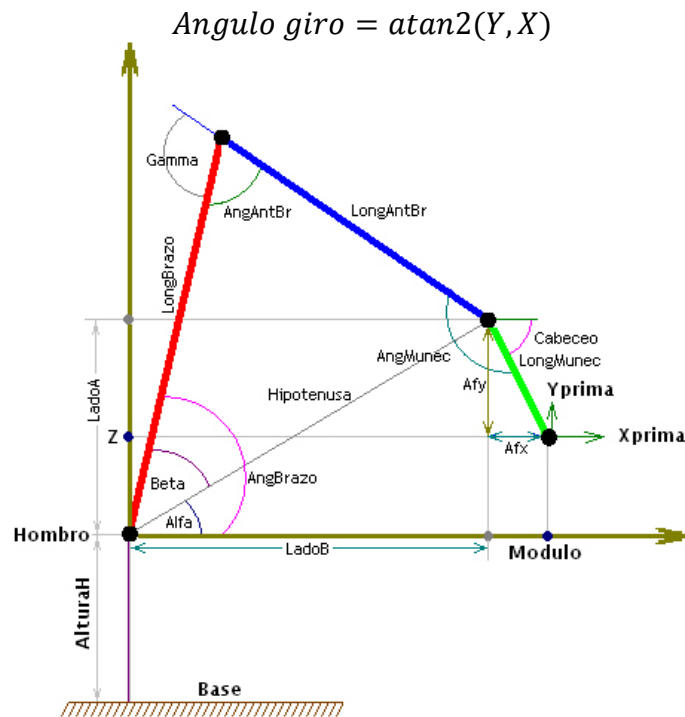


Figura 35: modelo geométrico para 5GL

Para el motor 1, correspondiente a la pinza de la mano del brazo robot, se toma únicamente la opción de estar cerrada o abierta, de tal modo que la lectura cuando la pinza está abierta es de 850 y cuando está cerrada 700.

Para el motor 2 se obtienen los siguientes datos:

Valor	Ángulo Muñeca (grados)	Ángulo Cabeceo (grados)
780	208	-28
822	195	-15
828	190	-10
884	180	0
963	155	25

Donde:

$$Angulo\ Muñeca = 180 - cabeceo$$

Y el modelo matemático para el motor 2:

$$Valor\ lectura\ arduino = 3x + 855$$

Para el motor 3 se obtienen los siguientes datos:



Valor	Ángulo Antebrazo
1005	255
762	200
665	180
260	90
100	55

Y el modelo matemático para el motor 3:

$$\text{Valor lectura arduino} = 4.5x - 142.5$$

Para el motor 4 se obtienen los siguientes datos:

Valor	Ángulo Brazo
1000	40
879	75
845	90
729	120
675	135

Y el modelo matemático para el motor 4:

$$\text{Valor lectura arduino} = 1136 - 3.4x$$

Para el motor 5 se obtienen los siguientes datos:

Valor	Ángulo Brazo
245	-70
392	-35
540	0
686	35
830	70

Y el modelo matemático para el motor 5:

$$\text{Valor lectura arduino} = 4.17x + 537.5$$

De éste modo, a partir de la cinemática inversa, se podría calcular la posición exacta del extremo final del brazo.

En éste punto, se plantea la opción de limitar los ángulos de acción de los distintos ejes, de tal modo que no lleguen a su límite en que la corona sin fin del motor se salta, pudiendo llegar a dañarse o afectar a los engranajes de la reductora.

## 5.5 CINEMÁTICA INVERSA Y ALGORITMO DE BRESENHAM

En un primer momento se pretende controlar la posición XYZ del extremo final del brazo robot, de tal modo que sea posible indicar unas nuevas coordenadas y, mediante el algoritmo de Bresenham, realizar un desplazamiento lineal de una a otra posición.

El algoritmo de cálculo de la cinemática inversa en lenguaje C para arduino:

```
modulo=sqrt(pow(x,2)+pow(y,2));
xprima=modulo;
yprima=z;
afx=cos(cabeceo)*longmuneca;
ladob=xprima-afx;
afy=sin(cabeceo)*longmuneca;
ladoa=yprima-afy-alturah;
hipotenusa=sqrt(pow(ladoa,2)+pow(ladob,2));
alfa=atan2(ladoa, ladob);
beta=acos((pow(longbrazo,2)-pow(longantbr,2)-
pow(hipotenusa,2))/(2*longbrazo*hipotenusa));
angbrazo=alfa+beta;
gamma=acos((pow(longbrazo,2)+pow(longantbr,2)-
pow(hipotenusa,2))/(2*longbrazo*longantbr));
angantbr=-(pi-gamma);
angmunec=cabeceo-angbrazo-angantbr;
anggiro=atan2(y,x);

param5=3*(anggiro*180/pi)+885;
param4=4.5*(angbrazo*180/pi)-142.5;
param3=1136-3.4*(angantbr*180/pi);
param2=4.17*(angmunec*180/pi)+537.5;
```

Mediante experimentación se comprueba que, debido a las limitaciones propias del robot, incapaz de realizar movimientos de pocos grados, se descarta dicha opción pues los movimientos descritos no son naturales.

Alimentando en activador (Enable) de cada uno de los motores mediante PWM, se determina que no es capaz de hacer mover su eje en una proporción menor a 120. Teniendo en cuenta que tenemos una resolución entre 0 y 255, correspondiendo con 0 y 5v respectivamente, y que transmiten una tensión entre 0 y 6v al motor, se calcula que, activando el Enable del motor con unos 2.35 voltios le estamos alimentando con unos 2.8 voltios. Con esa tensión, y calculando el mínimo desplazamiento de los ejes, se determina que no es posible aplicar la cinemática inversa y el algoritmo de Bresenham para el brazo robot Velleman KSR10.

Sin embargo, y para compensar esta imposibilidad, se establecerá mediante el algoritmo de programación de arduino, tres posibles movimientos de robot:

- M0: trayectoria independiente, en la que todos los motores actúan de forma simultanea.





- M1: trayectoria de recogida, en la que el brazo se desplaza en el plano XY para posteriormente desplazarse en altura, pudiendo recoger elementos desde la parte superior sin interceptarlos.
- M2: trayectoria de retorno tras suministro, en la que el brazo se desplaza en el plano Z para posteriormente desplazarse en XY, de tal modo que no intercepte el elemento suministrado.

## 5.6 PID

Para mejorar la precisión del brazo robot en sus desplazamientos, e intentando que su comportamiento sea natural sin tener que corregir su posición en exceso, se determina el uso de un PID con parámetros de sintonización propios para cada uno de los motores del brazo. Esto es debido, principalmente, a la diferencia de peso soportado por cada uno de los ejes.

Debido a sus limitaciones, y tras diversas pruebas, se determina que, según cada uno de los motores, y llegando a una zona cercana al setpoint de posición deseado, se modificarán los parámetros de sintonización, siendo más suaves. La principal limitación de los motores está en el arranque, siendo posible la reducción de tensión por debajo de los límites anteriormente indicados siempre y cuando no se produzca un paro.

La sintonización se llevará a cabo de forma manual, puesto que es imposible aplicar métodos de monitorización de movimientos y lectura de sensores y tensión de alimentación de los motores.

Se observa que el mejor comportamiento del brazo corresponde con un valor elevado de la constante proporcional ( $K_p = [6,8]$ ) y un valor reducido de la variable integral ( $K_i = [0.1,2]$ ). Es posible variar dichos parámetros para mejorar la velocidad o la precisión según se desee.

En caso de detectar cierta inestabilidad del brazo en torno al setpoint marcado, es posible aumentar la constante  $K_i$ , siempre que las especificaciones de uso no requieran una precisión elevada.

Además, se establece un valor máximo y mínimo de salida del PID de 205 y 120 respectivamente, de tal modo que la alimentación del brazo en modo automático estará comprendida entre 2.8v y 4.8v. Es posible variar estos valores en un entorno entre [100, 255] según se desee mejorar la velocidad o la precisión.

Estos valores mínimos y máximos en Arduino suponen que, para valores menores o mayores determinados por el algoritmo PID, se activará el motor con su valor inferior o superior según convenga, manteniendo unos parámetros de estabilidad y fiabilidad.

El comportamiento del PID será el siguiente: el motor será alimentado con una tensión reducida proporcional a la distancia al setpoint deseado en modo directo. En caso de



superar dicho setpoint durante la aproximación, el PID será configurado de modo inverso para recuperar la posición con la mayor precisión posible.

Como se puede observar en la gráfica, la tensión de alimentación directa e inversa se reducen proporcionalmente al setpoint, teniendo un

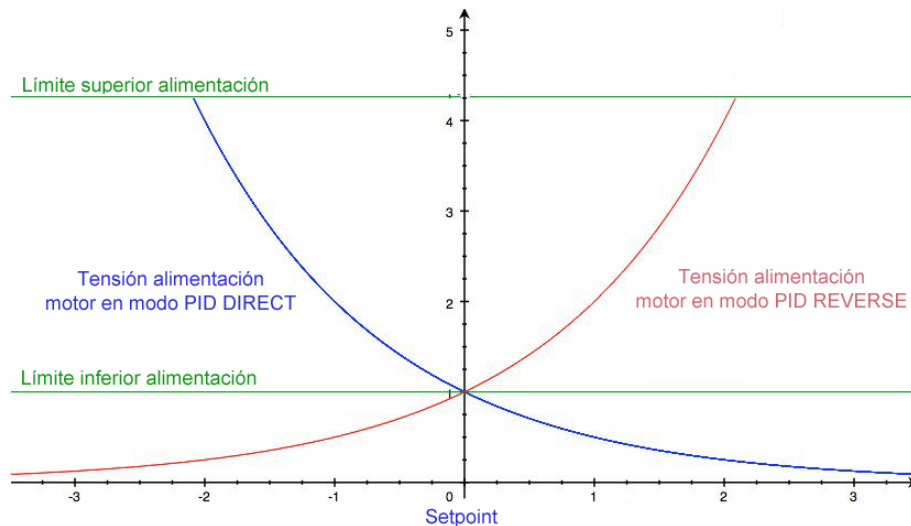


Figura 36: comportamiento deseado del PID

## 5.7 PROGRAMACIÓN DE DISPOSITIVOS

En éste punto se tratará de diferenciar las funcionalidades del software de Arduino y de .NET, determinando la actuación de cada uno de ellos en el entorno y estableciendo un protocolo de comunicación entre ellos.

Para evitar posibles fallos en la comunicación que supongan un posible descontrol del brazo robot, se establece que el funcionamiento operativo completo sea controlado por Arduino, dejando únicamente las funciones de programación de trayectorias y control manual al software .NET

En caso de una pérdida de comunicación serie, el brazo robot interrumpirá su funcionamiento. No obstante, se instalará un botón reset en el propio controlador.

La comunicación se llevará en ambos sentidos mediante una comunicación serie a 9600 baudios. Se ha verificado, de modo experimental, la pérdida de datos durante la transmisión, lo que tendrá que ser controlado y verificado durante las transmisiones, de tal forma que la comunicación sea correcta.

Para la creación de trayectorias, sería .NET quien gestione las rutinas, pudiendo ser almacenadas en el PC y transferidas a Arduino en cualquier momento.



Arduino recibirá una trama con la configuración completa de la trayectoria, la cual será segmentada y almacenada en un array de configuración.

El número total de movimientos quedará limitado a 25.

## 5.8 CONCLUSIONES

Tras el estudio de las diferentes posibilidades de control, se determina que:

- Arduino Mega 2560 será el controlador utilizado para mantener la comunicación con el dispositivo de control, en nuestro caso el PC, y el brazo robot. Los cálculos de posicionamiento de control automático serán gestionados por Arduino, reduciendo las atribuciones de la aplicación PC.
- Cada eje del brazo robot tendrá instalada una resistencia para el cálculo de su posición.
- Los motores quedan controlados a partir de puertos en H L293D y variando la velocidad a partir de las salidas PWM de Arduino.
- Cada motor tendrá configurado un PID, buscando la mayor rapidez y precisión en su posicionamiento.
- No es posible aplicar la cinemática inversa y el algoritmo de Bresenham al brazo robot por lo que se especifican tres tipos de movimientos para el control total del brazo con movimientos naturales.
- Al descartar la aplicación de la cinemática inversa, no es necesario el uso del modelado matemático que relaciona el valor de la resistencia con el ángulo del eje, si no que se mantendrá el uso del parámetro recibido en Arduino con una resolución entre 0 y 1024, de tal forma que se eviten errores de precisión.
- El software Arduino controlará el movimiento automatizado, los PID de cada motor, los límites de movimiento de cada eje para evitar deteriorar los engranajes de la reductora y almacenará la información de programación de movimientos.
- El software .NET será el en cargado de interactuar con el usuario, permitiendo el control manual, el control automático, la programación de trayectorias así como controles de parada automática y resto de funcionalidades típicas de control.
- Arduino estará conectado al PC mediante cable USB, utilizando la alimentación de éste último para alimentar la placa y la circuitería lógica restante.
- Los motores serán alimentados mediante una fuente de alimentación externa de 6V y 1000mA, amperaje máximo soportado por un puente en H L293D para dos canales.
- La PCB diseñada posee un circuito regulador de tensión que podría ser conectado a la entrada de la fuente de alimentación para alimentar Arduino y la circuitería lógica, de tal modo que sea posible la supresión del cable USB y Arduino y el PC quedasen conectados por otro medio conectado a los pines 0 y 1, por ejemplo, un módulo bluetooth.

## 6. INTERCONEXIÓN DEL SISTEMA

En este capítulo se detalla el diseño eléctrico de la placa de comunicación de arduino con el brazo robot Velleman KSR10 y el resto de particularidades de conexión (conexión de entradas y salidas de arduino, adaptación de alimentaciones, etc).

### 6.1 DETALLE CONEXIONADO

A continuación se detalla el esquema de conexionado de Arduino, los puentes en H que controlan el sentido de giro y velocidad de los motores y los diferentes sensores instalados en el brazo robot:

<b>MOTOR1</b>		
Descripción	Pin Arduino	Pin Puente L293D
Motor 1 Enable	2	1 (IC1)
Motor 1 DCHA	22	2 (IC1)
Motor 1 IZQ	23	7 (IC1)
Motor 1 +	-	3 (IC1)
Motor 1 -	-	6 (IC1)

<b>MOTOR2</b>		
Descripción	Pin Arduino	Pin Puente L293D
Motor 2 Enable	7	9 (IC1)
Motor 2 DCHA	24	10 (IC1)
Motor 2 IZQ	25	15 (IC1)
Motor 2 +	-	14 (IC1)
Motor 2 -	-	11(IC1)

<b>MOTOR3</b>		
Descripción	Pin Arduino	Pin Puente L293D
Motor 3 Enable	4	1 (IC2)
Motor 3 DCHA	26	2 (IC2)
Motor 3 IZQ	27	7 (IC2)
Motor 3 +	-	3 (IC2)
Motor 3 -	-	6 (IC2)

<b>MOTOR4</b>		
Descripción	Pin Arduino	Pin Puente L293D
Motor 4 Enable	5	9 (IC2)
Motor 4 DCHA	28	10 (IC2)
Motor 4 IZQ	29	15 (IC2)
Motor 4 +	-	14 (IC2)
Motor 4 -	-	11(IC2)



<b>MOTORS</b>		
Descripción	Pin Arduino	Pin Puente L293D
Motor 5 Enable	6	1 (IC3)
Motor 5 DCHA	30	2 (IC3)
Motor 5 IZQ	31	7 (IC3)
Motor 5 +	-	3 (IC3)
Motor 5 -	-	6 (IC3)

<b>LED</b>	
Descripción	Conexión
Led+	CN8+
Led-	CN8- GND

<b>L293D (IC1)</b>		
Pin	Conexión	Detalle
1	Arduino Pin 2	EN 1,2
2	Arduino Pin 22	Input 1
3	Motor 1 +	Output 1
4	GND	GND
5	GND	GND
6	Motor 1 -	Output 2
7	Arduino Pin 23	Input 2
8	Vcc2 (6v 1000mA)	Vcc2
9	Arduino Pin 7	EN 3,4
10	Arduino Pin 24	Input 3
11	Motor 2 +	Output 3
12	GND	GND
13	GND	GND
14	Motor 2 -	Output 4
15	Arduino Pin 25	Input 4
16	Arduino +5v	Vcc

<b>L293D (IC2)</b>		
Pin	Conexión	Detalle
1	Arduino Pin 4	EN 1,2
2	Arduino Pin 26	Input 1
3	Motor 3 +	Output 1
4	GND	GND
5	GND	GND
6	Motor 3 -	Output 2
7	Arduino Pin 27	Input 2
8	Vcc2 (6v 1000mA)	Vcc2
9	Arduino Pin 5	EN 3,4
10	Arduino Pin 28	Input 3
11	Motor 4 +	Output 3



12	GND	GND
13	GND	GND
14	Motor 4 -	Output 4
15	Arduino Pin 29	Input 4
16	Arduino +5v	Vcc

L293D (IC3)		
Pin	Conexión	Detalle
1	Arduino Pin 6	EN 1,2
2	Arduino Pin 30	Input 1
3	Motor 5 +	Output 1
4	GND	GND
5	GND	GND
6	Motor 5 -	Output 2
7	Arduino Pin 31	Input 2
8	Vcc2 (6v 1000mA)	Vcc2
9	GND	EN 3,4
10	GND	Input 3
11	GND	Output 3
12	GND	GND
13	GND	GND
14	GND	Output 4
15	GND	Input 4
16	Arduino +5v	Vcc

SENSORES			
Descripción	Pin Arduino	+5v	GND
Sensor 1	A1	CN7+	CN7- GND
Sensor 2	A2	CN7+	CN7- GND
Sensor 3	A3	CN7+	CN7- GND
Sensor 4	A4	CN7+	CN7- GND
Sensor 5	A5	CN7+	CN7- GND

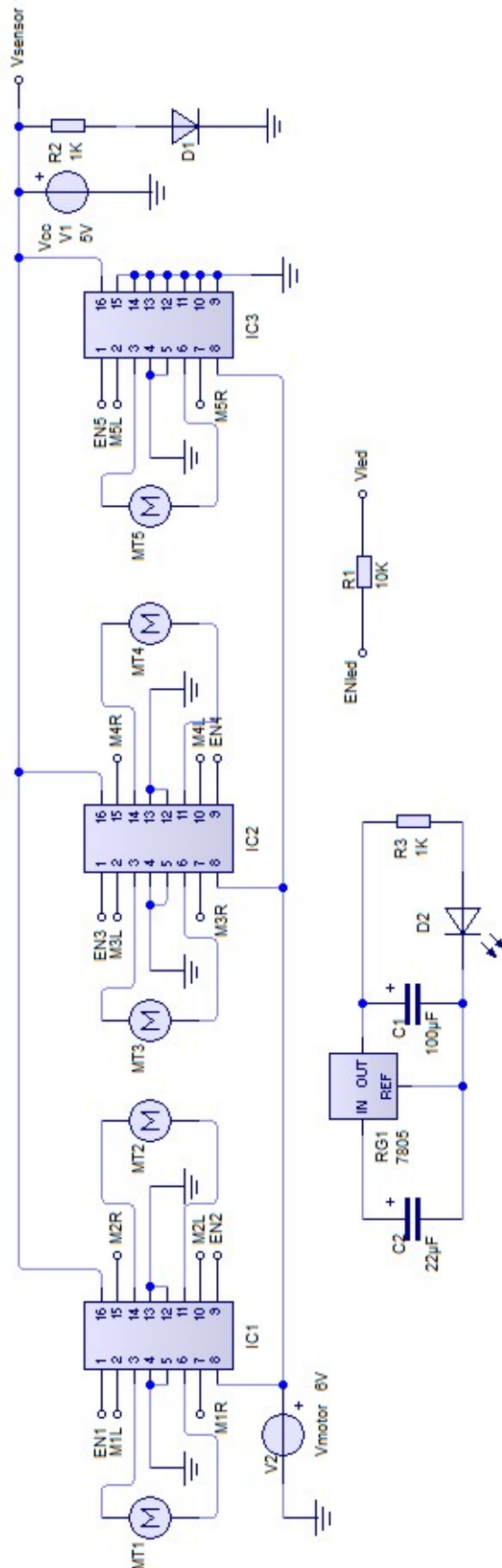
ARDUINO		
Pin	Descripción	Conexión
0	Conexión serie RX	Sin conectar (USB)
1	Conexión serie TX	Sin conectar (USB)
2	Motor 1 Enable	IC1 1
4	Motor 3 Enable	IC2 1
5	Motor 4 Enable	IC2 9
6	Motor 5 Enable	IC3 1
7	Motor 2 Enable	IC2 9
22	Motor 1 DCHA	IC1 2
23	Motor 1 IZQ	IC1 7
24	Motor 2 DCHA	IC1 10

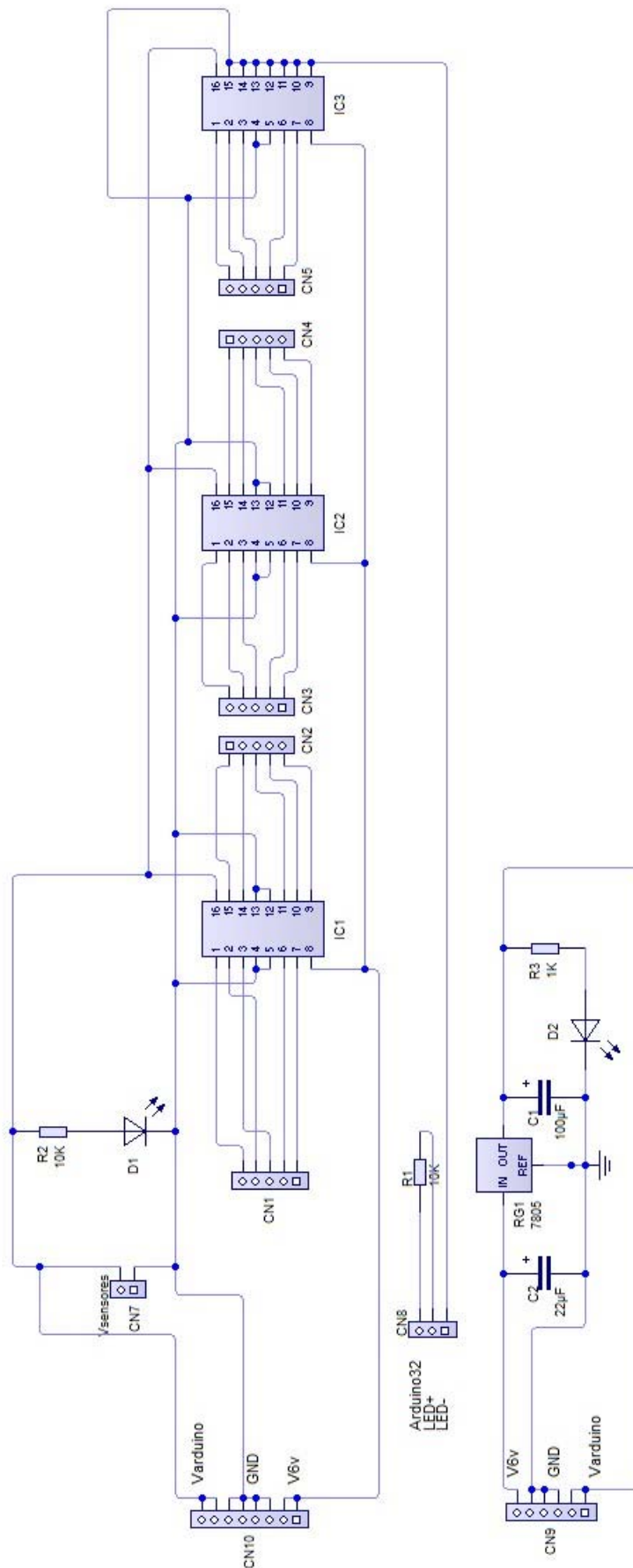


25	Motor 2 IZQ	IC1 15
26	Motor 3 DCHA	IC2 2
27	Motor 3 IZQ	IC2 7
28	Motor 4 DCHA	IC2 10
29	Motor 4 IZQ	IC2 15
30	Motor 5 DCHA	IC3 2
31	Motor 5 IZQ	IC3 7
32	Led	CN8 A
A1	Sensor 1	Sensor 1
A2	Sensor 2	Sensor 2
A3	Sensor 3	Sensor 3
A4	Sensor 4	Sensor 4
A5	Sensor 5	Sensor 5
+5v	Alimentación	CN10 +5v
GND	Alimentación	CN10 +5v
GND	Pulsador reset	Pulsador reset
Reset	Pulsador reset	Pulsador reset

## 6.2 ESQUEMA ELÉCTRICO

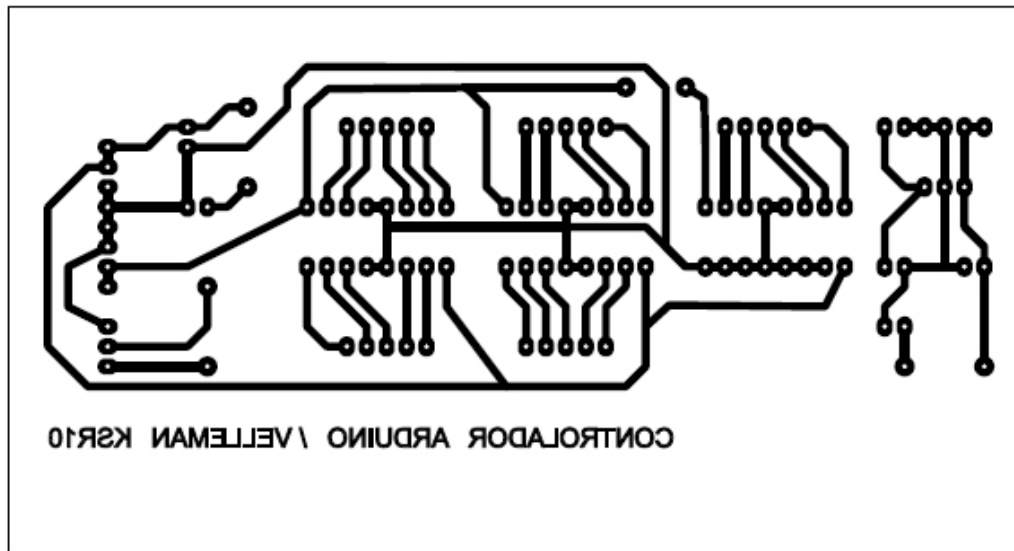
A continuación se muestra el esquema eléctrico:



