Departamento de Ingeniería de Sistemas y Automática

Trabajo de Fin de Grado

Diseño del circuito electrónico de control y programación de una mano subactuada para el robot humanoide TEO

Titulación:

Grado en Ingeniería en Tecnologías Industriales

Autor: Víctor César Sanz Labella

Tutor: Juan Miguel García Haro

Leganés, Madrid, septiembre de 2014

Resumen

Para la correcta interacción de un robot humanoide con el medio que lo rodea, una vez que se ha conseguido un desplazamiento del mismo por el entorno de manera segura, es totalmente imprescindible la capacidad de desplazar otros objetos. Para ello, es necesario la instalación de una mano que permita agarrar estos objetos de manera eficaz, de forma que se adapte a la forma del objetivo. No obstante, el control preciso para manipular adecuadamente objetos, al ser una tarea compleja, necesitaría de un dispositivo con hasta más grados de libertad que el propio robot.

El robot TEO, humanoide en desarrollo por la Universidad Carlos III de Madrid, incorpora una mano subactuada, la cual permite poder asir objetos mediante un mecanismo de seis grados de libertad, pero con solamente un actuador. Esto puede facilitar enormemente el control de esta mano, garantizando un buen agarre de objetivo.

En este Trabajo Fin de Grado se ha realizado el diseño de la electrónica y del software que permita el control de la mano subactuada a través de la infraestructura de la que dispone el robot humanoide TEO.

Este documento recoge las decisiones de diseño tomadas y las pruebas realizadas para conseguir un funcionamiento correcto del dispositivo, así como el proceso seguido para ello.

Abstract

Once a safe movement within the environment has been assured for a humanoid robot, the capability to drag and move objects becomes essential for a correct interaction within the objects of this environment. In order to accomplish this objective the incorporation of a hand that allows to grip these objects in an effective manner, adapting to the objectives shape. Nevertheless, as the manipulation of objects is a complex task, a device with several additional degrees of freedom than the actual robot is required for finesse.

TEO, a humanoid robot in development by the Carlos III University, incorporates an underactuated hand, which enables to grasp objects through a six degrees of freedom mechanism, but just using one actuator. This provides an easy control of the hand, ensuring a reliable grip.

In this Final Bachelor Project, electronics and software have been designed to achieve a correct operation of the underactuated hand through the already implemented architecture of robot TEO.

As well, the present document collects the design decisions taken and tests performed in order to obtain a proper device operation .

Agradecimientos

Estos agradecimientos son, no solamente por la ayuda y apoyo recibido para este trabajo, sino por el de toda la carrera en general.

En primer lugar, se lo agradezco a mi familia, especialmente a mis padres, que me han proporcionado todas las facilidades posibles para poder estudiar, han estado para todo lo que he necesitado y me han formado como persona, al igual que a mi abuela, a la que le habría hecho mucha ilusión verme ingeniero.

También se lo agradezco a mis amigos, tanto a los de colegio e instituto, por haberme aguantado más tiempo y darme tranquilidad al saber que siempre están ahí para cualquier cosa, como a los de universidad, con los que las clases se hacen más llevaderas y fáciles por su ayuda, tanto académica como anímica, que hace que uno vaya contento a clase. Por suerte, tengo cerca a casi todos los amigos de siempre, muchos de la universidad continúan conmigo en el máster y, los que no, espero que no se libren de verme muy a menudo.

Y, cómo no, en lo relativo a este trabajo, agradezco a mi tutor su ayuda y experiencia, al igual que a todas las personas del laboratorio 1.3.C13, que siempre estaban dispuestas a echar un cable con la electrónica, aunque estuviesen con otra cosa en ese momento.

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Introducción

La convivencia de robots con humanos ha dejado de ser un hecho que solo podía observarse en libros y películas de ciencia ficción para integrarse en la sociedad actual. Hace tiempo que ya se ha conseguido en el campo industrial, donde humanos trabajan junto a robots, que realizan tareas mecánicas a gran velocidad o levantan objetos pesados con facilidad. Un paso más en esta integración es la utilización de los mismos, primero para ciertas labores fuera de la industria y su extensión a otros ámbitos cotidianos como apoyo o sustitución para tareas tediosas. Ejemplo de estas funciones serían el uso de cuadricópteros para la búsqueda de supervivientes en desastres, como por ejemplo terremotos, o la popular aspiradora autónoma Roomba.

