Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Diseño de un sistema de almacenamiento automático mediante Arduino

Autor:

Alonso Hermosilla, Alfredo

Tutor:

Zalama Casanova, Eduardo

Ingeniería de Sistemas y Automática

Valladolid, julio 2017.

RESUMEN

Este proyecto consiste en el diseño y la construcción de un pequeño almacén automatizado en el que gracias a un microcontrolador Arduino se mueve una transpaleta que carga y descarga palés entre los nueve cubículos del almacén y una cinta transportadora. El sistema cuenta con modos de funcionamiento manual y automático, y con posibilidades de manejo mediante joystick y botones, monitor serie de Arduino y aplicación Android. La idea es que la maqueta sirva para su utilización en las prácticas de la asignatura Mecatrónica de cuarto curso del Grado en Electrónica Industrial y Automática, y por ello se incorporan gran parte de los elementos electrónicos, sensores, motores y mecanismos vistos en la asignatura.

PALABRAS CLAVE

- Almacén
- Automático
- Arduino
- Android
- Mecatrónica

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

GLOSARIO DE TÉRMINOS Y ABREVIATURAS

IDE: Entorno de Desarrollo Integrado

PC: Ordenador Personal

PLC: Controlador Lógico Programable

AS/RS: Sistema Automatizado de Almacenaje y Recuperación

PCB: Placa de Circuito Impreso

PLA: Ácido PoliLáctico (polímero)

ABS: Acrilonitrilo Butadieno Estireno (plástico)

CAD: Diseño Asistido por Ordenador

CAM: Fabricación asistida por Ordenador

DM: Densidad Media (madera)

CNC: Control Numérico por Computador

PWM: Modulación por Ancho de Pulso

I2C: protocolo de comunicación serie “Inter-Integrated Circuit”

USB: Bus Serie Universal

LED: Diodo Emisor de Luz

GND: Masa de un circuito

AVR: familia de microcontroladores de ATMEL

Sketch: programa que se sube a un microcontrolador

SRAM: Memoria Estática de Acceso Aleatorio

EEPROM: Memoria Programable de Solo Lectura Borrable Eléctricamente

TFT: tecnología de Transistor de Película Delgada (utilizada en pantallas LCD)

LCD: Pantalla de cristal líquido

XML: Lenguaje de Marcado Extensible

MAC: dirección de Control de Acceso al Medio

1 INTRODUCCIÓN Y OBJETIVOS

1.1 MARCO DE TRABAJO Y JUSTIFICACIÓN DEL TFG

A lo largo de las últimas décadas se ha comprobado que la interacción e integración entre distintos campos de la ingeniería da lugar a una sinergia que permite el desarrollo de proyectos más complejos abarcando muchos campos de aplicación con mejores resultados que actuando por separado como lo hacían tradicionalmente. Por ello, la comunicación y trabajo conjunto entre los diferentes departamentos de ingeniería de las empresas es un hecho indispensable para mantenerse competitivas en el mercado.

Ante este hecho incuestionable, durante los años ochenta surgió el término "mecatrónica" como la integración de diversas disciplinas de la ingeniería como son la mecánica, la electrónica, el control y la informática para el mejor diseño y desarrollo de productos y procesos industriales gracias a su gran versatilidad. El conjunto de disciplinas que abarca la mecatrónica se puede observar en la imagen de la figura 1. [1]

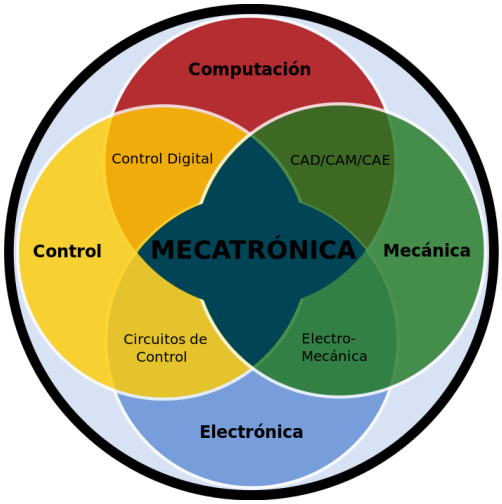


Figura 1. Disciplinas que integran la mecatrónica

Los principales objetivos de la mecatrónica son:

- La automatización de la maquinaria para agilizar los procesos de fabricación.
- El desarrollo de nuevos productos inteligentes que mejoren la calidad de vida de los usuarios.
- La armonía entre los elementos mecánicos y electrónicos para facilitar los procesos productivos.

Un ingeniero mecatrónico es, por tanto, un profesional con un conocimiento multidisciplinar que puede desarrollar sistemas automatizados que integren elementos de diversos campos de la ingeniería siendo capaz de seleccionar los mejores métodos para conseguir las mejores calidades, funcionalidades y costes. Este tipo de profesional está muy demandado en numerosos campos industriales como la industria manufacturera, automovilística, alimentación, textil, comunicaciones, etc.

La gran importancia de la mecatrónica en el ámbito industrial ha dado lugar a que en el Grado en Electrónica Industrial y Automática de la Universidad de Valladolid se oferte con carácter optativo la asignatura "Mecatrónica" dentro del primer cuatrimestre del cuarto curso. En esta asignatura se pretende dar una visión global de la mecatrónica como integración de disciplinas mediante un carácter eminentemente práctico. Además de clases teóricas sobre programación de microcontroladores, sensores, actuadores o mecanismos, la asignatura consta de un gran número de horas prácticas en las que los alumnos aprenden a diseñar piezas en 3D mediante el uso del programa de diseño CAD Autodesk Inventor y a programar microcontroladores Arduino para su uso con diferentes sensores, actuadores y elementos electrónicos. Todo lo aprendido en la asignatura se plasma con la realización de un proyecto mecatrónico que se presenta ante la clase al finalizar el cuatrimestre. [2]

Hasta ahora las prácticas de programación de Arduino de la asignatura consisten en montar en una protoboard un circuito simple empleando el elemento correspondiente a cada práctica (motor paso a paso, sevomotor, LCD, sensor de ultrasonidos…) y programar el Arduino para realizar ciertas acciones con ese componente. Este Trabajo Fin de Grado está destinado al diseño y construcción de un sistema mecatrónico consistente en un almacén automatizado de un tamaño manejable que contenga gran parte de los elementos utilizados en los laboratorios de la asignatura. De esta manera, los alumnos podrán realizar sus prácticas sobre un sistema en el que todos los elementos están integrados para cumplir con una serie de funcionalidades en lugar de ir montando y desmontando pequeños circuitos en placas de pruebas. Con ello se pretende conseguir una mayor motivación por parte de los estudiantes, así como una mejora en el aprendizaje de los contenidos de la asignatura.

Los alumnos podrán realizar prácticas de dificultad creciente empezando por controlar cada uno de los elementos que conforman la maqueta de forma individual para posteriormente ir integrando unos con otros creando programas más complejos que subirán al microcontrolador Arduino que gobierna el sistema.

No obstante, se va a desarrollar un programa integrador en el que se explotan todas las funcionalidades del dispositivo para que pueda utilizarse por cualquier usuario de forma similar a como podría manejarse un almacén automatizado industrial. Además, la estructuración del programa permitirá que muchas de las funciones contenidas puedan servir de apoyo a los alumnos o incluso que su desarrollo sea el fin de alguna de las prácticas.

El programa permitirá el manejo del almacén en modo manual o automático y se podrá controlar mediante tres interfaces:

- Un cuadro de mandos físico compuesto por un joystick y tres botones, elementos que bien pueden ser protagonistas de algunas de las prácticas e la asignatura.
- El monitor serie del IDE de Arduino con el que es conveniente que los alumnos se familiaricen pues resulta útil principalmente a la hora de programar y depurar errores.
- Una aplicación Android que dotará al dispositivo de un mayor atractivo y rapidez de manejo pudiendo además despertar el interés de los alumnos en el mundo de la programación de aplicaciones móviles.

Tras la construcción del primer prototipo que es lo que abarca este TFG, el departamento y el responsable de la asignatura considerarán la fabricación de más unidades pudiendo incluir las modificaciones oportunas para que todos los alumnos de la asignatura cuenten con un ejemplar con el que desarrollar sus prácticas.

1.2 ESTADO DEL ARTE: LOS ALMACENES AUTOMÁTICOS

El crecimiento global experimentado por muchas empresas de diversos sectores a lo largo de las últimas décadas ha dado lugar a necesidades de mejora de sus sistemas de almacenaje de productos o materias primas para conseguir una mayor productividad, eficiencia y seguridad en los procesos de introducción y extracción de los materiales para su posterior tratamiento o venta. Este hecho junto con el gran desarrollo tecnológico producido desde mediados del siglo XX ha dado lugar a la aparición y expansión de los almacenes automatizados en sustitución de los tradicionales en los que prima la mano de obra poco cualificada y el uso de carretillas elevadoras.

No obstante, los almacenes tradicionales siguen jugando un papel importante principalmente en las empresas en las que los volúmenes de producción no son muy elevados y por tanto no compensa económicamente

el gran desembolso que conlleva la adquisición de un almacén automatizado. Además de los motivos económicos, los almacenes tradicionales cuentan con las ventajas de que son más fáciles de adquirir, su construcción es más simple y rápida, y pueden localizarse en cualquier lugar pudiendo por tanto ser ubicados próximos a los lugares de consumo.

Sin embargo, las grandes empresas con altos niveles de producción han ido migrando hacia soluciones más automatizadas en las que el factor humano pierde peso en favor de las máquinas que gracias a los avances de la informática y las comunicaciones industriales son programadas para llevar a cabo las acciones necesarias para la factoría.

Los almacenes automatizados se basan en el principio de la mercancía al hombre, consistente en que los productos agrupados en unidades de carga se transportan mediante transelevadores desde las estanterías hasta los puestos de cabecera donde se ponen a disposición del operario o hasta sistemas de clasificación, transporte o expedición. Los elementos principales que integran los almacenes automáticos son los sistemas de movimiento de las cargas, las estanterías en las que se almacenan, el software de control que ordena todo el proceso y los sistemas de preparación de pedidos. En la figura 2 se muestra una imagen general de un almacén automatizado típico.

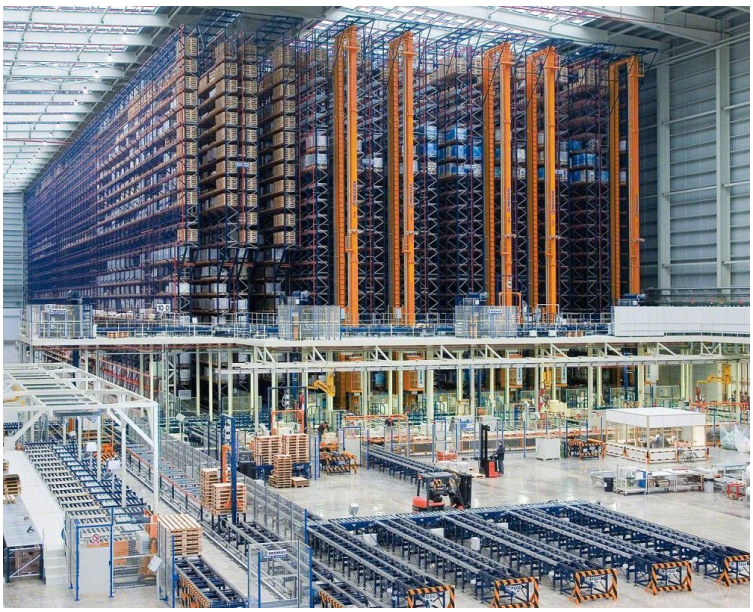


Figura 2. Almacén automatizado

Entre las principales ventajas de la elección de almacenes automatizados pueden mencionarse las siguientes:

- La superficie necesaria para el almacén es considerablemente menor a la de los convencionales puesto que las estanterías pueden llegar hasta gran altura al no tener las limitaciones de las carretillas elevadoras. Además, los pasillos cuentan con una anchura mucho menor pues es suficiente con que sean ligeramente más anchos que las cargas que se mueven por ellos consiguiendo así un alto aprovechamiento del espacio.

- Alto rendimiento y ahorro del tiempo gracias a la elevada velocidad de los desplazamientos consiguiendo ciclos de alimentación y extracción muy reducidos dando lugar a un gran número de movimientos diarios.

- Gestión automática mediante un sistema de control que permite registrar las entradas y salidas de productos, así como su ubicación y otros datos como prioridades en la extracción de mercancías. Además, los recorridos de las máquinas se optimizan para conseguir el mayor rendimiento en el proceso.

- Versatilidad: los almacenes automáticos son utilizables en gran variedad de sectores pues se pueden almacenar mercancías de todo tipo desde materias primas hasta productos listos para la venta siempre que se puedan agrupar en unidades de carga transportables por los transelevadores.

- Seguridad: al ser sistemas automatizados la presencia de operarios en el almacén en contacto con las máquinas y cargas es mínima por lo que el riesgo de accidentes laborales es mucho menor al de los sistemas de almacenaje tradicional.

Para la selección de la configuración de almacén más adecuada de entre las posibilidades del mercado, una industria debe realizar un estudio técnico previo en el que se aborden cuestiones como la base de carga (contenedor, caja, bandeja…), la disposición de las estanterías según el espacio disponible y las características del flujo de productos, la frecuencia de operaciones o la velocidad necesaria para cumplir con los requerimientos.

En los almacenes, las unidades de carga deben someterse a un control de entrada en el que se compruebe que sus dimensiones y pesos son adecuadas para su almacenaje. También es necesaria la implementación de planes de mantenimiento preventivo para minimizar o evitar las paradas por averías en alguno de los componentes que conforman el almacén.

A continuación, se detallan las formas más comunes de almacenaje automatizado presentes en el mercado actual desde sistemas AS/RS (automated storage and retrieval system) hasta carruseles rotatorios:

- **Transelevadores:** máquinas con una plataforma o transpaleta que se desplazan a lo largo de un pasillo para la ubicación o extracción de mercancías en las estanterías de almacenaje situadas a ambos lados del pasillo. Estas estanterías pueden llegar hasta alturas de 40 metros pudiendo adaptar el sistema a las necesidades de cada almacén en cuanto a dimensiones, capacidad de carga o tiempos de ciclo. Además, estos sistemas pueden adecuarse a condiciones de trabajo especiales como humedad extrema o temperaturas de congelación. La figura 3 consiste en un transelevador que se mueve entre dos estanterías de paquetes.

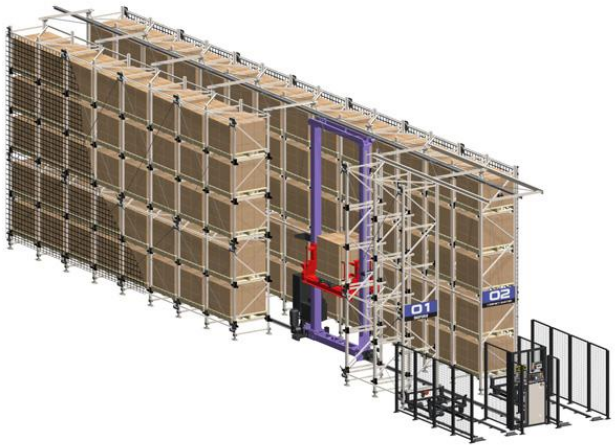


Figura 3. Almacén con transelevador

Los transelevadores realizan tres tipos de movimientos: longitudinal a lo largo del pasillo sobre un carril situado en el suelo, vertical a lo largo de la propia columna del transelevador y transversal para la ubicación o extracción del material.

Un procesador central (PC o PLC) da las órdenes a los diferentes transelevadores para realizar los movimientos de productos requeridos optimizando todas las transacciones dentro del almacén. Este procesador recibe las órdenes de realizar movimientos por parte del "host" de la empresa y las transmite a los transelevadores a través de un sistema de coordenadas indicando la posición en la que se tienen que situar. Los accionamientos del transelevador se controlan con variadores de frecuencia vectoriales que controlan el posicionamiento con técnicas como telémetros láser o infrarrojos.

Se trata de la forma más común de almacenamiento automático y cuenta con dos variantes según el peso de las cargas que manipula:

- **Unit Load:** diseñados para transportar cargas de mucho peso (hasta 10.000 kg) y dimensiones variadas. Un ejemplo de este tipo de transelevador se presenta en la figura 4. Lo más habitual es el uso de palés como base de almacenaje aunque también se pueden utilizar otros elementos como contenedores o bandejas.

Figura 4. Transelevador Unit load de Mecalux

- **Mini Loads:** sistemas diseñados para manejar y almacenar cargas pequeñas con pesos de hasta 300kgs mediante cajas de plástico o cartón, bandejas, contenedores o incluso en unidades de producto suelto. Como se aprecia en la figura 5, en uno de los extremos de la estantería estos sistemas suelen contar con una zona de picking formada por transportadores en los que el transelevador deposita cargas para ser acercadas al operario.

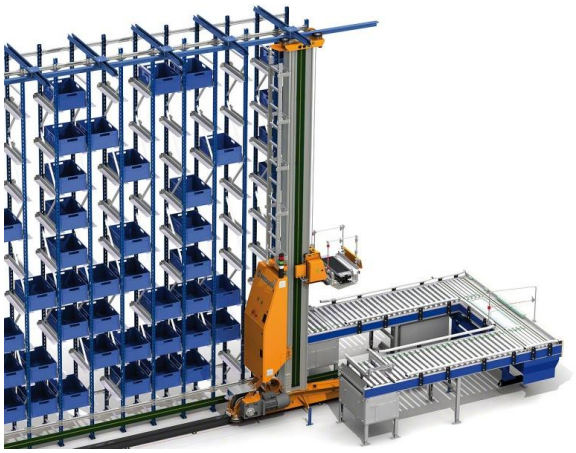


Figura 5. Transelevador Mini Load de Mecalux

- **Almacén automático vertical o Pallet Shuttle:** en este sistema en lugar de haber varios pasillos con un transelevador en cada uno, únicamente hay un pasillo central con un sistema de transelevador o de elevador con lanzaderas. Este dispositivo dispone de un carro móvil que una vez situado frente al canal que corresponda, se introduce en la estantería desplazándose sobre guías pudiendo cargar y descargar palés a una distancia de hasta 40 metros de profundidad de forma que se consigue un almacenamiento mucho más denso. Este tipo de almacén se muestra en la figura 6.

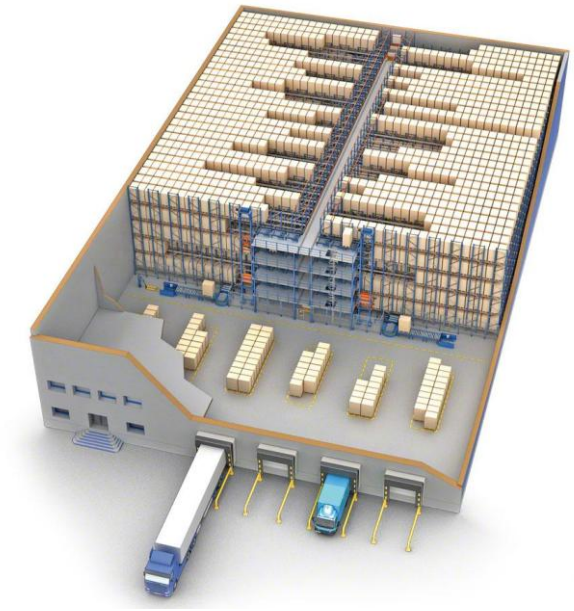


Figura 6. Almacén Pallet Shuttle

El elemento móvil del pasillo central puede tratarse de un transelevador como los descritos en el apartado anterior (figura 8), pero también hay casos en los que se trata de un conjunto de un elevador vertical y tantas lanzaderas como pisos tenga el almacén (figura 7). El elevador sube las cargas a la entrada del almacén hasta el piso que corresponda y las deposita en la lanzadera de dicho piso que se mueve horizontalmente sobre raíles a gran velocidad hasta llegar a la posición deseada donde el pallet shuttle se introduce en el almacén. En estos sistemas el número de movimientos o ciclos/hora se multiplica por el número de niveles de modo que es la forma de almacén más eficiente cuando se precisa de una alta capacidad, compactación y velocidad de movimientos. [3-7]

Figura 7. Pallet Shuttle con elevador y lanzaderas

Figura 8. Pallet Shuttle con transelevador

- **Carrusel rotativo horizontal:** los sistemas de carrusel consisten en una pista transportadora ovalada con una serie de arcas acordes a los pesos que contienen, las cuales pueden estar suspendidas del techo o montadas sobre el suelo. La mayor parte de estos sistemas son semiautomáticos ya que son manejados por un operario situado en la zona de carga y descarga que ordena el movimiento rotatorio para que el arca requerida llegue a su posición y pueda introducir o extraer objetos manualmente. No obstante, también existen carruseles controlados por ordenador que operan de forma automática pudiendo ser una alternativa a los AS/RS de minicarga. Un sistema típico de carrusel horizontal puede apreciarse en la figura 9.

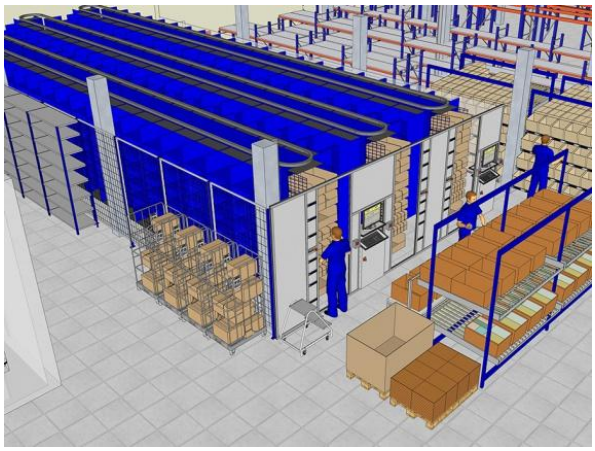


Figura 9. Carrusel rotativo horizontal

- **Carrusel rotativo vertical o paternoster:** el principio de funcionamiento de este sistema es el mismo que el de los horizontales pero con movimiento rotatorio vertical de forma que se ocupa menos espacio pero es necesaria una mayor altura de la nave estando limitada su capacidad por el techo del almacén. Son menos habituales que los horizontales al tener generalmente menor capacidad de almacenamiento y resultar más costoso su movimiento al tener que vencer a la fuerza de la gravedad. Su disposición típica se puede ver en la figura 10. [8]

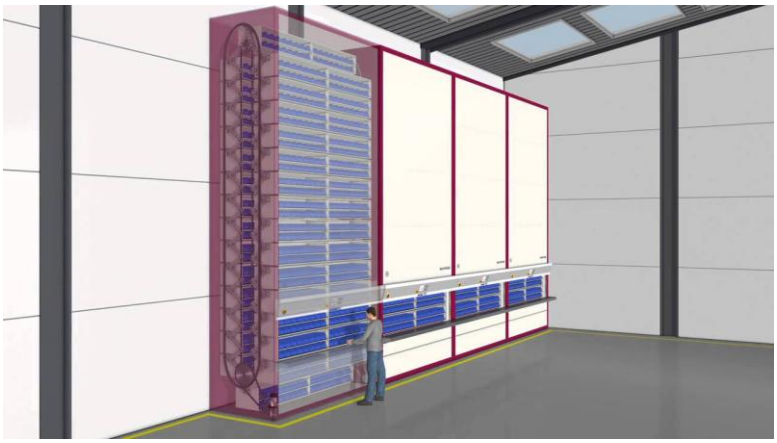


Figura 10. Carrusel rotativo vertical

En cuanto a proveedores de todos estos sistemas de almacenaje automático algunos de los más importantes a nivel mundial son los siguientes:

- **System Logistics:** proveedor global de soluciones innovadoras de intralogística y manipulación de material para la optimización de la cadena de soporte de almacenes, centros de distribución y plantas de producción de todo el mundo. Especialmente dedicados a la industria alimentaria desarrollan soluciones "a medida" de automatización de almacenes y *picking* que incluyen transelevadores, sistemas de manipulación, software y servicios.
  Desde su nacimiento ha formado parte de **SYSTEM**, un grupo italiano de propiedad privada afincado en Fiorano (Módena, Italia) que desarrolla soluciones con tecnología de última generación para la automatización industrial en diferentes sectores. [9]

- **Schaefer Systems International:** es una de las mayores proveedoras mundiales de soluciones de sistemas automatizados, almacenamiento y distribución, manipulación de materiales y software de logística. Desde su nacimiento en Burbach, Alemania

en 1937 esta empresa familiar ha crecido hasta contar con 16 centros de producción y más de 8.000 empleados. [10]

- **Westfalia:** empresa comprometida durante más de 35 años a innovar implementar soluciones de logística utilizando los AS/RS y equipos de manipulación de materiales convencionales. Westfalia Technologies Inc., con su oficina central en York, Pensilvania, es un proveedor líder en soluciones de logística para plantas, almacenes y centros de distribución. Desde 1992, Westfalia ha ayudado a empresas en las industrias de comidas, bebidas y químicos a optimizar los procesos en almacenes y maximizar el rendimiento y los ahorros. [11]

- **Daifuku:** empresa japonesa líder en el sector que desarrolló el primer sistema de almacenamiento y recogida automatizado en 1966. [12]

Dentro del mercado español destaca la empresa **Mecalux** que es una de las compañías punteras en el mercado de sistemas de almacenaje. Su actividad consiste en el diseño, fabricación, comercialización y prestación de servicios relacionados con las estanterías metálicas, almacenes automáticos y otras soluciones de almacenamiento. Compañía líder en España, se sitúa en el tercer puesto mundial en el ranking de su sector, con ventas en más de 70 países. [3]

Otras importantes empresas españolas dedicadas al almacenaje automático son ULMA Handling Systems (empresa que colabora con DAIFUKU) [7], Noega Systems [6] y Moinsa [4].

Figura 11. Empresas líderes en sistemas de almacenaje automatizados

1.3 OBJETIVOS

El objetivo principal de este TFG es el diseño y la construcción de un sistema mecatrónico consistente en un almacén automatizado de un tamaño manejable para poder ser utilizado por alumnos del Grado en Electrónica Industrial y Automática de la Uva en las prácticas de la asignatura "Mecatrónica". Se pretende que los alumnos aprendan la programación de microcontroladores Arduino y el uso de diferentes elementos electrónicos, sensores y actuadores de una forma práctica sobre un sistema en el que todos estos elementos se encuentran integrados interaccionando entre sí. En dichas prácticas los alumnos podrán aprender el uso y programación individual de cada elemento y según se vaya cogiendo soltura ir creando programas más complejos en los que el comportamiento de unos elementos dependa de otros y de la interacción con el usuario.

Asimismo, se pretende que el sistema pueda ser utilizado por cualquier usuario sin conocimientos técnicos de forma similar a como podría utilizarse un almacén automatizado real del tipo AS/RS. Para ello se realizará un programa de Arduino que contemple la posibilidad de utilización en dos modos:

- **Manual:** el usuario controla el movimiento de los diferentes motores en velocidad y sentido.

- **Automático:** el usuario solicita la realización de acciones como el almacenaje, extracción o movimiento de elementos en el almacén que son realizadas de forma automática por el sistema.

En el programa se incluye también la posibilidad de utilizar tres interfaces de control como son un cuadro de mandos físico, el monitor serie de Arduino y una aplicación Android para móviles y tablets que deberá ser programada mediante Android Studio y que se comunicará con el microcontrolador via bluetooth.

Para la consecución de estos objetivos principales se deben ir logrando una serie de hitos desarrollados de forma parcialmente paralela (dado que las decisiones tomadas en unos pueden influir en otros) hasta conseguir el sistema perfectamente operativo:

- Diseño mecánico del sistema AS/RS y una cinta transportadora de forma que se utilicen los tres tipos de motores vistos en la asignatura: motores paso a paso, de corriente continua y servomotores. Elección de todos los elementos necesarios, así

como el diseño de las piezas que se imprimirán en una impresora 3D.

- Diseño electrónico eligiendo los componentes que se van a utilizar abarcando una buena parte de los componentes empleados en las prácticas de la asignatura. Con los componentes determinados se deberá elegir una forma de alimentar el circuito adecuada a sus necesidades y se procederá al diseño de una placa de circuito impreso en la que se montarán o conectarán dichos componentes.

- Diseño del almacén con la ubicación de todos los componentes mecánicos y electrónicos de la forma más optimizada posible sin que unos elementos interfieran con otros, consiguiendo un buen acceso a los elementos más habitualmente manipulados y ofreciendo un buen aspecto visual al usuario.

- Adquisición de todos los componentes y piezas necesarios para proceder a la construcción tanto mecánica como electrónica del dispositivo.

- Programación del código Arduino que implemente todos los modos e interfaces mencionados de la forma más clara posible haciendo uso de funciones específicas de los diferentes dispositivos para poder ser utilizadas como parte de las prácticas de la asignatura.

- Programación de la aplicación Android para el manejo del almacén con un entorno visual agradable y sencillo, así como compatibilidad con dispositivos Android de diferentes tamaños de pantalla para que cualquier usuario pueda descargarla en su terminal.

1.4 ESTRUCTURA DE LA MEMORIA

La memoria del TFG se estructura en diferentes capítulos, algunos de los cuales prácticamente coinciden con los objetivos parciales mencionados, y divisiones de estos en subcapítulos.

Los ocho capítulos que componen la memoria son los siguientes:

- **Capítulo 1. Introducción y objetivos:** es el capítulo presente en el que se hace una introducción al proyecto describiendo en qué consiste, los fines que tiene y los objetivos marcados. Además, se hace un pequeño estudio sobre los almacenes automatizados industriales.

- **Capítulo 2. Diseño mecánico:** en este apartado se explica el proceso de diseño físico del sistema, así como la adquisición de los

elementos necesarios y la construcción del sistema diseñado. El capítulo se desglosa en cuatro subcapítulos correspondientes a los cuatro sistemas diferenciados del montaje: el propio almacén, el sistema de movimiento AS/RS, la transpaleta de carga/descarga y la cinta transportadora.

- **Capítulo 3. Diseño electrónico:** este capítulo abarca la elección de todos los componentes electrónicos que van a componer el sistema, así como el esquema de conexiones ente todos ellos a partir del cual se diseña la placa de circuito impreso en la que se soldarán o conectarán los diferentes elementos.

- **Capítulo 4. Programación Arduino:** a lo largo de este capítulo se explica el programa que se implementará en el microcontrolador Arduino que gobierna el sistema. En cada apartado se explican las funciones de un mismo grupo haciendo uso de diagramas de flujo para mayor claridad.

- **Capítulo 5. Programación Android:** inicialmente se hace una breve introducción a la programación en Android Studio y a continuación se detalla el procedimiento de programación de la aplicación realizada explicando los códigos xml y JAVA de cada actividad o fragmento incluyendo capturas de las diferentes pantallas para que la explicación sea más visual.

- **Capítulo 6. Estudio económico:** en este capítulo se estudian los costes que suponen la realización de este proyecto desglosando en costes directos e indirectos.

- **Capítulo 7. Conclusiones y líneas futuras:** en este apartado se detallan las conclusiones que se han podido extraer de la realización del proyecto, así como las posibles mejoras y ampliaciones que podrían realizarse en un futuro.

- **Capítulo 8. Bibliografía:** por último, se incorpora una lista de los recursos utilizados en el desarrollo del trabajo para cada uno de los aspectos de los que se compone.

Al final de la memoria se incorporan además una serie de anexos como son: una guía de usuario del almacén para que cualquier usuario pueda utilizarlo sin problema, los códigos Arduino y Android programados, y los planos tanto de las piezas diseñadas como del esquema electrónico y la PCB.

2 DISEÑO MECÁNICO

Lo primero a determinar para la realización de este TFG es el aspecto físico, eligiendo todos los elementos y mecanismos que van a formar parte del sistema. A lo largo de este capítulo se explica el proceso de diseño del almacén, elección de mecanismos, materiales y actuadores, así como la construcción de cada subsistema e integración en el conjunto. En el diseño de cada subsistema influyen las decisiones tomadas en otros por lo que, aunque en cada apartado se detalla uno de ellos, en realidad se ha ido diseñando todo en paralelo adecuando las características de unos a las necesidades de otros.

El diseño de las piezas necesarias se ha realizado con Autodesk Inventor, software CAD de diseño de sólidos en 3D creado por la empresa Autodesk. Se trata de un software paramétrico, lo cual significa que las dimensiones y formas de las piezas están determinadas por parámetros. Estos parámetros son variables o fórmulas que se encuentran almacenadas en un contenedor del archivo de diseño de tal forma que, si se modifican, el diseño se actualiza para reflejar el cambio. [13]

El diseño de piezas con Inventor se fundamenta en la creación de bocetos en planos y extruirlos para crear una forma en 3 dimensiones. Las piezas se pueden unir en ensamblajes mediante uniones y restricciones de diferentes tipos que permiten hacer una simulación del comportamiento del sistema en la realidad.

Para el proyecto se han ido diseñando las piezas necesarias que se obtendrán gracias a la impresión 3D, y además se han creado modelos sencillos de las piezas que se van a comprar como los motores o los husillos para poder hacer un ensamblaje de todo el conjunto de forma que pueda intuirse el aspecto final en la maqueta real y así puedan corregirse fallos de diseño antes de la construcción física.

El material empleado en la impresora 3D para la fabricación de las piezas ha sido PLA, uno de los dos filamentos más utilizados por este tipo de aparatos. Se trata de un termoplástico obtenido de recursos renovables como el almidón de maíz o la caña de azúcar por lo que, a diferencia de su principal competidor, el ABS (derivado del petróleo), es biodegradable y no emite gases nocivos. También se diferencia en que no requiere de cama caliente y necesita de menor temperatura para la impresión (190-200°C) por los 230-245°C del ABS de forma que el consumo de energía es menor y la impresión más sencilla. Por contra, es más frágil teniendo una vida más corta y su

proceso de mecanizado, taladrado, pintado o pegado tiende a ser más complicado. [14]

Para la impresión, las piezas diseñadas en Inventor se exportan como ficheros .stl que son los que entienden las herramientas CAM, las cuales se encargan de convertir estos ficheros en otros interpretables por los controles numéricos de las impresoras. El software CAM utilizado ha sido el Cura. En él aparece un modelo de la impresora seleccionada donde se colocan las piezas en la disposición deseada sobre la cama. Tras ello se escogen los parámetros de impresión como altura de capa, densidad o velocidad de impresión y por último se genera el G-code que se envía al controlador de la impresora para comenzar la fabricación.

2.1 BASE Y ESTANTERÍA

La estantería conforma el conjunto de cubículos en los que podrán ser almacenados los palés que lleguen al almacén. Sus dimensiones deben ir en consonancia con el sistema AS/RS que cargue y descargue los palés, así como con el tamaño de éstos. La base sobre la que descansan todos los elementos de la maqueta tendrá unas dimensiones lo más reducidas posible tales que quepan todos los componentes.

2.1.1 DISEÑO

En primer lugar, se ha determinado que la matriz de cubículos sea cuadrada ya que se considera lo más adecuado por estética. En cuanto al número de filas y columnas, 2x2 parece demasiado reducido mientras que con 4x4 sería necesario que los cubículos (y por tanto los palés) fueran bastante pequeños o bien que el tamaño de la maqueta aumentara más de lo deseado. Por tanto, se ha considerado una estantería de tres filas y tres columnas como la estructura ideal para el almacén.

Tanto la base como los bordes exteriores y baldas de la estantería se van a construir a partir de un tablero cuadrado de DM de 16mm de espesor y 122cm de lado. Este material es un aglomerado elaborado con fibras de madera aglutinadas con fibras sintéticas mediante presión y calor. Tiene una estructura homogénea y uniforme, así como una textura fina gracias a la cual sus caras y cantos presentan un acabado perfecto. Además, se trata de un tablero de bajo coste en el mercado actual y posee la rigidez necesaria para aguanta sin problemas todos los elementos que se van a colocar sobre él. [15]

Para la división vertical entre cubículos se va a utilizar un tablero de contrachapado de 4mm de espesor dado que su función no es estructural ni tiene que soportar ningún peso. Así además se gana en espacio horizontal de los cubículos debido al pequeño grosor del elemento separador.

Los cubículos tendrán unas dimensiones de 80mm de ancho, 80 de profundidad y 74 de alto. Estas medidas se han determinado en consonancia con las de los palés, la transpaleta y el transelevador en su conjunto que se explican en el siguiente apartado.

Cabe mencionar que la primera fila de cubículos no se encuentra a ras de la base, sino que está elevada 40mm (56 contando la balda) debido a que la transpaleta no se va a poder situar a ras de la base. Esto se debe a que la posición del principio de los tornillos verticales por los que sube y baja el mecanismo está condicionado por la altura de los motores y sus acoples como se verá en el siguiente apartado. Además, a los pies de la estantería en el centro de la maqueta se va a situar la PCB por lo que la transpaleta debe quedarse a una distancia prudencial para no tocar sus elementos.

Tras varias modificaciones a lo largo del proceso de diseño de otros elementos, finalmente las dimensiones de la base se han fijado en 70x30cm. El conjunto base-estantería diseñado en Inventor puede verse en la figura 12 y para más detalle, al igual que con el resto de piezas, puede consultarse el plano correspondiente en el Anexo 4 al final del documento.

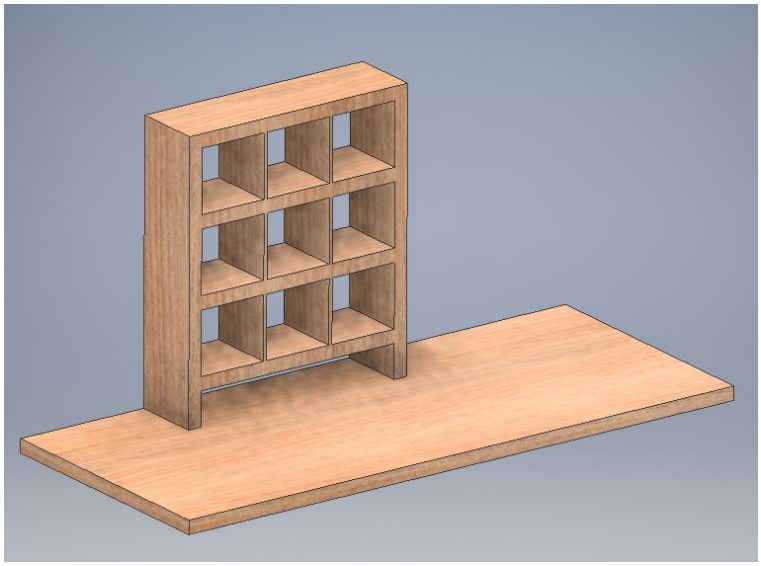


Figura 12. Estantería en la base

2.1.2 CONSTRUCCIÓN

Para la construcción de la estantería inicialmente se ha adquirido el tablero de DM cortado en las siguientes piezas por parte del proveedor haciendo uso de una mesa de corte vertical:

Unidades	Dimensiones (cm)	Finalidad
1	70x30	Base
1	28x8	Cubierta superior de la estantería
3	24,8x8	Baldas de la estantería
2	31x8	Paredes laterales

También se ha adquirido un tablero de 40x60cm y 4mm de espesor del que mediante una regla y un cúter se han obtenido seis piezas cuadradas de 8cm de lado que servirán como división entre cubículos.

Por un lado de la cubierta y la balda inferior, y por los dos lados de las otras dos baldas se han hecho dos hendiduras de 4mm de ancho y 4mm de profundidad en las posiciones apropiadas para que al introducir en ellas las piezas cuadradas anteriores se obtengan tres cubículos del mismo ancho. Para conseguir estas hendiduras se ha hecho uso de una sierra ingletadora de madera.

Para las uniones de las baldas y la cubierta con las dos paredes laterales se ha seguido el siguiente procedimiento utilizando un taladro, una broca con tope, espigas de madera, marcadores y cola blanca. La mayoría de estos elementos pueden apreciarse en la figura 13.

- Con el taladro y la broca (de diámetro igual a las espigas y los marcadores) se hacen dos agujeros en una de las dos piezas a unir. La profundidad de dichos taladros debe ser igual a la mitad de la longitud de las espigas, lo cual se consigue gracias al tope de la broca.
- Se introducen los marcadores en los agujeros realizados y se coloca la segunda pieza en la posición en la que se desea que quede. Haciendo un poco de presión se consigue que los marcadores dejen una pequeña muesca en la pieza quedando así determinada la posición exacta de los agujeros a realizar en ella.
- Se sacan los marcadores de la primera pieza y se taladra la segunda en las marcas con una profundidad igual a la anterior de forma que el conjunto de los agujeros de las dos piezas tenga la misma longitud que una espiga.
- Se introducen las espigas en los agujeros de una de las piezas y a continuación se inserta la otra habiendo encolado las espigas previamente. Tras unos minutos en los que se seca la cola, las dos piezas quedan perfectamente unidas.

Una vez se insertan en las hendiduras los cuadrados separadores ya se tiene la estantería cuyo aspecto final es el de la figura 14.

Figura 13. Broca con tope, espigas y marcadores

Figura 14. Estantería

Con el mismo método que para unir las baldas y las paredes de la estantería, se unen las paredes a la base en la posición determinada en el diseño.

2.2 SISTEMA DE CARGA Y DESCARGA

El sistema de carga y descarga debe ser un mecanismo situado delante de la estantería a una distancia constante de ésta que se mueva entre cubículos gracias al sistema AS/RS. Debe ser capaz de entrar en un cubículo hasta la posición adecuada para coger una carga y salir hasta su posición habitual para transportar la carga por delante de la estantería hasta su posición final.

2.2.1 ELECCIÓN Y DISEÑO DEL MECANISMO

En primer lugar, hay que determinar la forma de coger y dejar cargas. Una posibilidad contemplada ha sido el uso de una pinza que al cerrarse agarrara la carga y al abrirse la soltara, pero este mecanismo implicaría utilizar un actuador adicional para el movimiento de la pinza. Además, se debería incluir un sensor de fuerza o las cargas deberían ser siempre del mismo tamaño y forma para que la pinza las cogiera adecuadamente. Estos inconvenientes han llevado a optar por el sistema más utilizado en los almacenes automatizados reales, que son los palés.

Los palés se han diseñado de forma que su tamaño sea algo menor al ancho de los cubículos para que haya cierta holgura a la hora de introducirlos.

El elemento terminal del sistema de carga/descarga será una especie de transpaleta con dos brazos de tamaño acorde a los palés dejando también algo de holgura para poder cogerlos más fácilmente.

Para coger una carga, la transpaleta se colocará enfrente del palé por el movimiento del AS/RS y una vez encarada introducirá los brazos en los agujeros del palé con un movimiento lineal. Con el movimiento vertical del AS/RS se elevará el palé dejando de tener contacto con el estante (o la cinta) para terminar retrayendo los brazos y así tener el palé con la carga encima fuera de la estantería o cinta. El proceso de descarga será análogo, pero entrando al cubículo de forma que el palé quede algo por encima del estante y posteriormente bajando hasta que se apoye y los brazos pierdan el contacto con el palé. El elemento terminal del sistema y el palé se muestran en las figuras 15 y 16. La transpaleta cuenta con un agujero para colocar un sensor de ultrasonidos que reconocerá si los cubículos están ocupados o no.

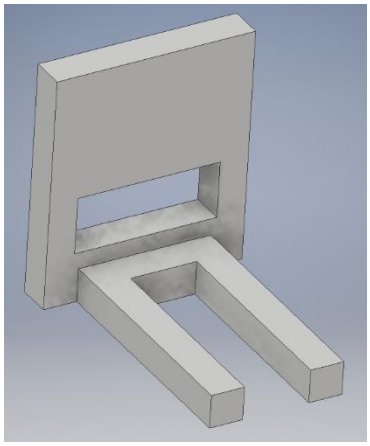


Figura 15. Transpaleta

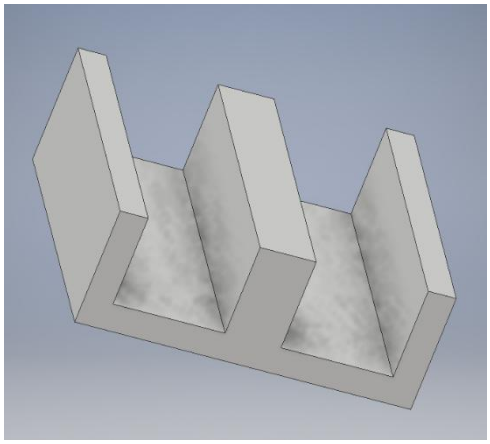


Figura 16. Palé

Por tanto, lo que se necesita de este mecanismo es un movimiento lineal horizontal perpendicular a la estantería. Los actuadores lineales son por lo general más caros, aparatosos y difíciles de conseguir que los rotativos de forma que se va a optar por utilizar un motor con un mecanismo de conversión de movimiento rotacional en lineal.

Se trata de un movimiento entre dos posiciones concretas para el que es necesaria bastante precisión mientras que el hecho de que el motor pueda girar una vuelta entera no es relevante. Por ello, el mejor actuador posible es un servomotor ya que un motor de corriente continua no es válido para casos en los que se requiera precisión y los motores paso a paso que sí son precisos son más pesados y aparatosos.

Queda determinar la forma de convertir el giro del servomotor a un movimiento lineal de la transpaleta entre dentro y fuera de los cubículos. Para ello hay que determinar primeramente la distancia necesaria entre ambas

posiciones. Para que al introducir el palé quede perfectamente situado en el estante se considera suficiente que la punta de los brazos llegue a introducirse 6cm dentro del cubículo (la profundidad total es 8cm). Al salir debe haber un margen de al menos 1cm hasta los estantes para que no haya posibilidad de choque de forma que el recorrido total del movimiento ha de ser de unos 7cm.

Algunos mecanismos considerados han sido el piñón-cremallera o una leva, desestimados el primero por la dificultad a la hora de obtener o fabricar los sistemas dentados y la segunda por el gran tamaño que sería necesario para desplazar los 7cm.

Finalmente se ha optado por un mecanismo biela-manivela con cuyo diseño en Inventor se ha asegurado que se pueden alcanzar los 7cm de una forma relativamente sencilla. El eje del servomotor estará acoplado a una manivela circular por su centro, la cual se unirá a la biela en una articulación cerca del borde del círculo. El otro extremo de la biela tendrá un agujero por el que se introducirá (con cierta holgura para que pueda girar) una varilla incrustada en los agujeros de la pieza de la figura 17. Esta pieza irá pegada a la transpaleta por el otro extremo de forma que, al girar la manivela, la biela empuje la varilla y con ella esta pieza y la transpaleta. Para que el movimiento sea perfectamente lineal, la pieza de la figura 17 deberá desplazarse por una guía que impida el movimiento en los demás ejes. Esta guía tendrá un agujero lo suficientemente grande para que la biela no choque con ella en su movimiento. El diseño completo del mecanismo se puede ver en las figuras 18 y 19.

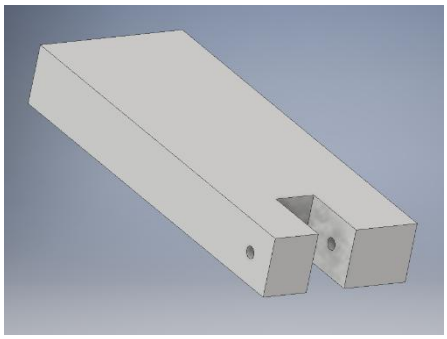


Figura 17. Émbolo del sistema biela-manivela

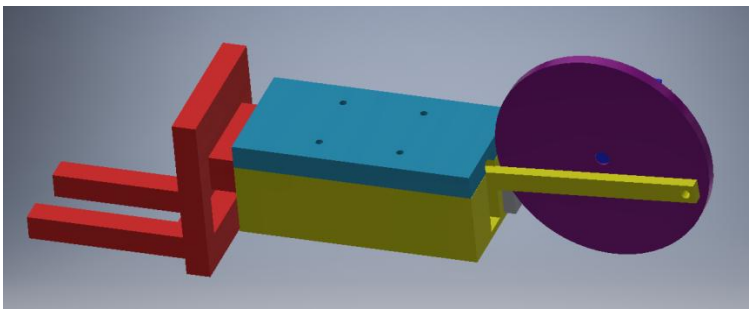


Figura 18. Diseño del sistema de carga/descarga 1

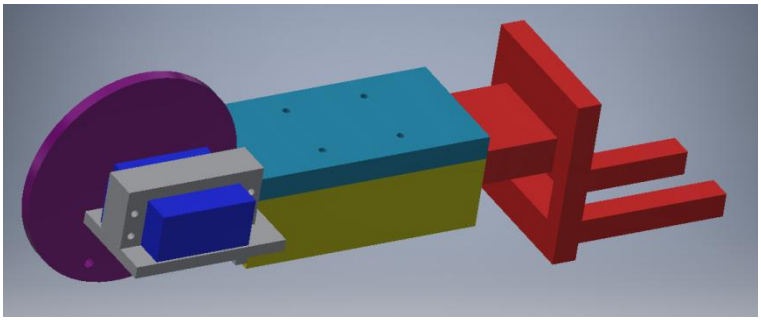


Figura 19. Diseño del sistema de carga/descarga 2

Pegada a la guía habrá una pieza sobre la que descansará el servomotor, que quedará fijado con una abrazadera pegada a la pieza de apoyo. Además, la guía (dividida en una parte de arriba y otra de abajo para facilitar la impresión y el montaje) cuenta con taladros arriba y abajo para poder unir el mecanismo al sistema AS/RS.

2.2.2 SERVOMOTOR

Un servomotor es un tipo de motor que sirve para posicionar el eje en una posición determinada y mantenerse en ella mientras no se indique lo contrario. A cambio, solo puede girar 180º (más o menos). Está formado por cuatro componentes principales: un motor de corriente continua, una reductora, un sensor de posición rotativo y un circuito electrónico de control.

Su funcionamiento es el siguiente: el sensor mide la posición en la que se encuentra el eje y se compara este valor con el de una señal de control que establece la posición deseada. La diferencia entre ambos es una señal de error que si no es nula es utilizada por el circuito electrónico para determinar el giro del motor de forma que se reduzca dicho error hasta llegar a la posición deseada.

La señal de entrada que determina la posición deseada del eje debe ser una señal PWM de unos 14-20ms (no es un valor crítico) de periodo. Lo que sí es crítico es el tiempo de pulso ya que es lo que determina la posición. Estos tiempos están bastante estandarizados (puede haber pequeñas diferencias) de forma que pulsos de 1ms corresponden a unos 0º mientras que pulsos de 2ms dan lugar a posicionarse de unos 180º. Tiempos intermedios llevan al motor a posiciones intermedias proporcionalmente.

Gracias a esta estandarización de modelos en cuanto a la equivalencia tiempo de pulso-posición, Arduino cuenta con una librería propia para el control de servos que es compatible con la mayoría de los servomotores del mercado. Con esta librería la programación es muy sencilla por lo que, con su uso, los alumnos pueden aprender a manejar estos actuadores rápidamente.

Para generar la señal PWM la librería emplea alguno de los contadores internos del Arduino automáticamente por lo que no es necesario que la salida a la que se conecte el servo tenga posibilidad de PWM. No obstante, hay que tener cuidado porque el uso de esta librería puede provocar que se deshabilite la opción de PWM de algunos pines al utilizar el contador correspondiente a ellos.

De entre los modelos posibles se ha escogido el servo SG5010 por su reducido precio y poco peso. Se alimenta a 5V por lo que puede compartir alimentación con el Arduino. Presenta un torque máximo de 5,5kgcm que es un valor mucho mayor del necesario considerando una carga máxima de 500g. Este hecho quedará patente más adelante en los cálculos del AS/RS. Su peso son solo 47g y en la figura 20 puede observarse su aspecto.

Figura 20. Servomotor SG5010

2.2.3 CONSTRUCCIÓN

Una vez impresas todas las piezas necesarias, se empieza por comprobar que el tamaño del palé en relación a la transpaleta es adecuado y que el sensor de ultrasonidos encaja correctamente en el agujero de ésta. Estos elementos se pueden observar en la figura 21. En la figura 22 se muestra una pieza con una tuerca atornillada en su interior. Esta tuerca forma parte del sistema AS/RS que se explica en el siguiente capítulo, pero se muestra aquí porque su camisa es el elemento por el que se une al sistema de carga/descarga como se puede apreciar en la figura 23 donde se encuentra unida a la pieza superior de la guía.

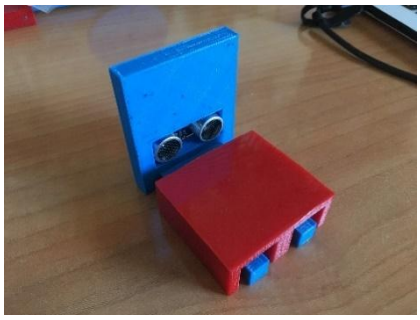


Figura 21. Transpaleta con palé

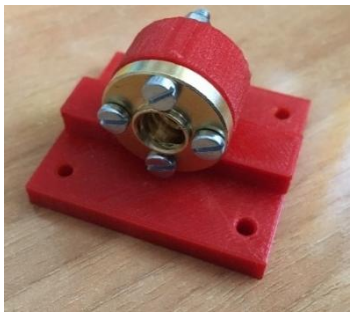


Figura 22. Tuerca dentro de su camisa

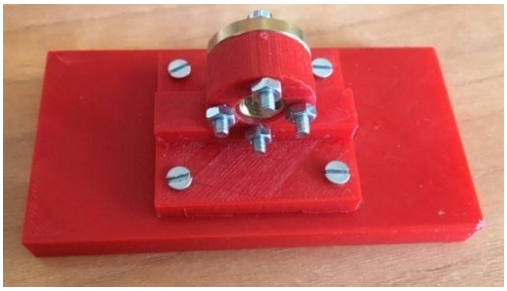


Figura 23. Camisa unida a la tapa de la guía

La parte inferior de la guía debe tener el tamaño apropiado en relación al émbolo para que éste pueda deslizar suavemente sobre ella pero sin tener movimiento más que en el eje que debe. Tal y como se han imprimido, el émbolo queda un poco prieto por lo que ha sido necesario lijar estas piezas hasta conseguir el resultado deseado. Tras ello se ha pegado a la guía la pieza soporte del servo tal y como aparece en la figura 24.

Dejando por un momento la guía, se introduce una varilla de acero de 3mm de diámetro y 40cm de largo por el agujero del émbolo de un extremo. Al sobresalir por el medio de la pieza, se introduce la biela en la varilla para a continuación seguir metiendo la varilla por el agujero del otro extremo del émbolo y así la biela quede atrapada sin poder salir. El agujero de la biela es 1mm más grande que la varilla de forma que pueda girar sin problemas. Uniendo el otro extremo de la biela a la manivela con un tornillo M3, una tuerca normal y otra autoblocante se consigue formar entre ellas una unión articulada dando como resultado el montaje de la figura 25.

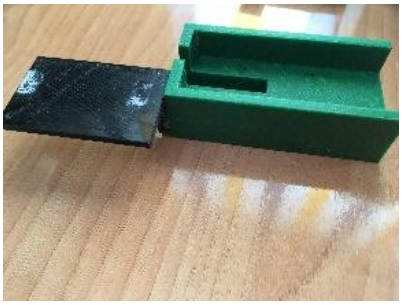


Figura 24. Guía con soporte del servo

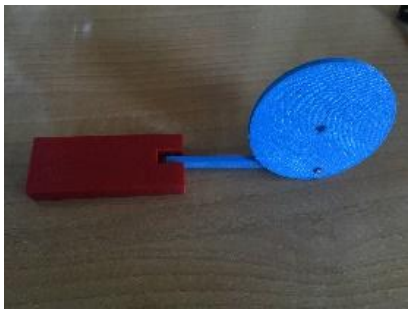


Figura 25. Elementos del sistema biela-manivela

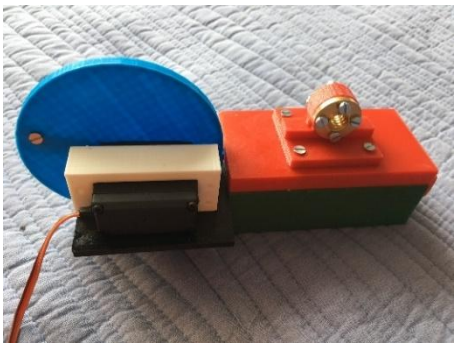


Figura 26. Biela manivela con servo

Se engancha el servo a la manivela de forma que su giro entre 30 y 180º cuadre con el movimiento del sistema biela-manivela para extender y retraer la transpaleta completamente. Una vez unido a la abrazadera, ésta se pega en su posición sobre el soporte del servo quedando como resultado la figura 26.

Pegando la transpaleta al émbolo ya se tiene el sistema casi completo que se muestra en la figura 27. Como paso final, se atornilla a la parte inferior de la guía una pieza similar a la camisa de la tuerca que como ya se verá en el sistema AS/RS tendrá en su interior una varilla actuando como guía de forma que el sistema de carga/descarga se mantenga siempre perfectamente horizontal. Esta pieza atornillada a la guía se presenta en la figura 28.

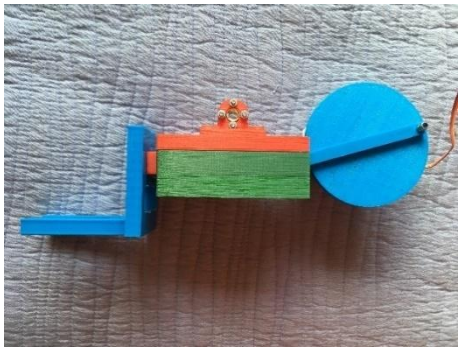


Figura 27. Sistema de carga/descarga

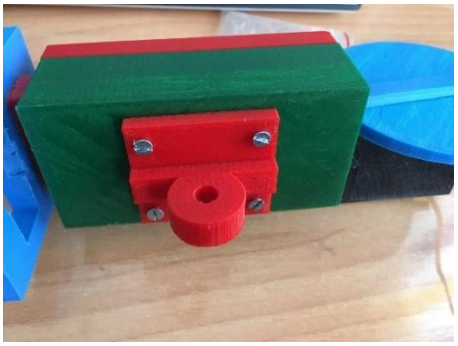


Figura 28. Pieza para guiar el sistema de carga/descarga

2.3 SISTEMA AS/RS

El sistema transelevador tiene por función el transporte del elemento de carga/descarga para situarlo en cada momento enfrente del cubículo que se requiera o de la cinta transportadora. Por tanto, un factor fundamental en el diseño de este sistema es que se llegue sin problema a los nueve cubículos y a la posición final de la cinta.

2.3.1 ELECCIÓN Y DISEÑO DEL MECANISMO

Lo más adecuado para este tipo de sistemas es optar por un sistema cartesiano de dos ejes que realice dos movimientos independientes (vertical y horizontal). El tipo de sistema necesario guarda bastante similitud con los que implementan las impresoras 3D o las máquinas CNC solo que en éstas hay movimiento en los tres ejes del espacio. Por ello, para el diseño de este mecanismo se ha investigado acerca de los elementos empleados en este tipo de máquinas y de hecho muchos de los componentes adquiridos son los mismos que se utilizan en algunos modelos de impresoras 3D.

El movimiento en cada eje se realiza en la mayoría de los casos gracias a motores paso a paso y un sistema tornillo-tuerca, aunque también es común el uso de correas para alguno de los ejes. El sistema tornillo-tuerca consiste en un tornillo o husillo en el que hay una tuerca roscada. Al girar el husillo por el movimiento del motor al que está acoplado, la tuerca y todo lo que esté unido a ella avanza por el husillo en un movimiento lineal. Este mecanismo, así como los motores paso a paso, se explican en la asignatura por lo que utilizándolos en el sistema los alumnos podrán ver una aplicación real de la teoría estudiada.

Figura 29. Husillo, tuerca, acople y rodamientos

Para una mayor estabilidad del sistema y para contar con mayor fuerza a la hora de vencer a la gravedad, se ha considerado adecuado utilizar dos motores para el movimiento vertical. Cada uno de ellos se sitúa en uno de los extremos del sistema transelevador elevando el eje horizontal por sus dos extremos. El eje de cada motor va unido a un acople flexible que permite la transmisión de movimiento desde el eje del motor (4mm de diámetro en el caso de los motores empleados) al husillo de 8mm al que se une por el otro extremo. La flexibilidad de este elemento permite que si el eje y el tornillo no se encuentran perfectamente alineados el movimiento se pueda transmitir igualmente sin problemas. En el extremo superior, el tornillo se sujeta con un rodamiento modelo KP08 que lo permite girar libremente, pero impidiendo que se tambalee. El acople, el husillo, la tuerca que se desplaza por él y el rodamiento se venden a menudo conjuntamente como en el pack que se muestra en la figura 29. No obstante, la mayoría de los acoples son para motores de ejes de 5mm de diámetro y el modelo utilizado tiene un eje de 4mm por lo que ha sido necesario adquirir acoples de dicha medida.

Los husillos se fabrican de longitudes múltiplos de 100cm de forma que la longitud del tornillo escogido es 300cm al tratarse de la medida más cercana superior a la distancia entre el punto más bajo (la cinta) y el más alto (el tercer estante) que deberá alcanzar el aparato de carga/descarga.

Para fijar tanto los motores como los rodamientos que sujetan los tornillos se ha diseñado la pieza de aluminio en forma de U de la figura 30 que por la parte de abajo se atornilla al soporte del motor mostrado en la figura 31 y por arriba a la pieza auxiliar de la figura 32 que hace de unión entre el rodamiento y la torre de aluminio. Esta pieza de aluminio se atornilla a la base mediante tirafondos por lo que una vez unida no es conveniente desatornillarla.

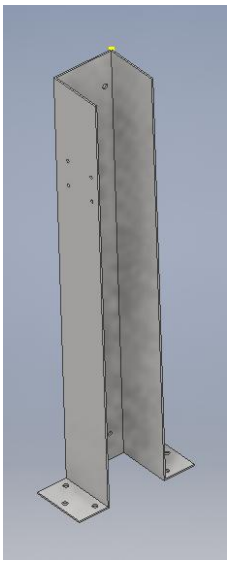


Figura 30. Torre de aluminio

Figura 31. Soporte motores verticales

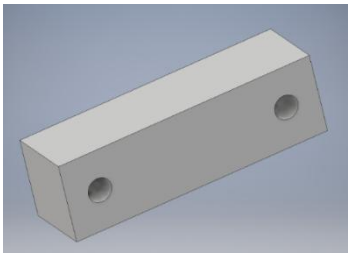


Figura 32. Unión rodamiento-torre de aluminio

Para poder atornillar el soporte del motor a la torre (ya unida a la base) no puede estar unido ya al motor porque sería imposible el acceso del destornillador. Por tanto, el motor debe colocarse posteriormente y para ello no es posible que quede apoyado sobre la base porque el eje impediría introducirlo en su posición de forma que deberá colocarse ligeramente elevado y apoyado sobre un calzo (elemento no indispensable si el motor queda perfectamente fijo y horizontal). Todo esto se ha tenido en cuenta en el diseño de las torres de aluminio, las cuales no son exactamente iguales, sino que presentan dos diferencias:

- En la de la derecha además de atornillarse con tres tirafondos a la base por los dos laterales de la U, también se hace por la parte

posterior para mayor sujeción. La de la izquierda no cuenta con esta característica ya que está situada al borde de la base así que esto obligaría a alargar la base varios centímetros más, cosa que no compensa pues realmente con seis tornillos la torre ya queda suficientemente fija.

- La torre izquierda cuenta con cuatro taladros en el lateral de la parte delantera de la maqueta destinados a colocar la pantalla de la que se hablará más adelante.

El diseño completo de una de las torres con su motor y demás elementos se plasma en la figura 33.

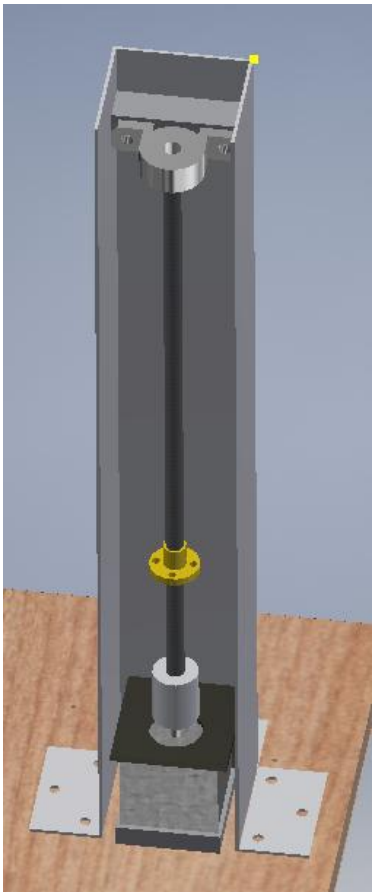


Figura 33. Eje vertical completo

Una vez determinado cómo va a ser el movimiento vertical se pasa a determinar el eje horizontal. En primer lugar, se ha diseñado la pieza de la figura 34 (ya mostrada anteriormente) que actúa como una camisa en la que se introduce y atornilla la tuerca que va por los husillos verticales. Con ella se tiene una plataforma que se mueve con la tuerca y a la que se pueden unir los elementos del eje horizontal. Esta pieza se utiliza tanto en el husillo vertical izquierdo como en el derecho.

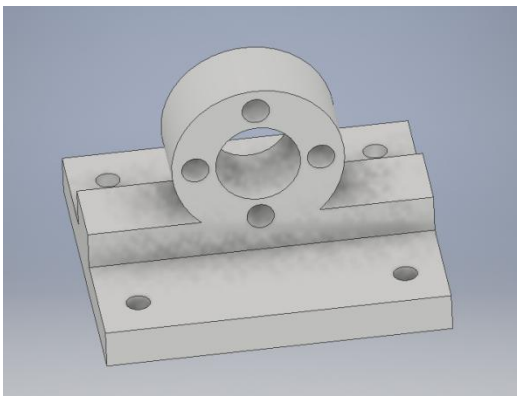


Figura 34. Camisa de las tuercas

En el eje horizontal el sistema es el mismo: un motor paso a paso hace girar un husillo por el que se mueve una tuerca roscada en él. En este caso, el husillo horizontal se apoya y gira sobre un rodamiento al principio del mismo y otro al final. Para construir este sistema hay que diseñar una pieza sobre la que se apoye el motor y un rodamiento, y otra sobre la que se sujete el rodamiento del otro extremo. Estas dos piezas se deben poder atornillar a las camisas de cada tuerca de los tornillos verticales.

Figura 35. Soporte del motor horizontal

La pieza de la figura 35 (para facilitar la impresión 3D se dividirá en dos piezas) es la diseñada para sujetar el motor horizontal y el rodamiento que se encuentra junto a él. Por detrás va atornillada a la camisa de la tuerca de la torre de la izquierda. El motor se une a ella mediante el soporte de la figura 36 que se atornilla a los dos agujeros de la parte central. Los dos agujeros de delante permiten fijar el rodamiento mientras que en el agujero de abajo se insertará un extremo de una varilla de 6mm de diámetro que servirá como guía para que el sistema de carga/descarga se mantenga horizontal. El pequeño saliente que se puede ver en la parte inferior a uno de los lados de dicho agujero será lo que accione el final de carrera vertical.

Figura 36. Soporte motor horizontal

La pieza de la figura 37 (también separada en dos para imprimir) tiene por fin sujetar el rodamiento del otro extremo donde acaba el tornillo. Va atornillada a la camisa de la tuerca del tornillo derecho y su agujero ciego inferior supone el final de la varilla guía.

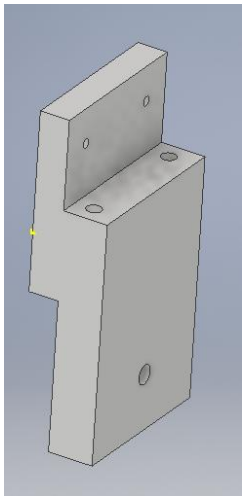


Figura 37. Soporte rodamiento

La longitud del tornillo horizontal, a diferencia de los verticales, es de 400mm pues es la medida estándar mínima necesaria para llegar desde los cubículos de la columna izquierda hasta la cinta situada a la derecha de la estantería.

El sistema AS/RS completo sobre la base con la estantería se muestra en la figura 38 incluyendo ya el sistema de carga/descarga.

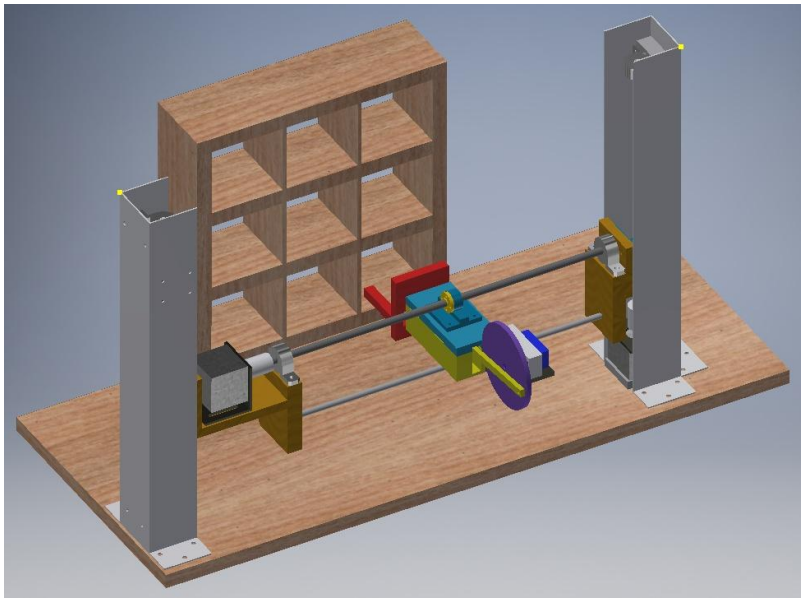


Figura 38. Diseño completo del sistema AS/RS

2.3.2 CÁLCULOS

Teniendo el mecanismo determinado y todas las piezas físicamente se ha procedido a hacer los cálculos para conocer el par necesario que deben poder suministrar los motores paso a paso. En primer lugar, se determinan los pesos de todos los elementos que intervienen en el sistema AS/RS. La suma de todos los componentes que forman el sistema de carga y descarga es de 266g. Este peso más el de la carga junto con el palé es el que debe mover el motor horizontal. Los motores verticales además de este peso tienen que poder con todo el eje horizontal que pesa 841g con la dificultad añadida de tener que vencer a la fuerza de la gravedad.