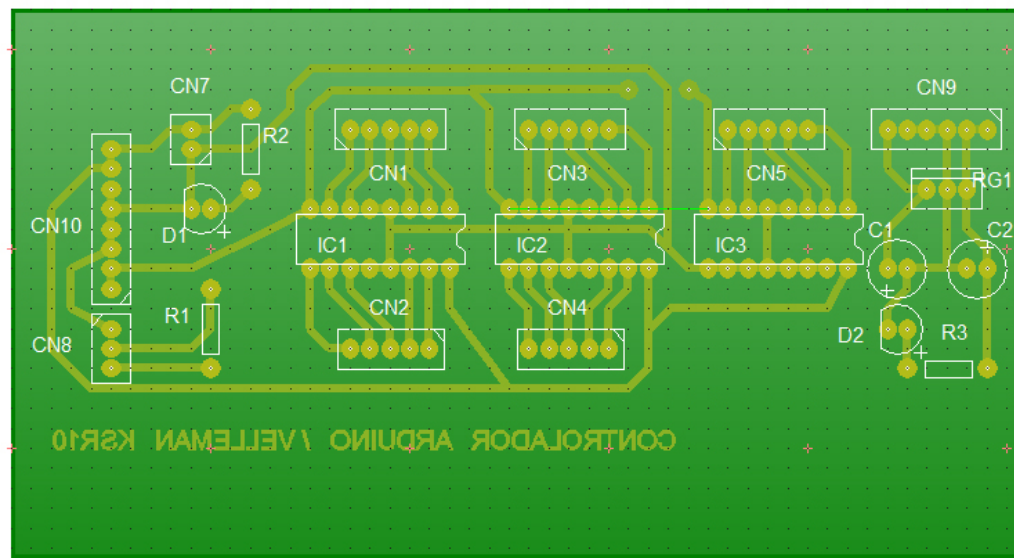


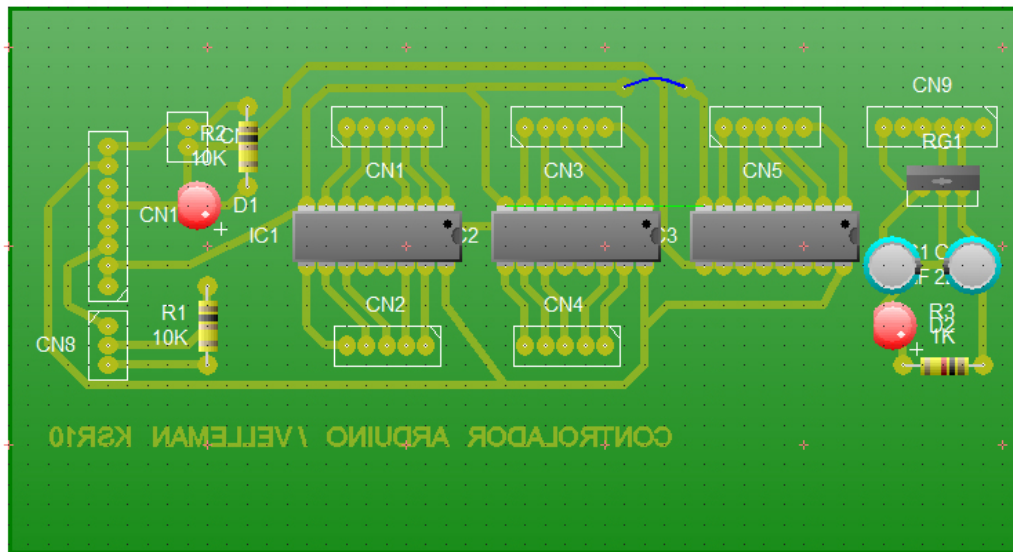


A continuación se muestra la placa diseñada, la cual puede ser impresa en una placa de una sola capa:



El esquema de conexiones de la placa diseñada es el siguiente:





LISTADO COMPONENTES	
IDENTIFICADOR	COMPONENTE
IC1	L293D
IC2	L293D
IC3	L293D
CN1	Conector hembra 5
CN2	Conector hembra 5
CN3	Conector hembra 5
CN4	Conector hembra 5
CN5	Conector hembra 5
CN7	Conector hembra 2
CN8	Conector hembra 3
CN9	Conector hembra 6
CN10	Conector hembra 10
R1	Resistencia 10K
R2	Resistencia 10K
R3	Resistencia 10K
RG1	LM7805
D1	Diodo Led
D2	Diodo Led
C1	Condensador electrolítico 22uf
C2	Condensador electrolítico 100uf

El esquema de conexiones, considerando la lectura de arriba hacia abajo o izquierda a derecha, de cada uno de los conectores es el siguiente:

CN10		Alimentación
CN10	1	+5v (libre)
	2	+5v (Arduino)
	3	GND (Arduino)
	4	GND (libre)
	5	GND (libre)
	6	GND (Vcc2 6v 1000mA)
	7	+6v (Vcc2 6v 1000mA)
	8	+6v (Vcc2 6v 1000mA) (libre)

CN9		Alimentación (sin USB)
CN9	1	+6v (Vcc2 6v 1000mA) (libre; se conectaría a CN10-8)
	2	GND (libre; se conectaría a CN10-5)
	3	GND (libre)
	4	GND (libre; se conectaría a CN10-1)
	5	+5v (libre)
	6	+5v (libre)

CN8		Led
CN8	1	Led – (GND)
	2	Led +
	3	Arduino Pin LED

CN7		Alimentación sensores
C7	1	+5v sensores
	2	GND sensores

CN5		Control Motor 5
CN5	1	EN 5 – Pin Arduino 6
	2	Motor 5 DCHA – Pin Arduino 30
	3	Motor 5 +
	4	Motor 5 -
	5	Motor 5 IZQ – Pin Arduino 31

CN4		Control Motor 4
CN4	1	EN 4 – Pin Arduino 5
	2	Motor 4 DCHA – Pin Arduino 28
	3	Motor 4 +
	4	Motor 4 -
	5	Motor 4 IZQ – Pin Arduino 29

CN3		Control Motor 3
CN3	1	EN 3 – Pin Arduino 4
	2	Motor 3 DCHA – Pin Arduino 26
	3	Motor 3 +
	4	Motor 3 -
	5	Motor 3 IZQ – Pin Arduino 27

CN2		Control Motor 2
CN2	1	EN 2 – Pin Arduino 7
	2	Motor 2 DCHA – Pin Arduino 25
	3	Motor 2 +
	4	Motor 2 -
	5	Motor 2 IZQ – Pin Arduino 26

CN1		Control Motor 1
CN1	1	EN 1 – Pin Arduino 2
	2	Motor 1 DCHA – Pin Arduino 22
	3	Motor 1 +
	4	Motor 1 -
	5	Motor 1 IZQ – Pin Arduino 23

En la documentación adjunta en disco portable se encuentra el esquema eléctrico así como el esquema de la placa diseñada.

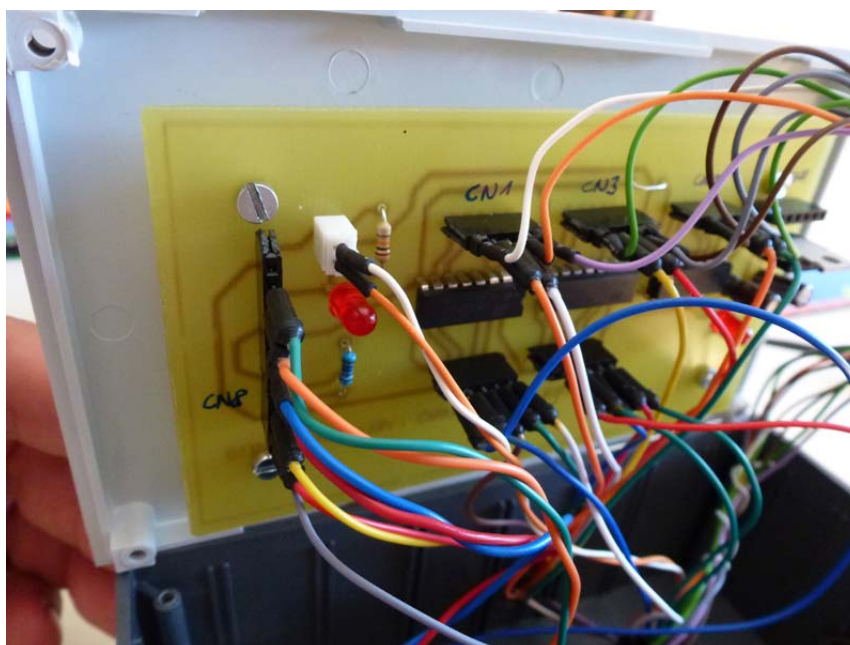


Figura 37: detalle conexionado PCB

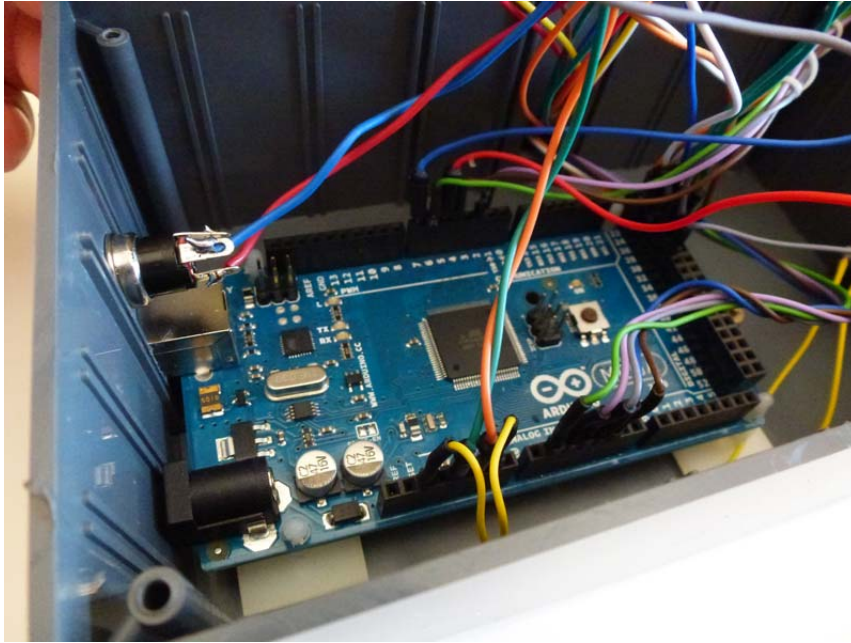


Figura 38: detalle conexaso Arduino Mega



# 7. DESARROLLO DE SOFTWARE

En este apartado se van a detallar todas las particularidades del software desarrollado.

## 7.1 INTRODUCCIÓN

El brazo robot Velleman KSR10 va a ser controlado mediante una aplicación desarrollada en Visual basic .NET, la cual interactuará con el usuario para la creación de trayectorias y el control manual del brazo robot.



Figura 39: esquema funcional

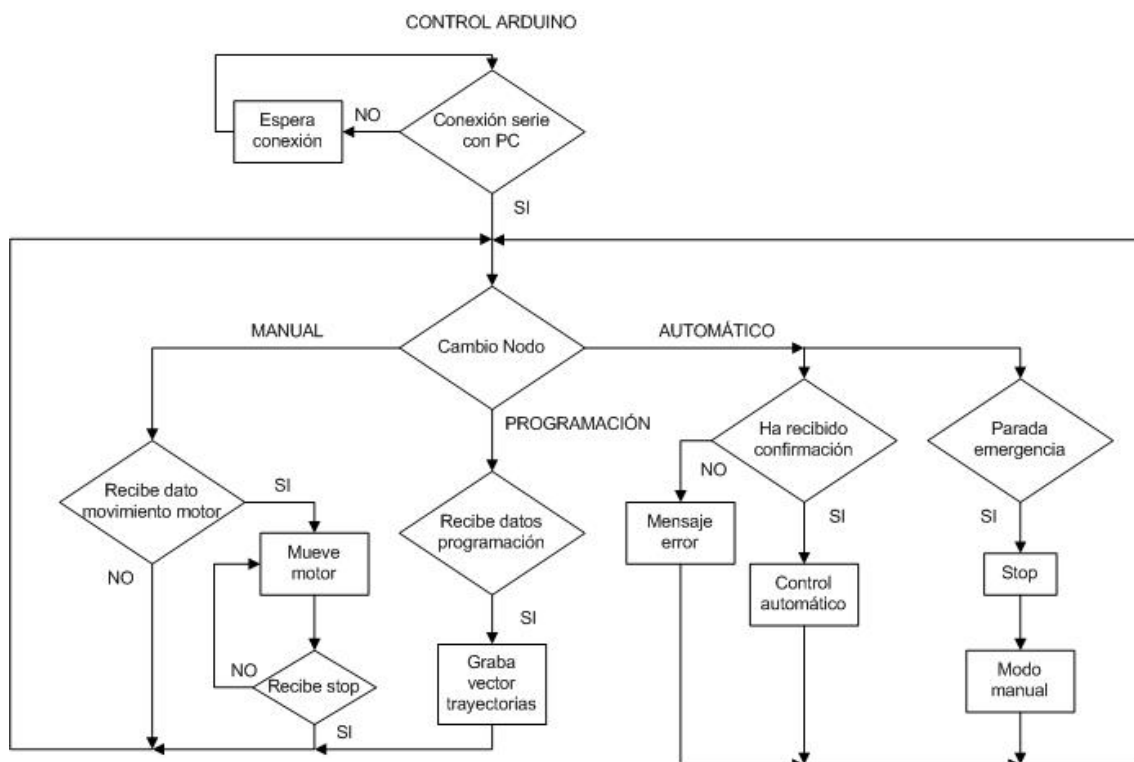


Figura 40: diagrama flujo Arduino

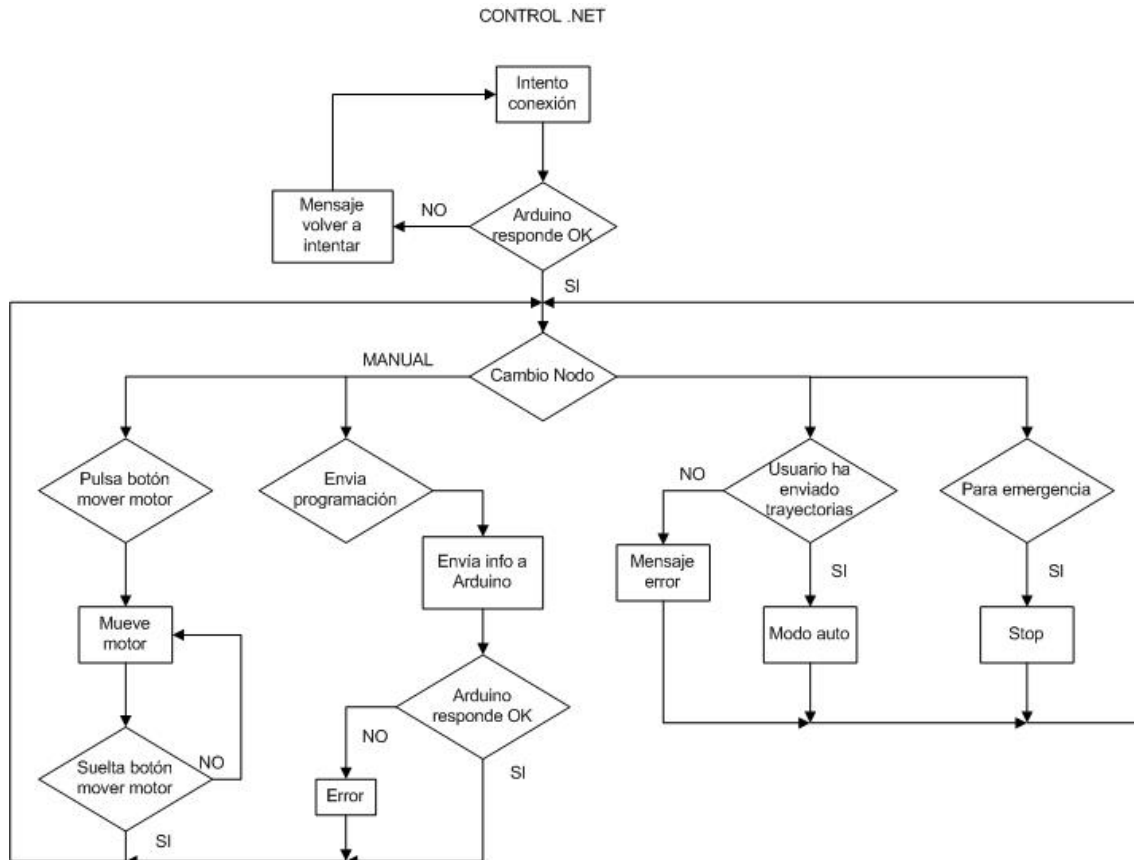


Figura 41: diagrama flujo .NET

Se han establecido unos requisitos funcionales y no funcionales específicos y básicos para el desarrollo del software y son detallados en los puntos 7.2 y 7.3:

## 7.2 REQUISITOS NO FUNCIONALES

Éstos son los requisitos no funcionales de la aplicación desarrollada:

Requisito no funcional	Referencia	Nombre de Funtional Requirement	Descripción
Arduino estará programado en lenguaje C#.	NFR-0001	Plataforma Arduino	El sistema deberá estar construido en C# y funcionar en la plataforma Arduino.
El software controlador tendrá sistema operativo Windows. El desarrollo de software en Visual Basic .NET	NFR-0002	Plataforma PC	El sistema deberá estar construido en .NET por lo que únicamente funcionará en Windows



Software portable, escalable y extensible	NFR-0003	Alta flexibilidad	El sistema deberá ser altamente flexible ante nuevas funcionalidades añadidas mediante módulos o paquetes
Interfaz amigable, sencilla e intuitiva	NFR-0004	Interface	Botones sencillos e intuitivos, facilidad en la operativa, etc.
Tratamiento eficaz de errores detectados en la transmisión	NFR-0005	Tratamiento de errores	Control de excepción de error en las comunicaciones

### 7.3 REQUISITOS FUNCIONALES

Éstos son los requisitos funcionales de la aplicación desarrollada:

Requisito funcional	Referencia	Descripción	Ubicación
El sistema proporcionará una opción de control manual del brazo robot	FR-0001	Se incluirán una serie de botones de control desde los que el usuario puede accionar cualquiera de los motores	.NET
El sistema proporcionará una opción de control automático del brazo robot	FR-0002	Se incluirán una serie de opciones para crear trayectorias que puedan ser transferidas a Arduino, el cual gestionará la automatización	.NET
El software controlador permitirá la parada de emergencia en modo automático	FR-0003	En cualquier momento durante la ejecución del modo automático, será posible la parada de emergencia. En éste caso, el robot	.NET



		pasa a modo de control manual	
El software controlador permitirá almacenar trayectorias	FR-0004	Existirá una opción para almacenar la secuencia de automatización en archivos de texto formato csv	.NET
El software controlador permitirá cargar trayectorias almacenadas	FR-0005	Existirá una opción para almacenar la secuencia de automatización desde un archivo de texto csv previamente cargado	.NET
El software controlador permitirá la edición de trayectorias creadas	FR-0006	Existirán las opciones de eliminación de trayectorias o eliminación de movimientos concretos	.NET
El software controlador enviará las instrucciones manuales a través del puerto serie	FR-0007	Cada uno de los botones envía una instrucción de movimiento en el evento onmousedown y de paro en onmouseup	.NET
El software controlador enviará las instrucciones de configuración de trayectorias a través del puerto serie	FR-0008	Se creará una trama con la información relativa a todas las acciones descritas y se enviará mediante puerto serie. Arduino lo decodificará y almacenará en un array de trayectorias	.NET
Los controladores PID serán gestionados a	FR-0009	Los controladores de posicionamiento de los motores	Arduino



través de Arduino		serán gestionados por Arduino y su librería PID	
-------------------	--	---	--

## 7.4 TRANSMISIÓN DE CONFIGURACIÓN DE TRAYECTORIAS

Una de las “limitaciones” de Arduino en relación a otros lenguajes de programación, es la inexistencia del tipo de datos “String”. Por tanto, al recibir una cadena de caracteres habrá que procesarla carácter a carácter almacenándolo, por ejemplo, en un vector de caracteres. Esto supone un inconveniente a la hora de recibir los parámetros de configuración de trayectorias.

Como solución se ha ideado un sistema de transmisión de tramas, de tal modo que el software .NET creará una trama de información con la configuración completa de todas las instrucciones así como control de errores de envío.

Como se vio con anterioridad y para solventar el problema de aplicación de cinemática directa y algoritmo de Bresenham, se han definido tres tipos de movimientos:

- Movimiento tipo 0: todos los motores se activan simultáneamente. Es útil cuando el brazo no puede interceptar ningún elemento.
- Movimiento tipo 1: inicialmente se posiciona sobre la posición XY definida (motor 5) y posteriormente se activan el resto de motores para lograr la posición del plano Z deseada. Es útil cuando el brazo va a recoger un elemento.
- Movimiento tipo 2: inicialmente se posiciona sobre la posición del plano Z deseada y posteriormente sobre la XY. Es útil cuando el brazo vuelve de dejar un elemento.

En Arduino se ha declarado la función “posición(int param)” a la que se envía un parámetro entero (0, 1, 2) según el tipo de movimiento deseado.

Del mismo modo, se ha creado la función “pinza(int param)” a la que se envía un parámetro entero (0,1) según se desee abrir la pinza (0) o cerrarla (1).

Se han declarado un tipo y subtipo común para Arduino y .NET, de tal modo que ambos entiendan cualquier tipo de instrucción:

Tipo	
Descripción	Carácter
Movimiento	M
Pinza (mano)	H



Subtipo	
Descripción	Carácter
Cierra (pinza(0))	C
Cierra (pinza(1))	A
Posicion(0)	I
Posicion (1)	L
Posicion (2)	T

Cara instrucción contendrá un tipo, un subtipo y la posición de los 5 motores –si corresponde con un movimiento-.

Por tanto, la trama genérica para cada instrucción es la siguiente:

TRAMA ORDEN						
Tipo	Subtipo	M1	M2	M3	M4	M5

La trama completa de envío tendrá la siguiente estructura:

TRAMA ENVÍO CONFIGURACIÓN					
Contador 1	Orden 1	Orden 2	...	Contador 2	#

Donde Contador 1 corresponde con el número de instrucciones y Contador 2 el número de caracteres enviados hasta el separador entre la última orden y dicho contador.

El carácter “#” se utiliza para indicar al receptor el final de la trama. Para separar las órdenes se utilizará el carácter “\$” y para separar los parámetros de las órdenes el carácter “!”.

Tras la recepción de la trama, se verifica que el número de órdenes recibidas corresponda con Contador 1 y que el número de caracteres recibidos corresponde con Contador 2.

Tras esta verificación, se hace un SPLIT de los datos, que corresponde con separar los elementos en diferentes vectores para ser tratados. De ese modo:

SPLIT (trama) → Quedan separados Contador1, Orden1, Orden2,..., Contador 2.

SPLIT (Orden1) → Quedan separados los elementos tipo, subtipo y movimientos.

SPLIT (Movimientos) → Quedan separados los datos para M1, M2, M3, M4 y M5.

Una vez recibida ésta información, Arduino la almacenará en un vector de 7 elementos, en el que se mantiene el mismo formato:

VECTOR ARDUINO						
Tipo	Subtipo	M1	M2	M3	M4	M5



En el caso de tratarse de una instrucción de cierre o apertura de la mano, únicamente habrá información en los elementos 0 y 1 del vector.

En el capítulo 7.5 se especifica el algoritmo de tratado de ésta información así como en el 7.6 cómo es generado en .NET.

## 7.5 SOFTWARE ARDUINO

### 7.5.1 Librerías

Para el desarrollo del software del controlador se van a utilizar las librerías STRING y PID\_V1 de Arduino.