El robot humanoide a escala real TEO, que está siendo desarrollando por el grupo RoboticsLab del Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid, tiene como función la interacción en el espacio humano, bien para realizar tareas que lo relacionen, como podría ser la asistencia, o bien para sustituirlos en entornos que puedan resultar dañinos o peligrosos para la salud de los mismos.

Para estas funciones es imprescindible interactuar con objetos, para lo cual es necesario disponer de un dispositivo adecuado que lo permita.

La mano subactuada que incorpora TEO, una FetchHand Lite de la empresa Lacquey, provee la fiabilidad necesaria para una manipulación básica correcta, destinando para ello pocos recursos en cuanto al control de la misma.

1.2. Objetivos

Este trabajo ha abordado el diseño de la electrónica y la programación necesaria para conseguir el movimiento de la mano subactuada FetchHand, cumpliendo los siguientes requisitos:

- Buena integración de la placa electrónica con la estructura, acoplándose en la misma mano, de manera que no interfiera en la mecánica al agarrar objetos.

- Integración con la red actual de comunicación (CAN bus) y resto de hardware, incluyendo la tarjeta de comunicación del ordenador que controla, entre otros dispositivos, la mano.

- Fiabilidad en la electrónica.

- Sencillez en la programación, tanto del microcontrolador de la mano como en el PC del robot que la controla.

- Facilidad para la integración del control mediante el *middleware* YARP, herramienta mediante la cual se controlan los movimientos del robot.

1.3. Estructura del documento

El documento consta de cinco capítulos, de los cuales este primero está dedicado a la introducción y presentación del trabajo, así como los objetivos marcados.

El segundo capítulo hace referencia al estado del arte, presentando al lector los sistemas utilizados y sobre los que se aplicará el diseño explicado en esta memoria con el fin de que pueda hacerse una idea general, como el robot TEO, la mano que actualmente monta y otras posibles, alternativas que podrían haberse tomado para la electrónica de control en la mano, características del bus de comunicaciones utilizado y presentación de las librerías para el control del robot.

El tercer capítulo trata sobre el diseño electrónico que se ha realizado, explicando los componentes utilizados y describiendo a grandes rasgos el proceso seguido. También se comentan las pruebas de validación realizadas para asegurar el funcionamiento correcto de los bloques que componen el diseño.

El cuarto capítulo está relacionado con la programación del dispositivo diseñado y la realizada para implementar en el ordenador que lo controla. Por otra parte, también se incluyen las pruebas de software realizadas para la comprobación del funcionamiento tanto del código escrito como del dispositivo en general, pues su funcionamiento depende directamente de cómo se programe, haciendo que en ocasiones sea difícil distinguir si un error es debido a fallo de hardware o software.

El capítulo quinto analiza los objetivos cumplidos en la realización del trabajo y los resultados obtenidos, y concluye con sugerencias de mejoras futuras.

Para finalizar, se incluyen como anexos algunos datos de interés del proyecto, ente los que se encuentran la lista de componentes, un sencillo presupuesto y la documentación de una clase implementada en C++.

2. Estado del arte

En este capítulo se pondrá al lector en situación sobre algunos aspectos del trabajo para que sea más fácil su comprensión y se tenga una visión más amplia, como descripción del robot sobre el que se aplicará este trabajo, presentación del tecnologías que están aplicadas, otras alternativas existentes respecto a aspectos del diseño, etc.