Algunos parámetros que se deben determinar para aplicar las fórmulas que permiten calcular el par son:

- Paso de los husillos (p): 2mm.
- Diámetro de los husillos (D): 8mm.
- Masa del husillo de 40cm (m_{40}): 124g.
- Masa del husillo de 30cm (m_{30}): 93g.
- Coeficiente de rozamiento entre tuerca (latón) y husillo (acero) (μ): 0,1.
- Velocidad máxima de desplazamiento (v): 20mm/s (en realidad no se alcanzará tanto).
- Tiempo de aceleración deseado(t_a): 0,2s.
- Carga máxima admisible: 500g.
- Rendimiento del sistema: se considera un valor del 90%.

Con la carga máxima escogida queda determinado que la masa máxima a mover por el motor horizontal es de 766g mientras que los verticales tendrán que levantar 1607g.

Para empezar, se calcula la velocidad angular requerida (n_{in}) en rpm con la ecuación 1 a partir de la velocidad lineal y el paso de los husillos.

$$n_{in} = \frac{60}{p} v = 600rpm$$

Ecuación 1. Velocidad angular requerida de los motores paso a paso

Por medio de la ecuación 2 se determina el torque de entrada necesario (M_{in}). F_L es la fuerza de la carga que hay que vencer. En el caso horizontal es solo la fuerza de rozamiento, pero en el vertical es además todo el peso del eje horizontal.

Horizontal: $M_{in} = \frac{p \cdot F_L}{2\pi\eta} = \frac{p \cdot \mu \cdot m_{carga/descarga} \cdot g}{2\pi\eta} = 0{,}27mNm$

Vertical: $M_{in} = \frac{p \cdot F_L}{2\pi\eta} = \frac{p \cdot (\mu \cdot m_{eje\ horizontal} \cdot g + m_{eje\ horizontal} \cdot g)}{2\pi\eta} = 6{,}13mNm$

Ecuación 2. Torque de entrada sistema transelevador

Además del par continuo hay que determinar el par de aceleración para lo cual hay que determinar el momento de inercia de los husillos (Js) mediante la ecuación 3 en la que r es el radio del husillo:

$$J_s = \frac{1}{2} m_{husillo} \cdot r^2$$

Ecuación 3. Momento de inercia de los husillos

El resultado para los verticales es de $7{,}44 \cdot 10^{-7}$ kg•m² mientras que para el horizontal es $9{,}92 \cdot 10^{-7}$ kg•m².

Un dato que en un principio se ha obviado al no disponer de él pero al determinar el motor se ha incorporado al cálculo, es el momento de inercia del propio motor que tiene un valor de: $J_{in}=5{,}4 \cdot 10^{-6}$ kg•m².

Ya se tienen todos los datos para aplicar la ecuación 4 con la que calcular el torque de aceleración ($M_{in,\alpha}$).

$$M_{in,\alpha} = \left(J_{in} + J_s + \frac{m_{carga} \cdot p^2}{\eta \cdot 4\pi^2} \right) \cdot \frac{\pi \cdot n_{in}}{30 t_a}$$

Ecuación 4. Torque de aceleración sistema transelevador

El resultado para el caso horizontal es 2,04mNm mientras que para el vertical (teniendo en cuenta los dos ejes) el resultado es 3,92mNm.

El torque máximo que va a tener que suministrar el motor horizontal es por tanto de 0,27+2,04=2,31mNm. Por su parte cada motor vertical tendrá que proporcionar un máximo de (6,13+3,92)/2=5,03mNm.

Este sistema cuenta con mucha menor complejidad y materiales más ligeros que las impresoras 3D o CNCs. Por ello, estos valores de par son mucho menores a los pares nominales de cualquiera de los motores paso a paso utilizados en sistemas de este tipo en los que se utilizan estos husillos. Por lo tanto, no hay problema en la elección del motor pues cualquiera de los presentes en el mercado para empleo con este tipo de mecanismos proporciona el par necesario. [16]

2.3.3 MOTORES PASO A PASO

Los motores paso a paso son actuadores electromecánicos que convierten pulsos eléctricos en desplazamientos rotativos incrementales por lo que presentan gran precisión y repetitividad de posicionamiento. Su velocidad angular está determinada por la frecuencia con la que se mandan los pulsos digitales.

Entre las especificaciones más habituales de un motor paso a paso están:

- La tensión nominal de funcionamiento,
- El torque máximo y la corriente que se consume al entregar dicho torque.
- El número de fases, generalmente 2.
- El número de pasos por vuelta, que depende de la reductora que presente ya que por sí mismo un motor de dos fases (valor habitual) solo daría cuatro pasos por vuelta.
- Resistencia e inductancia de los arrollamientos.

Los motores paso a paso pueden ser de dos tipos: unipolares y bipolares.

- **Unipolares:** poseen cuatro bobinas que tienen un extremo común (conexión a 5 hilos) como en la figura 39 o un extremo común dos a dos (conexión a 6 hilos). Dicho extremo común se conecta a alimentación o a tierra permanentemente. Para el movimiento del motor, con cada pulso que llega se energiza una de las bobinas y se deja de energizar la anterior siguiendo una determinada secuencia. De esta forma, el imán del rotor se reorienta y consiguientemente el motor avanza un paso. El recorrido de esa secuencia en una dirección u otra determina el sentido de giro del motor. Si en lugar de alimentar las bobinas de una en una se sigue una secuencia

como la de la figura 40, el motor avanza a medios pasos y la resolución por tanto se duplica.

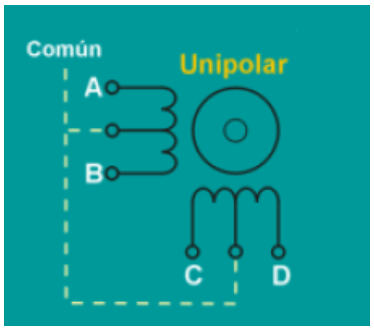


Figura 39. Esquema de un motor unipolar

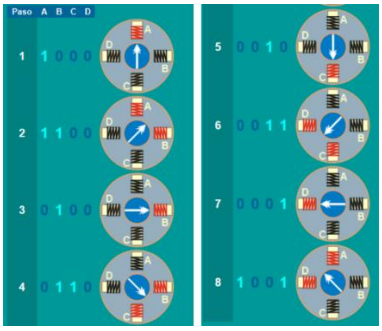


Figura 40. Secuencia de medios pasos

- **Bipolares:** presentan dos bobinas sin ningún extremo común como puede apreciarse en el esquema de la figura 41. La secuencia de pasos se consigue mediante cambios de polaridad en ellas. Si con cada pulso una de las bobinas cambia su polaridad (alternativamente) el rotor va girando dando pasos completos. Para conseguir el doble de resolución se puede pasar por un estado intermedio en el que la bobina que va a cambiar de polaridad se quede sin alimentación. De igual forma que los unipolares, el sentido de giro queda determinado por la dirección de la secuencia.

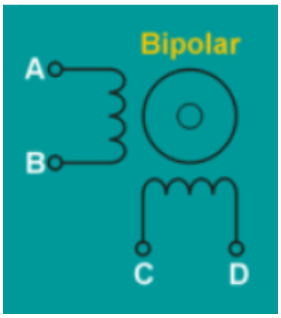


Figura 41. Esquema de un motor bipolar

Generalmente los motores bipolares son más pequeños, baratos y son capaces de proporcionar mayor torque por lo que son más comunes que los unipolares. Por contra, el control de los bipolares es más complicado y se requiere un controlador más complejo. [16-17]

Como se ha dicho, para elegir los motores paso a paso el par no es un problema por lo que se utilizan otros criterios como su tensión nominal o el precio. Los motores más extendidos para este tipo de sistemas son los de la familia Nema y los 42BYG, todos ellos bipolares. Algunos tienen tensiones nominales de 12V mientras que la de otros es de unos 4V. Puesto que 12V es junto con 5V el valor más estandarizado de fuentes de alimentación y además es la tensión nominal del motor de corriente continua (ver apartado 2.4) la mejor opción es utilizar motores de 12V para facilitar la alimentación y no necesitar elementos adicionales que bajen la tensión. Atendiendo a los precios de los motores de 12V se ha optado por el modelo 42BYG de Makeblock que es ligeramente más barato a otras opciones.

Este motor cuenta con dos fases y 200 pasos por vuelta al igual que la mayoría de los modelos consultados. Su torque nominal es de 40Ncm y la corriente que consume cuando suministra dicho par es de 1,7A (valor mucho mayor a la que se alcanzará en el proyecto). Al ser bipolar cuenta con cuatro cables que se corresponden a los dos extremos de las dos bobinas, rojo-azul una y verde-negro la otra. En la figura 42 puede apreciarse el aspecto de los motores utilizados.

Figura 42. Motor paso a paso 42BYG

2.3.4 CONSTRUCCIÓN

Eje horizontal

Para la construcción de este sistema en primer lugar se han imprimido las piezas diseñadas necesarias y se han adquirido los husillos (con el resto de elementos) y motores.

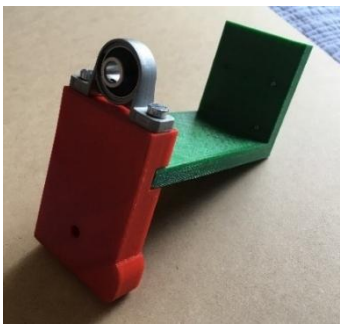


Figura 43. Soporte del motor horizontal

Contando con todos los elementos se comienza por introducir y atornillar las dos tuercas restantes de los husillos a sus camisas tal y como se hizo con la del sistema de carga y descarga.

Figura 44. Montaje del motor horizontal

Se unen las dos piezas que forman la pieza de la figura 35 y se atornilla el rodamiento en sus agujeros correspondientes tal y como se observa en la figura 43.

Siguiendo con dicha pieza se atornilla una de las camisas con tuerca por los cuatro agujeros de la parte posterior y por último se atornilla el soporte del motor horizontal tras lo cual se ha montado el propio motor.

Introduciendo el tornillo en el rodamiento y uniéndolo al motor con el acople se llega al montaje de la figura 44.

También se pegan las dos piezas que forman la de la figura 37, se unen a la camisa restante y se atornillan al otro rodamiento teniendo como resultado lo mostrado en la figura 45.

Figura 45. Rodamiento de la derecha en su soporte

Se introduce la tuerca del sistema de carga/descarga en el tornillo y en la varilla-guía. Por último, introduciendo el tornillo en el rodamiento de la figura 45 y la varilla (de 38cm) en los dos agujeros destinados a tal fin, ya se tiene el eje horizontal completo.

Ejes verticales

Lo primero para poder montar los ejes verticales ha sido proporcionar los planos de las torres de aluminio al trabajador que, cortando con una radial, doblando con un tornillo de banco y taladrando un tubo de 60x60 ha fabricado las dos piezas necesarias. El resultado se presenta en la figura 46 donde aparecen con dos tapas para una mejor estética.

Figura 46. Torres de aluminio

Teniendo las torres se atornillan los soportes de los motores a ellas (figura 47) y a continuación se introducen y atornillan los motores al soporte con los propios tornillos del soporte dando como resultado lo que aparece en la figura 48.

Se introducen las tuercas (ya unidas al eje horizontal) en los tornillos verticales y éstos se unen a los ejes de los motores mediante los acoples flexibles que se aprietan tanto a tornillo como a eje con una llave Allen.

Figura 47. Soporte de motor atornillado en la torre

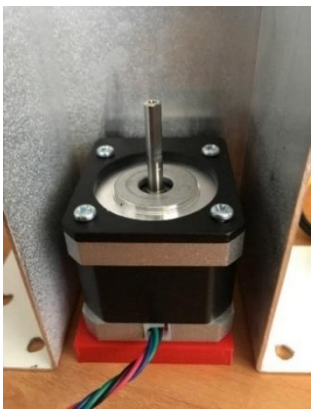


Figura 48. Motor vertical colocado en su posición

Como se aprecia en la figura 49, en la parte superior de las torres se atornillan los rodamientos junto con la pieza de la figura 32 de forma que el eje del rodamiento quede alineado con el del motor y el extremo del tornillo quede sujeto por el rodamiento. El sistema ya queda montado teniendo el resultado de la figura 50 y solo queda atornillar las torres a la base de madera en el lugar apropiado.

Figura 49. Rodamiento vertical en la torre

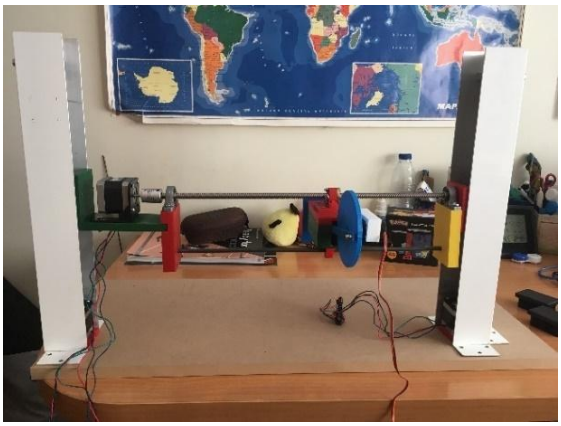


Figura 50. Sistema AS/RS completo

2.4 CINTA TRANSPORTADORA

La entrada y salida de palés del almacén se va a realizar, al igual que en muchos almacenes automatizados reales, a través de una cinta transportadora. Esta cinta va a ubicarse a la derecha de la estantería paralelamente a ella y su longitud no va a poder ser muy elevada ya que ello supondría aumentar el tamaño de la base de la maqueta haciéndola menos manejable.

En el proceso de sacar un palé del almacén, el sistema de carga/descarga lo dejará en el extremo de la cinta más cercano a la estantería. A continuación, el palé recorrerá la cinta accionada por un motor hasta el otro extremo de donde deberá ser retirado. En el caso de introducir un palé el proceso será análogo pero en sentido contrario.

2.4.1 DISEÑO

Para el diseño de la cinta en primer lugar se han establecido las medidas que va a tener para que sean acordes a las necesidades del sistema. Como los palés tienen un ancho de 55mm la cinta debe tener algo más así que se ha establecido una medida de 75mm. El largo efectivo (sin contar la parte que

rodea a los rodillos) va a ser de 20cm (y el total de 50cm), con lo que se consigue que haya sitio para ubicar el motor a la derecha de la torre de aluminio derecha sin alargar la base en demasía. El material con el que se va a hacer la cinta va a ser una plancha de goma EVA, un polímero termoplástico flexible, liviano y fácil de cortar.

La cinta avanzará por el giro de dos rodillos situados en sus extremos a la distancia necesaria para que la cinta se encuentre suficientemente tensa. El largo de los rodillos será de 82mm para ser ligeramente mayor al ancho de la cinta y se van a obtener de un tubo de fontanería de 26mm de diámetro interior y 32mm de exterior.

La pieza de la figura 51 se colocará en los extremos de los rodillos a modo de tapón. Por el agujero central se introducirá una varilla de 3mm de diámetro que recorrerá todo el rodillo y se apoyará por ambos extremos sobre el soporte de la cinta de forma que los rodillos estén en el aire y la cinta no toque el suelo. Además, la varilla del rodillo derecho irá acoplada al motor que transmitirá el movimiento.

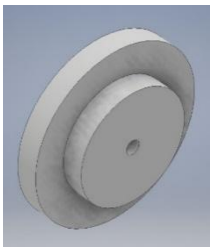


Figura 51. Tapón de los rodillos

Este soporte que se muestra en la figura 52 será lo que se una a la base de la maqueta permitiendo que la cinta quede en el aire. Además, permitirá que la parte superior de la cinta tenga una superficie de apoyo de forma que al poner un peso sobre ella no se tienda a hundir consiguiendo la estabilidad necesaria en el sistema. Se fabricará a partir del mismo tablero de madera con el que se hicieron los separadores verticales de los cubículos de la estantería.

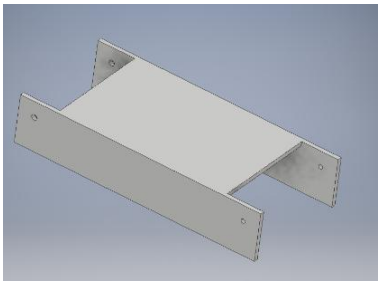


Figura 52. Soporte de la cinta

El sistema de la cinta completo con la propia cinta colocada sobre los rodillos y el soporte se presenta en la figura 53.

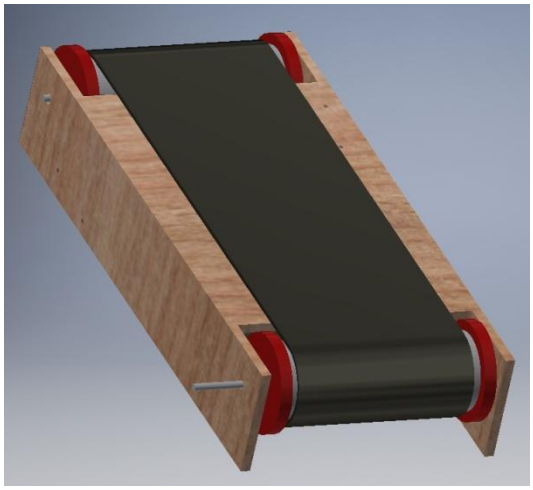


Figura 53. Diseño de la cinta

Por último, mediante una pieza cilíndrica con agujeros de los tamaños apropiados por cada extremo se consigue el acople de la varilla del rodillo de la derecha al motor que posibilitará el movimiento de la cinta. En la figura 54 se puede apreciar la posición de la cinta ya unida al motor en el diseño de la maqueta.

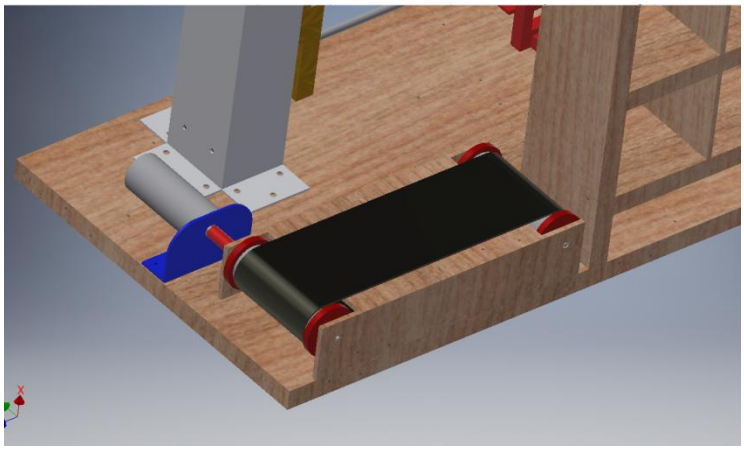


Figura 54. Ubicación de la cinta en la maqueta

2.4.2 CÁLCULOS

Con la cinta diseñada hay que determinar, al igual que con el sistema AS/RS, el par necesario que deberá poder suministrar el motor que la mueva. La carga máxima a mover es la misma que la supuesta para el transelevador por lo que sobre la cinta se moverá un peso máximo (m_{carga}) de 522g (500g de carga y 22g del palé).

Los datos necesarios de elementos del sistema para el cálculo del par son los siguientes:

- Velocidad de la carga sobre la cinta (v_L): se va a considerar una velocidad de 0,2m/s, lo cual equivale a recorrer la cinta entera en un segundo.
- Diámetro exterior de los rodillos (d_{ex}): 3,2cm (radio: r_{ex}).
- Diámetro interior de los rodillos (d_{in}): 2,6cm (radio: r_{in}).
- Masa de los rodillos (m_{rod}): 37g.
- Coeficiente de rozamiento (μ): 0,38.
- Tiempo de aceleración deseado (t_a): 0,2s.
- Rendimiento considerado (η): 90%.

Lo primero a determinar es la velocidad de rotación necesaria para cumplir con la velocidad lineal requerida. Con la ecuación 5 se determina que esta velocidad es de 119,37rpm de tal manera que el motor que se escoja deberá poder girar al menos a dicho valor.

$$n_{in} = \frac{60 v_L}{\pi \cdot d}$$

Ecuación 5. Velocidad angular requerida para el motor de la cinta

El torque de entrada (M_{in}) se calcula con la ecuación 6 donde F_L es la fuerza de la carga que hay que vencer, que será la fuerza de rozamiento de la cinta con la carga y los rodillos.

$$M_{in} = \frac{d \cdot F_L}{2\eta} = \frac{d\mu g(m_{carga} + 2m_{rodillo})}{2\eta} = 39{,}47 mNm$$

Ecuación 6. Torque de entrada de la cinta transportadora

Además del torque continuo hay que determinar el torque de aceleración para lo cual antes hay que obtener el valor del momento de inercia de los rodillos a través de la ecuación 7.

$$J = \frac{1}{2} m_{rod}({r_{ex}}^2 + {r_{in}}^2) = 7{,}86 \cdot 10^{-6} kgm^2$$

Ecuación 7. Momento de inercia de los rodillos

El torque de aceleración ($M_{in,\alpha}$) se obtiene de la ecuación 8 donde J_1 y J_2 son los momentos de inercia de los dos rodillos (mismo valor) mientras que d_1 y d_2 son los diámetros de los dos rodillos(también iguales).

$$M_{in,\alpha} = \left(J_1 + \frac{J_2 \cdot {d_1}^2}{\eta \cdot {d_2}^2} + \frac{m_{carga} \cdot {d_1}^2}{4\eta}\right)\frac{\pi \cdot n_{in}}{30 t_a} = 10{,}32 mNm$$

Ecuación 8. Torque de aceleración de la cinta transportadora

El torque máximo obtenido de la suma de los dos pares (continuo y aceleración) da como resultado 49,79mNm=0,508kgcm por lo que el motor para mover la cinta deberá superar dicho valor. [18]

2.4.3 MOTOR DE CORRIENTE CONTINUA

Se ha optado porque la cinta sea movida por un motor de corriente continua puesto son capaces de proporcionar grandes velocidades. Además, teniendo motores paso a paso en el sistema transelevador y un servomotor en el sistema de carga/descarga, incorporando un motor de corriente continua se tendrán en la maqueta los tres motores que se estudian en la asignatura y así los alumnos podrán practicar el manejo de todos ellos.

Un motor de corriente continua convierte energía eléctrica en energía mecánica de la forma que se explica a continuación de forma simplificada. En el estátor (parte fija) se crea un campo magnético por la acción de imanes permanentes o de un devanado de cobre sobre un núcleo de hierro. El rótor (parte móvil) contiene un bobinado de cobre por el que circula una corriente continua. Debido a la circulación de dicha corriente dentro del campo magnético del estátor, se produce una fuerza sobre el conductor según la ley de Lorentz que da lugar al giro del rótor.

Los parámetros más significativos para la elección de un motor de corriente continua son:

- Tensión nominal: aunque el motor puede funcionar a tensiones menores con peores prestaciones o a tensiones mayores reduciendo la vida útil, este es el voltaje recomendado para alimentar al motor.
- Corriente y par nominal: valores máximos que se pueden proporcionar de forma continuada sin dañar el motor.
- Corriente y par de bloqueo: valores máximos en los que el motor se queda bloqueado sin girar.
- Velocidad nominal al alimentar al motor a la tensión nominal.
- Potencia máxima: Producto máximo de la velocidad angular por el torque. Se produce a unas condiciones específicas de operación.

El motor escogido para el proyecto es el modelo EMG30 que se trata de un pequeño motor muy utilizado en aplicaciones de robótica. Cuenta con la gran ventaja de incorporar un codificador incremental interno de dos canales

gracias al cual será posible conocer la posición y velocidad del eje del motor en cada instante. Gracias a ello se podrá implementar un controlador PID para regular dicha posición o velocidad generando las señales adecuadas para enviar al controlador del motor del que se hablará en el capítulo de electrónica.

Un codificador de tipo incremental como el que posee el motor escogido se compone de:

- Un disco acoplado al eje del motor que cuenta con una serie de muescas equiespaciadas a lo largo de su perímetro.
- Un emisor de luz que puede ser visible o infrarroja a un lado del disco.
- Un fotorreceptor situado al otro lado del disco que puede ser un diodo o un transistor.

Con el giro del motor, el haz de luz entre el emisor y el receptor se va interrumpiendo con el paso de cada muesca del disco. De esta forma, contando el número de pulsos del fotorreceptor y conociendo el número de muescas del disco, se puede determinar el ángulo que ha girado el motor en un periodo de tiempo y por tanto su velocidad.

Un codificador incremental de dos canales posee dos conjuntos de estos elementos, encontrándose los discos desfasados 90º, lo que da lugar a pulsos de la forma de la figura 55. Gracias a ello se puede conocer el sentido de giro ya que, al producirse un flanco en un canal, el estado del otro es diferente según sea el sentido.

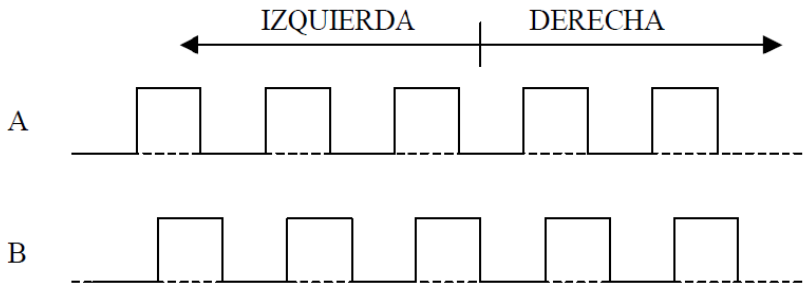


Figura 55. Pulsos de un codificador incremental de dos canales

Además de contar con el codificador incremental, el motor escogido que se presenta en la figura 56 cuenta con las siguientes características:

- Tensión nominal de 12V.

- Torque nominal de 1,5kgcm, valor unas tres veces mayor del necesario determinado en los cálculos.
- Corriente nominal de 530mA, valor no muy elevado por lo que es adecuado para el proyecto.
- Velocidad nominal de 170rpm, valor superior a las 119 requeridas.
- Número de pulsos del codificador incremental por vuelta: 360. [19]

El motor se sujetará a la base gracias al soporte de la figura 57 especialmente diseñado para este motor.

Figura 56. Motor CC EMG30

Figura 57. Soporte del motor EMG30

2.4.4 CONSTRUCCIÓN

La construcción de la cinta transportadora comienza con el corte de las piezas de madera necesarias a partir del tablero de 4mm de espesor del que se obtuvieron los separadores verticales de los cubículos. Las dimensiones de estas piezas, así como la posición de los taladros por los que irán las varillas, se puede consultar en el plano correspondiente al final del documento. Las piezas cortadas (figura 58) se unen adecuadamente gracias a unas eles de aluminio (figura 59) consiguiendo ya tener montado el soporte de la cinta (figura 60).

Figura 58. Piezas para el soporte de la cinta

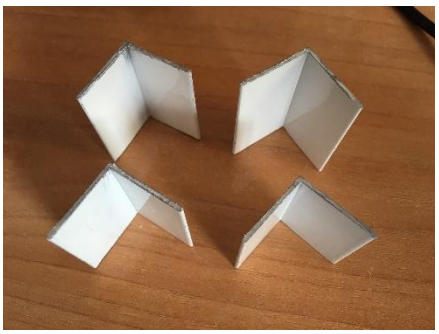


Figura 59. Eles de aluminio

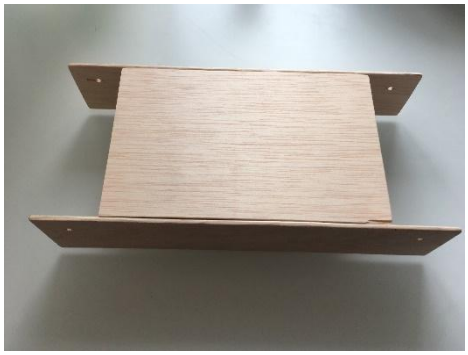


Figura 60. Soporte de la cinta

Se introducen los tapones cuyo diseño se mostró en la figura 51 en los extremos de los rodillos hechos con tubo de fontanería tal y como puede apreciarse en la figura 61. Se corta un rectángulo de goma EVA de 50x7,5cm y se pegan los extremos con celo ya que el espesor es tan fino que ha resultado imposible hacerlo con pegamento.

Se introducen las varillas de 3mm en el interior de los rodillos y se coloca uno de ellos en su posición en el soporte. A continuación, se introduce la cinta y por último se coloca el segundo rodillo introduciendo la varilla por los agujeros del soporte. Metiendo una de las varillas en el acople del motor se tiene ya la cinta completa a falta de unirla al motor (figura 62).

Figura 61. Rodillos de la cinta

En la figura 63 se puede observar la cinta ya acoplada al motor (colocado en su soporte) en su posición en la maqueta. Faltaría únicamente unir a la base el soporte de la cinta (con otro juego de eles de aluminio) y el soporte del motor (atornillando con tirafondos).

Figura 62. Cinta en la maqueta

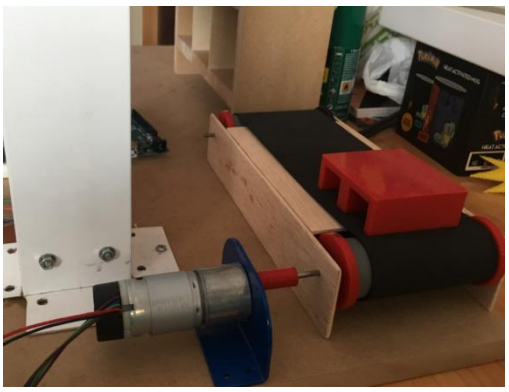


Figura 63. Cinta transportadora

3 DISEÑO ELECTRÓNICO

La mecánica anteriormente descrita realiza las acciones físicas del sistema, pero para ello tiene que estar controlada por una electrónica a su vez controlada en mayor o menor medida por el usuario. Un microcontrolador se encarga, gracias al programa almacenado en él, de ejecutar una serie de instrucciones dependientes de la interacción del usuario a través de una de las interfaces de control. Este dispositivo toma medidas de ciertos elementos como sensores o elementos manejados por el usuario, y en función de ellas y de su programación ordena realizar acciones a los motores (a través de sus controladores) y a otros elementos electrónicos complementarios.

La elección, funcionamiento y utilización en el proyecto de todos estos elementos se detalla a lo largo de este capítulo.

3.1 MICROCONTROLADOR

El microcontrolador utilizado para el proyecto ha de ser uno de los modelos Arduino, dado que son los que se explican y utilizan en la asignatura debido a sus numerosas ventajas en relación a otras opciones.

Arduino es una plataforma de electrónica de código abierto (open-source) basada en hardware y software libres y fáciles de utilizar gracias en parte al hecho de haber sido desarrollados conjuntamente. A lo largo de sus pocos años de vida (se inició en 2005 en Italia), Arduino ha experimentado un gran crecimiento y ha sido utilizado en numerosos proyectos de múltiples disciplinas desde objetos cotidianos hasta instrumentos científicos de lo más complejos. Su carácter abierto permite que usuarios de todo tipo (estudiantes, aficionados o profesionales) puedan contribuir haciendo accesibles a todo el mundo sus ideas, códigos y proyectos. Esto da lugar a que su evolución sea tan rápida y haya tantos recursos en la red que cada vez más gente se anime a unirse a esta plataforma por lo sencillo que resulta su aprendizaje. [20]

Hay muchas plataformas de microcontroladores que incorporan paquetes fáciles de usar para su programación pero Arduino, además de hacer esto, proporciona las siguientes ventajas:

- **Barato:** Las placas Arduino tienen un precio considerablemente menor al de la mayoría de otros microcontroladores, lo cual hace

más factible su utilización por estudiantes y aficionados, así como para educación en colegios y universidades.

- **Entorno de programación multiplataforma:** A diferencia de muchos de los sistemas de programación de otros microcontroladores que solo corren en Windows, el software de Arduino es multiplataforma pudiéndose utilizar en sistemas operativos Windows, Linux y Macintosh OSX.

- **Entorno de programación simple y claro:** el IDE de Arduino es sencillo de utilizar para principiantes. En él se programa en un lenguaje de programación muy similar a C con lo que usuarios con cierto conocimiento de este lenguaje no deben tener ningún problema para familiarizarse rápidamente con el lenguaje Arduino. Además, su flexibilidad permite que usuarios más avanzados puedan realizar modificaciones y ampliaciones a su gusto.

- **Código abierto y ampliable:** el software de Arduino se publica como *OpenSource* pudiendo ser ampliado y mejorado por cualquier programador. Arduino cuenta con librerías oficiales para el control de ciertos elementos, pero por la red hay muchas más que pueden ser utilizadas por cualquiera para facilitar la labor de programación.

- **Hardware *OpenSource* y ampliable:** al igual que el software, las placas de Arduino pueden ser construidas y modificadas por cualquier diseñador de circuitos electrónicos puesto que sus planos están publicados para que cualquiera tenga acceso a ellos.

Existen numerosas placas con diferentes características y precios, pero con muchos elementos comunes:

- Placa de circuito impreso.
- Microprocesador que en la mayoría de las placas es un AVR de Atmel.
- USB para alimentación o para subir programas desde el ordenador.
- Pines de alimentación, GND y entrada/salida tanto analógicos como digitales algunos de los cuales pueden utilizarse como PWM, interrupciones y comunicaciones serie o I2C.
- Botón de reinicio.
- Cristal oscilador.
- LEDs indicadores.
- Gran parte de las placas tienen otro conector de alimentación además del USB que junto con un regulador de tensión permite alimentar a 5V aunque la fuente sea mayor.

Se pueden distinguir tres grupos de placas Arduino en función de sus características:

- **Placas de nivel principiante:** son fáciles de utilizar y perfectas para la iniciación a Arduino. Cuentan con procesadores de menor velocidad que otras placas, menor capacidad de memoria y pocos pines de entrada/salida por lo que su precio es reducido. Los modelos más populares de este grupo son el Uno (placa más utilizada), Nano (pequeño tamaño para proyectos compactos) y Leonardo.

- **Placas con características mejoradas:** cuentan con mejores procesadores, más memoria y mayor cantidad pines para poder realizar proyectos más complejos. Las placas más utilizadas de este grupo son el Arduino Mega y el Due.

- **Placas Internet de las cosas:** estas placas añaden la funcionalidad extra de poner conectarse a internet por sí mismas sin necesidad de Shields u otros elementos. Son más caras que todas las anteriores y entre ellas destacan la YUN (conexión WIFI) y la ETHERNET.

En la figura 64 se muestra una tabla con todos los productos Arduino disponibles en el mercado.

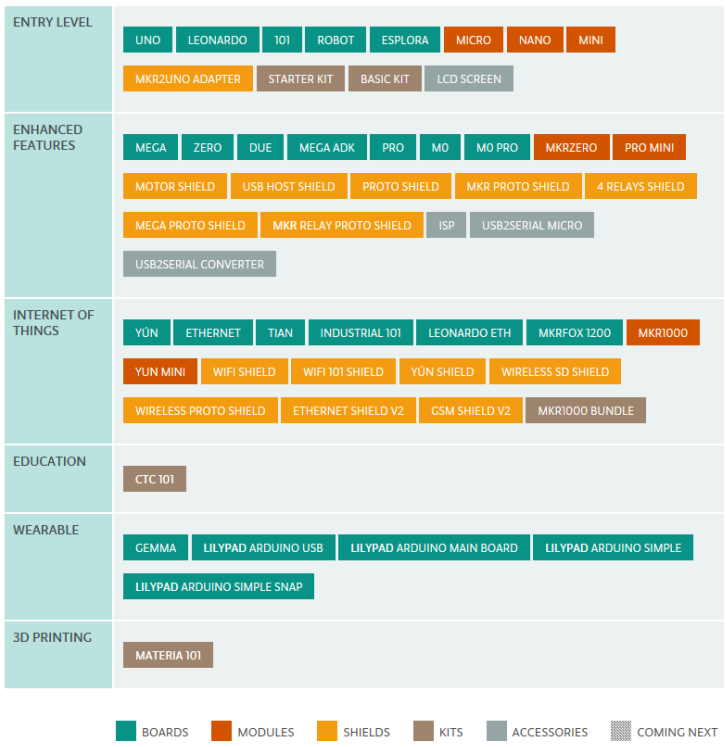


Figura 64. Productos Arduino por categorías

Los Arduinos cuentan con tres tipos de memorias de diferentes capacidades según la placa que desempeñan diferentes funciones:

- **Memoria Flash:** es una memoria no volátil (no se elimina su contenido al perder la alimentación) en la que se almacena el programa que se sube desde el ordenador para ser ejecutado. Una parte de ella está reservada para el *bootloader*, un pequeño programa previamente guardado que permite cargar programas sin hardware adicional. Este programa se ejecuta al encender o resetear el Arduino de forma que si llega un nuevo *sketch* por el puerto serie lo guarda en la memoria *flash* y en caso contrario da paso al *sketch* cargado previamente. Para cargar un nuevo programa siempre se hace un *autoreset* previamente para que se ejecute el *bootloader*. [22]

- **Memoria SRAM:** memoria volátil (se pierde el contenido al perder la corriente) de capacidad mucho más reducida que la anterior en la que se almacenan y modifican las variables del programa en ejecución. Debe tenerse cuidado de no utilizar muchas variables de tamaños grandes para no saturar esta memoria ya que en ese caso el programa no se ejecutará o lo hará de manera errónea.

- **Memoria EEPROM:** es una memoria no volátil de reducido tamaño en la que se puede almacenar información que se desea mantener a largo plazo como pueden ser variables de configuración que vayan a sufrir pocas modificaciones e interese mantener entre encendidos del Arduino para que el programa no tenga que cargarlas en la SRAM cada vez que se ejecute. No es conveniente hace un uso excesivo de esta memoria al menos en cuanto a escrituras puesto que tiene un máximo de ciclos de escritura tras el cual puede dejar de funcionar. [21]

Además, sobre los Arduinos pueden montarse diferentes *Shields* que son placas de expansión que dan nuevas funcionalidades al Arduino. Algunas de las más utilizadas proveen al Arduino de características como: conexión Ethernet, WIFI, bluetooth, controladores de motores, GPS o controladores de pantallas táctiles TFT.

Microcontrolador del proyecto: Arduino Mega

Para la elección del Arduino hay que tener en cuenta las necesidades del proyecto principalmente en cuanto a pines necesarios. A continuación, se hace un recuento de todas las entradas y salidas requeridas para todos los elementos que se detallan en los siguientes apartados:

- Controladores de motores paso a paso: 2 salidas (STP y DIR) x 2 (horizontal y vertical) = 4 salidas digitales.
- Controlador de motor CC: 3 salidas digitales (IN1, IN2 Y ENA) de los que uno debe tener posibilidad de PWM (ENA).
- Codificadores incrementales: 2 entradas digitales con interrupción.
- Servomotor: 1 salida digital.
- Finales de carrera: 1 x 2 (vertical y horizontal) = 2 entradas digitales con interrupción.
- Botones: 1 x 3 = 3 entradas digitales.
- Joystick: 1 entrada digital (botón) y 2 entradas analógicos (eje x e y).
- LCD: 6 salidas digitales (RS, E, DB4, DB5, DB6 y DB7).
- Sensores de ultrasonidos: 1 salida (TRIG) x 2 = 2 salidas digitales y 1 entrada (ECHO) x 2 = 2 entradas digitales.
- Zumbador: 1 salida digital.
- Bluetooth: 1 entrada y 1 salida digitales (RX Y TX) que se deben conectar a TX y RX del Arduino.

La suma total es de 29 entradas y salidas digitales (4 con interrupción y 1 con PWM) y 2 analógicos necesarios. De los grupos de placas anteriormente descritos hay que descartar el primero ya que el dispositivo con más pines digitales apenas cuenta con 20 y solo dos de ellos tienen posibilidad de interrupción. Al no utilizar conexión a internet tampoco tiene sentido escoger una placa del tercer grupo que encarecería el precio innecesariamente.

La elección se reduce por tanto a placas dentro del segundo grupo en el que las más habituales y fáciles de conseguir son el MEGA y el DUE. Ambas cuentan con similar número de pines (suficientes para el proyecto) y precio, pero se distinguen en tres cosas: la placa DUE cuenta con un procesador más rápido y mayor capacidad de memoria que el MEGA pero la capacidad y velocidad de este último son más que suficientes al no tratarse de un sistema crítico en cuanto a velocidad ni que vaya a contener programas excesivamente complejos. La otra diferencia es que el DUE se alimenta a 3,3V mientras que el MEGA lo hace a 5V como la mayoría de placas y elementos electrónicos que se van a conectar a él. Este hecho sí es diferenciador para la elección ya que, con un controlador de 3,3V se complicaría el sistema de alimentación y el diseño de la PCB.

El microcontrolador elegido para el proyecto es por tanto un Arduino MEGA 2560 (figura 65) que cuenta con las características que se muestran en la tabla 1.

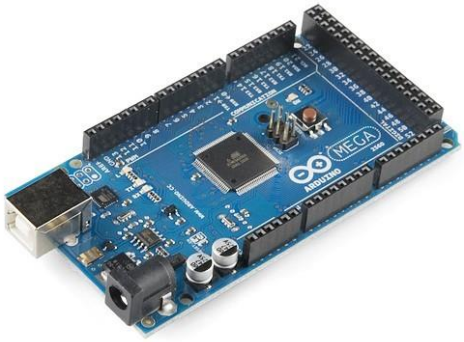


Figura 65. Arduino MEGA 2560

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Tabla 1. Características técnicas del Arduino MEGA 2560

Además de las características anteriores, en lo que respecta al proyecto cabe mencionar que cuenta con 6 pines con posibilidad de interrupción (se necesitan 4) y los pines se encuentran distribuidos a lo largo de tres de los laterales de la placa, lo cual facilitará el diseño de la PCB.

3.2 CONTROLADORES DE MOTORES

Los controladores de motores son circuitos integrados que sirven para gobernar el movimiento de motores eléctricos estableciendo su sentido y velocidad con instrucciones simples del microcontrolador y pocos pines de comunicación entre ambos. De forma simplificada, según las señales que envía el Arduino, el controlador permite o no el paso de corriente desde la fuente de alimentación de 12V hasta las bobinas del motor. Según el tipo de motor unos controladores son más adecuados que otros (aunque todos se basan en el uso de puentes H) de forma que para este proyecto se utiliza un modelo para el control de los motores paso a paso y otro para el motor de corriente continua.

3.2.1 MOTOR DE CORRIENTE CONTINUA

En robótica y proyectos de Arduino, los controladores de motores más extendidos para el control de motores de corriente continua de no demasiada potencia como es el EMG30 utilizado son los modelos L293D y L298N.

La estructura de ambos es básicamente un conjunto de dos puentes H gracias a los que se pueden controlar dos motores en ambos sentidos de giro. Además, en el L293D también se pueden controlar cuatro motores si es en un único sentido al tratarse de un montaje de medios puentes.

Los dos tienen una tensión de alimentación de su lógica interna de 5V y pueden alimentar motores de tensiones entre 4,5 y 36V el L293D, y entre 4,5 y 46V el L298N. Por tanto, en cuanto a tensión ambos cubren ampliamente lo solicitado por el motor escogido que tiene una tensión nominal de 12V.

Donde sí hay una diferencia apreciable es en la corriente. El L293D puede suministrar de manera continua una corriente de 600mA mientras que el L298N puede llegar hasta los 2A. El motor tiene una corriente nominal de 530mA por lo que el margen con el controlador más pequeño no es demasiado amplio. Probablemente no se llegue a la corriente nominal pues no se espera que el motor tenga que mover cargas muy elevadas, pero por si acaso se ha considerado mejor opción utilizar el L298N con el que no debería haber ningún problema de corriente. [23-24]

Además, el L298N se vende habitualmente integrado en una placa con radiador, conectores, diodos de protección y un regulador de tensión de 5V de gran capacidad de corriente gracias al cual solo con la alimentación del motor

se puede alimentar su circuito lógico e incluso el Arduino y el resto de componentes de 5V sin utilizar fuentes o reguladores adicionales.

El controlador adquirido ha sido por tanto un L298N (figura 66) montado sobre una placa que cuenta con un regulador de tensión de 5V LM2596S que es capaz de proporcionar una corriente de 3A con tensiones de entrada de hasta 40V a diferencia de otros módulos similares que incorporan reguladores que solo funcionan si la entrada es menor a 12V. [25]

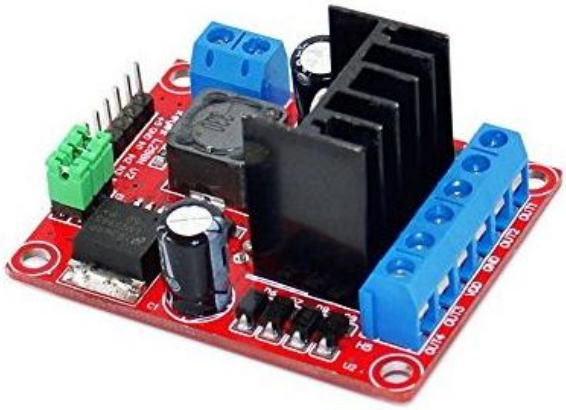


Figura 66. Placa del L298N

Como se ha comentado, estos controladores están basados en puentes H, que son circuitos como el mostrado en la figura 67 formados por cuatro transistores formando una "H" con el motor en el centro. La base de los transistores está conectada a los pines de control del integrado a través de unas puertas AND y NOT que dan lugar al siguiente funcionamiento:

En primer lugar, el pin ENABLE, a ser una de las entradas de todas las puertas AND debe estar activo para que haya posibilidad de tránsito de corriente por los transistores y por tanto de movimiento del motor.

Las otras dos entradas (DIR-L y DIR-R o IN1 e IN2) deben tener valores distintos para que el motor se mueva. En el caso de que la primera esté a 0 y la segunda a 1 conducen los transistores Q1 y Q4 dando lugar a que el motor gire en el sentido de las agujas del reloj (caso mostrado en la figura 67). En el caso contrario conducen Q2 y Q3 provocando un movimiento del motor en el sentido contrario a las agujas del reloj. En el caso de que ambas señales sean iguales, conducen Q2 y Q4 (1) o Q1 y Q3 (0) provocando que los dos terminales del motor se conecten entre sí y el motor se vaya frenando hasta pararse, periodo de tiempo durante el cual el motor actúa como generador de

electricidad. Lo que no puede suceder nunca (y de ello se encargan los inversores) es que conduzcan al mismo tiempo dos transistores de la misma rama ya que se produciría un cortocircuito y ambos transistores se destruirían.

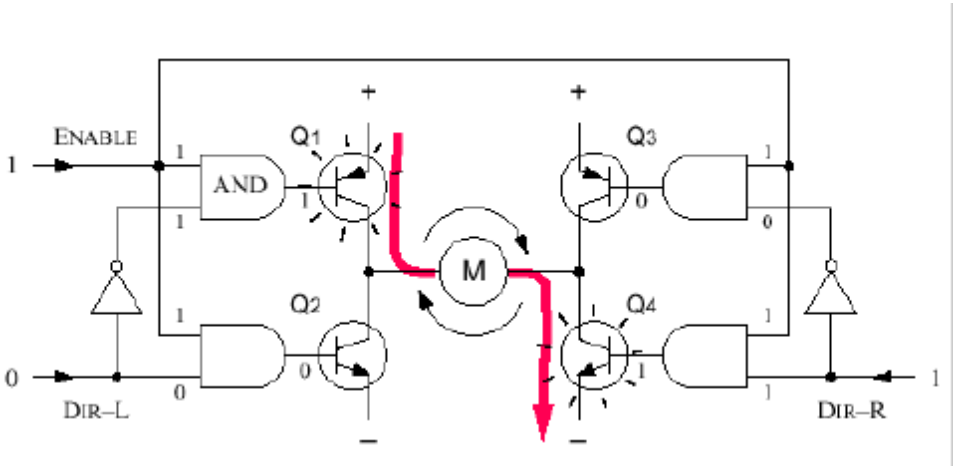


Figura 67. Circuito de un puente H

Conocido el funcionamiento del circuito, la forma más habitual de controlar la velocidad de un motor es utilizar una señal PWM en el pin ENABLE de tal forma que su ciclo de ocupación (relación entre el tiempo de conducción y el de corto) determine el ciclo de encendido del motor estableciendo con ello su velocidad. Con los pines IN1 e IN2 se controla el sentido de giro y la puesta en marcha o parada del motor.

El controlador tiene un conector con los nombres VDD y GND en el que debe conectarse la alimentación del motor (12V) y dos conectores para motores formados por los pares OUT1-OUT2 y OUT3-OUT4 por lo que habrá que conectar los terminales del motor en uno de ellos. Los pines ENA, IN1 e IN2 o ENB, IN3 e IN4 (según a donde se haya conectado el motor) se tendrán que conectar a tres pines digitales del Arduino teniendo que tener el primero de ellos posibilidad de PWM. Por último, si se desea alimentar el Arduino a través del regulador de tensión del módulo como va a ser el caso, el conector de nombres +5V y GND tendrá que conectarse a los pines homónimos del Arduino. Un apunte necesario es que las masas de ambas alimentaciones han de estar unidas para un correcto funcionamiento. [26-27]

3.2.2 MOTORES PASO A PASO

Como ya se ha explicado en el apartado de mecánica, los motores paso a paso que se van a utilizar para el sistema transelevador son motores bipolares de 12V y 1,7A de corriente nominal. Sin embargo, como se ha calculado, la fuerza que van a tener que hacer (y por tanto la corriente) será bastante inferior a la nominal. Con estos datos hay que escoger el controlador más adecuado de los que ofrece el mercado.

Descartando modelos como el ULN2003A (para motores unipolares) o el DRV8834 (para tensiones pequeñas 2,5-10,8V) se ha determinado que los dos controladores más populares para este tipo de motores son el A4988 y el DRV8834. Son muy similares entre sí pues tienen el mismo tamaño, prácticamente los mismos pines y una estructura interna muy similar pudiendo considerarse que el segundo es una versión más potente del primero. Estos controladores son los empleados para el control de los motores de la mayoría de las impresoras 3D y máquinas CNC presentes en el mercado y bajo ciertas condiciones son compatibles entre sí pudiendo utilizar indistintamente uno u otro.

Mientras que el A4988 puede llegar a proporcionar 2A (1A sin disipador) a tensiones máximas de 35V con 16 micropasos, el DRV8825 puede llegar a 2,5A (1,5 sin disipador), 45V y 32 micropasos. Estas mayores características no tienen gran relevancia para este proyecto ya que no se van a requerir esos niveles de tensión, corriente o micropasos. Por ello se ha optado por emplear controladores A4988 que aún son los más extendidos, suponen un menor coste y son más silenciosos. [28-29]

A4988

Se trata del controlador de motores paso a paso bipolares más extendido que únicamente requiere de dos entradas digitales para el control de un motor, una para el sentido de giro y otra para avanzar un paso. Cuenta con cinco resoluciones de micropasos, ajuste de la limitación de corriente y protecciones frente a altas temperaturas, sobrecorrientes, cortocircuitos y sobretensiones. Por lo general es un dispositivo robusto y fiable siempre que se conecte de manera adecuada (no tiene protección contra inversión de polaridad) y se incorpore un disipador en caso de ser necesario. Una precaución importante es que no se debe conectar y desconectar el motor o el controlador mientras están alimentados, pues el integrado podría sufrir daños.

El propio chip Allegro's A4988 se monta en una placa de pequeño tamaño desarrollada por Pololu que cuenta con 16 pines con las siguientes funciones:

- **ENABLE:** habilita las salidas a los motores cuando está a LOW y las desactiva a HIGH. Si no se conecta a nada se mantiene a LOW gracias a una resistencia pull-down.
- **MS1, MS2 y MS3:** determinan el modo de microstepping utilizado. Sin conectar están a nivel bajo y el controlador funciona con pasos completos.
- **RESET:** lleva al controlador a un estado inicial en el que desactiva las salidas si se encuentra a nivel bajo. Este pin no puede quedar al aire por lo que para no utilizarlo es habitual conectarlo al SLEEP que tiene una resistencia pull-up.
- **SLEEP:** a nivel bajo desactiva gran parte de la circuitería y las salidas. Esto permite minimizar el consumo de corriente cuando el motor no está en uso. A nivel alto (estado natural) el controlador funciona de forma normal.
- **STEP:** pin por el que se envían pulsos que corresponden con pasos del motor.
- **DIR:** pin que determina el sentido de giro del motor.
- **VDD y GND:** alimentación lógica del dispositivo que debe estar entre 3 y 5,5V.
- **1A y 1B:** salidas hacia una de las bobinas del motor.
- **2ª y 2B:** salidas hacia la otra bobina del motor.
- **VMOUT Y GND:** alimentación del motor que deberá estar entre 8 y 35V.

El montaje más simple que se puede hacer con el controlador y de hecho es el que se va a hacer en el proyecto es el mostrado en la figura 68. Como el consumo de corriente no va a ser muy elevado y los motores no tendrán un ciclo fijo de apagado-encendido, los pines ENABLE y SLEEP no se conectan a nada para que el controlador esté siempre funcionando a pleno rendimiento. Para no utilizar el RESET, éste se conecta a SLEEP consiguiendo así que permanezca a nivel alto.

Debido al paso de los tornillos (2mm) y el número de pasos por vuelta del motor (200) la resolución con pasos completos es de 0,01mm, valor más que suficiente para el proyecto, de forma que no se van a utilizar micropasos y por tanto MS1, MS2 y MS3 se dejan al aire.

STEP y DIR se conectan a dos pines digitales del Arduino siendo los mismos dos pines en el caso de los dos controladores correspondientes a los motores verticales de tal forma que siempre den los mismos pasos en la misma dirección. Los pines de alimentación lógica a la alimentación del propio microcontrolador. Las salidas del motor se conectan a las bobinas del motor teniendo cuidado de que los pines del mismo número sean los dos extremos de una misma bobina (rojo-azul a 1A-1B y verde-negro a 2A-2B o viceversa). Por último, los pines de alimentación del motor se conectan a la

fuente de 12V incorporando un condensador de desacoplo de 100µF para proteger al integrado frente a picos de tensión. [30]

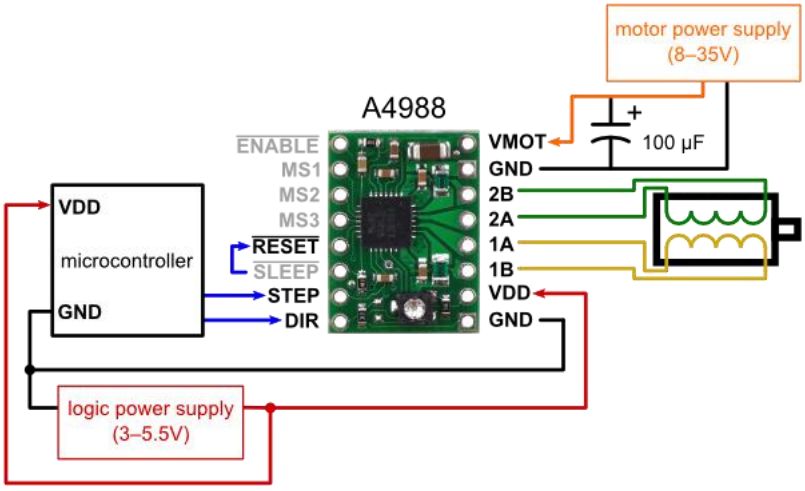


Figura 68. Esquema de conexiones del A4988

Como se puede intuir, este controlador es mucho más complejo que el L298N utilizado para el control del motor de continua, pero también se basa en la utilización de puentes H con los que se controla el paso y el sentido de la corriente por las bobinas. El esquema interno del integrado se presenta en la figura 69 destacando los dos puentes H que controlan las dos bobinas y unos bloques PWM que permiten las diferentes configuraciones de micropasos. Estos bloques funcionan permitiendo el paso de mayor o menor corriente por las bobinas de forma que al enviar un pulso por el STEP no se pase de no conducir a conducir el máximo de corriente (o viceversa), sino que solo se produzca un salto de corriente que dé lugar a un avance del motor de una fracción de paso o micropaso.

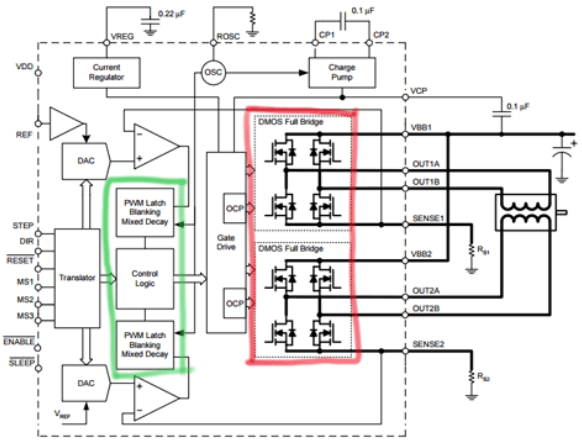


Figura 69. Esquema interno del A498

El tamaño de los saltos de corriente en las bobinas del motor y por tanto el recorrido angular de los micropasos depende del estado de los pines MS1, MS2 y MS3 siguiendo la tabla 2. No obstante, como se ha mencionado, no se hará uso de los micropasos para este proyecto.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Tabla 2. Configuración de pines A4988 para tipos de micropaso

En la parte superior izquierda de la figura 69 se puede apreciar un bloque llamado “regulador de corriente”, el cual sirve para limitar la corriente máxima que se va a entregar a las bobinas del motor. Este circuito es fundamental ya que alimentando un motor a su tensión nominal de funcionamiento, si se deja libertad en la corriente que circula por él, ésta será probablemente mayor a la nominal provocando daños el motor. De igual forma, como la resistencia de las bobinas es constante, para que la corriente sea la nominal habría que alimentar al motor a tensiones muy pequeñas con las que ni se movería.

Con el regulador de corriente del integrado se puede limitar el valor de corriente máxima al valor que se desee y poder por tanto alimentar al motor a su tensión nominal o incluso tensiones mayores sin que se produzca ningún daño. Puesto que la ley de Ohm debe cumplirse siempre y la resistencia de las bobinas no varía, la forma que tiene el regulador de limitar la corriente es proporcionando una señal pulsada PWM que interrumpe la señal. De esta forma, aunque la corriente en conducción supere la nominal, la corriente promedio no será mayor a la establecida. Este mecanismo de limitación de intensidad se denomina **Chopping**.

La intensidad nominal del 42BYG adquirido es de 1,7A a la cual entrega su par nominal de 40Ncm por lo que la corriente deberá calibrarse para no sobrepasar nunca ese valor. Realmente en el proyecto nunca se va a necesitar alcanzar dicho valor y se recomienda calibrar a una corriente algo menor a la nominal (alrededor del 70% de ésta), así que se va a tomar como intensidad máxima 1,2A.

Los A4988 tienen un pequeño potenciómetro que sirve precisamente para hacer esta regulación. Para ello hay que medir la tensión entre dicho potenciómetro (o el pin VREF) y GND, y mover el potenciómetro hasta obtener la tensión de referencia correspondiente a la corriente límite deseada. Esta tensión se calcula por medio de la ecuación 9.

$$V_{REF} = 8 \cdot I_{MAX} \cdot R_{CS}$$

Ecuación 9. Voltaje de referencia de un A4988

R_{cs} es una resistencia del integrado que en los modelos fabricados hasta enero de 2017 (como es el caso de los adquiridos) es de 50mΩ, mientras que en los modelos actuales se ha sustituido por una de valor 68mΩ. Para distinguirlas Pololu proporciona la imagen de la figura 70 en la descripción del componente.

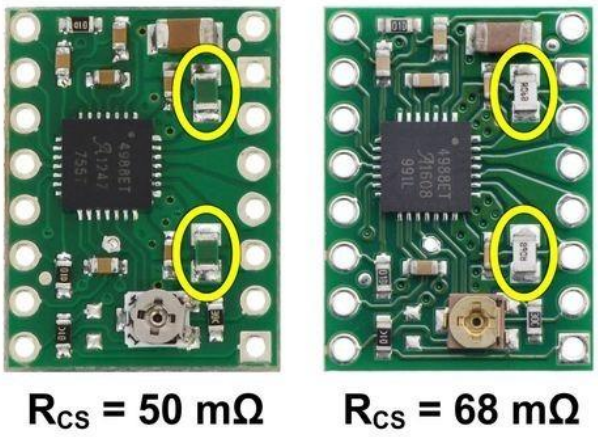


Figura 70. Identificación de Rcs en los A4988

La tensión que deberá marcar el potenciómetro es por tanto:

$$V_{REF} = 8 \cdot 1{,}2 \cdot 0{,}05 = 0{,}48V$$

En la figura 71 se muestra el ajuste de limitación de corriente realizado sobre los controladores adquiridos para el proyecto. Para ello se conecta el controlador a la alimentación de 5V dejando sin conectar los motores ni los 12V, se mide la tensión en el potenciómetro con un polímetro y se mueve la posición de éste hasta alcanzar un valor cercano a 0,48V. [31]

Cada uno de los tres controladores se montará sobre dos filas de ocho pines hembra en la PCB. Además, se incorpora un pequeño disipador encima del chip para su mejor refrigeración, aunque sin superar 1A de corriente no sería indispensable.

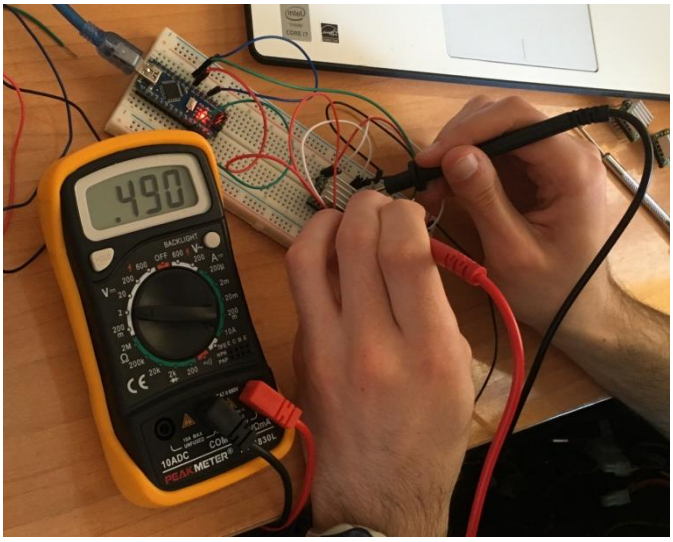


Figura 71. Limitación de corriente en un A4988

3.3 SENSORES

Los sensores necesarios para el almacén son dos sensores de distancia de pequeño tamaño con rangos desde casi 0 hasta unos 30cm. La toma de medidas con estos elementos no debe ser muy compleja de programar para que su uso sea una práctica adecuada para aprender a programar en Arduino por parte de los alumnos.

Uno de ellos se colocará en la transpaleta para reconocer si los diferentes cubículos del almacén se encuentran ocupados y así determinar si se puede o no introducir o sacar un objeto de ellos. El otro se ubicará al final de la cinta transportadora para detectar la llegada de palés al almacén y poder transportarlos por la cinta hasta la posición adecuada para que el transelevador los recoja.

La mayoría de los sensores de distancia utilizables en este tipo de proyectos se basan en la utilización de luz infrarroja como el SHARP GP2Y0A21YK0F o en los ultrasonidos como el HC-SR04. Habiendo trabajado con ambos se ha optado por incorporar el HC-SR04 (figura 72) por los siguientes motivos:

- Mejor funcionamiento comprobado experimentalmente con mediciones que se ajustan mejor a la realidad.
- Linealidad tensión-distancia mientras que en los sensores infrarrojos ambas magnitudes se relacionan siguiendo una curva. Por este motivo, para obtener la distancia hay que linealizar la curva

a tramos e incorporar dicha linealización al programa complicándolo sobremanera.

- Posibilidad de medición de distancia desde prácticamente cero mientras que el SHARP solo puede emplearse a partir de los 10cm Este valor es mucho mayor al necesario puesto que para ser recogido de la cinta, el palé tiene que quedar a unos 3cm del sensor o si no habría que modificar considerablemente la estructura del almacén. Por tanto, no es posible el uso de sensores SHARP.
- Menor coste.
- Menor consumo (15mA) por los 30mA del SHARP. [32-33]

El único inconveniente de los sensores de ultrasonidos respecto a los SHARP es que si el objeto que tienen delante no tiene una superficie plana paralela al sensor las medidas pueden ser incorrectas siendo motivo para descartar este tipo de sensores en muchos casos. Sin embargo, al igual que en los almacenes reales, las cargas que se van a utilizar tienen forma prismática y se colocarán en el palé de forma que queden paralelas al sensor por lo que este hecho no debería ser un problema.

Figura 72. Sensor de ultrasonidos HC-SR04

Este sensor cuenta con cuatro pines, dos de los cuales son para la alimentación de 5V y GND. Los otros dos corresponden al emisor y el receptor de ultrasonidos: el pin TRIG al ponerse a nivel alto da lugar a la emisión de la onda (con un pulso de unos microsegundos es suficiente) y el pin ECHO se pone a nivel alto cuando al sensor le llega la onda rebotada. Si con el Arduino se mide el tiempo desde el envío del pulso con TRIG hasta que es recibido por ECHO, conocida la velocidad del sonido se puede determinar la distancia a la que se encuentra el objeto en el que ha rebotado la onda.

3.4 CUADRO DE MANDOS

El cuadro de mandos es el nombre que se da a un conjunto de dispositivos físicos presentes en el sistema que pueden ser manejados por el usuario y que suponen una de las tres interfaces a través de las cuales se puede controlar el almacén. Con ellos se debe poder escoger en cada momento entre las opciones disponibles de los menús, manejar los motores en modo manual, ordenar acciones del modo automático y detener el movimiento de los motores en cualquier momento.

Pensando paralelamente en cómo se va a diseñar el código del Arduino, en la facilidad de manejo y en conseguir un buen aspecto visual, se ha decidido que dicho cuadro de mandos conste de tres pulsadores con embellecedores de diferentes colores (gracias a los cuales se podrá identificar cada botón en los menús del LCD) y un joystick.

Mientras el joystick servirá fundamentalmente para el control de los motores en el modo manual permitiendo su movimiento a diferentes velocidades, los botones se emplearán para el resto de las funciones necesarias mencionadas.

El esquema de conexión de los pulsadores con el Arduino es el mostrado en la figura 73. Una de las patillas se conecta a tierra y la otra a un pin digital activando mediante software la resistencia pull-up interna de dicho pin digital. De esta forma, cuando el pulsador no está accionado, la corriente va desde +5V hasta el pin a través de la resistencia y el pin lee '1'. Por el contrario, al pulsar el botón, la corriente va desde el pin a tierra por el pulsador y se lee '0'.

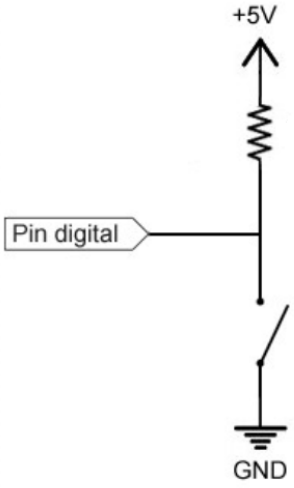


Figura 73. Esquema de conexión de un pulsador

En paralelo con cada pulsador se colocará un condensador de 100nF para filtrar los posibles rebotes producidos al presionar o soltar el botón. De esta forma se evita que se produzcan lecturas erróneas y el programa no se ejecute como debe.

Por su parte, el joystick cuenta con dos potenciómetros cuyo terminal intermedio se desplaza con el movimiento del joystick de tal manera que la resistencia de cada uno será proporcional a la posición del joystick en cada

uno de los dos ejes (vertical y horizontal). El joystick cuenta además con un pequeño pulsador que se conectará de igual forma que los demás.

Se pretende que el cuadro de mandos se encuentre lo más accesible posible al usuario de tal manera que estos elementos no podrán estar soldados en la placa general. Para poder fijar el joystick a la maqueta y a la vez conseguir una conexión sencilla del joystick a la placa, se ha adquirido una pequeña placa diseñada para montar este tipo de joysticks. Esta placa cuenta con agujeros en las esquinas para poder atornillar e interconecta las patillas del joystick de forma que de ella solo salgan cinco cables: VCC (tensión de 5V), VERT (tensión del potenciómetro del eje vertical), HORIZ (tensión del potenciómetro del eje horizontal), SEL (pulsador) y GND. Las patillas VERT y HORIZ se deben conectar a pines analógicos del Arduino para poder leer el nivel de tensión de los potenciómetros. SEL al igual que los demás pulsadores irá a un pin digital. El joystick junto a la placa en la que se va a soldar se muestran en la figura 74.

Figura 74. Joystick y placa para su montaje

Para un mejor aspecto visual, el joystick y los botones se colocarán en una pieza atornillada a la base cuyo diseño se muestra en la figura 75. Los botones quedarán incrustados en los agujeros de su tamaño y el joystick (atornillado a la base) sobresaldrá por el agujero grande. Por la parte posterior la pieza está abierta de forma que puedan salir los cables hacia la PCB.