Para incluir ambas:

```
#include <PID_v1.h>
#include <string.h>
```

### 7.5.2 Variables y Constantes

Se han definido unas constantes para control de algunos elementos:

```
//Tamaño del buffer de lectura del puerto serie para recepción
//de configuración del modo automático
#define BUFFSIZ 450
//Tamaño de parámetros almacenados para el control automático
#define ORDENSIZ 7
//Máximo de movimientos programados como trayectoria
#define INSTRUCCIONES 25
```

Manualmente se han calculado los valores mínimo y máximo de los sensores fuera de los cuales existe riesgo de daño de los engranajes de las reductoras de los motores:

```
#define M1MAX 900
#define M1MIN 700
#define M2MAX 950
#define M2MIN 780
#define M3MAX 1000
#define M3MIN 100
#define M4MAX 990
#define M4MIN 575
#define M5MAX 820
#define M5MIN 190
```

Se han declarados los pines de salida de control de los motores:

```
//Motor 1 --> Mano
int motor1EnablePin = 2;
int motor1Pin1 = 22;
int motor1Pin2 = 23;
```

```
//Motor 2 --> Muñeca
int motor2EnablePin = 7;
int motor2Pin1 = 25;
int motor2Pin2 = 24;
//Motor 3 --> Codo
int motor3EnablePin = 4;
int motor3Pin1 = 26;
int motor3Pin2 = 27;
//Motor 4 --> Hombro
int motor4EnablePin = 5;
int motor4Pin1 = 28;
int motor4Pin2 = 29;
//Motor 5 --> Base
int motor5EnablePin = 6;
int motor5Pin1 = 30;
int motor5Pin2 = 31;
//Led
int lightPin=32;
```

Se han declarado los pines de lectura de los sensores:

```
int sensorPin1 = A1;
int sensorValue1 = 0;
int mem_sensor1=0;
int sensorPin2 = A2;
int sensorValue2 = 0;
int mem_sensor2=0;
int sensorPin3 = A3;
int sensorValue3 = 0;
int mem_sensor3=0;
int sensorPin4 = A4;
int sensorValue4 = 0;
int mem_sensor4=0;
int sensorPin5 = A5;
int sensorValue5 = 0;
int mem_sensor5=0;
```

Declaración de arrays para programación de la automatización:

```
int pos[INSTRUCCIONES][ORDENSIZ];
char buffer[BUFFSIZ];
```

Declaración de parámetros de controladores PID:

```
double aggKp1=6, aggKi1=0.1, aggKd1=0;
double consKp1=0.2, consKi1=0.05, consKd1=0;
double aggKp2=6, aggKi2=0.1, aggKd2=0;
double consKp2=0.6, consKi2=0.05, consKd2=0;
double aggKp3=8, aggKi3=0.1, aggKd3=0;
double consKp3=2.2, consKi3=0.05, consKd3=0;
double aggKp4=8, aggKi4=0.1, aggKd4=0;
double consKp4=0.2, consKi4=0.05, consKd4=0;
double aggKp5=8, aggKi5=0.1, aggKd5=0;
double consKp5=0.2, consKi5=0.05, consKd5=0;

PID myPID1(&Input1, &Output1, &Setpoint1, consKp1, consKi1,
consKd1, DIRECT);
```



```
PID myPID2(&Input2, &Output2, &Setpoint2, consKp2, consKi2,
consKd2, DIRECT);
PID myPID3(&Input3, &Output3, &Setpoint3, consKp3, consKi3,
consKd3, DIRECT);
PID myPID4(&Input4, &Output4, &Setpoint4, consKp4, consKi4,
consKd4, DIRECT);
PID myPID5(&Input5, &Output5, &Setpoint5, consKp5, consKi5,
consKd5, DIRECT);
```

El resto de variables declaradas para diversas funcionalidades:

```
int var,var_old,var2;
int control=1;
int total_progs=0;
String cadena;
int dato_puerto_serial;
int intervalo=0;
int control_modo=0; // 0 manual; 1 auto
int control_prog=0;

double Setpoint1, Input1, Output1,Setpoint_old1;
double Setpoint2, Input2, Output2,Setpoint_old2;
double Setpoint3, Input3, Output3,Setpoint_old3;
double Setpoint4, Input4, Output4,Setpoint_old4;
double Setpoint5, Input5, Output5,Setpoint_old5;
```

Cabe destacar el uso de la variable `control_modo`, que valdrá 0 o 1 según el modo de funcionamiento del brazo robot, y `control_prog`, que valdrá 1 mientras se esté recibiendo información de programación de trayectorias.

### 7.5.3 SETUP

En éste apartado se declaran los modos de funcionamiento de los pines y se inicia la comunicación a través del puerto serie:

```
pinMode(motor1Pin1, OUTPUT);
pinMode(motor1Pin2, OUTPUT);
pinMode(motor1EnablePin, OUTPUT);
digitalWrite(motor1EnablePin, HIGH);
pinMode(motor2Pin1, OUTPUT);
pinMode(motor2Pin2, OUTPUT);
pinMode(motor2EnablePin, OUTPUT);
digitalWrite(motor2EnablePin, HIGH);
pinMode(motor3Pin1, OUTPUT);
pinMode(motor3Pin2, OUTPUT);
pinMode(motor3EnablePin, OUTPUT);
digitalWrite(motor3EnablePin, HIGH);
pinMode(motor4Pin1, OUTPUT);
pinMode(motor4Pin2, OUTPUT);
pinMode(motor4EnablePin, OUTPUT);
digitalWrite(motor4EnablePin, HIGH);
pinMode(motor5Pin1, OUTPUT);
pinMode(motor5Pin2, OUTPUT);
pinMode(motor5EnablePin, OUTPUT);
digitalWrite(motor5EnablePin, HIGH);
pinMode(lightPin, OUTPUT);
```

```
digitalWrite(lightPin, LOW);

//Inicialización de puerto serie
Serial.begin(9600);
Setpoint_old1=analogRead(sensorPin1);
Setpoint_old2=analogRead(sensorPin2);
Setpoint_old3=analogRead(sensorPin3);
Setpoint_old4=analogRead(sensorPin4);
Setpoint_old5=analogRead(sensorPin5);

myPID1.SetMode(AUTOMATIC);
myPID2.SetMode(AUTOMATIC);
myPID3.SetMode(AUTOMATIC);
myPID4.SetMode(AUTOMATIC);
myPID5.SetMode(AUTOMATIC);
myPID1.SetOutputLimits(120, 205);
myPID2.SetOutputLimits(120, 205);
myPID3.SetOutputLimits(120, 205);
myPID4.SetOutputLimits(120, 205);
myPID5.SetOutputLimits(120, 205);
```

### 7.5.4 LOOP

La estructura principal de la función loop es la siguiente:

```
if (Serial.available() > 0) {
  dato_puerto_serial = Serial.read();
  if(dato_puerto_serial == 'P'){
    parada_emergencia();
  }

  if(dato_puerto_serial == 'M'){
    //modo manual
    control_mod0=0;
  }else if(dato_puerto_serial == 'N'){
    //modo auto
    control_mod0=1;
  }
}
```

De éste modo se seleccionará el modo de funcionamiento manual o automático, en paralelo con la opción parada de emergencia.

### 7.5.5 Control Manual

En el modo manual, se recibirá un carácter de control que activará uno u otro motor en uno u otro sentido y que se parará al recibir el carácter de control “STOP”. El carácter de control de parada es el mismo para todos los motores. Como se verá posteriormente, el motor se accionará al pulsar el botón (evento onmousedown) y se parará al soltar el botón (evento onmouseup). Por ejemplo, para controlar el MOTOR1:

```
//Control MOTOR 1 --> MANO
if (dato_puerto_serial == 'Q') {
  do{
```



```
    dato_puerto_serial = Serial.read();
    if(dato_puerto_serial=='O')break;
    sensorValue1 = analogRead(sensorPin1);
    if(sensorValue1<M1MAX && dato_puerto_serial!='O'){
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, HIGH);
    }else{
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, LOW);
    }
    }while(dato_puerto_serial!='O');
}
if (dato_puerto_serial == 'W') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='O')break;
        sensorValue1 = analogRead(sensorPin1);
        if(sensorValue1>M1MIN && dato_puerto_serial!='O'){
            digitalWrite(motor1Pin1, HIGH);
            digitalWrite(motor1Pin2, LOW);
        }else{
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, LOW);
        }
    }while(dato_puerto_serial!='O');
}
//Cuando no se recibe información útil, se recibirán 'O'
//En caso de recibir 'O' --> se paran todos los motores
if (dato_puerto_serial == 'O') {
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);
    digitalWrite(motor3Pin1, LOW);
    digitalWrite(motor3Pin2, LOW);
    digitalWrite(motor4Pin1, LOW);
    digitalWrite(motor4Pin2, LOW);
    digitalWrite(motor5Pin1, LOW);
    digitalWrite(motor5Pin2, LOW);
}
}
```

Lo que va a suceder durante el tiempo de ejecución es que, el motor se acciona mientras se haya activado su funcionamiento y se parará cuando el usuario suelte el botón o se llegue al límite establecido para ese eje. El resto de eventos para los otros cuatro motores se comportan de igual modo. Puede verse el código completo en el Anexo al documento.

### 7.5.6 Programación de Trayectorias

La programación de trayectorias se lleva a cabo de forma manual. Antes de enviar la trama de información de trayectorias, la aplicación .NET envía una solicitud a Arduino:

```
if(dato_puerto_serial=='Y'){
    Serial.print("Y-ACCEPTED");
    control_prog=1;
}
```

```
        readString();  
    }
```

Arduino responderá aceptando la comunicación de programación y llamará a la función `readString()`. A continuación se detalla el funcionamiento de dicha función.

Cada vez que Arduino se prepara para recibir una nueva programación, inicializa el vector de lectura del buffer así como el array de control de trayectorias:

```
memset(buffer, 0, sizeof(buffer));  
for(filas=0;filas<INSTRUCCIONES;filas++){  
    for(columnas=0;columnas<7;columnas++){  
        pos[filas][columnas]=0;  
    }  
}  
Serial.flush();
```

A continuación se almacenará la trama de configuración recibida en el vector. Como se verá más adelante, el carácter de fin de trama es #:

```
Do{  
    while (Serial.available(>0){  
        buffer[buffSize]=Serial.read();  
        if(buffer[buffSize]=='#'){control_prog=0;break;}  
        buffSize++;  
    }  
}while(control_prog==1);
```

Una vez recibida la cadena completa, es preciso separar la configuración de cada una de las instrucciones. En C no existe la función `SPLIT`, que permite separar una cadena de caracteres en varios elementos a partir de un elemento común. Como solución, se declaran vectores que contendrán los caracteres de cada uno de los elementos. Tal y como se especifica en el apartado anterior, se hace un `SPLIT` a la trama completa, un `SPLIT` a la orden y un `SPLIT` a la información de movimientos. Por ejemplo:

```
while ((str = strtok_r(p, "$", &p)) != NULL){  
}
```

Donde la variable "str" corresponde con un vector donde se ha almacenado la información separada, en este caso, por el carácter "\$".

Como en el vector se van a almacenar datos numéricos correspondientes a los parámetros de los sensores, se ha definido el vector como entero, por lo que se van a convertir los parámetros tipo y subtipo en valores numéricos, de tal modo que:

Tipo		
Descripción	Carácter	Valor almacenado vector
Movimiento	M	1
Pinza (mano)	H	2

Subtipo		
Descripción	Carácter	Valor almacenado vector
Cierra (pinza(0))	C	1
Cierra (pinza(1))	A	2
Posicion(0)	I	3
Posicion (1)	L	4
Posicion (2)	T	5

A continuación se detalla el código básico de la función que trata la información mencionada.

```
while ((str = strtok_r(p, "$", &p)) != NULL){
    //cadena en str
    if(postmp==0){total_progs=atoi(str);}
    else if(postmp>0 && postmp<=total_progs){
        p2=str;
        subpostmp=0;
        countbucle=0;
        while ((str2 = strtok_r(p2, "!", &p2)) != NULL){
            if(subpostmp==0){subpostmp=atoi(str2);}
            else{
                if(countbucle==1){
                    if(str2[0]=='M'){
                        pos[subpostmp-1][0]=1;
                    }else if(str2[0]=='H'){
                        pos[subpostmp-1][0]=2;
                    }
                }else if(countbucle==2){
                    if(str2[0]=='C'){
                        pos[subpostmp-1][1]=1;
                    }else if(str2[0]=='A'){
                        pos[subpostmp-1][1]=2;
                    }else if(str2[0]=='I'){
                        pos[subpostmp-1][1]=3;
                    }else if(str2[0]=='L'){
                        pos[subpostmp-1][1]=4;
                    }else if(str2[0]=='T'){
                        pos[subpostmp-1][1]=5;
                    }
                }else if(countbucle==3){
                    if(str2!=""){
                        p3=str2;
                        posmotor=2;
                        while ((str3 = strtok_r(p3, ";",
                            &p3)) != NULL){
                            pos[subpostmp-1][posmotor]
                                =atoi(str3);
                            posmotor++;
                        }//fin while str3
                    }//fin if str2=""
                }//fin countbucle==3
            }//fin subpostmp==1
            countbucle++;
        }//fin while str2
    }//fin else postmp>0 y <total_progs
    else{
        total_chars=atoi(str);
    }
}
```



```
    }  
    postmp++;  
} //fin while str
```

Una vez recibida y procesada la trama, hay que verificar si la información es correcta:

```
if(total_chars<10){  
    verifica_chars=buffSize-2;  
}else if (total_chars>=10 && total_chars<100){  
    verifica_chars=buffSize-3;  
}else if (total_chars>=100){verifica_chars=buffSize-4;}  
  
if(verifica_chars==total_chars){  
    //transmission-OK  
    Serial.print("Y-OK");  
}else{  
    //transmission-NOOK  
    Serial.print("Y-NOOK");  
}  
control_prog=0;
```

### 7.5.7 Envío información sensores

Cuando la aplicación .NET requiere la posición de los ejes (lectura sensores), se envía un requerimiento a Arduino para enviar una cadena con dicha información:

```
//Si parametro=I (Información) se muestra el valor actual de los  
sensores  
if(dato_puerto_serial == 'I'){  
    cadena="M1:"+String(sensorValue1);  
    cadena=cadena+";M2:"+String(sensorValue2);  
    cadena=cadena+";M3:"+String(sensorValue3);  
    cadena=cadena+";M4:"+String(sensorValue4);  
    cadena=cadena+";M5:"+String(sensorValue5);  
    Serial.print(cadena);  
}
```

Si el envío corresponde con la configuración de trayectorias, se envía con la cabecera "PGM@".

```
if(dato_puerto_serial == 'U'){  
    cadena="PGM@M1:"+String(sensorValue1);  
    cadena=cadena+";M2:"+String(sensorValue2);  
    cadena=cadena+";M3:"+String(sensorValue3);  
    cadena=cadena+";M4:"+String(sensorValue4);  
    cadena=cadena+";M5:"+String(sensorValue5);  
    Serial.print(cadena);  
}
```

### 7.5.8 Control Automático

Cuando se ejecuta el modo automático, se ejecutará un bucle que finalizará cuando los motores hayan alcanzado la posición indicada por la última orden de la lista de trayectorias.

El bucle se ejecutará continuamente, pasando a la siguiente orden al llegar a la posición deseada. En caso de que el robot se desplace demasiado deprisa y supere el setpoint deseado para cualquiera de los ejes, en la siguiente ejecución se ejecutará el PID en modo "REVERSE" para corregir la posición.

Según la conversión realizada a la información recibida en la trama de configuración, se ejecutara una u otra función:

```
do{  
    if(pos[control-1][0]==1){  
        //movimiento  
        Setpoint2=pos[control-1][3];  
        Setpoint3=pos[control-1][4];  
        Setpoint4=pos[control-1][5];  
        Setpoint5=pos[control-1][6];  
        if(pos[control-1][1]==3){  
            //movimiento indiferente  
            posicion(0);  
        }else if(pos[control-1][1]==4){  
            //movimiento coger  
            posicion(1);  
        }else if(pos[control-1][1]==5){  
            //movimiento vuelve de coger  
            posicion(2);  
        }  
    }else if(pos[control-1][0]==2){  
        //mano  
        if(pos[control-1][1]==1){  
            //cierra  
            pinza(1);  
        }else if(pos[control-1][1]==2){  
            //abre  
            pinza(0);  
        }  
    }  
}  
}while(control<=total_progs && control_modo==1);
```

### 7.5.9 Función Parada de Emergencia

Durante el funcionamiento en modo automático, es posible accionar la parada de emergencia, de tal forma que el robot se pare, pasando a modo manual. Para ello, la aplicación .NET envía la instrucción a Arduino, el cual para todos los motores:

```
void parada_emergencia(){  
    Serial.print("PARADA EMERGENCIA");  
}
```

```
control_mod0=0;
digitalWrite(motor1EnablePin, HIGH);
digitalWrite(motor2EnablePin, HIGH);
digitalWrite(motor3EnablePin, HIGH);
digitalWrite(motor4EnablePin, HIGH);
digitalWrite(motor5EnablePin, HIGH);
digitalWrite(motor1Pin1, LOW);
digitalWrite(motor1Pin2, LOW);
digitalWrite(motor2Pin1, LOW);
digitalWrite(motor2Pin2, LOW);
digitalWrite(motor3Pin1, LOW);
digitalWrite(motor3Pin2, LOW);
digitalWrite(motor4Pin1, LOW);
digitalWrite(motor4Pin2, LOW);
digitalWrite(motor5Pin1, LOW);
digitalWrite(motor5Pin2, LOW);
control=1;
}
```

Además, para confirmar a .NET que el robot se ha parado, envía una cadena de control.

#### **7.5.10 Función control Mano – Modo Automático**

A continuación se detalla el funcionamiento de la función de control de la pinza. Como parámetros de entrada puede recibir un 0 o un 1, según se desee abrir o cerrar la mano respectivamente:

```
void pinza(int controlpinza){
    int c1=0;
    if (controlpinza==1){
        Setpoint1=700;
    }else{
        Setpoint1=810;
    }
    if(Setpoint1>Setpoint_old1){
        myPID1.SetControllerDirection(DIRECT);
    }else{
        myPID1.SetControllerDirection(REVERSE);
    }
    if((analogRead(sensorPin1)>(Setpoint1+15))||(analogRead(sensorPin1)<(Setpoint1-15))){
        giral(sensorPin1, motor1Pin1, motor1Pin2, 30,
        motor1EnablePin, Setpoint1,myPID1);
        c1=0;
    }else{
        Setpoint_old1=analogRead(sensorPin1);
        digitalWrite(motor1Pin1,LOW);
        digitalWrite(motor1Pin2,LOW);
        analogWrite(motor1EnablePin,0);
        c1=1;
    }
    if(c1==1){
        control=control+1;
    }
}
```



Cuando se ejecuta esta función, se establece el setpoint deseado. Según la posición actual y la posición deseada, se configurará el PID, de tal modo que, si al aproximarse a una posición, es superada, el PID actuará para contrarrestar el error a una velocidad proporcional al error.

Para accionar el control, se ejecuta la función “gira1”.

```
void gira1(int id_sensor, int motor_dcha, int motor_izq, int
reduccion, int id_enable, double Setpoint, PID myPID){
  Input1 = analogRead(id_sensor);
  //distancia entre el punto actual y el definitivo
  double gap = abs(Setpoint-Input1);

  if(Setpoint>Input1){
    digitalWrite(motor_dcha,LOW);
    digitalWrite(motor_izq,HIGH);
  }else{
    digitalWrite(motor_izq,LOW);
    digitalWrite(motor_dcha,HIGH);
  }
  //si estamos cerca del punto, reducimos parametro PID

  if(gap<reduccion){
    myPID.SetTunings(consKp1, consKi1, consKd1);
  }else{
    //si estamos alejados, parametros mas agresivos para PID
    myPID.SetTunings(aggKp1, aggKi1, aggKd1);
  }

  myPID.Compute();

  var=analogRead(id_sensor);

  if(var!=var_old){
    var2=Output1;
    var_old=var;
  }
  analogWrite(id_enable,Output1);
}
```

Como puede observarse, al aproximarse al setpoint deseado, los parámetros del PID serán más suaves.

### **7.5.11 Función control posición – Modo Automático**

Para controlar la posición de los distintos ejes, se ha diseñado una función acorde a los tres modos de desplazamientos.

```
void posicion (int control_trayectoria){
  int c2=0;
  int c3=0;
  int c4=0;
  int c5=0;
  int control_prioridad;
```



```
if(control_trayectoria==1){
  control_prioridad=1;
}else if(control_trayectoria==2){
  control_prioridad=0;
}else{
  control_prioridad=2;
}
do{
  dato_puerto_serial = Serial.read();
  if(dato_puerto_serial == 'P'){
    parada_emergencia();
    break;
  }
}
if(control_prioridad==0 || control_prioridad==2){
  if(Setpoint2>Setpoint_old2){
    myPID2.SetControllerDirection(DIRECT);
  }else{
    myPID2.SetControllerDirection(REVERSE);
  }
}

if((analogRead(sensorPin2)>(Setpoint2+5)) || (analogRead(sensorPin2)<(Setpoint2-5))){
  gira2(sensorPin2, motor2Pin1, motor2Pin2, 30,
  motor2EnablePin, Setpoint2,myPID2);
  c2=0;
}else{
  Setpoint_old2=analogRead(sensorPin2);
  digitalWrite(motor2Pin1,LOW);
  digitalWrite(motor2Pin2,LOW);
  analogWrite(motor2EnablePin,0);
  c2=1;
}
if(Setpoint3>Setpoint_old3){
  myPID3.SetControllerDirection(DIRECT);
}else{
  myPID3.SetControllerDirection(REVERSE);
}

if((analogRead(sensorPin3)>(Setpoint3+5)) || (analogRead(sensorPin3)<(Setpoint3-5))){
  gira3(sensorPin3, motor3Pin1, motor3Pin2, 100,
  motor3EnablePin, Setpoint3,myPID3);
  c3=0;
}else{
  Setpoint_old3=analogRead(sensorPin3);
  digitalWrite(motor3Pin1,LOW);
  digitalWrite(motor3Pin2,LOW);
  analogWrite(motor3EnablePin,0);
  c3=1;
}
if(Setpoint4>Setpoint_old4){
  myPID4.SetControllerDirection(DIRECT);
}else{
  myPID4.SetControllerDirection(REVERSE);
}

if((analogRead(sensorPin4)>(Setpoint4+5)) || (analogRead(sensorPin4)<(Setpoint4-5))){
  gira4(sensorPin4, motor4Pin1, motor4Pin2, 100,
```





```
        motor4EnablePin, Setpoint4,myPID4);
        c4=0;
    }else{
        Setpoint_old4=analogRead(sensorPin4);
        digitalWrite(motor4Pin1,LOW);
        digitalWrite(motor4Pin2,LOW);
        analogWrite(motor4EnablePin,0);
        c4=1;
    }
    if(c2==1 && c3==1 && c4==1){
        control_prioridad=2;
    }
} //fin control prioridad==0

if(control_prioridad==1 || control_prioridad==2){
    if(Setpoint5>Setpoint_old5){
        myPID5.SetControllerDirection(DIRECT);
    }else{
        myPID5.SetControllerDirection(REVERSE);
    }

    if((analogRead(sensorPin5)>(Setpoint5+5)) || (analogRead(sensorPin5)<(Setpoint5-5))){
        gira5(sensorPin5, motor5Pin1, motor5Pin2, 100,
        motor5EnablePin, Setpoint5,myPID5);
        c5=0;
    } else{
        Setpoint_old5=analogRead(sensorPin5);
        digitalWrite(motor5Pin1,LOW);
        digitalWrite(motor5Pin2,LOW);
        analogWrite(motor5EnablePin,0);
        c5=1;
    }
    if(c5==1){
        control_prioridad=2;
    }
}

} //fin control prioridad==0

if(c2==1 && c3==1 && c4==1 && c5==1){
    control=control+1;
}

}while(c2!=1 || c3!=1 || c4!=1 || c5!=1);
}
```

Como puede observarse, la filosofía de desplazamiento de cada motor es la misma que la descrita para la mano pero, según el valor de entrada de la función, se dará prioridad a un movimiento y otro.

## 7.6 SOFTWARE .NET

La aplicación .NET es el entorno gráfico de interacción con el usuario. Corresponde con un archivo ejecutable .exe que funciona bajo sistemas operativos Windows.

Como requisito indispensable para la ejecución del mismo, es preciso instalar el Framework .NET 4 o superior en el sistema operativo.

La versión mínima del Sistema Operativo en la que se asegura que la aplicación funciona correctamente es Windows 7.



Figura 41: entorno gráfico .NET

### 7.6.1 Introducción

La estructura general de la aplicación corresponde con un formulario (ventana) y una clase asociada que engloba todas las funciones propias declaradas.

Como curiosidad, cabe destacar la diferencia entre los eventos de los botones:

- Onclick: se ejecuta al hacer click sobre sobre el botón y una vez que se ha soltado el botón del ratón.
- Onmousedown: se ejecuta al hacer click sobre el botón.
- Onmouseup: se ejecuta al soltar el botón del ratón tras hacer clic sobre el botón.

Existen otros muchos eventos pero, para la aplicación diseñada, solo se necesitan los descritos.

### 7.6.2 Librerías

Para interactuar con el puerto serie se han cargado las siguientes librerías:



```
Imports System.IO
Imports System.IO.Ports
```

### 7.6.3 Variables y Constantes

Se han definido unas constantes para control de algunos elementos:

Declaración y definición del puerto serie:

```
Shared _serialPort As SerialPort
Dim entradadedatos As String
Dim count As Integer
Dim valor_enviado As String
Dim estado_led As Integer
Dim estado_mod0 As Integer
Dim estado_arduino As Integer
Dim arduino_programado As Integer
Dim estado_programando As Integer
Dim contador_movimientos As Integer
Dim control_prog_accion As Integer
Dim cadena_control_errores_recepcion As String
Dim intervalo_lectura_prog As Integer
Const total_movimientos_permitidos As Integer = 25
Const DELIMITADOR As String = "#"
```

### 7.6.4 Inicialización Formulario

Al cargar el formulario (ventana), se inicializan las variables utilizadas, dejando la aplicación a la espera de conexión a un puerto serie. El usuario podrá escoger el puerto serie al que desee conectar dentro de un listado. Si el puerto serie está disponible y corresponde con Arduino, se habilitarán los controles del mismo.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
modo_trabajo_label.Text = "Esperando Conexión"
control_prog_accion = 0
contador_movimientos = 0
arduino_programado = 0
estado_programando = 0
Timer1.Enabled = False
boton_led.Visible = True
boton_led2.Visible = False
estado_led = 0
'boton_mod0.Text = "MODO AUTOMATICO"
boton_mod0.Visible = True
boton_mod02.Visible = False
estado_mod0 = 0
GetSerialPortNames()
MessageBox.Show("Por favor, selecciona el puerto de
conexión y pulsa conectar en el menú de control")
deshabilita_funciones()
End Sub
```



La función que deshabilita todas las funcionalidad mientras no esté conectado a Arduino:

```
Sub deshabilita_funciones()  
    boton_g.Enabled = False  
    boton_w.Enabled = False  
    boton_led.Enabled = False  
    boton_led2.Enabled = False  
    boton_e.Enabled = False  
    boton_r.Enabled = False  
    boton_d.Enabled = False  
    boton_x.Enabled = False  
    boton_s.Enabled = False  
    boton_a.Enabled = False  
    boton_f.Enabled = False  
    boton_z.Enabled = False  
    boton_mod0.Enabled = False  
    boton_mod2.Enabled = False  
    boton_parada.Enabled = False  
    cierra_mano.Enabled = False  
    abre_mano.Enabled = False  
    boton_guardapos_0.Enabled = False  
    boton_guardapos_1.Enabled = False  
    boton_guardapos_2.Enabled = False  
    boton_elimina_item.Enabled = False  
    envia_programacion.Enabled = False  
    boton_papelera.Enabled = False  
    boton_salvar_configuracion.Enabled = False  
    boton_cargar_desde_archivo.Enabled = False  
    boton_info.Enabled = False  
    boton_copia_consola.Enabled = False  
  
End Sub
```

La función que consulta el listado de puertos Serie y los muestra en un menú lista:

```
Sub GetSerialPortNames()  
    For Each sp As String In My.Computer.Ports.SerialPortNames  
        ListBox1.Items.Add(sp)  
    Next  
End Sub
```

### **7.6.5 Conexión con Arduino**

Durante la inicialización del formulario, se ha cargado el listado de puertos serie disponible en el elemento ListBox1. Una vez seleccionado un elemento de dicho Listbox1, se habilitará el botón conectar:

```
Private Sub ListBox1_SelectedIndexChanged(sender As Object, e As  
EventArgs) Handles ListBox1.SelectedIndexChanged  
    boton_conectar.Visible = True  
End Sub
```

Al hacer clic sobre el botón conectar, se configuran los parámetros correspondientes al puerto serie. Es preciso que el baudrate corresponda con el especificado en Arduino.



