

Trabajo Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

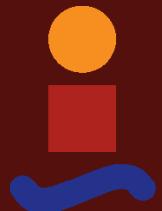
Integración de robot manipulador con
posicionador basado en Arduino

Autor: Antonio Pérez García

Tutor: Luis Fernando Castaño Castaño

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022



Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Integración de robot manipulador con posicionador basado en Arduino

Autor:
Antonio Pérez García

Tutor:
Luis Fernando Castaño Castaño
Profesor Contratado Doctor

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Integración de robot manipulador con posicionador basado en Arduino

Autor: Antonio Pérez García
Tutor: Luis Fernando Castaño Castaño

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

El trabajo de fin de grado es la culminación de mucho esfuerzo realizado durante años de mis padres Antonio y Rosario. Agradezco su paciencia y apoyo durante todo este tiempo.

Antonio Pérez García

Sevilla, 2022

Resumen

En el laboratorio de automática

Abstract

In the lab of automatics

Índice Abreviado

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
<i>Notación</i>	XI
1 Introducción	1
1.1 Modos de funcionamiento	1
1.2 Distribución de los controles	2
2 Descripción del hardware	5
2.1 Arduino Mega 2560	5
2.2 Ethernet Shield de Arduino	7
2.3 Motor DC y Driver L298N	8
2.4 Encoder Rotativo Incremental	9
2.5 Calibre digital	10
2.6 LCD	11
2.7 Sensor fotoeléctrico OMRON	11
2.8 Cinta transportadora con elementos integrados	12
2.9 Convertidor CC-CC con salida USB	13
2.10 Controladora IRC5C	13
2.11 Optoacoplador TLP621-4	13
3 Desarrollo de placa de conexiones	15
3.1 Software utilizado. KiCAD.	15
3.2 Señales	15
3.3 Señales digitales en 24V y en 5V	16
3.4 Calibre digital	19

3.5 Placas finales	21
4 Planificación de caja	23
4.1 Base	23
4.2 Tapa	24
5 Desarrollo en Arduino	27
5.1 Diagrama de flujo	27
5.2 Programa final	28
6 Desarrollo en Robotstudio	43
6.1 Funcionamiento sin microcontrolador	43
6.2 Funcionamiento con microcontrolador	45
7 Resultados	49
8 Conclusiones	51
<i>Índice de Figuras</i>	53
<i>Índice de Tablas</i>	55
<i>Índice de Códigos</i>	57
<i>Bibliografía</i>	59

Índice

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
<i>Notación</i>	XI
1 Introducción	1
1.1 Modos de funcionamiento	1
1.1.1 Modo local con microcontrolador	2
1.1.2 Modo remoto con microcontrolador	2
1.1.3 Modo local sin microcontrolador	2
1.1.4 Modo remoto sin microcontrolador	2
1.2 Distribución de los controles	2
2 Descripción del hardware	5
2.1 Arduino Mega 2560	5
2.1.1 Alimentación	6
2.1.2 Memoria	6
2.1.3 Entradas y Salidas	6
2.1.4 Comunicación	7
2.2 Ethernet Shield de Arduino	7
2.3 Motor DC y Driver L298N	8
2.3.1 Conexionado del módulo L298N	8
2.4 Encoder Rotativo Incremental	9
2.4.1 Especificaciones técnicas	9
2.4.2 Conexionado del Encoder	10
2.5 Calibre digital	10
2.6 LCD	11
2.7 Sensor fotoeléctrico OMRON	11
2.7.1 Conexiones del sensor fotoeléctrico	11
2.8 Cinta transportadora con elementos integrados	12
2.9 Convertidor CC-CC con salida USB	13
2.10 Controladora IRC5C	13
2.11 Optoacoplador TLP621-4	13
3 Desarrollo de placa de conexiones	15

3.1	Software utilizado. KiCAD.	15
3.2	Señales	15
3.3	Señales digitales en 24V y en 5V	16
3.3.1	Conversión de 24V a 5V	17
3.3.2	Función sin microcontrolador y emergencia	18
3.4	Calibre digital	19
3.4.1	Alimentación calibre	19
3.4.2	Amplificación señales calibre	20
3.5	Placas finales	21
3.5.1	Placa <i>Tapa</i>	21
3.5.2	Placa <i>Fondo</i>	22
4	Planificación de caja	23
4.1	Base	23
4.2	Tapa	24
5	Desarrollo en Arduino	27
5.1	Diagrama de flujo	27
5.2	Programa final	28
6	Desarrollo en Robotstudio	43
6.1	Funcionamiento sin microcontrolador	43
6.2	Funcionamiento con microcontrolador	45
7	Resultados	49
8	Conclusiones	51
<i>Índice de Figuras</i>		53
<i>Índice de Tablas</i>		55
<i>Índice de Códigos</i>		57
<i>Bibliografía</i>		59

Notación

ABB	Asea Brown Boveri
CC	Corriente Continua
E/S	Entrada/Salida
EEPROM	Electrically Erasable Programmable Read-Only Memory
g	Gramo
GND	Tierra
GPIO	General-Purpose Input/Output. Pines de uso general
I2C	Inter-Integrated Circuit
ICSP	In-circuit Serial Programming
IP	Internet Protocol
kB	Kilobyte
kΩ	Kilohmio
LCD	Liquid-Crystal Display. Pantalla de cristal líquido
LED	Light-Emitting Diode
mA	Miliamperio
MHz	Megahercios
mm	Milímetros
PWM	Modulación por ancho de pulso
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
TCP	Transmission Control Protocol
TTL	Transistor-Transistor Logic
TWI	Two-Wire Interface
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
V	Voltio
ℂ	Cuerpo de los números complejos
$\ \mathbf{v}\ $	Norma del vector \mathbf{v}
$\langle \mathbf{v}, \mathbf{w} \rangle$	Producto escalar de los vectores \mathbf{v} y \mathbf{w}
$ \mathbf{A} $	Determinante de la matriz cuadrada \mathbf{A}
$\det(\mathbf{A})$	Determinante de la matriz (cuadrada) \mathbf{A}
\mathbf{A}^\top	Transpuesto de \mathbf{A}
\mathbf{A}^{-1}	Inversa de la matriz \mathbf{A}
\mathbf{A}^\dagger	Matriz pseudoinversa de la matriz \mathbf{A}
\mathbf{A}^H	Transpuesto y conjugado de \mathbf{A}

\mathbf{A}^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
□	Fin de la solución
e.o.c.	En cualquier otro caso
e	número e
e^{jx}	Exponencial compleja
$e^{j2\pi x}$	Exponencial compleja con 2π
e^{-jx}	Exponencial compleja negativa
$e^{-j2\pi x}$	Exponencial compleja negativa con 2π
IRe	Parte real
Ilm	Parte imaginaria
sen	Función seno
tg	Función tangente
arc tg	Función arco tangente
$\sin^y x$	Función seno de x elevado a y
$\cos^y x$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
$\frac{\partial y}{\partial x}$	Derivada parcial de y respecto a x
x°	Notación de grado, x grados.
$\Pr(A)$	Probabilidad del suceso A
$E[X]$	Valor esperado de la variable aleatoria X
σ_X^2	Varianza de la variable aleatoria X
$\sim f_X(x)$	Distribuido siguiendo la función densidad de probabilidad $f_X(x)$
$\mathcal{N}(m_X, \sigma_X^2)$	Distribución gaussiana para la variable aleatoria X , de media m_X y varianza σ_X^2
\mathbf{I}_n	Matriz identidad de dimensión n
diag(\mathbf{x})	Matriz diagonal a partir del vector \mathbf{x}
diag(\mathbf{A})	Vector diagonal de la matriz \mathbf{A}
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
$\stackrel{\text{def}}{=}$	Igual por definición
$\ \mathbf{x}\ $	Norma-2 del vector \mathbf{x}
$ \mathbf{A} $	Cardinal, número de elementos del conjunto \mathbf{A}
$\mathbf{x}_i, i = 1, 2, \dots, n$	Elementos i , de 1 a n , del vector \mathbf{x}
$d\mathbf{x}$	Diferencial de x
\leqslant	Menor o igual
\geqslant	Mayor o igual
\backslash	Backslash
\Leftrightarrow	Si y sólo si
$x = a + 3 \stackrel{\uparrow}{=} 4$	Igual con explicación
$\frac{a}{b}$	Fracción con estilo pequeño, a/b
Δ	Incremento

$b \cdot 10^a$	Formato científico
\xrightarrow{x}	Tiende, con x
O	Orden
TM	Trade Mark
$\mathbb{E}[x]$	Esperanza matemática de x
C_x	Matriz de covarianza de x
R_x	Matriz de correlación de x
σ_x^2	Varianza de x

1 Introducción

En las prácticas de laboratorio realizadas por parte de alumnos durante la docencia de los cursos de Robótica del Departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de Sevilla, los alumnos deben programar el movimiento de un brazo robótico presente en dicho laboratorio. El objetivo es que una pieza sea trasladada por el robot desde un punto de la mesa de trabajo hacia una segunda posición. El proceso se realiza colocando manualmente la pieza, con los problemas que ello conlleva. Por un lado, la precisión es cuestionable, ya que el propio alumno no tiene una referencia sobre la cual poder repetir el proceso de forma eficaz y el error introducido al sistema es alto. Por otro lado, al invadir el espacio de trabajo del brazo robótico continuamente se producen riesgos innecesarios impropios de las normas de seguridad en la industria.

Este trabajo es la continuación de Tapia[4] e Hinojosa[5], que sentaron las bases del proyecto. En esta ocasión, el enfoque es en la implementación sobre los equipos del laboratorio. Por ello, el sistema creado debe quedar compacto en una caja donde se realicen las conexiones electrónicas y todos los dispositivos. Además el sistema debe tener componentes fácilmente sustituibles para facilitar las reparaciones.

El sistema cuenta con las siguientes características:

- Posicionamiento de piezas en medidas de ejes X e Y que el usuario requiera para interactuar con el robot.
- Conexión entre Arduino y RobotStudio mediante protocolo TCP/IP para comunicaciones.
- Funcionamiento sin Arduino mediante señales digitales del robot.
- Funcionamiento sin conexión directa entre RobotStudio y Arduino.

1.1 Modos de funcionamiento

Todas las características especificadas previamente no pueden cumplirse al mismo tiempo ya que existirían conflictos entre las mismas, por lo que el sistema debe tener modos de funcionamiento específicos y diferenciados en los que se activen o desactiven dichas características.

La primera consideración es determinar el dispositivo que gobierna el sistema o *máster*. Por ello, se puede diferenciar cuando el máster es la controladora del robot (o RobotStudio durante una simulación) o el sistema caja (Arduino o la propia electrónica interna). Respectivamente serán los modos remoto y local.

Por otro lado, como el microcontrolador presente en el Arduino puede estar funcionando o no, se deben añadir las dos posibilidades. Se tiene el modo con microcontrolador y sin microcontrolador.

En total, se cuenta con cuatro modos de funcionamiento que se describen a continuación.

1.1.1 Modo local con microcontrolador

Las órdenes del sistema están proporcionadas por los periféricos de entrada como pueden ser los botones o interruptores bipolares presentes en la caja, mientras que el microcontrolador es el encargado de gestionar el posicionamiento y mover el motor cuando le sea indicado.

En caso de que esté disponible la conexión con la controladora del robot, el Arduino comunica mediante conexión TCP/IP la posición de la pieza en los ejes X e Y además del estado del sensor fotoeléctrico y del sistema.

1.1.2 Modo remoto con microcontrolador

El gobierno del sistema pasa a ser parte de la controladora del robot, convirtiendo al microcontrolador en esclavo. El microcontrolador sigue encargándose del posicionamiento y movimiento del motor, pero las órdenes pasan a ser recibidas mediante conexión TCP/IP.

Como en el caso anterior, se envía la posición, estado del sistema y del sensor fotoeléctrico mediante conexión TCP/IP a la controladora.

1.1.3 Modo local sin microcontrolador

Al no tener el microcontrolador operativo, el posicionamiento deja de funcionar y las funciones del sistema pasan a ser más básicas. El movimiento del motor queda a cargo de la electrónica interna del sistema. Para interactuar con el motor y moverlo se realizará mediante los pulsadores de avance y retroceso.

El robot queda no comunicado y solo recibe las señales digitales.

1.1.4 Modo remoto sin microcontrolador

Este modo comparte la mayoría de características con el anterior y, físicamente es idéntico. La única diferencia es en el modo de uso, ya que el control del movimiento de la cinta se realiza con las señales digitales de avance y retroceso que permiten el movimiento.

1.2 Distribución de los controles

Para una interacción con el sistema se crea una interfaz hombre-máquina que permite cambiar entre los distintos modos de funcionamiento, visualizar información, controlar el sistema en el modo local y realizar paradas de emergencia en caso de ser necesario. Para ello, se prototipa la interfaz mostrada en la figura 1.1, que posteriormente se aplicará en el diseño de la caja.