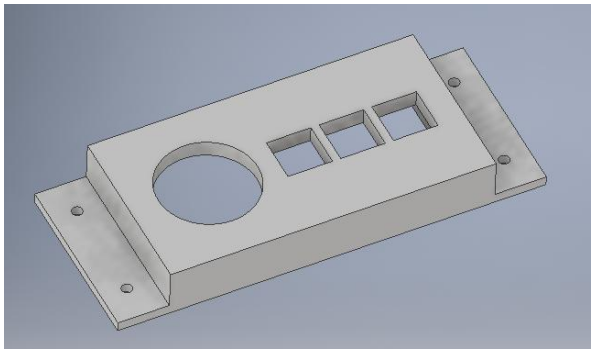


Figura 75. Pieza base para el cuadro de mandos

En la figura 76 se puede ver la pieza real con los elementos montados sobre ella.

Figura 76. Cuadro de mandos

3.5 ELEMENTOS SEÑALIZADORES

Para manejar el almacén desde el cuadro de mandos es necesario contar con algún elemento que muestre los menús y los diferentes mensajes para que el usuario pueda elegir y actuar en consecuencia por medio de los botones. El elemento escogido para este fin es una pantalla LCD de Sparkfun, dispositivo utilizado en las prácticas de la asignatura que se muestra en la figura 77. [34]

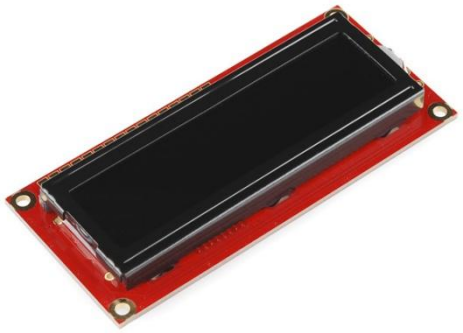


Figura 77. Pantalla LCD Sparkfun 16x2

La pantalla cuenta con dos líneas de 16 caracteres cada una y es compatible con la librería "LiquidCrystal" propia de Arduino que permite programarla de manera sencilla.

La función de cada uno de sus pines, así como su forma de conexión con el Arduino se detalla en la tabla 3.

El LCD se ubicará en la torre de aluminio izquierda a una altura de aproximadamente tres cuartas partes de la altura de la torre de forma que se pueda visualizar sin problemas. Para poder fijar la pantalla a la torre se ha diseñado y fabricado la pieza mostrada en la figura 78 que cuenta con ocho agujeros de 3mm. Cuatro de ellos sirven

para atornillar la pieza a la torre (en los taladros hechos para tal fin) mediante tornillos y tuercas M3, mientras que los otros cuatro coinciden en disposición y tamaño (también 3mm) con los agujeros de las esquinas del LCD de forma que ambos elementos se puedan unir con cuatro tornillos. El espacio entre la parte posterior de la pieza unida a la torre y el LCD permite sacar cables desde los pines de éste hacia abajo para ser llevados hasta la PCB.

Pin del LCD	Descripción	Conexión al Arduino
V_{SS}	GND	GND
V_{DD}	5V	5V
V0	Ajusta el contraste de la pantalla	Punto medio de un potenciómetro de 10k conectado por sus extremos a 5V y GND
RS	Controla los registros del LCD donde se leen o escriben datos	Pin digital
R/W	Lectura/Escritura	GND (solo se va a hacer escritura)
E	Habilitador de lectura o escritura	Pin digital
D0-D3	Bits que se leen o escriben	Al aire (en el modo de 4 bits que se va a utilizar estos bits no se usan)
D4-D7	Bits que se leen o escriben	4 pines digitales
A	Ánodo de los LEDs de retroiluminación	5V a través de una resistencia de 220Ω para limitar la corriente de los LEDs
K	Cátodo de los LEDs de retroiluminación	GND

Tabla 3. Pines del LCD y sus conexiones

En la figura 79 se muestra el LCD montado sobre la torre de aluminio.

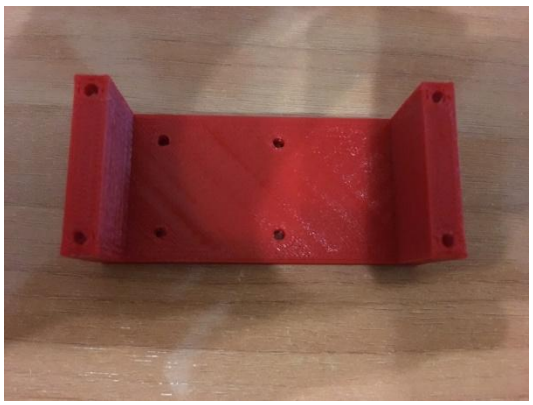


Figura 78. Soporte del LCD

Figura 79. LCD en el montaje

Además del LCD para mostrar mensajes, también se desea incorporar algún elemento para emitir señales acústicas que indiquen cosas como: pulsaciones de botones correctas, detección de objetos en la cinta, pulsación de finales de carrera o finales de acciones. Puesto que es un elemento ya empleado en las prácticas cuya programación es sencilla se ha optado por incorporar el zumbador piezoeléctrico de la figura 80 para este fin.

Figura 80. Zumbador piezoeléctrico

La piezoelectricidad es una propiedad de ciertos cristales que consiste en que al ser sometidos a una tensión mecánica aparece una diferencia de potencial en su superficie proporcional a dicha tensión. Este fenómeno sucede también al contrario, es decir, si se somete al cristal (habitualmente cuarzo) a un campo eléctrico o voltaje exterior se deforma proporcionalmente. Por este efecto, si se somete a una señal eléctrica variable como un PWM, el zumbador vibra a la frecuencia de la señal produciendo un sonido que variará en función de dicha frecuencia.

Para hacer funcionar el zumbador basta con conectar el positivo a un pin digital y el negativo a tierra. Si se quiere programar el PWM en sí, el pin debe ser uno de los que soporta dicha señal. Sin embargo, si se hace uso de la función “tone”, el Arduino utiliza uno de sus temporizadores para la oscilación de la señal pudiendo conectar el zumbador a cualquier pin digital. Como este elemento no hay necesidad de que esté especialmente accesible o visible se soldará directamente sobre la placa introduciendo un potenciómetro de 10k entre su patilla negativa y GND de tal manera que se pueda regular el voltaje aplicado al zumbador y con ello el volumen de sus sonidos.

3.6 FINALES DE CARRERA

El Arduino debe saber en cada momento la posición en la que se encuentra el transelevador y así poder enviar las órdenes correctas para moverlo a la posición que se requiera en el modo automático. Esto se consigue incorporando dos finales de carrera, uno para el movimiento vertical y otro para el horizontal. Gracias a ellos también se limitará el movimiento del transelevador en los ejes impidiendo que éste llegue a los extremos de los tornillos de avance pudiendo chocar con otros elementos.

Figura 81. Final de carrera

Al inicio del programa, se moverá el transelevador a la posición origen (activación de los dos finales de carrera) y a partir de ahí se irán contando los pasos que dan los motores de cada eje en todos los movimientos que se hagan. Al llegar a los finales de carrera o un número de pasos máximo correspondiente al otro extremo de los tornillos los motores se pararán para que en ningún caso se pueda pasar de ahí.

Los finales de carrera que se muestran en la figura 81 funcionan de la misma forma que los pulsadores (y se conectan igual) pero éstos deberán ir unidos a pines con posibilidad de interrupción ya que la parada de los motores al ser pulsados debe ser instantánea sin esperar a que se compruebe su estado en la ejecución normal del programa.

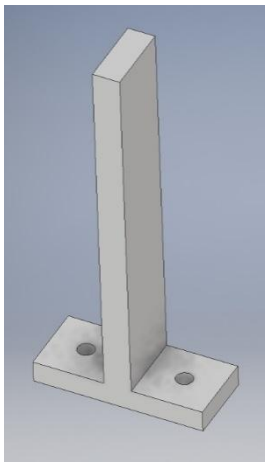


Figura 82. Soporte final de carrera vertical

El final de carrera horizontal se coloca pegado directamente en el lateral del soporte del rodamiento del extremo contrario al motor de tal forma que la guía del sistema biela-manivela lo accione al llegar a dicho extremo. En cambio, para el vertical ha sido necesario diseñar la pequeña pieza de la figura 82 que se atornilla a la base y en cuyo extremo superior va pegado el final de carrera. Esta pieza se ubicará en la posición exacta para que el final de carrera se accione por el saliente del soporte del motor horizontal a la altura correspondiente a la cinta transportadora (punto más bajo que va a ser necesario alcanzar).

3.7 BLUETOOTH

Por último, queda definir el elemento a través del cual se va a realizar la comunicación entre el Arduino y el móvil o tablet en el que se ejecute la aplicación Android. Las opciones para ello son la comunicación a través de WiFi utilizando por ejemplo un módulo WiFi ESP8266 o a través de bluetooth con un HC05.

La comunicación WiFi cuenta con la ventaja de poder comunicar ambos elementos remotamente sin importar la distancia entre ellos siempre que

ambos dispongan de conexión a Internet. Por el contrario, el bluetooth solo tiene un alcance de unos metros, pero realmente para este proyecto no es necesario más ya que no se va a requerir utilizar el almacén sin estar físicamente junto a él. Además, con el bluetooth no hay dependencia de disponer de un router para la conexión y la programación de dicha conexión tanto en Arduino como en Android es más sencilla que con WiFi.

El elemento utilizado va a ser por tanto un HC05, que se trata de un módulo bluetooth de bajo coste configurable por comandos AT que puede funcionar como maestro o como esclavo. Es importante diferenciarlo del HC-06, que se trata de un módulo muy similar, pero con menos funcionalidades (solo puede trabajar como esclavo), el cual podría utilizarse también para este proyecto pero se complicaría el procedimiento y podría dar problemas más fácilmente. Dado que los precios de uno y otro son similares no tiene sentido utilizar el HC-06. La diferencia más evidente entre ambos es que el HC-05 cuenta con seis patillas mientras que el HC-06 solo posee cuatro.

Con el módulo bluetooth se va a conseguir una funcionalidad adicional que es poder subir programas al Arduino por bluetooth desde el ordenador, lo cual resulta mucho más cómodo que conectando un cable USB entre ambos elementos. El procedimiento llevado a cabo para conseguir esto se detalla a continuación.

3.7.1 CARGA DE PROGRAMAS AL ARDUINO POR BLUETOOTH

COMPONENTES

Los elementos utilizados para conseguir subir programas al Arduino por conexión bluetooth además del módulo HC05 son los siguientes:

- Condensador de 100nF
- Transistor NPN 2N2222
- Resistencia de 10kΩ

Los valores del condensador y la resistencia no son algo crítico, sino que posiblemente con valores similares pueda funcionar sin problemas. De igual forma, el modelo del transistor podría ser cualquier otro NPN que soportase la tensión y corriente.

CONFIGURACIÓN DEL BLUETOOTH

En primer lugar, se ha configurado el módulo bluetooth para establecer su velocidad de transmisión (*baudrate*) para que sea igual a la utilizada por el bootloader del Arduino con el que se trabaja. Algunos de los Arduinos más comunes cuentan con *bootloaders* a las siguientes velocidades:

- Arduino Nano 328: 57600 baudios.
- Arduino Uno: 115200 baudios.
- Arduino Mega: 115200 baudios.

Además, puede configurarse el nombre del bluetooth, su pin de acceso o su modo maestro-esclavo.

Para la configuración se emplea el programa Arduino llamado "*config_bluetooth.ino*" (ver Anexo 2) que simplemente envía caracteres entre el bluetooth y el IDE de Arduino. Tanto la comunicación con el monitor del IDE como con el bluetooth son comunicaciones serie. La primera de ellas solo puede producirse a través de los pines 0 (RX: receptor de datos) y 1 (TX: transmisor de datos) del Arduino por lo que no pueden utilizarse dichos pines para la comunicación bluetooth. Utilizando un Arduino Mega como es el caso, la comunicación bluetooth puede realizarse a través de alguno de los otros tres pares de pines que soportan comunicación serie (14-15, 16-17 o 18-19) pero por si acaso se realizara la configuración con otro Arduino que no dispusiera de esta posibilidad se ha empleado la librería "*SoftwareSerial*" gracias a la cual pueden definirse dos pines digitales cualquiera como RX y TX de una nueva comunicación serie.

En la instrucción "*SoftwareSerial BT(10,11)*" el primer valor es el pin del Arduino que va a hacer como RX y el segundo el que actuará como TX. Por tanto, como los pines RX-TX bluetooth-Arduino deben ir cruzados, el RX del bluetooth debe conectarse al pin 11 del Arduino y el TX al pin 10. Los pines EN y STATE no hay que conectarlos.

Una vez subido el programa hay que entrar en el modo de configuración del bluetooth. Para ello hay que presionar el botón que se encuentra al lado del pin "EN" mientras el módulo no está alimentado y con él presionado conectar la alimentación. Si el procedimiento es satisfactorio el LED del módulo pasará de parpadear rápidamente a un parpadeo más lento.

En este momento el bluetooth debe poder configurarse mediante comandos AT de tal manera que enviando dichos comandos al bluetooth mediante el puerto serie del IDE de Arduino el bluetooth debe responder "OK" indicando su correcta configuración. Un detalle muy importante es que la velocidad de transmisión del módulo en el modo de configuración es de

38400 baudios independientemente de la que tenga en su modo normal. Por ello en el programa Arduino de configuración se establece dicha velocidad para la comunicación con el bluetooth.

Los comandos que pueden utilizarse son los siguientes:

- **AT:** si responde "OK" el bluetooth está listo para la configuración.
- **AT+NAME=(nombre):** establece el nombre con el que será visto por otros dispositivos. Por defecto es HC05 y se ha cambiado a "Almacen". Si se construyeran más maquetas habría que poner al bluetooth de cada una un nombre diferente
- **AT+PSWD=(contraseña):** establece la contraseña que deberá introducirse en los dispositivos que quieran emparejarse a él. Enviando AT+PSWD? el bluetooth responde con su contraseña actual que por defecto es 1234.
- **AT+ROLE=0:** indica al bluetooth que va a trabajar como esclavo. Este modo ya viene por defecto, pero no está de más enviar el comando para asegurarse. Si se quisiera que trabajara como maestro para por ejemplo comunicar dos HC05, se tendría que poner 1 en lugar de 0.
- **AT+UART=115200,0,0:** permite establecer la velocidad de transmisión que en este caso debe ser la del bootloader del Arduino Mega. Este es el único comando de configuración obligatorio para conseguir el objetivo de subir programas al Arduino por bluetooth.
- **AT+INIT:** al enviar este comando se sale del modo de configuración y el LED del módulo vuelve a parpadear rápidamente.

ESQUEMA DE CONEXIÓN

Una vez el módulo está configurado ya se puede montar el circuito final con todos los componentes. El HC05 irá montado sobre pines hembra mientras que el resto de elementos se soldarán directamente a la placa. El esquema de conexión se muestra en la figura 83. Los pines RX y TX del bluetooth en este caso tienen que conectarse a los pines 0 y 1 del Arduino, ya que es por ellos por los que se suben los programas al Arduino. Debe tenerse cuidado de no conectar el Arduino al ordenador cuando el bluetooth está encendido pues la comunicación no funcionará y pueden llegar a dañarse los componentes al estar dos elementos conectados a los mismos pines. Por tanto, cuando se realice esta conexión el Arduino no debe estar alimentado a través del ordenador.

El pin EN debe permanecer sin conexión, pero el STATE en este caso es fundamental ya que a través de él se reinicia el Arduino, paso previo necesario al envío del programa por RX-TX.

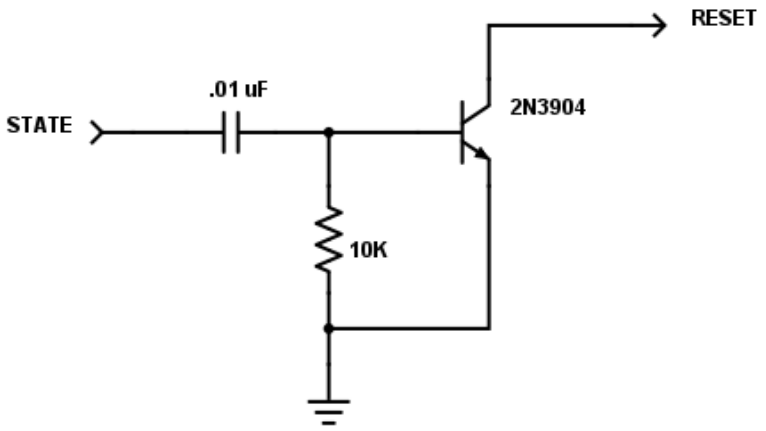


Figura 83. Circuito para subir programas por bluetooth

Al enviar la orden de subir el programa a través de bluetooth, se envía un pulso al pin State polarizando por tanto la unión base-emisor en directa. Con ello se consigue que el transistor conduzca y que el pin reset del arduino se conecte a tierra durante un tiempo determinado por la red RC formada por el condensador y la resistencia. De esta forma se resetea el Arduino y el programa puede empezar a cargarse. [35]

CARGA DEL PROGRAMA

Con las conexiones realizadas ya se puede proceder a subir un programa al Arduino vía bluetooth. En primer lugar, hay que emparejar el bluetooth con el ordenador para lo cual hay que introducir la contraseña que se haya configurado o, en caso de que no se haya modificado, la que viene por defecto (1234). Al quedar emparejado, el bluetooth disminuye la frecuencia de parpadeo de su LED.

En “Más opciones de bluetooth” se comprueba el puerto COM asignado al HC05 como saliente, se selecciona dicho puerto en el IDE de Arduino y se ordena subir el programa.

En caso de dar algún problema se puede probar a apagar y encender el bluetooth y en caso de no dar resultado desemparejar y volver a emparejar el bluetooth con el ordenador.

3.8 ALIMENTACIÓN

Teniendo ya todos los elementos que van a formar el circuito determinados hay que elegir la forma de alimentarlos teniendo en cuenta que es necesaria una tensión de 12V para los motores paso a paso y el de continua, y otra de 5V para el resto de los elementos.

Para la elección el aspecto más importante a tener en cuenta es el consumo de cada uno de los elementos de forma que se sepa la potencia necesaria que tendrá que suministrar la fuente. El consumo de los elementos de 5V (menos el servo) es un valor más o menos estable y se puede conocer fácilmente gracias a sus hojas de datos. Sin embargo, el consumo de corriente de los motores depende de la fuerza que tengan que hacer por lo que el valor aproximado se ha determinado experimentalmente.

Una vez construida la maqueta y con los elementos necesarios para el movimiento de los motores conectados en una placa de pruebas, se ha programado un código sencillo en el Arduino que permite mover los motores y poder determinar por tanto la corriente que consume cada uno. Para ello se ha hecho uso de una fuente de alimentación auxiliar de un ordenador y un polímetro.

En la tabla 4 aparecen detallados todos estos consumos, así como su suma total.

Componente	Consumo (mA)
Arduino Mega	95
Bluetooth	50
Zumbador	40
Sensor de ultrasonidos HC-SR04	15x2=30
LCD	25
Joystick	--
Botones	--
Finales de carrera	--
Servomotor	260
Total consumo 5V	500
Motores paso a paso	2400 (los tres motores juntos funcionando a la vez)
Motor CC	200
Total consumo 12V	2600

Tabla 4. Consumos de corriente de los elementos

El valor total de corriente necesaria sería por tanto de 3,1A aunque los ensayos de los motores no se han realizado con la máxima carga especificada de 500g por lo que este valor podría ser algo mayor. Además, es conveniente tener cierto margen de forma que se establece que la fuente deberá proveer unos 5-6A.

Como se remarcó en el apartado correspondiente, el controlador del motor de continua cuenta con un regulador de tensión que puede proporcionar 5V y 3A (valor mucho mayor de los 500mA de consumo de los elementos de 5V) por lo que al contar con este elemento para la alimentación de 5V no será necesaria más que una fuente de 12V que alimente los motores y dicho regulador.

Sabiendo la corriente que debe proveer la fuente, para la elección del modelo se han tenido en cuenta aspectos como el precio, el tamaño, el peso y la sencilla conexión a la PCB. La fuente escogida es la que aparece en la figura 84. Esta fuente de la marca "Salcar" proporciona un máximo de 6A (72W) y posee un cable lo suficientemente largo para que la conexión a la red no sea un problema y el almacén pueda ubicarse donde se desee. Su tamaño y peso reducidos la hacen muy manejable y a la salida cuenta con un conector jack macho. Para su conexión a la placa se ha adquirido el componente de la figura 85 que se trata de un conector jack hembra para soldar en PCB del mismo tamaño que el de la fuente. [36]

Figura 84. Conector jack PCB

Figura 85. Fuente de alimentación

Junto al conector jack de alimentación se colocará un interruptor de palanca como el de la figura x con el cual se podrá encender y apagar el dispositivo. Además, es necesario incorporar otro interruptor que permita o no el paso de los 5V de salida del regulador hacia el Arduino y resto de

elementos para que cuando se conecte el Arduino al ordenador para manejarlo con el puerto serie no haya dos alimentaciones de 5V que puedan provocar problemas. También es necesario que el HC05 no se alimente en caso de que se use el monitor serie porque habría dos elementos conectados a los mismos pines imposibilitando la comunicación. Para ello, este último interruptor contará con dos canales activados con la misma palanca de forma que si la alimentación entra por el regulador, el bluetooth estará alimentado y en caso de entrar por el ordenador no lo estará.

Figura 86. Interruptor de palanca

Estos dos interruptores se conectarán con cables pinchados a pines macho soldados a la placa y se colocarán en piezas como la de la figura 87 que se atornillarán a la base cerca del borde delantero para que los interruptores se encuentren accesibles al usuario.

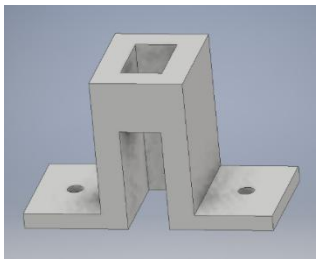


Figura 87. Soporte del interruptor

3.9 CONEXIONES Y DISEÑO DE LA PLACA DE CIRCUITO IMPRESO

Ya se han determinado todos los elementos electrónicos que van a formar parte del sistema y se conocen las conexiones a realizar de todos ellos con el Arduino y otros elementos auxiliares (condensadores, transistor, potenciómetros…). Ahora hay que proceder a ubicar todos estos elementos en una placa de circuito impreso, determinar la forma de conexión de todos ellos a la placa y en función de estas cosas escoger los pines concretos del Arduino a los que irán conectados los diferentes pines de cada componente.

En primer lugar, se ha escogido y adquirido una placa de circuito impreso virgen fotosensible sobre la que crear el circuito una vez esté diseñado. Las placas más comunes tienen unas medidas de 100x160mm pero debido a que el Arduino Mega ocupa bastante espacio no es un tamaño suficiente para poder ubicar todos los componentes así que se ha optado por una placa de 100x220mm con doble cara de revestimiento fotosensible. Gracias a esta característica se podrá rutar por los dos lados de la placa, hecho fundamental ya que la gran cantidad de componentes hace imposible el rutado a una sola cara.

El software utilizado para el diseño de la PCB es el Microsim 8, programa que permite diseñar circuitos electrónicos, simularlos y diseñar placas para ellos. El programa, proporcionado por el departamento de Tecnología Electrónica de la EII de la Uva, cuenta con una librería de elementos creada por dicho departamento que cuenta con gran cantidad de dispositivos para simular junto con los *footprints* de los encapsulados con los que se pueden encontrar dichos dispositivos en el mercado. Sin embargo, algunos de los elementos del proyecto como el Arduino Mega, el HC05 o los A4988 no se encuentran en dicha librería por lo que en primer lugar ha sido necesario crear símbolos y *footprints* de dichos elementos y guardarlos en una nueva librería. Para ello se han tomado todas las dimensiones de los elementos de sus respectivas hojas de datos para que la huella sea lo más fiel posible y a la hora de montarlo en la placa no haya ningún problema.

La forma de conexión de los elementos a la placa elegida ha sido la siguiente:

- Los elementos auxiliares como condensadores, resistencias, transistor y potenciómetros, así como el zumbador se van a soldar directamente a la placa al no necesitar que estén muy accesibles para el usuario. Solo los potenciómetros pueden ser manipulados, pero no será una acción muy habitual al servir únicamente para ajustar volumen del zumbador y contraste del LCD por lo que no hay problema en que se encuentren en la placa. Asimismo, el conector jack al que se conectará la fuente también se soldará en el extremo de la placa junto a la ubicación de la propia fuente.

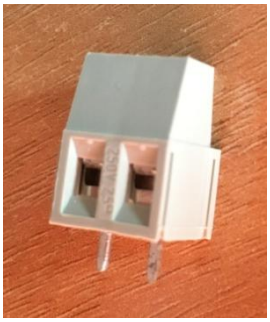


Figura 88. Borne para PCB

- Los cables procedentes de los motores y el controlador L298N se conectarán a través de bornes para PCB de dos pines como el que se muestra en la figura 88, los cuales se encontrarán necesariamente en los bordes de la placa lo más cerca posible de los elementos a los que van conectados.

- Los A4988 y el HC05 irán pinchados sobre pines hebra soldados a la placa mientras que el Arduino estará boca abajo pinchado sobre pines macho.

- Por último, el resto de elementos que deben estar separados de la PCB como el LCD, los ultrasonidos, el joystick, los botones o los interruptores se conectarán a través de cables a pines hembra (macho en el caso de los interruptores). Al ser muchos elementos no ha sido posible utilizar bornes para la conexión de todos ellos debido a su mayor tamaño y necesidad de ubicación en los extremos de la placa.

Haciendo un esbozo de la posición aproximada de cada elemento en la placa teniendo en cuenta su posición en la maqueta y el pin o tipo de pin al que debe ir conectado se ha determinado la asociación de pines del Arduino y los elementos que aparece en la tabla 5.

Pines de elementos	Pines del Arduino
VERT (Joystick)	A0
HORIZ (Joystick)	A1
SEL (Joystick)	A2 (un pin analógico puede actuar como digital y así están juntos todos los pines del mismo elemento)
DIR (A4988 verticales)	A3 (no tiene que ser analógico pero se elige por ubicación del componente en la placa)
STEP (A4988 verticales)	A4 (igual que el anterior)
TX (HC05)	0 (RX de la comunicación serie por la que se suben programas)
RX (HC05)	1 (TX de la comunicación serie por la que se suben programas)
Pulsador 3	2 (no se va a utilizar como interrupción en el programa pero este pin tiene esa posibilidad de forma que el pulsador puede ser empleado así si se desea en algún momento)
Pulsador 2	3 (igual que el anterior)
Pulsador 1	4
Señal del Servo	5
D7-D4 (LCD)	6-9
E (LCD)	10
RS	11
DIR (A4988 horizontal)	12
STEP (A4988 horizontal)	13
Final de carrera vertical	18 (pin para utilizar con interrupción)
Final de carrera vertical	19 (igual que el anterior)
Canal A del codificador incremental	20 (igual que el anterior)
Canal B del codificador incremental	21 (igual que el anterior)
TRIG (Ultrasonidos transelevador)	39
ECHO (Ultrasonidos transelevador)	41
Zumbador	43
ENA (L298N)	45 (pin con PWM)
IN1 (L298N)	47
IN2 (L298N)	49
TRIG (Ultrasonidos cinta)	51
ECHO (Ultrasonidos cinta)	53

Tabla 5. Asignación de pines

La sucesión de pines impares del final se debe a que en un borde del Arduino hay una doble fila de pines siendo los de valor impar los que se encuentran hacia el exterior. Se han asociado los pines de forma que no queden pines sueltos, sino que se agrupen en tiras lo más numerosas posibles.

En el esquemático del programa se realizan todas estas conexiones de pines, se unen las alimentaciones y tierras, y se montan todos los circuitos mencionados con anterioridad como el del bluetooth. El esquema completo se puede ver en el anexo "Planos" al final del documento. Previamente a empezar el diseño de la placa, se asigna a cada elemento el encapsulado correspondiente a los elementos físicos para que el footprint sea el correcto. Tras ello se crea la lista de redes (netlist) y se abre la aplicación de diseño de placas.

Una vez se delimitan las dimensiones de la PCB se procede a ubicar los footprints de los elementos en las posiciones que se consideran más adecuadas (figura 89) teniendo en cuenta la ubicación de los componentes y de la placa físicamente (en el centro de la maqueta a los pies del almacén). El Arduino se coloca más o menos en el centro de la placa para facilitar el rutado a todos los elementos, teniendo cuidado de que ningún elemento se ubique entre el puerto de conexión del cable USB y el borde de la placa de forma que se pueda conectar al ordenador y por tanto se pueda manejar por el monitor serie. El conector jack se coloca en el extremo superior izquierdo puesto que la fuente de alimentación se colocará en la izquierda de la maqueta. Como por el contrario el motor CC y por tanto el L298N estarán a la derecha y es necesario llevar los 12V hasta ellos, el borde trasero se reserva para las pistas que lleven esta alimentación de un lado a otro.

Como es de esperar, los conectores de todas las señales del motor CC y el L298N incluidas las alimentaciones de entrada (12V) y salida (5V) se encuentran en el borde derecho de la placa. Los A4988 de los motores verticales deben estar cerca ya que comparten los pines STEP y DIR además de la alimentación de 12V por lo que, aunque cada motor se ubica en un lado ambos controladores se colocan a la izquierda cerca de la alimentación de 12V. El controlador del motor horizontal se coloca también cerca de ellos para no tener que mover demasiado los 12V por la placa y por la ubicación a la izquierda del motor horizontal. Los conectores de cada motor están junto a su controlador correspondiente.

Los pines del joystick se colocan cerca de los pines analógicos y el resto de pines de elementos de fuera de la placa como el LCD, los botones o los finales de carrera (también los conectores del servo) se emplazan en el borde delantero para facilitar la conexión de los cables.

Los pines del ultrasonidos de la cinta se colocan en la parte trasera derecha ya que el elemento se ubica cerca de esa posición. El resto de elementos la mayoría de los cuales van soldados, se colocan en las posiciones más favorables de cara al rutado de la placa.

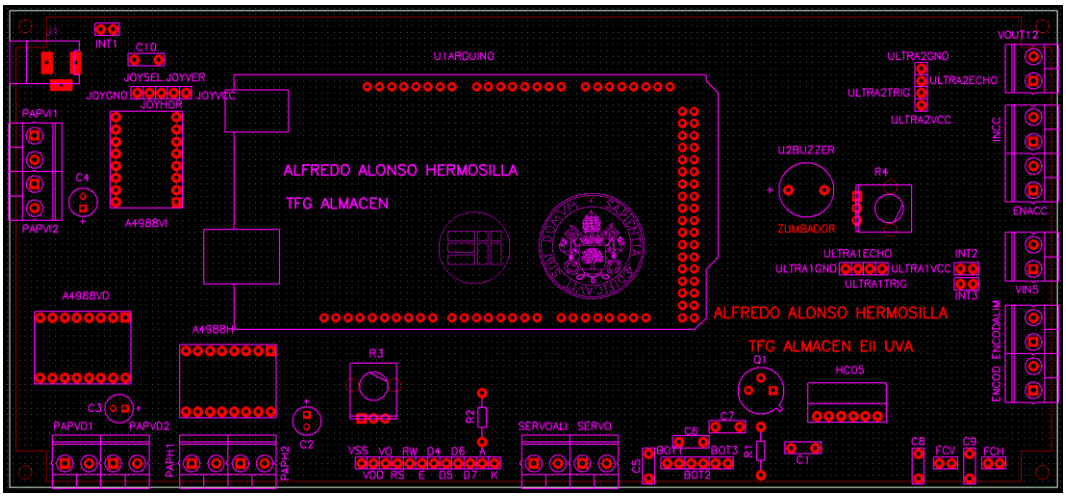


Figura 89. Ubicación de los componentes en la PCB

Con los componentes colocados se procede a rutar la placa con un ancho de pista acorde a las necesidades y siguiendo las normas básicas de rutado como son:

- Separación mínima de 0,3mm entre pistas para circuitos de 5-10V, lo cual se cumple ampliamente.
- Evitar los ángulos agudos y rectos en los cambios de dirección de las pistas.
- Distancia de separación lo más uniforme posible entre pistas que discurran paralelas.
- Pistas radiales a los pads, no tangenciales.
- Concurrencia de no más de cuatro pistas en un pad. [37]

Para calcular el ancho de pista necesario hace falta conocer el valor de tres parámetros que son:

- El grosor de la capa de cobre sobre la placa que en la mayoría de placas y concretamente en la utilizada es de 35µm (1 onza por pie cuadrado).
- El máximo incremento de temperatura permitida en la pista sobre la temperatura ambiente que se va a tomar como 20°C.

- La corriente máxima que va a poder pasar por la pista que en el peor de los casos (pista de alimentación de 12V) serán 3,5A (3,1 determinados más un margen).

La fórmula con la que se calcula el ancho necesario es la de la ecuación x. L es el grosor de las pistas en onzas por pie cuadrado y el área expresada en mils al cuadrado se obtiene mediante la ecuación x donde I es la máxima corriente en amperios, ΔT el incremento de temperatura admisible en grados centígrados y k1, k2 y k3 coeficientes que varían en función de si la pista es interna o externa. Como en este caso no hay capas internas, todas las pistas son externas y por tanto los coeficientes son k1=0,0647, k2=0,4281 y k3=0,6732. [38]

$$Ancho = \frac{Area}{L * 1{,}378}$$

Ecuación 10. Ancho de pista de PCB

$$Area = \left[\frac{I}{k1 * \Delta T^{k2}}\right]^{1/k3}$$

Ecuación 11. Área de pista de PCB

Aplicando dichas ecuaciones el área da un valor de 55,87mils^2 y el ancho mínimo necesario sale de 40,54mils. Para el rutado se escoge un valor algo mayor del teórico como es 50mils. El resultado del rutado es el mostrado en la figura 90 donde las pistas de la cara de soldadura se representan en azul y las de la cara de componentes en rojo. Se ha intentado utilizar las máximas pistas posibles en la cara de soldadura de tal forma que la tarea de soldar los componentes sea lo más sencilla posible.

En las cuatro esquinas se han puesto agujeros para poder atornillar a la base y en una parte vacía de la cara de componentes se ha añadido un texto para identificar la placa.

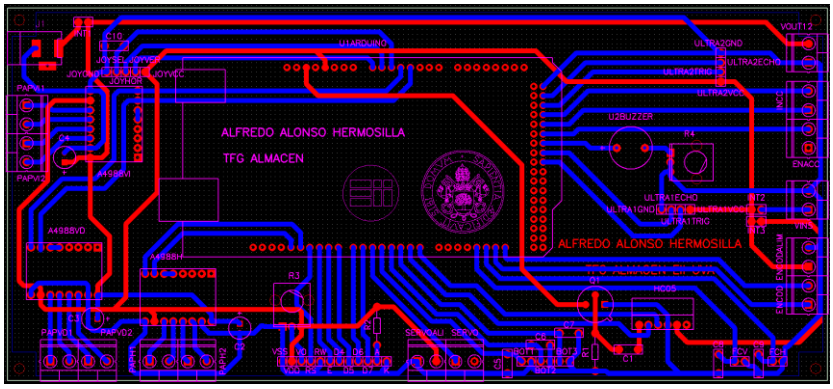


Figura 90. Rutado de la PCB

Por último, se ha creado un plano de masa por la cara de soldadura y otro de alimentación de 5V por la cara de componentes de tal forma que se necesite menos ácido al tener que comer menos cobre y se mejore la respuesta al ruido. Para ello, en el diseño de las pistas previo se ha tenido cuidado de poner todas las pistas de GND en la cara de soldadura y todas las de 5V en la de componentes.

Con la placa diseñada se generan los ficheros gerber, de taladrado y los planos de los fotolitos necesarios para la fabricación. En el anexo de planos se pueden ver los fotolitos de serigrafía, cara de soldadura y cara de componentes.

3.10 MONTAJE

La fabricación de la placa diseñada se ha realizado en el centro tecnológico CARTIFF con el procedimiento típico de taladrado, insolado, revelado y atacado con ácido. El resultado se muestra en la figura 91 (cara de componentes).

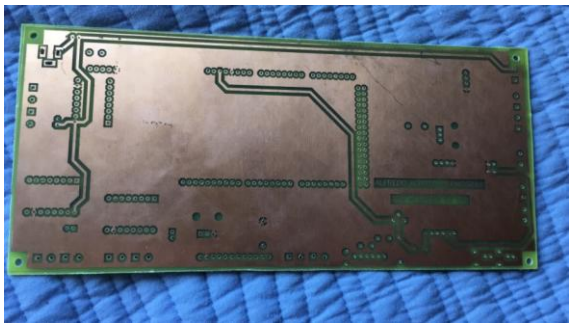


Figura 91. Cara de componentes de la PCB

Con un soldador y estaño se han soldado todos los conectores, pines y elementos que se estableció que iban en la propia placa obteniendo el resultado de la figura 92.

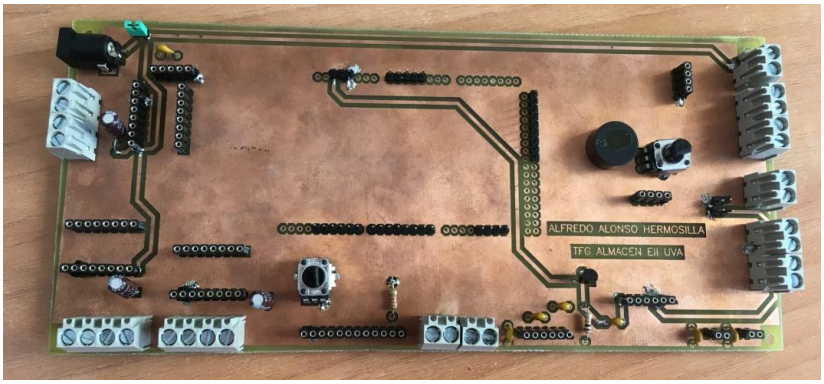


Figura 92. PCB con los elementos soldados

Por último, se pinchan el Arduino, los A4988 y el HC05, y se conectan los cables de cada uno de los elementos a sus conectores y pines ordenándolos de la mejor manera posible empleando canaletas de plástico y cinta adhesiva de doble cara. Los cables del sensor de la transpaleta se hacen pasar por una pequeña pieza guía para evitar que se enganchen con los elementos de la placa. Una vez que se ha comprobado que todo funciona correctamente se atornilla la placa, así como el L298N y el cuadro de mandos en sus posiciones adecuadas. En la figura 93 se muestra la maqueta en su estado final con todo montado.

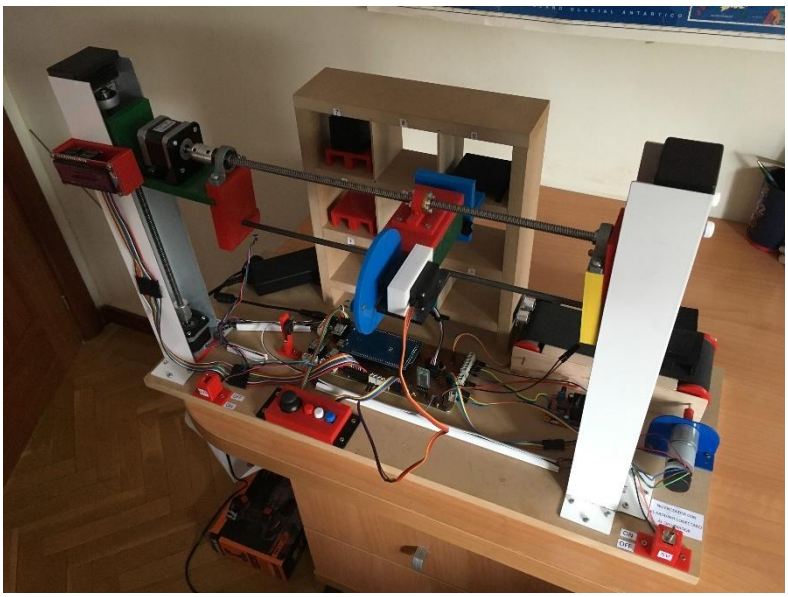


Figura 93. Almacén automatizado construido

4 PROGRAMACIÓN ARDUINO

En este capítulo se explica el programa creado para subir al microcontrolador Arduino que gobierna el sistema. Gracias a este código se podrá utilizar el almacén en sus diferentes modos y a través de las diferentes interfaces. Se recomienda que cuando los alumnos reprogramen el Arduino para realizar sus prácticas, al terminar vuelvan a subir este programa de forma que el almacén esté siempre preparado para su uso con todas sus posibilidades de funcionamiento.

El programa se ha realizado en el IDE de Arduino y se ha dividido en varios archivos contenidos en la misma carpeta del programa. Cada archivo contiene un grupo de funciones con características comunes de forma que el código queda claro y ordenado. Además, los archivos de la misma carpeta aparecen como pestañas dentro del entorno de programación y se comportan en cuanto a la compilación como si fueran un único fichero de forma que la programación resulta muy cómoda.

En cada uno de los apartados del capítulo se explica cada uno de estos archivos y los programas completos se encuentran en el Anexo 2: Código Arduino.

4.1 CONFIGURACIÓN Y MENÚS

En primer lugar, hay que mencionar que por defecto los programas de Arduino cuentan con una función ***setup*** y una función ***loop***. La primera se ejecuta al principio del programa nada más encender el microcontrolador o subir un nuevo programa. En ella se realizan generalmente las funciones de configuración y una vez termina no vuelve a ejecutarse a menos que se apague, resetee o reprograme el Arduino. La segunda se ejecuta cíclicamente cuando termina la anterior y hasta que sucede alguna de las tres cosas anteriores no deja de hacerlo.

El programa realizado cuenta con muchas más funciones, pero todas ellas (excepto las de interrupción) tienen que ser llamadas desde alguna de estas dos funciones o desde otras que hayan sido llamadas por éstas.

En el archivo ***programa_tfg*** se encuentran las funciones ***setup*** y ***loop*** así como las dos funciones de elección de interfaz y modo de funcionamiento: ***menu_interfaz*** y ***menu_modo***.

Al principio del código se incluyen las librerías que se van a utilizar (control del servomotor y la pantalla LCD), se definen los pines de todos los elementos conectados al Arduino y se declaran las variables y constantes globales del programa que necesitan ser accesibles desde varias de las funciones. Entre ellas se encuentran los valores (en pasos de los motores) de las posiciones de cada uno de los cubículos y la cinta con respecto al origen marcado por los finales de carrera.

Figura 94. Diagrama de flujo del "loop"

Tras ello se encuentra la función ***setup*** en la cual se establece el modo de cada pin digital (entrada o salida), se crean las interrupciones de los codificadores incrementales y finales de carrera, y se inicializan el servo, el LCD y el puerto serie (ya sea para el monitor serie o para el bluetooth). La velocidad de transmisión de datos por la comunicación serie se establece en 115200 baudios de forma que se pueda programar el Arduino por bluetooth.

El ***loop*** del programa sigue el diagrama de flujo de la figura 94 que tras comprobar si los finales de carrera están pulsados realiza una llamada a la función de elección de interfaz y a continuación a la de elección de modo dentro de la cual ya se entra al modo escogido. Si se sale del modo se vuelve a pedir la selección de un modo y en caso de solicitud del usuario de cambiar de interfaz se pasa a una nueva iteración del ***loop***.

La función ***menu_interfaz*** llamada desde el loop sigue el flujo de la figura 95. Una parte de código que aparece en esta función pero también en muchas otras es el de mostrar mensajes por el LCD. Como solo se dispone de dos líneas de 16 caracteres, a menudo no es posible mostrar todo lo que se quiere al mismo tiempo, que generalmente son menús con diferentes opciones. La solución a esto ha sido crear un bucle en el que cada cierto tiempo se incremente un contador y se cambie un mensaje por otro acorde al valor de dicho contador. Cuando se terminan los mensajes se vuelve a mostrar el primero y así hasta que se escoge una de las opciones por medio de los botones. En esta función al pulsar un botón se asigna a la variable ***interfaz*** el valor correspondiente para indicar la interfaz de control escogida.

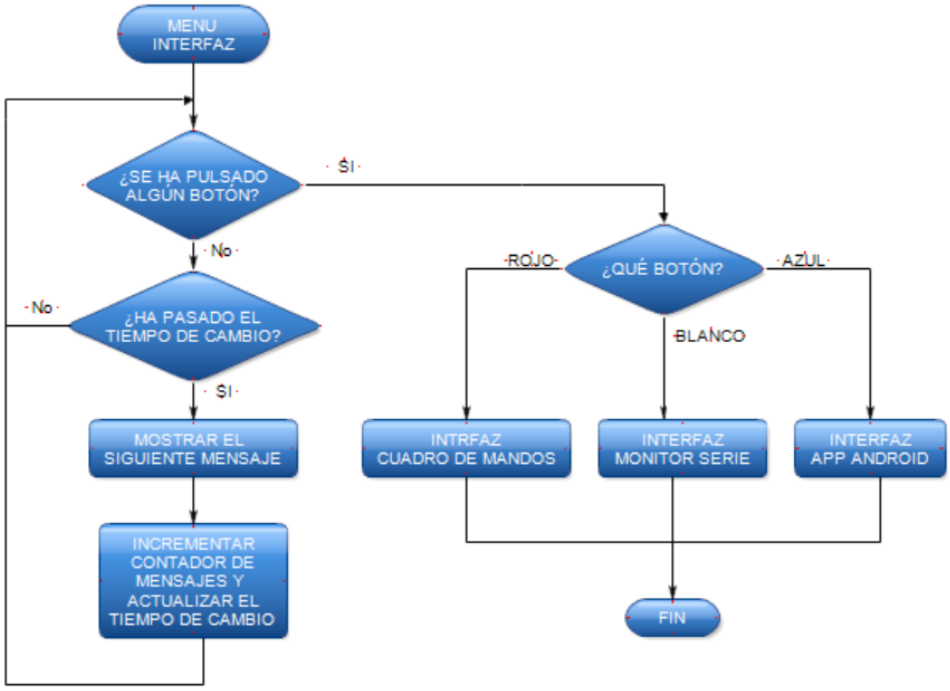


Figura 95. Diagrama de flujo de la función "*menu_interfaz*"

La función ***menu_modo*** presenta una estructura muy similar a ***menu_interfaz*** por lo que no se incluye su diagrama. En ella se solicita el modo de funcionamiento a través de la interfaz escogida y una vez elegido se llama a la función correspondiente a dicho modo. En el caso de la interfaz por aplicación móvil solo hay modo manual por lo que al entrar en esta función habiendo elegido dicha interfaz simplemente se llama a la función de manejo manual desde la aplicación.

4.2 FUNCIONES BÁSICAS DE ELEMENTOS

El archivo denominado ***funciones_basicas*** se compone de las funciones de manejo de varios de los dispositivos que forman parte del sistema de forma individual. Cada vez que en el programa se requiera tomar una medida de distancia, emitir un sonido o el movimiento de algún motor se llamará a la función correspondiente de este archivo. También se incluyen las funciones de interrupción correspondientes a los finales de carrera y los codificadores incrementales. Estas funciones además podrán ser herramientas importantes

para el desarrollo de las prácticas de la asignatura o directamente su creación y utilización puede ser una práctica en sí misma.

La primera función de este archivo es ***ultrasonidos*** en la cual se toma un dato de distancia de uno de los sensores de ultrasonidos. Según el valor de un parámetro de entrada el sensor a utilizar será uno u otro. La función simplemente lleva a cabo tres pasos: se lanza un pulso de 10 microsegundos por el TRIGGER del sensor, se mide el tiempo que tarda en ser captado por el ECHO haciendo uso de la función de Arduino "pulseIn" y se realiza el cálculo para determinar la distancia a la que se encuentra el objeto en el que ha rebotado la onda. Este valor de distancia es devuelto por la función a donde haya sido llamada.

La función ***bip*** únicamente sirve para ahorrar poner el identificador del zumbador cada vez que se quiera emitir un pitido, condición indispensable de la función de Arduino "tone". En esta función simplemente se hace una llamada a "tone" indicando el pin del zumbador, la frecuencia del sonido y su duración. Estos dos últimos parámetros se pasan como argumentos a ***bip***.

Las funciones ***stop_horizontal*** y ***stop_vertical*** son las funciones de interrupción que se llaman al cambiar el estado del final de carrera correspondiente tal y como se definió en el ***setup***. En estas funciones simplemente se comprueba si se ha pulsado o se ha soltado el final de carrera y se modifica una variable booleana en consecuencia. Además, si se pulsa se emite un pitido del zumbador para informar de dicho hecho.

La función ***pulso_pap*** es la encargada de enviar pulsos a los motores paso a paso según se requiera. Para conseguir un movimiento continuo del motor (no solo un paso) esta función debe llamarse de forma recursiva dentro de un bucle. Deben pasarse como parámetros el eje a mover (vertical u horizontal), el sentido del movimiento y la velocidad lineal de avance deseada. En ella se cambia el estado del pin STEP del A4988 correspondiente en caso de que haya pasado el tiempo de pulso correspondiente a la velocidad de avance y el sistema de carga/descarga no se encuentre en uno de los extremos de su movimiento. Uno de estos extremos (para cada eje) está limitado por un final de carrera, pero el otro se determina vía software en esta función impidiendo que se envíen pulsos (en el sentido correspondiente) en caso de que se haya llegado a un número máximo de pasos. Cada vez que se envía un nuevo pulso se actualiza el contador de pulsos de ese eje para así saber en todo momento la posición del transelevador. El diagrama de flujo de esta función puede observarse en la figura 96.

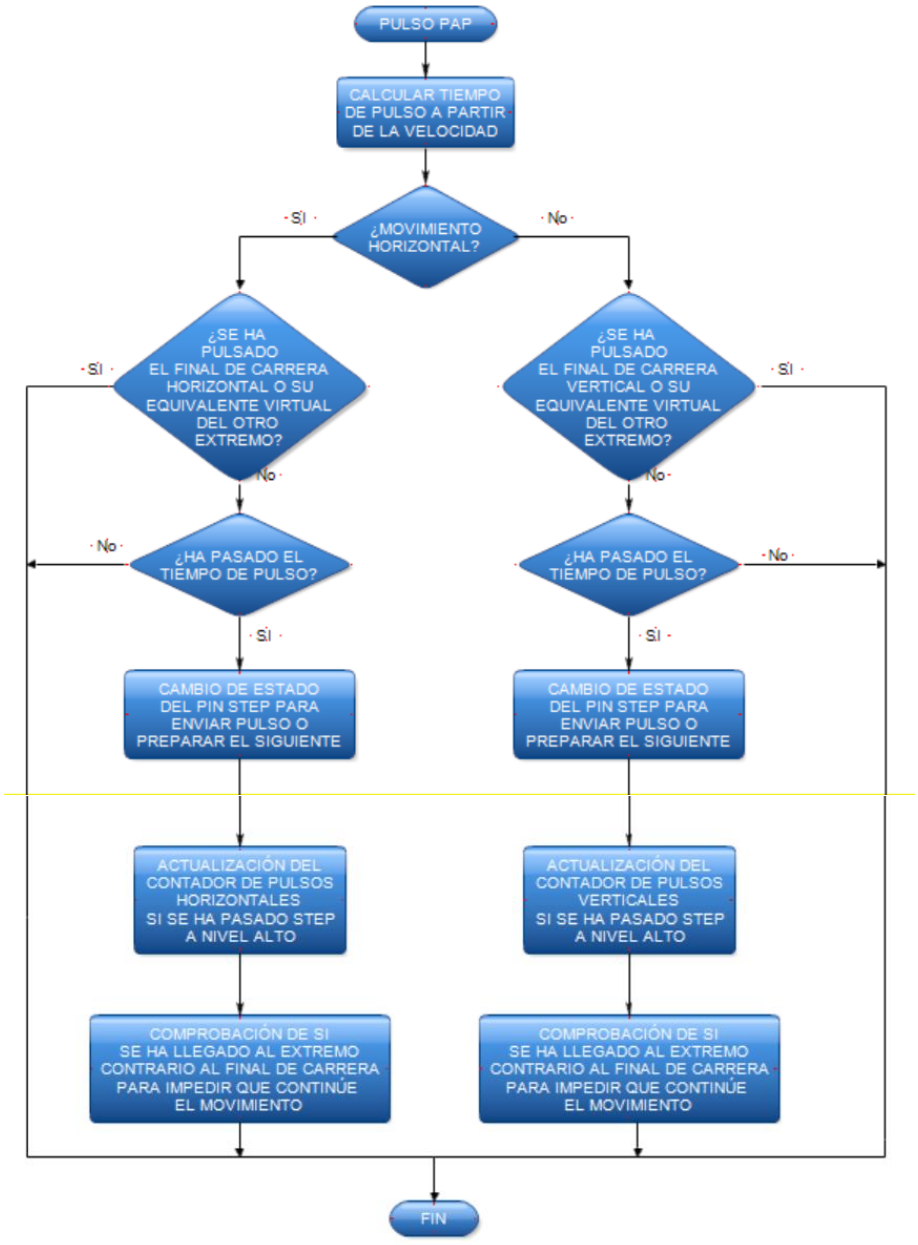


Figura 96. Diagrama de flujo de la función "pulso_pap"

Para el movimiento del servomotor del sistema de carga/descarga se ha programado la función ***mover_servo***. Si se ordena al servo moverse a una posición mediante la instrucción correspondiente de la librería "servo.h", el actuador se mueve a su máxima velocidad hasta llegar a la posición solicitada. Esta velocidad es demasiado elevada para la aplicación de este proyecto puesto que si la transpaleta se mueve tan deprisa la carga del palé podría caerse. Por ello, en esta función se ordena el movimiento de un solo grado en caso de que haya pasado un tiempo que se introduce como

parámetro, el cual será proporcional a la velocidad. Para que se llegue a la posición deseada (también parámetro de la función), la función debe llamarse de forma recursiva ya que, si no, solo se avanzará un grado (en caso de que haya pasado suficiente tiempo) o el servo no se moverá (en caso contrario). Cuando se llega a la posición deseada se devuelve true para indicar el fin del movimiento y en caso contrario se devuelve false.

En el programa siempre se mueve el servo entre las posiciones de 30 y 180º que se corresponden con la posición de la transpaleta extendida y recogida respectivamente. Además, el tiempo entre movimientos es siempre una constante de 20ms pero los alumnos podrán experimentar utilizando diferentes posiciones y velocidades en sus prácticas siempre que tengan cuidado de que la transpaleta no choque con el almacén y no caigan cargas encima de la placa. En la figura 97 se puede apreciar el diagrama de flujo de esta función.

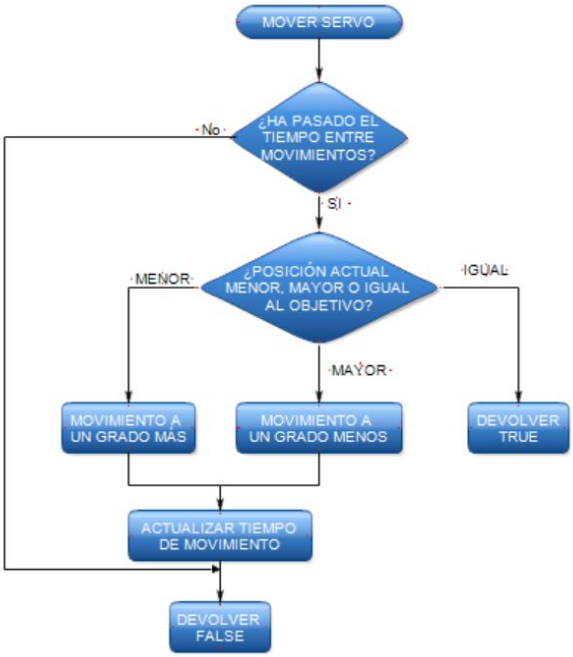


Figura 97. Diagrama de flujo de la función “mover_servo”

Es necesario mencionar que se ha modificado el orden de la lista de los temporizadores del Arduino que utiliza la librería servo.h porque tal y como está en la librería original no se podría utilizar el pin 45 como PWM al emplear el servo el temporizador 5 que utiliza este pin. Esta funcionalidad es necesaria ya que a él se conecta el pin ENA del controlador del motor de continua así que se ha cambiado el puesto de dicho temporizador en la lista por el número 4. La librería modificada se incluye en el código que se proporcionará junto con el informe del TFG.

Para el control del motor de corriente continua que mueve la cinta transportadora se han programado varias funciones que hacen posible el control de la posición o la velocidad del motor haciendo uso de un regulador PID.

Los controladores PID son los mecanismos de control más extendidos y empleados en los procesos industriales por el buen resultado que dan en la mayoría de los casos. Aunque puede presentar muchas variantes y estructuras, su esquema básico se presenta en la figura 98. Se trata de un PID paralelo que actualmente es lo más habitual habiendo quedado obsoleto el uso de PIDs serie. Cabe mencionar que un PID es un regulador basado en señal, es decir, su acción se basa en el valor de una señal medida y otra de referencia, pero no incorpora conocimiento explícito del proceso como sí hace por ejemplo un controlador predictivo que supone un control mucho más complejo y avanzado.

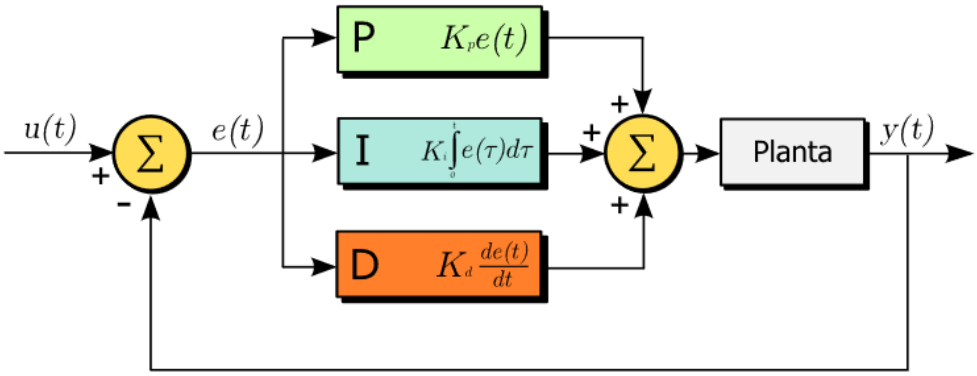


Figura 98. Esquema de un controlador PID

El funcionamiento es el siguiente: con un sensor se mide el valor real de la variable que se quiere que controle el PID (variable controlada) y se compara con el valor que se desea que tenga esa variable (referencia). La diferencia entre ellas es una señal de error que se introduce al regulador y con la cual éste calcula una señal de salida (variable manipulada) a introducir al actuador correspondiente de la planta para conseguir reducir el error hasta que sea nulo.

El controlador PID posee tres acciones que generalmente actúan en paralelo como son la proporcional, la integral y la derivativa. Las dos últimas no son obligatorias y deben incluirse o no en función de las características y la respuesta del sistema. Esto da lugar a controladores P, PD o PI que carecen de alguna de las acciones típicas de un PID.

La acción proporcional consiste en el producto del error por la constante proporcional (Kp). Esta constante es el primer parámetro de sintonía del regulador que deberá ajustarse para disminuir el error estacionario lo mayor posible (no eliminarlo) sin que se produzcan sobrepicos elevados. Los controladores P solo se utilizan en sistemas cuya función de transferencia posea un integrador (acción integral en el propio sistema) o en procesos en los que no sea importante tener error estacionario nulo.

La acción integral actúa integrando el error respecto al tiempo y multiplicando dicho valor por la constante integral Ki (inversa del tiempo integral Ti). Con ello se consigue una salida proporcional al error acumulado en el tiempo que permite eliminar completamente el error estacionario. El valor de Ki debe ajustarse para conseguir una respuesta lo más rápida posible, pero sin sobrepicos considerables. Los reguladores PI son realmente los más empleados ya que suelen dar mejores prestaciones que los PID en sistemas rápidos, con ruidos o en los que sea habitual el cambio de la referencia.

Un fenómeno relacionado con la acción integral que se produce de forma habitual es el **wind-up,** el cual consiste en lo siguiente: los actuadores y señales de salida de los reguladores tienen un rango limitado de operación como en este caso la señal PWM que se envía al controlador del motor (valor entre 0 y 255). Este hecho físico puede dar lugar a que la señal llegue a saturación pero el término integral siga intentando aumentar la señal de control por encima (o debajo) de estos límites mientras siga habiendo error. Esto se traduce en que, al llegar a la referencia, la señal de control tarda un tiempo en bajar del límite (hasta que se reduce el error integral), produciéndose sobrepicos indeseados en la salida. Por ello es importante incluir sistemas antiwindup en los reguladores PI y PID que mejoran la respuesta tal y como se puede apreciar en el gráfico de la figura 99.

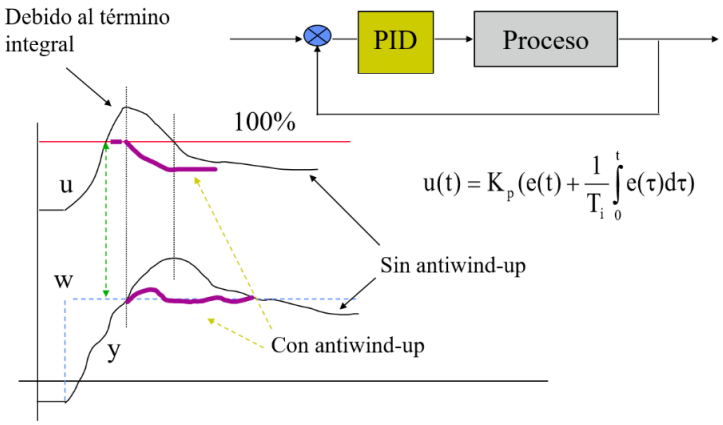


Figura 99. Ejemplo de wind-up y antiwind-up

Por último, la acción derivativa cosiste en derivar el error y multiplicarlo por la constante derivativa Kd (o tiempo derivativo Td). Con esta componente se suavizan los cambios en la señal de control asociados a cambios rápidos del error, evitando sobrepicos. Si el error crece, la acción derivativa acelera la señal de control mientras que si el error decrece se modera la señal de control para evitar oscilaciones. De utilizar esta acción debe incorporarse un filtro derivativo que limite la ganancia derivativa a altas frecuencias. Aun así, la acción derivativa no es muy utilizada debido a que saltos en la referencia o señales ruidosas provocan valores inadecuados de la señal de control. Por ello, los reguladores PID completos suelen limitarse a procesos lentos como por ejemplo el control de temperatura, en los que desde el cambio en la referencia o la aparición de la perturbación hasta que se refleja el hecho en la salida transcurre un intervalo de tiempo considerable.

Una vez comentado someramente el funcionamiento de los controladores PID se procede a exponer las funciones creadas para su implementación en el proyecto:

Las funciones ***contarA*** y ***contarB*** son las funciones de interrupción del codificador incremental de dos canales que posee el motor. Como se especificó en el ***setup***, estas funciones son llamadas cada vez que se produce un flanco (ya sea de subida o de bajada) en el canal correspondiente. En ellas se comprueba en primer lugar el tipo de flanco del canal en cuestión y a continuación se comprueba el estado del otro canal para determinar el sentido de giro y así saber si incrementar o decrementar el contador de pulsos.

La función ***compute*** es la que implementa el controlador PID. A ella se pasan la referencia y la medida de posición o velocidad junto con el intervalo de tiempo entre mediciones que será diferente según el caso. En primer lugar, se escogen los parámetros del controlador Kp, Ki y Kd según si el control es de posición o de velocidad. La sintonía de los parámetros se detalla más adelante. Tras ello se calculan los errores proporcional, derivativo e integral (si la salida no está saturada para evitar el wind-up) y se aplica la fórmula del PID para obtener la salida. En caso de sobrepasar -255 o 255 se limita la salida a esos valores. Realmente el término derivativo no se utiliza en ninguno de los dos controles ya que para la posición se utiliza un regulador P y para la velocidad un PI. Sin embargo, se mantiene en el código para que los alumnos puedan experimentar con diferentes tipos de reguladores y valores de los parámetros.

La función ***girar*** recibe el valor PWM determinado por el PID y simplemente pone los pines IN1 e IN2 en el estado correspondiente al signo del PWM (para el sentido de giro) y envía el valor absoluto al ENABLE.

Las funciones ***motorcc*** y ***motorcc_pos*** deben ser llamadas desde el programa de forma recursiva para implementar un control de velocidad o de posición respectivamente. Se debe pasar como argumento la referencia de velocidad en rpm o posición en grados (coincide con el número de pulsos de los codificadores incrementales al dar 360 pulsos por vuelta) según el caso. En el programa realizado solo se llama a la función de velocidad, pero se ha incorporado la de posición también ya que podrá ser útil para las prácticas de la asignatura. En ellas se comprueba si ha transcurrido el tiempo entre cálculos y si es así se llama a la función ***compute*** cuyo resultado se aplica a la función ***girar*** de forma que el motor se mueva. En el caso de la función de velocidad antes de llamar a estas funciones hay que calcular la velocidad en rpm en función de los pulsos de los codificadores incrementales dados desde el último cálculo. Además, se resetea el contador de pulsos para la siguiente iteración y se actualiza el tiempo de cálculo (cosa que también se hace en la función de posición).

4.2.1 SINTONÍA PID MOTOR CC

Para la sintonía de los parámetros del PID se ha utilizado el programa MATLAB 2015 en el que se ha programado la función que se puede ver al inicio del Anexo 2: Código Arduino basándose en la proporcionada en la asignatura. En esta función se leen los datos que envía el Arduino por el puerto serie (de posición o de velocidad medidos) y se muestran en una gráfica con las escalas de los ejes adecuadas para una correcta visualización. El número de datos a leer se introduce como parámetro al llamar a la función.

La toma de datos con MATLAB se ha realizado programando el Arduino con un código que incluye únicamente las funciones del motor CC siendo la función ***motorcc*** o ***motorcc_pos*** el ***loop*** del programa y poniendo un único "*Serial.print*" que saque la velocidad o posición medidas cada vez que se realice un cálculo.

Gracias a las gráficas de MATLAB se puede observar el comportamiento del sistema con diferentes valores de los parámetros del regulador y así poder lograr la mejor sintonía posible.

Sintonía de velocidad

La velocidad del motor es una variable que puede cambiar rápidamente al modificar el PWM introducido y en esta aplicación va a ser común el cambio de la referencia de velocidad ya que es precisamente lo que se hace por ejemplo con el joystick del cuadro de mandos. Como se ha explicado, estos

hechos hacen que un regulador PID no sea lo más adecuado, sino que la respuesta será mejor empleando un regulador PI.

Un hecho importante a destacar es que el motor no se mueve con un cambio de velocidad lineal entre los valores de PWM 0 y 255, sino que hasta un valor de aproximadamente 40 permanece parado y una vez se sobrepasa dicho umbral empieza a girar a la velocidad correspondiente al valor PWM. Por ello, para los ensayos se va a hacer ir al motor desde parado hasta una velocidad bastante superior a ese valor que va a ser 100rpm.

En primer lugar, se ha incorporado solo el término proporcional con el que se debe conseguir una respuesta con el menor error posible, pero sin sobrepicos considerables.

Poniendo un valor de ganancia proporcional Kp=1 (figura 100) la velocidad llega hasta unas 45rpm pero en el transitorio presenta un sobrepico de hasta 80rpm, valor demasiado elevado por lo que esta ganancia proporcional se considera excesiva. A menores valores de Kp el error estacionario es mayor pero el sobrepico se reduce llegando a concluir que un buen compromiso entre ambas cosas se consigue con Kp=0,7 (figura 101) con el que se llega a 33rpm de estacionario y 46rpm de sobrepico.

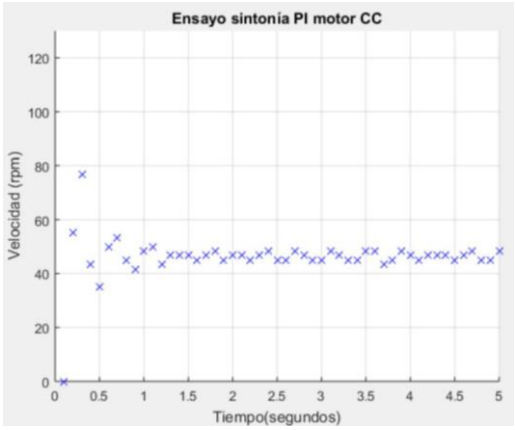


Figura 100. Regulador P con Kp=1

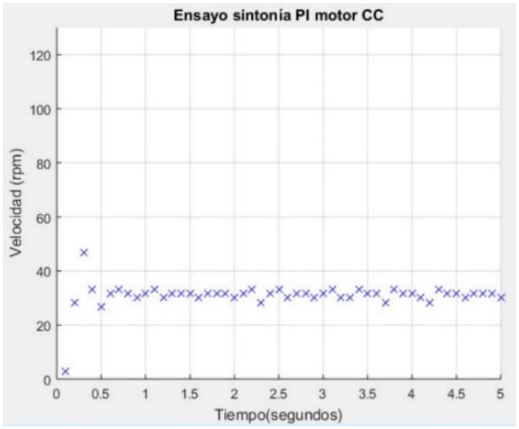


Figura 101. Regulador P con Kp=0,7

Fijando Kp en 0,7 se incorpora el término integral para eliminar el error estacionario. Se empieza probando con un valor bajo de Ki=0,001 con el que la respuesta (figura 102) es muy lenta no llegando al estacionario hasta pasados casi 5 segundos. Con Ki=0,005 (figura 103) el sistema es mucho más rápido pues se estabiliza en menos de 1 segundo pero se produce un sobrepico que llega hasta las 110rpm. Reduciendo ligeramente a Ki=0,004 (figura 104) se ralentiza ligeramente la llegada al estacionario (algo más de 1 segundo) pero el sobrepico queda prácticamente eliminado por lo que se escoge este valor para la sintonía.