```
Private Sub boton_conectar_Click(sender As Object, e As
EventArgs) Handles boton_conectar.Click
    estado_modo = 0
    SerialPort1.Close()
    SerialPort1.PortName = ListBox1.SelectedItem.ToString
    SerialPort1.BaudRate = 9600
    SerialPort1.DataBits = 8
    SerialPort1.Parity = Parity.None
    SerialPort1.StopBits = StopBits.One
    SerialPort1.Handshake = Handshake.None
    Try
        SerialPort1.Open()
        valor_enviado = "M"
        Timer1.Enabled = True
        If SerialPort1.IsOpen Then
            SerialPort1.Write(valor_enviado.ToString)
            consola.Text = "PC> " & "REQUEST CONNECT" &
vbCrLf & consola.Text
            boton_conectar.Visible = False
            boton_parada.Visible = False
            ListBox1.Visible = False
            estado_conexion.Text = "Estado: conectando..."
            estado_arduino = 1
            estado_conexion.Visible = True
            Timer2.Enabled = True
            modo_trabajo_label.Text = "MANUAL"
        End If
    Catch ex As Exception
        MessageBox.Show("Error conectando al puerto serie
seleccionado. Selecciona otro puerto serie.")
    End Try

End Sub
```

En caso de seleccionar un puerto no disponible se mostrará un mensaje de error.

Del mismo modo, si el puerto serie está disponible pero no corresponde con Arduino o éste no responde, se mostrará un mensaje de error, simulando un timeout. Para ello se ha creado un elemento Timer2, con una espera de 5 segundos, que verifica si Arduino ha respondido la petición:

```
Private Sub Timer2_Tick(sender As Object, e As EventArgs) Handles
Timer2.Tick
    If estado_arduino = 2 Then
        Timer2.Enabled = False
    ElseIf estado_arduino = 1 Then
        Timer2.Enabled = False
        MessageBox.Show("No se ha encontrado Arduino en el puerto
seleccionado. Selecciona otro puerto serie.")
        boton_conectar.Visible = True
        ListBox1.Visible = True
        estado_conexion.Visible = False
    Else
        Timer2.Enabled = False
        MessageBox.Show("No se ha encontrado Arduino en el puerto
seleccionado. Selecciona otro puerto serie.")
        boton_conectar.Visible = True
    End If
End Sub
```



```
        ListBox1.Visible = True
        estado_conexion.Visible = False
    End If
End Sub
```

Si el Puerto está disponible, se habilita la lectura continuada del Puerto serie mediante un Timer configurado a 100 milisegundos. Éste será el encargado de tratar la información recibida mediante el puerto y su funcionamiento se detalla en el siguiente apartado.

### **7.6.6 Lectura puerto serie**

La lectura del puerto serie se lleva a cabo a partir de un timer que se ejecuta ilimitadamente, leyendo la información recibida a través del puerto serie. Es éste evento el que controla la información recibida y sus acciones correspondientes.

Su funcionamiento es simple: compara la cadena o cabecera de la cadena con algunos patrones predefinidos y actúa en consecuencia.

Además, éste es el evento que se comunica con Arduino para transmitir la programación de trayectorias. Cabe destacar el proceso de transmisión de configuración de trayectorias a Arduino:

Inicialmente, al pulsar sobre el botón programar, se envía un carácter solicitando a Arduino entrar en su modo de programación. Cuando Arduino confirma su espera, envía la cadena "Y-ACCEPTED" a través del puerto Serie. En ese momento, se ejecuta la función `obtiene_info_prog()`, que genera la trama a enviar a Arduino.

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
    Dim tabla2() As String
    Dim tabla() As String
    Dim subtabla() As String
    Dim n As Integer
    Dim lectura As String = SerialPort1.ReadExisting
    Dim verifica As Integer
    Dim verifica2 As Integer
    Dim verifica3 As Integer
    Dim verifica4 As Integer
    Dim variableins As String
    Dim verifica_validez_registro As Integer
    verifica_validez_registro = 0
    If lectura <> "" Then
        consola.Text = "ARDUINO> " & lectura & vbCrLf &
        consola.Text
    End If
    verifica = InStr(lectura, "PGM")
    verifica2 = InStr(lectura, "Y-ACCEPTED")
    verifica3 = InStr(lectura, "Y-OK")
    verifica4 = InStr(lectura, "Y-NOOK")
    If verifica <> 0 And estado_programando = 0 Then
        tabla2 = Split(lectura, "@")
        If UBound(tabla2, 1) <> 1 Or verifica <> 1 Then
```



```
valor_enviado = "U"
SerialPort1.Write(valor_enviado.ToString)
consola.Text = "PC> " & "ERROR RECEIVE. GET VALUES TO
ADD" & vbCrLf & consola.Text
Else
tabla = Split(tabla2(1), ";")
If UBound(tabla, 1) <> 4 Then
valor_enviado = "U"
SerialPort1.Write(valor_enviado.ToString)
consola.Text = "PC> " & "ERROR RECEIVE. GET VALUES
TO ADD" & vbCrLf & consola.Text
Else
If intervalo_lectura_prog = 0 Then
cadena_control_errores_recepcion = lectura
intervalo_lectura_prog =
intervalo_lectura_prog + 1
valor_enviado = "U"
SerialPort1.Write(valor_enviado.ToString)
Else
If cadena_control_errores_recepcion = lectura
Then
verifica_validez_registro = 1
Else
consola.Text = "PC> " & "ERROR RECEIVING
INFO. RESTARTING RECEPTION" & vbCrLf &
consola.Text
'Se ha producido un error en la recepción
de la primera o la segunda trama
'Inicializo y programo de nuevo
intervalo_lectura_prog = 0
verifica_validez_registro = 0
valor_enviado = "U"
SerialPort1.Write(valor_enviado.ToString)
End If
End If
If verifica_validez_registro = 1 Then
If control_prog_accion = 0 Then
variableins = "IND"
ElseIf control_prog_accion = 1 Then
variableins = "COG"
ElseIf control_prog_accion = 2 Then
variableins = "REG"
Else
variableins = ""
End If
If variableins <> "" Then
contador_movimientos =
contador_movimientos + 1
tabla2(1) = tabla2(1).Replace("M1:", "")
tabla2(1) = tabla2(1).Replace("M2:", "")
tabla2(1) = tabla2(1).Replace("M3:", "")
tabla2(1) = tabla2(1).Replace("M4:", "")
tabla2(1) = tabla2(1).Replace("M5:", "")

DataGridView1.Rows.Add(contador_movimientos.ToS
tring, "MOV", variableins, tabla2(1))
End If
End If
End If
End If
```



```
ElseIf verifica2 <> 0 Then
    valor_enviado = obtiene_info_prog()
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND B" & vbCrLf & consola.Text
    'consola.Text = "PC> " & valor_enviado.ToString
ElseIf verifica3 <> 0 Then
    MessageBox.Show("Transferencia completada con éxito")
    arduino_programado = 1
    estado_programando = 0
ElseIf verifica4 <> 0 Then
    MessageBox.Show("No ha sido posible programar arduino en
    éste momento. Reintentalo y, si el problema persiste,
    reinicie ambas unidades")
    arduino_programado = 0
    estado_programando = 0
ElseIf lectura = "CONNECTED" And estado_programando = 0 Then
    estado_conexion.Text = "Estado: CONECTADO"
    estado_conexion.Visible = True
    estado_arduino = 2
    habilita_funciones()
ElseIf lectura = "PARADA EMERGENCIA" And estado_programando =
0 Then
    estado_conexion.Text = "Estado: CONECTADO (P)"
    boton_modos2.Visible = False
    boton_modos.Visible = True
    boton_parada.Visible = False
    modo_trabajo_label.Text = "MANUAL"
    cambia_controles_programacion(True)
    'boton_modos.Text = "MODO AUTOMATICO"
    estado_modos = 0
ElseIf lectura <> "" And estado_programando = 0 And lectura <>
"CONNECTED" And verifica = 0 And verifica2 = 0 And verifica3 =
0 And verifica4 = 0 And lectura <> "PARADA EMERGENCIA" Then
    tabla = Split(lectura, ";")
    If UBound(tabla, 1) <> 4 Then
        valor_enviado = "I"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "ERROR RECEIVE. GET INFO" &
        vbCrLf & consola.Text
    Else
        For n = 0 To UBound(tabla, 1)
            subtabla = Split(tabla(n), ":")
            If subtabla(0) = "M5" Then
                M5_state.Text = subtabla(1)
            End If
            If subtabla(0) = "M4" Then
                M4_state.Text = subtabla(1)
            End If
            If subtabla(0) = "M3" Then
                M3_state.Text = subtabla(1)
            End If
            If subtabla(0) = "M2" Then
                M2_state.Text = subtabla(1)
            End If
            If subtabla(0) = "M1" Then
                M1_state.Text = subtabla(1)
            End If
        Next
    End If
Next
```



```
        End If
    End If
End Sub
```

### 7.6.7 Control manual de motores

En el modo manual, los motores se activan en el evento onmousedown y se desactivan en el evento onmouseup. Por ejemplo:

```
Private Sub boton_x_MouseDown(sender As Object, e As
MouseEventArgs) Handles boton_x.MouseDown
    valor_enviado = "X"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND BASE X" & vbCrLf &
consola.Text
End Sub

Private Sub boton_x_MouseUp(sender As Object, e As
MouseEventArgs) Handles boton_x.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND BASE X STOP" & vbCrLf &
consola.Text
End Sub
```

### 7.6.8 Definición de posiciones y eventos en trayectorias

Tal y como se especificaba anteriormente, se han definido tres tipos de desplazamientos así como los eventos “apertura de mano” y “cierre de mano”.

En modo manual es posible crear una trayectoria. Tan sólo es necesario desplazar Arduino a las posiciones deseadas y hacer clic sobre el tipo de comportamiento que deseamos tenga para llegar a ese punto. Estos movimientos son compatibles con apertura y cierre de la mano.

Para ello, se han insertado 3 botones de movimiento y 2 de control de la mano. Al hacer clic sobre ellos, se obtienen los parámetros de posicionamiento si es necesario – se envía un parámetro de obtención de información a Arduino y se procesa mediante Timer1 programado para la lectura continuada del puerto serie -.

El código fuente de los botones de control de la mano:

```
Private Sub cierra_mano_Click(sender As Object, e As EventArgs)
Handles cierra_mano.Click
    If (contador_movimientos < total_movimientos_permitidos)
    Then
        contador_movimientos = contador_movimientos + 1
        DataGridView1.Rows.Add (contador_movimientos
.ToString, "MANO", "CIERRA", "")
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
movimientos programados")
    End If
End Sub
```



```
Private Sub abre_manos_Click(sender As Object, e As EventArgs)
Handles abre_manos.Click
    If (contador_movimientos < total_movimientos_permitidos)
    Then
        contador_movimientos = contador_movimientos + 1
        DataGridView1.Rows.Add(contador_movimientos
        .ToString, "MANO", "ABRE", "")
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
        movimientos programados")
    End If
End Sub
```

El elemento DataGridView1 corresponde con una tabla mostrada en pantalla que almacena la información en el formato de orden definido en la trama: tipo de movimiento, subtipo de movimiento y valor de los sensores (si corresponde).

Para el caso de los botones de movimiento:

```
Private Sub boton_guardapos_0_Click(sender As Object, e As
EventArgs) Handles boton_guardapos_0.Click
    If (contador_movimientos < total_movimientos_permitidos)
    Then
        intervalo_lectura_prog = 0
        control_prog_accion = 0
        valor_enviado = "U"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "GET VALUES TO ADD POS 0" &
        vbCrLf & consola.Text
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
        movimientos programados")
    End If
End Sub
```

```
Private Sub boton_guardapos_1_Click(sender As Object, e As
EventArgs) Handles boton_guardapos_1.Click
    If (contador_movimientos < total_movimientos_permitidos)
    Then
        intervalo_lectura_prog = 0
        control_prog_accion = 1
        valor_enviado = "U"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "GET VALUES TO ADD POS 1" &
        vbCrLf & consola.Text
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
        movimientos programados")
    End If
End Sub
```

```
Private Sub boton_guardapos_2_Click(sender As Object, e As
EventArgs) Handles boton_guardapos_2.Click
    If (contador_movimientos < total_movimientos_permitidos)
    Then
        intervalo_lectura_prog = 0
        control_prog_accion = 2
```



```
        valor_enviado = "U"  
        SerialPort1.Write(valor_enviado.ToString)  
        consola.Text = "PC> " & "GET VALUES TO ADD POS 2" &  
        vbCrLf & consola.Text  
    Else  
        MessageBox.Show("Ha alcanzado el número máximo de  
        movimientos programados")  
    End If  
End Sub
```

Cuando un botón de posición solicita la posición a Arduino, se envía el parámetro “U” y se recupera la información a través de Timer1 con la cabecera “PGM”.

La información de posición de los sensores de solicita a Arduino de forma duplicada, del tal forma que se comparan ambas tramas de transmisión. Si no coinciden, se repetirá el bucle. En caso de error, se mostrará en la consola.

### 7.6.9 Envío configuración trayectorias

Cuando Arduino confirma a .NET que entra en estado “programación”, se ejecuta la función que crea la trama a enviar. Previamente se ha verificado que el usuario ha creado una trayectoria.

```
Private Sub envia_programacion_Click(sender As Object, e As  
EventArgs) Handles envia_programacion.Click  
    'cadena_programacion = obtiene_info_prog()  
    If contador_movimientos > 0 Then  
        'Si es el primer intento  
        If estado_programando = 0 Then  
            estado_programando = 1  
            valor_enviado = "Y"  
            SerialPort1.Write(valor_enviado.ToString)  
            consola.Text = "PC> " & "PROGRAMMING REQUEST  
            COM" & vbCrLf & consola.Text  
        Else  
            'Si se produce algún error intentando programar  
            el arduino  
            valor_enviado = obtiene_info_prog()  
            SerialPort1.Write(valor_enviado.ToString)  
            consola.Text = "PC> " & "SEND B" & vbCrLf &  
            consola.Text  
        End If  
    Else  
        MessageBox.Show("Antes de enviar la programación,  
        genere una trayectoria de automatización")  
    End If  
End Sub
```

La función `obtiene_info_prog()` es la encargada de generar la trama de envío. Para ello recorre todos los elementos de la tabla `Datagridview1` y añade los separadores oportunos.

```
Private Function obtiene_info_prog() As String
```



```
Dim cadena As String
Dim contador_caracteres As Integer
Dim tipo As String
Dim subtipo As String
'Formato Trama
'Contador movimientos&orden1&orden2&total_caracteres#
'Separador instrucciones $
'Separador parametros orden !
'Fin trama #
cadena = contador_movimientos.ToString

For Each row As DataGridViewRow In DataGridView1.Rows
  If Not row.IsNewRow Then
    If row.Cells(1).Value.ToString = "MOV" Then
      tipo = "M"
    ElseIf row.Cells(1).Value.ToString = "MANO" Then
      tipo = "H"
    Else
      tipo = ""
    End If
    If row.Cells(2).Value.ToString = "CIERRA" Then
      subtipo = "C"
    ElseIf row.Cells(2).Value.ToString = "ABRE" Then
      subtipo = "A"
    ElseIf row.Cells(2).Value.ToString = "IND" Then
      subtipo = "I"
    ElseIf row.Cells(2).Value.ToString = "COG" Then
      subtipo = "L"
    ElseIf row.Cells(2).Value.ToString = "REG" Then
      subtipo = "T"
    Else
      subtipo = ""
    End If
    cadena = cadena & "$" &
    row.Cells(0).Value.ToString & "!" & tipo & "!" &
    subtipo & "!" & row.Cells(3).Value.ToString

  End If
Next
contador_caracteres = Len(cadena)
cadena = cadena & "$" & contador_caracteres & "#"
Return cadena
End Function
```

### **7.6.10 Almacenamiento información trayectorias, carga y borrado de eventos**

Se han diseñado diversas funcionalidades con el fin de facilitar la creación de trayectorias, carga y almacenamiento de las mismas por parte del usuario:

- Botón guardar: almacena la información de la trayectoria en formato csv.
- Botón cargar: carga un archivo de trayectorias almacenado previamente.
- Botón eliminar trayectorias: elimina el listado de trayectorias del DataGridView1.
- Botón suprimir evento: elimina el último evento insertado en el listado DataGridView1.



Al hacer clic sobre el botón “salvar”, y siempre que se hayan insertado eventos a la trayectoria, se ofrecerá al usuario la opción de seleccionar un nombre para el archivo. La extensión con la que se almacenan los archivos de trayectorias es .jyn.

```
Private Sub boton_salvar_configuracion_Click(sender As Object, e
As EventArgs) Handles boton_salvar_configuracion.Click
    Dim saveFileDialog1 As New SaveFileDialog()
    saveFileDialog1.Filter = "arduino|*.jyn"
    saveFileDialog1.Title = "Grabar configuración"
    saveFileDialog1.ShowDialog()

    If saveFileDialog1.FileName <> "" Then
        Try
            Using archivo As StreamWriter = New
                StreamWriter(saveFileDialog1.FileName)

                Dim linea As String = String.Empty
                With DataGridView1
                    For fila As Integer = 0 To .RowCount - 1
                        linea = String.Empty
                        For col As Integer = 0 To
                            .Columns.Count - 1
                                linea = linea & .Item(col,
                                    fila).Value.ToString & DELIMITADOR
                            Next
                        With archivo
                            .WriteLine(linea.ToString)
                        End With
                    Next
                End With
            End Using

            'error
            Catch ex As Exception
                MsgBox(ex.Message.ToString,
                    MsgBoxStyle.Critical)
            End Try
        End If
    End Sub
```

Al hacer clic sobre el botón “cargar”, se elimina cualquier información de eventos creada y se importa el archivo. Se ofrecerá la opción de selección de archivo al usuario:

```
Private Sub boton_cargar_desde_archivo_Click(sender As Object, e
As EventArgs) Handles boton_cargar_desde_archivo.Click
    Dim openFileDialog1 As New OpenFileDialog()
    Dim fieldValues As String()
    Dim miReader As IO.StreamReader
    openFileDialog1.Filter = "arduino|*.jyn"
    openFileDialog1.Title = "Cargar configuración"

    If MsgBox("¿Está seguro que desea eliminar la
        programación completa y cargarla desde un archivo de
        texto?", MsgBoxStyle.YesNo, "Confirmación") =
        MsgBoxResult.Yes Then
        If DataGridView1.Rows.Count > 0 Then
            DataGridView1.Rows.Clear()
            contador_movimientos = 0
        End If
    End Sub
```



```
End If
openFileDialog1.ShowDialog()
If openFileDialog1.FileName <> "" Then
    Try
        miReader =
            File.OpenText(openFileDialog1.FileName)

        While miReader.Peek() <> -1
            fieldValues =
                miReader.ReadLine().Split(DELIMITADOR)

            DataGridView1.Rows.Add(fieldValues(0).ToS
                tring, fieldValues(1).ToString,
                fieldValues(2).ToString,
                fieldValues(3).ToString)
            contador_movimientos =
                contador_movimientos + 1
        End While
        miReader.Close()

        'error
        Catch ex As Exception
            MsgBox(ex.Message.ToString,
                MsgBoxStyle.Critical)
        End Try
    End If
End If

End Sub
```

Los botones para eliminar toda la información de trayectorias, así como para eliminar el último evento:

```
Private Sub boton_papelera_Click(sender As Object, e As
EventArgs) Handles boton_papelera.Click
    If DataGridView1.Rows.Count > 0 Then
        If MsgBox("¿Está seguro que desea eliminar la
programación completa?", MsgBoxStyle.YesNo,
"Confirmación") = MsgBoxResult.Yes Then
            DataGridView1.Rows.Clear()
            contador_movimientos = 0
        End If
    End If
End Sub

Private Sub boton_elimina_item_Click(sender As Object, e As
EventArgs) Handles boton_elimina_item.Click
    If DataGridView1.Rows.Count > 0 Then
        If MsgBox("¿Está seguro que desea eliminar el último
elemento insertado?", MsgBoxStyle.YesNo,
"Confirmación") = MsgBoxResult.Yes Then
            DataGridView1.Rows.RemoveAt(contador_movimientos
- 1)
            contador_movimientos = contador_movimientos - 1
        End If
    End If
End Sub
```

### 7.6.11 Cambio Modo Manual / Automático

Es posible alternar entre el modo manual y el automático haciendo clic sobre el botón habilitado para tal función. Sin embargo, no es posible cambiar al modo automático si previamente no se ha cargado una configuración de trayectorias en Arduino.

Inicialmente, cuando arranca la aplicación, el modo únicamente será Manual debido a la falta de programación inicial.

```
Private Sub boton_mod0_MouseDown(sender As Object, e As
MouseEventArgs) Handles boton_mod0.MouseDown
    'SerialPort1.Open()
    If arduino_programado = 1 Then
        valor_enviado = "N"
        estado_mod0 = 1
        'boton_mod0.Text = "MODO MANUAL"
        boton_mod02.Visible = True
        boton_parada.Visible = True
        modo_trabajo_label.Text = "AUTO"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "SEND AUTO MODE" & vbCrLf &
        consola.Text
        boton_mod0.Visible = False
        cambia_controles_programacion(False)
    Else
        MessageBox.Show("No puede cambiar a modo manual
        mientras no envíe una programación de automatización a
        arduino")
    End If
End Sub

End Sub

Private Sub boton_mod02_MouseDown(sender As Object, e As
MouseEventArgs) Handles boton_mod02.MouseDown
    'SerialPort1.Open()
    valor_enviado = "M"
    estado_mod0 = 0
    'boton_mod0.Text = "MODO AUTOMATICO"
    boton_mod0.Visible = True
    boton_parada.Visible = False
    modo_trabajo_label.Text = "MANUAL"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND MANUAL MODE" & vbCrLf &
    consola.Text
    boton_mod02.Visible = False
    cambia_controles_programacion(True)
End Sub
```

### 7.6.12 Parada de Emergencia en modo Automático

Al hacer clic sobre el botón "Parada de Emergencia" se enviará un carácter de parada a Arduino.



```
Private Sub boton_parada_MouseDown(sender As Object, e As  
MouseEventArgs) Handles boton_parada.MouseDown  
    valor_enviado = "P"  
    SerialPort1.Write(valor_enviado.ToString)  
    consola.Text = "PC> " & "STOP ALL INSTRUCTIONS" & vbCrLf  
    & consola.Text  
End Sub
```

Cuando Arduino recibe la solicitud de parada de emergencia, la ejecuta y envía la confirmación de la parada, la cual es recogida a través de Timer1 y actúa:

```
estado_conexion.Text = "Estado: CONECTADO (P)"  
boton_modos2.Visible = False  
boton_modos.Visible = True  
boton_parada.Visible = False  
modo_trabajo_label.Text = "MANUAL"  
cambia_controles_programacion(True)  
estado_modos = 0
```

### **7.6.13 Consola de transmisión y detección de errores**

Se ha creado un elemento Textbox en el que se muestra la información enviada y recibida a través del puerto Serie. Además, se ha incluido un botón para copiar al portapapeles toda la información recogida en la misma.



## 8. GUIA DE USUARIO

### 8.1 INTRODUCCIÓN

En este capítulo se detallan las pautas a seguir para la puesta en marcha del brazo robot, el funcionamiento de los modos manual y automático así como la configuración de trayectorias. Es importante que el sistema creado se manipule única y exclusivamente por personal cualificado para garantizar la seguridad del sistema.

La fiabilidad y el buen funcionamiento del sistema dependen de la fidelidad con que se sigan las instrucciones indicadas a lo largo del presente capítulo.

### 8.2 DESCRIPCIÓN DEL SISTEMA

El sistema de control del Brazo Robot Velleman KSR10 está compuesto por el propio brazo robot y su controladora así como por ordenador de control.

Los requerimientos de software son, aparte de tener instalada la aplicación de control, un Sistema Operativo Windows 7 o superior y la versión Framework .NET 4 o superior. Es posible descargar Framework .NET en la página de Microsoft: <http://www.microsoft.com/es-es/download/details.aspx?id=30653>

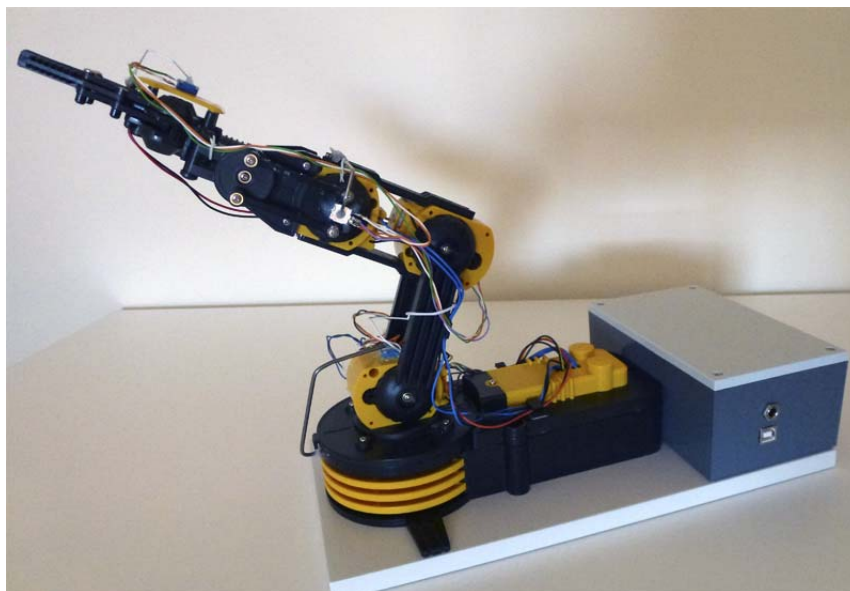


Figura 42: brazo robot Velleman KSR10 y su controladora

El brazo robot dispone de una conexión de alimentación 6V 1000mA y una conexión USB que comunicará su controladora con el ordenador de control.