En este diseño se observa un LCD que será el encargado de mostrar la información relevante como la posición o el modo en el que se encuentra el sistema. Por otra parte, hay dos botones y una flecha de selección que permiten avanzar por los distintos menús y seleccionar las opciones durante el modo local. Además, las flechas serán las responsables del movimiento en el modo sin microcontrolador local. También se dispone de dos interruptores de dos posiciones que permiten seleccionar los modos "local/remoto" y "con microcontrolador/sin microcontrolador". Por último,

una seta de emergencia es imprescindible para realizar una parada de emergencia en caso necesario. La seta está situada en un lugar de fácil acceso y que no interfiera en el uso normal del sistema.

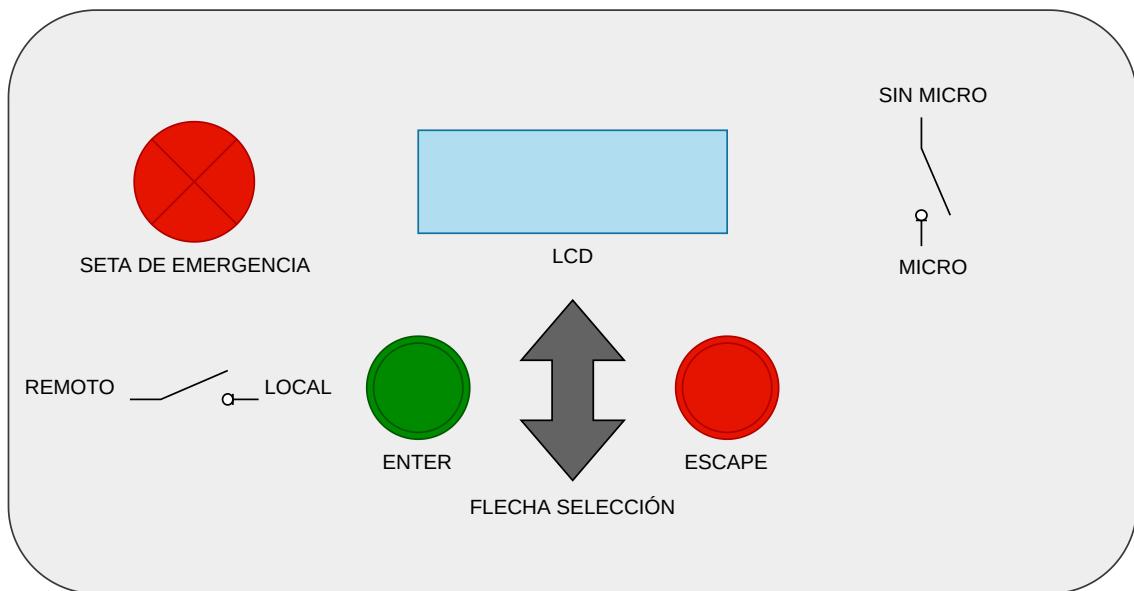


Figura 1.1 Interfaz hombre-máquina. Distribución en la tapa..

Además, se debe crear una serie de conexiones que permitan la interacción robot-cinta para el modo remoto sin microcontrolador. Esto se debe realizar mediante un conector que permita acceder a dichos pines de forma sencilla y queden expuestos. Para ello, se introducirá un conector similar a la figura 1.2. Los pines que debe contener son:

- Retroceso.
- Avance.
- Local.
- Micro.
- Emergencia.
- Fotoeléctrico.

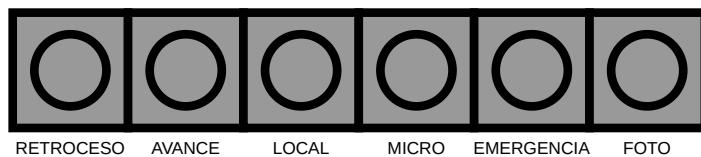


Figura 1.2 Distribución de conexiones digitales.

2 Descripción del hardware

Una vez conocidas las funciones que debe desempeñar el sistema es importante definir el hardware a usar para que sea capaz de cumplimentar los requerimientos establecidos. Los principales dispositivos utilizados en el proyecto son los siguientes:

2.1 Arduino Mega 2560

Se trata de una placa de desarrollo que cuenta con el microcontrolador ATmega2560 y todo lo necesario para prototipar un sistema. Cuenta con 54 pines de entrada y salida (GPIO), de los cuales 15 pueden utilizarse como salida PWM, 4 puertos UART, oscilador de 16 MHz, conexión USB tipo B, pines ICSP de programación y botón de reinicio.

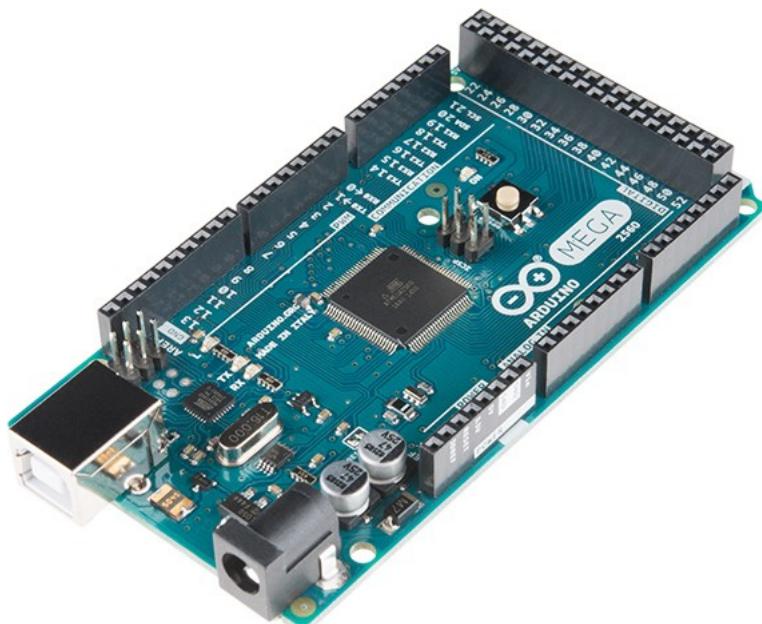


Figura 2.1 Arduino Mega 2560.

Tabla 2.1 Características Arduino Mega 2560.

Microcontrolador	ATmega2560
Tensión de funcionamiento	5 V
Voltaje de entrada recomendado	7-12 V
Voltaje de entrada límite	6-20 V
Pines de E/S digitales	54 (de los cuales 15 con salida PWM)
Pines de entrada analógica	16
Corriente CC por pin	20 mA
Corriente CC para pin 3.3 V	50 mA
Memoria Flash	256 kB, 8 kB utilizados por el gestor de arranque
SRAM	8 kB
EEPROM	4 kB
Frecuencia de reloj	16 MHz
Longitud	101.52 mm
Anchura	53.3 mm
Peso	37 g

2.1.1 Alimentación

El Arduino Mega 2560 puede ser alimentado mediante la conexión USB tipo B proporcionada por un ordenador o por una fuente de alimentación externa. Además, puede ser alimentado por los pines de alimentación de la placa, que se describen a continuación:

- VIN. Entrada de tensión de la placa cuando hay alimentación externa.
- 5V. Pin que produce 5V regulados.
- 3V3. Pin que produce 3.3V regulados con un consumo máximo de 50 mA.
- GND. Pin de conexión a masa.
- IOREF. Referencia de tensión de trabajo del microcontrolador.

2.1.2 Memoria

El ATmega2560 cuenta con 256 kB de memoria flash de los cuales 8 kB se usan para el gestor de arranque, 8 kB de memoria SRAM y 4 kB de EEPROM.

2.1.3 Entradas y Salidas

El Arduino Mega cuenta con 54 pines que pueden ser utilizados como salida o entrada digital mediante las funciones `pinMode()`, `digitalWrite()` y `digitalRead()`. Los niveles lógicos son de 5V. Cada pin soporta una corriente de 20 mA y cuenta con una resistencia de pull-up de entre 20 y 50 kΩ. Además, algunos pines tienen funciones especiales:

- Comunicación serie. Se usa para recibir y transmitir datos en serie TTL. Son las parejas 0 y 1, 14 y 15, 16 y 17 y 18 y 19. Los pines 0 y 1 son los correspondientes a la comunicación con el conversor USB-TTL integrado en la placa
- Interrupciones externas. Se utilizan para activar interrupciones en el software. Son los pines 2, 3, 18, 19, 20 y 21.

- Salida PWM. Proporcionan una salida modulada con 8 bits de precisión con la función `analogWrite()`. Son los pines del 2 al 13 y del 44 al 46.
- Comunicación SPI. Pines 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Permiten comunicación SPI con otros dispositivos utilizando la biblioteca SPI.
- LED integrado. En el pin 13 hay un LED integrado que puede encenderse con un valor HIGH y apagarse con un valor LOW.
- Comunicación TWI. Pines 20 (SDA) y 21 (SCL). Permite comunicación I2C/TWI.

2.1.4 Comunicación

La placa Arduino Mega cuenta con diversas formas para comunicarse con un ordenador o con otros dispositivos. El ATmega2560 cuenta con cuatro UART de hardware para comunicación TTL a 5V, una conexión SPI y una conexión I2C. Mediante uno de los puertos UART el ATmega2560 se comunica con un ATmega16U2 que canaliza la conexión mediante USB a un ordenador y permite la comunicación entre ambos. Esto permite tanto programar la placa mediante el IDE de Arduino como recibir mediante monitor serie datos simples.

2.2 Ethernet Shield de Arduino

El Ethernet Shield de Arduino se trata de una placa que añade la funcionalidad de conectar una placa de Arduino a una red mediante conexión Ethernet. Se trata de una placa basada en el chip Wiznet W5100 que provee de una pila de red IP capaz de soportar protocolos TCP y UDP. Usa la librería Ethernet para leer y escribir flujos de datos.



Figura 2.2 Arduino Ethernet Shield.

La placa cuenta con varios LED que proporcionan información:

- ON. Indica que la placa está alimentada.
- LINK. Indica presencia de enlace de red.
- 100M. Indica la existencia de conexión de red de 100 Mb/s.
- RX. Indica que se reciben datos cuando parpadea.
- TX. Indica que se transmiten datos cuando parpadea.

Unos puntos importantes del Ethernet Shield son:

- Funciona a 5V.
- Tiene un microcontrolador W5100 con 16k de buffer y es independiente de la memoria del ATmega2560.
- Se comunica con el ATmega2560 mediante SPI.
- Soporta 4 conexiones simultáneas.
- Utiliza la librería Ethernet.
- Dispone de lector de tarjetas microSD para guardar ficheros.
- Utiliza los pines 10, 11, 12 y 13 para comunicarse con el W5100 mediante SPI.

2.3 Motor DC y Driver L298N

El principal actuador de este proyecto es un motor de corriente continua que es el que mueve la cinta transportadora, permitiéndola avanzar, retroceder o pararse. Este motor funciona a 24V y tiene infinidad de uso, ya que son sencillos de operar y tienen un bajo coste.

Para controlar dicho motor se utiliza un Driver L298N, que está basada en un puente H. Dicho módulo es capaz de controlar dos motores de corriente continua o un motor paso a paso bipolar de hasta 2A.

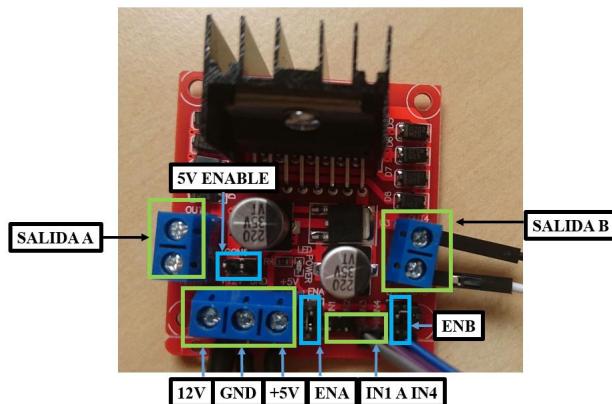


Figura 2.3 Driver L298N.

La principal funcionalidad del módulo es separar la parte asociada a la potencia (que funciona a 24V en este caso) de la parte asociada al control (que funciona a 5V).

2.3.1 Conexión del módulo L298N

El módulo cuenta con un circuito integrado LM7805 que proporciona 5V que puede ser activado o desactivado mediante un jumper.

Mientras el jumper esté activo, la placa admite tensiones de alimentación de entre 6 y 12V. Así, la conexión de 5V será una salida.

Por otra parte, si el jumper está desactivado la placa admite tensiones de alimentación de entre 12 y 35V. En este caso se deberá proporcionar 5V de referencia a la placa para el funcionamiento de la parte lógica.