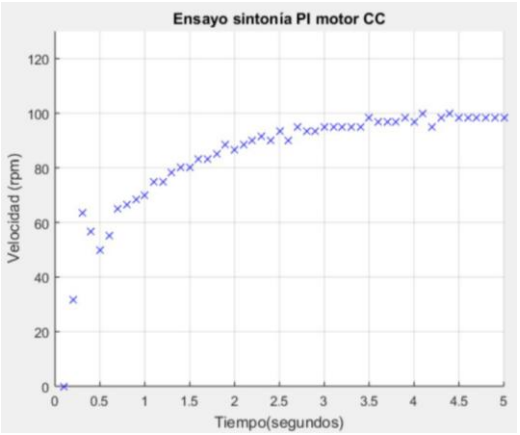


Figura 102. Regulador PI con Kp=0,7 y Ki=0,001

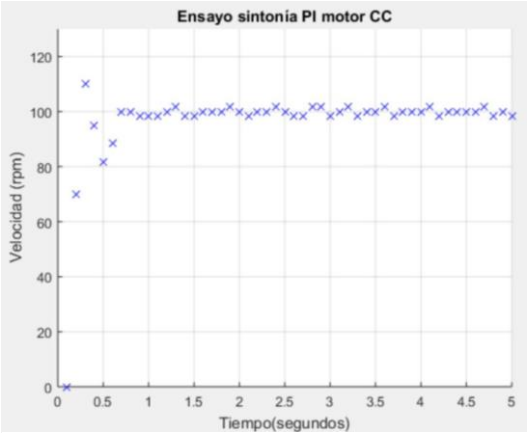


Figura 103. Regulador PI con Kp=0,7 y Ki=0,005

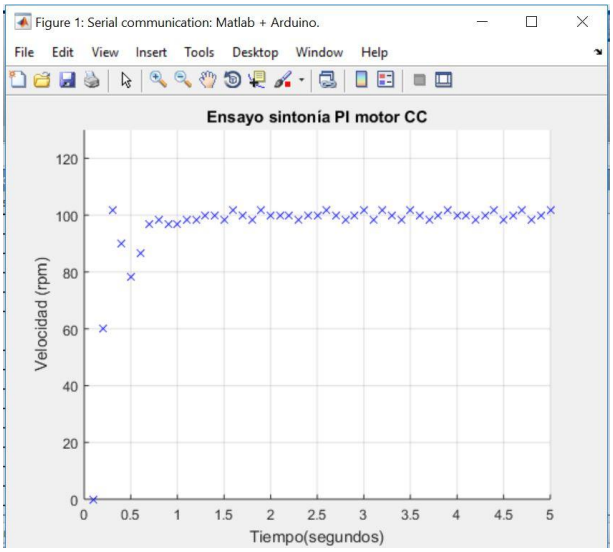


Figura 104. Regulador PI con Kp=0,7 y Ki=0,004

Por último, se comprueba que incluyendo el término derivativo por muy pequeño que sea (Kd=0,001) no se consigue mejorar el sistema y de hecho empeora ligeramente ya que se produce un pequeño sobrepico de 5rpm (figura 105).

Por tanto, la mejor sintonía encontrada para el control de velocidad es:

Kp=0,7 Ki=0,004 y Kd=0

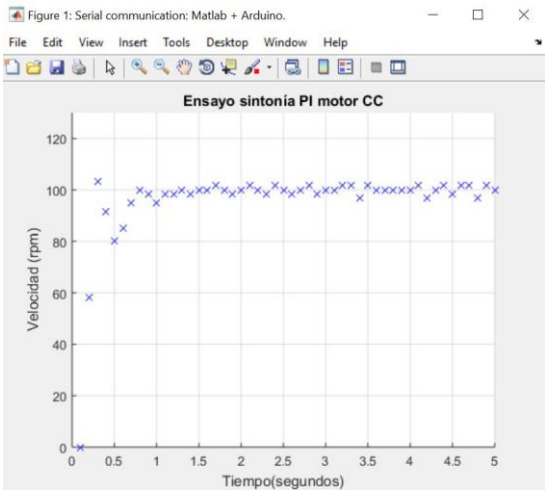


Figura 105. Regulador PID con Kp=0,7, Ki=0,004 y Kd=0,001

Sintonía de posición

Para la sintonía del controlador de posición es un problema el hecho de que el motor no se mueva hasta un cierto valor de PWM. Este defecto provoca que al no poder moverse a pequeñas velocidades sea más complicado ajustar la posición a la deseada.

Además, el intervalo de tiempo entre cálculos tampoco es fácil de escoger ya que si el tiempo es grande (100ms con los que se hace el control de velocidad) se producen muchos pulsos del codificador incremental de una iteración a otra siendo por tanto complicado llegar a una posición concreta teniendo en cuenta que no se pueden tener velocidades pequeñas. Como se puede apreciar en la figura 106, si se utiliza un intervalo de 100ms, el sistema es completamente oscilatorio alrededor de la referencia de 100 con unos sobrepicos muy amplios en ambos sentidos.

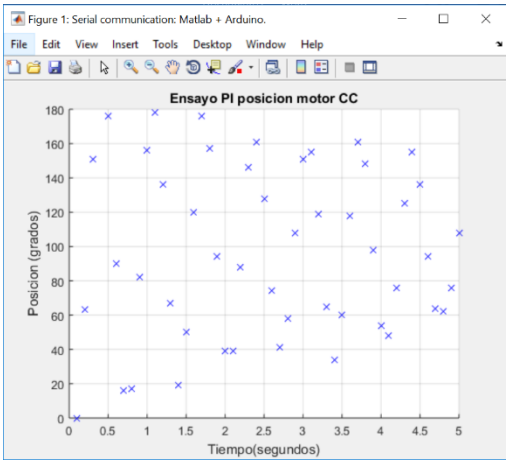


Figura 106. Regulador PI con tiempo entre cálculos de 100ms

Por el contrario, con tiempos pequeños (10ms) el motor no tiene tiempo de reaccionar al valor de PWM calculado en cada iteración de manera que si se está moviendo a una cierta velocidad y se calcula que debe pararse (o casi), el motor sigue moviéndose varias iteraciones más produciéndose un sobrepaso en la posición. Por este motivo es complicado no tener un transitorio con sobrepicos.

Cuando el sentido de giro se revierte tras un sobrepico, se vuelve a acercar a la posición deseada, pero llega un punto en el que el PWM baja por debajo de lo necesario para que el motor se mueva y se mantiene quieto a una cierta distancia de la posición deseada. En el caso de implementar un regulador PI, en el momento en el que el error integral aumenta tanto que el PWM vuelve a ser el necesario para que el motor se mueva se consigue llegar a la posición deseada, pero se vuelve a sobrepasar en el otro sentido y el ciclo vuelve a empezar no llegando nunca al objetivo. Este ciclo se puede apreciar en la figura 107 en la que se puede ver que con un PI de Kp=1, Ki=0.007 y la referencia de posición en 100 el motor oscila entre aproximadamente 80 y 120rpm.

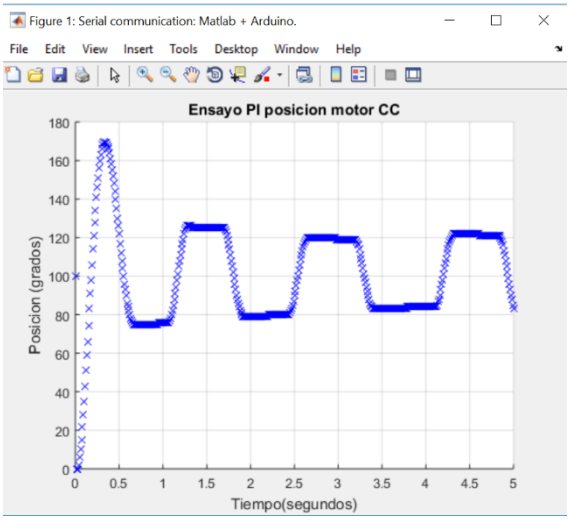


Figura 107. Regulador PI con Kp=1 y Ki=0,007

Este comportamiento se puede mejorar haciéndolo menos oscilatorio por medio de un aumento de la ganancia proporcional. Con ello se consigue que los valores entre los que oscila la posición se acerquen más a la referencia y se produzcan menos oscilaciones como se puede apreciar en las figuras 108 y 109 con Kp=2 y Kp=10. Sin embargo, no es posible eliminar completamente las oscilaciones puesto que, al no estar exactamente en la referencia, el error integral sigue aumentando hasta que llega un momento en que provoca que el motor se mueva produciéndose una oscilación.

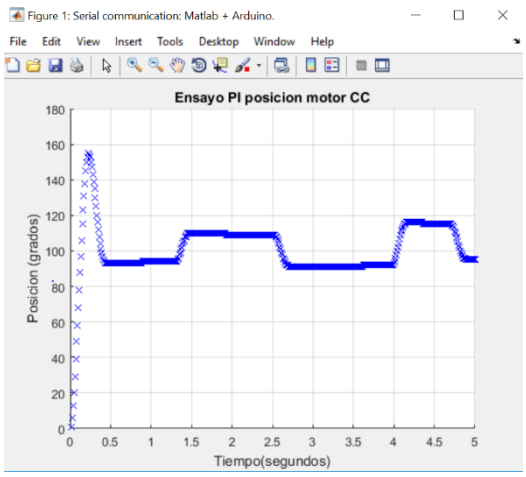


Figura 108. Regulador PI con Kp=2 y Ki=0,007

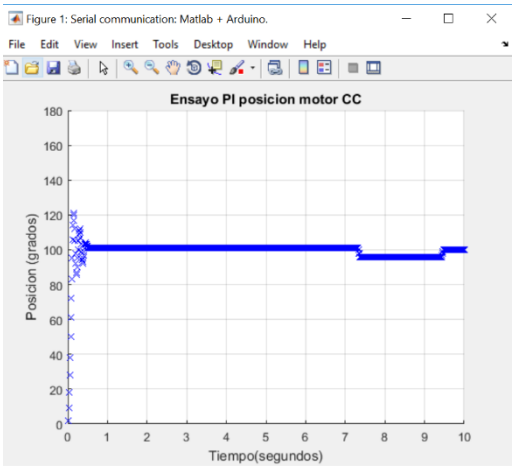


Figura 109. Regulador PI con Kp=10 y Ki=0,007

Como es lógico, cuanto menor es la constante integral (inverso del tiempo integral) las oscilaciones se producen cada menos tiempo ya que se tarda más en acumular el error necesario para que se vuelva a mover el motor. Este hecho se muestra en las figuras 110 y 111 en las cuales el Ki vale 0,02 y 0,002 respectivamente.

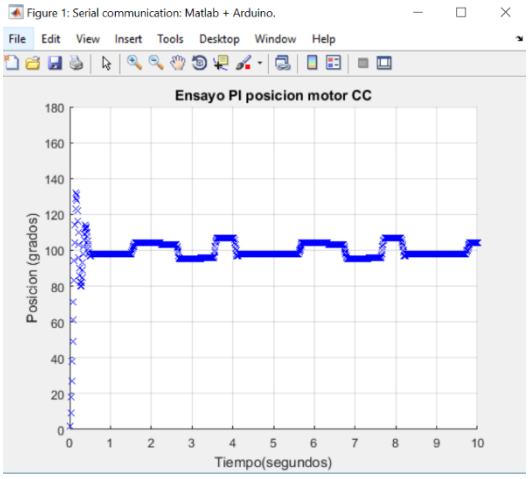


Figura 110. Regulador PI con Kp=10 y Ki=0,02

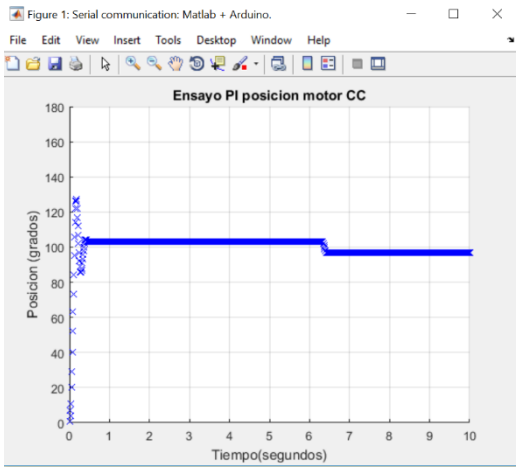


Figura 111. Regulador PI con Kp=10 y Ki=0,002

Se concluye que un controlador PI no es adecuado al no poder eliminar las oscilaciones por muy pequeñas o poco frecuentes que se puedan hacer. En el caso de utilizar solo un controlador P, cuando el motor se para a una cierta distancia de la referencia tras el transitorio inicial ya no se mueve de ahí. Esto es debido a que la acción proporcional siempre da como resultado el mismo PWM que será un valor insuficiente para que el motor se mueva. El error estacionario no se puede eliminar solo con un controlador P, pero ajustando correctamente la ganancia proporcional se puede intentar conseguir una respuesta estable que se mantenga bastante cerca de la referencia.

Al tener una resolución bastante buena (un grado) gracias a los codificadores incrementales y no ser realmente necesaria tanta precisión ya que la aplicación no lo requiere, se puede dar por buena una respuesta de este tipo en la que no se logre la posición exacta pero sí una buena aproximación estable.

Con una Kp=1 (figura 112) se obtiene una respuesta bastante buena en la que apenas hay sobrepico y el sistema se estabiliza en 104 (referencia en 100).

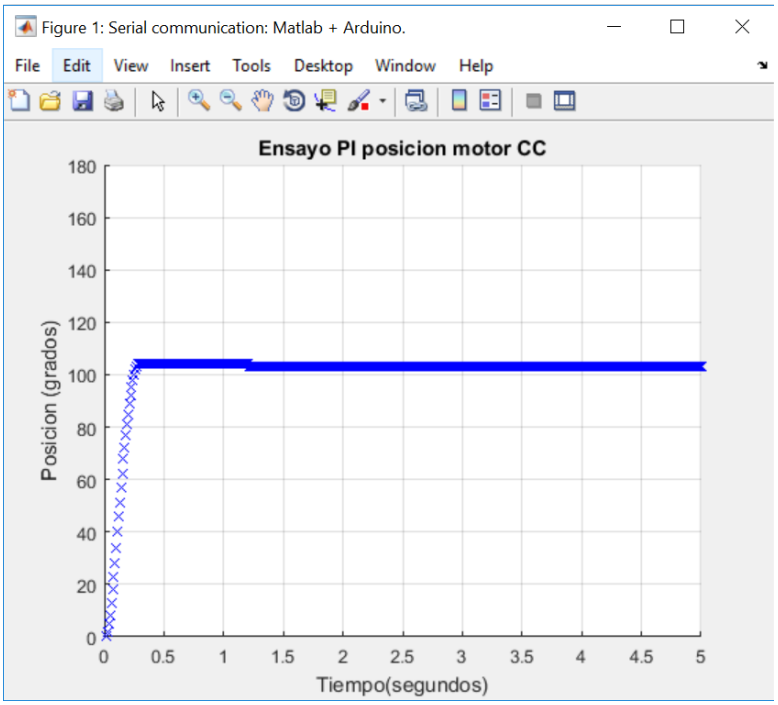


Figura 112. Regulador P con Kp=1

Aumentando hasta Kp=10 (figura 113) el transitorio es más oscilatorio pero también se consigue mayor precisión al llegar a 101, solo un grado por encima de la referencia.

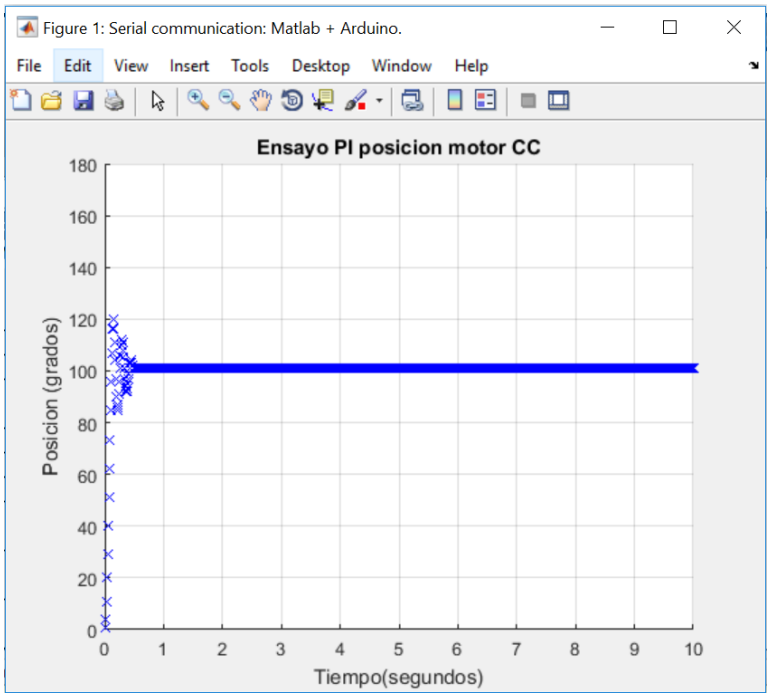


Figura 113. Regulador P con Kp=10

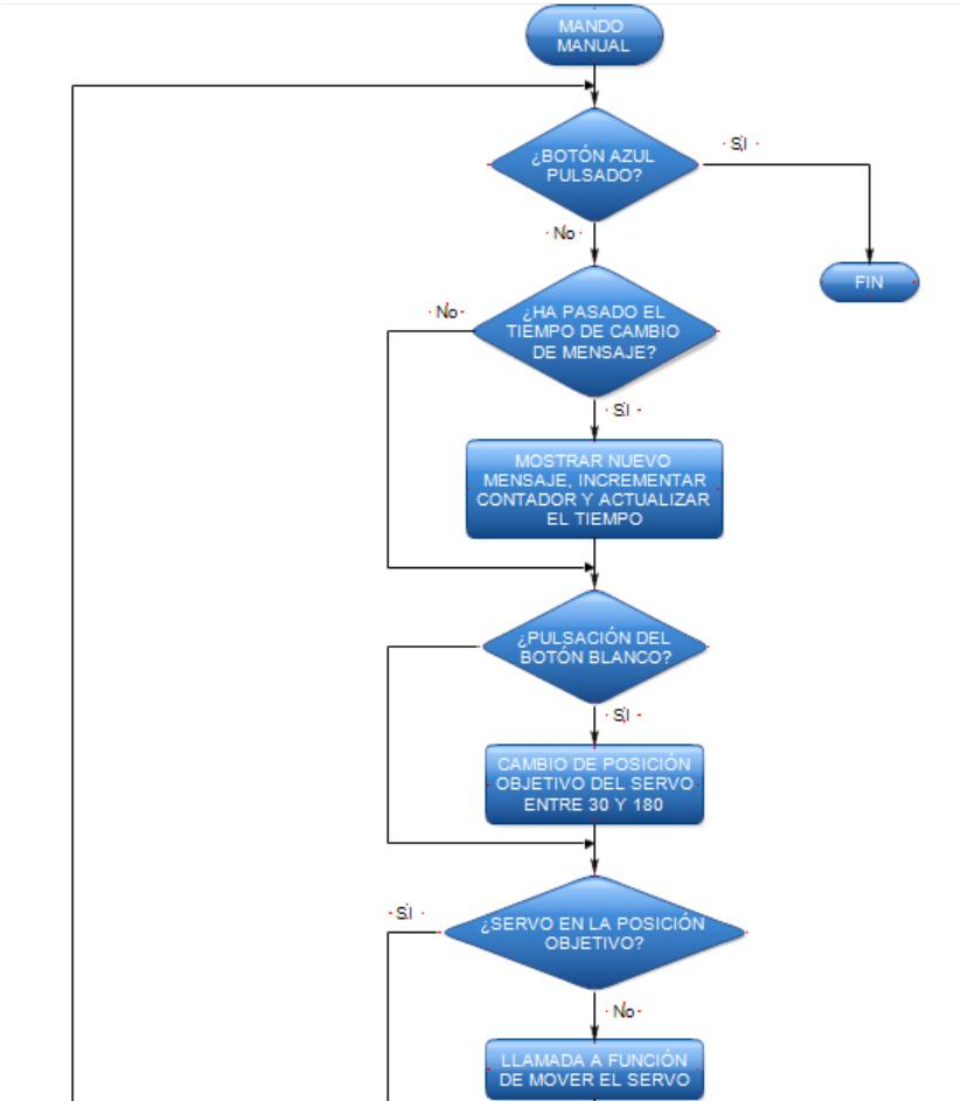
Por último, aumentando Kp por encima de 10 no se consigue mejorar el valor final (101) y el transitorio se va haciendo más largo con más oscilaciones por lo que no tiene sentido llegar hasta valores tan altos.

Puede concluirse que debido al problema de la no linealidad de la velocidad del motor con el PWM es mejor utilizar un regulador P que un PI para controlar la posición del motor, y el valor de Kp deberá elegirse dentro de un rango de valores 1-10 en función de si se desea mayor precisión o menores sobrepicos.

4.3 INTERFAZ CUADRO DE MANDOS

La interfaz por el cuadro de mandos consiste en manejar el almacén mediante un joystick y tres botones físicos situados en la parte delantera de la maqueta para una buena accesibilidad por parte del usuario. Este archivo de código se compone de dos funciones correspondientes al modo manual y al automático desde esta interfaz y una tercera función de elección de un cajón desde el cuadro de mandos.

La función del modo manual llamada ***mando_manual*** cuyo diagrama de flujo se puede ver en la figura 114 funciona de la siguiente manera: mientras no se pulsa el botón azul se permanece en la función y en caso de hacerlo se vuelve al menú de modos. De igual forma que en los menús de interfaces y modos, se muestran diferentes mensajes por el LCD que se van sucediendo cada 1,5 segundos indicando las acciones posibles. Las pulsaciones del botón rojo hacen que el joystick pase de manejar los motores paso a paso (movimientos vertical y horizontal) a manejar el motor de continua (movimiento horizontal) y viceversa. Según la posición del joystick los motores se moverán en un sentido u otro y con mayor o menor velocidad. Por último, pulsando el botón blanco se extiende o recoge la transpaleta (según su estado en el momento de la pulsación). Al estar ejecutando la función recursivamente dentro del bucle del botón azul se consigue que se produzcan los movimientos de los motores correctamente.



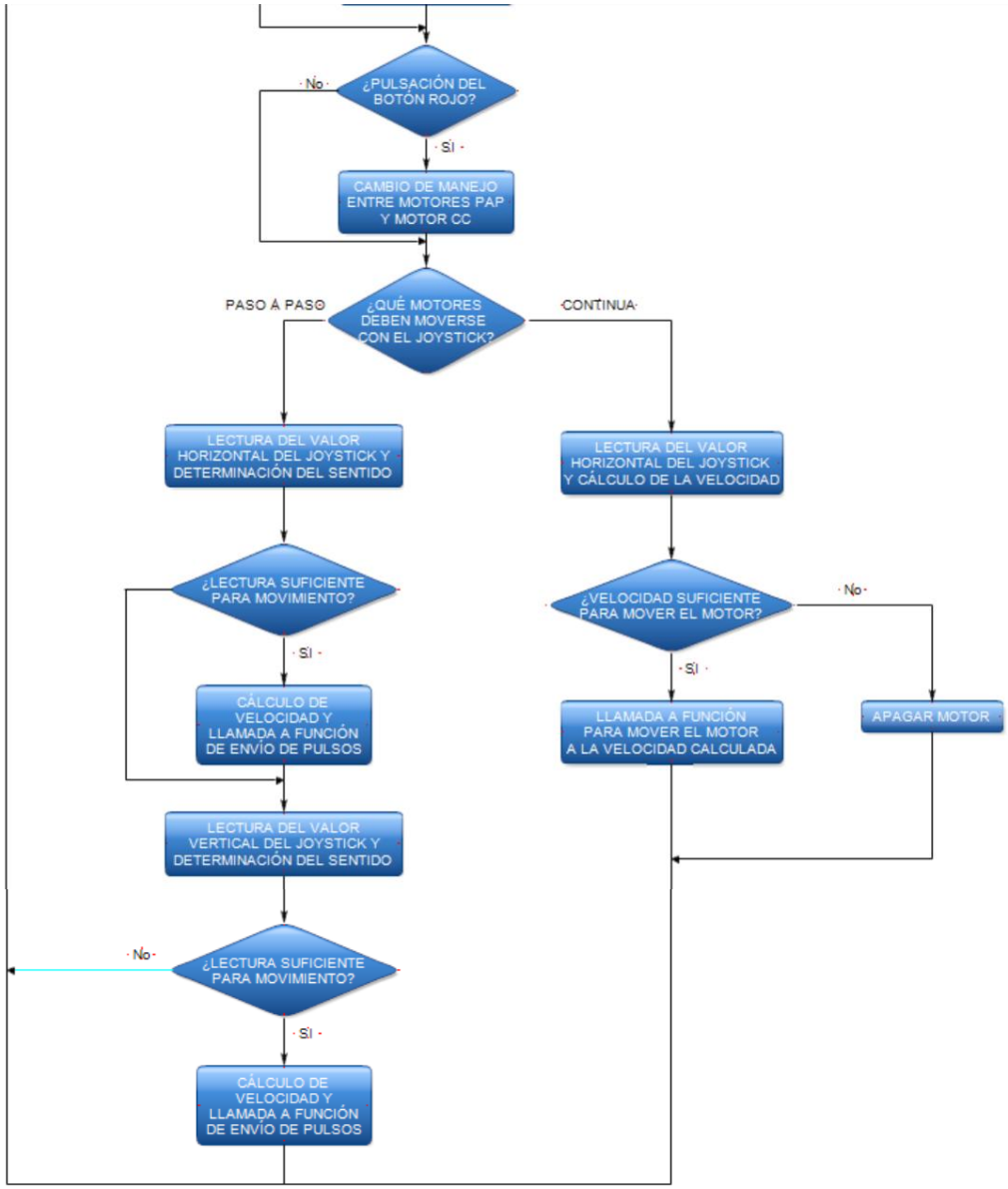


Figura 114. Diagrama de bloques de la función “mando_manual”

En la función ***mando_automatico*** lo primero que se hace es escoger el modo de reconocimiento del almacén. Este procedimiento consiste en determinar los cajones ocupados y vacíos para que posteriormente el Arduino sepa si se puede meter o sacar algo en un cajón concreto o no. El estado de los cajones puede decirlo el usuario a través de los botones o el transelevador se puede ir moviendo automáticamente por delante de todos los cubículos tomando medidas del sensor de ultrasonidos para determinar la presencia o no de objetos. Según escoja el usuario se llamará a la función de reconocimiento con un parámetro diferente que determinará cuál de las dos opciones se lleva a cabo.

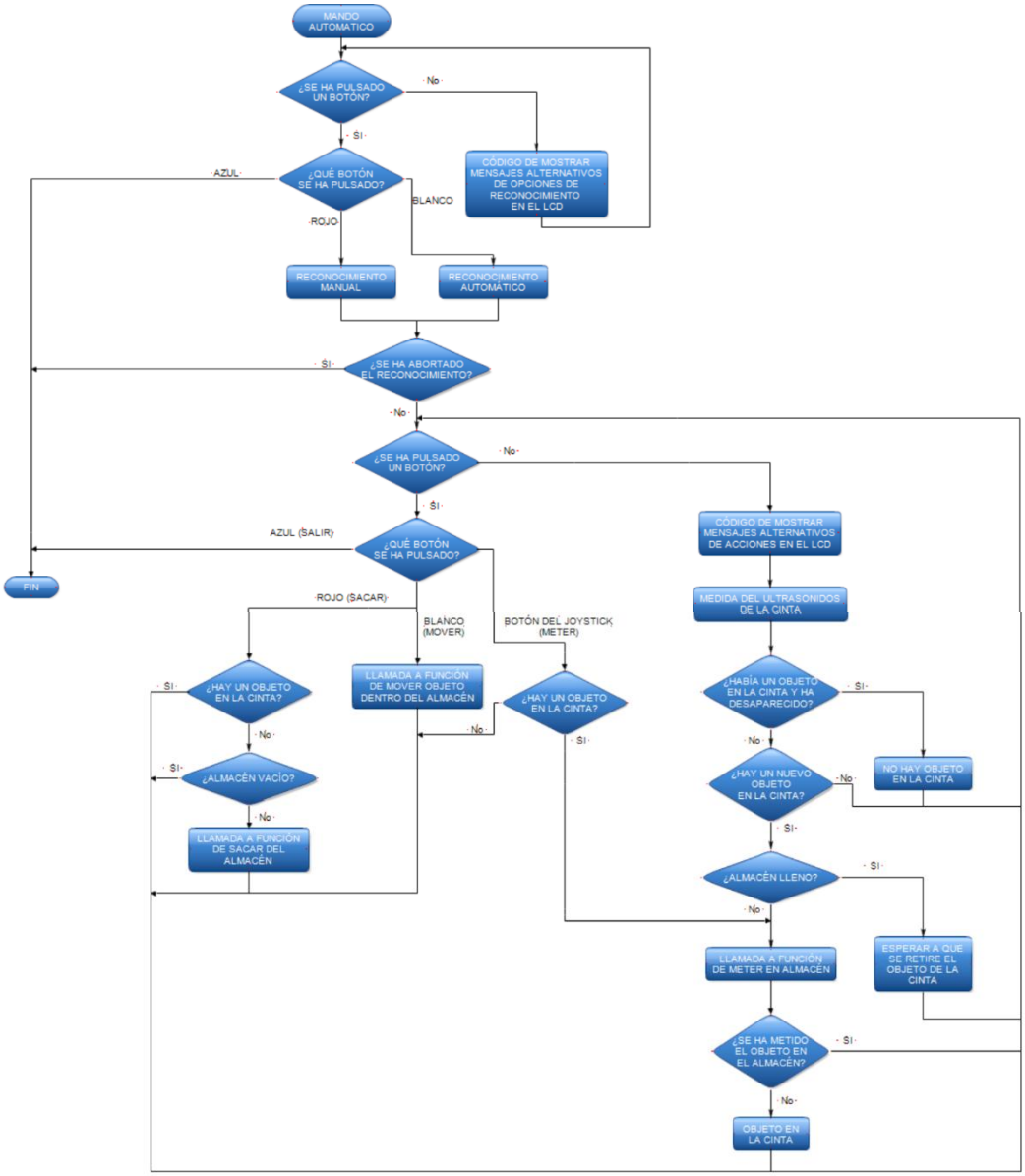


Figura 115. Diagrama de flujo de la función "mando_automatico"

Tras el reconocimiento hay un menú para escoger la acción a realizar: sacar un palé del almacén por la cinta, mover un palé entre cubículos o meter un palé que se encuentre en la cinta en el almacén. Si se detecta un nuevo palé en la cinta mediante el sensor de ultrasonidos mientras se espera a que se escoja una acción, se llama a la función de meter un palé en el almacén en la que se da la opción de no meter el palé por el momento. Si se obvia el objeto hay posibilidad de volver a llamar a la función de meter objeto más adelante escogiéndolo con el cuadro de mandos. En todo momento si se pulsa el botón azul se puede abortar la acción que se esté realizando y salir al menú de modos finalizando la función. El diagrama de flujo que aparece en la figura 115 muestra con más detalle las acciones realizadas en esta función.

Por último, en la función ***escoger_cajon,*** que se llamará desde las funciones de acciones del almacén en caso de utilizar el cuadro de mandos, pide al usuario escoger un cubículo para realizar alguna acción sobre él. Para ello se pide desde el LCD escoger una fila con los botones y posteriormente una columna. Si se ha elegido correctamente se devuelve el número de cajón correspondiente y si no se repite la elección.

4.4 INTERFAZ MONITOR SERIE

Para el manejo del sistema desde el monitor serie de Arduino se utilizan las dos funciones de este archivo que se corresponden con el modo manual y automático. A ellas se accede escogiendo la interfaz y modo correspondiente en los menús del inicio.

En la función ***monitor_manual*** se muestra un menú con las opciones que se muestran en la tabla 6 donde se explica el proceder al escoger cada una de ellas.

OPCIÓN	ACCIÓN	PROCEDIMIENTO
1	Movimiento vertical	Se pide una velocidad entre 1 y 10mm/s (con signo) y se envían pasos a los motores verticales para moverse a esa velocidad y sentido hasta que se introduce un comando, momento en que se para y se vuelve a mostrar el menú de acciones.
2	Movimiento horizontal	Se pide una velocidad entre 1 y 10mm/s (con signo) y se envían pasos a los motores verticales para moverse a esa velocidad y sentido hasta que se introduce un comando, momento en que se para y se vuelve a mostrar el menú de acciones.
3	Movimiento de la cinta	Se pide una velocidad entre 40 y 190rpm (con

		signo) y se llama recursivamente a la función de control de velocidad del motor cc para moverse a esa velocidad y sentido hasta que se introduce un comando, momento en que se para el motor y se vuelve a mostrar el menú de acciones.
4	Movimiento de la transpaleta	Se cambia la posición objetivo del servo y se llama a la función de mover el servo hasta que devuelva true indicando que el movimiento ha finalizado.
OTRO	Salir	Se sale de la función al menú de modos.

Tabla 6. Acciones de la función "monitor_manual"

La función ***monitor_automatico*** es análoga a la del modo automático desde el cuadro de mandos, pero mostrando los menús y mensajes por el monitor serie y escogiendo desde el teclado las opciones. Realmente ambas funciones podrían haberse fusionado como sí se ha hecho en el caso de las funciones de acciones del almacén, pero se ha preferido dejarlas separadas para mayor claridad.

4.5 INTERFAZ APLICACIÓN MÓVIL

En la aplicación Android solo se ha implementado el modo manual por la alta complejidad que supondría la programación del modo automático de forma que el archivo de código Arduino para esta interfaz solo posee una función llamada ***app*** que se llama desde el menú de modos directamente si previamente se ha escogido esta interfaz.

En esta función hay un bucle del que no se sale mientras no llegue desde la app el comando '0' que significa el fin de la conexión bluetooth y por tanto se sale al menú de interfaces. Mientras se permanece en el bucle se comprueba si llega un comando (de tipo byte) y si es así se realizan las labores oportunas según el comando que haya llegado. Estas acciones se detallan en la tabla 7. Según los valores de ciertas variables que están determinados por los comandos se ordena o no el movimiento de los diferentes motores llamando a sus funciones correspondientes de forma recursiva al estar dentro del bucle de la función.

COMANDO	SIGNIFICADO	ACCIONES
0	Fin de la conexión bluetooth	Se apaga el motor CC y se sale al menú de interfaces.
1	Extensión o recogida de la transpaleta	Se cambia la posición objetivo (30-180) del servo y se dice que el movimiento no está concluido para que el servo se mueva.
2	Encendido o apagado del motor CC	Se cambia el estado de la variable que determina si el motor CC se mueve o no.
3	Cambio de sentido del motor CC	Se cambia la variable que determina el sentido y se cambia de signo la velocidad.
4-28	Diferentes posiciones del joystick de la app que determinan el movimiento de los motores paso a paso	Según el comando que sea se establece si tienen que moverse o no los ejes vertical y horizontal. En caso afirmativo se determina su sentido de movimiento y la velocidad (hay dos valores posibles 3 y 8mm/s).
Otros (29-170)	Velocidad en rpm del motor CC determinada por una barra deslizante de la aplicación.	El valor determina la velocidad en valor absoluto con la que llamar a la función de control de posición del motor. Según el sentido se cambia de signo dicha velocidad o no.

Tabla 7. Funciones de los comandos recibidos en la función “app”

4.6 FUNCIONES DE ACCIONES DEL ALMACÉN

Este último archivo del código Arduino llamado “movimientos” es el más extenso y cuenta con todas las funciones de las que hace uso el modo automático (ya sea desde cuadro de mandos o monitor serie) para realizar los movimientos que permiten completar las acciones que solicita el usuario.

Para poder saber en qué posición se encuentra el sistema de carga/descarga lo primero que se debe hacer es calibrar llevándolo a la posición origen en la que se pulsan los dos finales de carrera. En ese momento se inicializan a 0 los contadores de pasos de los motores y ya se pueden realizar movimientos conociendo en cada instante la posición del sistema. Este procedimiento se realiza en la función ***calibrar*** que es llamada antes de hacer el reconocimiento automático o después del manual. Su diagrama de bloques se muestra en la figura 116.

La función ***reconocimiento*** es la encargada de determinar al inicio del modo los cajones ocupados y vacíos para así saber dónde se pueden meter palés y de dónde se pueden sacar. Según se indique como parámetro de la función el reconocimiento se puede hacer:

- Manual: se pregunta por el LCD o monitor serie el estado de los cajones tras lo cual se realiza la calibración.

- Automático: se calibra y se mueve la transpaleta por delante de todos los cubículos tomando medidas de distancia con el sensor para determinar la presencia o no de objetos.

Al final del reconocimiento se pregunta si los datos recogidos son correctos o hay algún error que hace necesario repetir el reconocimiento. Se puede apreciar el funcionamiento de la función con más detalle en el diagrama de flujo de la figura 117.

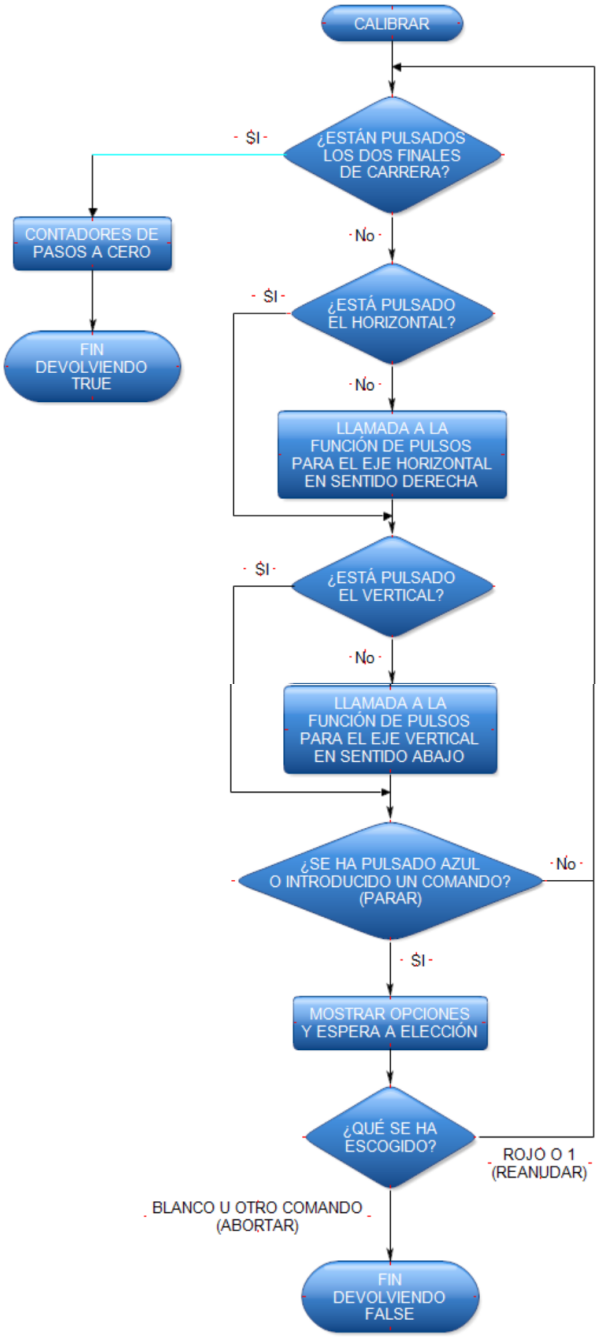


Figura 116. Diagrama de flujo de la función “calibrar”

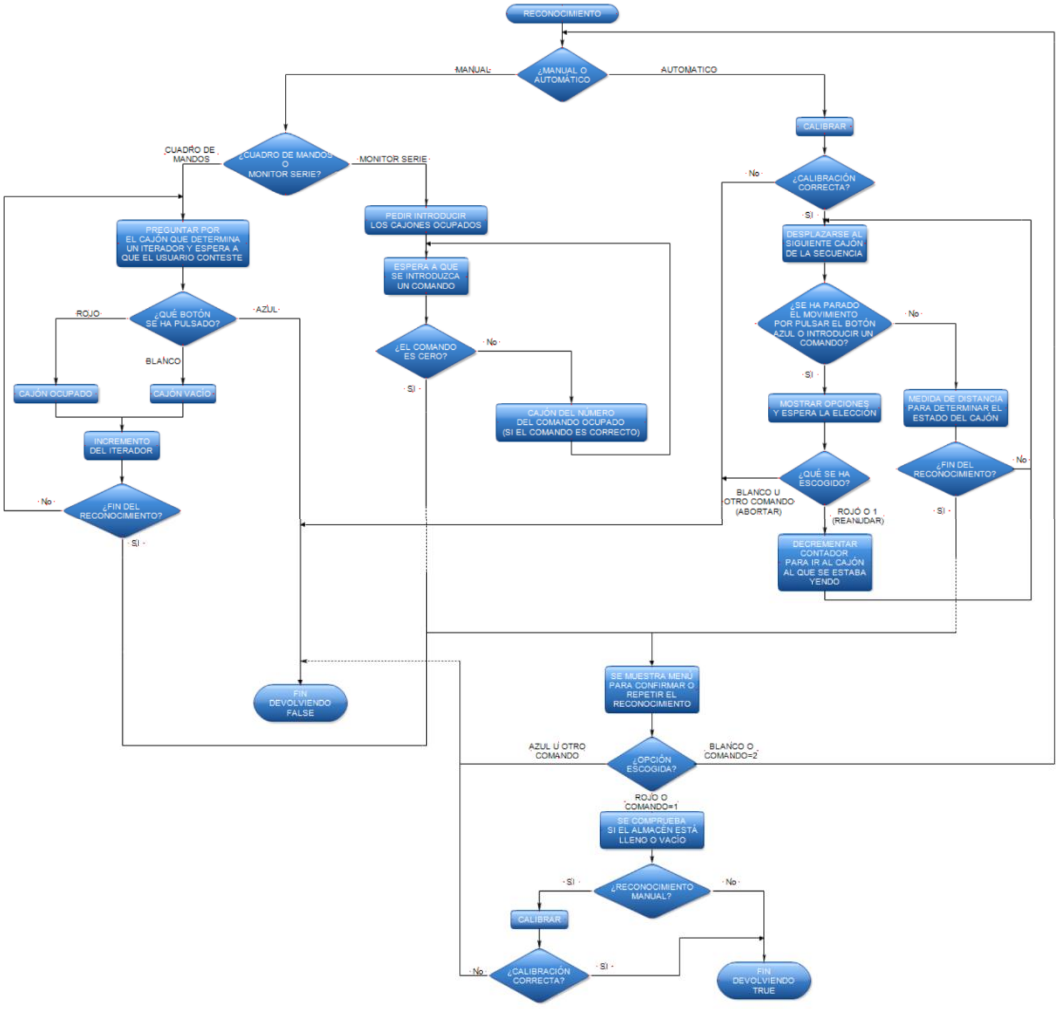


Figura 117. Diagrama de flujo de la función “reconocimiento”

La función ***coger_dejar_objeto*** lleva a cabo el proceso de coger o dejar un palé en un cubículo de la estantería o en la cinta. Para ello debe pasarse como argumento el cajón en cuestión y una variable que indique si se va a coger o a dejar. Al comienzo se asigna la posición en pasos verticales y horizontales a la que se debe ir (posición del cajón) que tendrá una ligera diferencia en la componente vertical en función de si se va a dejar o a coger. Tras ello se ordenan los siguientes cuatro movimientos secuencialmente que pueden ser interrumpidos en cualquier momento y abortar el proceso con el botón azul o introduciendo un comando (según la interfaz).

1. Movimiento de los paso a paso para situar el sistema de carga/descarga frente al cajón pedido.
2. Movimiento del servo de la posición 180 a 30 para meter la transpaleta en el cajón.
3. Ligera subida o bajada con los paso a paso verticales para coger o dejar el palé, respectivamente.

4. Movimiento del servo de la posición 30 a 180 para sacar la transpaleta del cajón.

En la figura 118 se presenta el diagrama de flujo de esta función.

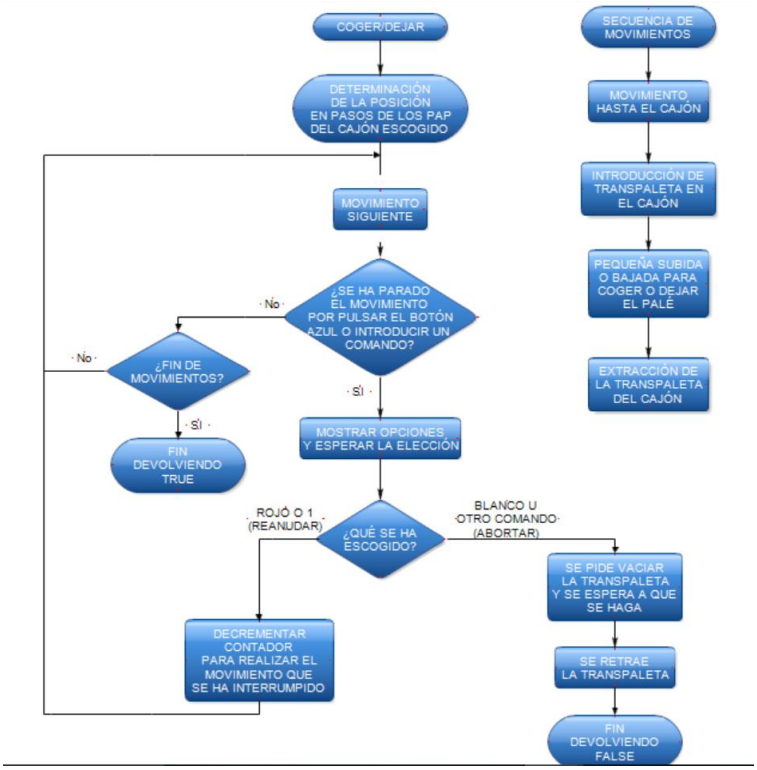


Figura 118. Diagrama de flujo de la función “coger_dejar_objeto”

La función ***meter_en_almacen*** es la que se llama cuando un nuevo objeto es detectado en la cinta o cuando se selecciona “meter objeto” habiendo uno en espera. Devuelve true si el usuario decide no meter el objeto dejándolo en espera y false si se decide meter el objeto en el almacén. No se diferencia si finalmente se mete el objeto o se aborta el proceso porque en ambos casos ya no habrá un objeto en la cinta (al abortar se pide sacar el palé manualmente) y se volverá al menú de acciones. En la función se dan tres opciones:

- Meter el objeto escogiendo el cajón en el que hacerlo.
- Meter el objeto en el primer cajón vacío que haya.
- Obviar el objeto y volver al menú de acciones.

Tras escoger se mueve la cinta para llevar el palé a la posición de recogida por el sistema de carga/descarga y se llama a la función ***meter*** que pide el cajón y mete el palé en él (si hay que escoger el cajón) o a la función ***coger_dejar_objeto*** que mete el palé sin preguntar por el cajón utilizando el

primero libre que haya. El funcionamiento completo de la función puede verse en el diagrama de la figura 119.

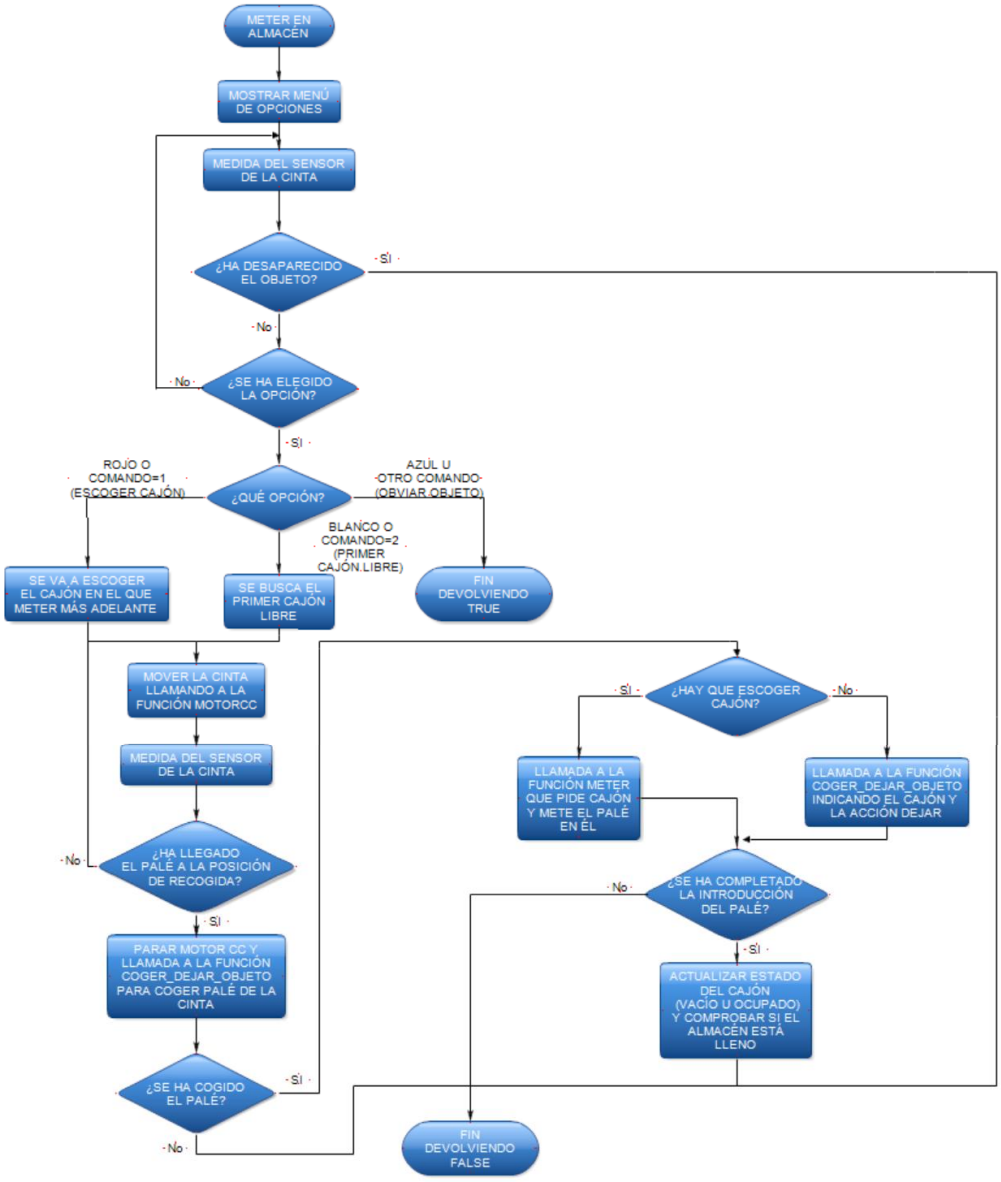


Figura 119. Diagrama de bloques de la función “meter_en_almacen”

La función ***sacar_del_almacen*** es la que se llama cuando en el menú de acciones se selecciona la acción de sacar un objeto y el almacén no está vacío. En ella se saca un palé de un cajón llamando a ***sacar*** para posteriormente llamar a ***coger_dejar_objeto*** que deposita el palé en la cinta. Por último, se mueve la cinta hasta que el palé llega al otro extremo y se solicita al usuario sacarlo del sistema. Su diagrama de bloques aparece en la figura 120.

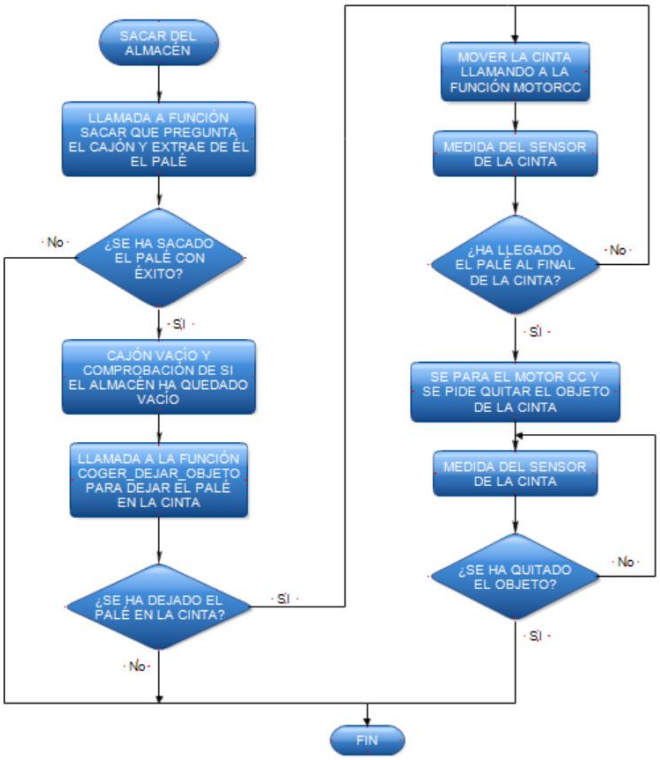


Figura 120. Diagrama de bloques de la función “sacar_del_almacen”

La tercera opción del menú de acciones es mover un palé entre dos cubículos del almacén. Al seleccionar esta acción se llama a la función ***mover_en_almacen*** que simplemente llama en primer lugar a la función ***sacar*** para sacar un palé de un cajón y a continuación a ***meter*** para meterlo en otro. Además, se actualiza el estado de los dos cajones implicados.

Las dos funciones restantes del fichero que ya se han mencionado en la explicación de otras son ***meter*** y ***sacar***, en las cuales se pide escoger un cajón y se mete o saca un palé en dicho cajón llamando a ***coger_dejar_objeto***. El cajón escogido deberá estar vacío en el caso de meter y ocupado en caso de sacar. Si no es así se volverá a pedir seleccionar un cajón hasta que se elija uno que lo esté. Si se completa el procedimiento correctamente se devuelve true y si se aborta false. Como ambas funciones son análogas en procedimiento solo se muestra el diagrama de una de ellas (sacar) en la figura 121.

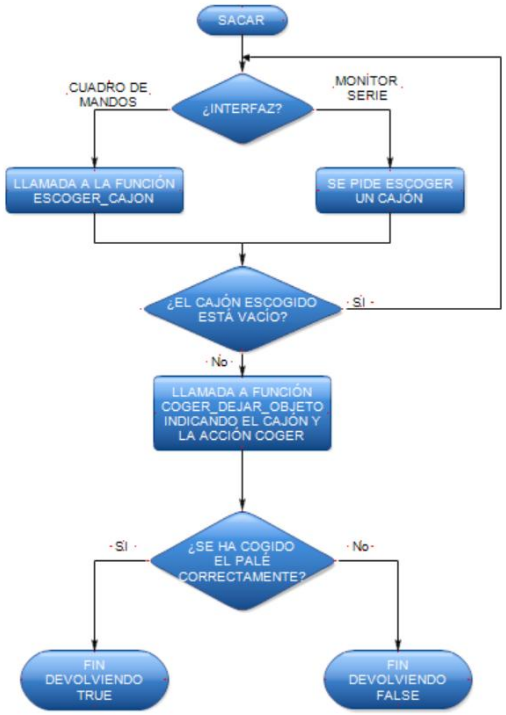


Figura 121. Diagrama de bloques de la función "sacar"

4.7 PROPUESTA DE PRÁCTICAS

Con todos los elementos con los que cuenta el sistema construido será posible realizar muchas de las prácticas de la asignatura “Mecatrónica” sobre la maqueta. Algunas ideas acerca del contenido de dichas prácticas se comentan a continuación:

- **Zumbador y botones:** para aprender a manejar el zumbador con la función “tone” de Arduino se podría pedir que se hiciera un programa en el que pulsando un botón se emitieran pitidos y con los otros dos se aumentara o disminuyera la frecuencia del sonido emitido. Se podrían hacer variantes como que el pitido se mantuviera mientras el botón siguiera pulsado o que con una pulsación empezara a pitar y con otra dejara de hacerlo.

- **LCD:** para aprender simultáneamente a manejar un LCD y a usar la función “millis” que tan útil es para la programación Arduino se

podría pedir programar uno de los menús del programa en el que se fueran sucediendo diferentes mensajes en la pantalla cada cierto tiempo.

- **Sensor de ultrasonidos:** se puede pedir programar la función "ultrasonidos" y hacer un programa que muestre en el LCD el valor de los dos sensores. A mayores se puede pedir que según un sensor detecte algo más cerca o más lejos el zumbador emita pitidos a mayor o menor frecuencia emulando a un sensor de aparcamiento.

- **Servomotor:** teniendo siempre cuidado de que la transpaleta esté en un lugar seguro, se puede comenzar por aprender a utilizar la librería "servo" de Arduino para mover al servo a diferentes posiciones de giro. Posteriormente se puede pedir programar la función "mover_servo" en la que se introduzca la posición deseada y el tiempo entre movimientos de un grado que determinará la velocidad de posicionamiento. Se deberá hacer un programa que solicite los parámetros por el monitor serie y llame a la función dentro de un bucle para que tenga efecto.

- **Motores paso a paso:** inicialmente se puede pedir realizar movimientos sencillos con los motores enviando una serie de pasos en un sentido y a continuación en el otro para familiarizarse con el funcionamiento de este tipo de motores. Una vez hecho esto se puede pedir programar una función similar a "pulso_pap" en la que los motores se muevan según los parámetros de eje, sentido y velocidad introducidos como parámetros. Se puede hacer un código que pida por el monitor serie los datos de entrada a la función y se llame a esta dentro de un bucle. Si se meten nuevos datos se actualizan los parámetros con los que se llama a la función.

- **Joystick:** Habiendo hecho la práctica anterior se puede pedir determinar los parámetros de la función "pulso_pap" programada a partir de las lecturas del joystick aprendiendo así a utilizar la función "map" de Arduino y las lecturas analógicas.

- **Motor de corriente continua:** para empezar a utilizar este actuador se puede pedir ordenar su giro a diferentes velocidades enviando diferentes valores de PWM a la señal del ENABLE, y sentidos cambiando el estado de IN1 e IN2. Una vez se tenga soltura se puede proporcionar la función "compute" del control PID y pedir programar las funciones "girar" y "contar" que no tienen demasiada dificultad, así como un programa que las llame de forma similar a como lo hacen "motorcc" o "motorcc_pos" para implementar un control PID de velocidad o posición.

- **Matlab:** para aprender a comunicar Matlab con el Arduino se puede proporcionar el código de Matlab que se ha utilizado para las

sintonías de forma que los alumnos puedan realizar diferentes pruebas de sintonías de los parámetros del PID observando y entendiendo los diferentes resultados que obtengan en las gráficas.

- **Bluetooth:** es complicado que en la asignatura haya tiempo para ver algo de programación en Android Studio pero lo que sí se puede hacer es enseñar a los alumnos a descargarse la aplicación creada y decirles los comandos que se envían por bluetooth con cada elemento de la misma. De esta forma podrán hacer un programa en Arduino que lea los comandos que lleguen desde la App y se realicen las acciones que consideren oportunas según el valor del comando recibido.

5 PROGRAMACIÓN APLICACIÓN ANDROID

Android es un sistema operativo creado por la compañía Google enfocado principalmente a los dispositivos móviles como teléfonos móviles y tabletas. Está gobernado por un *kérnel* basado en Linux que se encarga de acoplar todos los componentes del terminal para que funcionen correctamente en el sistema operativo.

Su gran éxito y aceptación en el poco tiempo de vida que tiene se debe en gran parte al hecho de que, a diferencia de sus competidores Apple y Windows Phone, es una plataforma *OpenSource* o de código abierto, de forma que es posible acceder a su código fuente. Por ello, aunque está programado en Java es posible que cualquier desarrollador cree aplicaciones con otros lenguajes como C y las compile a código de Android. [39]

Mediante el programa Android Studio u otros softwares que simplifican la programación como App Inventor, cualquier programador más o menos experimentado puede crear sus propias aplicaciones ejecutables en cualquier dispositivo con sistema operativo Android sin más que conectar el dispositivo al ordenador por USB.

5.1 CONCEPTOS GENERALES DE ANDROID STUDIO

En primer lugar, es necesario conocer el concepto de actividad. Una actividad es cada uno de los procesos de la aplicación que se compone de una interfaz gráfica (definida en el archivo .xml) y un código JAVA en el que se especifica el comportamiento de los elementos definidos en el XML. Las diferentes actividades pueden llamarse unas a otras y pasarse parámetros de forma que todas estén interrelacionadas formando el conjunto de la aplicación.

Los archivos .java se encuentran en la carpeta java del proyecto que se ubica en la siguiente ruta “nombre_proyecto\app\src\main” mientras que los .xml se sitúan en esa misma ruta dentro de *res* y *layout*.

En el archivo XML se definen y colocan en la pantalla los elementos visuales como pueden ser botones, imágenes, textos, casillas, listas… que deben estar contenidos dentro de *layouts*. Estos *layouts* son como contenedores de elementos y pueden ser de diferentes tipos (LinearLayout, RelativeLayout, FrameLayout…) según la forma de organización de sus

elementos deseada. Una actividad puede tener un único *layout* en el que se incluyan todos los elementos o varios formando estructuras de árbol. Los *layouts* y todos los elementos contenidos en ellos son objetos de la clase View, la cual implementa funciones que permiten la interacción del usuario con la aplicación como puede ser por ejemplo la función "*onClick*" en la que se define el procedimiento a realizar cuando se pulsa un botón. Para cada elemento que se incorpora en el XML se definen una serie de comandos que definen características como su color, tamaño, posición o el identificador con el que se puede acceder al elemento en el código JAVA (id).

En el archivo JAVA se programa el funcionamiento de la actividad al lanzarse y según la interacción con el usuario. El código JAVA de una actividad (tras la inclusión de las librerías necesarias) consiste en una clase del nombre de la actividad que se trata de una clase extensión de la clase *"Activity"* o "*AppCompatActivity*". Esto quiere decir que la actividad puede hacer uso de las funciones definidas en dicha clase o modificar esas funciones para sustituir o ampliar las instrucciones definidas. Para ello se utiliza "*@Override*" antes de la definición de la función.

La clase de la actividad comienza con la declaración de las variables necesarias y a continuación se definen las funciones de la clase que pueden ser extensiones de funciones de la clase "*Activity*" o funciones propias de la actividad. Al crear una actividad, por defecto se incluye la función "*onCreate*", que se llama al abrirse la actividad y en la que deben incluirse las operaciones de configuración. Por defecto viene la asociación de la clase al archivo XML correspondiente.

Algunas instrucciones de configuración típicas que se utilizan en esta aplicación son las asociaciones de variables de JAVA con elementos del XML como pueden ser "*buttons*", "*seekBars*", "*textViews*", etc. [40]

Cada aplicación tiene un hilo principal sobre el que se ejecuta la aplicación, pero es posible crear hilos secundarios que realicen algunas tareas al mismo tiempo que el hilo principal ejecuta las suyas. Para ello se utiliza la clase "*AsyncTask*" que define una tarea asíncrona ejecutada en un hilo secundario. En esta clase hay una función "*doInBackground*" en la que se ejecuta el hilo secundario y otras dos funciones "*onPreExecute*" y "*onPostExecute*" que se ejecutan en el hilo principal antes y después de la ejecución de "*doInBackground*".

Un apunte a comentar es que cuando hay un error en el código (el error se pone en rojo) si se pulsa Alt+Intro el propio programa da opciones para solucionar el error como pueden ser incluir una librería o añadir alguna función en el código.

Por último antes de empezar con la explicación de la propia aplicación realizada, se va a mencionar el archivo "*AndroidManifest.xml*" que contiene la descripción de la aplicación. En él se indican datos como las actividades de la aplicación, algunas características de ellas, los permisos necesarios, el nombre o el icono de la aplicación.

5.2 ACTIVIDAD DE ELECCIÓN DE DISPOSITIVO BLUETOOTH

Inicialmente la aplicación se programó de forma que la conexión bluetooth se establecía en la actividad principal conectando directamente con el dispositivo bluetooth del almacén. Para ello se especificaba en el código la dirección MAC del dispositivo de forma que la conexión era casi instantánea al abrir la aplicación. Sin embargo, el hecho de que en el futuro pueda haber varios almacenes iguales con diferentes dispositivos bluetooth hace que esto no sea eficiente pues para cada almacén habría que tener una versión diferente de la aplicación en la que se definiera la dirección MAC del bluetooth correspondiente.

Por tanto, se ha optado por crear una actividad llamada "*lista_bluetooth*", en la cual se muestra una lista de los dispositivos bluetooth emparejados. Al seleccionar uno de ellos se pasa a la actividad principal, en la cual se procede a intentar conectarse con dicho dispositivo.

Como la actividad que debe mostrarse al abrir la aplicación debe ser "*lista_bluetooth*", este hecho debe especificarse en el "*AndroidManifest*" de la carpeta "*manifests*". Para ello debe ser dentro de la definición de dicha actividad donde aparezca el código:

<category Android:name="Android.intent.category.LAUNCHER" />

Además, en el archivo "*AndroidManifest*" hay que incluir los permisos de uso del bluetooth para que el programa funcione.

Figura 122. Actividad "lista_bluetooth"

XML

El código XML de esta actividad cuenta con un "*RelativeLayout*" como contenedor de elementos. Con este tipo de contenedor se pueden organizar los elementos referenciando la posición de cada uno con respecto a otro, herramienta muy útil para crear la interfaz gráfica. El "*layout*" cuenta en su interior con un "*TextView*" (visualizador de texto) para pedir la selección de un almacén (bluetooth conectado a él) y un "*ListView*" (lista de elementos) en el que se muestran los dispositivos bluetooth emparejados para poder seleccionar. Además, se establece una imagen de fondo de pantalla con el comando "*background*" que será la misma que para la actividad principal. Esta imagen, así como todas las que se quisieran incorporar a la aplicación tiene que encontrarse dentro de una carpeta "*mipmap*" o "*drawable*" dentro de "res" desde donde se selecciona. El aspecto visual de esta actividad se muestra en la figura 122.

JAVA

Para el establecimiento de la conexión bluetooth, la actividad utiliza las librerías "*bluetooth.BluetoothAdapter*", "*bluetooth.BluetoothDevice*" y "*bluetooth.BluetoothSocket*", funcionando de la siguiente manera:

- Se comprueba que el dispositivo tiene bluetooth y en el caso de que sea así se pregunta al usuario si desea conectarlo (en caso de que no lo esté ya) y en caso afirmativo lo conecta. El encendido del bluetooth se realiza mediante "*startActivityForResult*", que manda ejecutar una actividad que realiza dicho cometido. Al terminar se ejecuta la función "*onActivityResult*" en la que si la conexión ha sido satisfactoria se pasa al punto siguiente.
- Ya sea tras conectar el bluetooth o directamente (si ya estaba conectado) se llama a la función "*listaDispositivosvinculados*" en donde se obtiene el nombre y dirección MAC de todos los dispositivos bluetooth vinculados al móvil o tablet y se añaden al elemento *ListView* definido en el XML, declarado e instanciado previamente en el JAVA.
- En la función de pulsación de un elemento de la lista "*myListClickListener*" se obtiene un "*string*" con la dirección MAC del elemento seleccionado y por medio de un elemento "*Intent*" se lanza

la actividad "*MainActivity*" pasándole dicho dato como parámetro mediante "*putExtra*". [41-42]

5.3 ACTIVIDAD PRINCIPAL

Una vez que se ha escogido el dispositivo bluetooth con el que conectarse en la actividad "*lista_bluetooth*", se lanza esta actividad en la que se lleva a cabo la conexión bluetooth y el control del almacén con el envío de los diferentes comandos ya definidos en el capítulo de programación Arduino.

XML

En esta actividad también se define un "*RelativeLayout*" que ocupa toda la pantalla en el que se colocan todos los elementos. Estos elementos son:

- Elemento joystick perteneciente a una clase obtenida en la referencia bibliográfica [43].
- Tres botones para mover el servo entre sus dos posiciones, encender o apagar el motor de continua y cambiar el sentido del mismo.
- Una "*SeekBar*" (barra deslizante) de valores discretos entre 0 y 141 con un elemento deslizante ("*thumb*") definido en un archivo XML dentro de la carpeta "drawable".
- Un "*TextView*" con el texto "Velocidad" para indicar el cometido de la "*SeekBar*".

La distribución de estos elementos en la pantalla se puede observar en la captura de la figura 123.

Figura 123. Actividad "MainActivity"

JAVA

La clase JAVA de la "*MainActivity*" tras la declaración de las variables que va a utilizar se compone de las siguientes funciones:

- **"onCreate":** función de configuración que se llama al iniciar la actividad. En ella se carga el XML correspondiente a la clase, se recoge la dirección MAC enviada desde la otra actividad, se ejecuta la tarea asíncrona que realiza la conexión bluetooth y se asocian todos los elementos XML a variables de la clase. Además, se establece que las pulsaciones de los botones tendrán como resultado una llamada a la función de escucha correspondiente "*onClick*" y se implementa la función de escucha de la barra deslizante.

- **"ConnectBT":** se trata de una clase extensión de "*AsyncTask*" que realiza la conexión bluetooth en un hilo secundario. En la función "onPreExecute" se muestra un mensaje por delante de la pantalla indicando que se está haciendo la conexión, en "*doInBackground*" se intenta realizar la conexión bluetooth y en "*onPostExecute*" se quita el mensaje anterior y se muestra un mensaje de duración limitada indicando si la conexión se ha establecido satisfactoriamente o no. En el caso de que no se haya establecido la conexión se finaliza la actividad volviendo a la actividad "*lista_bluetooth*".

- **"msg":** función que simplemente lanza un "*Toast*" (mensaje de duración limitada) por pantalla con el texto que se le pasa como argumento.