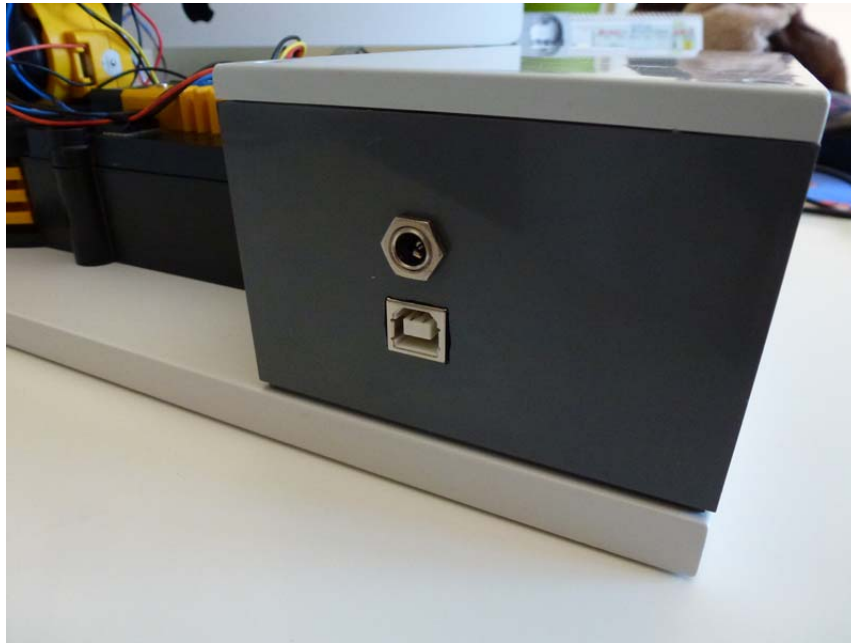


Figura 43: conexión de alimentación y USB del Brazo Robot en la parte trasera

Además, el brazo robot posee un botón reset a utilizar en caso de emergencia, siempre y cuando no responda el botón de parada de emergencia del ordenador o se produzca algún bloqueo o error de comunicación entre ambos dispositivos.

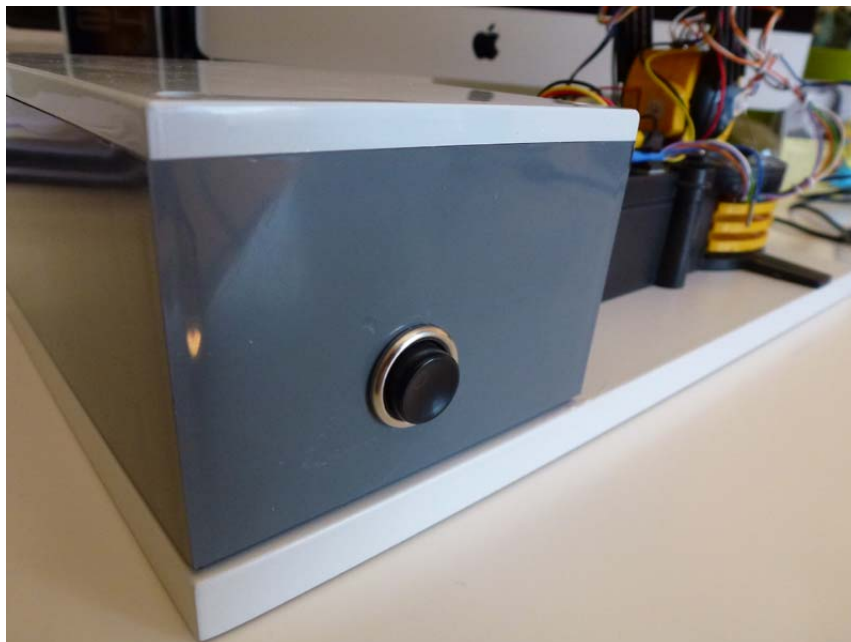


Figura 44: botón reset del Brazo Robot en la parte frontal

### 8.3 PUESTA EN MARCHA

Para la puesta en marcha del dispositivo, conecte la fuente de alimentación de 6V 1000mA a la controladora del Brazo Robot y a la red eléctrica.

A continuación, con el cable USB, comuníquese la controladora con el PC.

La primera vez que se conecte el brazo robot al ordenador, el Sistema Operativo pedirá autorización para instalar un nuevo hardware. Es posible seleccionar la instalación automática del dispositivo tras lo cual, debería ser instalado y reconocido correctamente por el Sistema Operativo Windows.

En la ruta Panel de Control >> Hardware y Sonido >> Administrador de dispositivos podemos verificar la información de Arduino:

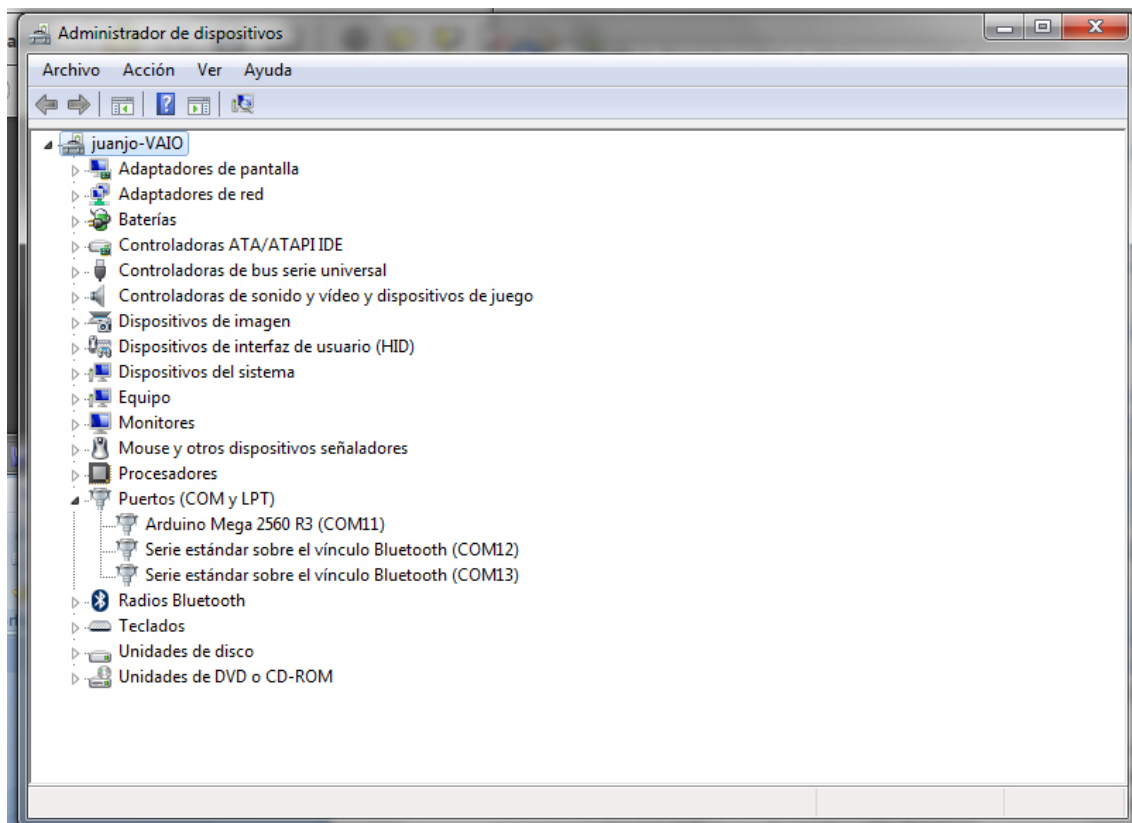


Figura 45: Arduino MEGA detectado como puerto COM

En la imagen puede consultarse además, el puerto COM asignado a Arduino. De este modo, no es necesario probar a conectar en todos los puertos Serie del equipo hasta encontrar a Arduino.

Windows 7 detecta correctamente Arduino. No es necesario descarga de drivers de Arduino. En caso de error en la instalación, es posible descargar la última versión del

driver del fabricante del chip FTDI en su página web oficial <http://www.ftdichip.com/Drivers/VCP.htm>.

A continuación, y una vez instalado Arduino correctamente, es posible abrir la aplicación Arduino Velleman KSR10.exe.

En primer lugar, es preciso seleccionar el puerto de comunicaciones.



Figura 46: Selector puerto COM

Tras seleccionar el puerto de comunicaciones de Arduino (puede comprobarse en el Panel de Control de Windows o probar hasta detectar en cual está instalado Arduino), se habilitará un botón de conexión junto al listview:



Figura 46: Selector puerto COM

Al hacer clic sobre el botón conectar, la aplicación intentará detectar si Arduino se encuentra conectado en dicho puerto. En caso afirmativo se activará el control manual y el estado será “conectado”. En caso de error, es posible que Arduino no esté conectado al puerto seleccionado. En caso de que el puerto sea correcto, desconecte Arduino del puerto USB y vuelva a conectarlo.



Figura 47: estado aplicación comunicada con Arduino

Tras la conexión a la aplicación, el modo de trabajo por defecto será manual. En capítulos posteriores se describe el funcionamiento de los modos manual y automático.

## 8.4 PANEL DE CONTROL

Desde el panel de control es posible realizar tareas de configuración de Arduino, conocer su estado así como la consola de comunicación serie entre Arduino y la aplicación.

The screenshot shows a software interface titled "Control" with several sections:

- Puerto Comunicaciones:** Shows "Estado: CONECTADO".
- Manual / Automático:** A selector with "MODO AUTO" and a robot icon.
- Automatización:** A table with columns "Orden", "Tipo", "Subtipo", and "Parametros".
- Consola:** A terminal window showing communication logs.

Labels on the right side of the image identify these sections:

- Información Comunicaciones
- Selector de Modo de Trabajo
- Configuración Automatización
- Consola

Orden	Tipo	Subtipo	Parametros
1	MANO	ABRE	
2	MOV	IND	657;915;297;...
3	MOV	COG	657;878;250;...
4	MANO	CIERRA	
5	MOV	REG	657;968;106;...

```
ARDUINO> Y-OK
PC> SEND B
ARDUINO> -ACCEPTED
ARDUINO> Y
PC> PROGRAMMING REQUEST COM
ARDUINO>
PCMM1: 657;M2: 968;M3: 106;M4: 966;M5: 588
ARDUINO>
```

Figura 48: Panel de Control

Además, se ha insertado un panel de control de posicionamiento de los sensores. Puede resultar útil su uso en caso de realizar tareas de reconfiguración del algoritmo de Arduino:

The "Posiciones" panel contains input fields for sensor positions:

Mano	657	Hombro	721
Muñeca	915	Base	582
Codo	297		

Figura 49: información de posicionamiento

## PUERTO DE COMUNICACIONES

Muestra la información relativa a la conexión entre Arduino y la aplicación de control. En caso de que el usuario pulse el botón de parada de emergencia, la aplicación pasará a modo manual y se indicará en éste campo su estado de Parada. Sin embargo, es posible accionar los controles manuales partir de este estado.

## CAMBIO DE MODO DE TRABAJO

Si se ha enviado una programación de trayectoria a Arduino, será posible alternar entre el modo manual y el automático haciendo clic sobre el icono correspondiente a dicha tarea.

## AUTOMATIZACIÓN

Control de trayectorias. En el apartado correspondiente se describe su uso y aplicación.

## CONSOLA

En la consola se muestra la información relativa a la comunicación entre Arduino y la aplicación de control. En caso de producirse errores de transmisión en las tramas de localización de posición o configuración de trayectorias, se mostrará la información de tal modo que sea el usuario quien decida el modo de actuar.

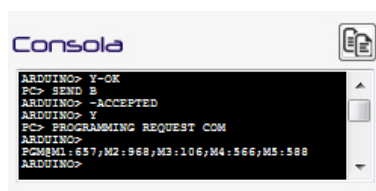


Figura 50: Consola de comunicaciones

Se ha insertado un botón para copiar la información de la consola al portapapeles y poder ser tratada en un archivo de bloc de notas, por ejemplo.

## 8.5 MODO MANUAL

Este es el modo de configuración en que se encuentra la aplicación de control tras conectar con el Brazo Robot Velleman KSR10 y siempre y cuando la comunicación haya sido correcta.





Figura 51: Modo de trabajo Manual

En el modo manual, es posible controlar cualquier movimiento del brazo robot haciendo clic sobre el icono de desplazamiento correspondiente.



Figura 52: Control de movimientos

Mientras mantenga pulsado el botón, el brazo robot se desplazará, parándose al llegar al límite de giro o cuando el usuario deje de pulsar. En caso de pulsar un botón y no observar movimiento, es posible que el eje haya llegado a su límite. Pruebe a pulsar en la dirección contraria.

Para el caso de la mano se ha incluido el botón de control del Led del extremo final del Brazo Robot.

En el modo manual es posible consultar los valores de posicionamiento del brazo robot. En caso de considerar que los límites de movimiento del brazo robot están limitados, es posible modificarlos a en el algoritmo de Arduino y consultarlos a través de dicha funcionalidad.



En el modo manual es posible configurar trayectorias, tal y como se describe en el siguiente apartado. No será posible alternar entre modo manual y automático si no se ha enviado una trayectoria de configuración a Arduino.

## 8.6 PROGRAMACIÓN DE TRAYECTORIAS

Es posible automatizar una serie de movimientos del Brazo Robot Velleman KSR10, de tal modo que sea capaz de desarrollar un trabajo de automatización sin la supervisión del usuario.

Se ha especificado un límite de 25 acciones.

Para crear un listado de acciones, es preciso conocer los diferentes modos de desplazamiento y acciones del mismo:

Botón	Descripción
	Acción Cierra Mano.
	Acción Abre Mano.
<b>M0</b>	Desplazamiento hasta la posición actual sin tener en cuenta la prioridad de ejes, es decir, todos los motores se activarán simultáneamente.
<b>M1</b>	Desplazamiento para recoger un objeto, es decir, en primer lugar se desplazará en el plano XY y posteriormente Z.
<b>M2</b>	Desplazamiento después de dejar un objeto, es decir, en primer lugar se desplazará en el plano Z y posteriormente en el plano XY.
	Trasferencia de la trayectoria a Arduino
	Grabar listado de trayectorias en archivo de texto
	Eliminar la última acción insertada
	Cargar listado de trayectorias desde un archivo de texto
	Eliminar la lista de trayectorias

Para crear una trayectoria, debe desplazarse posición por posición indicando la acción determinada. Por ejemplo, para ir de un punto A a recoger un objeto y desplazarse a un punto B a dejarlo, podría configurar:

- 1 – Acción Abre Mano.
- 2 – Desplace el brazo robot a una posición intermedia de paso. Sería posible desplazarse a ésta de cualquier forma, por tanto “M0”.
- 3 – A continuación, desplace el robot a la posición “A” donde se encuentra el objeto y seleccione “M1”.
- 4 – Acción Cierra Mano.
- 5 – Desplace el brazo robot a otra posición intermedia de paso. El desplazamiento debería ser “M2” o “M0” según las restricciones del espacio. Si el brazo puede interceptar algún elemento en el área de trabajo, seleccione “M2”. En caso contrario puede seleccionar “M0”.



- 6 – Desplace el brazo a la posición “B” destino y seleccione “M1”.
- 7 – Acción Abre Mano.
- 8 – Desplace el brazo robot a la posición de reposo y seleccione “M2”.

Si ha cometido algún error, puede borrar la última acción con el botón habilitado para tal acción; si desea eliminar la configuración completa, haga clic sobre el botón correspondiente.

Una vez creada la trayectoria, envíe la información a Arduino con el botón de transferencia correspondiente. Arduino recibirá la información por partida doble, de tal modo que se eviten errores de transmisión. Si se produce algún error, quedará a la espera de una nueva transmisión; para ello, pulse de nuevo el botón enviar configuración.

Además, es posible guardar trayectorias predefinidas en archivos de configuración que puede almacenar y cargar cuando desee. Para ello, utilice los botones habilitados para tal fin.

## 8.7 MODO MANUAL

Una vez cargada la configuración de trayectorias a Arduino, es posible alternar entre modo manual y automático. Para ello, haga clic sobre el botón automático y comenzará la ejecución de la trayectoria.



Figura 53: Control de modo

En caso de que se produzca algún inconveniente, es posible activar la parada de emergencia. Para ello, haga clic sobre el icono habilitado para tal fin y el brazo robot pasará a modo manual. Si el problema se ha debido a algún fallo o a la desconexión entre el brazo y el ordenador, puede pulsar el botón reset el brazo robot.

Arduino almacena la información de trayectorias hasta que sean sobrescritas o este sea reseteado o apagado. Se recomienda almacenar en archivos de texto las trayectorias repetitivas de modo que no sea necesario describirlas en cada reinicio.

## 8.8 MANTENIMIENTO PREVENTIVO

Para garantizar el buen funcionamiento del sistema, se recomienda que se sigan con rigor las pautas de mantenimiento:



1. El brazo robot debe estar situado en espacio seco y con la menor radiación de sol posible.
2. No deben manipularse los sensores de posicionamiento.
3. Es preciso verificar periódicamente que los sensores de posicionamiento no han sufrido daños. En caso de sufrir algún daño es posible que el brazo robot no detecte su posición y el movimiento del eje exceda su límite.



## 9. CONCLUSIONES

En el Mundo industrial que vivimos actualmente, la automatización es base fundamental en la industria, no sólo para suprimir tareas peligrosas o monótonas, si no por la eficiencia, la supresión de errores y la optimización de recursos.

Antes de conocer el objetivo de éste proyecto y los medios con los que contábamos, Arduino había llamado mi atención si bien es cierto que desconocía sus posibilidades.

El conocer los objetivos del proyecto, mezclando Arduino + robótica + programación, me convencieron para lograr el resultado que ahora podemos contemplar.

Haber tenido que estudiar a fondo el brazo robótico Velleman KSR10 y su automatización me ha introducido en el mundo de la robótica, un campo nunca tocado a lo largo de la carrera.

Por el contrario, otros campos sí tocados en la carrera han sido reforzados, mejorados e incluso me he percatado de la utilidad de metodologías, cálculos, algoritmos, etc. a los que nunca había dado un uso profesional, principalmente relacionados con los sistemas de control, la programación y la transmisión de datos.

Todos los objetivos marcados en un primer momento se han cumplido, si bien es cierto que me habría gustado poder perfeccionar algunos otros, como la aplicación de la cinemática inversa y el Algoritmo de Bresenham pero que, debido a las limitaciones de Brazo Robot Velleman KSR10 no ha sido posible. Sin embargo, el comportamiento del brazo es muy natural y la repetitividad del sistema de sensores instalado fiable.

Arduino supone un mundo aparte en paralelo con el desarrollo tecnológico. Haberlo estudiado con tanto detenimiento me ha llevado a comprender la importancia que tiene en el mundo de la formación y el diseño de prototipos. Creo que el desarrollo de Arduino no ha hecho más que empezar y que la evolución tecnológica lo hará más dependiente en la tecnología del futuro a corto plazo.

En general, este proyecto ha cumplido mis expectativas y mis conocimientos se han visto reforzados en los campos que considero más importantes para mi carrera: control y automatización de sistemas electrónicos, programación y transmisión de datos. Considero que ha sido un complemento importante y espero que éste aporte que hago en forma de documento sirva para generaciones venideras en su estudio de Arduino y la comunicación con sistemas informáticos.

No me gustaría acabar sin proponer una línea de trabajo para posibles mejoras sobre este brazo robot: basándonos en el software desarrollado para Arduino, implementar la comunicación bluetooth con una aplicación de control en S.O. Android para Smartphone. Además, sería interesante cifrar la conexión serie de tal modo que no sea posible interceptar la comunicación.



## 10. ANEXO

En este apartado se detalla todo lo referente a la instalación del software necesario para la programación y configuración de Arduino y de la aplicación de control desarrollada en Visual Studio 2012.

### 10.1 GUÍA INSTALACIÓN IDE ARDUINO

En este apartado se detalla la instalación de Arduino y de su Entorno de Desarrollo.

En primer lugar, acceda a la página oficial de arduino <http://arduino.cc>. En el apartado descarga se puede encontrar la última versión de dicho software para su sistema operativo. Si bien es cierto que se puede programar Arduino en MacOSX, Linux y Windows, se recomienda éste último lenguaje al ser el utilizado para el desarrollo de la aplicación .NET

El entorno de desarrollo de Arduino está escrito en java y basado en Processing, avr-gcc y otros programas de código abierto y se proporciona “como es”, sin ninguna garantía respecto a su funcionalidad, operativa, uso limitación o implicación de garantías de funcionamiento. Por este motivo, si detecta algún problema durante el uso o instalación de la aplicación, puede descargar la versión anterior estable.

Una vez que ha finalizado la descarga del IDE de Arduino, descomprima el fichero. Es preciso mantener la estructura de directorios. Una vez descomprimido el documento, puede ejecutar el IDE de Arduino haciendo clic sobre el archivo Arduino.exe

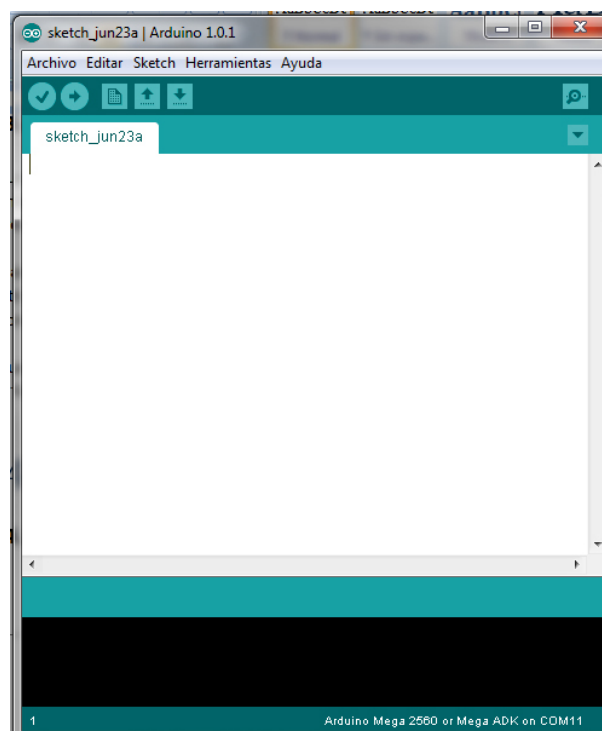


Figura 54: IDE de Arduino

## INSTALACIÓN DE LA PLACA ARDUINO

Conecte la placa Arduino al ordenador a través del puerto USB.

La primera vez que se conecte el brazo robot al ordenador, el Sistema Operativo pedirá autorización para instalar un nuevo hardware. Es posible seleccionar la instalación automática del dispositivo tras lo cual, debería ser instalado y reconocido correctamente por el Sistema Operativo Windows.

En la ruta Panel de Control >> Hardware y Sonido >> Administrador de dispositivos podemos verificar la información de Arduino:

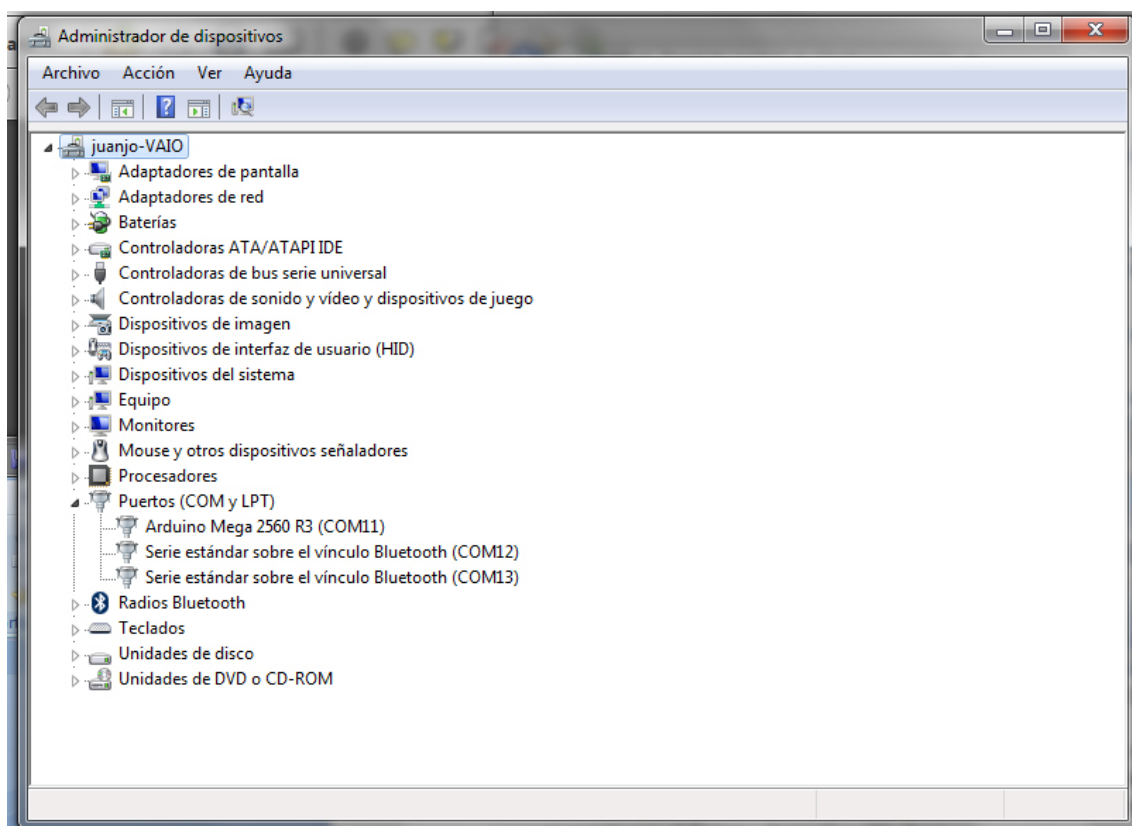


Figura 55: Arduino MEGA detectado como puerto COM

En la imagen puede consultarse además, el puerto COM asignado a Arduino.