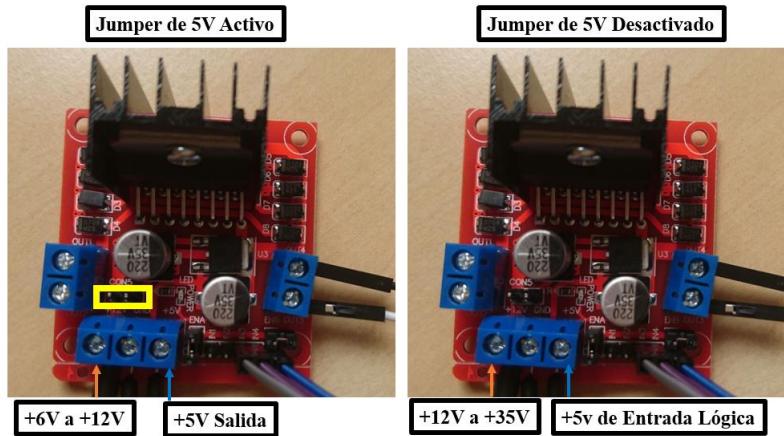


Figura 2.4 Tipos de conexiones del Driver L298N.

2.4 Encoder Rotativo Incremental

Para posicionar correctamente las piezas a lo largo de la cinta es necesario contabilizar el movimiento de la misma. Para ello, se cuenta con un encoder rotativo incremental de serie LPD3806-600BM el cual cuenta con una gran precisión.

Éste tiene dos salidas de onda cuadrada con un desfase de 90 grados entre ellas. Siempre que se produzca un flanco en A, será leída la señal B. Si B se encuentra en HIGH, el encoder se encontrará girando en sentido horario. Por el contrario, si B se encuentra en LOW, el encoder se encontrará girando en sentido antihorario.

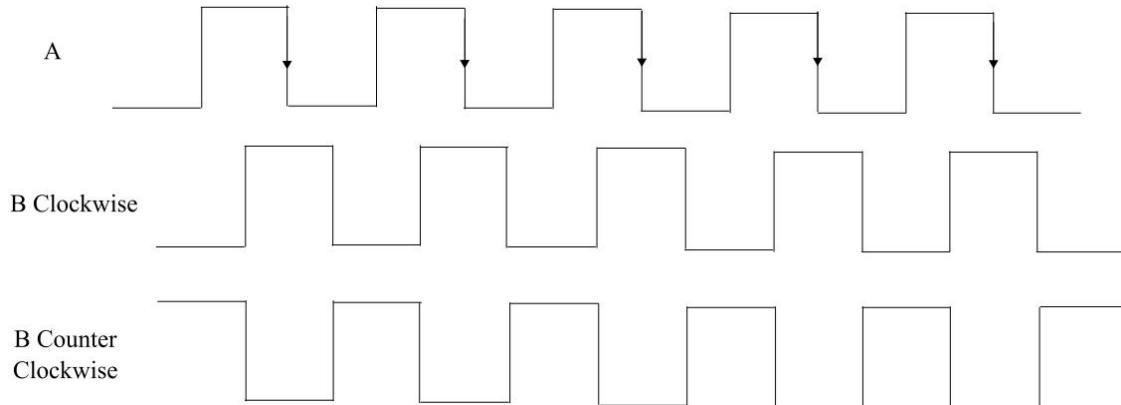


Figura 2.5 Señales encoder rotativo.

2.4.1 Especificaciones técnicas

El encoder rotativo de serie LPD3806-600BM cuenta con las siguientes especificaciones:

- 600 pulsos/revolución por cada fase. Por lo tanto, con las fases combinadas se cuenta con 2400 pulsos/revolución.
- Velocidad máxima: 5000 revoluciones/minuto.
- Respuesta de frecuencia: 0-30KHz

2.4.2 Conexionado del Encoder

El encoder cuenta con cuatro cuables de conexión:

- Rojo. Alimentación 5-24V.
- Negro. GND.
- Verde. Fase A.
- Blanco. Fase B.

2.5 Calibre digital

Para tomar la medida de la posición a lo ancho de la cinta se utiliza un calibre digital. Este instrumento se encuentra integrado en las cintas transportadoras del laboratorio. Cuentan con un LCD en el que se muestra la información de la medición y, además, cuenta con una tapa desmontable a la que se acceden cuatro pines que permiten comunicaciones con el Arduino. Estos pines son: alimentación a 1.5V, GND, señal de reloj y señal de datos. Todas las señales utilizan un nivel lógico de tensión de 1.5V, por lo que deben ser amplificadas a 5V para que el Arduino pueda comprender dichas señales.



Figura 2.6 Calibre digital y cables usados.

Mientras el calibre cuente con alimentación estará enviando tramas de datos con la medida de cada momento, por lo que es recomendable no utilizar pilas ya que siempre está funcionando. Algunas características importantes son:

- Cada trama cuenta con 24 bits, de los cuales 21 corresponden a la medición, uno para el signo, uno para la unidad (mm o in) y otro bit de acarreo.
- Los datos se transmiten por una señal de reloj (CLK) y una señal de datos (DATA).
- La línea de datos debe leerse en cada flanco de bajada de la señal de reloj.
- Los bits de la trama se inician por el menos significativo (LSB)
- El valor de la medida en mm debe ser multiplicado por 100.

2.6 LCD

La visualización de información e interacción del sistema utiliza un LCD 16x2. Este LCD cuenta con un módulo basado en microcontrolador que permite controlar la información mostrada mediante comunicación I2C, lo que permite reducir en gran cantidad los pines utilizados.

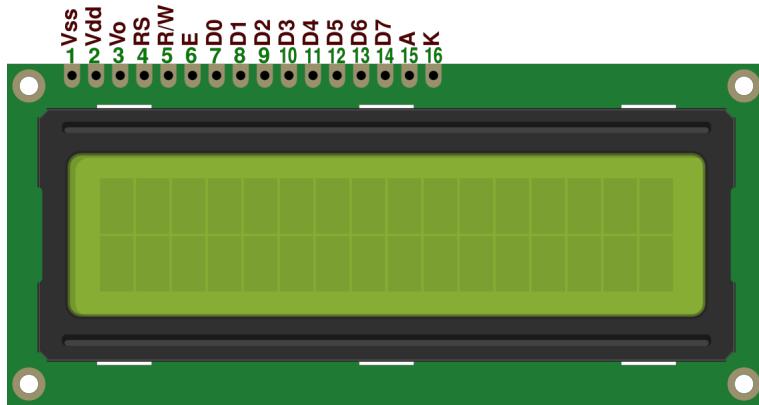


Figura 2.7 LCD 16x2.

La conexión se realiza mediante cuatro pines: alimentación 5V, GND, SCL y SDA. Además, la placa de adaptación cuenta con un potenciómetro que permite regular el brillo del LCD.



Figura 2.8 Módulo de control de LCD por I2C.

2.7 Sensor fotoeléctrico OMRON

Se utiliza un sensor fotoeléctrico para detectar que la pieza ha llegado a cierta posición arbitraria. Para ello se utiliza el sensor E3JK-R4M de la marca OMRON. Este sensor se encuentra incorporado en la cinta transportadora del laboratorio. Tiene dimensiones reducidas y tiene una alta capacidad de conmutación. El sensor es de tipo retroreflectivo polarizado, lo que permite detectar cuerpos brillantes. Además, posee un LED de color rojo que se enciende al detectar un objeto.

2.7.1 Conexiones del sensor fotoeléctrico

El sensor fotoeléctrico cuenta con cinco cables de conexión de diferentes colores:

- Marrón. Alimentación entre 12-24V.



Figura 2.9 Sensor fotoeléctrico OMRON E3JK-R4M.

- Azul. GND.
- Blanco. Salida común del relé.
- Negro. Salida de relé NA (Normalmente Abierto).
- Gris. Salida de relé NC (Normalmente Cerrado).

2.8 Cinta transportadora con elementos integrados

En el laboratorio de automática se encuentra una cinta transportadora con todos los elementos integrados, de modo que el proyecto consiste en definitiva en una interfaz entre estos elementos y el robot que le acompaña.

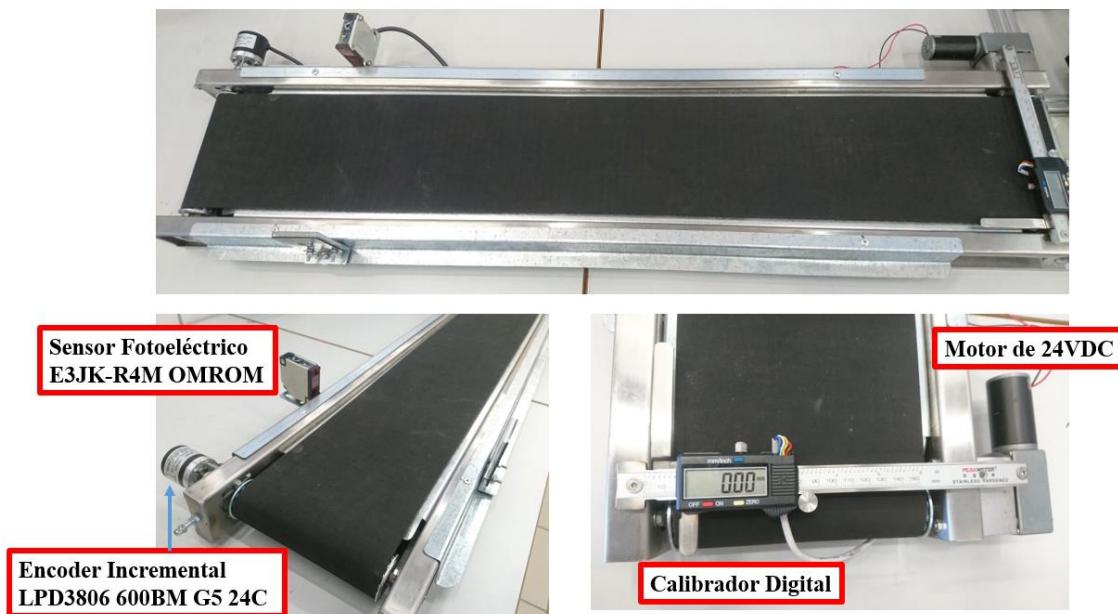


Figura 2.10 Cinta transportadora con elementos integrados.

2.9 Convertidor CC-CC con salida USB

Se trata de un pequeño dispositivo que se puede alimentar con un amplio rango de tensiones. En este caso, se puede alimentar con los 24V que proporciona la controladora del robot. Cuenta con cuatro salidas USB (que funcionan a 5V) con las que se puede alimentar la placa de Arduino y toda la electrónica del sistema.

2.10 Controladora IRC5C

El robot ABB IRB120 presente en los laboratorios del departamento cuentan con unas controladoras IRC5C, que son los dispositivos con los que se tiene que interactuar para comunicar la cinta transportadora con el robot. Estos dispositivos funcionan a 24V y cuentan con su propia fuente de alimentación. Por otra parte, cuentan con todas las conexiones necesarias para la realización del proyecto. Entre ellas, una conexión Ethernet a la que conectar el Ethernet Shield de Arduino.

Por otro lado, cuenta con una placa DSQC 652, que permite la conexión de señales digitales. Estas señales tienen un nivel lógico de 24V, necesitando un mínimo de 15V para detectar un '1' y un máximo de 5V para detectar un '0', por lo que las señales que les sean enviadas por esta vía deben ser adaptadas a dichas tensiones.

2.11 Optoacoplador TLP621-4

El circuito integrado TLP621-4 se trata de un chip que cuenta con cuatro aisladores ópticamente acoplados que constan con un diodo emisor de luz infrarroja y un fototransistor de silicio NPN encapsulado en plástico.

Las principales características son:

- Tensión de aislamiento en AC de 5300Vrms.
- Rango de temperatura de funcionamiento: -30°C a 100°C.
- Sin plomo.
- Paquete DIP de 16 pines.

Como dato relevante, la corriente máxima que debe pasar sobre los diodos es de 50 mA, por lo que las resistencias a colocar deberán condicionar este parámetro.

3 Desarrollo de placa de conexiones

Para simplificar todas las conexiones internas dentro de la caja y posibilitar ciertas funciones específicas, se crea una placa electrónica donde se conectan todos los dispositivos. De este modo se consigue simplificar el montaje y la posible actualización del sistema, ya que todas las conexiones internas desaparecen por completo. Además, para favorecer el orden de los cables dentro de la caja se crea una segunda placa más básica cuya única función es agrupar todas las conexiones existentes en la tapa. Se conocerá como placa *Fondo* a la placa general y placa *Tapa* a la superior.

3.1 Software utilizado. KiCAD.

Para la realización de la placa se ha utilizado el paquete de software KiCAD. Se trata de un software libre bajo licencia GNU General Public License, lo que permite su uso sin coste para el proyecto. KiCAD es un paquete de software orientado hacia el diseño electrónico (EDA por sus siglas en inglés: Electronic Design Automation) y consta de diversas aplicaciones que permiten realizar todo el trabajo.



Figura 3.1 Logo de KiCAD.