- **"enviar":** envía por bluetooth un dato de tipo byte que se le pasa como argumento. En caso de error muestra un mensaje indicando tal hecho por pantalla.

- **"onProgressChanged":** función incluida dentro del elemento de escucha de la barra deslizante llamada al mover la barra. En ella se transforma a byte el valor de la barra (0-141), se le suma 29 para tener un valor entre 29 y 170 y se envía dicho dato al Arduino. [44]

- **"onClick":** función de escucha de los botones en la que según el botón que se haya pulsado se envía el comando correspondiente y se cambia el texto del botón (Sacar/Meter transpaleta, Encender/Apagar el motor, Izquierda/Derecha).

- **"onJoystickMoved":** función de la clase "*JoystickView*" obtenida de la referencia bibliográfica [43] que se llama cada vez que se mueve el Joystick. En ella se establece que según la posición del Joystick se envíe un comando diferente entre 4 y 28. Solo se enviará

comando en el caso de que el último comando enviado no sea el mismo para no estar mandando datos que no tengan ningún efecto.

- **"onKeyDown":** función sobrescrita de la clase "*AppCompatActivity*" llamada al pulsar el botón de Atrás del dispositivo. En esta función por defecto se finaliza la actividad actual volviendo a la actividad que la llamó (en este caso "*lista_bluetooth").* En este caso también se va a realizar dicha acción, pero antes se ha establecido que se envíe el comando '0' (para que el Arduino sepa que finaliza la conexión) y se corte la comunicación con el dispositivo bluetooth del almacén. [45]

5.4 ADAPTACIÓN A DIFERENTES TAMAÑOS DE PANTALLA

Un problema encontrado al ejecutar la aplicación ha sido que los elementos no se visualizan de la misma forma en un móvil o en una tablet. Los XML creados para cada actividad se muestran correctamente en pantallas pequeñas como las de los móviles, pero al subir la aplicación a una tablet los elementos se visualizan muy pequeños en la parte superior.

Esto se debe a que el XML que se carga en un dispositivo debe ser diferente según el tamaño del aparato, no valiendo por ejemplo un diseño realizado con tamaño de pantalla de 5 pulgadas para un dispositivo de 10. El propio móvil o tablet detecta el archivo que debe utilizar, pero para ello es necesario programar diferentes XML con el mismo nombre adaptados a diferentes tamaños de pantalla.

Tras investigar acerca del tema se ha concluido que, a pesar de haber muchos formatos posibles de XML basados en el alto de la pantalla, ancho, ratio, tamaño, mínimo ancho..., con la creación de tres archivos diferentes ("*layout-normal*", "*layout-large*" y "*layout-xlarge*") debería ser suficiente para obtener buenos resultados en la mayoría de dispositivos.

El procedimiento ha sido crear para cada archivo XML tres copias iguales en nombre y elementos, pero con diferentes tamaños ("*normal*", "*large*" y "*xlarge*"). En cada una de ellas se establece un tamaño diferente de los elementos y se ordenan para que la visualización en pantalla sea similar sea el tamaño de dispositivo que sea. [46]

También habría que hacer archivos a mayores para el caso de girar la pantalla a modo apaisado, pero se ha preferido optar por impedir el giro de pantalla para la aplicación ya que la organización de los elementos no

quedaría muy bien estéticamente. Para ello se ha incluido en el "*AndroidManifest*" la línea

android:screenOrientation="portrait"

en cada una de las dos actividades, con la que se establece que la aplicación siempre se encuentre en vertical. [47]

6 ESTUDIO ECONÓMICO

En este capítulo se va a desarrollar el aspecto económico del TFG.

Es necesario conocer la viabilidad de un proyecto y por eso, es importante llevar a cabo un estudio de los costes que acompañe a los aspectos técnicos y teóricos.

En los siguientes apartados se especifican los correspondientes costes directos e indirectos que se necesitan asumir para la realización del proyecto.

6.1 RECURSOS EMPLEADOS

En la elaboración de este trabajo se han utilizado diversos recursos que se van a diferenciar en tangibles e intangibles.

Dentro de los recursos tangibles se pueden señalar los correspondientes a equipos informáticos y de fabricación que aparecen en la tabla 8.

EQUIPOS INFORMÁTICOS Y DE FABRICACIÓN	PRECIOS
Ordenador portátil Lenovo Yoga	997 €
Impresora 3D BQ Prusa I3 Hephestos	429 €
Impresora 3D Velleman K8200	477 €
TOTAL COSTE	**1.903 €**

Tabla 8. Coste de equipos informáticos y de fabricación

También hay que mencionar aquellos que han sido utilizados como herramientas de trabajo en la fabricación (tabla 9) y en las pruebas electrónicas (tabla 10).

HERRAMIENTAS DE FABRICACIÓN	PRECIOS
Sierra Ingletadora de madera	210 €
Amoladora eléctrica	119 €
Tornillo de banco	22 €
Taladro Steiner 12V	38 €
Cúter, regla, flexómetro, calibre	32 €
Llave inglesa y llaves Allen	19 €
Destornilladores	5 €
TOTAL COSTE	**445 €**

Tabla 9. Coste de herramientas de fabricación

EQUIPO ELECTRÓNICO	PRECIOS
Placa de pruebas	7 €
Polímetro	15 €
Soldador	15 €
Fuente de alimentación de ordenador	20 €
TOTAL COSTE	**57 €**

Tabla 10. Coste del equipo electrónico

En cuanto a los recursos intangibles se pueden nombrar los siguientes programas informáticos:

- Arduino IDE
- Matlab 2015
- Autodesk Inventor
- Cura
- Microsim 8
- Android Studio
- Microsoft Office Profesional 2016

Son programas que se descargan gratuitamente, a excepción de:

- Matlab 2015 cuyo precio es de 83,50 €
- Autodesk Inventor que supone una suscripción mensual de 260 €
- Microsoft Office Profesional 2.016, comprado por 539 €

Todos estos recursos no son utilizados exclusivamente en la realización de este proyecto por lo que a la hora de considerar su coste aplicable a él se tiene en cuenta el tiempo utilizado para calcular la amortización correspondiente.

6.2 COSTES DIRECTOS

Los costes directos son aquellos que se pueden asignar fácilmente al proyecto realizado, dado que intervienen de manera directa en su elaboración.

Deben diferenciarse en tres grupos diferentes:

- Coste de personal
- Coste de amortización de los recursos utilizados
- Coste de materiales empleados
- Coste de consumibles

6.2.1 COSTES DE PERSONAL

Para la realización de este proyecto se ha necesitado un ingeniero que llevara a cabo todo el proceso de diseño mecánico, diseño electrónico, montaje de la maqueta y programación de los códigos Arduino y Android.

Para calcular el coste de este trabajo se establece en primer lugar el sueldo bruto anual de un ingeniero y se le añade la seguridad social que la empresa pagará por él (35% del sueldo bruto) obteniendo en la tabla 11 el coste total anual de un ingeniero.

COSTE ANUAL DE UN INGENIERO	
Sueldo anual bruto	30.000 euros
Seguridad social	10.500 euros
COSTE TOTAL	**40.500 euros**

Tabla 11. Coste anual de un ingeniero

Una vez calculado el coste de un ingeniero durante un año, es necesario conocer el número de horas trabajadas en el mismo atribuibles al proyecto.

Según el Estatuto de los trabajadores, los trabajadores en España deben realizar un máximo de 40 horas semanales. Esta normativa regula el derecho a 30 días naturales de vacaciones, lo que supone más o menos 4 semanas de inactividad laboral. Restando estas 4 semanas de las 52 semanas que tiene un año se concluye que el profesional trabaja durante 48 semanas a lo largo de un año.

Teniendo en cuenta que en España se tiene derecho a 14 días de fiestas laborales entre las nacionales, autonómicas y locales, se deben descontar otros 14 días. [48]

Sabiendo que la jornada diaria es de 8 horas, se puede contabilizar:

48 semanas x 5 días x 8 horas = 1920 horas

Se deducen las horas de los días festivos:

14 días x 8 horas = 112 horas

Las horas computables como horas a trabajar al año en España son de:

1920 - 112 = 1808 horas

Una vez obtenida la cifra anual hay que calcular el coste por hora trabajada.

Coste por hora trabajada = 40.500 €/1.808 horas = 22,40 euros por hora de trabajo.

En la tabla 12 se presenta de forma orientativa el número de horas empleadas en las diferentes actividades realizadas para el desarrollo del presente proyecto.

DISTRIBUCIÓN TEMPORAL DEL TRABAJO	
Estudio del problema	70 horas
Diseño mecánico	130 horas
Diseño electrónico	110 horas
Construcción maqueta	90 horas
Programación Arduino	130 horas
Programación Android	110 horas
Elaboración de la documentación	110 horas
TOTAL	**750 HORAS**

Tabla 12. Distribución temporal de las tareas realizadas

Por tanto, el coste final de personal es de:

750 horas x 22,40 euros/hora = 16.800 euros

6.2.2 COSTES DE AMORTIZACIÓN

Para poder calcular estos costes se valoran los bienes amortizables según los precios detallados en el primer apartado.

Después se utilizan las tablas de coeficientes de amortización lineal de la Agencia Tributaria teniendo en cuenta el coeficiente máximo a aplicar para cada tipo de elemento según el tiempo utilizado, determinando finalmente el empleo de los coeficientes de amortización de la tabla 13. [49]

ELEMENTOS A AMORTIZAR	COEFICIENTE DE AMORTIZACIÓN
Equipos informáticos	25%
Programas informáticos	33%
Equipos electrónicos	20%
Herramientas	25%

Tabla 13. Coeficientes de amortización

En la tabla 14 se pueden consultar los precios con su correspondiente amortización.

ELEMENTOS AMORTIZABLES	PRECIO DE LOS ELEMENTOS	COSTE AMORTIZACIÓN
Equipos informáticos	1.903 €	475,75 €
Programas informáticos	622,50 €	205,42 €
Equipos electrónicos	57 €	11,40 €
Herramientas	445 €	111,25 €
TOTAL		**803,82 €**

Tabla 14. Costes de elementos amortizables

No se ha llevado a cabo la amortización del programa informático Autodesk Inventor ya que no se tiene en propiedad, sino que se paga por el derecho de uso mensualmente. Por ello se considera que constituye un coste indirecto que no se restringe a este proyecto.

Estas amortizaciones le corresponderían a un año de trabajo (1808 horas), por lo que hay que calcular lo correspondiente al tiempo utilizado en el proyecto.

Gastos de amortización asumibles por el proyecto:

750 horas x 803,82/1.808 = 333,44 €

6.2.3 COSTES DE MATERIALES EMPLEADOS

En este apartado se detallan los materiales utilizados para construir la maqueta que no son amortizables. Todos ellos, así como los precios que han supuesto y sus proveedores se detallan en la tabla 15.

MATERIALES	PRECIOS	PROVEEDOR
Tablero DM	12 €	Leroy Merlin
20 espigas de madera	3 €	Leroy Merlin
Tornillos, tuercas, arandelas	12 €	Ferretería Ortiz
Tubo fontanería Ø3mm 17cm	0,15 €	Ferretería Ortiz
Tubo en U aluminio 6cm 86cm	4,6 €	Aludevall
Varilla acero Ø3mm 1m	1,8 €	Biplano Aeromodelismo
Varilla acero Ø6mm 1m	3,2 €	Biplano Aeromodelismo
Tablero contrachapado 4mm 40x60cm	3,75 €	Oca-Dido
Goma Eva 2mm 70x47,5cm	1,4 €	Oca-Dido

Tornillo avance 400mm + tuerca + 2 rodamientos	9,44 €	Amazon
2 Tornillos avance 300mm + tuerca + 2 rodamientos	17,94 €	Amazon
2 Rollos PLA para impresora 3D	41,04 €	Amazon
Fuente de alimentación 12V 6A	12,99 €	Amazon
3 Motores paso a paso 42BYG	59,85 €	Makeblock
Motor CC EMG30	37,35 €	SuperRobotica
2 Soportes motores PAP verticales	11,86 €	Bricogeek
3 disipadores de aluminio	3,99 €	Bricogeek
Arduino Mega	41,75 €	Bricogeek
Bluetooth HC05	8,71 €	Bricogeek
Zumbador piezoeléctrico	2,42 €	Bricogeek
3 Controladores de motores A4988	25,41 €	Bricogeek
Servo SG5010 o 3001HB	14,76 €	Bricogeek
Soporte motor PAP horizontal	6,94 €	Todoelectrónica
Soporte motor CC	5,95 €	Todoelectrónica
80 cables Dupont 20cm macho-hembra	3,42 €	Todoelectrónica
40 cables Dupont 20cm macho-macho	1,82 €	Todoelectrónica
2 Ultrasonidos HC-SR04	5,38 €	Todoelectrónica
LCD	3,61 €	Todoelectrónica
Joystick + placa joystick	6,9 €	Todoelectrónica
3 Botones	0,75 €	Todoelectrónica
Placa fotosensible	6,69 €	Todoelectrónica
Controlador de motores L298N	10,9 €	Todoelectrónica
Interruptor palanca 1 canal	1,35 €	Electroson Castilla
Interruptor palanca 2 canales	1,77 €	Electroson Castilla
14 conectores para PCB	8,54 €	Electroson Castilla
2 tiras de 40 pines macho-hembra	2,5 €	Electroson Castilla
1 tira de 40 pines macho-macho	1,25 €	Electroson Castilla
2 Potenciómetros 10K	1,72 €	Electroson Castilla
2 resistencias 220Ω	0,1 €	Electroson Castilla
2 Finales de carrera	3,76 €	Electroson Castilla
3 Condensadores 100µF	0,63 €	Electroson Castilla
Transistor NPN	0,12 €	Electroson Castilla
7 Condensadores 100nF	0,91 €	Electroson Castilla
Conector Jack PCB 2,5mm	1,64 €	Farnell
3 acoples flexibles 4-8mm	6,92 €	Farnell
Gastos de envío	25 €	
Honorarios trabajos aluminio	30 €	
Honorarios trabajo PCB	45 €	
Coste total	**279,31 €**	

Tabla 15. Costes de materiales utilizados

6.2.4 COSTES DE CONSUMIBLES

En este apartado se tienen en cuenta todos los materiales que no se han detallado anteriormente, pero han sido también necesarios para la realización del proyecto. Materiales como pegamento, cola, cartuchos de tinta, estaño, bolígrafos, folios y demás material de oficina.

El coste aproximado de todo esto se supone de 50 euros.

6.2.5 COSTES DIRECTOS TOTALES

Los costes directos desglosados en los apartados anteriores, así como la suma total se presenta en la tabla 16.

COSTES DIRECTOS TOTALES	
COSTES DE PERSONAL	16.800 €
COSTES DE AMORTIZACIÓN	333,44 €
COSTES DE MATERIALES	279,31 €
COSTES CONSUMIBLES	50 €
COSTES DIRECTOS TOTALES	**17.462,75 €**

Tabla 16. Costes directos totales

6.3 COSTES INDIRECTOS

Estos costes hacen referencia a aquellos gastos necesarios para que se desarrolle el proyecto pero que no pueden considerarse costes directos. En estos gastos se tiene en cuenta el consumo eléctrico (alimentación de dispositivos electrónicos, calefacción, luz…), el consumo telefónico (Internet) y los servicios adicionales (desplazamientos, suscripción a programas, etc) tal y como aparece en la tabla 17.

COSTES INDIRECTOS	
Consumo eléctrico	120 €
Consumo telefónico (Internet)	80 €
Servicios adicionales	85 €
COSTE TOTAL	**285 €**

Tabla 17. Costes indirectos

6.4 COSTES TOTALES

Una vez calculados todos los costes directos e indirectos, basta con sumarlos para obtener el coste total del proyecto que se muestra en la tabla 18.

COSTES TOTALES	
COSTES DIRECTOS	17.539,24 €
COSTES INDIRECTOS	285 €
TOTAL COSTES	**17.824,24 €**

Tabla 18. Coste total del proyecto

7 CONCLUSIONES Y LÍNEAS FUTURAS

Para terminar con la explicación del Trabajo Fin de Grado realizado, en este capítulo se exponen las conclusiones que se han podido sacar de la realización del proyecto, los objetivos cumplidos y las posibles líneas futuras que podrían dar lugar a mejoras o ampliación de funcionalidades del sistema.

Con la realización de este proyecto se ha podido aprender y coger experiencia en numerosos campos de la ingeniería ya que al tratarse de un proyecto mecatrónico se ha tenido que trabajar en todas las ramas que engloban dicha disciplina como son la mecánica, la electrónica, la programación y el control. Por ello se considera que con el TFG se ha cumplido con los cometidos de ganar madurez en el desarrollo de proyectos de ingeniería y de integrar los conocimientos y capacidades adquiridos a lo largo de la titulación, así como otros conocimientos específicos para el trabajo.

El objetivo principal del TFG era crear un sistema mecatrónico que sirviera como base para la realización de las prácticas de la asignatura “Mecatrónica” a la vez que tuviera una entidad en sí mismo para poder ser manipulado por cualquier tipo de usuario para cuyo cumplimiento se han ido logrando los siguientes hitos:

- Diseño mecánico de una maqueta de un almacén automatizado formado por una estantería y tres mecanismos accionados por diferentes tipos de motores. Mediante un programa CAD se han diseñado las piezas necesarias y se ha realizado un ensamblaje de todo el sistema para determinar la ubicación exacta de todos los elementos, ir corrigiendo fallos y modificando los elementos hasta tener el diseño final que llevar a la realidad.

- Diseño de un esquema electrónico que incluye numerosos dispositivos de interacción con el usuario, sensores y actuadores además de un microcontrolador que se integran con la mecánica para formar un todo. Se ha conseguido diseñar una placa de circuito impreso que cumple con lo necesitado para el sistema, consiguiendo una distribución de elementos, conectores y cables bastante ordenada, no sin dificultades debido al gran número de elementos electrónicos, su ubicación en la maqueta y la distribución de las entradas/salidas del microcontrolador.

- Construcción de los sistemas mecánicos y electrónicos diseñados hasta conseguir el montaje completo con un buen aspecto visual y todas las funcionalidades estipuladas. Algunas piezas ha sido necesario repetirlas por algún fallo de la impresora o del diseño y la

fijación de ciertos elementos ha dado algunos problemas pero finalmente se considera que el resultado alcanzado ha sido bastante satisfactorio.

- Desarrollo del programa del microcontrolador que gobierna el sistema dotando al dispositivo de diferentes funcionalidades y posibilidades de control. El código se ha estructurado de tal forma que puede utilizarse fácilmente como apoyo en las prácticas de la asignatura con múltiples archivos y funciones agrupadas según su cometido. A pesar de los típicos problemas de integración entre el hardware y el software se ha conseguido implementar la mayoría de las funcionalidades que se deseaba consiguiendo un buen resultado.

- Programación de una aplicación Android para el manejo del almacén desde un móvil o tablet. Este objetivo ha supuesto uno de los mayores retos debido a que en la titulación no se ha visto nada acerca de este tipo de programación. Por ello, ha sido necesario bastante tiempo de investigación para finalmente desarrollar una sencilla aplicación con la que manejar el almacén en modo manual de forma equivalente a como se hace desde el cuadro de mandos. El modo automático no ha sido posible implementarlo porque requiere de conocimientos más avanzados para cuya adquisición sería necesario una importante inversión de tiempo.

Se espera que con el almacén construido se consiga contribuir al aprendizaje de los futuros alumnos de la asignatura “Mecatrónica” y a fomentar su interés y motivación por los campos con los que se relaciona.

Como líneas futuras, destaca la posibilidad de implementar el modo automático en la aplicación Android para lo cual habría que hacer una aplicación bastante más compleja con múltiples actividades y/o fragmentos, tareas asíncronas y una perfecta sincronización entre los comandos que enviara la aplicación y los que enviara el Arduino.

En la misma línea se podría incorporar un módulo WiFi al sistema como por ejemplo un ESP8266 de forma que también se pudiera manejar el almacén desde un servidor web o desde la propia aplicación sin la limitación de distancia que supone el bluetooth.

En cuanto a la parte física podría ser conveniente para futuras versiones elevar ligeramente la estantería debido a que cuando el sistema de carga/descarga se encuentra en los cajones de la fila inferior en ocasiones el cable del sensor se engancha con elementos de la placa como el módulo bluetooth. Gracias a la pequeña pieza que guía estos cables se ha conseguido reducir este problema, pero no se ha eliminado por completo.

También sería conveniente incluir un sistema de sujeción de las cargas encima de los palés porque con las vibraciones del sistema al moverse, a menudo se desplazan pudiendo llegar a caerse en trayectos largos.

8 BIBLIOGRAFÍA

Las fuentes consultadas para la realización del TFG se presentan a continuación dividiéndolas entre los diferentes capítulos de los que ha formado el informe.

Capítulo 1: Introducción y objetivos

Marco del trabajo y justificación del TFG

[1] Entrada de Wikipedia sobre Mecatrónica.

https://es.wikipedia.org/wiki/Ingenier%C3%ADa_mecatr%C3%B3nica

[2] Guía docente de la asignatura "Mecatrónica" del Grado en Electrónica Industrial y Automática.

Estado del arte: Los almacenes automáticos

[3] Web de la empresa Mecalux

https://www.mecalux.es/

[4] Móstoles Industrial S.A. "Almacenes Automáticos"

http://www.moinsa.es/moinsa/a3W5/d/pdf/log/AlmacenesAutomaticos.pdf

[5] Zona Logística "Los Almacenes Automáticos"

http://www.zonalogistica.com/articulos-6681/articulos-mas-leidos/los-almacenes-automaticos/

[6] Noega Systems "Almacenes robotizados: Técnicas de manutención y almacenaje"

https://www.noegasystems.com/blog/almacenamiento/almacenes-robotizados

[7] Web de ULMA Handling Systems

http://www.ulmahandling.com/es/

[8] Cadena de suministros “Sistemas de carrusel”

https://cadenadesuministros.wordpress.com/bodegas-automatizadas/5-2-sistemas-de-carrusel/

[9] Web de System Logistics

http://www.systemlogistics.com/spa/

[10] Web de SSI Schaefer

http://www.ssi-schaefer.es/

[11] Web de Westfalia USA

https://www.westfaliausa.com/es

[12] Web de Daifuku

http://www.daifuku.com/

Capítulo 2: Diseño mecánico

[13] 3D CAD Portal “Diseño Paramétrico - Modelado Paramétrico”

http://www.3dcadportal.com/diseno-parametrico-modelado-parametrico.html

[14] Createc 3D “ABS vs PLA ¿Qué material utilizamos?

https://createc3d.com/abs-vs-pla-que-material-utilizamos/

Base y estantería

[15] Entrada de Wikipedia “Tablero de fibra de densidad media”

https://es.wikipedia.org/wiki/Tablero_de_fibra_de_densidad_media

Sistema AS/RS

[16] Apuntes de la asignatura Mecatrónica sobre motores y mecanismos.

[17] 330 Ohms “Motores a pasos ¿unipolares o bipolares?

https://www.330ohms.com/blogs/blog/85507012-motores-a-pasos-unipolares-o-bipolares

Cinta transportadora

[18] Apuntes de la asignatura Mecatrónica sobre sensores, motores y mecanismos.

[19] Robot electronics "EMG30, mounting bracket and wheel specification"
https://www.robot-electronics.co.uk/htm/emg30.htm

Capítulo 3: Diseño electrónico

Microcontrolador

[20] Web de Arduino
https://www.arduino.cc/

[21] AJPD Soft "Tipos de memoria en el microcontrolador de Arduino, Flash, SRAM y EEPROM"
http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=571

[22] Aprendiendo Arduino "Bootloader"
https://aprendiendoarduino.wordpress.com/tag/bootloader/

Controladores de motores

[23] Hoja de datos del controlador L298N
https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

[24] Hoja de datos del controlador L293D
http://www.ti.com/lit/ds/symlink/l293.pdf

[25] Hoja de datos del regulador de tensión LM2596
http://www.ti.com/lit/ds/symlink/lm2596.pdf

[26] Electronilab "Uso de Driver L298N para motores DC y paso a paso con Arduino"
https://electronilab.co/tutoriales/tutorial-de-uso-driver-dual-l298n-para-motores-dc-y-paso-a-paso-con-arduino/

[27] Apuntes de la asignatura "Mecatrónica" sobre control de motores CC.

[28] Luis Llamas "Motores paso a paso con Arduino y driver A4988 o DRV8825"
https://www.luisllamas.es/motores-paso-paso-arduino-driver-a4988-drv8825/

[29] StaticBoards "La comparativa definitiva de los drivers para motores paso a paso: DRV8825 vs A4988"
https://www.staticboards.es/blog/drv8825-vs-a4988/#Driver_para_un_motor_paso_a_paso_unipolar

[30] Pololu "A4988 Stepper Motor Driver Carrier" y hoja de datos
https://www.pololu.com/product/1182

[31] dima3D "Motores paso a paso en impresión 3D: Calibración de corriente"
http://www.dima3d.com/motores-paso-a-paso-en-impresion-3d-iii-calibracion-de-corriente/

Sensores

[32] Hoja de datos del sensor de ultrasonidos HC-SR04
http://www.micropik.com/PDF/HCSR04.pdf

[33] Hoja de datos del sensor de infrarrojos GP2Y0A21YK0F
http://www.sharp.co.jp/products/device/doc/opto/gp2y0a21yk_e.pdf

Elementos señalizadores

[34] Hoja de datos de la pantalla LCD de Sparkfun
https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

Bluetooth

[35] Wayne's Tinkering Page "Inexpensively Program your Arduino via Bluetooth"
https://sites.google.com/site/wayneholder/inexpensively-program-your-arduino-via-bluetooth

Alimentación

[36] Hoja de datos del conector Jack para PCB
http://www.farnell.com/datasheets/1855458.pdf?_ga=2.145348151.249550035.1497465720-1194689601.1467484589

Diseño de la PCB

[37] Granabot "Las reglas generales de diseño de las pistas de la PCB"
http://www.granabot.es/Modulos/dpe/Apuntes/Tema%201.6.5.pdf

[38] Fº Javier Alexandre "Algoritmo para el cálculo del ancho de pista de una placa de circuito impreso"
https://cuningan.files.wordpress.com/2010/10/calculos-ancho-pista.pdf

Capítulo 5: Programación Aplicación Android

[39] ConfigurarEquipos "Qué es Android: Características y Aplicaciones"
http://www.configurarequipos.com/doc1107.html

Conceptos generales

[40] Tutorial de Android Studio de "Tutoriales Programación Ya"
http://www.tutorialesprogramacionya.com/javaya/androidya/androidstudioya/

Bluetooth

[41] Instructables "How to: Create an Android App With Android Studio to Control LED"
http://www.instructables.com/id/Android-Bluetooth-Control-LED-Part-2/

[42] Tutorialspoint "Android-Bluetooth"
https://www.tutorialspoint.com/android/android_bluetooth.htm

Joystick

[43] Instructables "A simple Android UI Joystick"
http://www.instructables.com/id/A-Simple-Android-UI-Joystick/

Barra deslizante

[44] Java Code Geeks “Android SeekBar Example”
https://examples.javacodegeeks.com/android/core/widget/seekbar/android-seekbar-example/

Botón de atrás

[45] Curso Android Studio “ir Atrás botón onKeyDown”
http://cursoandroidstudio.blogspot.com.es/2014/09/ir-atras-boton-onkeydown.html

Diferentes dispositivos

[46] java Hispano “Crear aplicaciones compatibles con múltiples pantallas”
http://www.javahispano.org/android/2012/6/4/crear-aplicaciones-compatibles-con-multiples-pantallas.html

[47] Curso Android Studio “Bloquear orientación de pantalla”
http://cursoandroidstudio.blogspot.com.es/2014/11/bloquear-orientacion-de-pantalla.html

Capítulo 6: Estudio económico

[48] Artículos 34, 37 y 38 del Estatuto de los Trabajadores.

[49] Tabla de coeficientes de amortización lineal de la Agencia Tributaria.

ANEXOS

ANEXO 1: MANUAL DE USUARIO

En este anexo se incluye una breve guía de utilización del sistema construido para que cualquier usuario pueda manipularlo sin problemas.

Encendido y apagado del aparato

Para encender el sistema se debe conectar la fuente de alimentación a la red y al conecto Jack de la PCB situado en la esquina trasera izquierda de la placa.

Tras ello bastará con poner el interruptor "12V" situado en la parte delantera izquierda de la maqueta en la posición *"ON"* para que el sistema se encienda. En el caso de conectar el Arduino al ordenador por USB para manejar el almacén desde el monitor serie del IDE de Arduino, el interruptor "5V" de la derecha deberá estar en la posición *"OFF"* para que no se alimente el módulo bluetooth que impediría la comunicación serie con el ordenador. En caso contrario el interruptor deberá estar a *"ON"* para permitir la alimentación lógica de 5V.

Para el apagado del equipo únicamente es necesario conmutar el interruptor "12V" a la posición *"OFF"*.

Utilización

El sistema está programado de forma que su manejo sea sumamente sencillo e intuitivo se use la interfaz de control que se use. Inicialmente hay que escoger con los botones del cuadro de mandos la interfaz de control que se va a utilizar según las opciones mostradas en la pantalla LCD situada en la torre de aluminio izquierda. A partir de ahí ya se puede utilizar la interfaz escogida.

Cuadro de mandos

Para el control desde el cuadro de mandos simplemente hay que observar los mensajes que se muestran en la pantalla LCD y escoger una de las opciones que se ofrecen en cada momento por medio de los botones y el joystick. En el modo manual se podrán manejar los movimientos de cada uno de los motores de la siguiente forma:

- Botón rojo: cada pulsación hace que con el joystick se cambie entre el manejo del movimiento del sistema de carga/descarga por el almacén y el movimiento de la cinta.

- Botón blanco: extiende o retrae la transpaleta según la posición en en la que se encuentre.
- Botón azul: sale del modo manual.
- Joystick: maneja en velocidad y dirección tanto el sistema transelevador como la cinta transportadora según las pulsaciones del botón rojo.

Si se escoge el modo automático en primer lugar hay que escoger el modo de reconocimiento: en el manual hay que ir introduciendo con los botones el estado de cada cajón entre ocupado o vacío mientras que en el automático el propio sistema se va moviendo por el almacén detectando si hay objetos o no en cada uno de los cajones. Tras el reconocimiento se podrá escoger una acción del menú del modo automático de entre las siguientes:

- Botón rojo: sacar un elemento del almacén por la cinta transportadora.
- Botón blanco: mover un elemento de un cajón a otro del almacén.
- Botón azul: salir del modo automático.
- Botón del joystick: meter un objeto en el almacén en caso de que lo haya en la cinta esperando.

Como medida de seguridad, cuando el sistema está moviéndose realizando alguna acción del modo automático se puede detener el movimiento pulsando el botón azul, tras lo cual se podrá seleccionar reanudar el movimiento o abortar la instrucción.

En caso de estar en el modo automático y querer introducir un nuevo palé en el almacén no hay más que colocarlo al final de la cinta del lado correcto para que la transpaleta pueda recogerlo. El sensor lo detectará y en la pantalla aparecerá un menú de opciones en el que se podrá elegir entre introducirlo en un cajón que se escoja (rojo), introducirlo en el primer cajón libre (blanco) o no introducirlo (azul). Si se escoge esta última, en el menú general del modo aparecerá la opción de meter el palé mientras siga estando en la cinta.

Cada vez que haya que escoger un cajón para meter o sacar un palé se pide seleccionar con los botones la fila y a continuación la columna en la que se encuentra el cajón deseado.

Monitor serie

En caso de utilizar el monitor serie, tras conectar el Arduino al ordenador por un cable USB, hay que abrir en el IDE de Arduino el monitor del puerto COM correspondiente a la conexión y seleccionar la velocidad de 115200 baudios. Tras ello se debe pulsar el botón blanco para escoger esta interfaz y

en ese momento ya aparecerá el menú de modos en la pantalla del ordenador. No es posible escoger la interfaz con el botón blanco antes de abrir el monitor serie dado que, con el establecimiento de la conexión producido al abrir el monitor, el Arduino se resetea y habría que volver a escoger la interfaz.

A partir de ahí el procedimiento es muy similar al explicado para el cuadro de mandos, pero ahora en lugar de escoger las opciones con botones se hace con la introducción de números por el teclado del ordenador.

Al no disponer de un elemento como el joystick para controlar la velocidad de los motores, cuando se seleccione algún movimiento en el modo manual a continuación hay que seleccionar la velocidad deseada para el movimiento dentro de un rango especificado. Para parar un movimiento que se haya ordenado únicamente se debe introducir un comando cualquiera por el teclado.

En el modo automático el reconocimiento manual se hace introduciendo los cajones ocupados directamente sin tener que especificar el estado de cada uno secuencialmente por lo que el procedimiento es más rápido que con el cuadro de mandos. Por lo demás es todo muy similar y para detener un movimiento basta con introducir un comando.

Aplicación Android

Para poder controlar el almacén desde el móvil o tablet en primer lugar hay que descargarse la aplicación para lo cual hay que conectar el terminal al ordenador y en Android Studio dar a ejecutar la aplicación en el dispositivo conectado.

A continuación, se debe emparejar el módulo bluetooth del almacén con el bluetooth del dispositivo. Para ello hay que entrar al bluetooth, esperar a que aparezca el HC05 que tiene por nombre “Almacén” e introducir la contraseña 1234. En el caso de que se hayan construido varias maquetas, el bluetooth de cada una habrá sido configurado con un nombre y contraseña que habrá que conocer.

Una vez emparejados y con el bluetooth activado (si no es así se podrá activar desde la aplicación) se entra a la aplicación y se selecciona de la lista que aparece el dispositivo bluetooth correspondiente. Si la conexión es satisfactoria aparece un mensaje que dice “Conectado” y se pasa a una segunda pantalla con un joystick, varios botones y una barra deslizante. En ese momento ya se puede seleccionar la interfaz con el botón azul del cuadro de mandos y empezar a manejar el almacén. Al establecerse la conexión el

Arduino se resetea así que al igual que con el monitor serie no tiene sentido escoger la interfaz antes de este momento.

Con el joystick de la parte superior se maneja el movimiento del transelevador al igual que se hacía con el cuadro de mandos aunque en este caso solo hay dos posiciones posibles de velocidad. Pulsando el botón verde de debajo se extiende o retrae la transpaleta según dónde se encuentre.

Los botones azules sirven para encender o apagar el motor de la cinta y para cambiar el sentido de movimiento entre izquierda y derecha. Por último, con la barra deslizante de abajo se modifica la velocidad del motor entre 29 y 170rpm.

Pulsando el botón de atrás del dispositivo la conexión finalizará y se volverá a la pantalla anterior en la que se podrá escoger la conexión a otro dispositivo.

ANEXO 2: CÓDIGO ARDUINO

En este anexo se incorporan por este orden: las funciones de MATLAB para la sintonía de los PIDs, el programa Arduino para la configuración de módulos bluetooth HC05 y el programa Arduino completo dividido en sus diferentes archivos.

Función de MATLAB para la sintonía del PID de velocidad

```
function Matlab_Arduino(numero_muestras)
% Matlab + Arduino Serial Port communication
close all;
clc;
y=zeros(1,1000); %Vector donde se guardarán los datos
%Inicializo el puerto serial que utilizaré
delete(instrfind({'Port'},{'COM10'}));
puerto_serial=serial('COM10');
puerto_serial.BaudRate=9600;
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%Abro el puerto serial
fopen(puerto_serial);
%Declaro un contador del número de muestras ya tomadas
contador_muestras=1;
%Creo una ventana para la gráfica
figure('Name','Serial communication: Matlab + Arduino.')
title('Ensayo sintonía PI motor CC');
xlabel('Tiempo(segundos)');
ylabel('Velocidad (rpm)');
grid on;
hold on;
%Bucle while para que tome y dibuje las muestras que queremos
while contador_muestras<=numero_muestras
 ylim([0 130]); %Rango de velocidades a visualizar
 xlim([0 numero_muestras/10]);
 y(contador_muestras)=fscanf(puerto_serial,'%f')';
 plot(contador_muestras/10, 100, 'r');
 hold on;
 plot(contador_muestras/10, y(contador_muestras), 'X-b');
 drawnow
 contador_muestras=contador_muestras+1;
end
%Cierro la conexión con el puerto serial y elimino las variables
fclose(puerto_serial);
delete(puerto_serial);
clear all;
end
```

Función de MATLAB para la sintonía del PID de velocidad

```
function Matlab_Arduino_pos(numero_muestras)
% Matlab + Arduino Serial Port communication
close all;
clc;
y=zeros(1,10000); %Vector donde se guardarán los datos
%Inicializo el puerto serial que utilizaré
delete(instrfind({'Port'},{'COM10'}));
puerto_serial=serial('COM10');
puerto_serial.BaudRate=9600;
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
%Abro el puerto serial
fopen(puerto_serial);
%Declaro un contador del número de muestras ya tomadas
contador_muestras=1;
%Creo una ventana para la gráfica
figure('Name','Serial communication: Matlab + Arduino.')
title('Ensayo PI posicion motor CC');
xlabel('Tiempo(segundos)');
ylabel('Posicion (grados)');
grid on;
hold on;
%Bucle while para que tome y dibuje las muestras que queremos
while contador_muestras<=numero_muestras
 ylim([0 180]); %Rango de posiciones a visualizar
 xlim([0 numero_muestras/100]);
 y(contador_muestras)=fscanf(puerto_serial,'%f')';
 %plot(contador_muestras/100, 100, 'r');
 %hold on;
 plot(contador_muestras/100, y(contador_muestras), 'X-b');
 drawnow
 contador_muestras=contador_muestras+1;
end
%Cierro la conexión con el puerto serial y elimino las variables
fclose(puerto_serial);
delete(puerto_serial);
clear all;
end
```

Programa de configuración del módulo bluetooth HC05

```
#include <SoftwareSerial.h>   // Incluimos la
librería  SoftwareSerial
SoftwareSerial BT(10,11);    // Definimos los pines RX y TX del
Arduino conectados al Bluetooth

void setup()
{
  BT.begin(38400);       // Inicializamos el puerto serie BT (Para
Modo AT 2)
  Serial.begin(38400);   // Inicializamos  el puerto serie
}

void loop()
{
  if(BT.available())     // Si llega un dato por el puerto BT se
envía al monitor serial
  {
    Serial.write(BT.read());
  }

  if(Serial.available())  // Si llega un dato por el monitor
serial se envía al puerto BT
  {
     BT.write(Serial.read());
  }
}
```

Programa principal

Fichero programa_tfg

```
/*****************************************************************
*****************************
Programa principal del TFG "Diseño de un sistema de almacenamiento
automático mediante Arduino"
  Autor: Alfredo Alonso Hermosilla
  Tutor: Eduardo Zalama Casanova  Departamento ISA
  Grado en Ingeniería Electrónica Industrial y Automática EII UVa
  Julio de 2017
*****************************************************************
*****************************/

/*INCLUSIÓN DE LIBRERÍAS NECESARIAS*/
#include <LiquidCrystal.h> //librería para control del LCD
#include <Servo.h> //librería para control del servomotor

/*DEFINICIÓN DE PINES UTILIZADOS*/

//Botones y Finales de carrera
#define boton_azul 2
#define boton_blanco 3
#define boton_rojo 4
#define fincarver 18 //final de carrera vertical
#define fincarhor 19 //final de carrera horizontal

//Servomotor
#define servo 5
Servo pinza;

//Pantalla LCD
LiquidCrystal lcd(11, 10, 9, 8, 7, 6);

//Drivers de motores paso a paso: pines de dirección (dir) y
pulsos (stp)
#define paphordir 12 //motor horizontal
#define paphorstp 13
#define papverdir A3 //motores verticales
#define papverstp A4

//Motor de corriente continua
#define encoderA 20
#define encoderB 21
#define enable 45 //pin para control de velocidad con PWM
#define IN1 47 //pines de dirección
#define IN2 49

//Sensores de ultrasonidos
#define echo_pinza 41
#define trig_pinza 39
#define echo_cinta 53
#define trig_cinta 51

//Zumbador
#define buzzer 43
```

```
//Joystick
#define joyver A0
#define joyhor A1
#define joysel A2

/*DECLARACIÓN DE VARIABLES GLOBALES DEL PROGRAMA*/

//Valores de pasos de los motores para los diferentes cajones y la
cinta
#define fila1 420
#define fila2 2660
#define fila3 5020
#define columna1 7350
#define columna2 5300
#define columna3 3200
#define cintah 80
#define cintav 370
#define margen 300 //pasos verticales para dejar o coger un palé

byte interfaz; //indica la interfaz de control del almacén
byte i = 0; //iterador para la sucesión de mensajes en la pantalla
del LCD
unsigned long t_cambio = millis(); //tiempo en que se produce un
cambio de mensaje en el LCD
int intervalo = 0; //tiempo entre cambios (0 para que se visualice
mensaje al comienzo sin esperas)
bool fcv_pulsado, fch_pulsado; //variables de estado de los
finales de carrera
bool sel_modo; //variable para indicar el cambio de modo sin
cambiar de interfaz
int contador = 0; //contador de pulsos de los encoders
bool cajones[9]; //vector con los cajones ocupados y libres
int cajon; //numero de cajón actual
const int intervalo_servo = 20; //tiempo entre cambios de posición
del servo
bool comienzo = true; //primera iteración del loop
bool lleno, vacio; //almacén lleno o vacío
byte comando; //comando a recibir por teclado o app
int pulsosh = 0, pulsosv = 0;
//posición del transelevador en términos de pulsos de los motores
paso a paso desde el origen


/*****************************************************************
*
Función principal de configuración ejecutada una sola vez al
inicio
del programa. No tiene entradas ni salida.
*****************************************************************
/
void setup() {
  //Se establece el modo de cada pin que funciona como digital
  //Entradas
  pinMode(boton_azul, INPUT_PULLUP);
  pinMode(boton_blanco, INPUT_PULLUP);
  pinMode(boton_rojo, INPUT_PULLUP);
  pinMode(fincarver, INPUT_PULLUP);
  pinMode(fincarhor, INPUT_PULLUP);
```

```
  pinMode(encoderA, INPUT_PULLUP);
  pinMode(encoderB, INPUT_PULLUP);
  pinMode(joysel, INPUT_PULLUP);
  pinMode(echo_pinza, INPUT);
  pinMode(echo_cinta, INPUT);

  //Salidas
  pinMode(paphordir, OUTPUT);
  pinMode(paphorstp, OUTPUT);
  pinMode(papverdir, OUTPUT);
  pinMode(papverstp, OUTPUT);
  pinMode(enable, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(trig_pinza, OUTPUT);
  pinMode(trig_cinta, OUTPUT);
  pinMode(buzzer, OUTPUT);

  //Interrupciones de finales de carrera y encoders
  attachInterrupt(digitalPinToInterrupt(fincarver), stop_vertical,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(fincarhor),
stop_horizontal, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderA), contarA,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderB), contarB,
CHANGE);

  lcd.begin(16, 2); //se indica el número de filas y columnas del
display
  Serial.begin(115200); //comunicación serie al baudrate del
bootloader del Arduino Mega

  pinza.attach(servo); //se asocia el elemento de la clase Servo
al pin del servo
  pinza.write(180); //posición de pinza recogida al comenzar
}


/*********************************************************
Función principal del progama que se ejecuta cíclicamente de
forma indefinida. No tiene entradas ni salida.
*********************************************************/
void loop() {

  //Comprobación del estado de los finales de carrera
  //Se inicializa su variable correspondiente según estén pulsados
o no
  if (digitalRead(fincarver))
    fcv_pulsado = false;
  else
    fcv_pulsado = true;

  if (digitalRead(fincarhor))
    fch_pulsado = false;
  else
    fch_pulsado = true;

  menu_interfaz(); //llamada a función de selección de interfaz
```

```
  do
  {
    menu_modo(); //llamada a función de selección de modo
  } while (sel_modo); //para seleccionar otro modo sin volver a
elegir interfaz
}

/*****************************************************************
****
Función de selección de la interfaz de control a través de los
tres
botones del cuadro de mandos. No tiene entradas ni salida.
*****************************************************************
***/
void menu_interfaz()
{
  i = 0; //reseteo del iterador de los mensajes de LCD

  //mientras no se pulsa un botón se permanece en este bucle
  while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul))
  {
    if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
    {
      intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

      switch (i) //se visualiza en el LCD lo que toca
      {
        case 0:
          lcd.clear();
          lcd.print("Escoja interfaz"); //se muestra un mensaje en
la primera fila
          lcd.setCursor(0, 1); //se coloca el cursor en la segunda
fila
          lcd.print("de control:"); //se continúa el mensaje en la
segunda fila
          break;

        case 1:
          lcd.clear();
          lcd.print("Rojo:");
          lcd.setCursor(0, 1);
          lcd.print("Cuadro de mandos");
          break;

        case 2:
          lcd.clear();
          lcd.print("Blanco:");
          lcd.setCursor(0, 1);
          lcd.print("Monitor serie");
          break;

        case 3:
          lcd.clear();
          lcd.print("Azul:");
          lcd.setCursor(0, 1);
```

```
          lcd.print("App Movil");
          break;
      }

      i++; //se incrementa el contador para que cambie el mensaje
      if (i == 4) //si se llega a 4 se resetea el contador
        i = 0;
      t_cambio = millis(); //se actualiza la variable t_cambio con
el instante del cambio actual
    }
  }
  bip(1000, 200); //sonido para indicar elección de interfaz

  if (!digitalRead(boton_rojo)) //según el botón pulsado se
inicializa interfaz a un valor
  {
    interfaz = 1;
  }
  else if (!digitalRead(boton_blanco))
  {
    interfaz = 2;
  }
  else
    interfaz = 3;

  delay(200);
}

/*****************************************************************
Función de selección del modo de funcionamiento desde la interfaz
que haya sido seleccionada. Llama a la función correspondiente a
la interfaz y modo escogidos y cuando termina dicha función
(salida del modo) vuelve a ejecutarse a menos que se requiera
cambiar de interfaz. No tiene entradas ni salida.
*****************************************************************/
void menu_modo()
{
  sel_modo = true; //repetir la función si no se indica lo
contrario
  bool opcion_correcta; //opción válida de modo de
funcionamiento  introducida o no

  if (interfaz == 1) //Interfaz mediante el cuadro de mandos
  {
    while (!digitalRead(boton_rojo)); //Se espera a que se deje de
pulsar el botón

    intervalo = 0; //para que se muestre el primer mensaje nada
más entrar al bucle siguiente
    i = 0; //Para empezar en el primer mensaje

    //mientras no se pulsa un botón se permanece en este bucle
    while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul))
    {
      if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
      {
```

```
        intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

        switch (i) //se visualiza en el LCD lo que toca
        {
          case 0:
            lcd.clear();
            lcd.print("Escoja modo de"); //se muestra un mensaje
en la primera fila
            lcd.setCursor(0, 1); //se coloca el cursor en la
segunda fila
            lcd.print("funcionamiento:"); //se continúa el mensaje
en la segunda fila
            break;

          case 1:
            lcd.clear();
            lcd.print("Rojo:");
            lcd.setCursor(0, 1);
            lcd.print("Manual");
            break;

          case 2:
            lcd.clear();
            lcd.print("Blanco:");
            lcd.setCursor(0, 1);
            lcd.print("Automatico");
            break;

          case 3:
            lcd.clear();
            lcd.print("Azul: Reelegir");
            lcd.setCursor(0, 1);
            lcd.print("interfaz");
            break;
        }

        i++; //se incrementa el contador para que cambie el
mensaje
        if (i == 4) //si se llega a 4 se resetea el contador
          i = 0;
        t_cambio = millis(); //se actualiza la variable t_cambio
con el instante del cambio actual
      }
    }
    bip(1000, 200); //sonido para indicar elección de modo
    if (!digitalRead(boton_rojo)) //si se escoge el modo manual se
va a la función correspondiente
    {
      lcd.clear();
      lcd.print("Modo");
      lcd.setCursor(0, 1);
      lcd.print("Manual");
      delay(200);
      mando_manual();
    }
    else if (!digitalRead(boton_blanco))  //si se escoge el modo
automático se va a la función correspondiente
    {
```

```
      lcd.clear();
      lcd.print("Modo");
      lcd.setCursor(0, 1);
      lcd.print("Automatico");
      while (!digitalRead(boton_blanco));
      delay(200);
      mando_automatico();
    }
    else //si se pulsa el azul no se repite la elección de modo y
por tanto se vuelve a elegir interfaz
      sel_modo = false;
  }
  else if (interfaz == 2) //Interfaz mediante el monitor serie del
ordenador
  {
    lcd.clear();
    lcd.print("Interfaz");
    lcd.setCursor(0, 1);
    lcd.print("monitor serie");

    Serial.println("");
    Serial.println("Seleccione el modo de funcionamiento: ");
    Serial.println("0-Reelegir interfaz");
    Serial.println("1-Modo Manual");
    Serial.println("2-Modo Automatico");
    Serial.println("");

    do
    {
      while (!Serial.available()); //se espera a que se introduzca
un comando
      comando = Serial.parseInt(); //se lee el comando
      Serial.println(comando);
      bip(1000, 200); //sonido para indicar elección de modo

      switch (comando) //según el modo escogido se va a una
función u otra
      {
        case 0:
          Serial.println("Se ha salido al menu de interfaz");
          sel_modo = false; //para no repetir el menú de modos y
volver a elegir interfaz
          opcion_correcta = true;
          break;

        case 1:
          Serial.println("Se ha escogido modo manual");
          opcion_correcta = true;
          monitor_manual();
          break;

        case 2:
          Serial.println("Se ha escogido modo automatico");
          opcion_correcta = true;
          monitor_automatico();
          break;

        //si el comando introducido no es correcto se pide volver
a introducirlo volviendo a ejecutar el bucle
```

```
        default:
          Serial.println("Modo incorrecto, vuelva a intentarlo");
          opcion_correcta = false;
          break;
      }
    } while (!opcion_correcta);
  }
  else //Interfaz mediante una aplicación Android
  {
    lcd.clear();
    lcd.print("Interfaz");
    lcd.setCursor(0, 1);
    lcd.print("aplicacion movil");

    app(); //función para control con aplicación móvil
  }
  delay(200);
}
```

Archivo funciones básicas

```
/*************************************************************
Funciones específicas para el uso de los diferentes elementos
del almacén como son los sensores de ultrasonidos, el zumbador
y los tres tipos de motores.
*************************************************************/

/*************************************************************
Función de toma de medidas de los sensores de ultrasonidos.
Entradas: bool indicando si se quiere tomar medida de sensor
de la cinta o del de la transpaleta.
Salida: float con la distancia medida en centímetros.
*************************************************************/
float ultrasonidos(bool cinta)
{
  unsigned long tiempo;
  float distancia;

  if (cinta) //si el ultrasonidos es el de la cinta
  {
    digitalWrite(trig_cinta, LOW); //para generar un pulso limpio
se pone el trigger a LOW 5us
    delayMicroseconds(5);
    digitalWrite(trig_cinta, HIGH); //se genera un pulso de 10us
    delayMicroseconds(10);
    digitalWrite(trig_cinta, LOW);
    tiempo = pulseIn(echo_cinta, HIGH); //se mide el tiempo que
tarda en volver el pulso, en microsegundos
  }
  else //si es el sensor de la pinza
  {
    digitalWrite(trig_pinza, LOW);
    delayMicroseconds(5);
    digitalWrite(trig_pinza, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig_pinza, LOW);
    tiempo = pulseIn(echo_pinza, HIGH);
```

```
  }

  delay(2);
  distancia = 0.017 * tiempo; //se obtiene la distancia en cm
  return distancia;
}

/*************************************************************
Función de pitidos del zumbador. No tiene entradas ni salida.
*************************************************************/
void bip(int frec, unsigned long t)
{
  tone(buzzer, frec, t); //pin del zumbador, frecuencia y tiempo
del pitido
}


/*************************************************
Función de interrupción llamada al cambiar el estado
del final de carrera vertical
*************************************************/
void stop_vertical()
{
  if (digitalRead(fincarver)) //se indica si está pulsado
modificando la variable
    fcv_pulsado = false;
  else
  {
    fcv_pulsado = true;
    bip(500, 400); //si se pulsa se emite un pitido
  }
}

/*************************************************
Función de interrupción llamada al cambiar el estado
del final de carrera horizontal
*************************************************/
void stop_horizontal()
{
  if (digitalRead(fincarhor)) //se indica si está pulsado
modificando la variable
    fch_pulsado = false;
  else
  {
    fch_pulsado = true;
    bip(500, 400); //si se pulsa se emite un pitido
  }
}

/*************************************************************************
***************
Función de envío de pulsos a los motores paso a paso. Debe
llamarse recursivamente
dentro de un bucle para conseguir el movimiento de los motores.
Entradas:
      -bool indicando si es el movimiento horizontal o vertical
(true horizontal).
      -bool con el sentido de avance (true izquierda y abajo).
```

```
      -float con la velocidad lineal a la que se debe avanzar en
um/s
Sin salida.
**********************************************************************
***************/
void pulso_pap(bool horiz, bool sentido, float vel)
{
  static unsigned long t_ant_ver = millis(), t_ant_hor = millis();
//tiempo en que se envía un pulso a los pap
  static int fch_virtual, fcv_virtual; //límites de movimiento en
los extremos opuestos a los finales de carrera
  float t_pulso; //tiempo de pulso correspondiente a la velocidad
requerida

  t_pulso = vel / 1000; //porque map no admite decimales se pasa
en um/s y aqui se transforma en mm/s
  t_pulso = t_pulso / 0.01; //mm/s -> pasos/s
  t_pulso = 1 / t_pulso; //pasos/s -> s/paso
  t_pulso = t_pulso / 2; //s por semipulso
  t_pulso = t_pulso * 1000; //ms por semipulso

  if (horiz) //motor horizontal
  {
    //si no se pide ir a la derecha con el final de carrera
pulsado o a la izquierda con el final de carrera virtual pulsado
    if ((sentido && !fch_virtual) || (!sentido && !fch_pulsado))
    {
      if (millis() - t_ant_hor > t_pulso) //si ha pasado el tiempo
de pulso
      {
        digitalWrite(paphordir, sentido); //se pone la dirección
solicitada
        digitalWrite(paphorstp, !digitalRead(paphorstp));  //se
cambia el estado del pin de pulsos
        t_ant_hor = millis(); //se actualiza el tiempo de pulso

        if (digitalRead(paphorstp)) //si se ha pasado a positivo
(paso del motor) se actualizan los pasos dados desde el origen
        {
          if (sentido)
            pulsosh++;
          else
            pulsosh--;
        }

        if (pulsosh == 7680) //si se llega a ese valor se pulsa el
final de carrera virtual para que no pueda avanzar más
        {
          fch_virtual = true;
          bip(500, 400);
        }
        else
          fch_virtual = false;
      }
    }
  }
  else //mismo procedimiento para los motores verticales
  {
```

```
    //si no se pide ir hacia abajo con el final de carrera pulsado
o hacia arriba con el final de carrera virtual pulsado
    if ((!sentido && !fcv_virtual) || (sentido && !fcv_pulsado))
    {
      if (millis() - t_ant_ver > t_pulso)
      {
        digitalWrite(papverdir, sentido);
        digitalWrite(papverstp, !digitalRead(papverstp));
        t_ant_ver = millis();

        if (digitalRead(papverstp))
        {
          if (sentido)
            pulsosv--;
          else
            pulsosv++;
        }

        if (pulsosv == 5400) //si se llega a ese valor se pulsa el
final de carrera virtual para que no pueda avanzar más
        {
          fcv_virtual = true;
          bip(500, 400);
        }
        else
          fcv_virtual = false;
      }
    }
  }
}

/*****************************************************************
*********
Función para el movimiento progresivo del servo entre dos
posiciones.
Debe ser llamada de forma recursiva hasta llegar a la posición
solicitada.
Entradas:
        -int con el intervalo de tiempo entre movimientos entre
posiciones
        contiguas en milisegundos (determina la velocidad).
        -int con la posición final del servo deseada.
Salida: bool indicando si se ha llegado a la posición final o no
*****************************************************************
********/
bool mover_servo(int intervalo_servo, int objetivo)
{
  static unsigned long t_mov = millis(); //tiempo en que se mueve
el servo
  static int pos = 180; //posición del servo

  //si ha pasado el tiempo necesario se ejecuta
  if (millis() - t_mov > intervalo_servo)
  {
    if (objetivo > pos) //si la posición deseada es mayor que la
actual
    {
      pos++; //se incrementa un grado y se ordena desplazarse a
esa posición
```

```
      pinza.write(pos);
    }
    else if (objetivo < pos) //si la posición deseada es menor que
la actual
    {
      pos--; //se decrementa un grado y se ordena desplazarse a
esa posición
      pinza.write(pos);
    }
    else //si la posición ya se ha alcanzado se devuelve true
indicando que el movimiento ha finalizado
      return true;

    t_mov = millis(); //se guarda el tiempo del cambio
  }
  return false; //se devuelve false para indicar que no se ha
llegado todavía al objetivo
}

/*****************************************************************
Función de interrupción del encoder del canal A que incrementa
o decrementa el contador de pulsos según el estado del canal B.
*****************************************************************/
void contarA()
{
  if (digitalRead(encoderA) == HIGH) //se mira flanco de subida en
canal A
  {
    if (digitalRead(encoderB) == LOW) //se mira canal B para ver
el sentido de giro
      contador += 1; //si el canal B está en estado contrario se
incrementa el contador
    else
      contador -= 1; //si está en el mismo se decrementa
  }
  else //debe de haber flanco de bajada en canal A
  {
    if (digitalRead(encoderB) == HIGH) //se mira canal B para ver
sentido de giro
      contador += 1;
    else
      contador -= 1;
  }
}

/*****************************************************************
Función de interrupción del encoder del canal B que incrementa
o decrementa el contador de pulsos según el estado del canal A.
*****************************************************************/
void contarB()
{
  if (digitalRead(encoderB) == HIGH) //se mira flanco de subida en
canal B
  {
    if (digitalRead(encoderA) == LOW) //se mira canal A para ver
el sentido de giro
      contador -= 1; //si el canal A está en estado contrario se
decrementa el contador
    else
```

```
      contador += 1; //si está en el mismo se incrementa
  }
  else //debe de haber flanco de bajada en canal B
  {
    if (digitalRead(encoderA) == HIGH) //se mira canal A para ver
el sentido de giro
      contador -= 1;
    else
      contador += 1;
  }
}

/*****************************************************************
**************
Función que controla la velocidad del motor de corriente continua
haciendo uso
de la función compute que implementa un PID. Debe llamarse
recursivamente para
que el PID siga realizando cálculos y se llegue y mantenga en la
referencia.
Entradas: int con la referencia de velocidad en rpm con signo para
indicar el
          sentido.
Sin salida.
*****************************************************************
**************/
void motorcc(int referencia)
{
  static unsigned long tiempo = 0; //momento en que se realiza un
cálculo
  double rpm, vel; //valor de velocidad en rpms y en PWM calculado
por el PID
  const int pulsos_rev = 360; //pulsos de los encoders que se
producen en una vuelta del motor
  const int dt = 100; //intervalo de tiempo entre mediciones

  if (millis() - tiempo > dt) //si ha pasado el tiempo entre
mediciones
  {
    if (contador != 1) //porque por error el encoder a veces saca
1
    {
      rpm = 60000 * contador; //cálculo de la velocidad a partir
de los pulsos de los encoders
      rpm = rpm / dt;
      rpm = rpm / pulsos_rev;
    }
    tiempo = millis(); //se almacena el tiempo actual
    contador = 0;  //se inicializan los pulsos.

    if (rpm < 200 && rpm > -200) //para evitar datos falsos
    {
      vel = compute(referencia, rpm, dt); //se llama a la función
del PID con la referencia y la velocidad
      girar(vel); //el PWM calculado se envía al motor
    }
  }
}
```

```
/*************************************************************
**************
Función de control de la posición del motor haciendo uno de la
función compute
que implementa un PID que calcula el PWM a introducir al motor
para llegar a la
posición deseada de giro.
Entradas: int con la referencia de posición deseada en grados.
Sin salida.
*************************************************************
*************/
void motorcc_pos(int referencia)
{
  double vel = 0; //PWM a introducir al enable del motor
  unsigned long tiempo = 0; //tiempo en que se produce un cálculo
  const int dt = 10; //intervalo entre cálculos

  if (millis() - tiempo > dt) //si ha pasado el intervalo entre
cálculos
  {
    //se ordena un nuevo cálculo pasando la referencia, la medida
de posición (pulsos) y el intervalo de tiempo
    vel = compute(referencia, contador, dt);
    girar(vel); //se pasa el valor calculado a la función que hace
moverse al motor
    tiempo = millis(); //se almacena el tiempo actual
  }
}

/*************************************************************
Función que envía el PWM solicitado al ENABLE del driver del
motor CC estableciendo el estado de IN1 e IN2 según el sentido.
Entradas: double con el PWM requerido con signo.
Sin salidas.
*************************************************************/
void girar(double pwm)
{
  if (pwm < 0) //si el valor de pwm es negativo -> una dirección
  {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
  }
  else //si es positivo la otra
  {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
  }
  pwm = abs(pwm); //valor absoluto para el pwm del motor
  analogWrite(enable, pwm); //envío del PWM al motor
}


/*************************************************************
**********************
Función que calcula mediante un PID el valor PWM que se debe
introducir al motor CC en
función de la medida y el set point de velocidad o de posición.
Entradas:
          -int con el set point deseado.
```

```
          -double de la medida de la posición o velocidad.
          -int del intervalo de tiempo entre mediciones.
Salida: double con el PWM calculado.
*****************************************************************
**********************/
double compute(int referencia, double medida, int dt)
{
  double salida; //valor PWM que calcula el PID
  static double last_error_p; //error proporcional de la iteración
anterior
  static double error_p, error_d, error_i; //errores proporcional,
derivativo e integral
  double Kp, Kd = 0, Ki; //parámetros del PID

  //El valor de dt es representativo de si se está controlando
posición o velocidad
  //En funcion de ello los parámetros del PID son diferentes
acorde a la mejor sintonía encontrada
  if (dt == 100)
  {
    Kp = 0.7;
    Ki = 0.004;
  }
  else
  {
    Kp = 10;
    Ki = 0;
  }

  error_p = referencia - medida; //cálculo del error proporcional
  error_d = (error_p - last_error_p) / dt; //cálculo del error
derivativo
  if (salida != 255 && salida != -255) //si no se satura el valor
del PWM se calcula el error integral (anti-windup)
    error_i = error_i + error_p * dt; //cálculo del error integral
  salida = Kp * error_p + Ki * error_i + Kd * error_d; //cálculo
de la salida
  last_error_p = error_p; //se guarda el error para la siguiente
iteración
  //saturación de máxima velocidad
  if (salida > 255)
    salida = 255;
  else if (salida < -255)
    salida = -255;
  return (salida); //se devuelve el valor calculado
}
```