2.1. Robot humanoide TEO

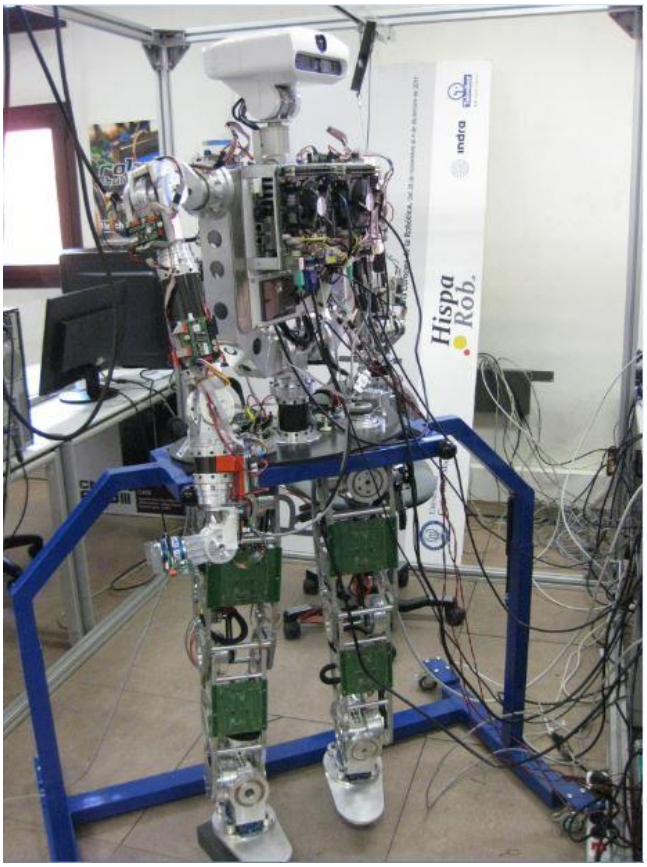


Figura 1. Robot humanoide TEO.

TEO (*Task Enviroment Operator*), robot en desarrollo por parte del grupo RoboticsLab de la Universidad Carlos III de Madrid, y también conocido por el nombre del proyecto Rh-2, es una evolución del robot Rh-1 y de su anterior versión Rh-0. Se trata de un robot antropomórfico de tamaño similar al de un adulto, pues mide 1,65 m de altura, con un peso estimado de tan solo 60 kg. A diferencia de su predecesor, Rh-1, está muy enfocado al bajo consumo energético, y cuya fuente de alimentación de 36 V de pila de combustible se está sustituyendo por baterías, para lograr así un mayor equilibrio entre autonomía y seguridad.

Este robot posee 28 DOF (Degrees Of Freedom, grados de libertad), mientras que el Rh-1 tenía solamente 21, y permite una alta movilidad, lo que conllevará poder desplazarse andando en cualquier dirección con una velocidad estimada de hasta 1 km/h, subir y bajar escaleras, moverse en entornos humanos, transportar hasta 2kg de peso [1].