Por un lado cuenta con *eeschema*, un editor de esquemas electrónicos donde se puede plantear la lógica de las conexiones de un modo abstracto. Por otro, se encuentra con *pcbnew*, un editor de circuitos impresos. A partir de un esquemático creado se pasa a un circuito impreso de modo fácil y siendo modificable siempre que sea necesario.

3.2 Señales

En primer lugar, la lista de señales presentes en el proyecto se describen en la tabla 3.1. Estas señales se conectan a sus respectivos dispositivos por conectores atornillados en la PCB definitiva y, posteriormente se realizan las transformaciones e interconexiones internas que corresponden.

Tabla 3.1 Listado de señales.

Señal	Observaciones	Nivel lógico de tensión (V)
24V	Alimentación de 24V	24
5V	Alimentación de 5V	5
1.5V	Alimentación de 1.5V	1.5
GND	Línea de masa	0
EMER	Señal de emergencia	5 y 24
LR	Estado local o remoto	5 y 24
MICRO	Estado con o sin microcontrolador	5 y 24
FOTO	Sensor fotoeléctrico	5 y 24
UP	Flecha hacia arriba	5
DOWN	Flecha hacia abajo	5
ENTER	Avance en los menús	5
ESC	Retroceso en los menús	5
AVANCE	Mov. del motor sin microcontrolador (+)	24
RETROCESO	Mov. del motor sin microcontrolador (-)	24
IN3	Sentido de giro L298N	5
IN4	Sentido de giro L298N	5
ENB	Velocidad de giro L298N	5
FASEA	Fase A del encoder	5
FASEB	Fase B del encoder	5
CLK	Señal reloj del calibre	1.5 y 5
DATA	Señal de datos del calibre	1.5 y 5
SDA	Señal de datos del LCD	5
SCL	Señal de reloj del LCD	5

El proyecto cuenta con dispositivos que funcionan a distintos niveles de tensión, por lo que se tiene que plantear las transformaciones a realizar. Los dos principales niveles de tensión son los 5V a los que funciona el Arduino y los 24V a los que funciona el motor y la controladora del robot. Por otra parte, también se cuenta con el nivel lógico de 1.5V del calibre digital.

El sistema se alimenta con 24V provenientes de la controladora y pasa a 5V mediante el convertidor externo con el que se cuenta, por lo que con obtener dichas conexiones del propio sistema ya se dispone las dos líneas. Sin embargo, en el caso de los 1.5V éstos deben ser generados dentro de la placa como se verá en la sección correspondiente más adelante.

3.3 Señales digitales en 24V y en 5V

El sistema cuenta con cuatro salidas digitales que deben estar representadas tanto en 24V como en 5V para que tanto el Arduino como la controladora conozcan de modo inequívoco el modo en el que se encuentra el sistema. Estas señales son:

- Microcontrolador (MICRO). '1'si el microcontrolador se encuentra desactivado y '0'si está operativo.
- Local/Remoto (LR). '1'si el sistema funciona en modo remoto y '0'si funciona en modo local.
- Emergencia (EMER). '1'si el sistema requiere una parada de emergencia.
- Sensor Fotoeléctrico (FOTO). '1'si hay una pieza detectada por el sensor.

Estas señales son generadas mediante contactos a la señal de 24V. Las señales MICRO y LR funcionan con dos interruptores de dos posiciones, EMER mediante una seta de emergencia con conexión normalmente abierta y FOTO con el sensor fotoeléctrico a la salida normalmente abierta, lo que hace que en el caso de activarse cierre el contacto con el común, que se encuentra a 24V. A los tres primeros se les llamará *switches* de estado, que son los que determinan el estado del sistema.

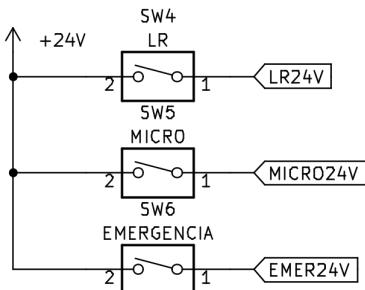


Figura 3.2 Switches de estado.

Además, se cuenta con dos entradas digitales que deben ser capaces de mover el motor en el caso de que el sistema se encuentre funcionando sin microcontrolador. Para ello estas señales deben actuar sobre los pines de dirección del L298N. Estas dos señales son:

- Avance (AVANCE). Actúa sobre el pin IN1 o IN3 del L298N.
- Retroceso (RETR). Actúa sobre el pin IN2 o IN4 del L298N.

Las señales se generan a 24V por defecto, por lo que es necesario su paso a 5V. Para ello se utilizan optoacopladores, unos dispositivos que mediante fotodiodos acoplados a fototransistores. En este proyecto se utiliza el circuito integrado TLP621-4 que cumple dicha función.

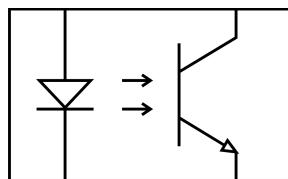


Figura 3.3 Ejemplo de optoacoplador.

3.3.1 Conversión de 24V a 5V

En la figura 3.4 se muestra el circuito empleado. Se utiliza un total de dos TLP621-4 para tener un total de 8 optoacopladores. Como la tensión máxima que reciben los diodos es de 24V, se colocan unas resistencias para evitar que se quemen. El valor de las resistencias utilizadas es de $8.2\text{ k}\Omega$, permitiendo que la corriente directa sea algo inferior a 3 mA. Como caso relevante, en el caso de la parada de emergencia, la resistencia será de $4.7\text{ k}\Omega$ para que tenga una corriente directa y, una mayor salida por ello para asegurar que se produzca la parada en caso de ser necesaria.

En el colector de los fototransistores se conectan directamente la línea de 5V para que, en caso de que la señal correspondiente se active, su equivalente de 5V también lo haga. En las señales FOTO, LR, MICRO y EMER se les añade un LED a la salida a modo de test para comprobar su funcionalidad, pero no se verán desde fuera de la caja.

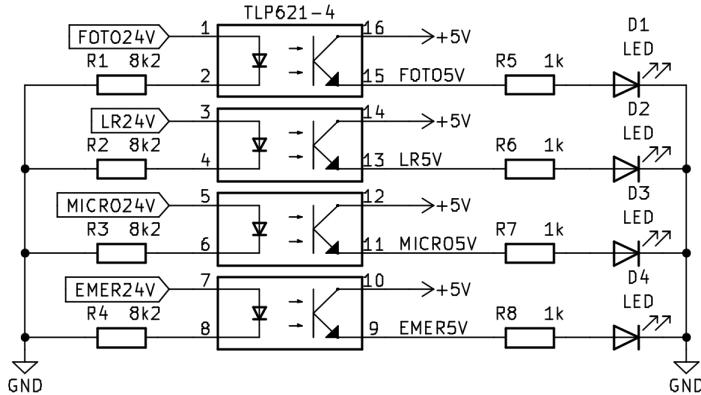


Figura 3.4 Implementación de optoacopladores.

3.3.2 Función sin microcontrolador y emergencia

El sistema debe funcionar tanto con como sin el microcontrolador, por lo que la parte en la que no funcione debe tener conexiones simples hacia la placa L298N. Para ello, cuando la señal MICRO sea un '1', se debe actuar sobre este dispositivo. Para evitar conflictos cuando el Arduino se encuentre conectado en este modo, los pines conectados a IN3, IN4 y ENB se configurarán como entrada, ya que como salida generarán conflictos.

En primer lugar, se actúa sobre el pin controlador de la velocidad de giro, ENB (o ENA). Para ello, mediante un búfer de tensión, se activa dicha señal al máximo cuando MICRO se encuentre en '1'. El búfer de tensión se utiliza para independizar la salida a ENB de la señal MICRO de 5V y permitir su uso con el Arduino en otros modos. La implementación se muestra en la figura 3.5. Para dicho búfer de tensión se emplea un canal del circuito integrado TL082.

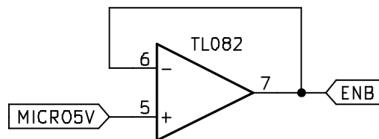


Figura 3.5 Búfer de tensión de la señal MICRO a 5V.

Para dar señal de movimiento se utilizan las flechas presentes en la caja. Estas flechas de selección tienen dos salidas independientes, por lo que las señales lógicas que recibe el Arduino pueden ir por un canal mientras que la señal de movimiento en modo sin microcontrolador por otro. De este modo, si las señales de movimiento se utilizan a 24V se puede conectar directamente al panel de señales digitales remoto, el cual interactúa directamente con la controladora del robot. Por ello, el terminal común que conecta la flecha de selección para el modo sin microcontrolador va conectado a la señal MICRO de 24V como se ve en la figura 3.6.

Por último, para que se produzca el movimiento además se debe actuar sobre IN3 o IN4 para que el motor se mueva hacia un lado o hacia el otro. Para ello, se vuelven a utilizar optoacopladores como se ve en la figura 3.7. La señal de AVANCE y la señal de RETROCESO actúa sobre el optoacoplador que le corresponde y pone el pin que le corresponde a '1', haciendo que el motor se mueva en el sentido deseado. Como la flecha solo activa uno de los dos sentidos cada vez, no se producirá el bloqueo del motor.

Sin embargo, interesa que se produzca dicho bloqueo en un determinado caso: la parada de emergencia. Si tanto IN3 como IN4 se ponen en alto a 1a la vez, el motor queda totalmente

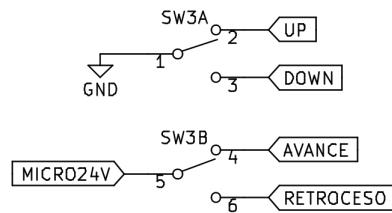


Figura 3.6 Conexiones flecha selección.

bloqueado y deja de moverse. Por ello, se conecta la señal EMER de 24V a dos optoacopladores independientes que, en caso de que sea necesario, produzca dicha parada. La misma se realizará tanto en modo sin microcontrolador como con éste activo, ya que depende directamente de la señal EMER.

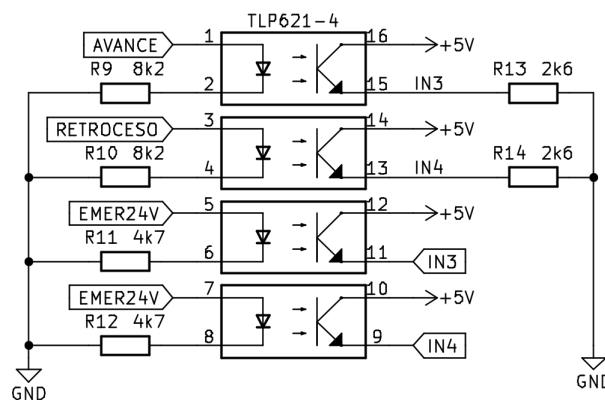


Figura 3.7 Implementación de optoacopladores para modo sin microcontrolador y emergencia.

3.4 Calibre digital

3.4.1 Alimentación calibre

Para el funcionamiento del calibre digital, éste debe ser alimentado por una fuente de 1.5V. Como lo que se dispone en este proyecto es de 5V y 24V lo más sencillo es colocar un divisor resistivo a la línea de 5V y utilizar un búfer de tensión para mantener dicho nivel lógico estable. Utilizando resistencias lo suficientemente altas el consumo es despreciable. Por ello se utiliza una resistencia de $39k\Omega$ y otra de $100k\Omega$. La tensión de salida obtenida será de:

$$V_{out} = 5V \frac{39k\Omega}{39k\Omega + 100k\Omega} \approx 1.4V \quad (3.1)$$

Los 1.4V que se obtienen entran dentro del rango de funcionamiento del calibre utilizando resistencias comerciales, por lo que el resultado es válido.

El consumo del divisor resistivo por otro lado será de:

$$I_{divisor} = \frac{5V}{39k\Omega + 100k\Omega} \approx 0.036mA \quad (3.2)$$

Se puede considerar un consumo despreciable.

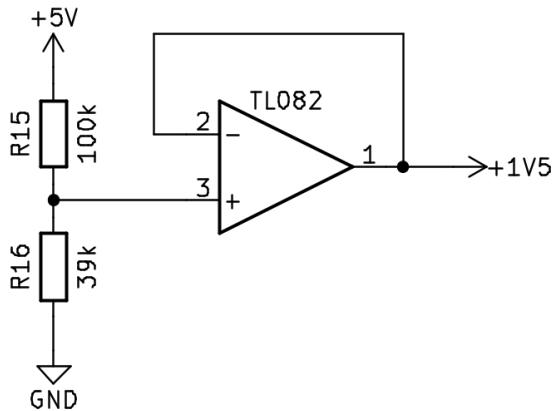


Figura 3.8 Alimentación calibre digital a 1.5V.

Para el búfer de tensión se utiliza un circuito integrado TL082, del cual se utiliza uno de los dos amplificadores operacionales con el que cuenta. Como resultado se obtiene el circuito de la figura 3.8.