Archivo cuadro de mandos

```
/*************************************************************
Funciones para el control manual y automático del almacén desde
el cuadro de mandos formado por un joystick y tres botones.
*************************************************************/

/*************************************************************
Función para control del almacén en modo manual desde el cuadro
de mandos. Se llama al escoger dicha interfaz y modo dentro del
menú de modos. No tiene entradas ni salida.
*************************************************************/
void mando_manual()
{
  bool joystick_pap = true; //determina si el joystick mueve los
motores paso a paso o el motor de continua
  bool lectura_rojo_ant = LOW, lectura_blanco_ant = LOW; //lectura
del botón rojo
  int lectura; //valor joystick
  float vel; //velocidad de los motores pap
  int objetivo = 180; //posición a la que enviar el servo
  bool fin_mov_servo = true; //indica si el servo ha terminado su
movimiento solicitado
  bool sentido; //sentido de los motores pap
  intervalo = 0; //para que se muestre el primer mensaje nada más
entrar al bucle siguiente
  i = 0; //Para empezar en el primer mensaje

  //mientras no se pulsa el botón azul se permanece en este bucle
  while (digitalRead(boton_azul))
  {
    if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
    {
      intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

      switch (i) //se visualiza en el LCD lo que toca
      {
        case 0:
          lcd.clear();
          lcd.print("Modo");
          lcd.setCursor(0, 1);
          lcd.print("Manual");
          break;

        case 1:
          lcd.clear();
          lcd.print("Joystick:");
          lcd.setCursor(0, 1);
          lcd.print("Movim PAP/CC");
          break;

        case 2:
          lcd.clear();
          lcd.print("Rojo:");
          lcd.setCursor(0, 1);
          lcd.print("Cambio PAP/CC");
```

```
          break;

        case 3:
          lcd.clear();
          lcd.print("Blanco:");
          lcd.setCursor(0, 1);
          lcd.print("Transpaleta");
          break;

        case 4:
          lcd.clear();
          lcd.print("Azul:");
          lcd.setCursor(0, 1);
          lcd.print("Cambio modo");
          break;
      }

      i++; //se incrementa el contador para que cambie el mensaje
      if (i == 5) //si se llega a 4 se resetea el contador
        i = 0;
      t_cambio = millis(); //se actualiza la variable t_cambio con
el instante del cambio actual
    }

    if (!digitalRead(boton_blanco) && lectura_blanco_ant) //si se
pulsa el botón del joystick
    {
      if (objetivo == 180) //cambia la posición en la que tiene
que estar el servo
        objetivo = 30;
      else
        objetivo = 180;

      fin_mov_servo = false; //para que se empiece a mover
    }
    lectura_blanco_ant = digitalRead(boton_blanco);

    if (!fin_mov_servo) //si no ha acabado el movimiento del servo
      fin_mov_servo = mover_servo(intervalo_servo, objetivo); //se
llama a la función de movimiento del servo

    if (!digitalRead(boton_rojo) && lectura_rojo_ant) //si se
pulsa el botón rojo
    {
      joystick_pap = !joystick_pap;
      bip(1000, 200);
      delay(200);
    }

    //para que no se cambie entre pap y cc en cada iteración
mientras está pulsado el botón
    lectura_rojo_ant = digitalRead(boton_rojo);

    if (joystick_pap) //movimiento de los motores paso a paso
    {
      //lectura de la posición del joystick con ajuste del cero al
centro
      lectura = analogRead(joyhor) - 524;
```

```
      if (lectura > 0) //según el signo de la lectura el motor
girará en un sentido u otro
        sentido = false;
      else
        sentido = true;

      lectura = abs(lectura);

      //dejando un margen alrededor del centro del joystick para
que los motores no se muevan sin tocarlo (50)
      if (lectura > 50)
      {
        //se mapea la lectura para convertirla en una velocidad
        //(um/s porque map solo funciona con enteros y en mm/s
habría pocos valores)
        vel = map(lectura, 50, 524, 1000, 10000); //ajustar
        pulso_pap(true, sentido, vel); //se llama a la función de
envío de pulsos
      }

      //de igual forma para el movimiento vertical
      lectura = analogRead(joyver) - 516;
      if (lectura > 0)
        sentido = true;
      else
        sentido = false;

      lectura = abs(lectura);

      if (lectura > 50)
      {
        vel = map(lectura, 50, 516, 1000, 10000); //ajustar
        pulso_pap(false, sentido, vel);
      }
    }
    else //movimiento del motor cc
    {
      lectura = analogRead(joyhor) - 524; //se lee el valor del
joystick

      lectura = map(lectura, -524, 524, -190, 190); //conversión a
rpm

      if (lectura < -40 || lectura > 40) //se descartan
velocidades pequeñas con las que el motor no se mueve y pita
        motorcc(lectura); //se llama a la función de movimiento
del motor con el valor deseado de velocidad
      else //si la velocidad no es suficiente no se activa el
motor
      {
        digitalWrite(IN1, LOW);
        digitalWrite(IN2, LOW);
      }
    }
  }
  bip(500, 200); //sonido para indicar salida del modo
}

/*****************************************************************
```

```
Función para control del almacén en modo automático desde el
cuadro
de mandos. Se llama al escoger dicha interfaz y modo dentro del
menú de modos. No tiene entradas ni salida.
**********************************************************************
/
void mando_automatico()
{
  bool salir = false; //salir del modo
  float distancia; //distancia que lee el ultrasonidos de la cinta
  intervalo = 0; //para que se muestre el primer mensaje nada más
entrar al bucle siguiente
  i = 0; //Para empezar en el primer mensaje
  bool obviar = false; //indica si hay un objeto en la cinta que
no se ha querido meter en el almacén de momento

  //mientras no se pulsa un botón se permanece en este bucle
  while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul))
  {
    if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
    {
      intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

      switch (i) //se visualiza en el LCD lo que toca
      {
        case 0:
          lcd.clear();
          lcd.print("Modo");
          lcd.setCursor(0, 1);
          lcd.print("Automatico");
          break;

        case 1:
          lcd.clear();
          lcd.print("Rojo: Registro");
          lcd.setCursor(0, 1);
          lcd.print("Manual");
          break;

        case 2:
          lcd.clear();
          lcd.print("Blanco: Registro");
          lcd.setCursor(0, 1);
          lcd.print("Automatico");
          break;

        case 3:
          lcd.clear();
          lcd.print("Azul: Reelegir");
          lcd.setCursor(0, 1);
          lcd.print("modo");
          break;
      }

      i++; //se incrementa el contador para que cambie el mensaje
      if (i == 4) //si se llega a 4 se resetea el contador
```

```
        i = 0;
      t_cambio = millis(); //se actualiza la variable t_cambio con
el instante del cambio actual
    }
  }

  //si se pulsa el rojo se llama a la función de reconocimiento
manual
  if (!digitalRead(boton_rojo))
  {
    bip(1000, 200);
    delay(200);
    if (!reconocimiento(true)) //si devuelve falso es que se ha
abortado y hay que salir de nuevo al menú
      salir = true;
  }
  else if (!digitalRead(boton_blanco)) //si se pulsa blanco se
llama a la función de reconocimiento automático
  {
    bip(1000, 200);
    delay(200);
    if (!reconocimiento(false)) //si devuelve falso es que se ha
abortado y hay que salir de nuevo al menú
      salir = true;
  }
  else //si se pulsa azul se sale de nuevo al menú
  {
    bip(500, 200);
    delay(200);
    salir = true;
  }

  while (!salir) //mientras que no se haya solicitado salir al
menú se permanece en este bucle
  {
    intervalo = 0; //para que se muestre el primer mensaje nada
más entrar al bucle siguiente
    i = 0; //Para empezar en el primer mensaje

    //mientras no se pulsa un botón se permanece en este bucle
    while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul) && digitalRead(joysel))
    {
      if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
      {
        intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

        switch (i) //se visualiza en el LCD lo que toca
        {
          case 0:
            lcd.clear();
            lcd.print("Modo");
            lcd.setCursor(0, 1);
            lcd.print("Automatico");
            break;

          case 1:
```

```
            lcd.clear();
            lcd.print("Rojo:");
            lcd.setCursor(0, 1);
            lcd.print("Sacar objeto");
            break;

          case 2:
            lcd.clear();
            lcd.print("Blanco: ");
            lcd.setCursor(0, 1);
            lcd.print("Mover objeto");
            break;

          case 3:
            lcd.clear();
            lcd.print("Azul: Reelegir");
            lcd.setCursor(0, 1);
            lcd.print("modo");
            break;

          case 4:
            lcd.clear();
            lcd.print("Boton joystick");
            lcd.setCursor(0, 1);
            lcd.print("Meter objeto");
            break;
        }

        i++; //se incrementa el contador para que cambie el
mensaje
        if (obviar) //si hay un objeto pendiente de introducir se
muestra el último mensaje
        {
          if (i == 5) //si se llega a 5 se resetea el contador
            i = 0;
        }
        else if (i == 4) //si se llega a 4 se resetea el contador
          i = 0;

        t_cambio = millis(); //se actualiza la variable t_cambio
con el instante del cambio actual

        distancia = ultrasonidos(true); //se toma medida del
ultrasonidos de la cinta
        if (distancia > 20 && obviar) //si había un objeto a la
espera y ya no lo hay (con doble comprobación)
        {
          distancia = ultrasonidos(true);
          if (distancia > 20)
          {
            bip(300, 700);
            lcd.clear();
            lcd.print("El objeto ha");
            lcd.setCursor(0, 1);
            lcd.print("desaparecido");
            obviar = false; //se indica que ya no hay objeto en
espera
            delay(2000);
          }
```

```
        }
        else if (distancia < 20 && !obviar) //si no había objeto y
ahora se detecta uno (con doble comprobación)
        {
          distancia = ultrasonidos(true); //se toma otra medida
del ultrasonidos de la cinta para evitar falsas medidas
          if (distancia < 20)
          {
            if (!lleno) //si el almacén no está lleno
            {
              bip(1300, 700);
              obviar = meter_en_almacen(); //se llama a la función
de meter objeto en el almacén
              //esta función devolverá true si se decide no meter
el objeto y
              //false si se introduce el objeto de forma que su
valor se asigna a obviar
            }
            else //si el almacén está lleno se indica y se pide
retirar el objeto de la cinta
            {
              bip(300, 700);
              lcd.clear();
              lcd.print("Retire objeto");
              lcd.setCursor(0, 1);
              lcd.print("Almacen lleno");

              while (true) //se espera a que se retira el objeto
de la cinta comprobando con el ultrasonidos
              {
                if (ultrasonidos(true) > 20)
                  if (ultrasonidos(true) > 20)
                    break;
              }
            }
          }
        }
      }
    }

    if (!digitalRead(boton_rojo)) //si se escoge la opción de
sacar un objeto
    {
      if (obviar) //si hay un objeto en la cinta se deniega la
acción puesto que podrían chocar
      {
        bip(300, 700);
        lcd.clear();
        lcd.print("Objeto en la");
        lcd.setCursor(0, 1);
        lcd.print("cinta");
        delay(2000);
      }
      else //si no hay un objeto en la cinta
      {
        if (!vacio) //si el almacén no está vacío se llama a la
función que saca objetos del almacén
        {
          bip(1000, 200);
```

```
          delay(200);
          sacar_del_almacen();
        }
        else //si está vacío se indica y se deniega la acción
        {
          lcd.clear();
          lcd.print("Almacen");
          lcd.setCursor(0, 1);
          lcd.print("vacio");
          delay(2000);
        }
      }
    }
    else if (!digitalRead(boton_blanco)) //si se escoge mover
objeto entre cajones se llama a la función correspondiente
    {
      bip(1000, 200);
      delay(200);
      mover_en_almacen();
    }
    else if (!digitalRead(boton_azul)) //si se escoge salir al
menú de selección de modos
    {
      bip(500, 200);
      delay(200);
      salir = true; //para que se salga del bucle de elegir acción
y se vuelva al menú de modos
    }
    else if (obviar) //si se pulsa el joystick y hay un objeto en
espera
    {
      bip(1000, 200);
      delay(200);
      obviar = meter_en_almacen(); //se vuelve a llamar a la
función de meter objeto en el almacén
    }
  }
}

/*****************************************************************
*
Función que devuelve un número de cajón escogido por el usuario
mediante el cuadro de mandos.
Entradas: bool indicando si se va a extraer o a coger del cajón a
          seleccionar para mostrar un mensaje por pantalla
adecuado.
Salida: int con el número de cajón seleccionado.
*****************************************************************
*/
int escoger_cajon(bool extraer)
{
  int fila, columna; //fila y columna del objeto a sacar
  int cajon; //número de cajón;
  bool correcto; //indica si se ha escogido bien o se desea
repetir la elección

  do {
    intervalo = 0; //para que se muestre el primer mensaje nada
más entrar al bucle siguiente
```

```
    i = 0; //Para empezar en el primer mensaje
    //mientras no se pulsa un botón se permanece en este bucle
    while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul))
    {
      if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
      {
        intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

        switch (i) //se visualiza en el LCD lo que toca
        {
          case 0:
            lcd.clear();
            lcd.print("Seleccione cajon");
            lcd.setCursor(0, 1);
            if (extraer)
              lcd.print("del que extraer");
            else
              lcd.print("en el que meter");
            break;

          case 1:
            lcd.clear();
            lcd.print("Escoja");
            lcd.setCursor(0, 1);
            lcd.print("fila");
            break;

          case 2:
            lcd.clear();
            lcd.print("Rojo:");
            lcd.setCursor(0, 1);
            lcd.print("Fila inferior");
            break;

          case 3:
            lcd.clear();
            lcd.print("Blanco:");
            lcd.setCursor(0, 1);
            lcd.print("Fila central");
            break;

          case 4:
            lcd.clear();
            lcd.print("Azul:");
            lcd.setCursor(0, 1);
            lcd.print("Fila superior");
            break;
        }

        i++; //se incrementa el contador para que cambie el
mensaje
        if (i == 5) //si se llega a 4 se resetea el contador
          i = 0;
        t_cambio = millis(); //se actualiza la variable t_cambio
con el instante del cambio actual
      }
```

```
    }

    bip(1000, 200);
    if (!digitalRead(boton_rojo)) //según el botón pulsado se
elige una fila
      fila = 1;
    else if (!digitalRead(boton_blanco))
      fila = 2;
    else
      fila = 3;
    delay(200);

    intervalo = 0; //para que se muestre el primer mensaje nada
más entrar al bucle siguiente
    i = 0; //Para empezar en el primer mensaje
    //mientras no se pulsa un botón se permanece en este bucle
    while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul))
    {
      if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
      {
        intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

        switch (i) //se visualiza en el LCD lo que toca
        {
          case 0:
            lcd.clear();
            lcd.print("Escoja");
            lcd.setCursor(0, 1);
            lcd.print("columna");
            break;

          case 1:
            lcd.clear();
            lcd.print("Rojo: Columna");
            lcd.setCursor(0, 1);
            lcd.print("izquierda");
            break;

          case 2:
            lcd.clear();
            lcd.print("Blanco:");
            lcd.setCursor(0, 1);
            lcd.print("Columna central");
            break;

          case 3:
            lcd.clear();
            lcd.print("Azul:");
            lcd.setCursor(0, 1);
            lcd.print("Columna derecha");
            break;
        }

        i++; //se incrementa el contador para que cambie el
mensaje
        if (i == 4) //si se llega a 4 se resetea el contador
```

```
          i = 0;
        t_cambio = millis(); //se actualiza la variable t_cambio
con el instante del cambio actual
      }
    }

    bip(1000, 200);
    if (!digitalRead(boton_rojo)) //según el botón pulsado se
elige una columna
      columna = 1;
    else if (!digitalRead(boton_blanco))
      columna = 2;
    else
      columna = 3;
    delay(200);

    //según la combinación de fila y columna se determina el
número de cajón escogido
    if (fila == 1)
    {
      if (columna == 1)
        cajon = 1;
      else if (columna == 2)
        cajon = 2;
      else
        cajon = 3;
    }
    else if (fila == 2)
    {
      if (columna == 1)
        cajon = 4;
      else if (columna == 2)
        cajon = 5;
      else
        cajon = 6;
    }
    else
    {
      if (columna == 1)
        cajon = 7;
      else if (columna == 2)
        cajon = 8;
      else
        cajon = 9;
    }

    //se pide confirmar si el cajón es correcto
    lcd.clear();
    lcd.print("Cajon ");
    lcd.print(cajon);
    lcd.print(" OK?");
    lcd.setCursor(0, 1);
    lcd.print("RojoSi BlancoNo");

    while (digitalRead(boton_rojo) && digitalRead(boton_blanco));

    bip(1000, 200);
    if (!digitalRead(boton_rojo)) //si se pulsa rojo se sale de la
función dando el cajón como correcto
```

```
      correcto = true;
    else //si se pulsa blanco se manda repetir el proceso de
elección ya que se ha producido un fallo
      correcto = false;
    delay(200);

  } while (!correcto);

  return cajon; //la función devuelve el cajón seleccionado
}
```

Archivo monitor serie

```
/*************************************************************
Funciones para el control del almacén desde el monitor serie
del IDE de Arduino teniendo el microcontrolador conectado
por cable al ordenador.
*************************************************************/

/*****************************************************************
Función para control del almacén en modo manual desde el monitor
serie. Se llama al escoger dicha interfaz y modo dentro del
menú de modos. No tiene entradas ni salida.
*****************************************************************/
void monitor_manual()
{
  bool salir = false; //indica si se debe salir de la función y
volver al menú
  int objetivo = 180; //posición a la que enviar el servo
  float vel; //velocidad de avance de los tornillos
  bool sentido; //sentido de movimiento de los motores

  while (!salir) //mientras no se pida  salir
  {
    //se muestra el menú de los posibles movimientos
    Serial.println("");
    Serial.println("Seleccione la accion a realizar:");
    Serial.println("1-Movimiento vertical");
    Serial.println("2-Movimiento horizontal");
    Serial.println("3-Mover cinta");
    Serial.println("4-Meter/sacar transpaleta");
    Serial.println("Otro-Salir al menu anterior (en cualquier
momento)");
    Serial.println("");

    while (!Serial.available()); //se espera a que se introduzca
un comando
    comando = Serial.parseInt(); //se lee el comando
    Serial.println(comando);
    bip(1000, 200); //sonido para indicar elección de modo

    switch (comando) //según el modo escogido se va a una función
u otra
    {
      case 1: //movimiento vertical
        Serial.println("Movimiento vertical");
```

```
        Serial.println("Escoja velocidad: (1-10 mm/s) Positivo:
Arriba  Negativo: Abajo");
        Serial.println("Fuera de rango: salir al menu");
        Serial.println("");

        while (!Serial.available()); //se espera a que se elija la
velocidad
        bip(1000, 200);
        vel = Serial.parseFloat(); //se lee el valor introducido
        Serial.println(vel);

        if (vel >= 1 && vel <= 10) //si está dentro de los valores
posibles y es positivo el sentido será false (hacia arriba)
        {
          sentido = false;
        }
        else if (vel >= -10 && vel <= -1) //si es positivo el
sentido será true (hacia abajo)
        {
          sentido = true;
        }
        else //si el valor no es correcto se sale del switch sin
hacer ninguna acción y se vuelve a mostrar el menú
          break;

        vel = abs(vel); //se coge el valor absoluto una vez
determinado el sentido
        vel *= 1000; //para pasar a um/s

        Serial.println("En movimiento: Pulse una tecla para
parar"); //se mueve hasta que se introduce un comando
        while (!Serial.available())
          pulso_pap(false, sentido, vel);
        bip(1000, 200);
        comando = Serial.read(); //al recibir un comando se para
de enviar pulsos
        break;

      case 2: //movimiento horizontal (análogo al vertical)
        Serial.println("Movimiento horizontal");
        Serial.println("Escoja velocidad: (1-10 mm/s) Positivo:
Derecha  Negativo: Izquierda");
        Serial.println("Fuera de rango: salir al menu");
        Serial.println("");

        while (!Serial.available());
        bip(1000, 200);
        vel = Serial.parseFloat();
        Serial.println(vel);

        if (vel >= 1 && vel <= 10)
        {
          sentido = false;
        }
        else if (vel >= -10 && vel <= -1)
        {
          sentido = true;
        }
        else
```

```
          break;

        vel = abs(vel);
        vel *= 1000;
        Serial.println("En movimiento: Pulse una tecla para
parar");
        while (!Serial.available())
          pulso_pap(true, sentido, vel);
        bip(1000, 200);
        comando = Serial.read();
        break;

      case 3: //movimiento de la cinta
        Serial.println("Movimiento cinta");
        Serial.println("Escoja velocidad: (40-190rpm) Positivo:
Derecha  Negativo: Izquierda");
        Serial.println("Fuera de rango: salir al menu");
        Serial.println("");


        while (!Serial.available()); //se espera a que se
introduzca velocidad
        bip(1000, 200);
        vel = Serial.parseInt(); //se lee la velocidad
        Serial.println(vel);

        //si está dentro del rango se llama a la función de
movimiento del motor cc recursivamente hasta que se envía un
comando
        if ((vel >= -190 && vel <= -40) || (vel >= 40 && vel <=
190))
        {
          Serial.println("En movimiento: Pulse una tecla para
parar");
          while (!Serial.available())
            motorcc(vel);
          bip(1000, 200);
          comando = Serial.read(); //cuando llega un comnado se
deja de llamar a la función del motor
        }
        else
          break;

        digitalWrite(IN1, LOW); //se para el motor
        digitalWrite(IN2, LOW);

        break;

      case 4: //movimiento de la transpaleta
        Serial.println("Moviendo transpaleta");

        if (objetivo == 180) //si estaba retraída se cambia el
objetivo para que se estire y viceversa
          objetivo = 30;
        else
          objetivo = 180;
```

```
        while (!mover_servo(intervalo_servo, objetivo)); //se
llama a la función de movimiento del servo hasta que acabe el
movimiento

        break;

      default: //si el comando no es uno de los anteriores se sale
al menú de modos
        Serial.println("Salida a menu");
        salir = true;
        break;
    }
  }
}

/*****************************************************************
Función para control del almacén en modo automático desde el
monitor serie. Se llama al escoger dicha interfaz y modo dentro
del
menú de modos. No tiene entradas ni salida.
*****************************************************************
/
void monitor_automatico()
{
  bool salir = false; //para indicar si se debe salir del modo o
no
  float distancia; //distancia que lee el ultrasonidos de la cinta
  bool obviar = false; //indica si hay un objeto en la cinta que
no se ha querido meter en el almacén de momento
  bool repetir_menu; //indica si se debe volver a mostrar el menú
de acciones o no
  comando = 0; //reseteo de la variable con la que se leen
comandos

  //se muestra un menú para los tipos de registros
  Serial.println("");
  Serial.println("Modo automatico");
  Serial.println("1-Reconocimiento manual");
  Serial.println("2-Reconocimiento automatico");
  Serial.println("Otro-Reelegir modo");
  Serial.println("");
  while (!Serial.available()); //se espera a que se elija el tipo
de reconocimiento
  bip(1000, 200);
  comando = Serial.parseInt(); //se lee y transforma el comando
  Serial.println(comando);

  if (comando == 1) //si se elige reconocimiento manual se llama a
la función de reconocimiento indicando el modo manual
  {
    if (!reconocimiento(true)) //si devuelve falso es que se ha
abortado y hay que salir de nuevo al menú
      salir = true;
  }
  else if (comando == 2) //si se elige reconocimiento automático
se llama a la función de reconocimiento indicando el modo
automático
  {
```

```
    if (!reconocimiento(false)) //si devuelve falso es que se ha
abortado y hay que salir de nuevo al menú
      salir = true;
  }
  else //si el comando es otro se sale al menú de modos
    salir = true;

  while (!salir) //mientras que no se pida salir
  {
    repetir_menu = true; //al volver a ejecutar el bucle tras
ejecutar una acción se debe volver a mostrar el menú
    while (!Serial.available()) //mientras no se introduzca un
comando se espera tomando medidas del ultrasonidos de la cinta
    {
      if (repetir_menu) //se muestra el menú de acciones si así se
indica
      {
        Serial.println("");
        Serial.println("Seleccione una accion");
        Serial.println("1-Sacar objeto");
        Serial.println("2-Mover objeto");
        Serial.println("3-Meter objeto (si hay objeto en
espera)");
        Serial.println("Otro-Reelegir modo");
        Serial.println("");
        repetir_menu = false; //para no repetir el menú en cada
iteración mientras no se introduzca un comando
      }

      distancia = ultrasonidos(true); //se toma medida del
ultrasonidos de la cinta
      if (distancia > 20 && obviar) //si había un objeto a la
espera y ya no lo hay (con doble comprobación)
      {
        if (ultrasonidos(true) > 20)
        {
          bip(300, 700);
          Serial.println("El objeto ha desaparecido");
          obviar = false; //se indica que ya no hay objeto en
espera
        }
      }
      else if (distancia < 20 && !obviar) //si no había objeto y
ahora se detecta uno (con doble comprobación)
      {
        distancia = ultrasonidos(true);
        if (distancia < 20)
        {
          if (!lleno) //si el almacén no está lleno
          {
            bip(1300, 700);
            obviar = meter_en_almacen(); //se llama a la función
de meter objeto en el almacén
            //esta función devolverá true si se decide no meter el
objeto y false si se introduce de forma que su valor se asigna a
obviar
            repetir_menu = true; //tras ello se debe volver a
mostrar el menú de acciones
          }
```

```
          else //si el almacén está lleno se indica y se pide
retirar el objeto de la cinta
          {
            bip(300, 700);
            Serial.println("Retire objeto, el almacen esta
lleno");

            while (true) //se espera a que se retira el objeto de
la cinta comprobando con el ultrasonidos
            {
              if (ultrasonidos(true) > 20)
                if (ultrasonidos(true) > 20)
                  break;
            }
            Serial.println("Gracias");
          }
        }
      }
    }

    bip(1000, 200);
    comando = Serial.parseInt(); //se lee la opción escogida
    Serial.println(comando);

    if (comando == 1) //si se escoge la opción de sacar un objeto
    {
      if (obviar) //si hay un objeto en la cinta se deniega la
acción puesto que podrían chocar
        Serial.println("Objeto en la cinta impide sacar del
almacen");
      else //si no hay un objeto en la cinta
      {
        if (vacio) //si está vacío se indica y se deniega la
acción
          Serial.println("Almacen vacio");
        else //si el almacén no está vacío se llama a la función
que saca objetos del almacén
          sacar_del_almacen();
      }
    }
    else if (comando == 2) //si se escoge mover objeto entre
cajones se llama a la función correspondiente
      mover_en_almacen();
    else if (comando == 3)  //si se pulsa el joystick
    {
      if (obviar) //si hay un objeto en espera
        obviar = meter_en_almacen();  //se vuelve a llamar a la
función de meter objeto en el almacén
      else
        Serial.println("No hay nada que meter");
    }
    else //si se escoge salir al menú de selección de modos se
pone salir a true y no se repite el bucle
      salir = true;
  }
}
```

Archivo app móvil

```
/*********************************************************
Función para el control manual desde aplicación Android.
Llamada desde el menú de modos habiendo elegido la interfaz
de la aplicación. Sin entradas ni salida.
*********************************************************/
void app()
{
  bool sentidov, sentidoh, sentidocc = false; //sentido de
movimiento de los motores
  int velocidadh, velocidadv, velocidadcc = 0; //velocidad de los
motores
  bool moverh = false, moverv = false, encendercc = false;
//activación de los motores
  comando = 17; //valor inicial de la variable que guarda los
comandos que llegan por bluetooth (todos los motores parados)
  int objetivo = 180; //posición a la que llevar al servo
  bool fin_mov_servo = true; //indica si el servo ha terminado su
movimiento

  do //bucle en el que se permanece mientras no llegue el comando
de salir del control por app Android (0)
  {
    if (Serial.available()) //si llega un comando se lee (al
enviarse bytes con read se obtiene el valor correcto sin necesidad
de parseInt)
    {
      comando = Serial.read();

      switch (comando) //según el comando se realiza una acción
      {
        case 0: //el 0 indica salida de la interfaz
          sel_modo = false; //se indica salida del menú de modos
para volver a elegir interfaz
          encendercc = false; //se apaga el motor cc si no lo está
          bip(1000, 200);
          break;

        case 1: //movimiento de la transpaleta
          if (objetivo == 180) //cambia la posición en la que
tiene que estar el servo
            objetivo = 30;
          else
            objetivo = 180;

          fin_mov_servo = false; //para que se empiece a mover
          bip(1000, 200);
          break;

        case 2: //encendido del motor cc
          encendercc = !encendercc;
          bip(1000, 200);
          break;

        case 3: //cambio de sentido del motor cc
          sentidocc = !sentidocc;
```

```
          velocidadcc = -velocidadcc; //la velocidad cambia de
signo
          bip(1000, 200);
          break;

        //de 4 a 28 son posiciones del joystick para el manejo de
los motores paso a paso
        case 4: //caso de arriba e izquierda a alta velocidad
(ambas)
          moverh = true; //se debe mover el motor horizontal
          moverv = true; //se debe mover el motor vertical
          sentidoh = true; //sentido izquierda
          sentidov = false; //sentido arriba
          velocidadh = 8000; //velocidad en um/s para entrar a la
función de pulsos
          velocidadv = 8000; //velocidad en um/s para entrar a la
función de pulsos
          break;

        case 5:
          moverh = true;
          moverv = true;
          sentidoh = true;
          sentidov = false;
          velocidadh = 8000;
          velocidadv = 3000;
          break;

        case 6:
          moverh = true;
          moverv = false;
          sentidoh = true;
          velocidadh = 8000;
          break;

        case 7:
          moverh = true;
          moverv = true;
          sentidoh = true;
          sentidov = true;
          velocidadh = 8000;
          velocidadv = 3000;
          break;

        case 8:
          moverh = true;
          moverv = true;
          sentidoh = true;
          sentidov = true;
          velocidadh = 8000;
          velocidadv = 8000;
          break;

        case 9:
          moverh = true;
          moverv = true;
          sentidoh = true;
          sentidov = false;
          velocidadh = 3000;
```

```
    velocidadv = 8000;
    break;

  case 10:
    moverh = true;
    moverv = true;
    sentidoh = true;
    sentidov = false;
    velocidadh = 3000;
    velocidadv = 3000;
    break;

  case 11:
    moverh = true;
    moverv = false;
    sentidoh = true;
    velocidadh = 3000;
    break;

  case 12:
    moverh = true;
    moverv = true;
    sentidoh = true;
    sentidov = true;
    velocidadh = 3000;
    velocidadv = 3000;
    break;

  case 13:
    moverh = true;
    moverv = true;
    sentidoh = true;
    sentidov = true;
    velocidadh = 3000;
    velocidadv = 8000;
    break;

  case 14:
    moverh = false;
    moverv = true;
    sentidov = false;
    velocidadv = 8000;
    break;

  case 15:
    moverh = false;
    moverv = true;
    sentidov = false;
    velocidadv = 3000;
    break;

  case 16:
    moverh = false;
    moverv = false;
    break;

  case 17:
    moverh = false;
    moverv = true;
```

```
    sentidov = true;
    velocidadv = 3000;
    break;

  case 18:
    moverh = false;
    moverv = true;
    sentidov = true;
    velocidadv = 8000;
    break;

  case 19:
    moverh = true;
    moverv = true;
    sentidoh = false;
    sentidov = false;
    velocidadh = 3000;
    velocidadv = 8000;
    break;

  case 20:
    moverh = true;
    moverv = true;
    sentidoh = false;
    sentidov = false;
    velocidadh = 3000;
    velocidadv = 3000;
    break;

  case 21:
    moverh = true;
    moverv = false;
    sentidoh = false;
    velocidadh = 3000;
    break;

  case 22:
    moverh = true;
    moverv = true;
    sentidoh = false;
    sentidov = true;
    velocidadh = 3000;
    velocidadv = 3000;
    break;

  case 23:
    moverh = true;
    moverv = true;
    sentidoh = false;
    sentidov = true;
    velocidadh = 3000;
    velocidadv = 8000;
    break;

  case 24:
    moverh = true;
    moverv = true;
    sentidoh = false;
    sentidov = false;
```

```
          velocidadh = 8000;
          velocidadv = 8000;
          break;

        case 25:
          moverh = true;
          moverv = true;
          sentidoh = false;
          sentidov = false;
          velocidadh = 8000;
          velocidadv = 3000;
          break;

        case 26:
          moverh = true;
          moverv = false;
          sentidoh = false;
          velocidadh = 8000;
          break;

        case 27:
          moverh = true;
          moverv = true;
          sentidoh = false;
          sentidov = true;
          velocidadh = 8000;
          velocidadv = 3000;
          break;

        case 28:
          moverh = true;
          moverv = true;
          sentidoh = false;
          sentidov = true;
          velocidadh = 8000;
          velocidadv = 8000;
          break;

        //entre 29 y 170 representan velocidades en rpm del motor
cc
        default:
          velocidadcc = (int)comando; //se convierte la velocidad
en byte a int

          if (!sentidocc) //si el sentido es izquierda se hace
negativo el valor de velocidad con el que llamar a la función del
motor cc
            velocidadcc = -velocidadcc;
          break;
      }
    }

    //si hay que mover el motor horizontal se llama a la función
de pulsos con los parámetros determinados por el comando recibido
    if (moverh)
      pulso_pap(true, sentidoh, velocidadh);
```

```
    //si hay que mover los motores verticales se llama a la
función de pulsos con los parámetros determinados por el comando
recibido
    if (moverv)
      pulso_pap(false, sentidov, velocidadv);

    //si llega el comando 1 o no se ha terminado el movimiento del
servo
    if (!fin_mov_servo)
    {
      fin_mov_servo = mover_servo(intervalo_servo, objetivo); //se
llama a la función de movimiento del servo
    }

    //si el motor cc está encendido
    if (encendercc)
      motorcc(velocidadcc); //se llama a su función con la
velocidad determinada por los comandos
    else //si debe estar apagado se ponen a LOW sus entradas
    {
      digitalWrite(IN1, LOW);
      digitalWrite(IN2, LOW);
    }

  } while (comando); //bucle continuo hasta recibir 0
}
```

Archivo movimientos

```
/*********************************************************
Funciones llamadas desde el modo automático que realizan
una operación de movimiento de los motores para cumplir
con la acción solicitada por el usuario.
*********************************************************/

/*********************************************************
Función para llevar el sistema de carga/descarga al
origen de forma que se conozca su posición.
Sin entradas.
Salida: bool indicando si se ha completado o se ha abortado
        la operación.
*********************************************************/
bool calibrar()
{
  while ((!fch_pulsado || !fcv_pulsado)) //mientras no se han
pulsado los dos finales de carrera se permanece en el bucle
  {
    if (!fch_pulsado) //si no se ha pulsado el final de carrera
horizontal
      pulso_pap(true, false, 10000); //se llama a la función de
los paso a paso para movimiento horizontal hacia la derecha

    if (!fcv_pulsado) //si no se ha pulsado el final de carrera
vertical
      pulso_pap(false, true, 10000); //se llama a la función de
los paso a paso para movimiento vertical hacia abajo
```

```
    if (!digitalRead(boton_azul)) //si se pulsa el botón azul
mientras se está calibrando se para el movimiento
    {
      bip(300, 700);
      delay(200);
      lcd.clear(); //se pregunta qué se desea hacer
      lcd.print("Rojo: Reanudar");
      lcd.setCursor(0, 1);
      lcd.print("Blanco: Abortar");

      while (digitalRead(boton_rojo) &&
digitalRead(boton_blanco)); //se espera a que se elija opción con
los botones

      if (!digitalRead(boton_rojo)) //si se escoge reanudar se
vuelve a entrar en el bucle continuando el movimiento
      {
        bip(1000, 200);
        delay(200);
        lcd.clear();
        lcd.print("Operacion");
        lcd.setCursor(0, 1);
        lcd.print("reanudada");
      }
      else //si se escoge abortar se sale de la función
devolviendo false para indicar que no se ha terminado la
calibración
      {
        bip(1000, 200);
        delay(200);
        return false;
      }
    }

    if (Serial.available()) //si se introduce un comando se para
el movimiento de los paso a paso y se muestran las opciones
    {
      comando = Serial.read();
      bip(300, 700);
      Serial.println("");
      Serial.println("Calibracion detenida");
      Serial.println("1-Reanudar");
      Serial.println("Otro-Abortar");
      Serial.println("");
      while (!Serial.available()); //se espera a que se elija la
opción

      comando = Serial.parseInt(); //se lee el comando y se
traduce el valor del código ASCII
      Serial.println(comando);

      if (comando == 1) //si se escoge reanudar se vuelve a entrar
en el bucle continuando el movimiento
      {
        bip(1000, 200);
        Serial.println("Operacion reanudada");
      }
```

```
      else //si se escoge abortar se sale de la función
devolviendo false para indicar que no se ha terminado la
calibración
      {
        bip(1000, 200);
        return false;
      }
    }
  }

  //si se pulsan los dos finales de carrera se sale del bucle, se
ponen a cero los
  //contadores de pulsos y se sale devolviendo true indicando así
que se ha calibrado correctamente
  pulsosh = 0;
  pulsosv = 0;
  return true;
}

/*****************************************************************
************************
Función para detectar la presencia o no de objetos en los
diferentes cajones o introducir
manualmente los cajones ocupados. Llamada al inicio del modo
automático para no permitir
meter nada en un cajón ocupado o sacar de un cajón vacío.
Entradas: bool que indique si se va a hacer reconocimiento manual
o automático.
Salida: bool que indica si se ha terminado el proceso
satisfactoriamente o se ha abortado.
*****************************************************************
************************/
bool reconocimiento(bool manual)
{
  bool repetir; //si se introduce un dato mal se puede repetir la
introducción de datos
  i = 0; //iterador para los mensajes del LCD
  float distancia; //distancia a objeto detectada por sensor de
ultrasonidos
  int num_cajon; //número de cajón actual

  do {
    repetir = false; //si no se indica lo contrario no se repite
el bucle de reconocimiento de cajones

    if (manual) //si se entra a la función con reconocimiento
manual
    {
      if (interfaz == 1) //caso de interfaz por cuadro de mandos
      {
        for (int j = 1; j < 10; j++) //bucle de 9 iteraciones para
los 9 cajones
        {
          //se pregunta si cada cajón está ocupado o no
          lcd.clear();
          lcd.print("Cajon ");
          lcd.print(j); //según la  iteración se pregunta por el
cajón correspondiente
          lcd.print(" Ocupado?");
```

```
          lcd.setCursor(0, 1);
          lcd.print("RojoSi BlancoNo"); //rojo para el caso de
estar ocupado y blanco si no

          //se espera a pulsar algún botón
          while (digitalRead(boton_rojo) &&
digitalRead(boton_blanco) && digitalRead(boton_azul));

          //si se pulsa rojo se guarda true en el elemento del
vector de cajones que toca
          if (!digitalRead(boton_rojo))
          {
            bip(1000, 200);
            delay(200);
            cajones[j - 1] = true; //el vector tiene elementos 0-8
y los cajones números 1-9
          }
          //si se pulsa blanco se guarda false en el elemento del
vector de cajones que toca
          else if (!digitalRead(boton_blanco))
          {
            bip(1000, 200);
            delay(200);
            cajones[j - 1] = false;
          }
          //si se pulsa azul se aborta y se sale devolviendo false
para indicar que se debe salir al menú de elección de modo
          else
          {
            bip(500, 200);
            delay(200);
            return false;
          }
        }
      }
      else //caso de interfaz por puerto serie
      {
        //se pide introducir los cajones ocupados
        Serial.println("Introduzca los cajones ocupados");
        Serial.println("Para terminar pulse 0");

        //se inicializan todos los cajones como vacíos
        for (int i = 0; i < 9; i++)
          cajones[i] = false;

        do //bucle ejecutado hasta que un comando sea cero
        {
          while (!Serial.available()); //se espera a que llegue un
comando
          bip(1000, 200);
          comando = Serial.parseInt(); //se lee y transforma el
comando
          Serial.println(comando);
          if (comando > 0 && comando < 10) //si el comando está
entre 1 y 9
            cajones[comando - 1] = true; //se pone como lleno el
cajón correspondiente
        } while (comando);
      }
```

```
    }
    else //si se entra a la función con reconocimiento automático
    {
      if (calibrar()) //se calibra para partir del origen el
reconocimiento y si la calibración es correcta se empieza
      {
        //según la interfaz se envían mensajes por el LCD o por el
puerto serie
        if (interfaz == 1)
        {
          lcd.clear();
          lcd.print("Reconociendo");
          lcd.setCursor(0, 1);
          lcd.print("el almacen");
        }
        else
          Serial.println("Reconociendo el almacen");

        for (int i = 1; i < 10; i++) //bucle de 9 iteraciones para
los 9 cajones
        {
          switch (i) //según la  iteración se va al cajón
correspondiente moviendo los paso a paso
          {
            case 1:
              //se desplaza el transelevador hasta la posición
correspondiente al cajón 3
              //si se pulsa el botón azul o se envía un comando
(depende de la interfaz) se aborta el movimiento saliendo del
bucle
              while (pulsosh < columna3 && digitalRead(boton_azul)
&& !Serial.available())
                pulso_pap(true, true, 10000); //se manda mover el
paso a paso horizontal hacia la izquierda hasta la posición de la
columna 3

              num_cajon = 2; //se indica que el cajón al que se ha
ido es el 3 (en el vector la posición 2
              break;

            case 2:
              while (pulsosv < fila2 - margen &&
digitalRead(boton_azul) && !Serial.available())
                pulso_pap(false, false, 10000);

              num_cajon = 5;
              break;

            case 3:
              while (pulsosv < fila3 - margen &&
digitalRead(boton_azul) && !Serial.available())
                pulso_pap(false, false, 10000);

              num_cajon = 8;
              break;

            case 4:
              while (pulsosh < columna2 && digitalRead(boton_azul)
&& !Serial.available())
```

```
                pulso_pap(true, true, 10000);

              num_cajon = 7;
              break;

            case 5:
              while (pulsosh < columna1 && digitalRead(boton_azul)
&& !Serial.available())
                pulso_pap(true, true, 10000);

              num_cajon = 6;
              break;

            case 6:
              while (pulsosv > fila2 - margen &&
digitalRead(boton_azul) && !Serial.available())
                pulso_pap(false, true, 10000);

              num_cajon = 3;
              break;

            case 7:
              while (pulsosv > fila1 - margen &&
digitalRead(boton_azul) && !Serial.available())
                pulso_pap(false, true, 10000);

              num_cajon = 0;
              break;

            case 8:
              while (pulsosh > columna2 && digitalRead(boton_azul)
&& !Serial.available())
                pulso_pap(true, false, 10000);

              num_cajon = 1;
              break;

            case 9:
              while (pulsosv < fila2 - margen &&
digitalRead(boton_azul) && !Serial.available())
                pulso_pap(false, false, 10000);

              num_cajon = 4;
              break;
          }

          if (!digitalRead(boton_azul)) //si se ha pulsado el
botón azul se para el movimiento que estuviera y se pregunta qué
hacer
          {
            bip(300, 700);
            delay(200);
            lcd.clear();
            lcd.print("Rojo: Reanudar");
            lcd.setCursor(0, 1);
            lcd.print("Blanco: Abortar");

            while (digitalRead(boton_rojo) &&
digitalRead(boton_blanco)); //se espera a que se elija
```

```
            bip(1000, 200);

            //si se escoge reanudar se decrementa el contador del
bucle for para que se continúe el movimiento abortado en lugar
            //de pasar al siguiente
            if (!digitalRead(boton_rojo))
            {
              delay(200);
              lcd.clear();
              lcd.print("Operacion");
              lcd.setCursor(0, 1);
              lcd.print("reanudada");
              i--;
            }
            else //si se decide abortar se sale de la función
devolviendo false para que se salga al menú de modos
            {
              delay(200);
              return false;
            }
          }

          if (Serial.available()) //mismo procedimiento si se
envía un comando que si se pulsa el botón azul
          {
            bip(300, 700);
            comando = Serial.read();
            Serial.println("");
            Serial.println("1-Reanudar");
            Serial.println("Otro-Abortar");
            Serial.println("");
            while (!Serial.available());

            comando = Serial.parseInt();
            Serial.println(comando);
            bip(1000, 200);
            if (comando == 1)
            {
              Serial.println("Operacion reanudada");
              i--;
            }
            else
              return false;
          }

          while (true) //al finalizar cada movimiento (llegar a un
cajón) se comprueba si está ocupado o no con el sensor de
ultrasonidos
          {
            distancia = ultrasonidos(false); //se toma medida del
sensor de ultrasonidos
            lcd.clear();
            lcd.print(distancia);
            if (distancia < 11) //si es menor a 11cm se considera
que hay algo en el cajón pero se realiza una segunda comprobación
            {
              //si la segunda medida coincide se guarda en el
vector de cajones que el cajón actual está ocupado y se sale del
bucle de medición
```

```
              if (ultrasonidos(false) < 11)
              {
                cajones[num_cajon] = true;
                bip(1000, 200);
                break;
              }
            }
            else //de igual forma se hace cuando se lee que no hay
nada en el cajón
            {
              if (ultrasonidos(false) > 11)
              {
                cajones[num_cajon] = false;
                break;
              }
            }
          } //si no coinciden las dos medidas se repite el bucle
        }
      }
      else //si la calibración inicial se ha abortado se devuelve
false indicando que se vuelva al menú de modos
        return false;
    }

    comando = 0; //reseteo de la variable que guarda el último
comando recibido por el puerto serie

    if (interfaz == 1) //caso de interfaz por cuadro de mandos
    {
      i = 0; //reseteo del iterador para el LCD
      //se espera a que se pulse un botón para confirmar o
rechazar los valores introducidos
      while (digitalRead(boton_rojo) && digitalRead(boton_blanco)
&& digitalRead(boton_azul))
      {
        //mensajes en display mostrados alternativamente
        if (millis() - t_cambio > intervalo) //si ha transcurrido
el intervalo entre cambios
        {
          switch (i) //se visualiza en el LCD lo que toca
          {
            case 0:
              lcd.clear();
              lcd.print("Ocupados: ");
              if (cajones[0] == true)
                lcd.print("1"); //el 1 se muestra (si está
ocupado) en la primera línea ya que no caben todos en la segunda
              lcd.setCursor(0, 1);
              for (int j = 2; j < 10; j++) //se muestran los
números de los cajones supuestamente ocupados
              {
                if (cajones[j - 1] == true)
                {
                  lcd.print(j);
                  lcd.print(" ");
                }
              }
              break;
```

```
            case 1:
              lcd.clear();
              lcd.print("Correcto?");
              lcd.setCursor(0, 1);
              lcd.print("RojoSi BlancoNo");
              break;

            case 2:
              lcd.clear();
              lcd.print("Azul: Reelegir");
              lcd.setCursor(0, 1);
              lcd.print("modo");
              break;
          }

          i++; //se incrementa el contador para que cambie el
mensaje
          if (i == 3) //si se llega a 3 se resetea el contador
            i = 0;
          t_cambio = millis(); //se actualiza la variable t_cambio
con el instante del cambio actual
        }
      }
    }
    else //caso de interfaz por puerto serie se muestran los
cajones ocupados y se pregunta si es correcto esperando a recibir
un comando
    {
      Serial.print("Cajones ocupados: ");
      for (int j = 1; j < 10; j++) //se muestran los números de
los cajones supuestamente ocupados
      {
        if (cajones[j - 1] == true)
        {
          Serial.print(j);
          Serial.print(" ");
        }
      }
      Serial.println("");
      Serial.println("Es correcto?");
      Serial.println("1-Si");
      Serial.println("2-No");
      Serial.println("Otro-Reelegir modo");
      Serial.println("");
      while (!Serial.available());

      comando = Serial.parseInt();
      Serial.println(comando);
    }

    //si se pulsa blanco o se envía 2 se ordena repetir la
introducción de los datos de los cajones por algún error
    if (!digitalRead(boton_blanco) || comando == 2)
    {
      bip(300, 200);
      delay(200);
      repetir = true;
    }
```

```
    //si se pulsa rojo o se envía 1 se confirma que los datos son
correctos y se sale de la función devolviendo true
    else if (!digitalRead(boton_rojo) || comando == 1)
    {
      bip(1000, 200);
      delay(200);

      lleno = true; //se comprueba si todos los cajones están
ocupados para determinar si el almacén está lleno
      for (int j = 0; j < 9; j++)
      {
        if (!cajones[j]) //en caso de haber un cajón vacío el
almacén ya no está lleno y se puede salir de la comprobación
        {
          lleno = false;
          break;
        }
      }

      if (!lleno) //si no está lleno se comprueba si está vacío
      {
        vacio = true;
        for (int j = 0; j < 9; j++)
        {
          if (cajones[j])  //en caso de haber un cajón ocupado el
almacén ya no está vacío y se puede salir de la comprobación
          {
            vacio = false;
            break;
          }
        }
      }

      if (manual)
      {
        if (interfaz == 1)
        {
          while (!calibrar()) //se lleva al sistema al origen
          {
            //si se aborta la calibración se espera a que se esté
listo para retomar dicha operación
            lcd.clear();
            lcd.print("Pulse un boton");
            lcd.setCursor(0, 1);
            lcd.print("para calibrar");

            //al pulsar un botón se continúa con la calibración
            while (digitalRead(boton_rojo) &&
digitalRead(boton_blanco) && digitalRead(boton_azul));
            bip(1000, 200);
            delay(200);
          }
        }
        else
        {
          while (!calibrar()) //se lleva al sistema al origen
          {
            Serial.println("Introduzca un comando para seguir con
la calibración");
```

```
            while (!Serial.available());
            comando = Serial.read();
            bip(1000, 200);
          }
        }
      }
      return true;
    }
    //si se pulsa azul u otro comando se ordena salir devolviendo
false para que se vuleva al menú
    else
    {
      bip(500, 200);
      delay(200);
      return false;
    }
  } while (repetir); //si se ha ordenado repetir se ordena
reiniciar el bucle de la función
}

/*****************************************************************
******************
Función para coger o dejar un objeto en un cajón determinado o en
la cinta.
Entradas:
        -int con el número del cajón (1-9).
        -bool indicando si se va a coger o dejar un palé.
Salida: bool que indica si se ha completado el proceso (true) o se
ha abortado (false)
*****************************************************************
******************/
bool coger_dejar_objeto(int cajon, bool coger)
{
  int fila, columna; //valor en pasos de los motores de la
posición de la fila y columna correspondiente al cajón
  bool sentidoh, sentidov; //sentido de movimiento de los motores
paso a paso

  switch (cajon) //según sea el cajón se inicializan fila y
columna al valor correspondiente a dicho cajón
  {
    case 1:
      if (coger) //si se va a "coger", el valor de pasos del
movimiento vertical se pone más bajo que si se va a "dejar"
        fila = fila1 - margen;
      else
        fila = fila1;
      columna = columna1;
      break;

    case 2:
      if (coger)
        fila = fila1 - margen;
      else
        fila = fila1;
      columna = columna2;
      break;

    case 3:
```

```
  if (coger)
    fila = fila1 - margen;
  else
    fila = fila1;
  columna = columna3;
  break;

case 4:
  if (coger)
    fila = fila2 - margen;
  else
    fila = fila2;
  columna = columna1;
  break;

case 5:
  if (coger)
    fila = fila2 - margen;
  else
    fila = fila2;
  columna = columna2;
  break;

case 6:
  if (coger)
    fila = fila2 - margen;
  else
    fila = fila2;
  columna = columna3;
  break;

case 7:
  if (coger)
    fila = fila3 - margen;
  else
    fila = fila3;
  columna = columna1;
  break;

case 8:
  if (coger)
    fila = fila3 - margen;
  else
    fila = fila3;
  columna = columna2;
  break;

case 9:
  if (coger)
    fila = fila3 - margen;
  else
    fila = fila3;
  columna = columna3;
  break;

case 0:
  if (coger)
    fila = cintav - margen;
  else
```

```
        fila = cintav;
      columna = cintah;
      break;
  }

  for (int j = 1; j < 5; j++) //bucle en el que se efectúan los
cuatro movimientos que componen coger o dejar un palet
  { //el sentido del bucle es que se pueda abortar cada uno de los
cuatro movimientos sin repetir código
    switch (j)
    {
      case 1: //el primer movimiento es el desplazamiento hasta el
cajón correspondiente
        //según la posición actual del transelevador, éste tendrá
que moverse en un sentido u otro (izq-dcha y arriba-abajo)
        //para llegar a la posición del cajón
        if (pulsosh < columna) //para ello se compara la posición
en pulsos actual con la del cajón
          sentidoh = true;
        else
          sentidoh = false;

        if (pulsosv < fila)
          sentidov = false;
        else
          sentidov = true;

        //mientras no se llegue al cajón o se ordene parar se
ordena el movimiento de los paso a paso
        while ((pulsosh != columna || pulsosv != fila) &&
digitalRead(boton_azul) && !Serial.available())
        {
          if (pulsosh != columna)
            pulso_pap(true, sentidoh, 5000);

          if (pulsosv != fila)
            pulso_pap(false, sentidov, 5000);
        }
        break;

        //el movimiento 2 consiste en la introducción de la
transpaleta en el cajón llamando a la función de movimiento del
servo
        //hasta que se complete el movimiento
      case 2:
        while (!mover_servo(intervalo_servo, 30) &&
digitalRead(boton_azul) && !Serial.available());
        break;

        //el movimiento 3 consiste en la ligera elevación o
descenso (movimiento de los paso a paso verticales) para coger el
palet
        //o para soltarlo (según el tipo de operación requerida)
      case 3:
        if (coger)
        {
          while (pulsosv != fila + margen &&
digitalRead(boton_azul) && !Serial.available())
            pulso_pap(false, false, 2000);
```

```
        }
        else
        {
          while (pulsosv != fila - margen &&
digitalRead(boton_azul) && !Serial.available())
            pulso_pap(false, true, 2000);
        }
        break;

        //por último se retrae la transpaleta moviendo el servo en
sentido contrario
      case 4:
        while (!mover_servo(intervalo_servo, 180) &&
digitalRead(boton_azul) && !Serial.available()); //se llama a la
función de movimiento del servo hasta que acabe el movimiento
        break;
    }

    //procedimiento similar al de la función "reconocimiento" para
el caso de pulsar el botón azul en mitad de un movimiento
    if (!digitalRead(boton_azul))
    {
      bip(300, 700);
      delay(200);
      lcd.clear();
      lcd.print("Rojo: Reanudar");
      lcd.setCursor(0, 1);
      lcd.print("Blanco: Abortar");

      while (digitalRead(boton_rojo) &&
digitalRead(boton_blanco));
      bip(1000, 200);

      if (!digitalRead(boton_rojo))
      {
        delay(200);
        lcd.clear();
        lcd.print("Operacion");
        lcd.setCursor(0, 1);
        lcd.print("reanudada");
        j--;
      }
      else //si se ordena abortar se pide vaciar la pinza antes de
continuar por si acaso ha sucedido un accidente
      {
        delay(200);
        lcd.clear();
        lcd.print("Vacie pinza y");
        lcd.setCursor(0, 1);
        lcd.print("pulse un boton");

        while (digitalRead(boton_rojo) &&
digitalRead(boton_blanco) && digitalRead(boton_azul));
        bip(1000, 200);
        delay(200);
        while (!mover_servo(intervalo_servo, 180)); //al salir se
retrae el servo para ponerlo en su posición habitual
        return false; //se devuelve false indicando que no se ha
terminado la operación de forma satisfactoria
```

```
      }
    }

    //mismo procedimiento en caso de enviar un comando por el
puerto serie
    if (Serial.available())
    {
      comando = Serial.read();
      Serial.println("");
      Serial.println("1-Reanudar");
      Serial.println("Otro-Abortar");
      Serial.println("");
      while (!Serial.available());

      comando = Serial.parseInt();
      Serial.println(comando);
      bip(1000, 200);
      if (comando == 1)
      {
        Serial.println("Operacion reanudada");
        j--;
      }
      else
      {
        Serial.println("Vacie la pinza y pulse una tecla");
        while (!Serial.available());
        comando = Serial.read();
        bip(1000, 200);
        while (!mover_servo(intervalo_servo, 180));
        return false;
      }
    }
  }
  return true; //si se ha compleatado el proceso se devuelve true
}

/*****************************************************************
************
Función que lleva a cabo la introducción de un objeto que llega a
la cinta en
un cajón del almacén si así lo quiere el usuario. Llamada al
llegar un nuevo
objeto o al seleccionarlo el usuario en el menú de acciones del
modo
automático habiendo un objeto en espera.
Sin entradas.
Salida: bool que valdrá true si el usuario ha decidido no
introducir el objeto
        y false en caso contrario
*****************************************************************
***********/
bool meter_en_almacen()
{
  int  j = 0; //iterador para recorrer el vector de cajones
  bool cajon_determinado = false; //variable que determina cuando
el cajón en el que introducir ha sido determinado
  bool escoger; //variable para indicar si se desea escoger un
cajón en el que introducir o no
  float distancia; //distancia medida por ultrasonidos
```

```
  i = 0; //reseteo del iterador para los mensajes del LCD
  intervalo = 0; //para que el primer mensaje del LCD se muestre
sin espera
  comando = 0; //reseteo de la variable de comandos del puerto
serie

  if (interfaz == 1) //caso de interfaz por cuadro de mandos
  {
    //se pregunta lo que se desea realizar y se espera a que se
elija con la pulsación de un botón
    while (digitalRead(boton_rojo) && digitalRead(boton_blanco) &&
digitalRead(boton_azul))
    {
      if (millis() - t_cambio > intervalo) //si ha transcurrido el
intervalo entre cambios
      {
        intervalo = 1500; //tiempo entre cambios para resto de
iteraciones (tras la primera)

        switch (i) //se visualiza en el LCD lo que toca
        {
          case 0:
            lcd.clear();
            lcd.print("Detectado objeto");
            lcd.setCursor(0, 1);
            lcd.print("entrante");
            break;

          case 1:
            lcd.clear();
            lcd.print("Rojo:");
            lcd.setCursor(0, 1);
            lcd.print("Escoger cajon");
            break;

          case 2:
            lcd.clear();
            lcd.print("Blanco: cajon");
            lcd.setCursor(0, 1);
            lcd.print("libre cualquiera");
            break;

          case 3:
            lcd.clear();
            lcd.print("Azul: Obviar");
            lcd.setCursor(0, 1);
            lcd.print("objeto");
            break;
        }

        i++; //se incrementa el contador para que cambie el
mensaje
        if (i == 4) //si se llega a 4 se resetea el contador
          i = 0;
        t_cambio = millis(); //se actualiza la variable t_cambio
con el instante del cambio actual

        if (ultrasonidos(true) > 20) //se comprueba si sigue el
objeto en la cinta
```

```
          if (ultrasonidos(true) > 20) //se toman dos medidas para
evitar datos erróneos
          {
            //si el objeto ya no está se sale de la función al no
haber nada que meter
            bip(300, 700);
            lcd.clear();
            lcd.print("El objeto ha");
            lcd.setCursor(0, 1);
            lcd.print("desaparecido");
            delay(2000);
            return false; //se devuelve false ya que no se ha
obviado el objeto sino que ha desaparecido
          }
      }
    }
  }
  else //interfaz por monitor serie
  {
    //se muestra el mismo menú de opciones y se espera a que se
introduzca un comando
    Serial.println("");
    Serial.println("Detectado objeto entrante");
    Serial.println("1-Escoger cajon donde introducirlo");
    Serial.println("2-Introducirlo en cajon cualquiera");
    Serial.println("Otro-Obviar objeto");
    Serial.println("");

    while (!Serial.available()) //mientras se espera a que se
elija se comprueba que el objeto siga en la cinta
    {
      if (ultrasonidos(true) > 20)
        if (ultrasonidos(true) > 20)
        {
          //si se quita el objeto se sale de la función
devolviendo false
          bip(300, 700);
          Serial.println("El objeto ha desaparecido");
          return false;
        }
    }
    bip(1000, 200);
    comando = Serial.parseInt(); //se lee el comando cuando llega
    Serial.println(comando);
  }

  if (!digitalRead(boton_rojo) || comando == 1) //si se elige
meter el objeto y escoger cajón
  {
    bip(1000, 200);
    delay(200);
    escoger = true; //se indica que se va a querer escoger cajón
cuando se haya cogido el objeto
  }
  else if (!digitalRead(boton_blanco) || comando == 2) //si se
elige meter el objeto pero no escoger cajón
  {
    bip(1000, 200);
    delay(200);
```

```
    escoger = false; //se indica que no se va a escoger
    do //mientras no se encuentre un cajón vacío se repite el
bucle en el que en cada iteración se comprueba el estado de un
cajón
    {
      if (!cajones[j]) //si el cajón que toca está vacío
      {
        cajon = j + 1; //se determina que se introducirá el palet
en este cajón
        cajon_determinado = true;
      }
      j++; //se incrementa el iterador para pasar a comprobar el
siguiente cajón si este estaba ocupado
    } while (!cajon_determinado);
  }
  else //si se escoge obviar el objeto
  {
    bip(500, 200);
    if (interfaz == 1) //se indica que se va a obviar el objeto
    {
      lcd.clear();
      lcd.print("Objeto obviado");
      delay(2000);
    }
    else
      Serial.println("Objeto obviado");

    return true; //se sale de la función indicando que no se ha
introducido el objeto
  }

  while (true) //bucle en el que se mueve la cinta hasta que el
ultrasonidos detecte que el palet ya está en disposición de ser
recogido
  {
    motorcc(-40); //se llama a la función de movimiento del motor
de continua con una velocidad de 40rpm en sentido antihorario
    distancia = ultrasonidos(true); //se lee el ultrasonidos
    lcd.clear();
    lcd.print(distancia);
    if (distancia < 3.1 && distancia > 1) //si la distancia es la
apropiada (3.1) (el 1 es para evitar falsas medidas como 0)
    {
      distancia = ultrasonidos(true); //se realiza una segunda
comprobación
      if (distancia < 3.1 && distancia > 1)
        break; //en caso de que el palet esté en el lugar
requerido se sale del bucle
    }
  }

  //se para el motor de la cinta
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);

  if (interfaz == 1)
  {
    lcd.clear();
    lcd.print("Cogiendo objeto");
```

```
    lcd.setCursor(0, 1);
    lcd.print("de la cinta");
  }
  else
    Serial.println("Cogiendo objeto de la cinta");

  //se llama a la función que coge y deja objetos indicando que
debe coger (true) de la cinta (valor 0)
  if (coger_dejar_objeto(0, true)) //si se realiza la función sin
abortar
  {
    if (escoger) //en caso de que se haya elegido la opción del
cajón
    {
      if (!meter()) //se llama a la función que pide el cajón y a
continuación mete el objeto en dicho cajón
        return false; //si se aborta el proceso se sale de la
función
    }
    else //si ya se ha determinado el cajón de forma automática
    {
      if (interfaz == 1)
      {
        lcd.clear();
        lcd.print("Dejando objeto");
        lcd.setCursor(0, 1);
        lcd.print("en el cajon ");
        lcd.print(cajon);
      }
      else
      {
        Serial.println("Dejando objeto en el cajon ");
        Serial.println(cajon);
      }

      //se llama a la función que coge o deja objetos sin
preguntar el cajón indicando el cajón y que la acción a realizar
es "dejar"
      if (!coger_dejar_objeto(cajon, false))
        return false; //si se aborta el proceso de dejar el palet
se sale de la función
    }

    cajones[cajon - 1] = true; //si se ha completado la
introducción del palet en un cajón se indica que ese cajón está
ocupado

    lleno = true; //se comprueba si el almacén se encuentra lleno
    for (int j = 0; j < 9; j++)
    {
      if (!cajones[j]) //con un cajón vacío que haya el almacén ya
no está lleno y se sale de la comprobación
      {
        lleno = false;
        break;
      }
    }
  }
```

```
  return false;  //se sale de la función indicando que no se ha
obviado el objeto dando igual si se ha completado o no el
propósito de
  //la función ya que en ambos casos se vuelve al menú de acciones
}

/*****************************************************************
***************
Función para sacar del almacén un elemento a través de la cinta
transportadora.
Se llama al escoger la opción de sacar en el menú de acciones del
modo automático.
No tiene entradas ni salidas.
*****************************************************************
**************/
void sacar_del_almacen()
{
  float distancia; //distancia medida con el ultrasonidos de la
cinta

  if (sacar()) //se llama a la función que pregunta el cajón y
coge el palé de él
  { //si dicha función se aborta se sale de esta función también y
si no se continúa
    cajones[cajon - 1] = false; //se indica que el cajón del que
se ha extraído está ahora vacío

    vacio = true; //se comprueba que el almacén no esté totalmente
vacío
    for (int j = 0; j < 9; j++)
    {
      if (cajones[j]) //con un cajón ocupado que haya el almacén
ya no está lleno y se sale de la comprobación
      {
        vacio = false;
        break;
      }
    }

    if (interfaz == 1)
    {
      lcd.clear();
      lcd.print("Sacando objeto");
      lcd.setCursor(0, 1);
      lcd.print("del almacen ");
    }
    else
      Serial.println("Sacando objeto del almacen");

    //con el palet sacado se llama a la función que coge y deja
objetos indicando que se desea dejar (false) el palé en la cinta
(0)
    if (coger_dejar_objeto(0, false))
    { //si se deja correctamente (sin abortar el proceso) se entra
en un bucle que ordena el movimiento de la cinta
      while (true)
      {
        motorcc(40); //se ordena el movimiento del motor cc a
40rpm en sentido horario
```

```
        //si tras dos comprobaciones del ultrasonidos de la cinta
se determina que el palet ha llegado al otro extremo se sale del
bucle
        distancia = ultrasonidos(true);
        if (distancia > 13)
        {
          distancia = ultrasonidos(true);
          if (distancia > 13)
            break;
        }
      }
      //se para el motor
      digitalWrite(IN1, LOW);
      digitalWrite(IN2, LOW);
      bip(1000, 200);

      //se pide sacar de la cinta el palet
      if (interfaz == 1)
      {
        lcd.clear();
        lcd.print("Quite el objeto");
        lcd.setCursor(0, 1);
        lcd.print("de la cinta");
      }
      else
        Serial.println("Extraiga el objeto de la cinta");

      //se espera a que el ultrasonidos detecte que el objeto ya
no está para poder proseguir saliendo de la función y yendo al
menú de acciones
      while (true)
      {
        if (ultrasonidos(true) > 20)
          if (ultrasonidos(true) > 20)
            break;
      }
      bip(1000, 200);
      if (interfaz != 1)
      {
        Serial.println("Gracias");
        delay(1000);
      }
    }
  }
}

/*****************************************************************
******
Función que mueve un palé de un cajón a otro del almacén. Llamada
cuando
se escoge la opción de mover en el menú de acciones del modo
automático.
No tiene entradas ni salidas.
*****************************************************************
*****/
void mover_en_almacen()
{
  if (sacar()) //se llama a la función que saca un palet de un
cajón que se escoge
```

```
  { //si no se aborta dicha función
    cajones[cajon - 1] = false; //se indica que ese cajón está
vacío ahora

    if (meter()) //se llama a la función que mete un palet en un
cajón que se escoge
      cajones[cajon - 1] = true; //se indica que ese cajón está
ocupado si se finaliza correctamente el proceso
  }
}

/*****************************************************************
**********************
Función que extrae un palé de un cajón preguntando previamente el
cajón del que hacerlo.
Sin entradas.
Salida: bool indicando si el proceso se ha completado (true) o se
ha abortado (false)
*****************************************************************
**********************/
bool sacar()
{
  bool repetir; //variable para indicar si se debe repetir la
elección del cajón

  do //bucle en el que se pide escoger un cajón hasta que se
selecciones uno ocupado
  {
    repetir = false; //en principio no habrá que repetir
    if (interfaz == 1) //caso interfaz cuadro de mandos
    {
      cajon = escoger_cajon(true); //se llama a la función de
escoger un cajón indicando que se va a extraer de él
      if (!cajones[cajon - 1]) //si el cajón escogido está vacío
se indica instando a elegir de nuevo
      {
        bip(300, 300);
        lcd.clear();
        lcd.print("Cajon vacio");
        lcd.setCursor(0, 1);
        lcd.print("Pruebe de nuevo");
        repetir = true; //se indica que deberá repetirse el bucle
de elección de cajón
        delay(1500);
      }
    }
    else //caso interfaz puerto serie
    {
      Serial.println("Escoja cajon del que sacar");
      while (!Serial.available()); //se espera a que se elija el
cajón con un comando

      cajon = Serial.parseInt(); //se lee el comando
      Serial.println(cajon);
      if (cajon > 0 && cajon < 10) //si el comando es correcto
      {
        if (!cajones[cajon - 1]) //si el cajón escogido está vacío
se indica instando a elegir de nuevo
        {
```

```
          bip(500, 200);
          Serial.println("Cajon vacio, pruebe de nuevo");
          repetir = true;
        }
        else //si el cajón está ocupado se sale del bucle y el
cajón queda seleccionado
          bip(1000, 200);
      }
      else //si el comando es erróneo se vuelve a entrar en el
bucle para pedir otro
      {
        bip(300, 300);
        Serial.println("Valor erroneo");
        repetir = true;
      }
    }
  } while (repetir); //mientras no se haya elegido un cajón
ocupado se vuelve a solicitar un cajón

  if (interfaz == 1)
  {
    lcd.clear();
    lcd.print("Cogiendo objeto");
    lcd.setCursor(0, 1);
    lcd.print("del cajon ");
    lcd.print(cajon);
  }
  else
  {
    Serial.print("Cogiendo objeto del cajon ");
    Serial.println(cajon);
  }

  //se llama a la función que coge y deja palets indicando que se
quiere coger (true) del cajón seleccionado
  return coger_dejar_objeto(cajon, true);
  //al salir se devuelve lo mismo que devuelva esta función que
podrá ser true (proceso exitoso) o false (proceso abortado)
}

/*****************************************************************
**********************
Función que extrae un palé de un cajón preguntando previamente el
cajón del que hacerlo.
Función con la misma estructura que la anterior pero teniendo que
escoger un cajón vacío
y ordenando dejar en lugar de coger.
Sin entradas.
Salida: bool indicando si el proceso se ha completado (true) o se
ha abortado (false)
*****************************************************************
**********************/

bool meter()
{
  bool repetir;

  do
  {
```

```
    repetir = false;
    if (interfaz == 1)
    {
      cajon = escoger_cajon(false);
      if (cajones[cajon - 1])
      {
        lcd.clear();
        lcd.print("Cajon ocupado");
        lcd.setCursor(0, 1);
        lcd.print("Pruebe de nuevo");
        repetir = true;
        delay(1500);
      }
    }
    else
    {
      Serial.println("Escoja cajon en el que introducir");
      while (!Serial.available());

      cajon = Serial.parseInt();
      Serial.println(cajon);
      if (cajon > 0 && cajon < 10)
      {
        if (cajones[cajon - 1])
        {
          bip(300, 300);
          Serial.println("Cajon ocupado, pruebe de nuevo");
          repetir = true;
        }
        else
          bip(1000, 200);
      }
      else
      {
        bip(300, 300);
        Serial.println("Valor erroneo");
        repetir = true;
      }
    }
  } while (repetir);

  if (interfaz == 1)
  {
    lcd.clear();
    lcd.print("Dejando objeto");
    lcd.setCursor(0, 1);
    lcd.print("en el cajon ");
    lcd.print(cajon);
  }
  else
  {
    Serial.print("Dejando el objeto en el cajon ");
    Serial.println(cajon);
  }

  return coger_dejar_objeto(cajon, false);
}
```

ANEXO 3: CÓDIGO ANDROID

En este anexo se adjunta el código desarrollado para la aplicación móvil empezando por el AndroidManifest para continuar con los códigos XML y JAVA de la actividad *lista_bluetooth* y terminar con los de la actividad *MinActivity*.

AndroidManifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="uva.almacenautomatizado">

    <uses-permission android:name="android.permission.BLUETOOTH"
/>
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/icono"
        android:label="Almacén"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait">
        </activity>
        <activity
            android:name=".lista_bluetooth"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"
/>

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

XML lista_bluetooth

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondo">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Selección del Almacén"
        android:id="@+id/textView"
        android:textSize="50sp"
        android:textColor="@android:color/holo_green_light"
        android:layout_alignParentTop="true"
        android:gravity="center"/>

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
        android:layout_below="@+id/textView"
        android:textAlignment="center"
        android:layout_marginTop="15dp"/>

</RelativeLayout>
```

JAVA lista_bluetooth

```
package uva.almacenautomatizado;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;

//Actividad para selección de un dispositivo bluetooth
public class lista_bluetooth extends AppCompatActivity {
```

```
    //Se declara la lista donde aparecerán los dispositivos
    ListView listaDispositivos;
    //Variable para el bluetooth
    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> dispVinculados;
    public static String BLUETOOTH_ADDRESS = "device_address";
    boolean bt_activado=true;

    //Función llamada al inicio de una actividad que realiza
funciones de configuración
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //acciones d ela
superclase
        setContentView(R.layout.activity_lista_bluetooth); //Se
cargan los elementos del layout correspondiente

        //Se declara la lista y se relaciona con la del layout
        listaDispositivos = (ListView)findViewById(R.id.listView);

        //Se comprueba que el dispositivo tiene bluetooth
        myBluetooth = BluetoothAdapter.getDefaultAdapter();

        if(myBluetooth == null) //Si no lo tiene
        {
            //Se muestra un mensaje, indicando al usuario que no
tiene conexión bluetooth disponible
            Toast.makeText(getApplicationContext(), "Bluetooth no
disponible", Toast.LENGTH_LONG).show();

            //Se finaliza la aplicación la aplicación
            finish();
        }
        else if(!myBluetooth.isEnabled()) //Si tiene bluetooth
pero no está activado
        {
            bt_activado=false;
            //Se pregunta al usuario si desea encender el
bluetooth
            Intent turnBTon = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            //Se intenta encender el bluetooth
            startActivityForResult(turnBTon,1);
        }

        //Si el bluetooth ya estaba conectado se muestra la lista
de dispositivos
        if(bt_activado)
            listaDispositivosvinculados();
    }

    //Función llamada al terminar el encendido del bluetooth
    @Override
    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        if (requestCode == 1) {
            //Si se ha conectado el bluetooth correctamente
            if (resultCode == RESULT_OK) {
                listaDispositivosvinculados(); //Se muestra la
```

```
lista de dispositivos
            }
        }
    }

    //Función para mostrar en una lista los dispositivos bluetooth
emparejados
    private void listaDispositivosvinculados()
    {
        dispVinculados = myBluetooth.getBondedDevices(); //Se
buscan los dispositivos
        ArrayList list = new ArrayList(); //Se declara un vector
para los elementos de la lista

        if (dispVinculados.size()>0) //Si se ha encontrado algún
dispositivo
        {
            for(BluetoothDevice bt : dispVinculados)
            {
                list.add(bt.getName() + "\n" + bt.getAddress());
//Se añaden los nombres y direcciones MAC de los disp. vinculados
a la lista
            }
        }
        else //Si no se encuentran dispositivos se envía un
mensaje indicándolo
        {
            Toast.makeText(getApplicationContext(), "No se han
encontrado dispositivos vinculados", Toast.LENGTH_LONG).show();
        }

        final ArrayAdapter adapter = new
ArrayAdapter(this,android.R.layout.simple_list_item_1, list);
        listaDispositivos.setAdapter(adapter); //Se crea un
adapter para poder recoger pulsaciones en elemetnos de la lista

listaDispositivos.setOnItemClickListener(myListClickListener);
//Escucha de pulsación de un elemento de la lista

    }

    //Función de escucha de pulsación de un elemento de la lista
de diapositivos vinculados
    private AdapterView.OnItemClickListener myListClickListener =
new AdapterView.OnItemClickListener()
    {
        public void onItemClick (AdapterView<?> av, View v, int
arg2, long arg3)
        {
            //Se cogen los 17 últimos caracteres del elemento, que
corresponden a la dirección MAC
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Se crea un intent para lanzar la actividad
principal
            Intent i = new Intent(lista_bluetooth.this,
MainActivity.class);
```

```
                //Se pasa la dirección MAC a la actividad principal
                i.putExtra(BLUETOOTH_ADDRESS, address);
                //Se lanza la actividad principal
                startActivity(i);
            }
        };
}
```

XML MainActivity

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondo">

    <uva.almacenautomatizado.JoystickView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:id="@+id/joystick"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:layout_below="@+id/joystick"
        android:layout_marginTop="30dp"
        android:background="@android:color/holo_green_light"
        android:textSize="20sp"
        android:id="@+id/btnservo"
        android:layout_centerHorizontal="true"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:layout_below="@+id/btnservo"
        android:layout_marginTop="40dp"
        android:layout_marginLeft="40dp"
        android:layout_marginStart="40dp"
        android:background="@android:color/holo_blue_dark"
        android:textSize="13sp"
        android:id="@+id/btnmotorcc" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:background="@android:color/holo_blue_dark"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_marginRight="40dp"
        android:layout_marginEnd="40dp"
        android:layout_alignTop="@+id/btnmotorcc"
        android:textSize="13sp"
        android:id="@+id/btnsentido" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="5dp"
        android:paddingRight="10dp"
        android:paddingTop="5dp"
        android:paddingLeft="10dp"
        android:layout_below="@+id/btnmotorcc"
        android:layout_marginTop="40dp"
        android:layout_centerHorizontal="true"
        android:text="VELOCIDAD"
        android:textSize="25sp"
        android:textStyle="bold"
        android:background="@android:color/holo_red_dark"
        android:id="@+id/tvvel"/>

    <SeekBar
        style="@style/Widget.AppCompat.SeekBar.Discrete"
        android:layout_width="300dp"
        android:layout_height="30dp"
        android:max="141"
        android:progress="1"
        android:layout_below="@+id/tvvel"
        android:layout_centerHorizontal="true"
        android:id="@+id/velcc"
        android:background="@android:color/holo_red_dark"
        android:thumb="@drawable/thumb"/>

</RelativeLayout>
```

JAVA MainActivity

```
package uva.almacenautomatizado;

import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.KeyEvent;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.Toast;

import java.io.IOException;
import java.util.UUID;

//Actividad principal de la aplicación
public class MainActivity extends AppCompatActivity implements
View.OnClickListener, JoystickView.JoystickListener {

    //variables globales de la actividad

    //variables para comunicación bluetooth
    String address = null;
    private ProgressDialog progress;
    BluetoothAdapter myBluetooth = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;
    static final UUID myUUID = UUID.fromString("00001101-0000-
1000-8000-00805F9B34FB");

    JoystickView joystick;
    Button btnservo, btnmotorcc, btnsentido;
    SeekBar velcc;

    byte dato, dato_ant; //dato a enviar por bluetooth y dato de
la llamada anterior a la función del joystick
    boolean dentro=false, encendido=false, derecha=false;
//posición del servo, estado del motor cc y su sentido de giro

    //función llamada al activarse la actividad en la que se
realiza la configuración
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //acciones de la
superclase

        Intent newint = getIntent(); //Llamada a esta actividad
desde lista_bluetooth
        address =
newint.getStringExtra(lista_bluetooth.BLUETOOTH_ADDRESS); //Se
recibe la dirección MAC obtenida en la actividad anterior
        setContentView(R.layout.activity_main); //Se cargan los
elementos del layout correspondiente

        new ConnectBT().execute(); //Se llama a la función para
conectar el bluetooth

        //Se asocian los elementos XML a variables java
        joystick=(JoystickView) findViewById(R.id.joystick);
        btnservo=(Button)findViewById(R.id.btnservo);
        btnservo.setText("Meter transpaleta");
        btnmotorcc=(Button)findViewById(R.id.btnmotorcc);
        btnmotorcc.setText("Activar cinta");
        btnsentido=(Button)findViewById(R.id.btnsentido);
        btnsentido.setText("Izquierda");
```

```
        velcc=(SeekBar)findViewById(R.id.velcc);

        //Se asocian los botones a la función de escucha de clicks
        btnservo.setOnClickListener(this);
        btnmotorcc.setOnClickListener(this);
        btnsentido.setOnClickListener(this);

        //Función de la Seekbar llamada al tocar, soltar o ser
movida la barra
        velcc.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            //función de movimiento de la barra
            @Override
            public void onProgressChanged(SeekBar seekBar, int
progress, boolean fromUser) {
                dato=(byte)progress; //se asigna a la variable
dato el valor de la barra
                dato+=29; //el valor de la barra 0-141 se pasa a
29-170 para ser velocidades en rpm
                enviar(dato); //se llama a la función para enviar
el dato por bluetooth
            }

            //en el caso de solo tocar no se hace ninguna acción
            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {

            }

            //en el caso de soltar tampoco se realiza ninguna
acción
            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {

            }
        });
    }

    //Función para enviar comandos por bluetooth (se debe pasar un
byte para enviar)
    public void enviar(byte dato)
    {
        if (btSocket!=null) //si el socket de comunición está
abierto
        {
            try //se intenta enviar el dato
            {
                btSocket.getOutputStream().write(dato);

            }
            catch (IOException e) //si no es posible se lanza un
mensaje de error
            {
                msg("Error");
            }
        }
    }

    //Función de escucha de clicks de los botones
```

```
    @Override
    public void onClick(View v) {
        switch(v.getId()) //se distingue el botón que ha sido
pulsado por su id
        {
            case R.id.btnservo: //si se pulsa el botón del servo
                dentro=!dentro; //se cambia el texto entre
meter/sacar
                if(dentro)
                    btnservo.setText("Sacar transpaleta");
                else
                    btnservo.setText("Meter transpaleta");
                enviar((byte)1); //se envía el comando 1
                break;

            case R.id.btnmotorcc: //si se pulsa el botón del
encendido del motor cc
                encendido=!encendido; //se cambia el texto entre
encender/apagar
                if(encendido)
                    btnmotorcc.setText("Apagar cinta");
                else
                    btnmotorcc.setText("Activar cinta");
                enviar((byte)2); //se envía el comando 2
                break;

            case R.id.btnsentido: //si se pulsa el botón de
sentido de giro
                derecha=!derecha; //se cambia el texto entre
izquierda/derecha
                if(derecha)
                    btnsentido.setText("Derecha");
                else
                    btnsentido.setText("Izquierda");
                enviar((byte)3); //se envía el comando 3
                break;
        }
    }

    //Función para establecer la conexión bluetooth en una tarea
en segundo plano
    private class ConnectBT extends AsyncTask<Void, Void, Void>
    {
        private boolean ConnectSuccess = true; //variable para
indicar correcta conexión

        @Override
        protected void onPreExecute() { //función inicial que
muestra un mensaje para indicar que se está conectando
            progress = ProgressDialog.show(MainActivity.this,
"Conectando...", "Por favor, espere!!!");
        }

        @Override //función principal que realiza la conexión
        protected Void doInBackground(Void... devices) {
            try {
                if (btSocket == null || !isBtConnected) { //si no
está hecha ya la conexión
                    myBluetooth =
```

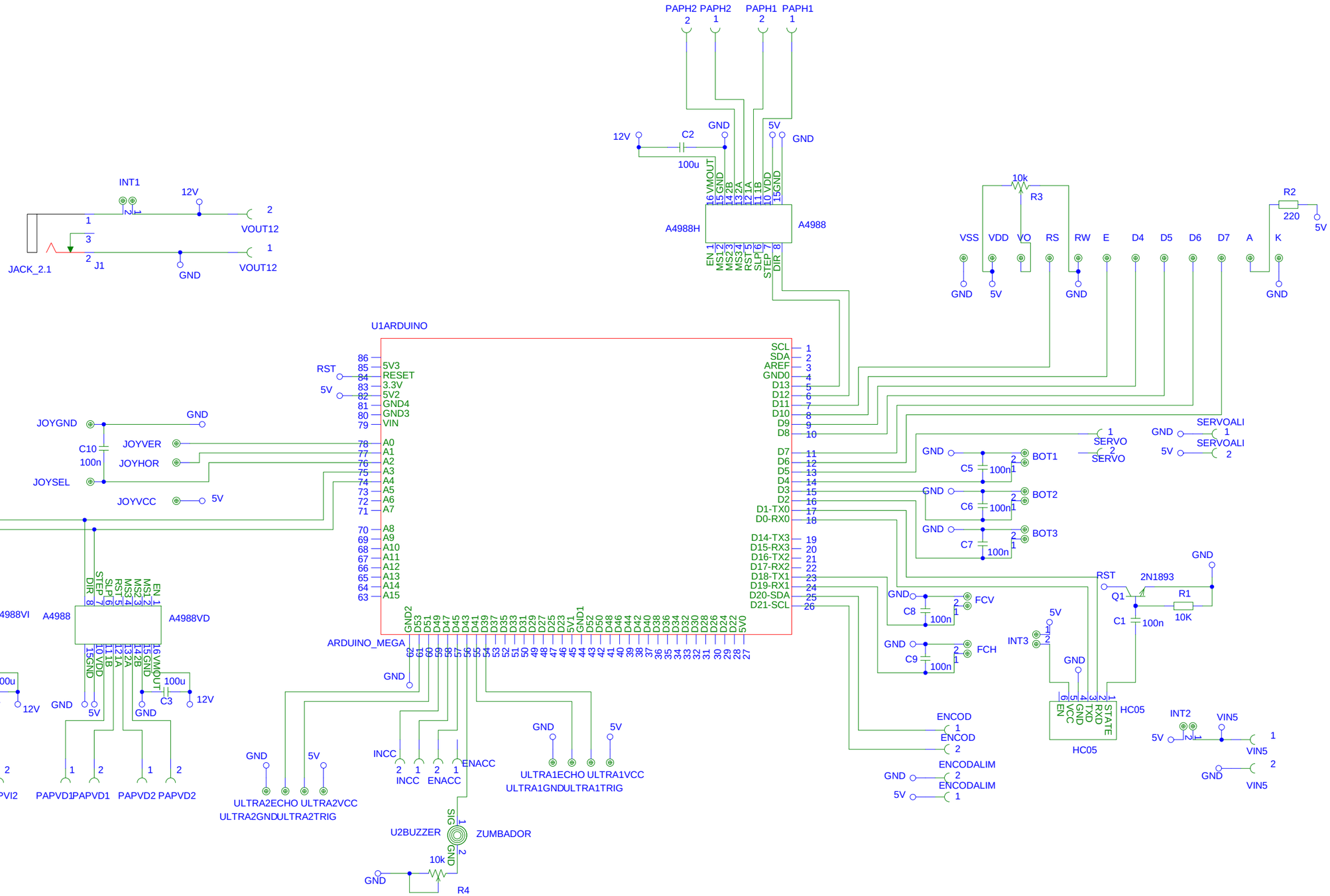
```
BluetoothAdapter.getDefaultAdapter();
                    BluetoothDevice dispositivo =
myBluetooth.getRemoteDevice(address);//Se conecta al dispositivo
                    btSocket =
dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID);

BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                    btSocket.connect(); //se abre el socket de
comunicación
                }
            } catch (IOException e) { //si no se ha podido
realizar la conexión se indica
                ConnectSuccess = false;
            }
            return null;
        }

        @Override //función posterior a la conexión
        protected void onPostExecute(Void result) {
            super.onPostExecute(result);

            if (!ConnectSuccess) { //Si la conexión ha fallado
                msg("Conexión Fallida"); //Se saca un mensaje
indicando el fallo
                finish(); //Finaliza la actividad volviendo a la
actividad de elegir dispositivo
            } else { //Si la conexión es correcta
                msg("Conectado"); //Se indica con un mensaje y
actualizando la variable correspondiente
                isBtConnected = true;
            }
            progress.dismiss(); //Se quita el mensaje que indicaba
que se estaba realizando la conexión
        }
    }

    //Función para sacar mensaje temporal por pantalla
    private void msg(String s) {
        Toast.makeText(getApplicationContext(), s,
Toast.LENGTH_LONG).show();
    }

    //Función llamada al mover el joystick
    @Override
    public void onJoystickMoved(float xPercent, float yPercent,
int id) {

        //según la posición del joystick se envían diferentes
datos por bluetooth
        if(xPercent < -0.6)
        {
            if(yPercent < -0.6)
                dato=4;
            else if(yPercent < -0.3)
                dato=5;
            else if(yPercent < 0.3)
                dato=6;
            else if(yPercent < 0.6)
                dato=7;
```

```
            else
                dato=8;
        }
        else if(xPercent < -0.3)
        {
            if(yPercent < -0.6)
                dato=9;
            else if(yPercent < -0.3)
                dato=10;
            else if(yPercent < 0.3)
                dato=11;
            else if(yPercent < 0.6)
                dato=12;
            else
                dato=13;
        }
        else if(xPercent < 0.3)
        {
            if(yPercent < -0.6)
                dato=14;
            else if(yPercent < -0.3)
                dato=15;
            else if(yPercent < 0.3)
                dato=16;
            else if(yPercent < 0.6)
                dato=17;
            else
                dato=18;
        }
        else if(xPercent < 0.6)
        {
            if(yPercent < -0.6)
                dato=19;
            else if(yPercent < -0.3)
                dato=20;
            else if(yPercent < 0.3)
                dato=21;
            else if(yPercent < 0.6)
                dato=22;
            else
                dato=23;
        }
        else {
            if (yPercent < -0.6)
                dato = 24;
            else if (yPercent < -0.3)
                dato = 25;
            else if (yPercent < 0.3)
                dato = 26;
            else if (yPercent < 0.6)
                dato = 27;
            else
                dato = 28;
        }

        //Se intenta enviar el dato correspondiente por bluetooth
si el dato es diferente al de la llamada anteror
        if (btSocket!=null && dato_ant!=dato)
        {
```

```
            try
            {
                btSocket.getOutputStream().write(dato);

            }
            catch (IOException e) //Si hay un error se muestra un
mensaje
            {
                msg("Error");
            }
        }

        dato_ant=dato; //Se guarda el valor del dato de la llamada
actual para compararlo en la siguiente llamada
    }

    //función para determinar la acción a realizar al pulsar los
botones propios del móvil o tablet
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        // TODO Auto-generated method stub
        if (keyCode == event.KEYCODE_BACK) { //si se pulsa el
botón de Atrás

            if (btSocket != null) { //Si hay conexión bluetooth
                try {
                    btSocket.getOutputStream().write((byte) 0);
//Se envía un 0 indicando que va a finalizar la comunicación
                }
                catch (IOException e) { //Mensaje de error si no
se puede enviar el comando
                    msg("Error");
                }

                try { //Se cierra el socket de comunicación
                    btSocket.close();
                }
                catch (IOException e) { //Mensaje de error si no
se puede cerrar
                    msg("Error");
                }
            }
            finish(); //Finaliza la actividad volviendo a la de
elección de dispositivo
        }
        return false;
    }
}
```

ANEXO 4: PLANOS

A continuación se incluyen los planos correspondientes al esquema electrónico, la fabricación de la PCB y las piezas diseñadas para las construcciones mecánicas.



UNIVERSIDAD DE VALLADOLID
ESCUELA DE INGENIERIAS INDUSTRIALES
TFG ALMACÉN AUTOMATIZADO

Diseño de: Alfredo Alonso Hermosilla

Tutor: Eduardo Zalama Casanova Dpto. ISA

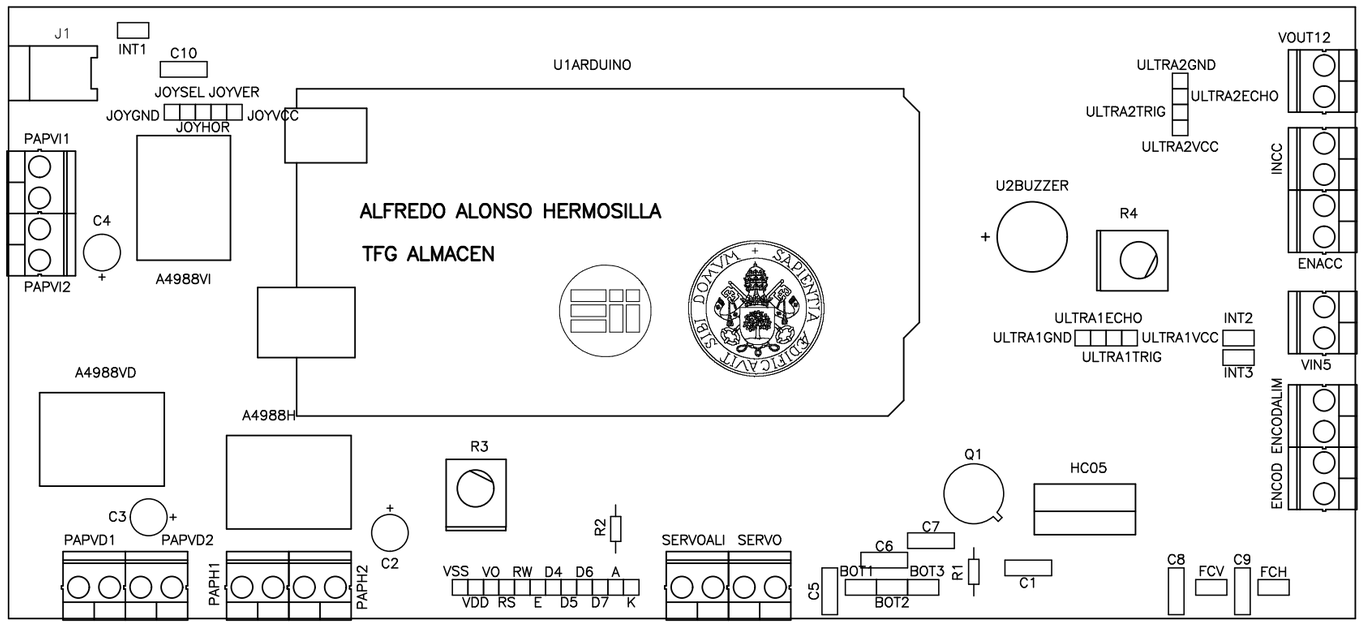
ESQUEMA ELECTRÓNICO

29/04/2017 A3

Plano 1 de 33

NOTAS PARA MONTAJE

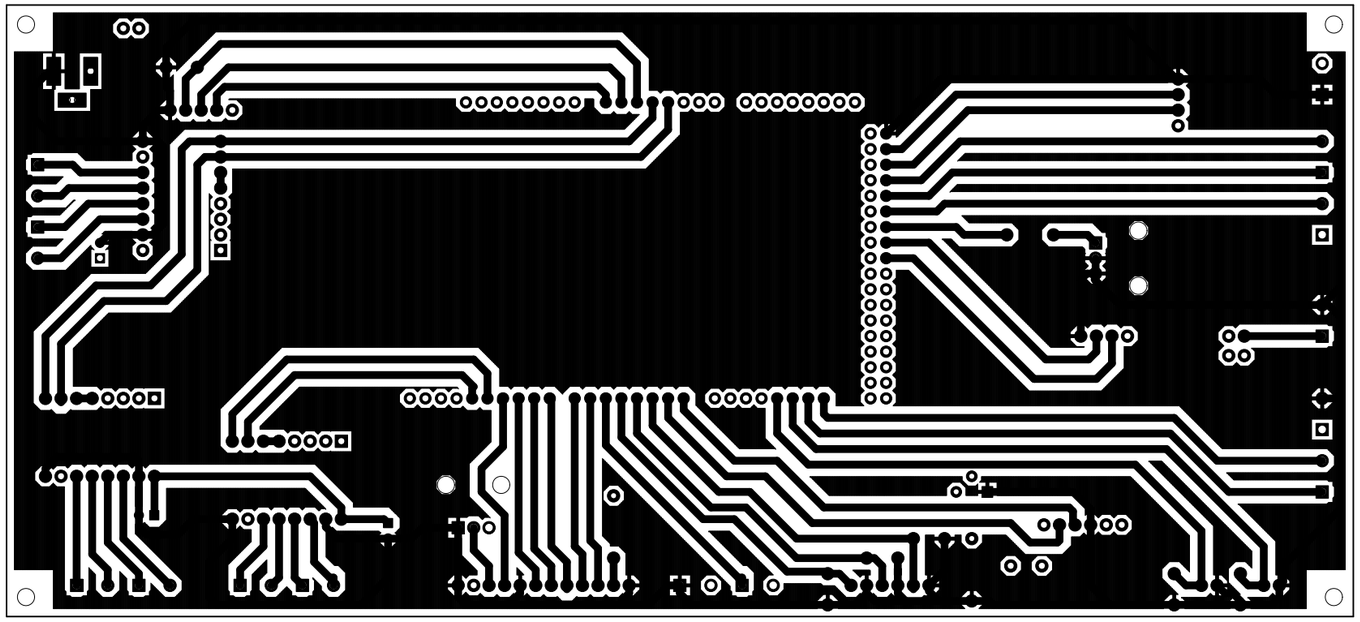
1. TODOS LOS COMPONENTES SE MONTARAN A RAS DE LA PCB EN LA CARA DE COMPONENTES
2. TODAS LAS PATILLAS SE CORTARAN COMO MAXIMO A 1.7mm (45mil) DE ALTURA
3. LA PCB DEBERA ESTAR LIBRE DE RESTOS DE SOLDADURA Y FLUX
4. LOS SIGUIENTES COMPONENTES SE MONTARAN EN PINES MACHO: U1ARDUINO
5. LOS SIGUIENTES COMPONENTES SE MONTARAN EN PINES HEMBRA: HC05, A4988VI, A4988VD Y A4988H
6. SE INSERTARAN TORNILLOS AUTORROSCANTES DE 3mm EN LOS TALADROS DE LAS ESQUINAS PARA ATORNILLAR A LA BASE



Diseno de ALFREDO ALONSO HERMOSILLA	Grado Electronica Industrial y Automatica	Fecha 10/05/2017	Escala 1:1
TFG Almacen Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA		
	PLACA DE CIRCUITO IMPRESO PLANO DE SERIGRAFIA		Plano: 2 de 33

NOTAS PARA MONTAJE

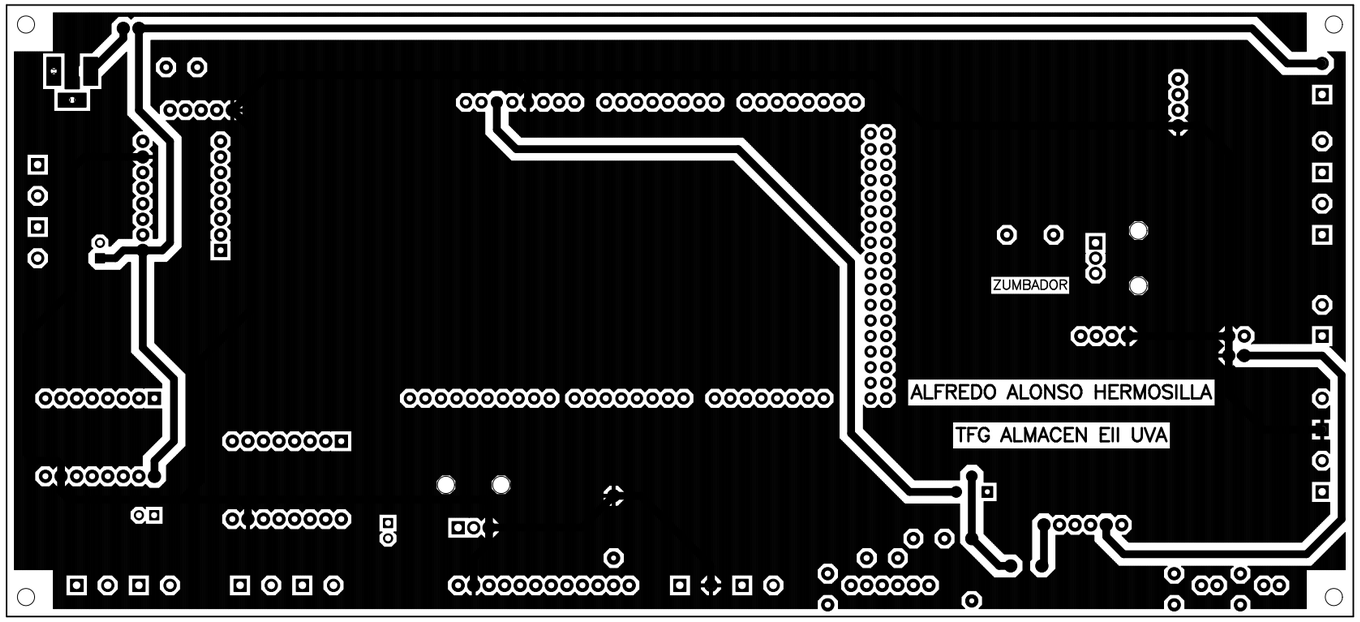
1. TODOS LOS COMPONENTES SE MONTARAN A RAS DE LA PCB EN LA CARA DE COMPONENTES
2. TODAS LAS PATILLAS SE CORTARAN COMO MAXIMO A 1.7mm (45mil) DE ALTURA
3. LA PCB DEBERA ESTAR LIBRE DE RESTOS DE SOLDADURA Y FLUX
4. LOS SIGUIENTES COMPONENTES SE MONTARAN EN PINES MACHO: U1ARDUINO
5. LOS SIGUIENTES COMPONENTES SE MONTARAN EN PINES HEMBRA: HC05, A4988VI, A4988VD Y A4988H
6. SE INSERTARAN TORNILLOS AUTORROSCANTES DE 3mm EN LOS TALADROS DE LAS ESQUINAS PARA ATORNILLAR A LA BASE



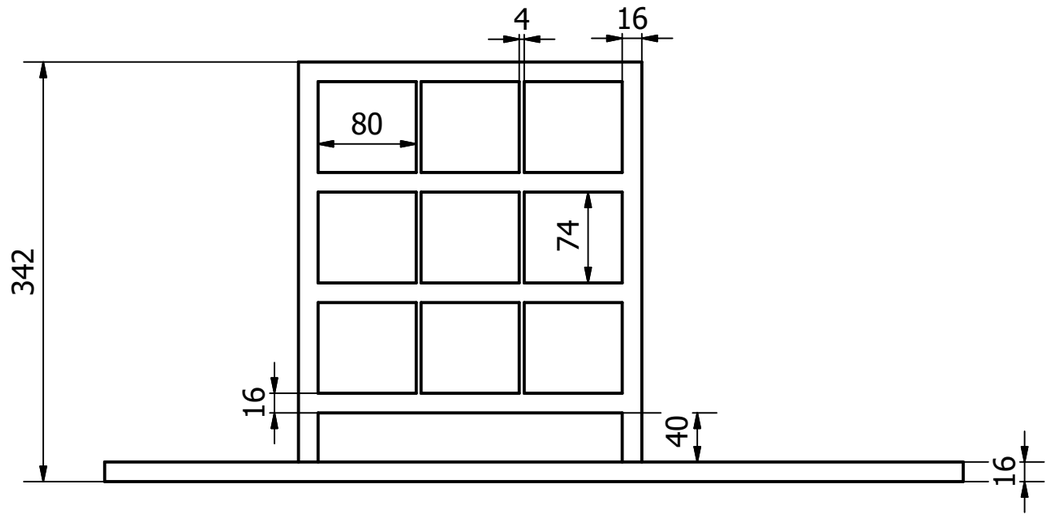
Diseno de ALFREDO ALONSO HERMOSILLA	Grado Electronica Industrial y Automatica	Fecha 10/05/2017	Escala 1:1
TFG Almacen Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA		
	PLACA DE CIRCUITO IMPRESO CARA DE SOLDADURA	Plano: 3 de 33	

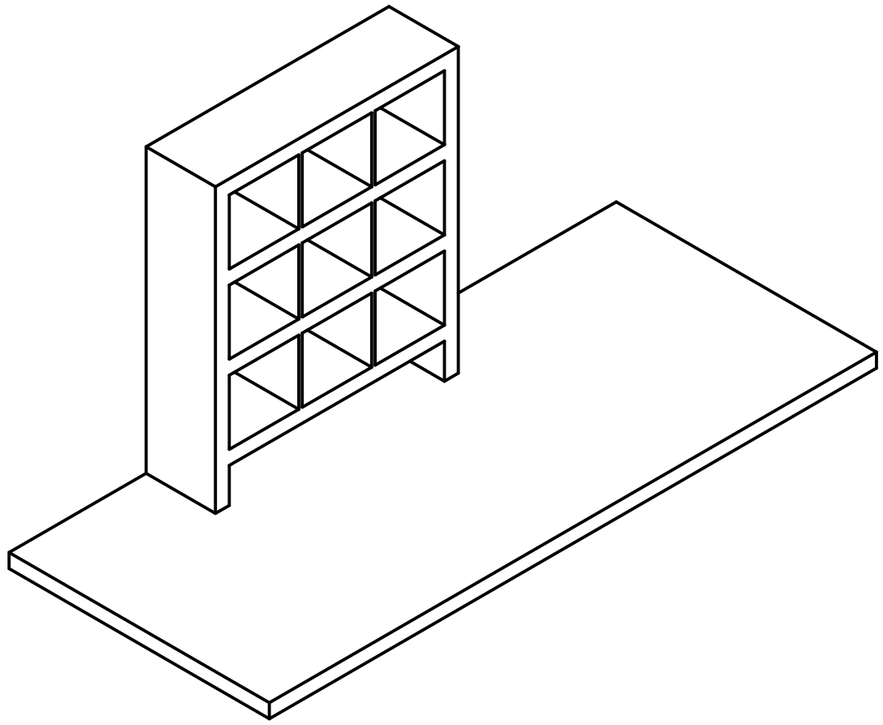
NOTAS PARA MONTAJE

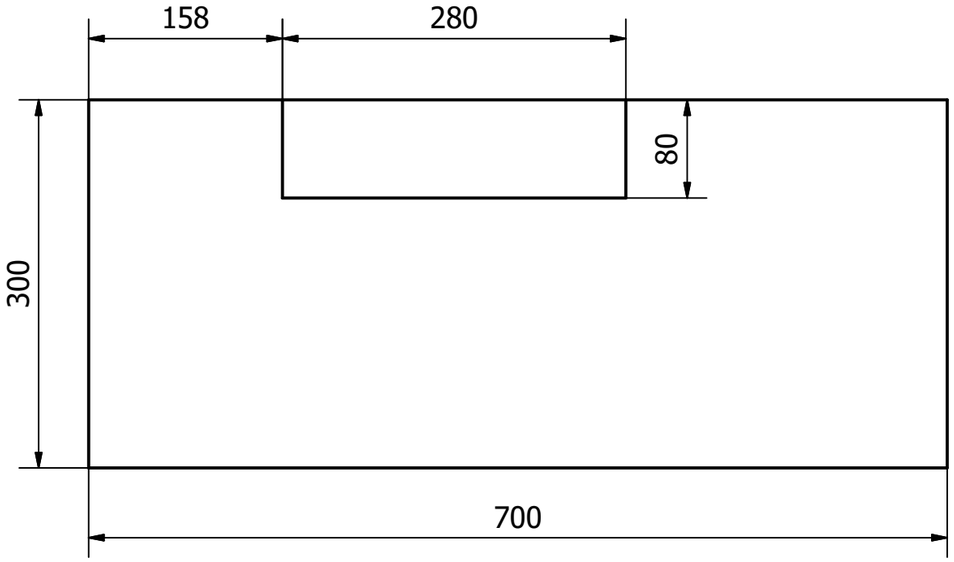
1. TODOS LOS COMPONENTES SE MONTARAN A RAS DE LA PCB EN LA CARA DE COMPONENTES
2. TODAS LAS PATILLAS SE CORTARAN COMO MAXIMO A 1.7mm (45mil) DE ALTURA
3. LA PCB DEBERA ESTAR LIBRE DE RESTOS DE SOLDADURA Y FLUX
4. LOS SIGUIENTES COMPONENTES SE MONTARAN EN PINES MACHO: U1ARDUINO
5. LOS SIGUIENTES COMPONENTES SE MONTARAN EN PINES HEMBRA: HC05, A4988VI, A4988VD Y A4988H
6. SE INSERTARAN TORNILLOS AUTORROSCANTES DE 3mm EN LOS TALADROS DE LAS ESQUINAS PARA ATORNILLAR A LA BASE



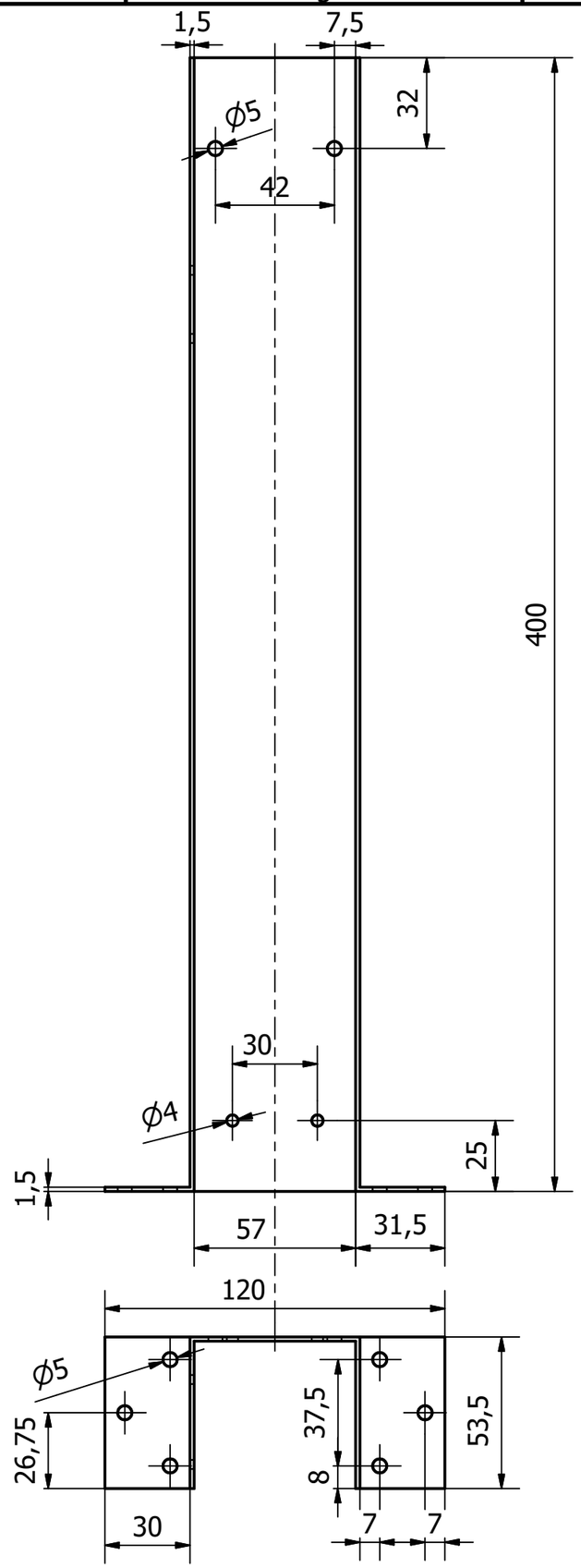
Diseno de ALFREDO ALONSO HERMOSILLA	Grado Electronica Industrial y Automatica	Fecha 10/05/2017	Escala 1:1
TFG Almacen Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA		
	PLACA DE CIRCUITO IMPRESO CARA DE COMPONENTES		Plano: 4 de 33

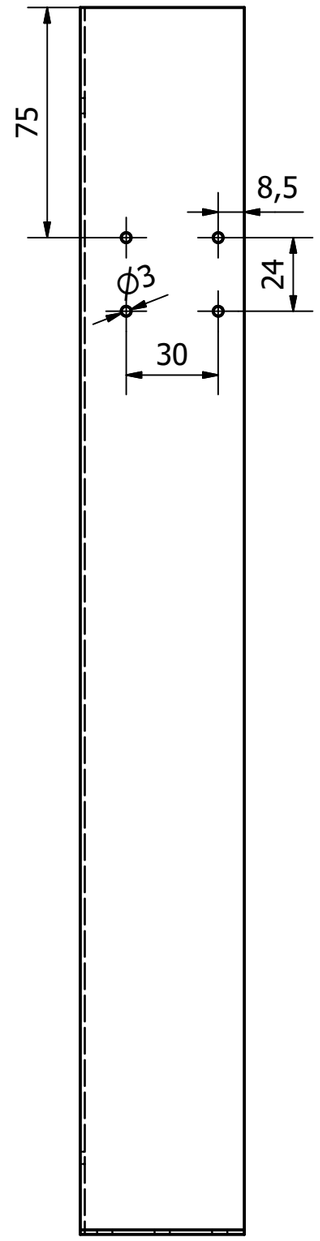


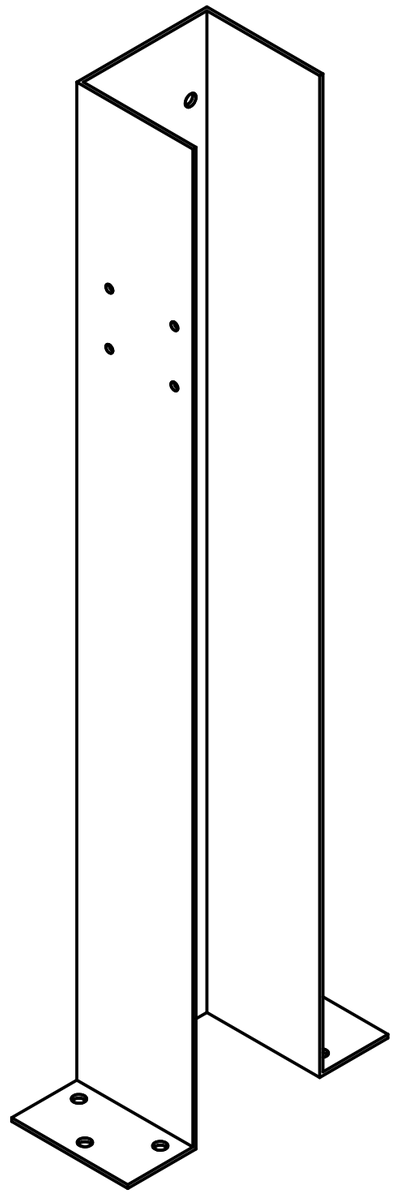




Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Base y Estantería	Plano 5 / 33

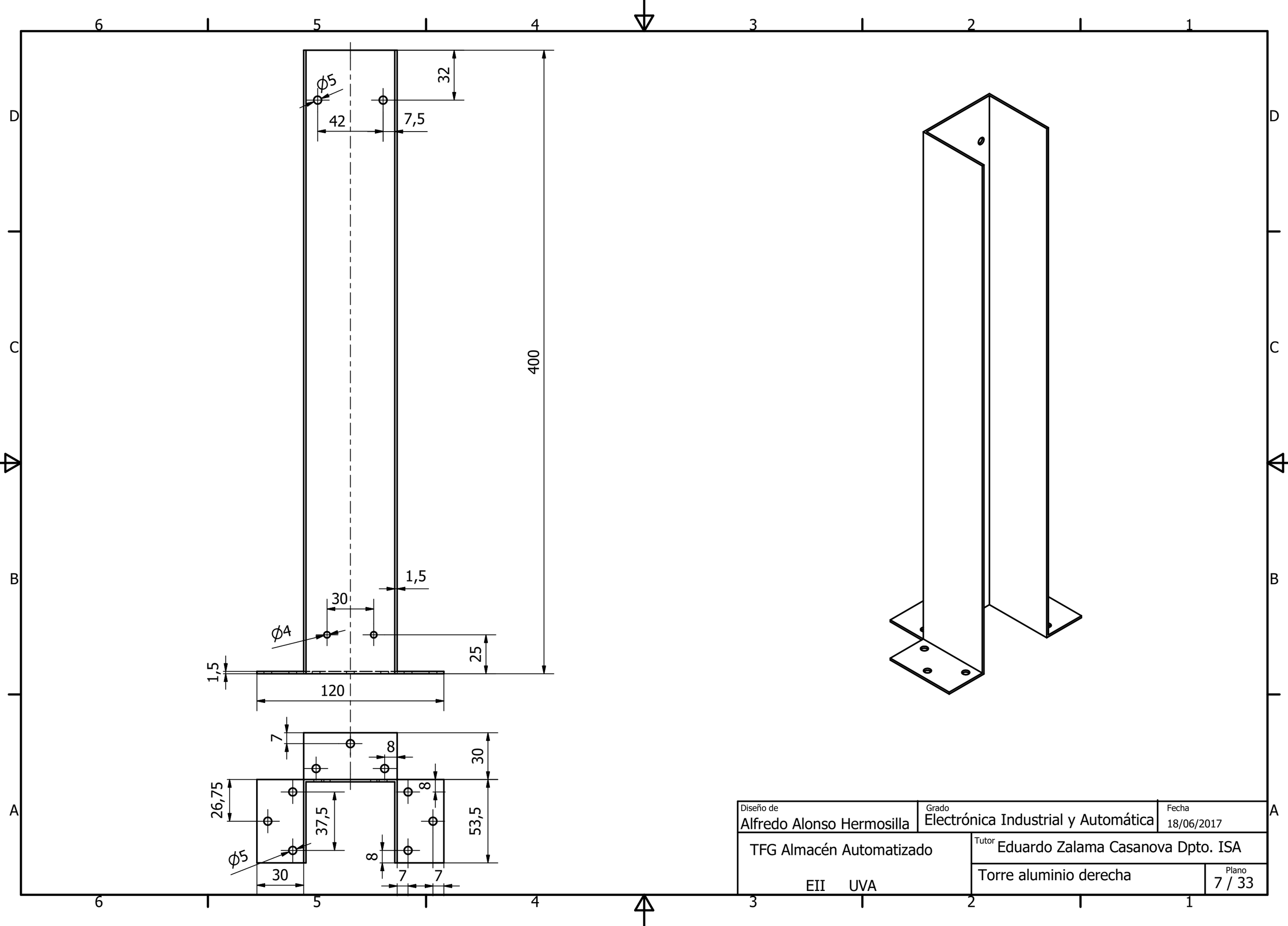




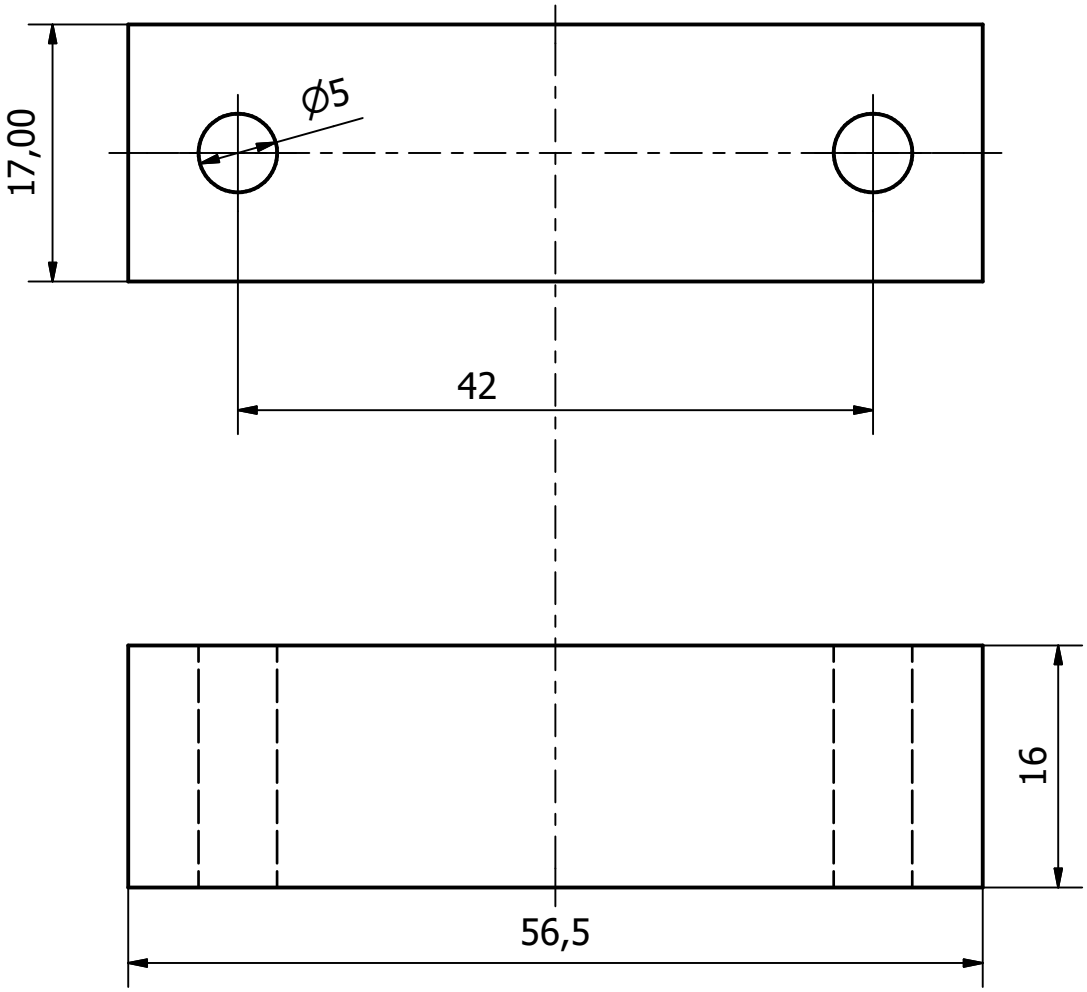


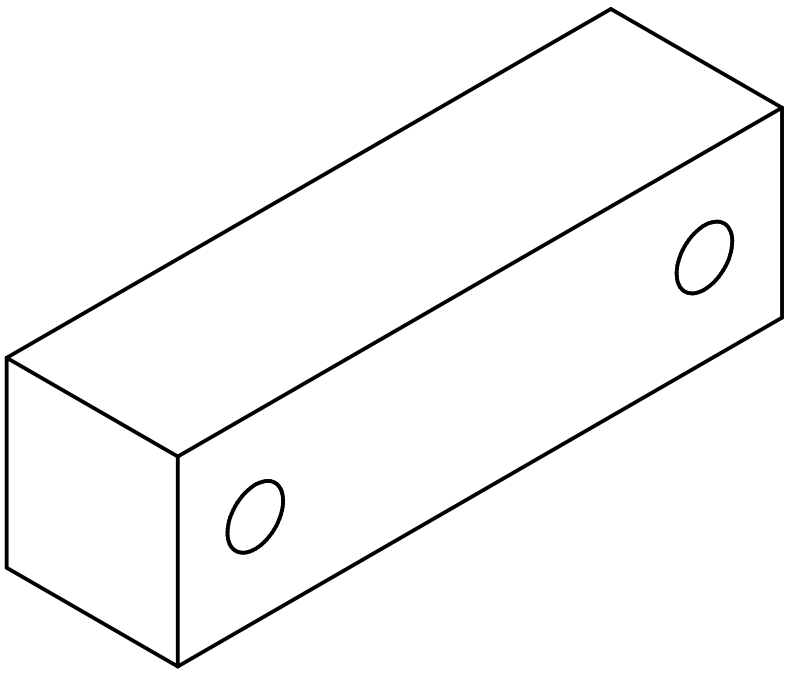
Simetría excepto por los agujeros mostrados en el perfil

Diseño de: Alfredo Alonso Hermosilla	Grado: Electrónica Industrial y Automática	Fecha: 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor: Eduardo Zalama Casanova Dpto. ISA	
	Torre aluminio izquierda	Plano: 6 / 33

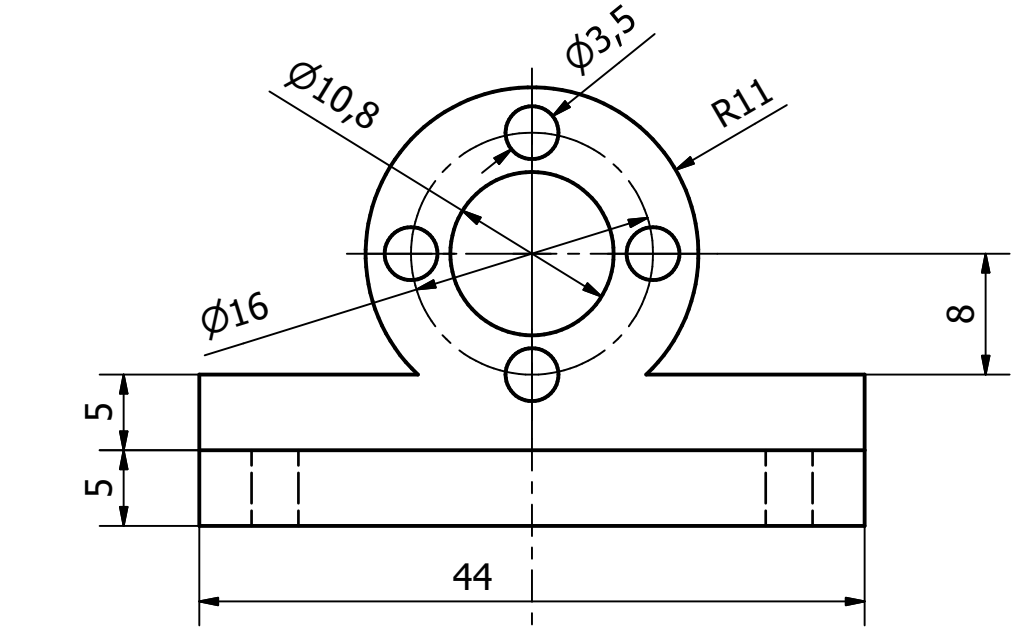


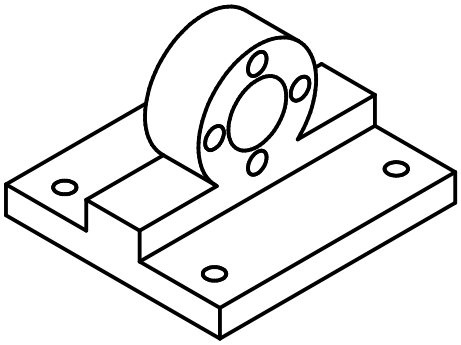
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Torre aluminio derecha	Plano 7 / 33

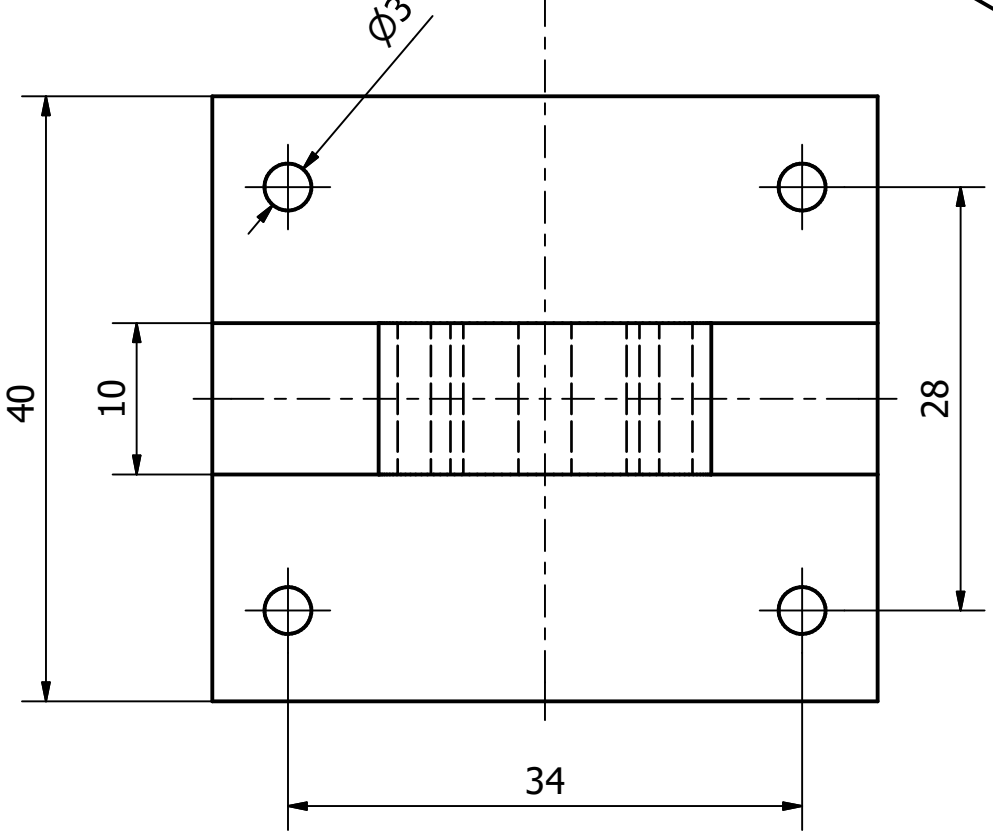




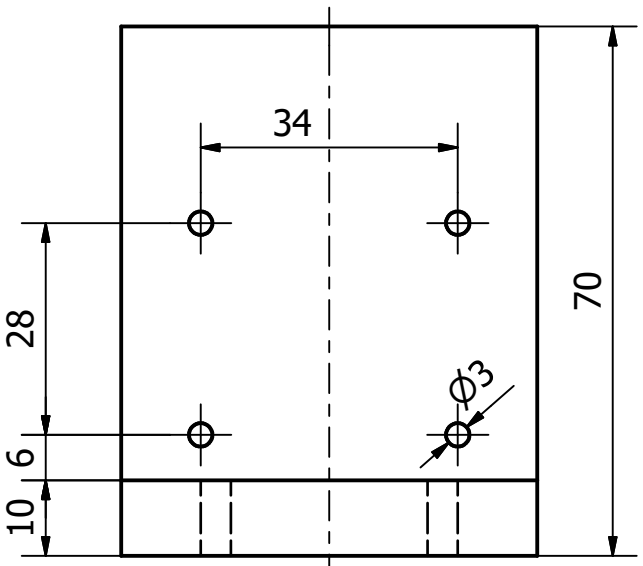
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte rodamiento vertical	Plano 8 / 33

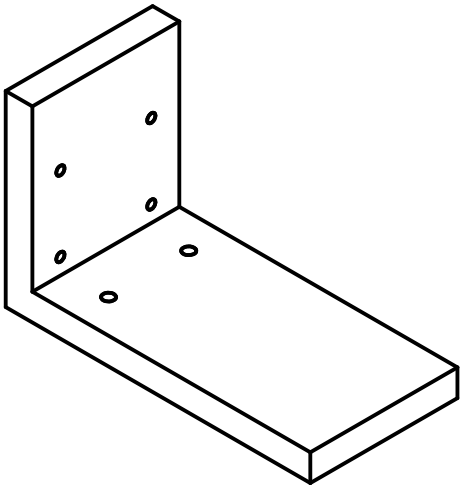


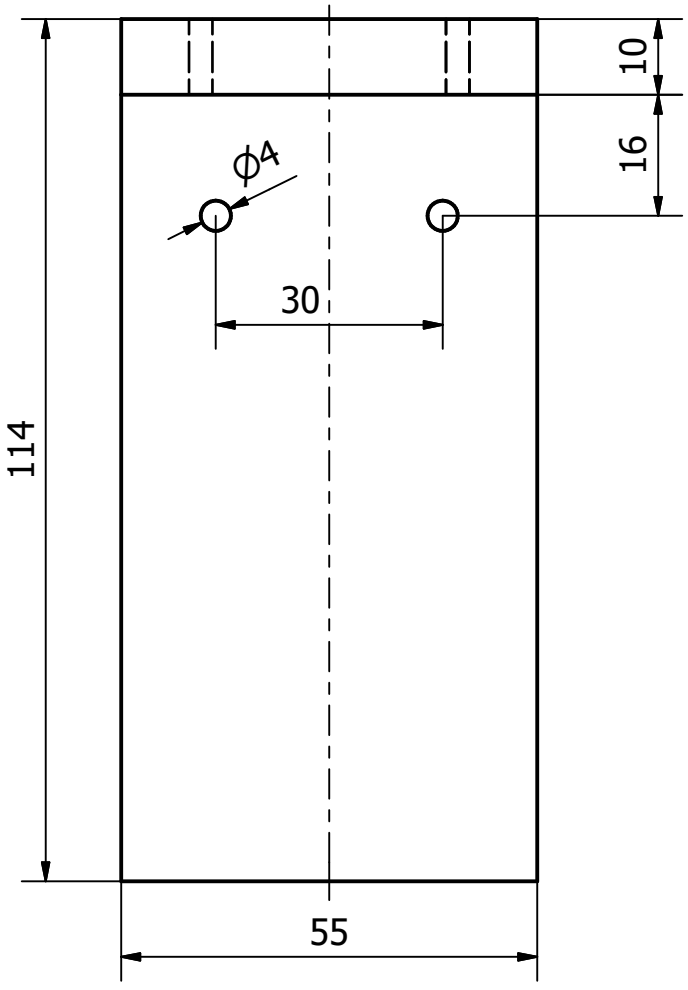




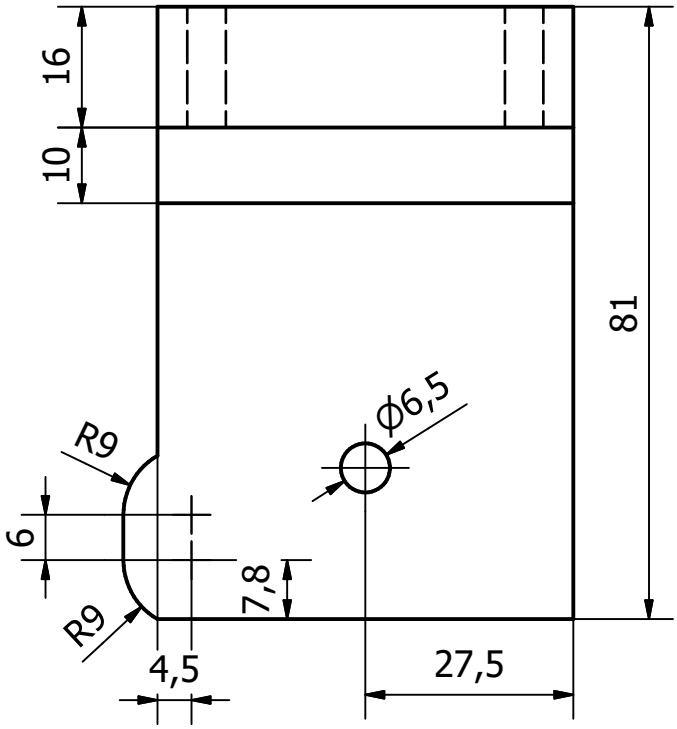
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Camisa de la tuerca	Plano 9 / 33

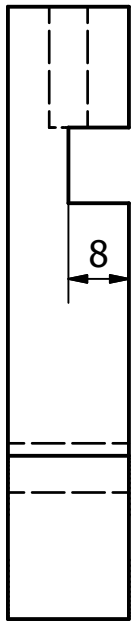


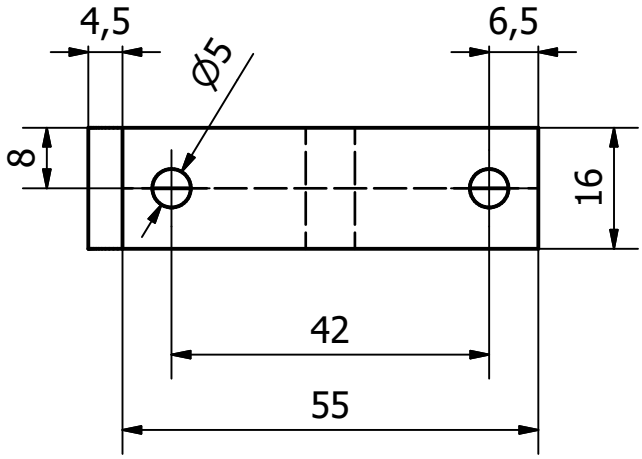


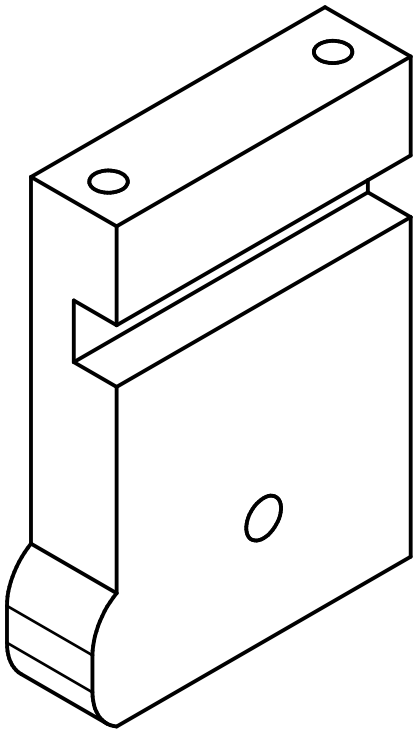


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte motor horizontal	Plano 10 / 33

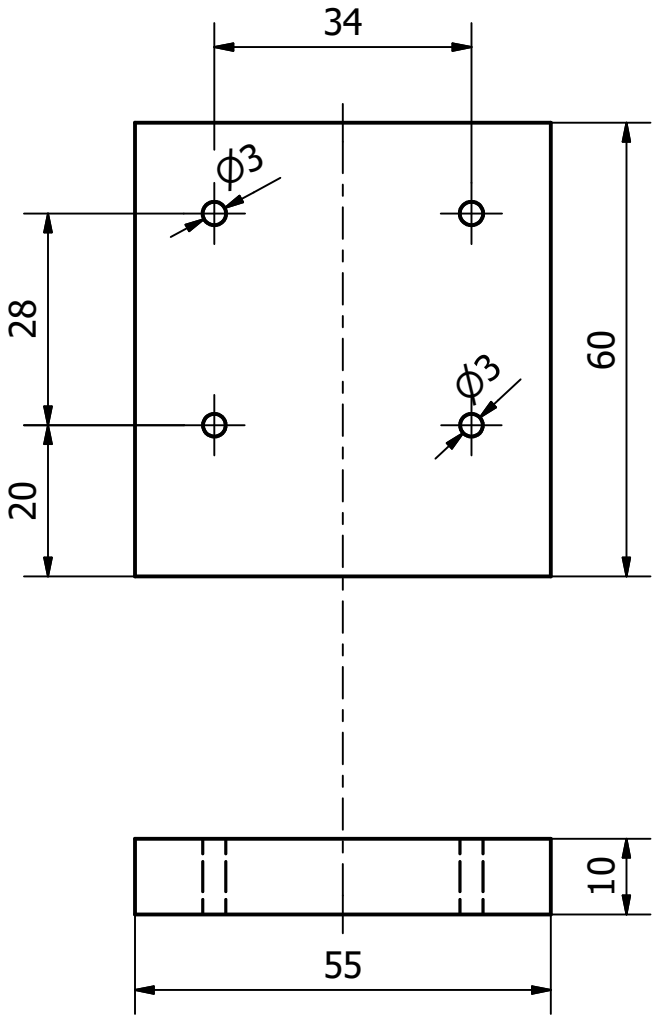


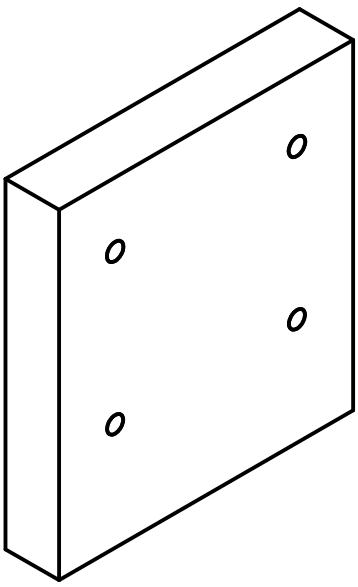




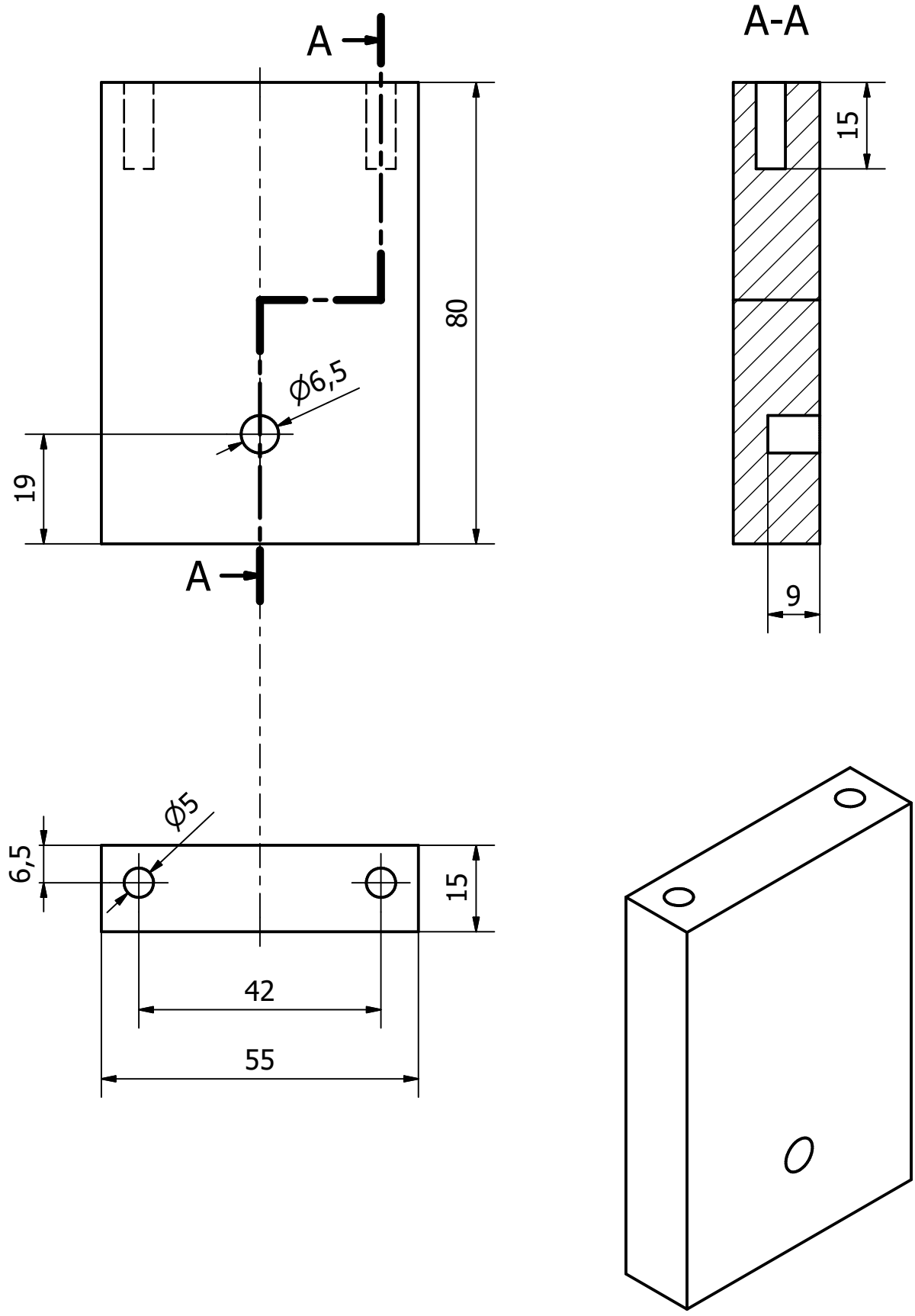


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte rodamiento izquierda	Plano 11 / 33

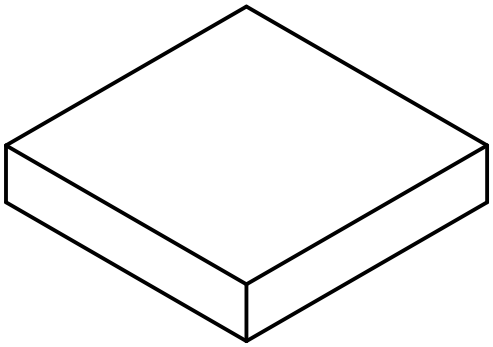


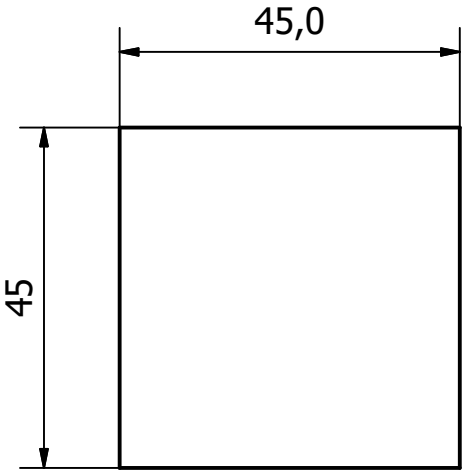


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Unión camisa-soporte rodamiento	Plano 12 / 33