Windows 7 detecta correctamente Arduino. No es necesario descarga de drivers de Arduino. En caso de error en la instalación o de utilizar otra versión de Sistema Operativo Windows, es posible descargar la última versión del driver del fabricante del chip FTDI en su página web oficial <http://www.ftdichip.com/Drivers/VCP.htm>.



## SELECCIÓN DE LA PLACA

Antes de cargar un Sketch en la placa Arduino, es preciso seleccionar el modelo a programar. En el IDE Arduino seleccione Tools>>Board.

En Herramientas>>Puertos Serie se selecciona el puerto serie asignado, el cual es visible en el panel de control, tal y como se muestra en la figura 55.

## INSTALACIÓN LIBRERÍAS

Las librerías proveen a Arduino de funcionalidad extra, por ejemplo: trabajar con otro hardware o manipular datos.

Habrán casos en que es preciso descargar librerías de Arduino no contenidas en el IDE por defecto. Este es el caso, por ejemplo, de la librería PID utilizada para la programación de los PID de control de los motores.

Esta librería está disponible en la página  
<http://playground.arduino.cc/Code/PIDLibrary>

Para instalar una librería, descargue la librería y descomprímala. Debería localizarse en una carpeta propia, y normalmente, contener dos archivos, uno con sufijo “.h” y otro con sufijo “.cpp”. Abra la carpeta sketchbook de Arduino y coloque la carpeta de la nueva librería dentro de la carpeta “libraries”. Al reiniciar IDE de Arduino se localizará la nueva librería en el menú Sketch>>Import Library.

## 10.2 GUÍA INSTALACIÓN VISUAL STUDIO 2012

Es posible editar el código de programación del archivo de control, desarrollado en Visual Basic .NET con cualquiera de las versiones de Visual Studio 2012.

Para instalar el mismo, puede acceder a la página de Microsoft <http://www.microsoft.com/visualstudio/esn>. Existe una versión gratuita (Visual Studio 2012 Express) para estudiantes.

Para instalar Visual Studio 2012 es necesario tener credenciales de administrador. Sin embargo, no es necesario tener estas credenciales para poder utilizar Visual Studio después de su instalación.

Si hay una versión anterior de Visual Studio instalada en el equipo, obtenga información sobre como ejecutar varias versiones en paralelo. Dispone de la información adecuada en [msdn.microsoft.com](http://msdn.microsoft.com)

Al comenzar a instalar la aplicación puede escoger la ruta de instalación:



Figura 56: Selección de ruta de instalación.

A continuación, seleccione las aplicaciones a instalar:

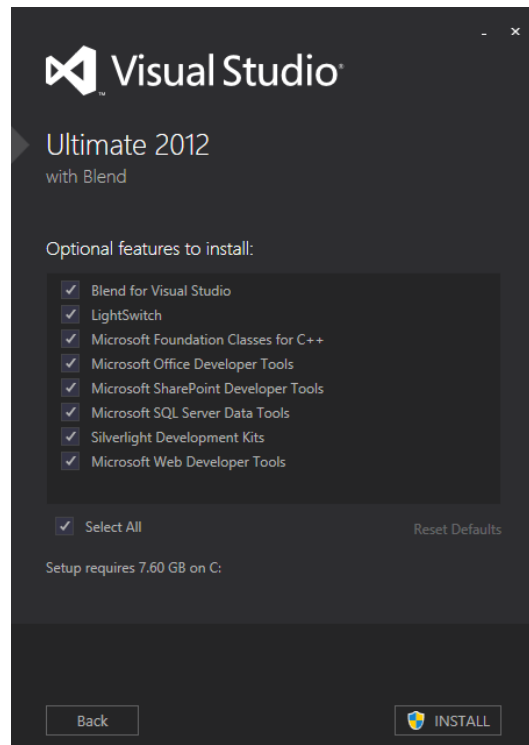


Figura 57: Selección de software a instalar.

Comenzará la instalación y descarga de complementos:





Figura 57: instalación.

Una vez finalizada la instalación, puede ejecutar Visual Studio:

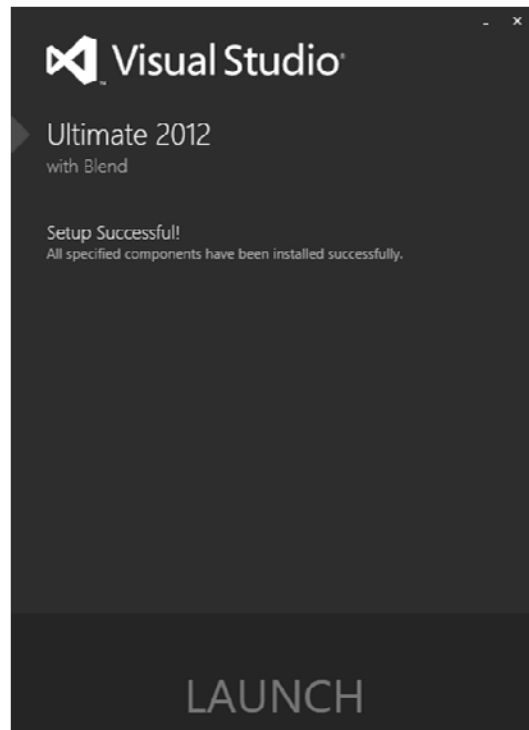


Figura 58: instalación completada

El entorno de desarrollo de .NET es el mostrado a continuación. Puede abrir el proyecto creado para el desarrollo de la aplicación de control. Una vez modificada, puede depurar y compilar la misma para generar el archivo .exe ejecutable:

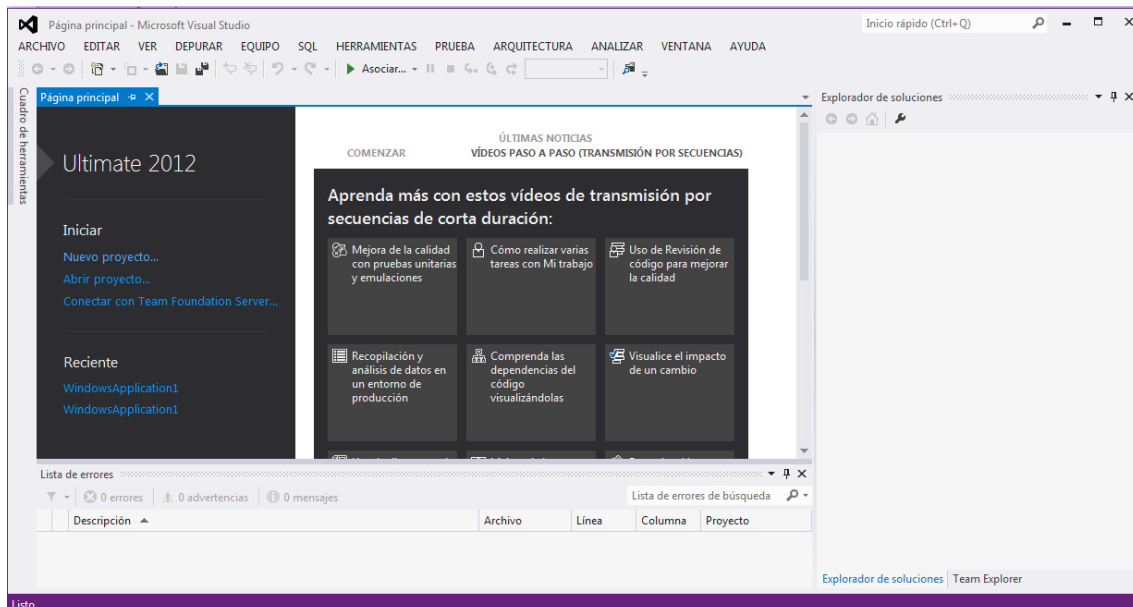


Figura 59: entorno de programación de Visual Studio 2012

### 10.3 CÓDIGO FUENTE ARDUINO

A continuación se adjunta el código fuente de la aplicación Arduino:

```
//Importación de librerías
#include <PID_v1.h>
#include <string.h>

//Declaración de constantes
#define BUFFSIZ 450
#define ORDENSIZ 7
#define INSTRUCCIONES 25

//Declaración de límites de funcionamiento de los distintos ejes
//Es posible modificar los parámetros siempre que no superen los
límites
//de acción de los ejes
#define M1MAX 900
#define M1MIN 700
#define M2MAX 950
#define M2MIN 780
#define M3MAX 1000
#define M3MIN 100
#define M4MAX 990
#define M4MIN 575
#define M5MAX 820
#define M5MIN 190

// Declaración de Pines de control de L293D
```



```
//Motor 1 --> Mano
int motor1EnablePin = 2;
int motor1Pin1 = 22;
int motor1Pin2 = 23;
//Motor 2 --> Muñeca
int motor2EnablePin = 7;
int motor2Pin1 = 25;
int motor2Pin2 = 24;
//Motor 3 --> Codo
int motor3EnablePin = 4;
int motor3Pin1 = 26;
int motor3Pin2 = 27;
//Motor 4 --> Hombro
int motor4EnablePin = 5;
int motor4Pin1 = 28;
int motor4Pin2 = 29;
//Motor 5 --> Base
int motor5EnablePin = 6;
int motor5Pin1 = 30;
int motor5Pin2 = 31;
//Led
int lightPin=32;

//Declaracion de Pines para los los sensores de posición
int sensorPin1 = A1;
int sensorValue1 = 0;
int mem_sensor1=0;
int sensorPin2 = A2;
int sensorValue2 = 0;
int mem_sensor2=0;
int sensorPin3 = A3;
int sensorValue3 = 0;
int mem_sensor3=0;
int sensorPin4 = A4;
int sensorValue4 = 0;
int mem_sensor4=0;
int sensorPin5 = A5;
int sensorValue5 = 0;
int mem_sensor5=0;

//Declaración de arrays automatización
int pos[INSTRUCCIONES][ORDENSIZ];
char buffer[BUFFSIZ];

//Declaración de variables genericas y de control de procesos
int var,var_old,var2;
int control=1;
int total_progs=0;
String cadena;
int dato_puerto_serial;
int intervalo=0;
int control_modo=0; // 0 manual; 1 auto
int control_prog=0;

//Declaración de otras variables
double Setpoint1, Input1, Output1,Setpoint_old1;
double Setpoint2, Input2, Output2,Setpoint_old2;
double Setpoint3, Input3, Output3,Setpoint_old3;
```



```
double Setpoint4, Input4, Output4, Setpoint_old4;
double Setpoint5, Input5, Output5, Setpoint_old5;

//Declaración de parametros de PID. Es posible modificarles de acuerdo
//a las necesidades de velocidad o precisión del brazo robot
double aggKp1=6, aggKi1=0.1, aggKd1=0;
double consKp1=0.2, consKi1=0.05, consKd1=0;
double aggKp2=6, aggKi2=0.1, aggKd2=0;
double consKp2=0.6, consKi2=0.05, consKd2=0;
double aggKp3=8, aggKi3=0.1, aggKd3=0;
double consKp3=2.2, consKi3=0.05, consKd3=0;
double aggKp4=8, aggKi4=0.1, aggKd4=0;
double consKp4=0.2, consKi4=0.05, consKd4=0;
double aggKp5=8, aggKi5=0.1, aggKd5=0;
double consKp5=0.2, consKi5=0.05, consKd5=0;

PID myPID1(&Input1, &Output1, &Setpoint1, consKp1, consKi1, consKd1,
DIRECT);
PID myPID2(&Input2, &Output2, &Setpoint2, consKp2, consKi2, consKd2,
DIRECT);
PID myPID3(&Input3, &Output3, &Setpoint3, consKp3, consKi3, consKd3,
DIRECT);
PID myPID4(&Input4, &Output4, &Setpoint4, consKp4, consKi4, consKd4,
DIRECT);
PID myPID5(&Input5, &Output5, &Setpoint5, consKp5, consKi5, consKd5,
DIRECT);

void setup() {
//Declaración de variables control de los motores
pinMode(motor1Pin1, OUTPUT);
pinMode(motor1Pin2, OUTPUT);
pinMode(motor1EnablePin, OUTPUT);
digitalWrite(motor1EnablePin, HIGH);
pinMode(motor2Pin1, OUTPUT);
pinMode(motor2Pin2, OUTPUT);
pinMode(motor2EnablePin, OUTPUT);
digitalWrite(motor2EnablePin, HIGH);
pinMode(motor3Pin1, OUTPUT);
pinMode(motor3Pin2, OUTPUT);
pinMode(motor3EnablePin, OUTPUT);
digitalWrite(motor3EnablePin, HIGH);
pinMode(motor4Pin1, OUTPUT);
pinMode(motor4Pin2, OUTPUT);
pinMode(motor4EnablePin, OUTPUT);
digitalWrite(motor4EnablePin, HIGH);
pinMode(motor5Pin1, OUTPUT);
pinMode(motor5Pin2, OUTPUT);
pinMode(motor5EnablePin, OUTPUT);
digitalWrite(motor5EnablePin, HIGH);

pinMode(lightPin, OUTPUT);
digitalWrite(lightPin, LOW);
//Inicialización de puerto serie
Serial.begin(9600);
Setpoint_old1=analogRead(sensorPin1);
Setpoint_old2=analogRead(sensorPin2);
Setpoint_old3=analogRead(sensorPin3);
Setpoint_old4=analogRead(sensorPin4);
Setpoint_old5=analogRead(sensorPin5);
myPID1.SetMode(AUTOMATIC);
```



```
myPID2.SetMode(AUTOMATIC);
myPID3.SetMode(AUTOMATIC);
myPID4.SetMode(AUTOMATIC);
myPID5.SetMode(AUTOMATIC);
//Especificación de las limitaciones del brazo robot
//para su control de velocidad mínima y máxima
myPID1.SetOutputLimits(120, 205);
myPID2.SetOutputLimits(120, 205);
myPID3.SetOutputLimits(120, 205);
myPID4.SetOutputLimits(120, 205);
myPID5.SetOutputLimits(120, 205);

}

void loop() {
  sensorValue1 = analogRead(sensorPin1);
  sensorValue2 = analogRead(sensorPin2);
  sensorValue3 = analogRead(sensorPin3);
  sensorValue4 = analogRead(sensorPin4);
  sensorValue5 = analogRead(sensorPin5);

  // Si el puerto serie está disponible
  if (Serial.available() > 0) {
    dato_puerto_serial = Serial.read();
    if(dato_puerto_serial == 'P'){
      parada_emergencia();
    }

    if(dato_puerto_serial == 'M'){
      //modo manual
      control_modos=0;
      digitalWrite(motor1EnablePin, HIGH);
      digitalWrite(motor2EnablePin, HIGH);
      digitalWrite(motor3EnablePin, HIGH);
      digitalWrite(motor4EnablePin, HIGH);
      digitalWrite(motor5EnablePin, HIGH);
      Serial.print("CONNECTED");
    }else if(dato_puerto_serial == 'N'){
      //modo auto
      control_modos=1;
      control=1;
      Serial.print("CONNECTED");
    }

    //Si el modo es automatico
    if(control_modos==1){
      //ejecución del bucle de automatización hasta llegar a la última
      acción almacenada
      do{
        //según el tipo de acción, se ejecuta una u otra función
        if(pos[control-1][0]==1){
          //movimiento
          Setpoint2=pos[control-1][3];
          Setpoint3=pos[control-1][4];
          Setpoint4=pos[control-1][5];
          Setpoint5=pos[control-1][6];
          if(pos[control-1][1]==3){
            //movimiento indiferente
            posicion(0);
          }else if(pos[control-1][1]==4){
```



```
        //movimiento coger
        posicion(1);
    }else if(pos[control-1][1]==5){
        //movimiento vuelve de coger
        posicion(2);
    }
}
}else if(pos[control-1][0]==2){
    //mano
    if(pos[control-1][1]==1){
        //cierra
        pinza(1);
    }else if(pos[control-1][1]==2){
        //abre
        pinza(0);
    }
}
}
}while(control<=total_progs && control_modo==1);

}

//fin control_modo=1 == AUTOMATICO

//si la aplicación de control solicita programación, arduino
responde
//informando que está listo para leer la configuración
if(control_modo==0){
    if(dato_puerto_serial=='Y'){
        Serial.print("Y-ACCEPTED");
        control_prog=1;
        //llama a función de configuración
        readString();
    }
}

//Si parametro=I (Información) se muestra el valor actual de
los sensores
if(dato_puerto_serial == 'I'){
    cadena="M1:"+String(sensorValue1);
    cadena=cadena+";M2:"+String(sensorValue2);
    cadena=cadena+";M3:"+String(sensorValue3);
    cadena=cadena+";M4:"+String(sensorValue4);
    cadena=cadena+";M5:"+String(sensorValue5);
    Serial.print(cadena);
}
//Si parametro=U (información) se envia la información de los
sensores
//para crear un punto de trayectoria
if(dato_puerto_serial == 'U'){
    cadena="PGM@M1:"+String(sensorValue1);
    cadena=cadena+";M2:"+String(sensorValue2);
    cadena=cadena+";M3:"+String(sensorValue3);
    cadena=cadena+";M4:"+String(sensorValue4);
    cadena=cadena+";M5:"+String(sensorValue5);
    Serial.print(cadena);
}

//Control MOTOR 1 --> MANO
if (dato_puerto_serial == 'Q') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='O')break;
        sensorValue1 = analogRead(sensorPin1);
    }
}
```



```
        if(sensorValue1<M1MAX && dato_puerto_serial!='0'){
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, HIGH);
        }else{
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}
if (dato_puerto_serial == 'W') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue1 = analogRead(sensorPin1);
        if(sensorValue1>M1MIN && dato_puerto_serial!='0'){
            digitalWrite(motor1Pin1, HIGH);
            digitalWrite(motor1Pin2, LOW);
        }else{
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}

//Control MOTOR 2 --> MUÑECA
if (dato_puerto_serial == 'R') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue2 = analogRead(sensorPin2);
        if(sensorValue2<M2MAX && dato_puerto_serial!='0'){
            digitalWrite(motor2Pin1, LOW);
            digitalWrite(motor2Pin2, HIGH);
        }else{
            digitalWrite(motor2Pin1, LOW);
            digitalWrite(motor2Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}
if (dato_puerto_serial == 'E') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue2 = analogRead(sensorPin2);
        if(sensorValue2>M2MIN && dato_puerto_serial!='0'){
            digitalWrite(motor2Pin1, HIGH);
            digitalWrite(motor2Pin2, LOW);
        }else{
            digitalWrite(motor2Pin1, LOW);
            digitalWrite(motor2Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}

//Controm MOTOR 3 --> CODO
if (dato_puerto_serial == 'A') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue3 = analogRead(sensorPin3);
```



```
        if(sensorValue3>M3MIN && dato_puerto_serial!='0'){
            digitalWrite(motor3Pin1, LOW);
            digitalWrite(motor3Pin2, HIGH);
        }else{
            digitalWrite(motor3Pin1, LOW);
            digitalWrite(motor3Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}
if (dato_puerto_serial == 'S') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue3 = analogRead(sensorPin3);
        if(sensorValue3<M3MAX && dato_puerto_serial!='0'){
            digitalWrite(motor3Pin1, HIGH);
            digitalWrite(motor3Pin2, LOW);
        }else{
            digitalWrite(motor3Pin1, LOW);
            digitalWrite(motor3Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}

//Control MOTOR 4 --> HOMBRO
if (dato_puerto_serial == 'D') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue4 = analogRead(sensorPin4);
        if(sensorValue4>M4MIN && dato_puerto_serial!='0'){
            digitalWrite(motor4Pin1, LOW);
            digitalWrite(motor4Pin2, HIGH);
        }else{
            digitalWrite(motor4Pin1, LOW);
            digitalWrite(motor4Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}
if (dato_puerto_serial == 'F') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue4 = analogRead(sensorPin4);
        if(sensorValue4<M4MAX && dato_puerto_serial!='0'){
            digitalWrite(motor4Pin1, HIGH);
            digitalWrite(motor4Pin2, LOW);
        }else{
            digitalWrite(motor4Pin1, LOW);
            digitalWrite(motor4Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}

//Control MOTOR 5 --> BASE
if (dato_puerto_serial == 'Z') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue5 = analogRead(sensorPin5);
```





```
        if(sensorValue5<M5MAX && dato_puerto_serial!='0'){
            digitalWrite(motor5Pin1, LOW);
            digitalWrite(motor5Pin2, HIGH);
        }else{
            digitalWrite(motor5Pin1, LOW);
            digitalWrite(motor5Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}
if (dato_puerto_serial == 'X') {
    do{
        dato_puerto_serial = Serial.read();
        if(dato_puerto_serial=='0')break;
        sensorValue5 = analogRead(sensorPin5);
        if(sensorValue5>M5MIN && dato_puerto_serial!='0'){
            digitalWrite(motor5Pin1, HIGH);
            digitalWrite(motor5Pin2, LOW);
        }else{
            digitalWrite(motor5Pin1, LOW);
            digitalWrite(motor5Pin2, LOW);
        }
    }while(dato_puerto_serial!='0');
}
if (dato_puerto_serial == 'L') {
    digitalWrite(lightPin, HIGH);
}
else if (dato_puerto_serial == 'K') {
    digitalWrite(lightPin, LOW);
}

//Cuando no se recibe información útil, se recibirán '0'
//En caso de recibir '0' --> se paran todos los motores
if (dato_puerto_serial == '0') {
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);
    digitalWrite(motor3Pin1, LOW);
    digitalWrite(motor3Pin2, LOW);
    digitalWrite(motor4Pin1, LOW);
    digitalWrite(motor4Pin2, LOW);
    digitalWrite(motor5Pin1, LOW);
    digitalWrite(motor5Pin2, LOW);
}

} //fin control_modo=0

} //fin if puerto serial disponible

} //fin loop

//FUNCION READSTRING()
//Lee trama de configuración de puerto serie y la segmenta en
instrucciones
//que se almacenarán en el vector de trayectorias
//Utiliza punteros para simular una función split que separe las
tramas
//por el caracter de control.
```



```
void readString() {
  char c;
  int buffSize = 0;
  int buc=0;
  char *p;
  char *str;
  char *str2;
  char *p2;
  char *str3;
  char *p3;
  int postmp=0;
  int subpostmp=0;
  int posmotor=2;
  int countbucle;
  int total_chars=0;
  int verifica_chars=0;
  int filas=0;
  int columnas=0;
  //vaciamos los arrays
  memset(buffer, 0, sizeof(buffer));
  for(filas=0;filas<INSTRUCCIONES;filas++){
    for(columnas=0;columnas<7;columnas++){
      pos[filas][columnas]=0;
    }
  }
  Serial.flush();
  do{

    while (Serial.available(>0){
      buffer[buffSize]=Serial.read();
      if(buffer[buffSize]=='#'){control_prog=0;break;}
      buffSize++;
    }

  }while(control_prog==1);
  p=buffer;
  while ((str = strtok_r(p, "$", &p)) != NULL){
    //cadena en str
    if(postmp==0){total_progs=atoi(str);}
    else if(postmp>0 && postmp<=total_progs){
      p2=str;
      subpostmp=0;
      countbucle=0;
      while ((str2 = strtok_r(p2, "!", &p2)) != NULL){
        if(subpostmp==0){subpostmp=atoi(str2);}
        else{
          if(countbucle==1){
            if(str2[0]=='M'){
              pos[subpostmp-1][0]=1;
            }else if(str2[0]=='H'){
              pos[subpostmp-1][0]=2;
            }
          }
          }else if(countbucle==2){
            if(str2[0]=='C'){
              pos[subpostmp-1][1]=1;
            }else if(str2[0]=='A'){
              pos[subpostmp-1][1]=2;
            }else if(str2[0]=='I'){
              pos[subpostmp-1][1]=3;
            }else if(str2[0]=='L'){