3.4.2 Amplificación señales calibre

En la lectura de la señal del calibre por parte del Arduino es necesario elevar los niveles de tensión a 5V. Para ello se toma como referencia el circuito visto en la web [6]. Éste utiliza un transistor junto con dos resistencias de modo que cuando el calibre envíe un '1'eleve la tensión a 5V, pero sin distorsionarse cuando haya un '0'y para no dar falsos positivos. El esquema implementado es el que aparece en la figura 3.9.

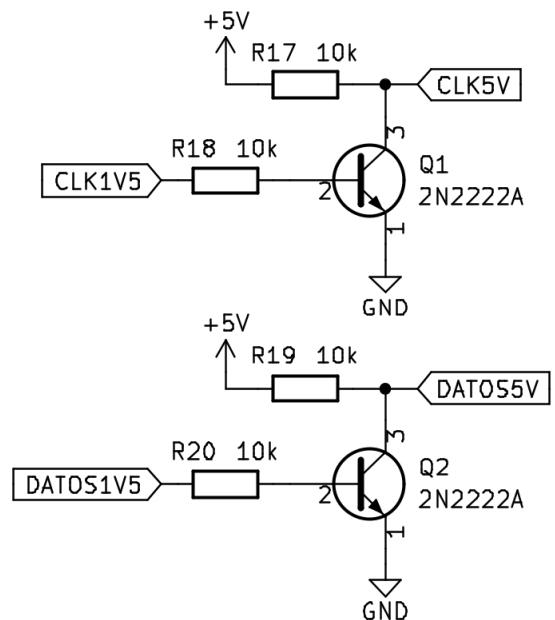


Figura 3.9 Amplificación señales calibre digital a 5V.

3.5 Placas finales

Una vez descrita la funcionalidad del sistema electrónico a implementar, el resto del proceso consiste en separar las conexiones entre las placas de circuito impreso y determinar las conexiones correspondientes a cada una de ellas.

3.5.1 Placa Tapa

La placa *Tapa* como su propio nombre indica irá colocada en la superficie superior de la caja. Su principal función es permitir que la caja se pueda abrir sin que haya peligro de desconexiones ni tirones entre los cables que conectan los distintos dispositivos que se encuentren en la tapa. Para ello, habrá un mazo de cables que se conecte a al fondo de la caja directamente y así simplificarlo todo.

Por otro lado, los dispositivos conectados a esta tapa están destinados a ser la interfaz de interacción con el usuario. Estos dispositivos son:

- LCD.
- Seta de emergencia.
- Botón intro y escape.
- Flecha de selección.
- Interruptores de estado (LR y MICRO)

El resultado final se puede observar en la figura 3.10. En ésta se puede observar los huecos para soldar los futuros conectores atornillables. Cuenta con dos zonas diferenciadas como se explicó previamente: una para la conexión de los dispositivos y otra para la conexión a la placa del fondo.

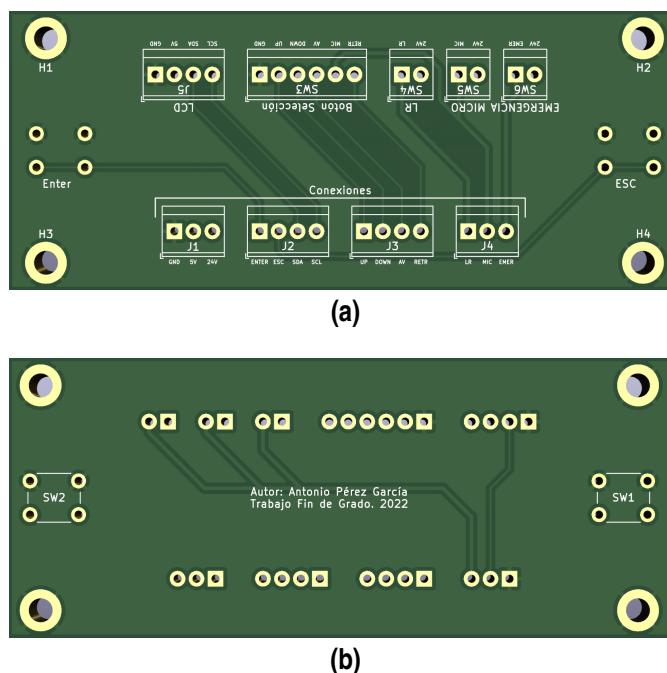


Figura 3.10 (a) Vista frontal de la placa tapa. (b) Vista trasera de la placa tapa.

Como se puede observar, originalmente hay hueco para que los botones sean soldados en la propia placa ya que estaba pensado otra distribución, pero que posteriormente fueron sustituidos,

por lo que en dichos huecos se sustituye por conectores atornillables.

3.5.2 Placa Fondo

La placa *Fondo* va situada en la base de la caja junto con el resto de dispositivos. Ésta se encarga de realizar las funciones indicadas en los puntos anteriores de este capítulo, además de ser el centro neurálgico del proyecto donde se producen todas las conexiones. En la figura 3.11 se ve el resultado final.

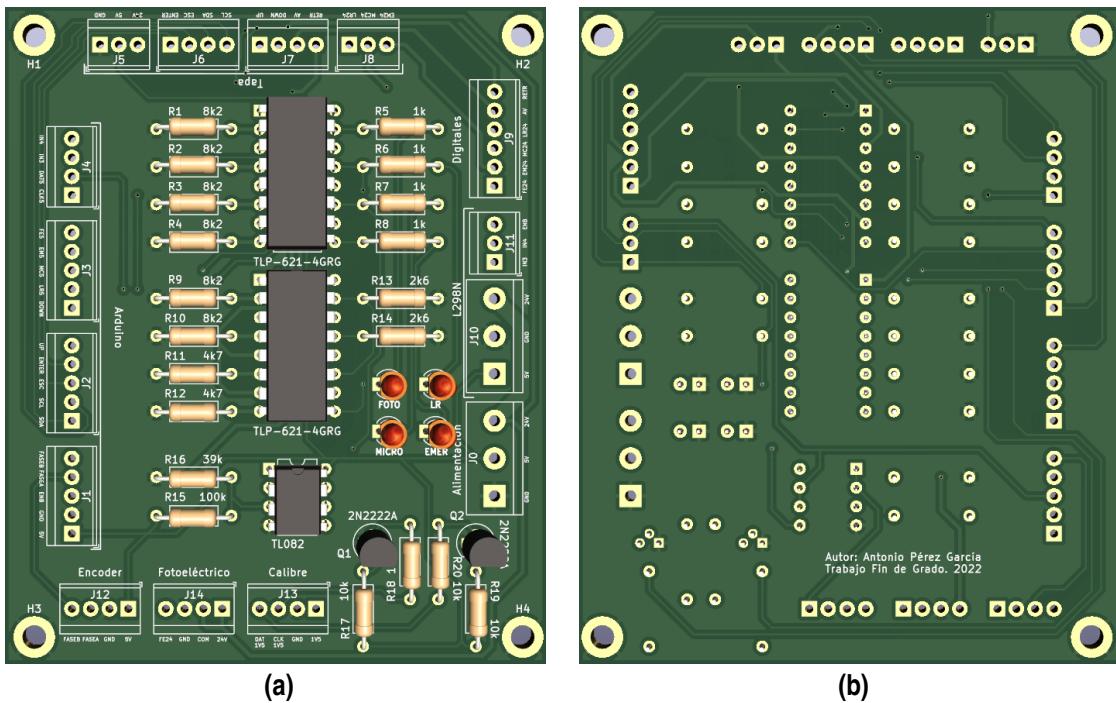


Figura 3.11 (a) Vista frontal de la placa fondo. (b) Vista trasera de la placa fondo.

En la figura se puede diferenciar varias zonas:

- Conexiones a la tapa en la zona superior.
- Conexiones al Arduino en la zona izquierda.
- Alimentación, conexiones a L298N y pines digitales en la zona derecha.
- Conexiones a los dispositivos (encoder, calibre y sensor fotoeléctrico) en la zona inferior.
- Zona central. Dedicada a realizar todas las interconexiones y las funciones que se han indicado durante el capítulo.

4 Planificación de caja

Para que todos los dispositivos que componen el sistema estén bien aislados del exterior y no haya problemas con las conexiones se planifica una caja estanca. Ésta deberá ser accesible para el reemplazo de dispositivos defectuosos y ser replicable.

Como referencia se toma de una caja de dimensiones 220 x 145 x 80 mm. Para reducir los costes, se decide imprimirla en 3D, ya que permite mayor adaptabilidad para colocar los lugares donde fijar los diferentes dispositivos. La caja estará dividida en una base y una tapa que se unen mediante tornillos y permiten una unión estable.

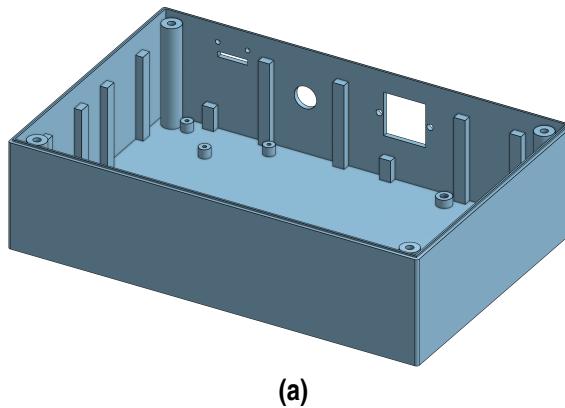
4.1 Base

La base debe tener las siguientes características:

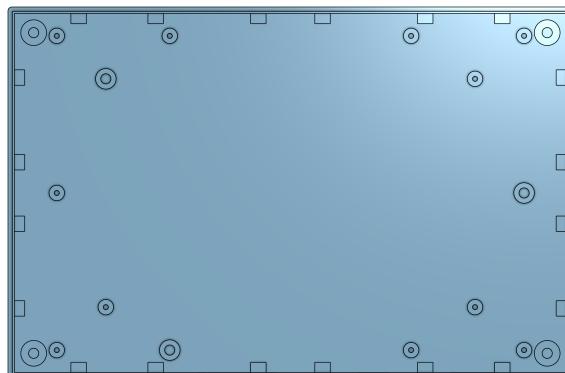
- Permitir fijar y extraer con facilidad todos los dispositivos.
- Orificio para entrada de cables de conexión.
- Hueco para conector hembra Ethernet.
- Hueco para conector de pines digitales.

Con todo ello se ha diseñado la base que se puede ver en la figura 4.1. Se muestra una imagen de la planta y otra general. Se puede observar que cuenta con distintos orificios en los que se debe introducir tuercas M3 durante la impresión en 3D para los distintos anclajes de dispositivos que se comenta a continuación y para la unión con la tapa para que quede estanca la caja.

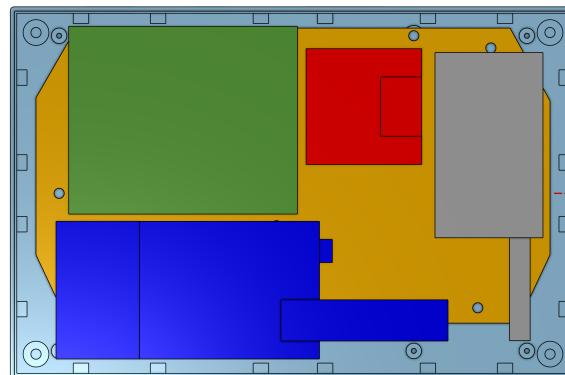
Además, para el anclaje de los dispositivos sobre la base, se ha diseñado un soporte que va atornillado a la misma. Para ello, se ha modelado de forma simplificada todos los dispositivos y se ha buscado la forma óptima de disponerlos. También se ha tenido en cuenta los cables que van conectados tanto a la fuente como al Arduino, modelándolos en el espacio que ocupan hasta que el ángulo de giro es aceptable para que el cable se pueda colocar de forma holgada y sin tensiones dentro de la caja. Posteriormente se ha creado el soporte en base a dicha disposición y contando con los orificios de cada dispositivo. En la figura 4.2 se puede observar el resultado final de este ensamblaje.



(a)

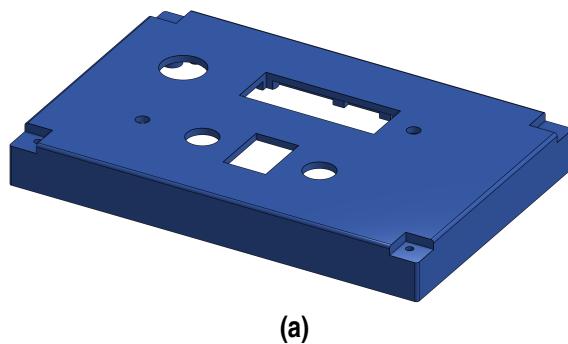


(b)

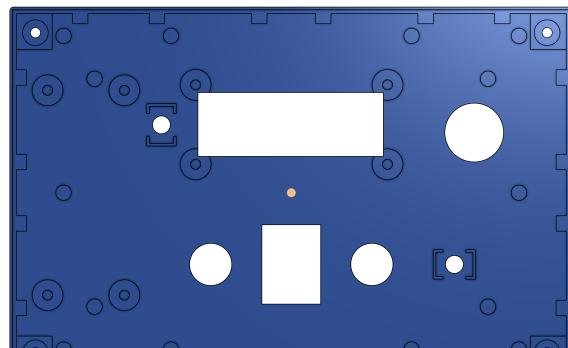
Figura 4.1 (a) Vista general de la base de la caja. (b) Vista de planta de la base de la caja.**Figura 4.2** Ensamblaje de la base con el soporte. Descripción por colores: a) Gris: Fuente. b) Rojo: L298N. c) Azul: Arduino. d) Verde: placa de conexiones. e) Amarillo: Soporte tapa.