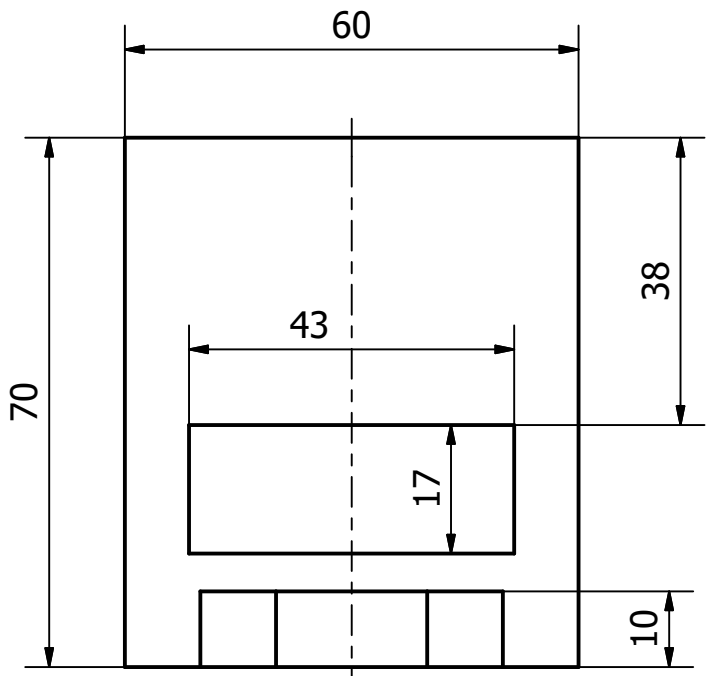


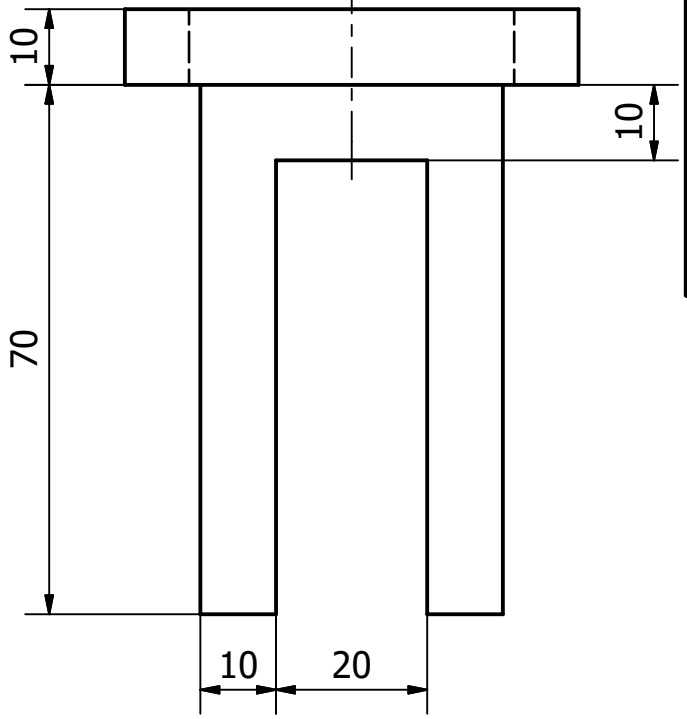
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte rodamiento derecho	Plano 13 / 33

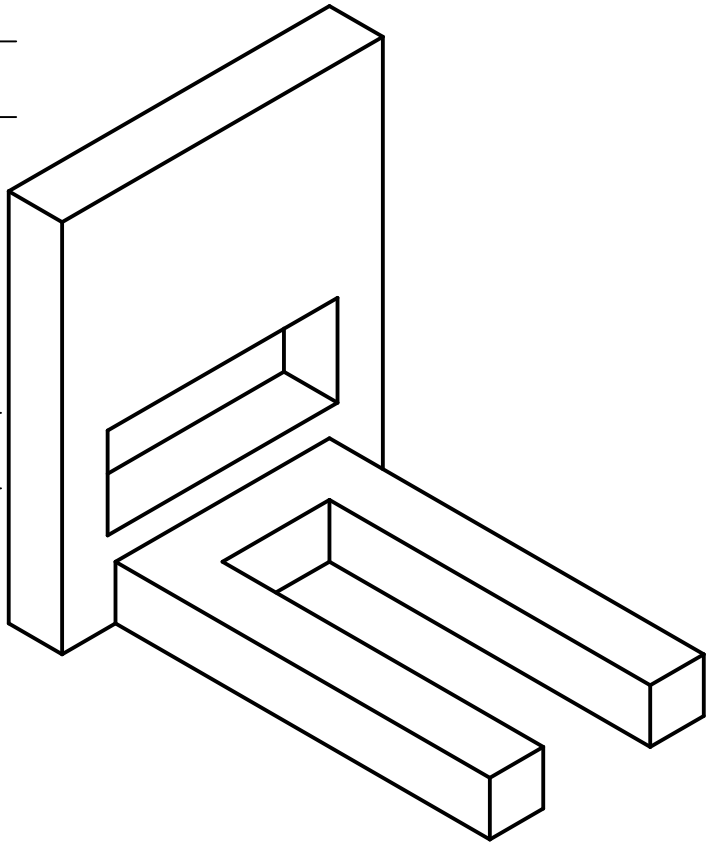




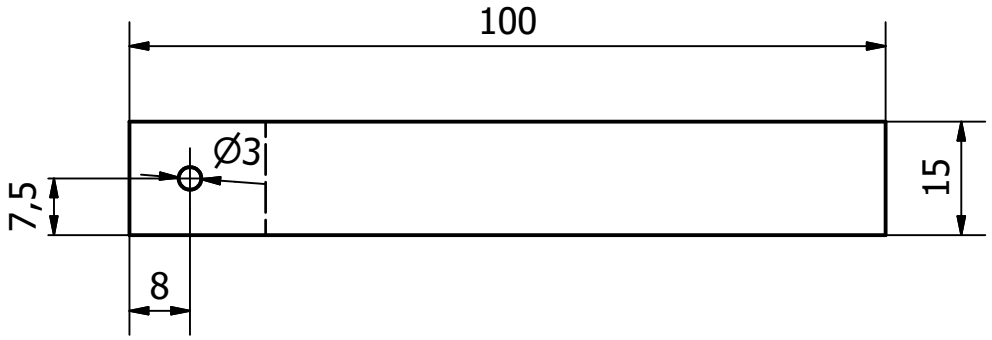
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Base motores verticales	Plano 14 / 33

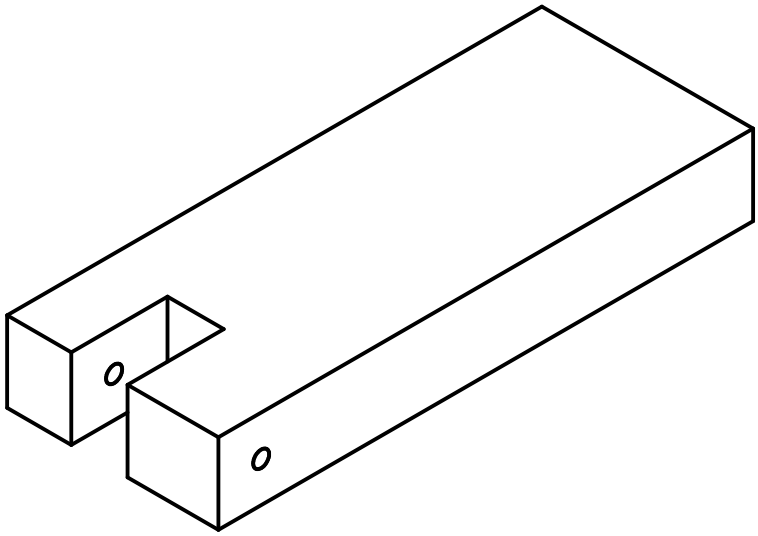






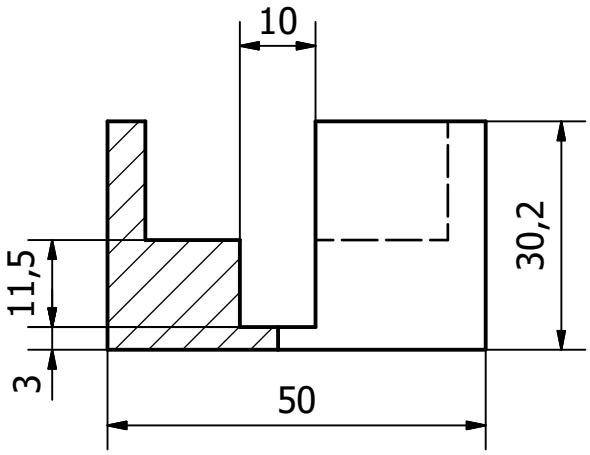
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Transpaleta	Plano 15 / 33

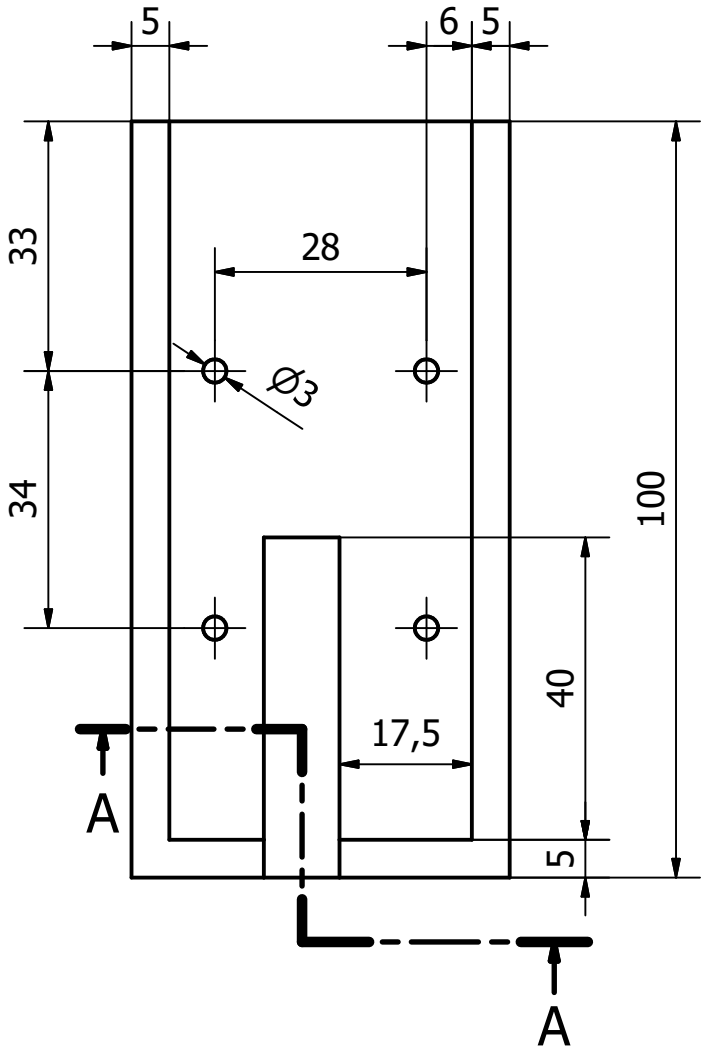


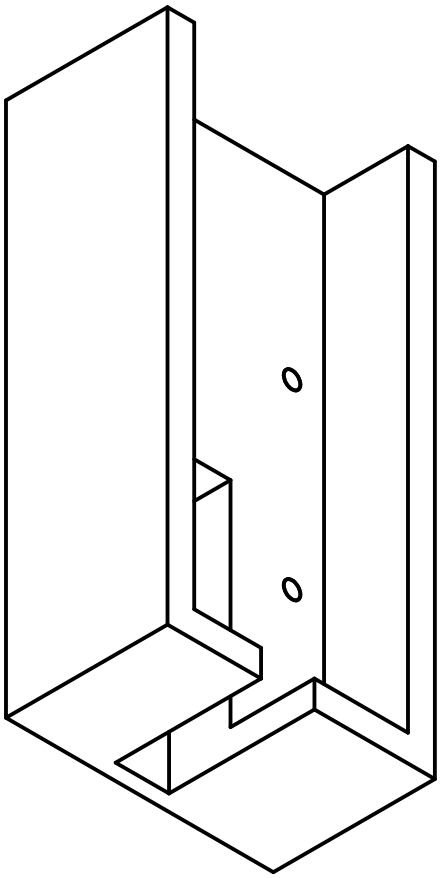


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Émbolo mecanismo biela-manivela	Plano 16 / 33

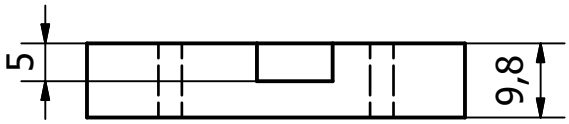
A-A

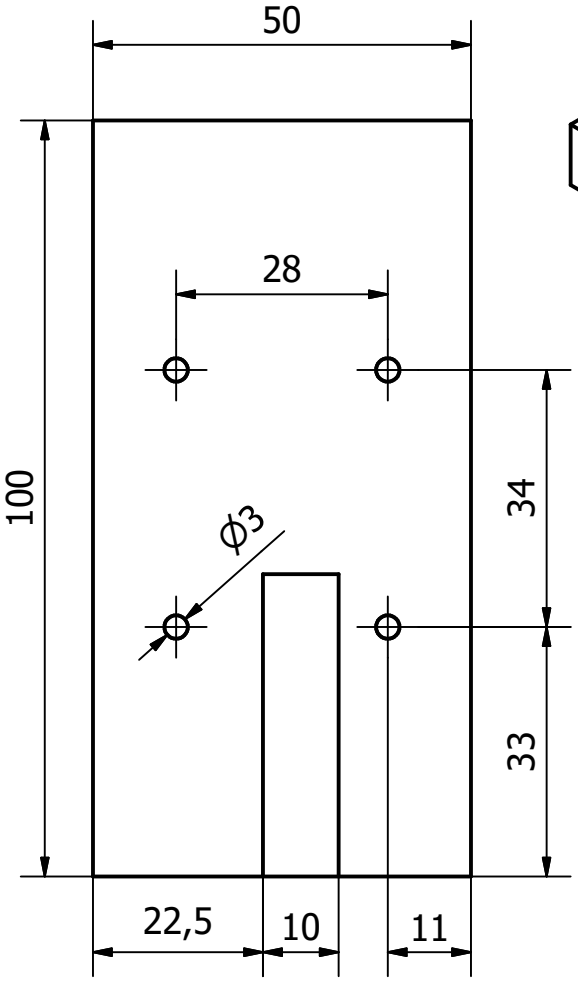


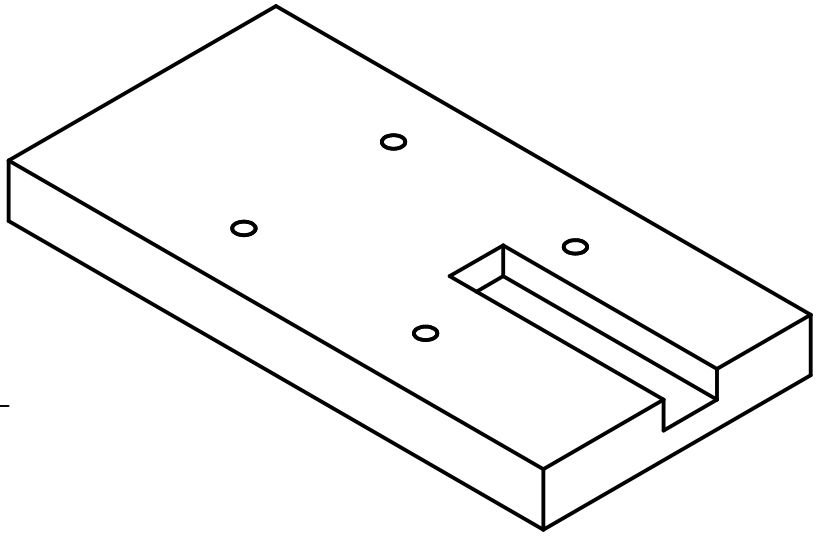




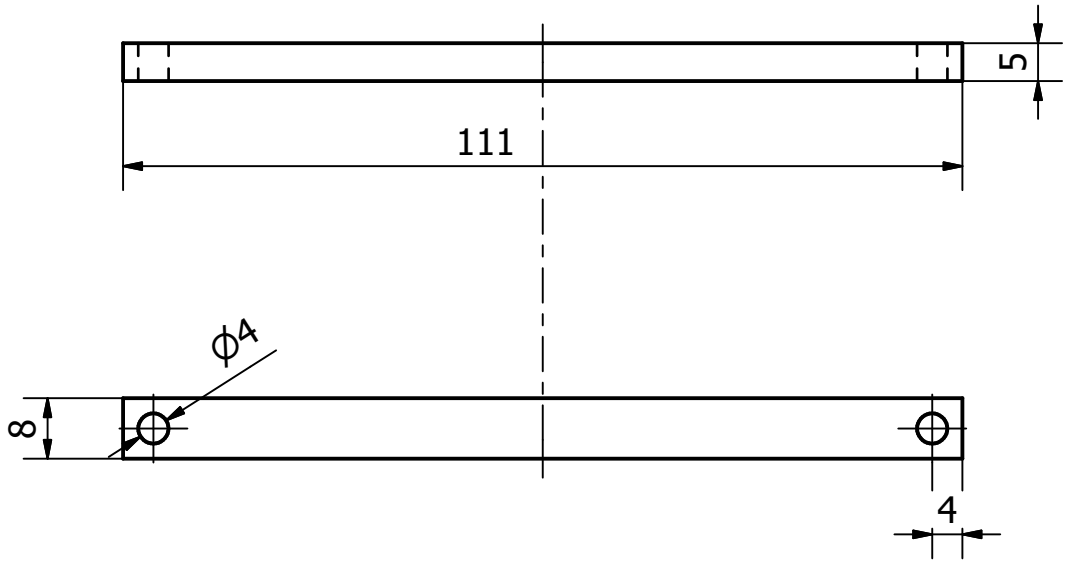
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Guía mecanismo biela-manivela	Plano 17 / 33

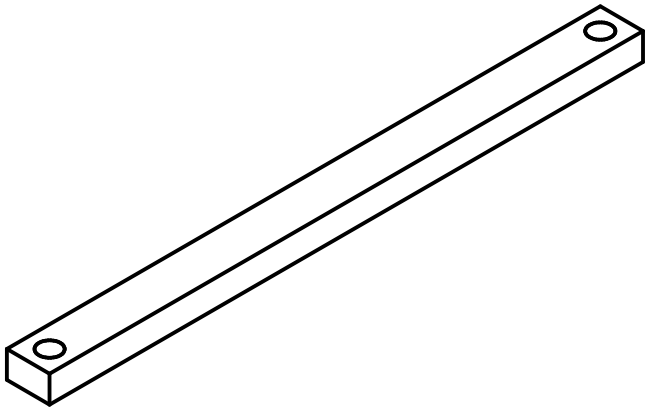




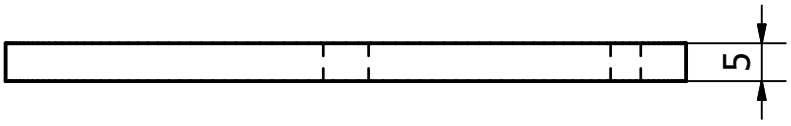


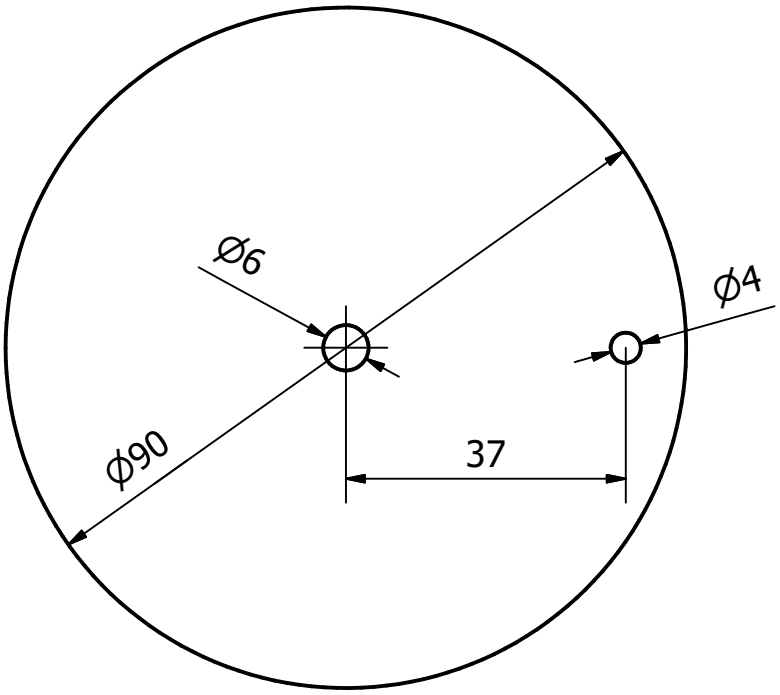
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Tapa guía biela-manivela	Plano 18 / 33

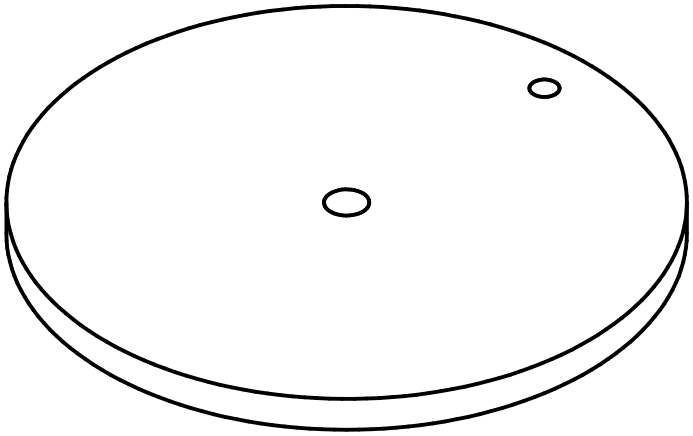




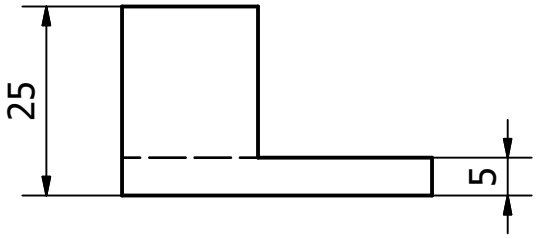
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Biela	Plano 19 / 33

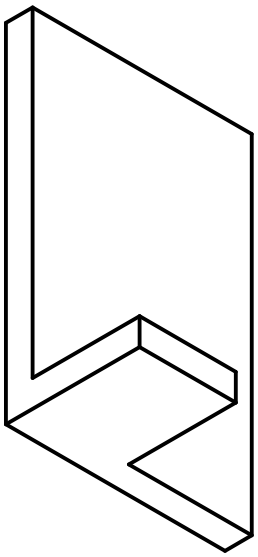


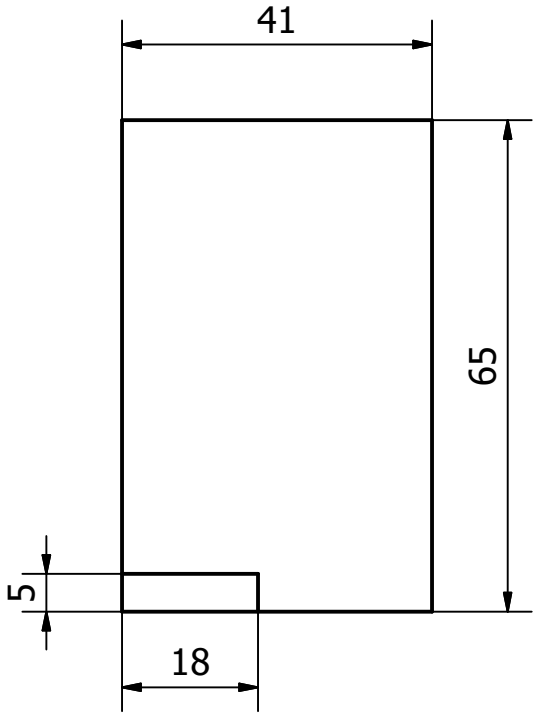




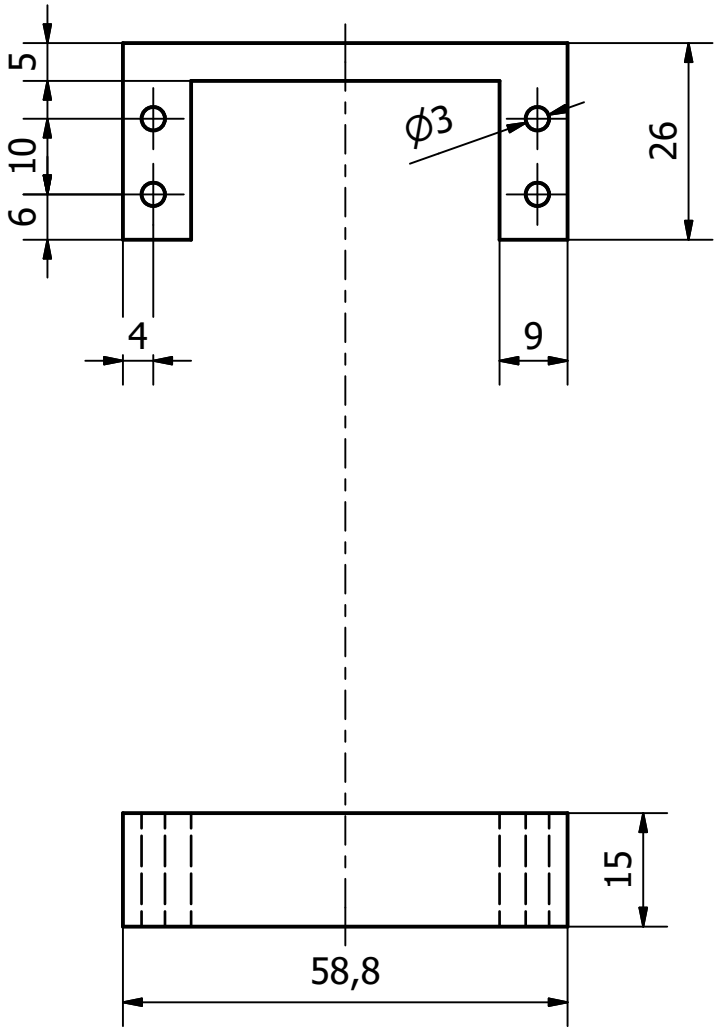
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Manivela	Plano 20 / 33

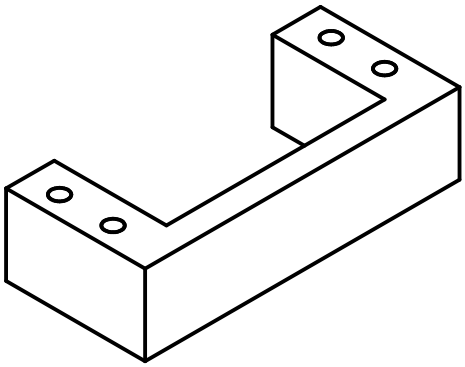




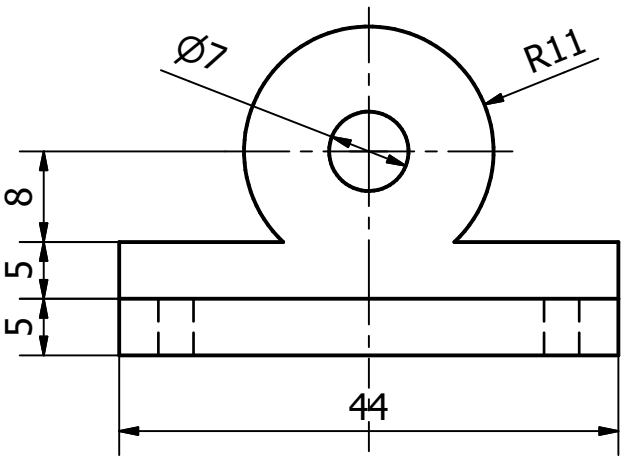


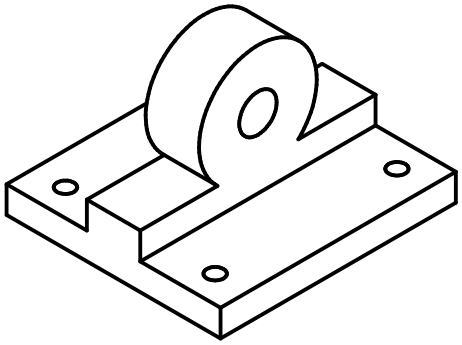
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte servomotor	Plano 21 / 33

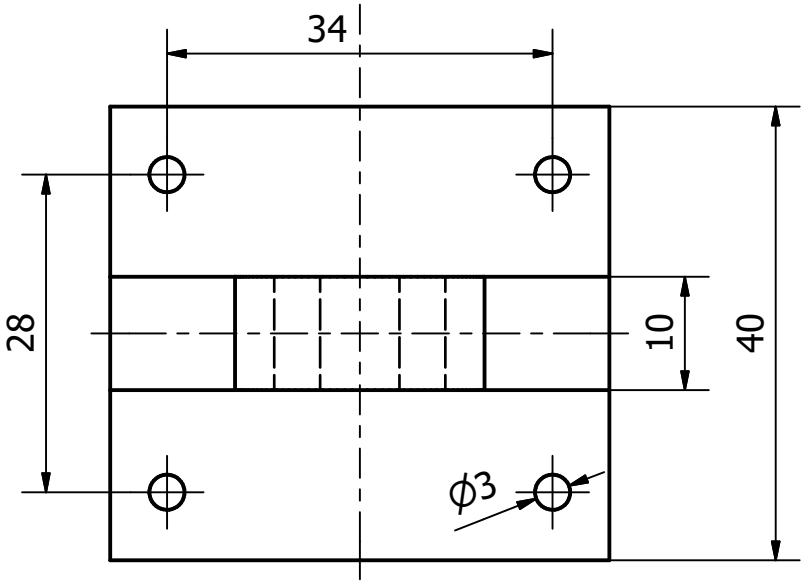




Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Sujeción servomotor	Plano 22 / 33

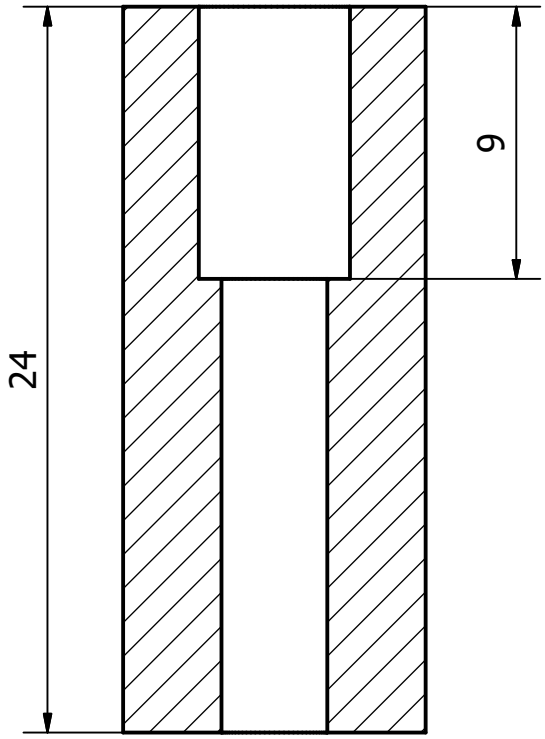


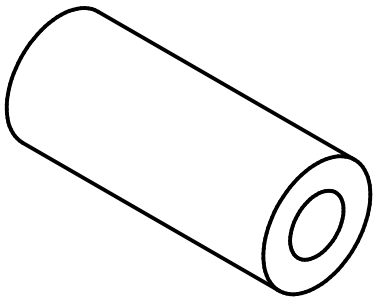


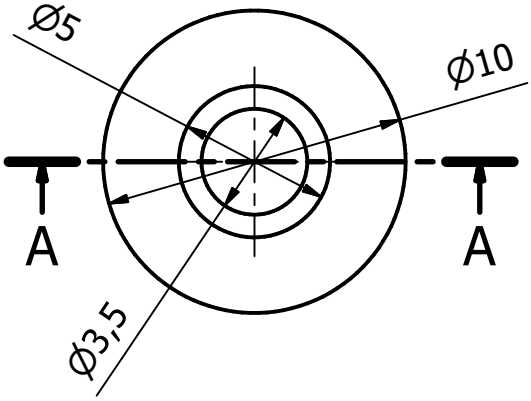


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Pieza guiada transelevador	Plano 23 / 33

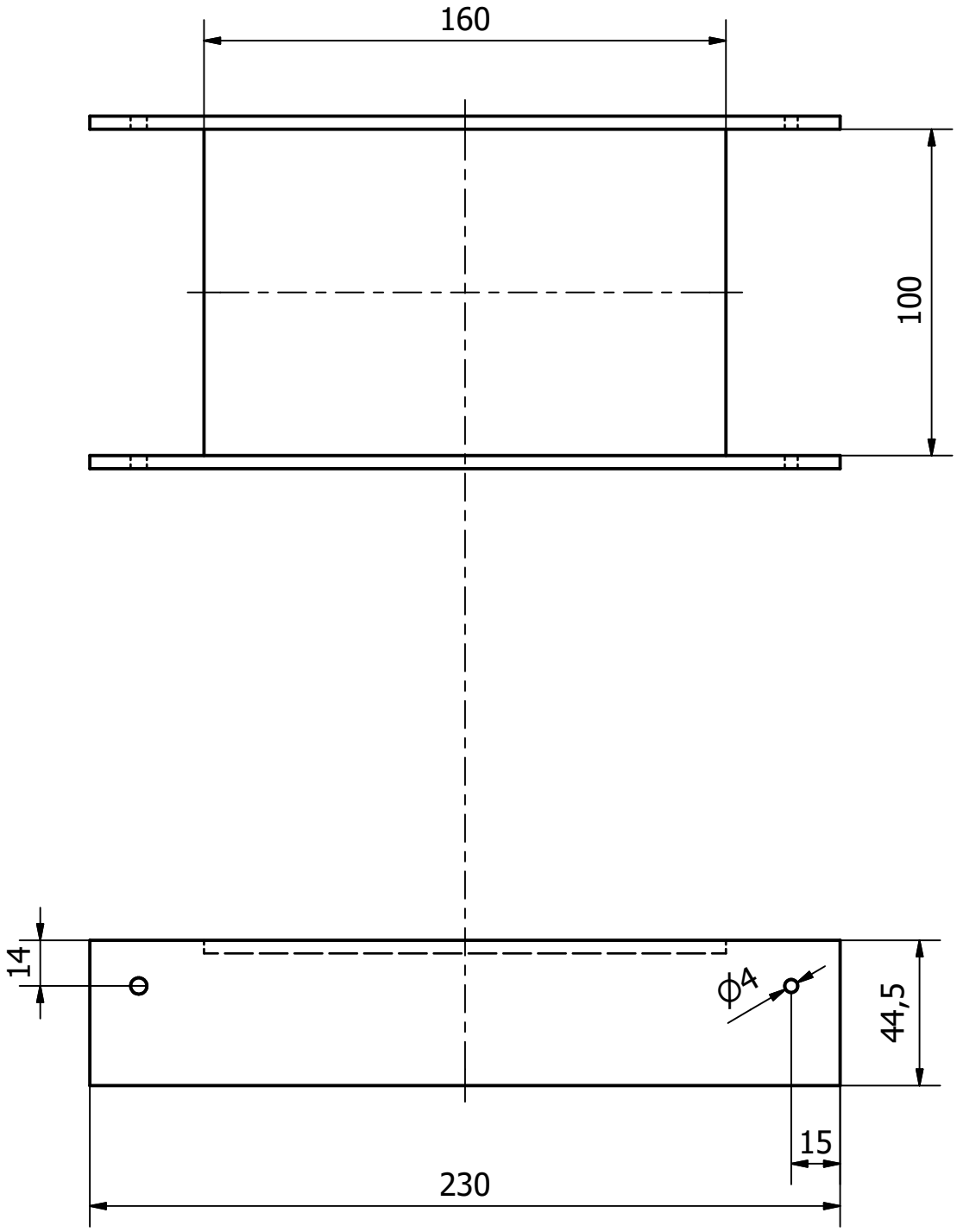
A-A



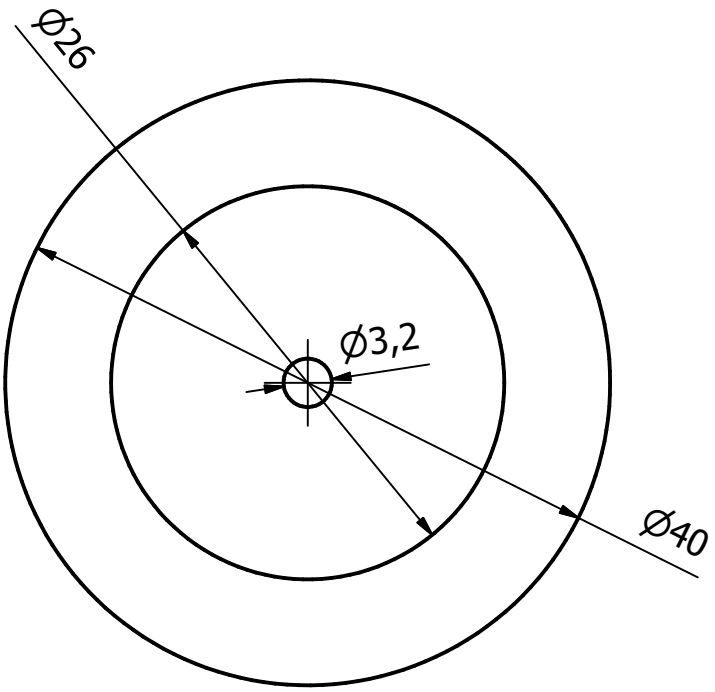


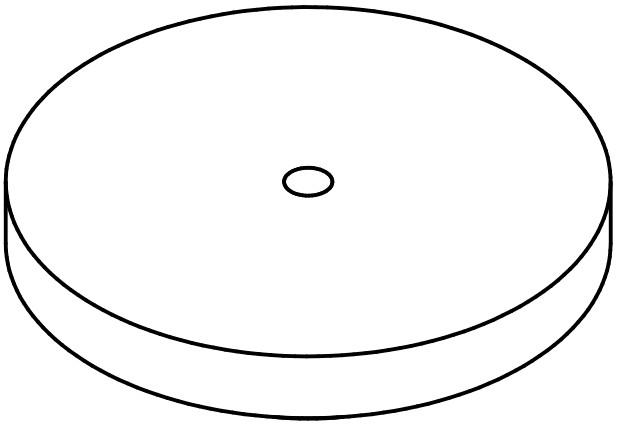


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Acople motor CC	Plano 24 / 33

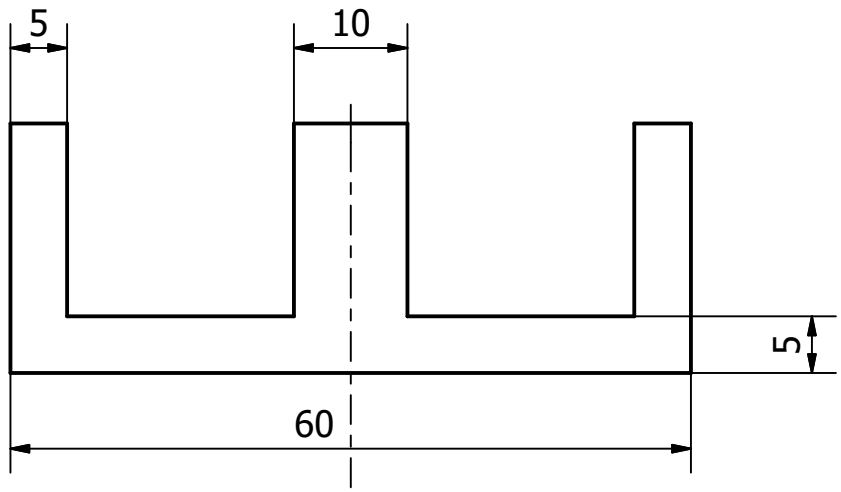


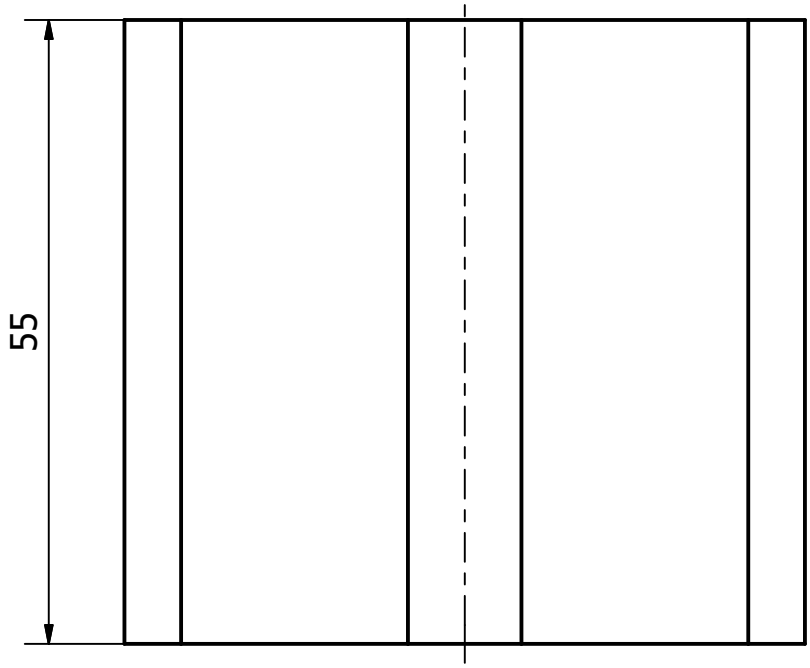
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte cinta transportadora	Plano 25 / 33

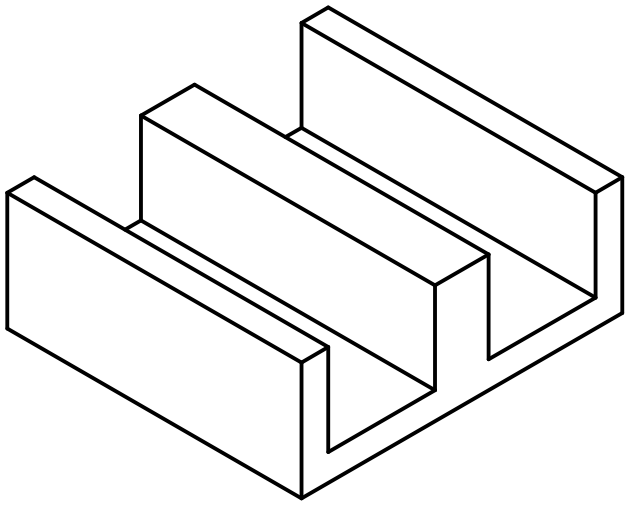




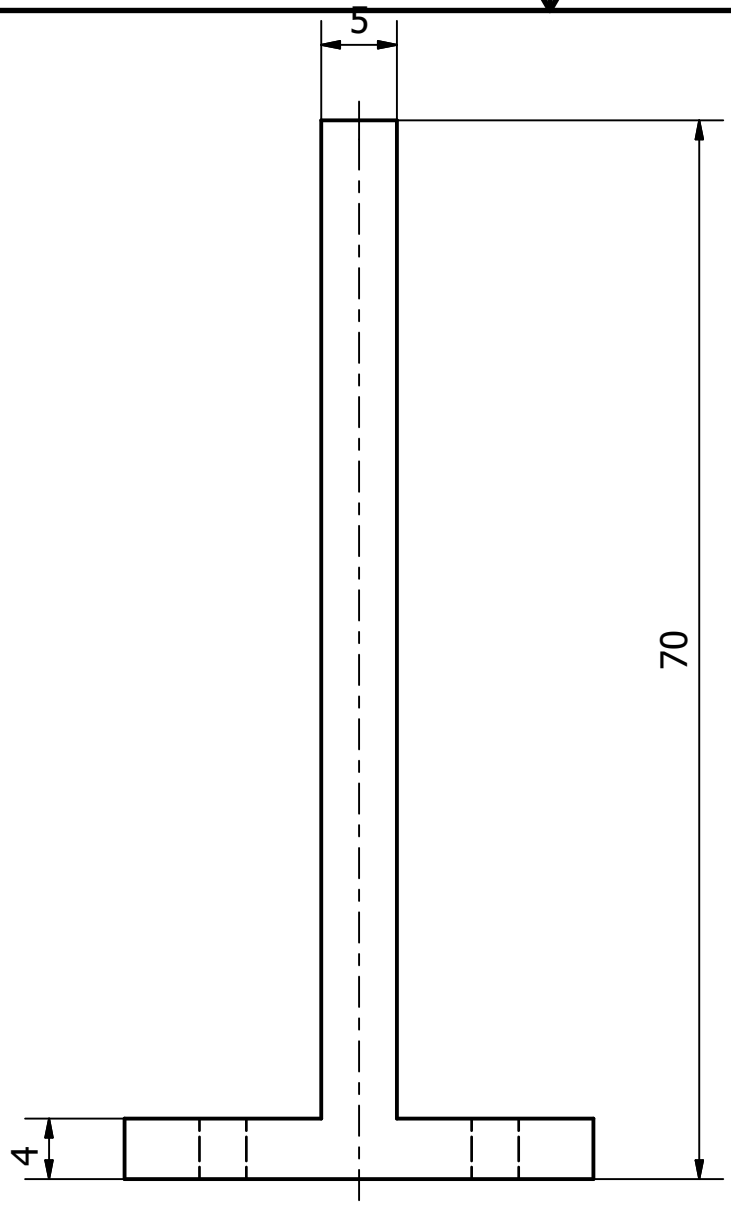
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Tapón rodillos cinta	Plano 26 / 33

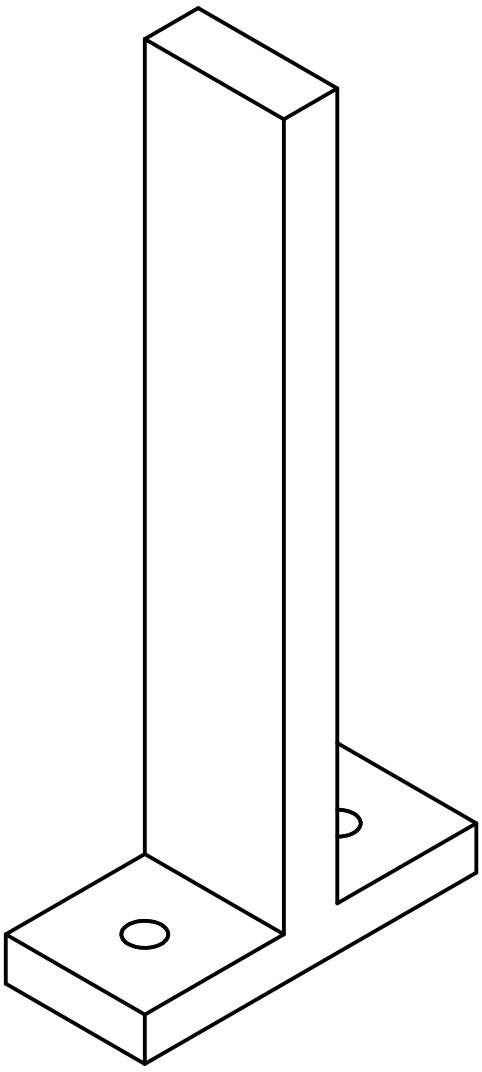




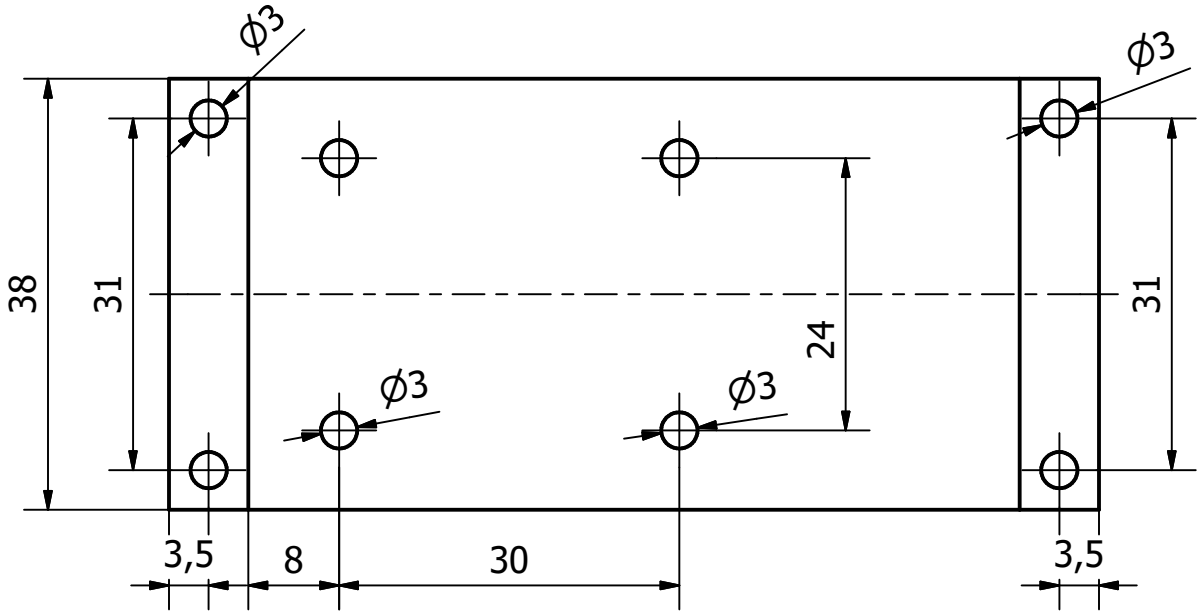


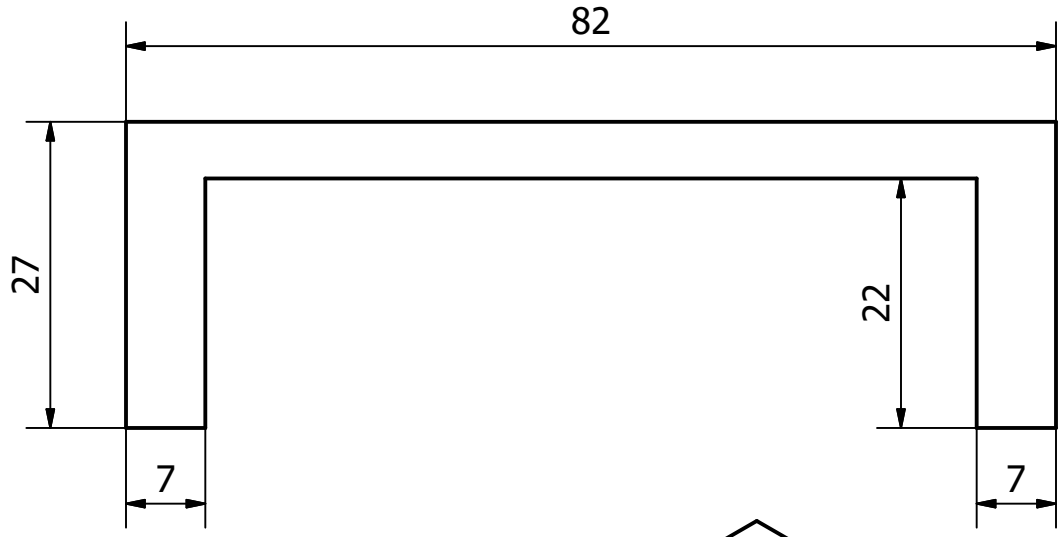
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Palé	Plano 27 / 33

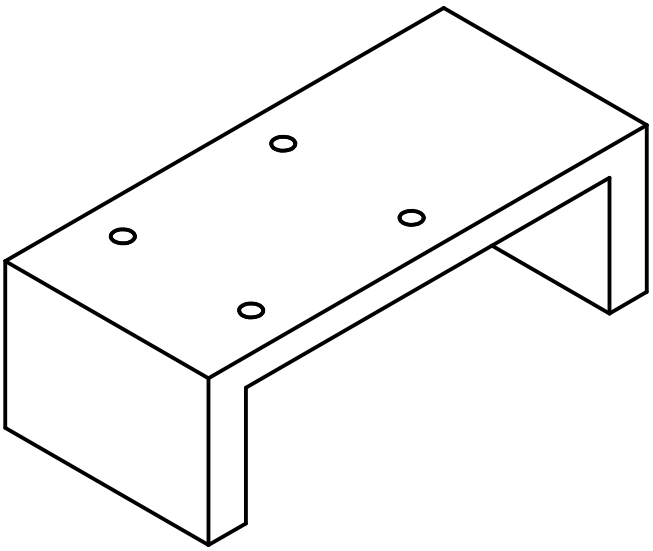




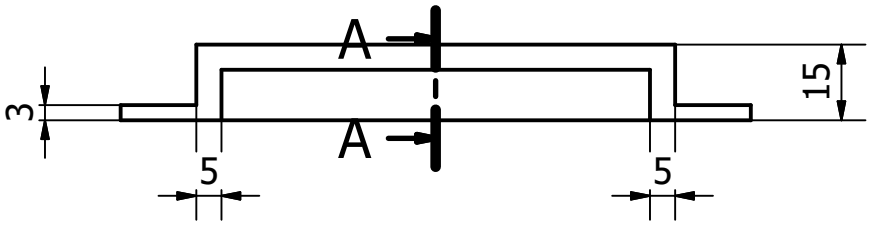
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte final de carrera vertical	Plano 28 / 33

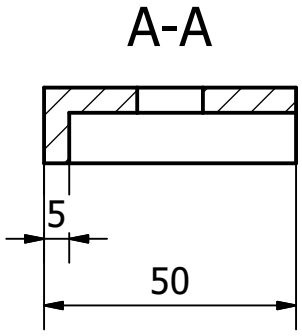


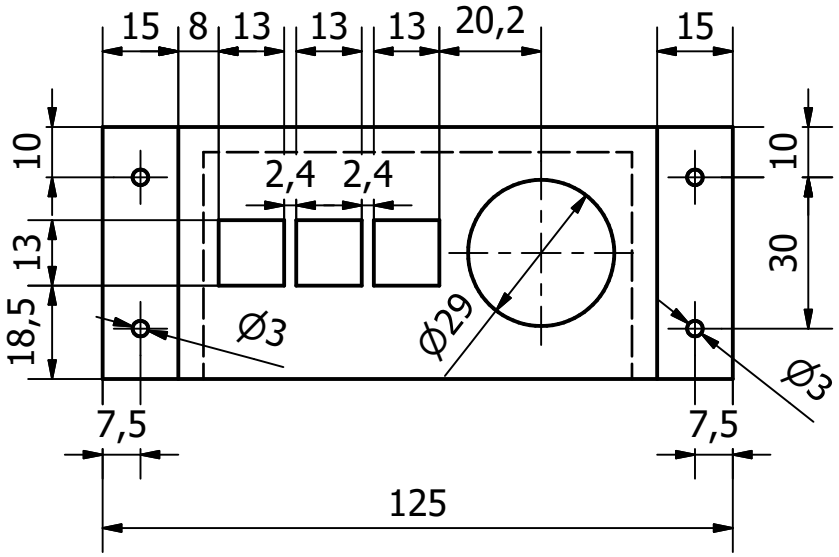


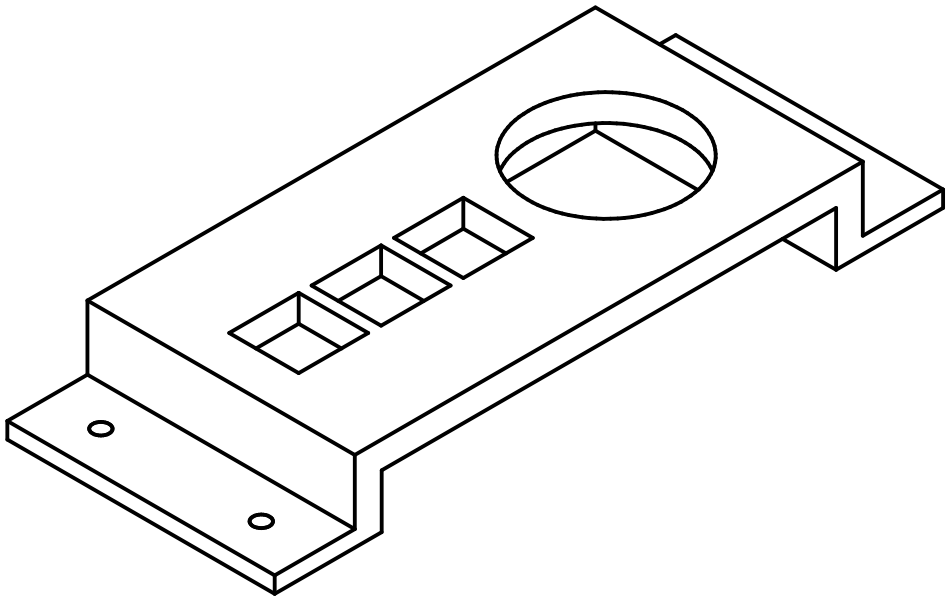


Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte LCD	Plano 29 / 33

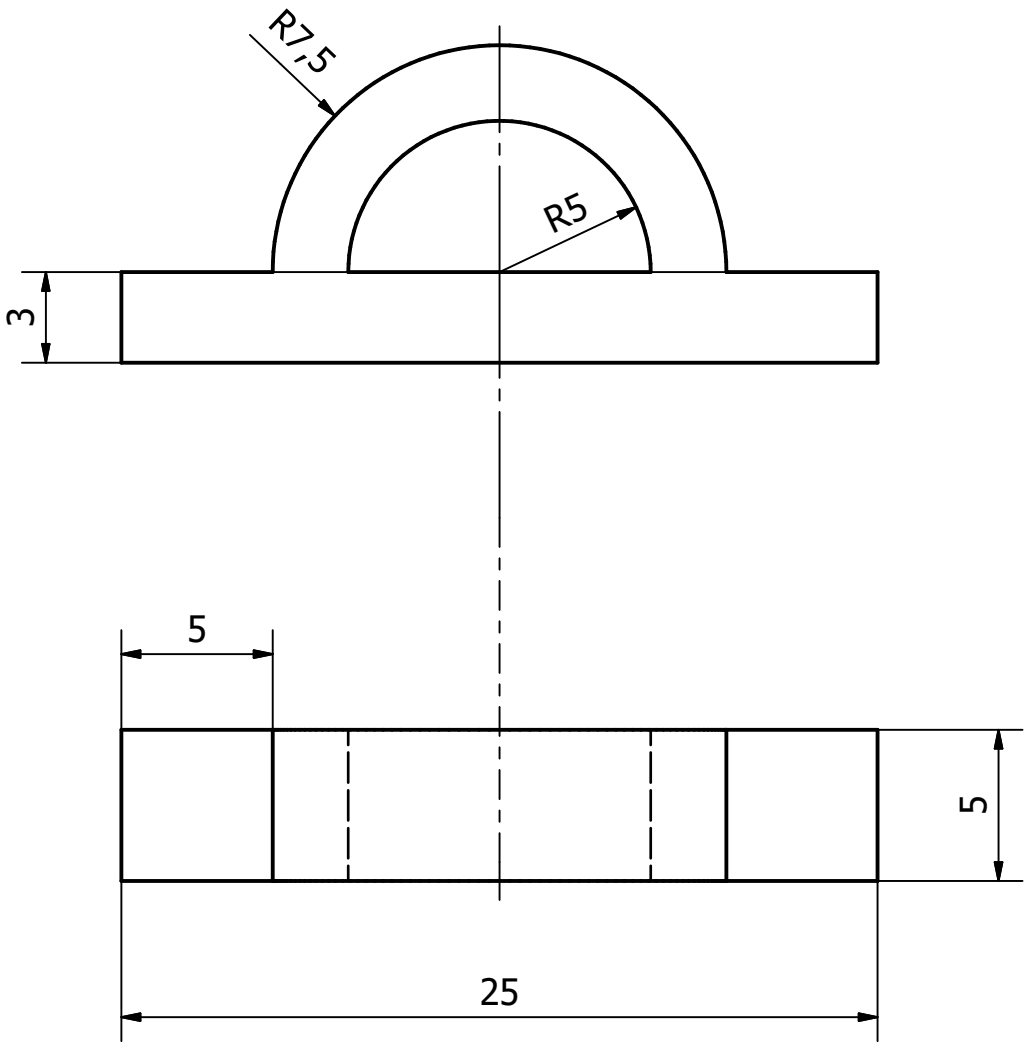


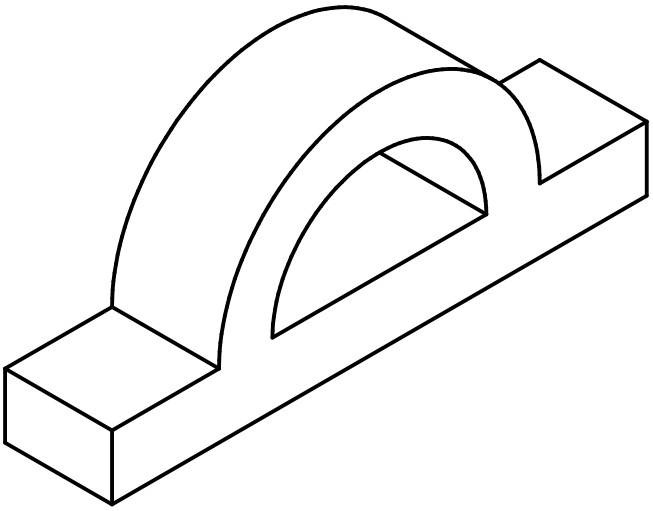




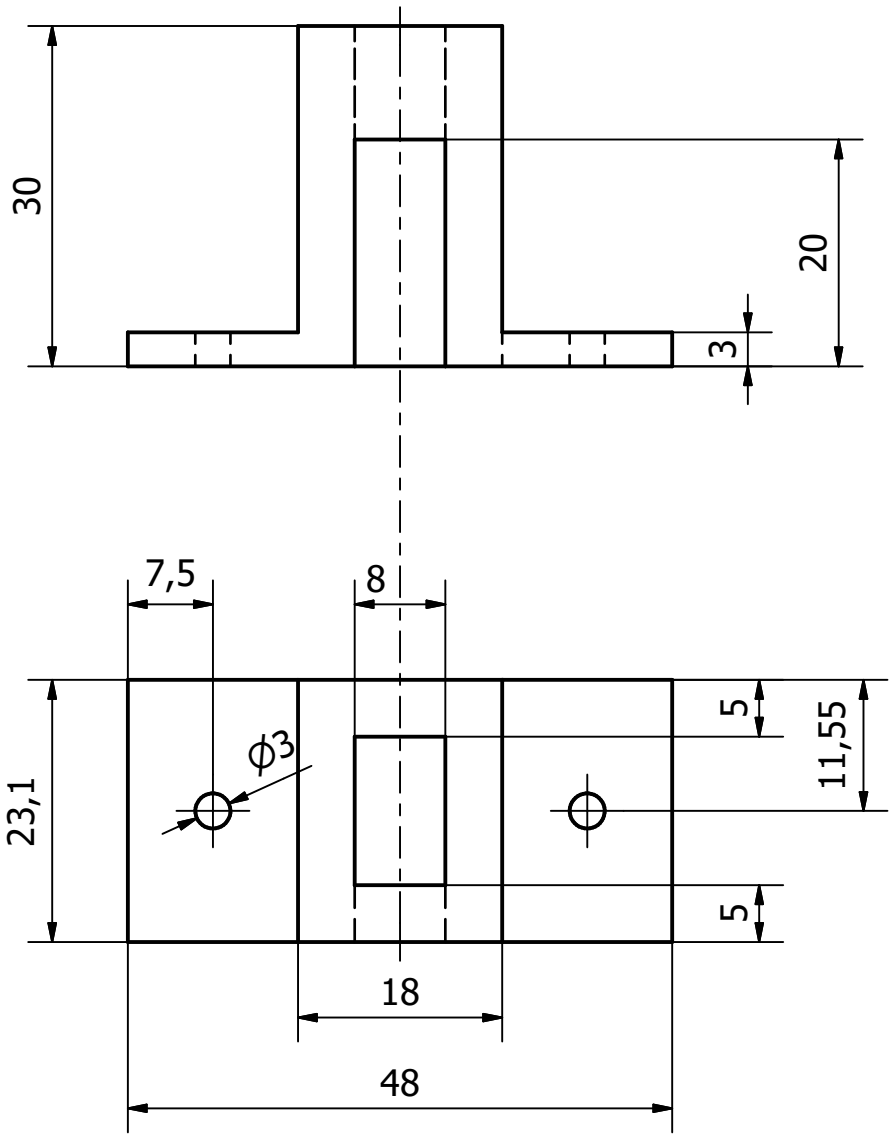


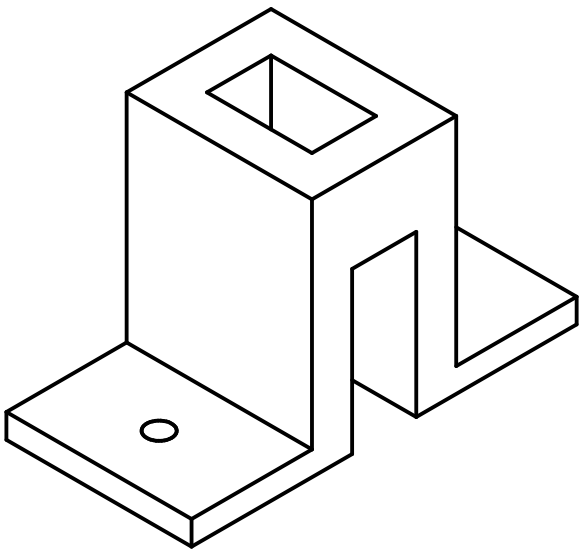
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Cuadro de mandos	Plano 30 / 33



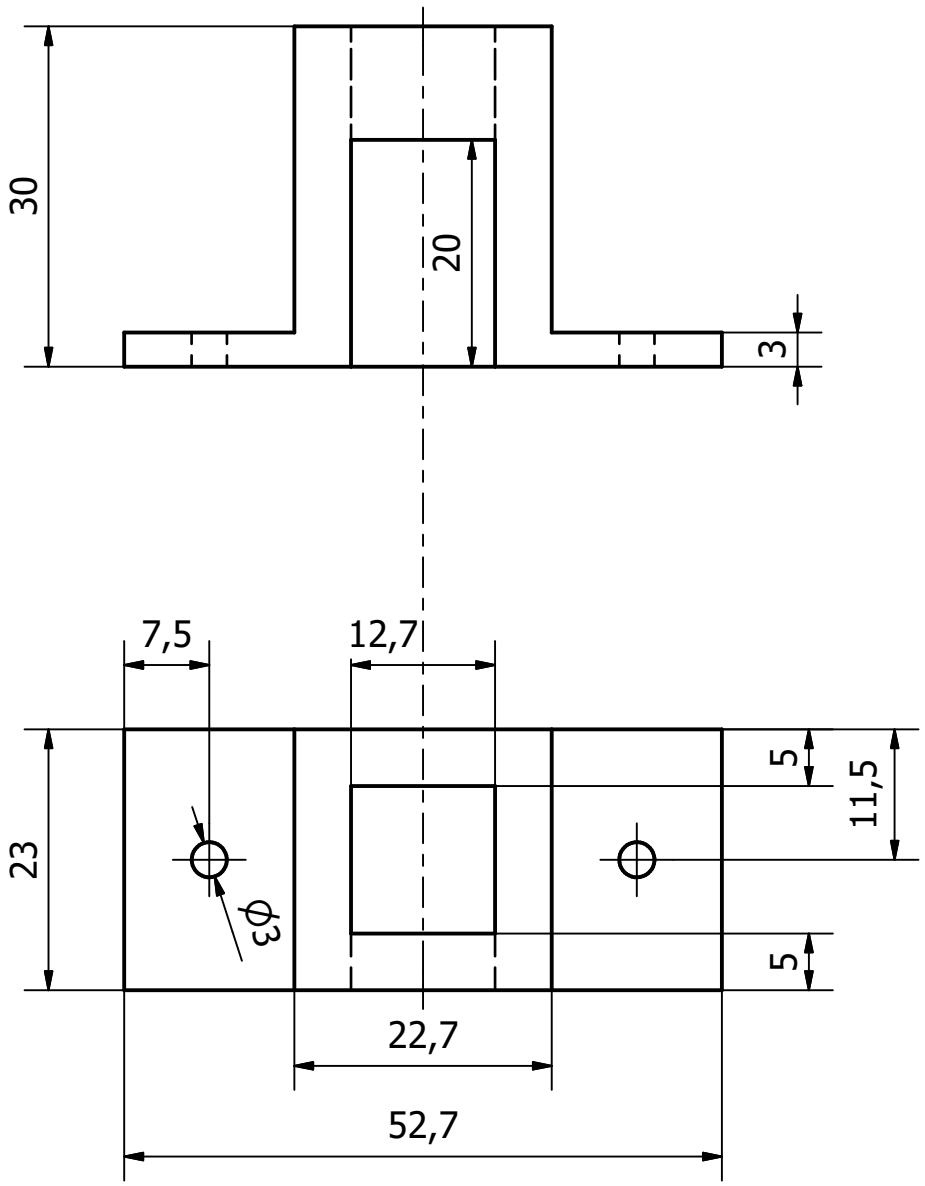


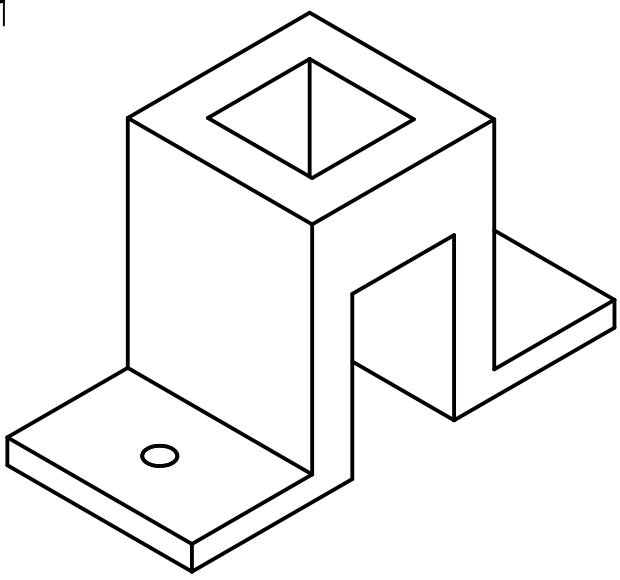
Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Guía cable ultrasonidos	Plano 31 / 33





Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte interruptor 1 canal	Plano 32 / 33





Diseño de Alfredo Alonso Hermosilla	Grado Electrónica Industrial y Automática	Fecha 18/06/2017
TFG Almacén Automatizado EII UVA	Tutor Eduardo Zalama Casanova Dpto. ISA	
	Soporte interruptor 2 canales	Plano 33 / 33