```



```
        pos[subpostmp-1][1]=4;
    }else if(str2[0]=='T'){
        pos[subpostmp-1][1]=5;
    }
}else if(countbucle==3){
    if(str2!=""){
        p3=str2;
        posmotor=2;
        while ((str3 = strtok_r(p3, ";", &p3)) !=
NULL){
            pos[subpostmp-1][posmotor]=atoi(str3);
            posmotor++;
        }//fin while str3
    }//fin if str2=""
}//fin countbucle==3
}//fin subpostmp==1
countbucle++;
}//fin while str2
}//fin else postmp>0 y <total_progs
else{
    total_chars=atoi(str);
}
postmp++;
}//fin while str

//para comprobar si la transmisión fue correcta tenemos que quitar
los caracteres del contador total y el de fin de trama
if(total_chars<10){
    verifica_chars=buffSize-2;
}else if (total_chars>=10 && total_chars<100){
    verifica_chars=buffSize-3;
}else if (total_chars>=100){verifica_chars=buffSize-4;}

if(verifica_chars==total_chars){
    //transmision-OK
    Serial.print("Y-OK");
}else{
    //transmision-NOOK
    Serial.print("Y-NOOK");
}
control_prog=0;
}

//FUNCION PARADA_EMERGENCIA()
//Si se acciona el botón parade de emergencia en la aplicación de
control,
//se pararán todos los motores y el controlador pasa a modo manual
void parada_emergencia(){
    Serial.print("PARADA EMERGENCIA");
    control_mod0=0;
    digitalWrite(motor1EnablePin, HIGH);
    digitalWrite(motor2EnablePin, HIGH);
    digitalWrite(motor3EnablePin, HIGH);
    digitalWrite(motor4EnablePin, HIGH);
    digitalWrite(motor5EnablePin, HIGH);
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);
}
```



```
        digitalWrite(motor3Pin1, LOW);
        digitalWrite(motor3Pin2, LOW);
        digitalWrite(motor4Pin1, LOW);
        digitalWrite(motor4Pin2, LOW);
        digitalWrite(motor5Pin1, LOW);
        digitalWrite(motor5Pin2, LOW);
        control=1;
    }

//FUNCION PINZA()
//abre o cierra la mano según el parametro de control controlpinza
//Esta función llama a la función girar, encargada del movimiento de
la misma
void pinza(int controlpinza){
    int c1=0;

    if (controlpinza==1){
        Setpoint1=700;
    }else{
        Setpoint1=810;
    }
    if(Setpoint1>Setpoint_old1){
        myPID1.SetControllerDirection(DIRECT);
    }else{
        myPID1.SetControllerDirection(REVERSE);
    }
}

if((analogRead(sensorPin1)>(Setpoint1+15))|| (analogRead(sensorPin1)<(S
etpoint1-15))){
    girar(sensorPin1, motor1Pin1, motor1Pin2, 30,
motor1EnablePin, Setpoint1,myPID1);
    c1=0;
}else{
    Setpoint_old1=analogRead(sensorPin1);
    digitalWrite(motor1Pin1,LOW);
    digitalWrite(motor1Pin2,LOW);
    analogWrite(motor1EnablePin,0);
    c1=1;
}
if(c1==1){
    control=control+1;
}
}

//FUNCION POSICION
//Esta función posiciona el robot según los parametros escogidos y
almacenados en el vector de trayectorias.
//Según el parámetro control_trayectoria determina la prioridad de
movimiento sobre el plano XY, Z o indiferente

void posicion (int control_trayectoria){
    int c2=0;
    int c3=0;
    int c4=0;
    int c5=0;
    int control_prioridad;
    if(control_trayectoria==1){
        control_prioridad=1;
    }else if(control_trayectoria==2){
```



```
    control_prioridad=0;
  }else{
    control_prioridad=2;
  }
  do{
    dato_puerto_serial = Serial.read();
    if(dato_puerto_serial == 'P'){
      parada_emergencia();
      break;
      break;
    }
    if(control_prioridad==0 || control_prioridad==2){

      if(Setpoint2>Setpoint_old2){
        myPID2.SetControllerDirection(DIRECT);
      }else{
        myPID2.SetControllerDirection(REVERSE);
      }
    }

    if((analogRead(sensorPin2)>(Setpoint2+5)) || (analogRead(sensorPin2)<(Setpoint2-5))){
      gira2(sensorPin2, motor2Pin1, motor2Pin2, 30,
      motor2EnablePin, Setpoint2,myPID2);
      c2=0;
    }else{
      Setpoint_old2=analogRead(sensorPin2);
      digitalWrite(motor2Pin1,LOW);
      digitalWrite(motor2Pin2,LOW);
      analogWrite(motor2EnablePin,0);
      c2=1;
    }

    if(Setpoint3>Setpoint_old3){
      myPID3.SetControllerDirection(DIRECT);
    }else{
      myPID3.SetControllerDirection(REVERSE);
    }
  }

  if((analogRead(sensorPin3)>(Setpoint3+5)) || (analogRead(sensorPin3)<(Setpoint3-5))){
      gira3(sensorPin3, motor3Pin1, motor3Pin2, 100,
      motor3EnablePin, Setpoint3,myPID3);
      c3=0;
    }else{
      Setpoint_old3=analogRead(sensorPin3);
      digitalWrite(motor3Pin1,LOW);
      digitalWrite(motor3Pin2,LOW);
      analogWrite(motor3EnablePin,0);
      c3=1;
    }

    if(Setpoint4>Setpoint_old4){
      myPID4.SetControllerDirection(DIRECT);
    }else{
      myPID4.SetControllerDirection(REVERSE);
    }
  }

  if((analogRead(sensorPin4)>(Setpoint4+5)) || (analogRead(sensorPin4)<(Setpoint4-5))){
```



```
        gira4(sensorPin4, motor4Pin1, motor4Pin2, 100,
motor4EnablePin, Setpoint4,myPID4);
        c4=0;
    }else{
        Setpoint_old4=analogRead(sensorPin4);
        digitalWrite(motor4Pin1,LOW);
        digitalWrite(motor4Pin2,LOW);
        analogWrite(motor4EnablePin,0);
        c4=1;
    }

    if(c2==1 && c3==1 && c4==1){
        control_prioridad=2;
    }
} //fin control prioridad==0

if(control_prioridad==1 || control_prioridad==2){

    if(Setpoint5>Setpoint_old5){
        myPID5.SetControllerDirection(DIRECT);
    }else{
        myPID5.SetControllerDirection(REVERSE);
    }

if((analogRead(sensorPin5)>(Setpoint5+5)) || (analogRead(sensorPin5)<(Setpoint5-5))){
        gira5(sensorPin5, motor5Pin1, motor5Pin2, 100,
motor5EnablePin, Setpoint5,myPID5);
        c5=0;
    } else{
        Setpoint_old5=analogRead(sensorPin5);
        digitalWrite(motor5Pin1,LOW);
        digitalWrite(motor5Pin2,LOW);
        analogWrite(motor5EnablePin,0);
        c5=1;
    }

    if(c5==1){
        control_prioridad=2;
    }

} //fin control prioridad==0

if(c2==1 && c3==1 && c4==1 && c5==1){
    control=control+1;
}

}while(c2!=1 || c3!=1 || c4!=1 || c5!=1);
}

//FUNCION GIRAL
//Esta es la función encargada de abrir o cerrar la mano según el
posicionamiento deseado
void giral(int id_sensor, int motor_dcha, int motor_izq, int
reduccion, int id_enable, double Setpoint, PID myPID){
    Input1 = analogRead(id_sensor);
    double gap = abs(Setpoint-Input1); //distancia entre el punto
actual y el definitivo
```



```
    if(Setpoint>Input1){
        digitalWrite(motor_dcha,LOW);
        digitalWrite(motor_izq,HIGH);
    }else{
        digitalWrite(motor_izq,LOW);
        digitalWrite(motor_dcha,HIGH);
    }

    if(gap<reduccion){ //si estamos cerca del punto, reducimos
parametro PID
        myPID.SetTunings(consKp1, consKi1, consKd1);
    }else{
        //si estamos alejados, utilizamos parametros mas agresivos para
PID
        myPID.SetTunings(aggKp1, aggKi1, aggKd1);
    }

    myPID.Compute();

    var=analogRead(id_sensor);

    if(var!=var_old){
        var2=Output1;
        var_old=var;
    }
    analogWrite(id_enable,Output1);
}

//FUNCION GIRA2
//Esta es la función encargada de activar el motor 2 según la posición
deseada

void gira2(int id_sensor, int motor_dcha, int motor_izq, int
reduccion, int id_enable, double Setpoint, PID myPID){
    Input2 = analogRead(id_sensor);
    double gap = abs(Setpoint-Input2); //distancia entre el punto actual
y el definitivo

    if(Setpoint>Input2){
        digitalWrite(motor_dcha,LOW);
        digitalWrite(motor_izq,HIGH);
    }else{
        digitalWrite(motor_izq,LOW);
        digitalWrite(motor_dcha,HIGH);
    }

    if(gap<reduccion){ //si estamos cerca del punto, reducimos
parametro PID
        myPID.SetTunings(consKp2, consKi2, consKd2);
    }else{
        //si estamos alejados, utilizamos parametros mas agresivos para
PID
        myPID.SetTunings(aggKp2, aggKi2, aggKd2);
    }

    myPID.Compute();

    var=analogRead(id_sensor);
```



```
    if(var!=var_old){
        var2=Output2;

        var_old=var;
    }
    analogWrite(id_enable,Output2);
}

//FUNCION GIRA3
//Esta es la función encargada de activar el motor 3 según la posición deseada

void gira3(int id_sensor, int motor_dcha, int motor_izq, int
reduccion, int id_enable, double Setpoint, PID myPID){
    Input3 = analogRead(id_sensor);
    double gap = abs(Setpoint-Input3); //distancia entre el punto actual
y el definitivo

    if(Setpoint>Input3){
        digitalWrite(motor_izq,LOW);
        digitalWrite(motor_dcha,HIGH);
    }else{
        digitalWrite(motor_dcha,LOW);
        digitalWrite(motor_izq,HIGH);
    }

    if(gap<reduccion){ //si estamos cerca del punto, reducimos
parametro PID
        myPID.SetTunings(consKp3, consKi3, consKd3);
    }else{
        //si estamos alejados, utilizamos parametros mas agresivos para
PID
        myPID.SetTunings(aggKp3, aggKi3, aggKd3);
    }

    myPID.Compute();

    var=analogRead(id_sensor);

    if(var!=var_old){
        var2=Output3;
        var_old=var;
    }
    analogWrite(id_enable,Output3);
}

//FUNCION GIRA4
//Esta es la función encargada de activar el motor 4 según la posición deseada
void gira4(int id_sensor, int motor_dcha, int motor_izq, int
reduccion, int id_enable, double Setpoint, PID myPID){
    Input4 = analogRead(id_sensor);
    double gap = abs(Setpoint-Input4); //distancia entre el punto actual
y el definitivo
```





```
if(Setpoint>Input4){
    digitalWrite(motor_izq,LOW);
    digitalWrite(motor_dcha,HIGH);
}else{
    digitalWrite(motor_dcha,LOW);
    digitalWrite(motor_izq,HIGH);
}

if(gap<reduccion){ //si estamos cerca del punto, reducimos
parametro PID
    myPID.SetTunings(consKp4, consKi4, consKd4);
}else{
    //si estamos alejados, utilizamos parametros mas agresivos para
PID
    myPID.SetTunings(aggKp4, aggKi4, aggKd4);
}

myPID.Compute();

var=analogRead(id_sensor);

if(var!=var_old){
    var2=Output4;
    var_old=var;
}
analogWrite(id_enable,Output4);
}

//FUNCION GIRAS5
//Esta es la función encargada de activar el motor 5 según la posición
deseada
void gira5(int id_sensor, int motor_dcha, int motor_izq, int
reduccion, int id_enable, double Setpoint, PID myPID){
    Input5 = analogRead(id_sensor);
    double gap = abs(Setpoint-Input5); //distancia entre el punto actual
y el definitivo

if(Setpoint>Input5){
    digitalWrite(motor_dcha,LOW);
    digitalWrite(motor_izq,HIGH);
}else{
    digitalWrite(motor_izq,LOW);
    digitalWrite(motor_dcha,HIGH);
}

if(gap<reduccion){ //si estamos cerca del punto, reducimos
parametro PID
    myPID.SetTunings(consKp5, consKi5, consKd5);
}else{
    //si estamos alejados, utilizamos parametros mas agresivos para
PID
    myPID.SetTunings(aggKp5, aggKi5, aggKd5);
}

myPID.Compute();

var=analogRead(id_sensor);
```



```
    if(var!=var_old){  
        var2=Output5;  
        var_old=var;  
    }  
    analogWrite(id_enable,Output5);  
}
```

## 10.4 CÓDIGO FUENTE VISUAL BASIC .NET

A continuación se adjunta el código fuente de la aplicación .NET:

```
Imports System.IO  
Imports System.IO.Ports  
  
'Clase Form1  
  
Public Class Form1  
    'En el evento de carga del formulario 1 se cierra el puerto 1 si  
    es que está abierto  
  
    Private Sub Form1_Disposed(ByVal sender As Object, ByVal e As  
    System.EventArgs)  
        SerialPort1.Close()  
    End Sub  
  
    'Declaración de variables utilizadas para el control de las  
    comunicaciones y los modos de trabajo  
    Shared _serialPort As SerialPort  
    Dim entradadedatos As String  
    Dim count As Integer  
    Dim valor_enviado As String  
    Dim estado_led As Integer  
    Dim estado_modos As Integer  
    Dim estado_arduino As Integer  
    Dim arduino_programado As Integer  
    Dim estado_programando As Integer  
    Dim contador_movimientos As Integer  
    Dim control_prog_accion As Integer  
    Dim cadena_control_errores_recepcion As String  
    Dim intervalo_lectura_prog As Integer  
    Const total_movimientos_permitidos As Integer = 25  
    Const DELIMITADOR As String = "#"  
  
    'Evento carga de la ventana  
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles MyBase.Load  
        'Se bloquean todos los botones y se espera a que el usuario seleccione  
        un puerto serie al que conectar  
        modo_trabajo_label.Text = "Esperando Conexión"  
        control_prog_accion = 0  
        contador_movimientos = 0  
        arduino_programado = 0  
        estado_programando = 0  
        Timer1.Enabled = False
```



```
    boton_led.Visible = True
    boton_led2.Visible = False
    estado_led = 0
    'boton_mod0.Text = "MODO AUTOMATICO"
    boton_mod0.Visible = True
    boton_mod02.Visible = False
    estado_mod0 = 0
    GetSerialPortNames()
    MessageBox.Show("Por favor, selecciona el puerto de conexión y
pulsa conectar en el menú de control")
    deshabilita_funciones()
End Sub

'Funcion que muestra un listado de puertos serie disponible para
que el usuario escoja a cual conectar
Sub GetSerialPortNames()
    ' Show all available COM ports.
    For Each sp As String In My.Computer.Ports.SerialPortNames
        ListBox1.Items.Add(sp)
    Next
End Sub

'Funcion que genera la trama de transmisión de la información de
configuración
Private Function obtiene_info_prog() As String
    Dim cadena As String
    Dim contador_caracteres As Integer
    Dim tipo As String
    Dim subtipo As String
    'Formato Trama
    'Contador movimientos&orden1&orden2&total_caracteres#
    'Separador instrucciones $
    'Separador parametros orden !
    'Fin trama #
    cadena = contador_movimientos.ToString

    For Each row As DataGridViewRow In DataGridView1.Rows
        If Not row.IsNewRow Then
            If row.Cells(1).Value.ToString = "MOV" Then
                tipo = "M"
            ElseIf row.Cells(1).Value.ToString = "MANO" Then
                tipo = "H"
            Else
                tipo = ""
            End If
            If row.Cells(2).Value.ToString = "CIERRA" Then
                subtipo = "C"
            ElseIf row.Cells(2).Value.ToString = "ABRE" Then
                subtipo = "A"
            ElseIf row.Cells(2).Value.ToString = "IND" Then
                subtipo = "I"
            ElseIf row.Cells(2).Value.ToString = "COG" Then
                subtipo = "L"
            ElseIf row.Cells(2).Value.ToString = "REG" Then
                subtipo = "T"
            Else
                subtipo = ""
            End If
            cadena = cadena & "$" & row.Cells(0).Value.ToString &
"!" & tipo & "!" & subtipo & "!" & row.Cells(3).Value.ToString
```



```
        End If
    Next
    contador_caracteres = Len(cadena)
    cadena = cadena & "$" & contador_caracteres & "#"
    Return cadena
End Function

Private Sub boton_q_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_q.MouseDown
    valor_enviado = "Q"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND HAND Q" & vbCrLf & consola.Text
End Sub

Private Sub boton_q_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_q.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND HAND Q STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_w_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_w.MouseDown
    valor_enviado = "W"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND HAND W" & vbCrLf & consola.Text
End Sub

Private Sub boton_w_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_w.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND HAND W STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_e_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_e.MouseDown
    valor_enviado = "E"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND WRIST E" & vbCrLf & consola.Text
End Sub

Private Sub boton_e_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_e.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND WRIST E STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_r_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_r.MouseDown
    valor_enviado = "R"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND WRIST R" & vbCrLf & consola.Text
End Sub
```



```
Private Sub boton_r_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_r.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND WRIST R STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_a_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_a.MouseDown
    valor_enviado = "A"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND ELBOW A" & vbCrLf & consola.Text
End Sub

Private Sub boton_a_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_a.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND ELBOW A STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_s_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_s.MouseDown
    valor_enviado = "S"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND ELBOW S" & vbCrLf & consola.Text
End Sub

Private Sub boton_s_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_s.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND ELBOW S STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_d_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_d.MouseDown
    valor_enviado = "D"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND SHOULDER D" & vbCrLf &
consola.Text
End Sub

Private Sub boton_d_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_d.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND SHOULDER D STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_f_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_f.MouseDown
    valor_enviado = "F"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND SHOULDER F" & vbCrLf &
consola.Text
```



```
End Sub

Private Sub boton_f_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_f.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND SHOULDER F STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_z_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_z.MouseDown
    valor_enviado = "Z"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND BASE Z" & vbCrLf & consola.Text
End Sub

Private Sub boton_z_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_z.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND BASE Z STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_x_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_x.MouseDown
    valor_enviado = "X"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND BASE X" & vbCrLf & consola.Text
End Sub

Private Sub boton_x_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_x.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND BASE X STOP" & vbCrLf &
consola.Text
End Sub

Private Sub boton_led_MouseDown(sender As Object, e As MouseEventArgs)
Handles boton_led.MouseDown
    If estado_led = 0 Then
        valor_enviado = "L"
        estado_led = 1
        boton_led2.Visible = True
    Else
        valor_enviado = "K"
        estado_led = 0
        boton_led.Visible = True
    End If
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND LED ON" & vbCrLf & consola.Text
    boton_led.Visible = False
End Sub

Private Sub boton_led_MouseUp(sender As Object, e As MouseEventArgs)
Handles boton_led.MouseUp
    valor_enviado = "O"
    SerialPort1.Write(valor_enviado.ToString)
```



End Sub

```
Private Sub boton_led2_MouseDown(sender As Object, e As  
MouseEventArgs) Handles boton_led2.MouseDown  
    If estado_led = 0 Then  
        valor_enviado = "L"  
        estado_led = 1  
        boton_led2.Visible = True  
    Else  
        valor_enviado = "K"  
        estado_led = 0  
        boton_led.Visible = True  
    End If  
    SerialPort1.Write(valor_enviado.ToString)  
    consola.Text = "PC> " & "SEND LED OFF" & vbCrLf & consola.Text  
    boton_led2.Visible = False
```

End Sub

```
Private Sub boton_led2_MouseUp(sender As Object, e As MouseEventArgs)  
Handles boton_led2.MouseUp  
    valor_enviado = "O"  
    SerialPort1.Write(valor_enviado.ToString)
```

End Sub

'Evento para cambiar el modo de funcionamiento de la aplicación a Automático

```
Private Sub boton_mod0_MouseDown(sender As Object, e As  
MouseEventArgs) Handles boton_mod0.MouseDown  
    If arduino_programado = 1 Then  
        valor_enviado = "N"  
        estado_mod0 = 1  
        boton_mod02.Visible = True  
        boton_parada.Visible = True  
        modo_trabajo_label.Text = "AUTO"  
        SerialPort1.Write(valor_enviado.ToString)  
        consola.Text = "PC> " & "SEND AUTO MODE" & vbCrLf &  
consola.Text  
        boton_mod0.Visible = False  
        cambia_controles_programacion(False)  
    Else  
        MessageBox.Show("No puede cambiar a modo manual mientras  
no envíe una programación de automatización a arduino")  
    End If
```