4.2 Tapa

La tapa debe ser el soporte para la interfaz humano-máquina y, por ello, sostener todos sus dispositivos. Por lo general, todos los dispositivos que van en la tapa son de tamaño reducido, a excepción de la seta de emergencia, por lo que no hay problemas con su distribución. La seta de emergencia se ubicará en la esquina superior izquierda, situándose justo encima de la placa de conexiones del fondo, ya que ésta es la de menor altura. En la figura 4.3 se muestra la tapa tanto en la planta como en una vista general.



(a)



(b)

Figura 4.3 a) Vista general de la tapa de la caja. b) Vista inferior de la tapa de la caja.

Además, se muestra en la figura 4.4 se muestra la distribución de los dispositivos. Ésta distribución permite cumplir con la planificación inicial que se observó en la figura 1.1 y facilitar la conexión entre las placas electrónicas al colocar la línea de interconexión de forma paralela.

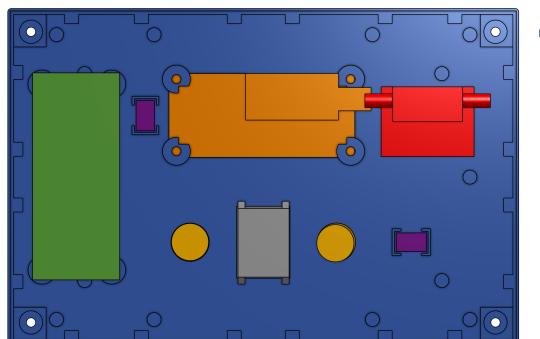


Figura 4.4 Vista inferior de la tapa ensamblada. Descripción por colores: a) Verde: placa de conexiones. b) Naranja: LCD c) Rojo: Seta de emergencia. d) Gris: Flechas de selección. e) Amarillo: Botones intro y escape. f) Morado: Selectores de modo.

5 Desarrollo en Arduino

Una parte imprescindible del proyecto es la programación del propio Arduino, que es el que centraliza todo el control del sistema. La parte relacionada con el control del motor y la programación del encoder está bien documentada en el trabajo [4], mientras que para la medición del calibre se ha partido de la web [6] al igual que en el diseño electrónico ya comentado en el capítulo 3.

5.1 Diagrama de flujo

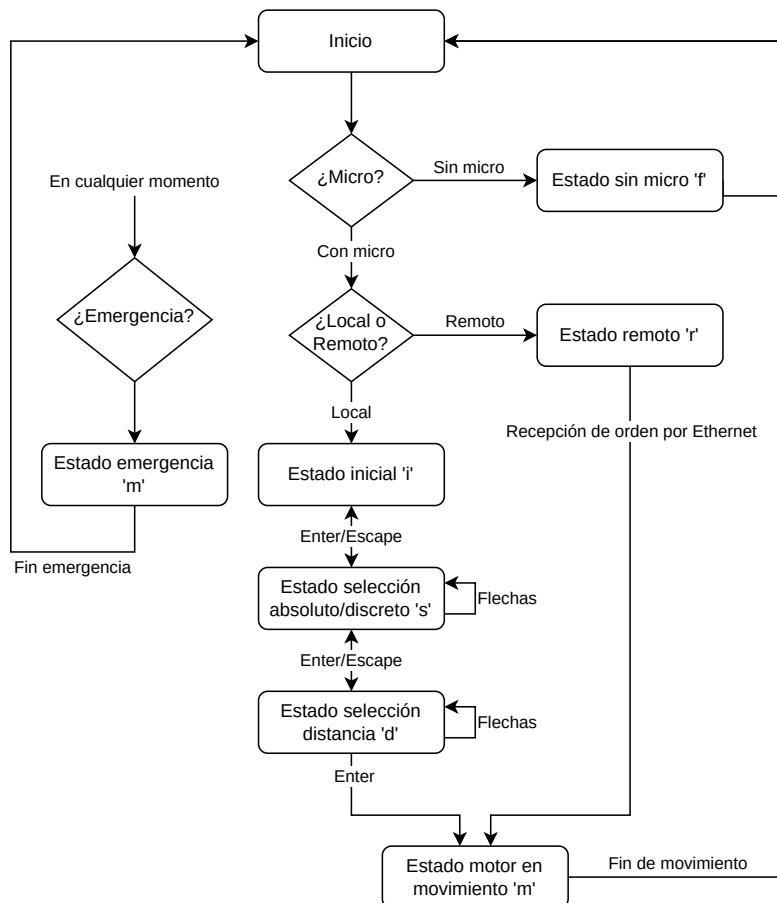


Figura 5.1 Diagrama de flujo del programa.

En el diagrama de la figura 5.1 se observa los distintos estados por los que pasa el programa. En primer lugar, se realiza una comprobación de los estados de microcontrolador y local para configurar los pines como corresponde. En caso de que no haya microcontrolador, se configurarán los pines IN3, IN4 y ENB como entradas y el dispositivo esperará a que se salga de dicho modo, manteniéndose en estado "f". Si se configura como remoto, el Arduino entrará en estado "r" (remoto) y comprobará continuamente si se recibe una orden por parte del controlador del ABB para realizar dicho movimiento.

El caso que tiene más estados intermedios es el modo local, ya que requiere varios menús dentro de la pantalla para la interacción con el usuario. En este modo de funcionamiento se avanza en el menú con el botón enter y se retrocede con el botón escape. Tras realizar las configuraciones del sistema, el sistema entrará en estado "i" (inicial), mostrado un mensaje de bienvenida en el LCD y esperando al botón enter para avanzar de menú. Una vez pulsado enter, se pasa al modo selección de avance absoluto o discreto, estado "s". En este estado, mediante las flechas de selección se cambia entre avance discreto y avance absoluto. Una vez avanzado al siguiente menú, entramos en el modo selección de distancia de movimiento "d". En este modo, mediante las flechas se irá incrementando o disminuyendo la posición final deseada. Por último, se tomará toda la información recibida por el usuario para pasar al estado de motor en movimiento "m".

Independientemente si se accede al estado "m" mediante modo local o remoto, la cinta se moverá hasta la posición deseada mediante el mismo algoritmo. Primero se aproxima la pieza a la posición final avanzando a velocidad constante para posicionar finalmente la pieza mediante un PID. Una vez terminada la operación, vuelve al modo inicial para seguir esperando órdenes.

La seta de emergencia puede ser pulsada en cualquier instante en todos los modos de funcionamiento, la cinta parará automáticamente y se entrará en estado emergencia "e". Se trata de un sistema de seguridad independiente y una vez activado este modo no se podrá realizar ninguna acción hasta que se desarme la seta de emergencia.

5.2 Programa final

A continuación se muestra el programa final. Este se divide en dos archivos, "main.h" y "main.ino". El programa puede ser cargado a la placa mediante el Arduino IDE que se puede descargar desde la página oficial [2]. Además, en la propia página se puede acceder a la documentación referente a las funciones utilizadas en el programa.

Por otro lado, dos librerías han sido utilizadas para el funcionamiento del LCD y de la conexión por Ethernet. Para el caso de la librería para el LCD se ha utilizado la librería [3], ya que la conexión se realiza mediante protocolo I2C. Este protocolo permite que un segundo microcontrolador integrado dentro de la placa del LCD sea el que lo controle, evitando tener que realizar todas las conexiones que requiere el mismo directamente hacia el Arduino y delegando el control al mismo. Para la conexión mediante Ethernet se ha utilizado la librería [7]. Este caso es similar al anterior, en el cual mediante el módulo Arduino Ethernet Shield acoplado al Arduino Mega se realiza una comunicación SPI con el microcontrolador W5100 del módulo, que es el que realiza todas las operaciones necesarias para permitir la conexión.

El archivo "main.h", mostrado a continuación en el código 5.1 es el encargado de definir todas las variables globales y definir macros para los pines. Este archivo se separa del principal para tener un acceso más rápido a las variables por si se necesita realizar un ajuste y que el código principal sea más fácil de leer. En éste se incluyen los parámetros del PID, la inicialización del estado, la configuración inicial de la posición y el desplazamiento, además de ciertas variables auxiliares que se utilizan a lo largo de todo el programa.

Código 5.1 Declaración de variables, "main.h".

```
// Definición de pines
#define faseA 2
#define faseB 3

#define BUTTON_ESC 22
#define BUTTON_ENTER 23
#define BUTTON_UP 24
#define BUTTON_DOWN 25

#define BUTTON_LOCAL 28
#define BUTTON_MICRO 29
#define BUTTON_EMERGENCIA 30

#define SENSOR_FOTO 31

#define CAL_CLK 32
#define CAL_DATA 33

#define ENB 5
#define IN3 40
#define IN4 41

// Declaración de objetos de LCD y relacionados con Ethernet
LiquidCrystal_I2C lcd(0x27,16,2);
byte MAC[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // Dirección MAC
del dispositivo
IPAddress IP(192,168,1,200); // IP estática del dispositivo
EthernetServer servidor(4012); // Puerto donde se transmite la informaci
ón

// Declaración de variables booleanas auxiliares
bool enter;
bool esc;
bool up;
bool down;
bool esc_ant = 0;
bool enter_ant = 0;
bool up_ant = 0;
bool down_ant = 0;

bool emergencia;
bool local;
bool micro;
bool fotoelectrico;

bool emergencia_ant;
bool local_ant;
bool micro_ant;
```

```

// Declaración e inicialización de variable de posición 'x' e 'y'
long posicion = 0;
long * pposicion = &posicion;
float posy = 0;

// Inicialización de variables de estado y desplazamiento
char estado = 'i';
bool discreto = 0;
long desplazamiento = 1000;
long objetivo = 0;

// Definición de parámetros de PID
float kp = 0.5;
float Ti = 2;
float Td = 0.1;
float T = 0.01;

```

Por último, se incluye el código 5.2 que es el código el archivo "main.ino". Este archivo contiene el programa en sí con todas sus funciones y protocolos. Éste cuenta con comentarios relevantes que explican el código que se muestra.

Código 5.2 Código principal Arduino, "main.ino".

```

// Inclusión de librerías
#include <Ethernet.h>
#include <LiquidCrystal_I2C.h>
#include "main.h"

// Definición de funciones:

// Funciones de interrupción de encoder
void cambiofaseA(void);
void cambiofaseB(void);

// Funciones relacionadas al movimiento del motor
void pararMotor(void);
void moverMotor(void);
void movimientoMotor(long objetivo, long* posicion, LiquidCrystal_I2C
    lcd);

// Función de lectura del calibre digital
float medidaCalibre(void);

// Funciones de conexión por Ethernet
bool comprobarRobot(void);
bool enviarRobot(char dato);
void ethconex(void);

// Función setup de configuración del sistema

```

```
void setup() {
    // Inicializar el LCD
    lcd.init();

    // Encender la luz de fondo.
    lcd.backlight();

    // Definición de pines
    pinMode(faseA, INPUT_PULLUP);
    pinMode(faseB, INPUT_PULLUP);

    pinMode(BUTTON_DOWN, INPUT);
    digitalWrite(BUTTON_DOWN, HIGH);
    pinMode(BUTTON_UP, INPUT);
    digitalWrite(BUTTON_UP, HIGH);
    pinMode(BUTTON_ENTER, INPUT);
    digitalWrite(BUTTON_ENTER, HIGH);
    pinMode(BUTTON_ESC, INPUT);
    digitalWrite(BUTTON_ESC, HIGH);

    pinMode(BUTTON_EMERGENCIA, INPUT);
    pinMode(BUTTON_LOCAL, INPUT);
    pinMode(BUTTON_MICRO, INPUT);

    pinMode(SENSOR_FOTO, INPUT);

    // En función del estado de las señales MICRO, EMER y LR la
    // configuración de IN3, IN4 y ENB pasa a INPUT u OUTPUT para evitar
    // conflictos
    if(!digitalRead(BUTTON_MICRO)){
        pinMode(IN3, OUTPUT);
        pinMode(IN4, OUTPUT);
        pinMode(ENB, OUTPUT);
        if(!digitalRead(BUTTON_EMERGENCIA)){
            estado = 'i';
        } else{
            estado = 'e';
        }
        if(digitalRead(BUTTON_LOCAL)){
            estado = 'r';
        }
    } else{
        pinMode(IN3, INPUT);
        pinMode(IN4, INPUT);
        pinMode(ENB, INPUT);
        estado = 'f';
    }

    // Asociación de interrupción a los pines del encoder
    attachInterrupt(digitalPinToInterrupt(faseA), cambiofaseA, RISING);
```

```
attachInterrupt(digitalPinToInterrupt(faseB), cambiofaseB, RISING);

posy = medidaCalibre();
posición = 0;
// Se inicia la comunicación Ethernet y la serie para depuración
Ethernet.begin(MAC, IP);
Serial.begin(9600);
Serial.print("Servidor en IP ");
Serial.println(Ethernet.localIP());
}

// Bucle infinito
void loop() {

    // Se lee el estado de los botones
    up = digitalRead(BUTTON_UP);
    down = digitalRead(BUTTON_DOWN);
    enter = digitalRead(BUTTON_ENTER);
    esc = digitalRead(BUTTON_ESC);

    // Se lee el estado de la configuración
    emergencia = digitalRead(BUTTON_EMERGENCIA);
    local = digitalRead(BUTTON_LOCAL);
    micro = digitalRead(BUTTON_MICRO);

    // En el caso de cambio de las señales de modo o emergencia, pasar
    // por setup() para reconfigurar los pines
    if(micro && (estado != 'f') && (estado != 'e')){
        setup();
    }

    if(local != local_ant){
        setup();
    }

    if(emergencia && (estado != 'e')){
        estado = 'e';
        lcd.clear();
    }

    // Máquina de estados
    switch(estado){
        // Estado inicial
        case 'i':
            lcd.setCursor(0, 0);
            lcd.print("MODO LOCAL    ");
            lcd.setCursor(0, 1);
            lcd.print("PULSE ENTER   ");
            digitalWrite(IN3, LOW);
            digitalWrite(IN4, LOW);
    }
}
```

```
analogWrite(ENB, 255);
// Envío por Ethernet del estado al robot
if(comprobarRobot()){
    enviarRobot(estado);
}
break;

// Estado selección discreto/absoluto
case 's':
    lcd.setCursor(0, 0);
    lcd.print("SELECCIONE MOV. ");
    if(discreto){
        lcd.setCursor(0, 1);
        lcd.print("AVANCE DISCRETO ");
    } else{
        lcd.setCursor(0, 1);
        lcd.print("AVANCE ABSOLUTO ");
    }
    if(comprobarRobot()){
        enviarRobot(estado);
    }
break;

// Estado selección distancia de movimiento
case 'd':
    if(comprobarRobot()){
        enviarRobot(estado);
    }
break;

// Estado movimiento del motor
case 'm':
    if(comprobarRobot()){
        enviarRobot(estado);
    }
    movimientoMotor(objetivo, pposicion, lcd);
    if(comprobarRobot()){
        enviarRobot(estado);
    }
    posy = medidaCalibre();
    lcd.setCursor(0, 0);
    lcd.print("POSICION FINAL ");
    lcd.setCursor(0, 1);
    lcd.print("          ");
    lcd.setCursor(0, 1);
    lcd.print("X=");
    lcd.print(posicion/10);
    lcd.print(" Y=");
    lcd.print(posy);
    delay(2000);
```

```
estado = 'i';
if(digitalRead(BUTTON_LOCAL)){
    estado ='r';
}
break;

// Estado sin micro
case 'f':
    lcd.setCursor(0, 0);
    lcd.print("SIN MICRO");
    break;

// Estado emergencia
case 'e':
    lcd.setCursor(0, 0);
    lcd.print("EMERGENCIA");
    break;

// Estado remoto
case 'r':
    lcd.setCursor(0, 0);
    lcd.print("MODO REMOTO ");
    lcd.setCursor(0, 1);
    lcd.print(" ");
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
    analogWrite(ENB, 255);
    if(comprobarRobot()){
        enviarRobot(estado);
    }
    break;

// Transiciones entre estados
switch(estado){
    case 'i':
        if(!enter && enter_ant){
            estado = 's';
        }
        break;

    case 's':
        if(!enter && enter_ant){
            estado = 'd';

            posy = medidaCalibre();
            lcd.setCursor(0, 0);
            lcd.print(" ");
            lcd.setCursor(0, 1);
            lcd.print(" ");
        }
}
```

```
lcd.setCursor(0, 0);
lcd.print("X=");
lcd.print(posicion/10);
lcd.print(" Y=");
lcd.print(posy);

lcd.setCursor(0, 1);
if(desplazamiento > 0){
    lcd.print("+");
}
lcd.print(desplazamiento/10);
lcd.print(" mm");
}
if(!esc && esc_ant){
estado = 'i';
}
if((!up && up_ant) || (!down && down_ant)){
discreto = !discreto;
}
break;

case 'd':
if(!enter && enter_ant){
estado = 'm';
if(discreto){
    objetivo = posicion + desplazamiento;
} else{
    objetivo = desplazamiento;
}
}
if(!esc && esc_ant){
estado = 's';
}
if(!up && up_ant){
desplazamiento = desplazamiento + 1000;
lcd.setCursor(0, 1);
lcd.print("          ");
lcd.setCursor(0, 1);
if(desplazamiento > 0){
    lcd.print("+");
}
lcd.print(desplazamiento/10);
lcd.print(" mm");
}
if(!down && down_ant){
desplazamiento = desplazamiento - 1000;
lcd.setCursor(0, 1);
lcd.print("          ");
lcd.setCursor(0, 1);
if(desplazamiento > 0){
```

```
        lcd.print("+");
    }
    lcd.print(desplazamiento/10);
    lcd.print(" mm");
}
break;

case 'f':
if(!micro){
setup();
}
break;

case 'e':
if(!emergencia){
setup();
}
break;
}

// Guardado de estados
esc_ant = esc;
enter_ant = enter;
up_ant = up;
down_ant = down;

emergencia_ant = emergencia;
local_ant = local;
micro_ant = micro;

delay(10);
}

// Funciones definidas en la cabecera

// Si hay cambio en la fase A, se comprueba la fase B para definir el
// sentido de giro
void cambiocaseA(void){
    bool fB = digitalRead(faseB);
    if(fB){
        posicion++;
    } else{
        posicion--;
    }
}

// Si hay cambio en la fase A, se comprueba la fase B para definir el
// sentido de giro
void cambiocaseB(void){
    bool fA = digitalRead(faseA);
```

```
if(fA){  
    posicion--;  
} else{  
    posicion++;  
}  
}  
  
// Parado de motor  
void pararMotor(void){  
    digitalWrite(IN3, LOW);  
    digitalWrite(IN4, LOW);  
    analogWrite(ENB, 255);  
}  
  
// Avance de motor a la velocidad especificada  
void moverMotor(int u){  
    if(u >= 0){ // Avance positivo  
        digitalWrite(IN3, LOW);  
        digitalWrite(IN4, HIGH);  
    } else{ // Avance negativo  
        digitalWrite(IN3, HIGH);  
        digitalWrite(IN4, LOW);  
        u = -u;  
    }  
  
    if(u > 200){  
        u = 200; // La salida máxima es de 255 pero se limita a 200  
    }  
  
    analogWrite(ENB, u);  
}  
  
// Función que coloca el sistema en la posición deseada  
void movimientoMotor(long objetivo, long* posicion, LiquidCrystal_I2C  
lcd){  
    lcd.setCursor(0, 0);  
    lcd.print("MOVIENDO... ");  
    lcd.setCursor(0, 1);  
    lcd.print(" ");  
    lcd.setCursor(0, 1);  
    lcd.print((*posicion)/10);  
    lcd.print("/");  
    lcd.print(objetivo/10);  
  
    long u = 0;  
    long ek = 0;  
    long ek1 = 0;  
    long ik = 0;  
    long D = 0;  
    int aux = 0;
```

```
bool emer = 0;

// Bucle PID. Se mantiene buscando la posición mientras no entre
// dentro de un margen y la acción de control no se encuentre por
// debajo del umbral de funcionamiento
while(((*posicion < objetivo - 5) || (*posicion > objetivo + 5) || (
    u > 55)) && !emer){

    emer = digitalRead(BUTTON_EMERGENCIA);

    // Se actualiza la posición actual en el LCD cada 120 ms
    aux++;
    if(aux > 3){
        lcd.setCursor(0, 1);
        lcd.print("          ");
        lcd.setCursor(0, 1);
        lcd.print((*posicion)/10);
        lcd.print("/");
        lcd.print(objetivo/10);
        aux = 0;
    }

    //Actualización de error integral
    ek1 = ek;
    ek = objetivo - *posicion;
    ik = ik + ek;
    D = ek - ek1;
    //Cálculo de la señal de control
    u = kp * (ek + (T/Ti) * ik + (Td/T) * D);

    // En caso de saturación, reiniciar el error integral
    if((abs(u) > 255) && (abs(ek) > 200)){
        ik = 0;
    }

    // Limitar la acción de control al máximo que permite Arduino
    if(u > 255){
        u = 255;
    }
    if(u < -255){
        u = -255;
    }

    if(!emer){
        moverMotor(u);
        delay(1000 * T);
    }
}

// Parar el motor cuando se alcance la posición deseada
```

```
pararMotor();

if(emerg){
lcd.clear();
lcd.print("EMERGENCIA");
delay(1000*T);
}
}

// Medida del calibre
float medidaCalibre(void){
    bool data;
    float medida;
    int value = 0;
    int signo = 0;

    unsigned long tempmicros;
    unsigned long tempmicros2;

    // Reintentar la medición hasta que se pueda realizar
    for(int j=0; j<10 && (value == 0); j++){

        tempmicros = micros();
        while (digitalRead(CAL_CLK)==LOW) {
            delayMicroseconds(1);
        }

        tempmicros2 = micros();
        if ((tempmicros2-tempmicros)>10000) {
            // Se leen los 24 bits que proporciona el calibre
            for (int i=0; i<24; i++) {
                while (digitalRead(CAL_CLK)==HIGH) {
                    delayMicroseconds(1);
                }
            }

            data = !digitalRead(CAL_DATA);

            // Se leen los datos cada bajada del flanco de reloj
            if(i<16){
                value |= data << i;
            }else{
                signo |= (data << (i-16));
            }
        }

        while (digitalRead(CAL_CLK)==LOW) {
            delayMicroseconds(1);
        }
    }

    // El bit 0x80 corresponde a las unidades, pulgadas o milímetros
}
```

```
// En caso de que sean pulgadas, se realiza la conversión a milímetros
if(signo & 0x80){
    medida = 25.4*float(value)/(2*1000);
} else{
    medida = float(value)/100;
}

// El bit 0x10 corresponde al signo
if(signo & 0x10){
    medida = -medida;
}
}

return medida;
}

// Comprobación de que existe una conexión con el robot
// Devuelve 1 si se produce la conexión y 0 en caso contrario
bool comprobarRobot(void){
    EthernetClient cliente = servidor.available();
    if (cliente) {
        return 1;
    } else{
        cliente.stop();
        return 0;
    }
}

// Función para enviar estado del sistema, sensor fotoélectrico y recibir la orden de movimiento del propio robot
// El robot envía la orden, ya sea de recibir el estado del sistema o el movimiento deseado
// Una vez recibido, el Arduino envía los datos necesarios y mueve la cinta si se encuentra en estado remoto
bool enviarRobot(char dato){
    EthernetClient cliente = servidor.available();
    String recepcion;
    if (cliente) {
        while (cliente.connected()) {
            if (cliente.available()) {
                recepcion = cliente.readString();
                Serial.println(recepcion);
                if(recepcion == "STATUS"){
                    String envio = (String) dato + ";X=" + String(posicion, 2) +
                    ";Y=" + String(posy, 2);
                    String fotoele = ";F=" + String(digitalRead(SENSOR_FOTO));
                    envio = envio + fotoele;
                    cliente.println(envio);
                    Serial.println(recepcion);
                }
            }
        }
    }
}
```

```
}

if(recepcion.indexOf('M') > -1){
    if(estado == 'r'){
        int igual_pos = recepcion.indexOf("=");
        int fin_pos = recepcion.indexOf(";",igual_pos);
        String movimiento = recepcion.substring(igual_pos+1, fin_pos)
        ;
        objetivo = movimiento.toInt();
        estado = 'm';
    }
}

if(recepcion.indexOf('R') > -1){
    if(estado == 'r'){
        int igual_pos = recepcion.indexOf("=");
        int fin_pos = recepcion.indexOf(";",igual_pos);
        String movimiento = recepcion.substring(igual_pos+1, fin_pos)
        ;
        objetivo = movimiento.toInt() + posicion;
        estado = 'm';
    }
}

cliente.stop();
}

return 0;
}
```


6 Desarrollo en Robotstudio

La idea principal del proyecto es orientar el sistema para la realización de prácticas de laboratorio, por lo que el trabajo a realizar por los alumnos está en la programación en Robotstudio. Por esta razón, este capítulo va orientado a dar las pautas necesarias para el funcionamiento del sistema y su integración en el ecosistema de ABB. Este capítulo se basa en gran medida al trabajo previo de Hinojosa[5].