End Sub

'Evento para cambiar el modo de funcionamiento de la aplicación a Manual

```
Private Sub boton_mod02_MouseDown(sender As Object, e As  
MouseEventArgs) Handles boton_mod02.MouseDown  
    valor_enviado = "M"  
    estado_mod0 = 0  
    boton_mod0.Visible = True  
    boton_parada.Visible = False  
    modo_trabajo_label.Text = "MANUAL"  
    SerialPort1.Write(valor_enviado.ToString)  
    consola.Text = "PC> " & "SEND MANUAL MODE" & vbCrLf &  
consola.Text  
    boton_mod02.Visible = False  
    cambia_controles_programacion(True)
```



```
End Sub

Private Sub boton_info_Click(sender As Object, e As EventArgs) Handles
boton_info.MouseClick
    valor_enviado = "I"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & valor_enviado.ToString & vbCrLf &
consola.Text
End Sub

Sub cambia_controles_programacion(ByVal valor As Boolean)
    If valor = True Then
        MessageBox.Show("Modo Manual activado")
    Else
        MessageBox.Show("Modo Manual desactivado")
    End If
    cierra_mano.Enabled = valor
    abre_mano.Enabled = valor
    boton_guardapos_0.Enabled = valor
    boton_guardapos_1.Enabled = valor
    boton_guardapos_2.Enabled = valor
    boton_elimina_item.Enabled = valor
    envia_programacion.Enabled = valor
    boton_papelera.Enabled = valor
    boton_salvar_configuracion.Enabled = valor
    boton_cargar_desde_archivo.Enabled = valor
    boton_info.Enabled = valor
    boton_copia_consola.Enabled = valor
    boton_q.Enabled = valor
    boton_w.Enabled = valor
    boton_led.Enabled = valor
    boton_led2.Enabled = valor
    boton_e.Enabled = valor
    boton_r.Enabled = valor
    boton_d.Enabled = valor
    boton_x.Enabled = valor
    boton_s.Enabled = valor
    boton_a.Enabled = valor
    boton_f.Enabled = valor
    boton_z.Enabled = valor
End Sub

Sub habilita_funciones()
    boton_q.Enabled = True
    boton_w.Enabled = True
    boton_led.Enabled = True
    boton_led2.Enabled = True
    boton_e.Enabled = True
    boton_r.Enabled = True
    boton_d.Enabled = True
    boton_x.Enabled = True
    boton_s.Enabled = True
    boton_a.Enabled = True
    boton_f.Enabled = True
    boton_z.Enabled = True
    boton_modos.Enabled = True
    boton_modos2.Enabled = True
    boton_parada.Enabled = True
    cierra_mano.Enabled = True
    abre_mano.Enabled = True
```





```
    boton_guardapos_0.Enabled = True
    boton_guardapos_1.Enabled = True
    boton_guardapos_2.Enabled = True
    boton_elimina_item.Enabled = True
    envia_programacion.Enabled = True
    boton_papelera.Enabled = True
    boton_salvar_configuracion.Enabled = True
    boton_cargar_desde_archivo.Enabled = True
    boton_info.Enabled = True
    boton_copia_consola.Enabled = True
End Sub

Sub deshabilita_funciones()
    boton_g.Enabled = False
    boton_w.Enabled = False
    boton_led.Enabled = False
    boton_led2.Enabled = False
    boton_e.Enabled = False
    boton_r.Enabled = False
    boton_d.Enabled = False
    boton_x.Enabled = False
    boton_s.Enabled = False
    boton_a.Enabled = False
    boton_f.Enabled = False
    boton_z.Enabled = False
    boton_mod0.Enabled = False
    boton_mod1.Enabled = False
    boton_mod2.Enabled = False
    boton_parada.Enabled = False
    cierra_mano.Enabled = False
    abre_mano.Enabled = False
    boton_guardapos_0.Enabled = False
    boton_guardapos_1.Enabled = False
    boton_guardapos_2.Enabled = False
    boton_elimina_item.Enabled = False
    envia_programacion.Enabled = False
    boton_papelera.Enabled = False
    boton_salvar_configuracion.Enabled = False
    boton_cargar_desde_archivo.Enabled = False
    boton_info.Enabled = False
    boton_copia_consola.Enabled = False
End Sub

'Timer1 controla las transmisiones recibidas a través del puerto
serie
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
    Dim tabla2() As String
    Dim tabla() As String
    Dim subtabla() As String
    Dim n As Integer
    Dim lectura As String = SerialPort1.ReadExisting
    Dim verifica As Integer
    Dim verifica2 As Integer
    Dim verifica3 As Integer
    Dim verifica4 As Integer
    Dim variableins As String
    Dim verifica_validez_registro As Integer
    verifica_validez_registro = 0
    'Muestra informacion en consola
    If lectura <> "" Then
```



```
        consola.Text = "ARDUINO> " & lectura & vbCrLf &
consola.Text
    End If
    'Segun la cabecera de recepcion, actua de uno u otro modo
    'PGM corresponde con información de posicionamiento para crear
posicion de trayectoria
    'Y-ACCEPTED corresponde con el estado de Arduino esperando
trama de configuración
    'Y-OK corresponde con la confirmación de programación correcta
    'Y-NOOK corresponde con información de programación erronea
    verifica = InStr(lectura, "PGM")
    verifica2 = InStr(lectura, "Y-ACCEPTED")
    verifica3 = InStr(lectura, "Y-OK")
    verifica4 = InStr(lectura, "Y-NOOK")
    If verifica <> 0 And estado_programando = 0 Then
        tabla2 = Split(lectura, "@")
        If UBound(tabla2, 1) <> 1 Or verifica <> 1 Then
            valor_enviado = "U"
            SerialPort1.Write(valor_enviado.ToString)
            consola.Text = "PC> " & "ERROR RECEIVE. GET VALUES TO
ADD" & vbCrLf & consola.Text
        Else
            tabla = Split(tabla2(1), ";")
            If UBound(tabla, 1) <> 4 Then
                valor_enviado = "U"
                SerialPort1.Write(valor_enviado.ToString)
                consola.Text = "PC> " & "ERROR RECEIVE. GET VALUES
TO ADD" & vbCrLf & consola.Text
            Else
                If intervalo_lectura_prog = 0 Then
                    cadena_control_errores_recepcion = lectura
                    intervalo_lectura_prog =
intervalo_lectura_prog + 1
                    valor_enviado = "U"
                    SerialPort1.Write(valor_enviado.ToString)
                Else
                    If cadena_control_errores_recepcion = lectura
Then
                        verifica_validez_registro = 1
                    Else
                        consola.Text = "PC> " & "ERROR RECEIVING
INFO. RESTARTING RECEPTION" & vbCrLf & consola.Text
                        'Se ha producido un error en la recepción
de la primera o la segunda trama
                        'Inicializo y programo de nuevo
                        intervalo_lectura_prog = 0
                        verifica_validez_registro = 0
                        valor_enviado = "U"
                        SerialPort1.Write(valor_enviado.ToString)
                    End If
                End If
            End If
        End If
    End If
    If verifica_validez_registro = 1 Then
        If control_prog_accion = 0 Then
            variableins = "IND"
        ElseIf control_prog_accion = 1 Then
            variableins = "COG"
        ElseIf control_prog_accion = 2 Then
            variableins = "REG"
        Else
            variableins = ""
        End If
    End If
End Sub
```



```
End If
If variableins <> "" Then
    contador_movimientos =
contador_movimientos + 1
    tabla2(1) = tabla2(1).Replace("M1:", "")
    tabla2(1) = tabla2(1).Replace("M2:", "")
    tabla2(1) = tabla2(1).Replace("M3:", "")
    tabla2(1) = tabla2(1).Replace("M4:", "")
    tabla2(1) = tabla2(1).Replace("M5:", "")

DataGridView1.Rows.Add(contador_movimientos.ToString, "MOV",
variableins, tabla2(1))
End If
End If
End If
End If

ElseIf verifica2 <> 0 Then
    valor_enviado = obtiene_info_prog()
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "SEND B" & vbCrLf & consola.Text
ElseIf verifica3 <> 0 Then
    MessageBox.Show("Transferencia completada con éxito")
    arduino_programado = 1
    estado_programando = 0
ElseIf verifica4 <> 0 Then
    MessageBox.Show("No ha sido posible programar arduino en
éste momento. Reinténtelo y, si el problema persiste, reinicie ambas
unidades")
    arduino_programado = 0
    estado_programando = 0
ElseIf lectura = "CONNECTED" And estado_programando = 0 Then
    estado_conexion.Text = "Estado: CONECTADO"
    estado_conexion.Visible = True
    estado_arduino = 2
    habilita_funciones()
ElseIf lectura = "PARADA EMERGENCIA" And estado_programando =
0 Then
    estado_conexion.Text = "Estado: CONECTADO (P)"
    boton_modos2.Visible = False
    boton_modos.Visible = True
    boton_parada.Visible = False
    modo_trabajo_label.Text = "MANUAL"
    cambia_controles_programacion(True)
    estado_modos = 0
ElseIf lectura <> "" And estado_programando = 0 And lectura <>
"CONNECTED" And verifica = 0 And verifica2 = 0 And verifica3 = 0 And
verifica4 = 0 And lectura <> "PARADA EMERGENCIA" Then
    tabla = Split(lectura, ";")
    If UBound(tabla, 1) <> 4 Then
        valor_enviado = "I"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "ERROR RECEIVE. GET INFO" &
vbCrLf & consola.Text
    Else
        For n = 0 To UBound(tabla, 1)
            subtabla = Split(tabla(n), ":")
            If subtabla(0) = "M5" Then
                M5_state.Text = subtabla(1)
            End If
        Next n
    End If
End If
```



```
        If subtabla(0) = "M4" Then
            M4_state.Text = subtabla(1)
        End If
        If subtabla(0) = "M3" Then
            M3_state.Text = subtabla(1)
        End If
        If subtabla(0) = "M2" Then
            M2_state.Text = subtabla(1)
        End If
        If subtabla(0) = "M1" Then
            M1_state.Text = subtabla(1)
        End If
    Next

        End If
    End If
End Sub

Private Sub boton_parada_MouseDown(sender As Object, e As
MouseEventArgs) Handles boton_parada.MouseDown
    valor_enviado = "P"
    SerialPort1.Write(valor_enviado.ToString)
    consola.Text = "PC> " & "STOP ALL INSTRUCTIONS" & vbCrLf &
consola.Text
End Sub

Private Sub ListBox1_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles ListBox1.SelectedIndexChanged
    boton_conectar.Visible = True
End Sub

'Evento del boton conexion en que se configura el puerto serie y
se intenta conectar al puerto seleccionado
Private Sub boton_conectar_Click(sender As Object, e As EventArgs)
Handles boton_conectar.Click
    estado_modos = 0
    SerialPort1.Close()
    SerialPort1.PortName = ListBox1.SelectedItem.ToString
    SerialPort1.BaudRate = 9600
    SerialPort1.DataBits = 8
    SerialPort1.Parity = Parity.None
    SerialPort1.StopBits = StopBits.One
    SerialPort1.Handshake = Handshake.None
    Try
        SerialPort1.Open()
        valor_enviado = "M"
        Timer1.Enabled = True
        If SerialPort1.IsOpen Then
            SerialPort1.Write(valor_enviado.ToString)
            consola.Text = "PC> " & "REQUEST CONNECT" & vbCrLf &
consola.Text

            boton_conectar.Visible = False
            boton_parada.Visible = False
            ListBox1.Visible = False
            estado_conexion.Text = "Estado: conectando..."
            estado_arduino = 1
            estado_conexion.Visible = True
            Timer2.Enabled = True
            modo_trabajo_label.Text = "MANUAL"
        End If
    End If
```



```
    Catch ex As Exception
        MessageBox.Show("Error conectando al puerto serie
seleccionado. Selecciona otro puerto serie.")
    End Try
End Sub

'Timer2 controla si se obtiene respuesta de conexión de arduino en
el puerto seleccionado
Private Sub Timer2_Tick(sender As Object, e As EventArgs) Handles
Timer2.Tick
    If estado_arduino = 2 Then
        Timer2.Enabled = False
    ElseIf estado_arduino = 1 Then
        Timer2.Enabled = False
        MessageBox.Show("No se ha encontrado Arduino en el puerto
seleccionado. Selecciona otro puerto serie.")
        boton_conectar.Visible = True
        ListBox1.Visible = True
        estado_conexion.Visible = False
    Else
        Timer2.Enabled = False
        MessageBox.Show("No se ha encontrado Arduino en el puerto
seleccionado. Selecciona otro puerto serie.")
        boton_conectar.Visible = True
        ListBox1.Visible = True
        estado_conexion.Visible = False
    End If
End Sub

'Funcion programación evento cierra mano
Private Sub cierra_mano_Click(sender As Object, e As EventArgs)
Handles cierra_mano.Click
    If (contador_movimientos < total_movimientos_permitidos) Then
        contador_movimientos = contador_movimientos + 1
        DataGridView1.Rows.Add(contador_movimientos.ToString,
"MANO", "CIERRA", "")
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
movimientos programados")
    End If
End Sub

'Funcion programación evento abre mano
Private Sub abre_mano_Click(sender As Object, e As EventArgs) Handles
abre_mano.Click
    If (contador_movimientos < total_movimientos_permitidos) Then
        contador_movimientos = contador_movimientos + 1
        DataGridView1.Rows.Add(contador_movimientos.ToString,
"MANO", "ABRE", "")
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
movimientos programados")
    End If
End Sub

'Funcion envia programación a arduino
Private Sub envia_programacion_Click(sender As Object, e As EventArgs)
Handles envia_programacion.Click
    If contador_movimientos > 0 Then
        'Si es el primer intento
```



```
        If estado_programando = 0 Then
            estado_programando = 1
            valor_enviado = "Y"
            SerialPort1.Write(valor_enviado.ToString)
            consola.Text = "PC> " & "PROGRAMMING REQUEST COM" &
vbCrLf & consola.Text
        Else
            'Si se produce algún error intentando programar el
            arduino
            valor_enviado = obtiene_info_prog()
            SerialPort1.Write(valor_enviado.ToString)
            consola.Text = "PC> " & "SEND B" & vbCrLf &
consola.Text
        End If
    Else
        MessageBox.Show("Antes de enviar la programación, genere
una trayectoria de automatización")
    End If
End Sub

'Funcion guarda posicion tipo 0; envia solicitud de posición a
Arduino y almacena al recibir en Timer1
Private Sub boton_guardapos_0_Click(sender As Object, e As EventArgs)
Handles boton_guardapos_0.Click
    If (contador_movimientos < total_movimientos_permitidos) Then
        intervalo_lectura_prog = 0
        control_prog_accion = 0
        valor_enviado = "U"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "GET VALUES TO ADD POS 0" & vbCrLf
& consola.Text
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
movimientos programados")
    End If
End Sub

'Funcion guarda posicion tipo 1; envia solicitud de posición a
Arduino y almacena al recibir en Timer1
Private Sub boton_guardapos_1_Click(sender As Object, e As EventArgs)
Handles boton_guardapos_1.Click
    If (contador_movimientos < total_movimientos_permitidos) Then
        intervalo_lectura_prog = 0
        control_prog_accion = 1
        valor_enviado = "U"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "GET VALUES TO ADD POS 1" & vbCrLf
& consola.Text
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
movimientos programados")
    End If
End Sub

'Funcion guarda posicion tipo 2; envia solicitud de posición a
Arduino y almacena al recibir en Timer1
Private Sub boton_guardapos_2_Click(sender As Object, e As EventArgs)
Handles boton_guardapos_2.Click
    If (contador_movimientos < total_movimientos_permitidos) Then
        intervalo_lectura_prog = 0
```



```
        control_prog_accion = 2
        valor_enviado = "U"
        SerialPort1.Write(valor_enviado.ToString)
        consola.Text = "PC> " & "GET VALUES TO ADD POS 2" & vbCrLf
    & consola.Text
    Else
        MessageBox.Show("Ha alcanzado el número máximo de
movimientos programados")
    End If
End Sub

Private Sub boton_copia_consola_Click(sender As Object, e As
EventArgs) Handles boton_copia_consola.Click
    Clipboard.SetText(consola.Text)
End Sub

Private Sub boton_papelera_Click(sender As Object, e As EventArgs)
Handles boton_papelera.Click
    If DataGridView1.Rows.Count > 0 Then
        If MsgBox("¿Está seguro que desea eliminar la programación
completa?", MsgBoxStyle.YesNo, "Confirmación") = MsgBoxResult.Yes Then
            DataGridView1.Rows.Clear()
            contador_movimientos = 0
        End If
    End If
End Sub

Private Sub boton_elimina_item_Click(sender As Object, e As EventArgs)
Handles boton_elimina_item.Click
    If DataGridView1.Rows.Count > 0 Then
        If MsgBox("¿Está seguro que desea eliminar el último
elemento insertado?", MsgBoxStyle.YesNo, "Confirmación") =
MsgBoxResult.Yes Then
            DataGridView1.Rows.RemoveAt(contador_movimientos - 1)
            contador_movimientos = contador_movimientos - 1
        End If
    End If
End Sub

Private Sub boton_salvar_configuracion_Click(sender As Object, e As
EventArgs) Handles boton_salvar_configuracion.Click
    Dim saveFileDialog1 As New SaveFileDialog()
    saveFileDialog1.Filter = "arduino|*.jyn"
    saveFileDialog1.Title = "Grabar configuración"
    saveFileDialog1.ShowDialog()

    If saveFileDialog1.FileName <> "" Then
        Try
            Using archivo As StreamWriter = New
StreamWriter(saveFileDialog1.FileName)

                Dim linea As String = String.Empty
                With DataGridView1
                    For fila As Integer = 0 To .RowCount - 1
                        linea = String.Empty
                        For col As Integer = 0 To .Columns.Count -
1
                            linea = linea & .Item(col,
fila).Value.ToString & DELIMITADOR
                        Next
                    End For
                End With
            End Using
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End If
End Sub
```



```
                With archivo
                    .WriteLine(linea.ToString)
                End With
            Next
        End With
    End Using

    'error
    Catch ex As Exception
        MsgBox(ex.Message.ToString, MsgBoxStyle.Critical)
    End Try
End If
End Sub

Private Sub boton_cargar_desde_archivo_Click(sender As Object, e As
EventArgs) Handles boton_cargar_desde_archivo.Click
    Dim openFileDialog1 As New OpenFileDialog()
    Dim fieldValues As String()
    Dim miReader As IO.StreamReader
    openFileDialog1.Filter = "arduino|*.jyn"
    openFileDialog1.Title = "Cargar configuración"

    If MsgBox("¿Está seguro que desea eliminar la programación
completa y cargarla desde un archivo de texto?", MsgBoxStyle.YesNo,
"Confirmación") = MsgBoxResult.Yes Then
        If DataGridView1.Rows.Count > 0 Then
            DataGridView1.Rows.Clear()
            contador_movimientos = 0
        End If
        openFileDialog1.ShowDialog()
        If openFileDialog1.FileName <> "" Then
            Try
                miReader = File.OpenText(openFileDialog1.FileName)

                While miReader.Peek() <> -1
                    fieldValues =
miReader.ReadLine().Split(DELIMITADOR)

                    DataGridView1.Rows.Add(fieldValues(0).ToString,
fieldValues(1).ToString, fieldValues(2).ToString,
fieldValues(3).ToString)
                    contador_movimientos = contador_movimientos +
1
                End While
                miReader.Close()

                'error
                Catch ex As Exception
                    MsgBox(ex.Message.ToString, MsgBoxStyle.Critical)
                End Try
            End If
        End If
    End Sub
End Class
```





## 10.5 DATASHEET L293D

### L293, L293D QUADRUPLE HALF-H DRIVERS

SLRS008B – SEPTEMBER 1986 – REVISED JUNE 2002

- Featuring Unitorde L293 and L293D Products Now From Texas Instruments
- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- Thermal Shutdown
- High-Noise-Immunity Inputs
- Functional Replacements for SGS L293 and SGS L293D
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

#### description

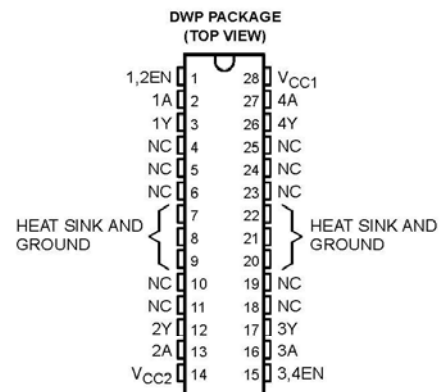
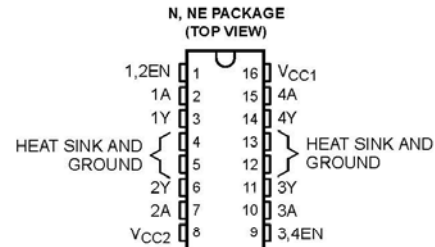
The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.

A  $V_{CC1}$  terminal, separate from  $V_{CC2}$ , is provided for the logic inputs to minimize device power dissipation.

The L293 and L293D are characterized for operation from 0°C to 70°C.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

**TEXAS INSTRUMENTS**  
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

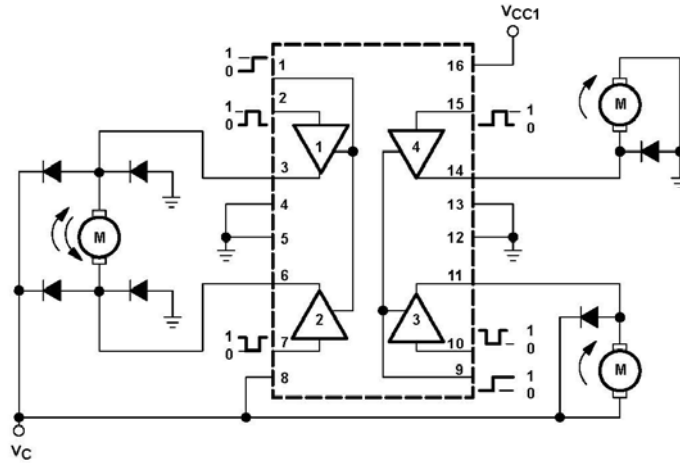
Copyright © 2002, Texas Instruments Incorporated

1

**L293, L293D  
QUADRUPLE HALF-H DRIVERS**

SLRS008B – SEPTEMBER 1986 – REVISED JUNE 2002

block diagram



NOTE: Output diodes are internal in L293D.

**TEXAS INSTRUMENTS  
AVAILABLE OPTIONS**

TA	PACKAGE
	PLASTIC DIP (NE)
0°C to 70°C	L293NE L293DNE

**Unitrode Products  
from Texas Instruments**

**AVAILABLE OPTIONS**

TA	PACKAGED DEVICES	
	SMALL OUTLINE (DWP)	PLASTIC DIP (N)
0°C to 70°C	L293DWP L293DDWP	L293N L293DN

The DWP package is available taped and reeled. Add the suffix TR to device type (e.g., L293DWPTR).



L293, L293D  
QUADRUPLE HALF-H DRIVERS

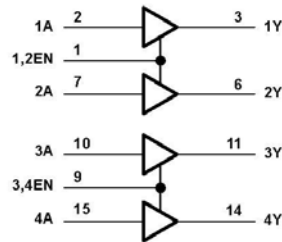
SLRS008B – SEPTEMBER 1986 – REVISED JUNE 2002

FUNCTION TABLE  
(each driver)

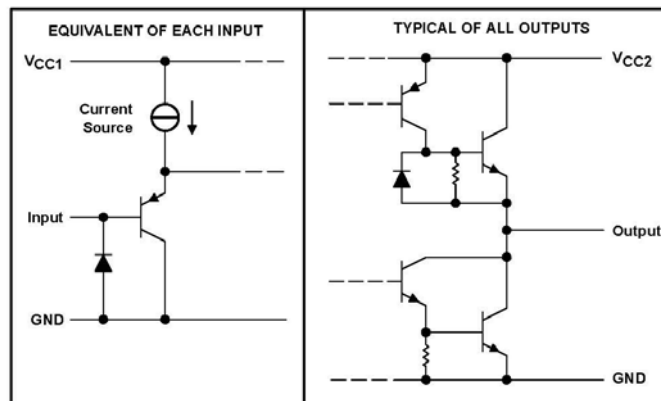
INPUTS†		OUTPUT Y
A	EN	
H	H	H
L	H	L
X	L	Z

H = high level, L = low level, X = irrelevant,  
Z = high impedance (off)  
† In the thermal shutdown mode, the output is  
in the high-impedance state, regardless of  
the input levels.

logic diagram



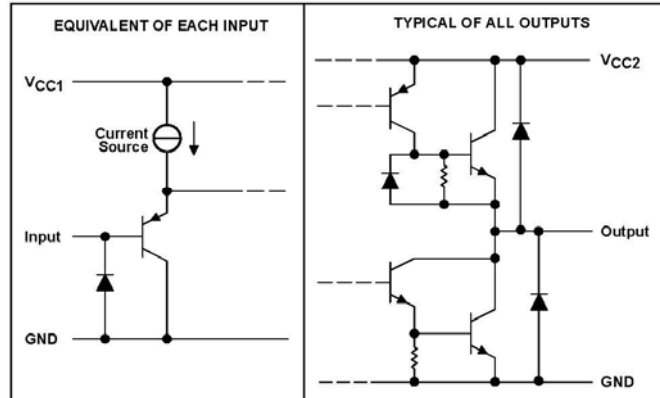
schematics of inputs and outputs (L293)



**L293, L293D  
QUADRUPLE HALF-H DRIVERS**

SLRS008B – SEPTEMBER 1986 – REVISED JUNE 2002

schematics of inputs and outputs (L293D)



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, $V_{CC1}$ (see Note 1)	36 V
Output supply voltage, $V_{CC2}$	36 V
Input voltage, $V_I$	7 V
Output voltage range, $V_O$	-3 V to $V_{CC2} + 3$ V
Peak output current, $I_O$ (nonrepetitive, $t \leq 5$ ms): L293	$\pm 2$ A
Peak output current, $I_O$ (nonrepetitive, $t \leq 100 \mu\text{s}$ ): L293D	$\pm 1.2$ A
Continuous output current, $I_O$ : L293	$\pm 1$ A
Continuous output current, $I_O$ : L293D	$\pm 600$ mA
Continuous total dissipation at (or below) 25°C free-air temperature (see Notes 2 and 3)	2075 mW
Continuous total dissipation at 80°C case temperature (see Note 3)	5000 mW
Maximum junction temperature, $T_J$	150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds	260°C
Storage temperature range, $T_{stg}$	-65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

- NOTES:
1. All voltage values are with respect to the network ground terminal.
  2. For operation above 25°C free-air temperature, derate linearly at the rate of 16.6 mW/°C.
  3. For operation above 25°C case temperature, derate linearly at the rate of 7.14 mW/°C. Due to variations in individual device electrical characteristics and thermal resistance, the built-in thermal overload protection may be activated at power levels slightly above or below the rated dissipation.



L293, L293D  
QUADRUPLE HALF-H DRIVERS

SLRS008B – SEPTEMBER 1986 – REVISED JUNE 2002

recommended operating conditions

		MIN	MAX	UNIT
Supply voltage	V <sub>CC1</sub>	4.5	7	V
	V <sub>CC2</sub>	V <sub>CC1</sub>	36	
V <sub>IH</sub>	High-level input voltage	V <sub>CC1</sub> ≤ 7 V	2.3 V <sub>CC1</sub>	V
		V <sub>CC1</sub> ≥ 7 V	2.3	7
V <sub>IL</sub>	Low-level output voltage	-0.3†	1.5	V
T <sub>A</sub>	Operating free-air temperature	0	70	°C

† The algebraic convention, in which the least positive (most negative) designated minimum, is used in this data sheet for logic voltage levels.

electrical characteristics, V<sub>CC1</sub> = 5 V, V<sub>CC2</sub> = 24 V, T<sub>A</sub> = 25°C

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
V <sub>OH</sub>	High-level output voltage	L293: I <sub>OH</sub> = -1 A L293D: I <sub>OH</sub> = -0.6 A		V <sub>CC2</sub> -1.8	V <sub>CC2</sub> -1.4		V
V <sub>OL</sub>	Low-level output voltage	L293: I <sub>OL</sub> = 1 A L293D: I <sub>OL</sub> = 0.6 A			1.2	1.8	V
V <sub>OKH</sub>	High-level output clamp voltage	L293D: I <sub>OK</sub> = -0.6 A			V <sub>CC2</sub> + 1.3		V
V <sub>OKL</sub>	Low-level output clamp voltage	L293D: I <sub>OK</sub> = 0.6 A			1.3		V
I <sub>IH</sub>	High-level input current	A	V <sub>I</sub> = 7 V		0.2	100	μA
		EN			0.2	10	
I <sub>IL</sub>	Low-level input current	A	V <sub>I</sub> = 0		-3	-10	μA
		EN			-2	-100	
I <sub>CC1</sub>	Logic supply current	I <sub>O</sub> = 0	All outputs at high level		13	22	mA
			All outputs at low level		35	60	
			All outputs at high impedance		8	24	
I <sub>CC2</sub>	Output supply current	I <sub>O</sub> = 0	All outputs at high level		14	24	mA
			All outputs at low level		2	6	
			All outputs at high impedance		2	4	

switching characteristics, V<sub>CC1</sub> = 5 V, V<sub>CC2</sub> = 24 V, T<sub>A</sub> = 25°C

PARAMETER	TEST CONDITIONS	L293NE, L293DNE			UNIT
		MIN	TYP	MAX	
t <sub>PLH</sub>	Propagation delay time, low-to-high-level output from A input		800		ns
t <sub>PHL</sub>	Propagation delay time, high-to-low-level output from A input		400		ns
t <sub>TLH</sub>	Transition time, low-to-high-level output		300		ns
t <sub>THL</sub>	Transition time, high-to-low-level output		300		ns

switching characteristics, V<sub>CC1</sub> = 5 V, V<sub>CC2</sub> = 24 V, T<sub>A</sub> = 25°C

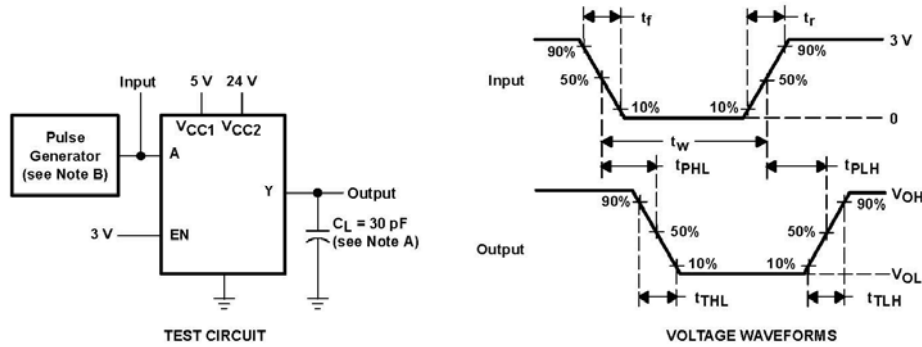
PARAMETER	TEST CONDITIONS	L293DWP, L293N L293DDWP, L293DN			UNIT
		MIN	TYP	MAX	
t <sub>PLH</sub>	Propagation delay time, low-to-high-level output from A input		750		ns
t <sub>PHL</sub>	Propagation delay time, high-to-low-level output from A input		200		ns
t <sub>TLH</sub>	Transition time, low-to-high-level output		100		ns
t <sub>THL</sub>	Transition time, high-to-low-level output		350		ns



### L293, L293D QUADRUPLÉ HALF-H DRIVERS

SLRS008B – SEPTEMBER 1986 – REVISED JUNE 2002

#### PARAMETER MEASUREMENT INFORMATION



- NOTES: A.  $C_L$  includes probe and jig capacitance.  
B. The pulse generator has the following characteristics:  $t_r \leq 10$  ns,  $t_f \leq 10$  ns,  $t_w = 10$   $\mu$ s, PRR = 5 kHz,  $Z_O = 50$   $\Omega$ .

Figure 1. Test Circuit and Voltage Waveforms



# 11. BIBLIOGRAFÍA

A continuación se enumeran los libros, documentos y páginas web consultadas de cara a la realización del proyecto:

- Páginas web:
  - o <https://sites.google.com/site/picuino/ziegler-nichols>
  - o <https://sites.google.com/site/proyectosroboticos/cinematica-inversa-i>
  - o [https://controls.engin.umich.edu/wiki/index.php/PIDTuningClassical#Ziegler-Nichols\\_Method](https://controls.engin.umich.edu/wiki/index.php/PIDTuningClassical#Ziegler-Nichols_Method)
  - o <http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Gu%C3%ADa-de-uso-PID-para-Arduino.pdf>
  - o <http://msdn.microsoft.com/>
  - o <http://arduino.cc/>
  
- Libros:
  - o Fundamentos de Robótica, Antonio Barrientos, **Universidad Politécnica de Madrid, ISBN: 8448156366.**
  - o Ingeniería de Control Moderna, Katsuhiko Ogata, ISBN: 8420536784
  
- Datasheet:
  - o Resistencias Variables CA9.
  - o Puente en H L293D.
  - o Regulador de tensión LM7805.