6.1 Funcionamiento sin microcontrolador

El caso más simple de funcionamiento es cuando no hay microcontrolador. En este caso se utilizarán las entradas y salidas digitales del controlador para realizar los movimientos. La conexión entre las salidas digitales de la caja y el controlador se realiza mediante un módulo de interfaz como el mostrado en la figura 6.1.



Figura 6.1 Amplificación señales calibre digital a 5V.

Dicho módulo se conecta a la placa DSQC 652 que se encuentra montada en el controlador del robot. Este dispositivo está diseñado para manejar señales digitales entre el sistema del robot y sistemas como el de este proyecto. Los pines expuestos en el módulo de interfaz son los siguientes:

- Entradas (DI, Digital Inputs). Pines del 1 al 8 que corresponden a los DI01 a DI08.
- Salidas (DO, Digital Outputs). Pines del 14 al 24 que corresponden a los DO01 a DO08.
- GND. Pin 24.

- VCC 24V. Pin 25.

En los pines de salida se conectarán las señales de Avance y Retroceso, mientras que en los de entrada estarán la señal del sensor fotoeléctrico, la emergencia, el estado de la señal local y la señal micro.

Las funciones referentes a las señales digitales se pueden consultar en el manual de RAPID [1]. Las más relevantes para el uso básico requerido en el laboratorio:

- SetDO, cambio de valor en salida digital. Uso: SetDo señalDO, valor;
- Lectura de entrada digital. Se puede realizar simpelmente utilizando el nombre de la señal. Usos: estado-actual := señalDI; IF señalDI = valor DO ...
- WaitDI, espera en el programa hasta que una señal de entrada tome cierto valor. Uso: WaitDI señalDI, valor;

Un ejemplo de programa es el que se puede ver en 6.1. Este programa consiste en hacer avanzar la cinta hasta que se enciende el sensor fotoeléctrico. Una vez detecta esa señal, el robot recoge la pieza y se repite la operación haciéndolo retroceder. Para ello, se debe definir el *workobject* de la cinta transportadora para mover el brazo robótico sobre sus ejes. Posteriormenete se debe definir la posición de recogida del brazo donde se activa el sensor fotoeléctrico. Las demás posiciones se generarán mediante *offsets* de la posición de recogida. El trabajo con el brazo robótico queda a cargo de futuros alumnos, aquí se muestra exclusivamente el trato con las señales digitales. Para simplificar los nombres de las señales se utilizan nombres descriptivos, no el que realmente aparece en RobotStudio.

Código 6.1 Ejemplo de programa sin microcontrolador.

```
MODULE Module1
    !Aquí deben estar definidas las posiciones y variables auxiliares
    !
    PROC main()
        !Se procede a avanzar
        SetDO señalAvance, 1;
        SetDO señalRetroceso, 0;

        !Se espera a que se active el sensor fotoeléctrico
        WaitDI señalFoto, 1;
        !Se para la cinta
        SetDO señalAvance, 0;
        SetDO señalRetroceso, 0;

        !Movimiento del brazo robótico para recoger la pieza y colocarla
        en una posición avanzada donde haya que retroceder
        ...

        !Misma operación a la inversa
        SetDO señalAvance, 0;
        SetDO señalRetroceso, 1;

        !Se espera a que se active el sensor fotoeléctrico
        WaitDI señalFoto, 1;
        !Se para la cinta
```

```

SetDO señalAvance, 0;
SetDO señalRetroceso, 0;

!Movimiento del brazo robótico para recoger la pieza y colocarla
en la posición original
...
ENDPROC
END MODULE

```

6.2 Funcionamiento con microcontrolador

El funcionamiento del sistema con microcontrolador permite al controlador del robot tener la información completa sobre la posición de la pieza a lo largo de la cinta y del estado del sistema. Para poder transmitir los datos se realizará una conexión TCP/IP donde Arduino y controladora se enviarán información mutuamente. Para que esta opción esté habilitada en RobotStudio, es necesario añadir la funcionalidad "PC Interface" durante la creación de la estación. El procedimiento para realizar la conexión y el envío de datos se encuentra explicado demostrado en Hinojosa[5], mientras que en el código 6.2 se encuentra un ejemplo funcional.

La comunicación es bidireccional: la controladora del robot envía una orden mientras que el Arduino responde en consecuencia. Existen tres órdenes enviadas por la controladora diferentes:

- Solicitud de posición y estado, se envía la cadena "STATUS" al Arduino.
- Orden de movimiento absoluto. Se envía una cadena con el carácter "M" seguido de la posición final en milímetros con el formato ";X=" + posicion.
- Orden de avance discreto. Se envía una cadena con el carácter "R" seguido de los milímetros que tiene que avanzar la cinta con el formato ";X=" + avance.

En el caso de recibir una cadena "STATUS", el Arduino responderá con una cadena con el siguiente formato:

Carácter de estado + ";X=" + posición + ";Y=" medida del calibre + ";F=" + valor del sensor fotoeléctrico.

El microcontrolador del Arduino está continuamente comprobando si existe una orden por parte de la controladora, ya que se comprueba en muchas partes del código, independientemente de si se encuentra en modo local o en modo remoto. Sin embargo, si se encuentra en modo local únicamente responderá a la solicitud de "STATUS", ya que no puede recibir órdenes de movimiento en este modo. En modo remoto, si recibe una orden de movimiento, pasará automáticamente al estado de movimiento con los datos recogidos y realizará dicho movimiento.

A continuación se incluye el código 6.2 necesario para la lectura del estado y la posición, además del envío de movimiento absoluto y discreto.

Código 6.2 Ejemplo de programa con microcontrolador.

```

MODULE Module1
!Declaración de variables del programa
!Objeto de socket de conexión
VAR socketdev my_socket;

!Variable de estado

```

```
VAR string estado:=0;

!Cadenas de caracteres de recepción
VAR rawbytes receive_string;
VAR string string1;
VAR string posx_str;
VAR string posy_str;
VAR string fotoele_str;

!Posiciones de los datos a lo largo de la cadena recibida
VAR num xpos;
VAR num ypos;
VAR num fepos;

!Valores numéricos finales recibidos
VAR num posx;
VAR num posy;
VAR byte fotoele;

!Comprobación de recepción correcta
VAR bool okposx:=true;
VAR bool okposy:=true;

!Bucle principal del programa
PROC main()
    !Se cierran los posibles sockets abiertos
    SocketClose my_socket;

    WaitTime 0.2;

    !Función de lectura de posición y estado
    leer;

    !Movimiento absoluto
    mov_abs(200);

    !Movimiento discreto
    mov_dis(100);

    WaitTime 0.5;
ENDPROC

!Función de apertura de socket
PROC abricomunicacion()
    SocketCreate my_socket; !crea el socket
    SocketConnect my_socket, "192.168.50.200", 4012;
ENDPROC

!Función de lectura de estado y
PROC leer()
```

```

!Se abre el socket y conecta al Arduino
abrimunicacion;

!Escribe en el socket la cadena "STATUS" para que el Arduino envíe sus datos
SocketSend my_socket,\Str:="STATUS";
WaitTime 0.1; !espera un tiempo

!Recibe la respuesta
ClearRawBytes receive_string;
SocketReceive my_socket \RawData := receive_string,\Time:-
WAIT_MAX;

!Desempaquetta los bytes y los convierte en una cadena de caracteres
UnpackRawBytes receive_string, 1, string1 \ASCII:=32;

!Se buscan las posiciones de cada variable a lo largo de la cadena
xpos      := StrFind(string1, 1, "=");
ypos      := StrFind(string1, xpos+1, "=");
fepos     := StrFind(string1, ypos+1, "=");

!Se trocea la cadena para obtener subcadenas con los datos recibidos
estado    := StrPart(string1, 0, 1);
posx_str  := StrPart(string1, xpos+1, ypos - xpos - 3);
posy_str  := StrPart(string1, ypos+1, fepos - ypos - 3);
fotoele_str := StrPart(string1, fepos+1, 1);

!Se transforman las cadenas a los tipos de datos que les corresponden
okposx    := StrToVal(posx_str,posx);
okposy    := StrToVal(posy_str,posy);
fotoele   := StrToByte(fotoele_str);

!Conversión de pulsos a milímetros
posx := posx / 100;

!Se cierra el socket
ClearRawBytes receive_string;
SocketClose my_socket;
ENDPROC

!Funciones de movimiento. Se envía una cadena con un carácter y la distancia a recorrer. Si el carácter es "M", el movimiento es absoluto, mientras que si es "R" es relativo.
PROC mov_abs(num distancia)
abrimunicacion;

```

```
SocketSend my_socket,\Str:="M;X=" + ValToStr(distancia*100) +
";";
WaitTime 0.1;
SocketClose my_socket;
ENDPROC

PROC mov_dis(num distancia)
abrimunicacion;
SocketSend my_socket,\Str:="R;X=" + ValToStr(distancia*100) +
";";
WaitTime 0.1;
SocketClose my_socket;
ENDPROC
ENDMODULE
```

Una vez obtenida la posición de la pieza, hay que tener en cuenta que estos están expresados en unos ejes concretos expresados desde el comienzo de la cinta. Por ello, hay que definir un *workobject* donde estos ejes tengan la misma dirección y, posteriormente encontrar una equivalencia para que el brazo robótico sea capaz de alcanzar la posición exacta de la pieza.

7 Resultados

8 Conclusiones

Índice de Figuras

1.1	Interfaz hombre-máquina. Distribución en la tapa.	3
1.2	Distribución de conexiones digitales	3
2.1	Arduino Mega 2560	5
2.2	Ethernet Shield	7
2.3	Driver L298N	8
2.4	Tipos de conexiones del Driver L298N	9
2.5	Señales encoder rotativo	9
2.6	Calibre digital y cables usados	10
2.7	LCD 16x2	11
2.8	Módulo de control de LCD por I2C	11
2.9	Sensor fotoeléctrico OMRON E3JK-R4M	12
2.10	Cinta transportadora con elementos integrados	12
3.1	Logo de KiCAD	15
3.2	Switches de estado	17
3.3	Ejemplo de optoacoplador	17
3.4	Implementación de optoacopladores	18
3.5	Búfer de tensión de la señal MICRO a 5V	18
3.6	Conexiones flecha selección	19
3.7	Implementación de optoacopladores para modo sin microcontrolador y emergencia	19
3.8	Alimentación calibre digital a 1.5V	20
3.9	Amplificación señales calibre digital a 5V	20
3.10	(a) Vista frontal de la placa tapa. (b) Vista trasera de la placa tapa	21
3.11	(a) Vista frontal de la placa fondo. (b) Vista trasera de la placa fondo	22
4.1	(a) Vista general de la base de la caja. (b) Vista de planta de la base de la caja	24
4.2	Ensamblaje de la base con el soporte. Descripción por colores: a) Gris: Fuente. b) Rojo: L298N. c) Azul: Arduino. d) Verde: placa de conexiones. e) Amarillo: Soporte tapa	24
4.3	a) Vista general de la tapa de la caja. b) Vista inferior de la tapa de la caja	25
4.4	Vista inferior de la tapa ensamblada. Descripción por colores: a) Verde: placa de conexiones. b) Naranja: LCD c) Rojo: Seta de emergencia. d) Gris: Flechas de selección. e) Amarillo: Botones intro y escape. f) Morado: Selectores de modo	25
5.1	Diagrama de flujo del programa	27
6.1	Amplificación señales calibre digital a 5V	43

Índice de Tablas

2.1	Características Arduino Mega 2560	6
3.1	Listado de señales	16

Índice de Códigos

5.1	Declaración de variables, "main.h"	29
5.2	Código principal Arduino, "main.ino"	30
6.1	Ejemplo de programa sin microcontrolador	44
6.2	Ejemplo de programa con microcontrolador	45

Bibliografía

- [1] ABB, *Rapid instructions, functions and data types*.
- [2] Arduino, <https://arduino.cc>.
- [3] Frank de Brabander, <https://github.com/fdebrabander/arduino-liquidcrystal-i2c-library>.
- [4] Jorge Andrés Tapia Herrera and Luis Fernando Castaño Castaño, *Control de equipo de posicionamiento de piezas semiautomático en zona de trabajo de robot*, Universidad de Sevilla, 2018.
- [5] Mauricio Hinojosa Rea, Luis Fernando Castaño Castaño, and David Muñoz de la Peña Sequedo, *Conexión de robotstudio y arduino mediante tcp/ip para la recolección y envío de datos de posicionamiento de cinta transportadora*, Universidad de Sevilla, 2019.
- [6] Martin Thalheimer, <https://sites.google.com/site/marthalprojects/home/arduino/arduino-reads-digital-caliper>.
- [7] Varios, <https://www.arduino.cc/en/reference/ethernet>.