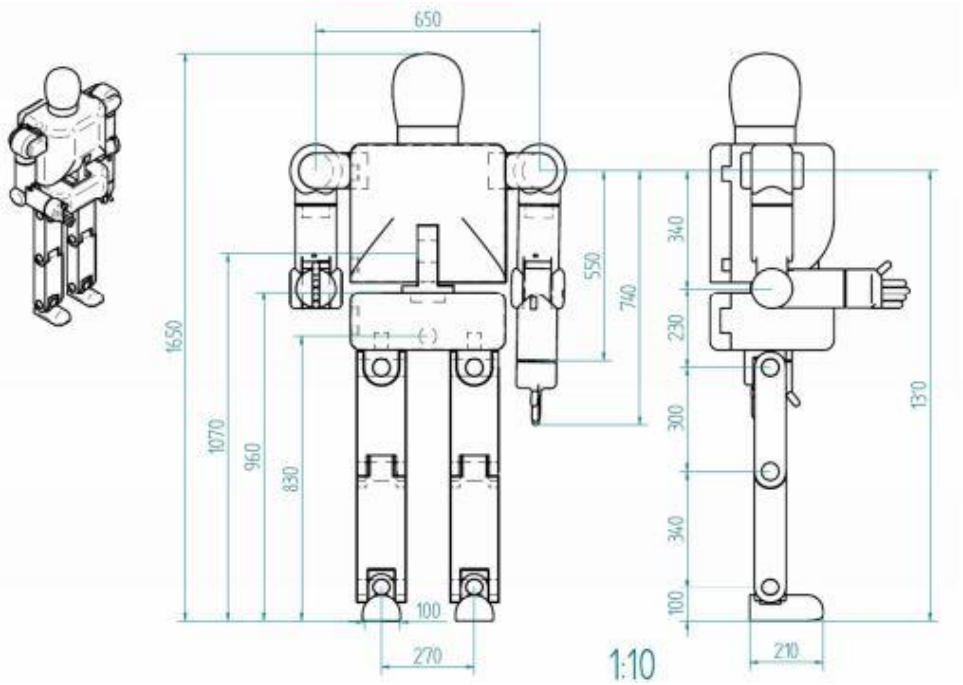


Figura 2. Medidas del robot humanoide TEO.

Para planificar y ejecutar los movimientos, TEO está equipado con dos ordenadores: uno para controlar el tren inferior, es decir, la locomoción, el cual es el ordenador principal, y otro para el tren superior, la manipulación. Cada ordenador, montado sobre el pecho del robot, dispone, entre otras cosas, de:

- Procesador Intel Core 2 Duo (1,6 - 3.3 GHz)
- 3 GB de memoria RAM
- Tarjeta miniPCI CAN (HICO.CAN)
- Conexión Ethernet de 1000 Mbps entre las CPU
- Placa SBC (Single-Board Computer) Gemini, con factor de forma de 5.25", que incluye el microprocesador citado.
- Expansión PCI, donde conectar las tarjetas lectoras de sensores de fuerza JR3.
- Unidad de almacenamiento de estado sólido (SSD) de 120GB, cuya instalación se ha realizado muy recientemente, conectado mediante puerto SATA.

Para el procesamiento de imágenes y de audio se utilizará un tercer microprocesador, no incorporado en estas placas, que ofrecerá una mejor integración y desempeño con el sistema de adquisición de imágenes de TEO.

En cuanto a la comunicación con los dispositivos a través del bus CAN, que es la parte que más afecta al presente trabajo, cabe destacar que la inclusión de dos tarjetas CAN, una por cada PC, con dos nodos CAN por tarjeta, permite la distribución del sistema de información por CANbus en cuatro ramas, una por cada extremidad, pudiendo conseguir un sistema con longitudes de red cortas, que permitan la máxima velocidad de 1 Mbps y evitando la saturación del mismo al estar compuesto solamente por no más de seis nodos.

La tarjeta HICO.CAN-miniPCI (figura 3) dispone de las siguientes características:

- Tarjeta de interfaz miniPCI tipo IIIA.
- Dispone de dos canales de CAN: CAN 2.0A y CAN 2.0B.
- Controlador Philips LPC2292 (ARM7), 60 MHz.
- Transceptores CAN integrados con señales TTL para aislamiento galvánico opcionales.
- Opción de personalizar el firmware por medio de 32 bits integrados.
- Soporta todos los estantadares (CIA-DS-102) con un 100% de velocidad de transmisión.
- Timestamp de 1µs de resolución.
- Buffer interno de capacidad de 500 mensajes CAN por nodo.
- Opcional: rango de temperatura de -40 ºC hasta 85 ºC, aislamiento galvánico de 3000 V.
- El dispositivo soporta Windows 7, Windows CE, Windows XP, Windows 98/ME, Windows 2000, Windows NT, Linux 2.6 y QNX 6.3.86
- Con un adaptador el módulo se puede usar en un bus normal PCI, PC/104+ o
- PCI-104.
- Dos nodos CAN independientes que cumplen con la especificación CAN 2.0B
- LEDs verdes de indicación de mensaje enviado/recibido y rojos para detección de problemas
- Conector 1x 14 pines, con cable incluido para adaptador a 2x DB-9.
- Consumo de corriente máximo de 80 mA.

Figura 3. Tarjeta HICO.CAN-miniPCI.[1]

2.2. Presentación de alternativas existentes para manipulación de objetos.

En este apartado se presentan posibles soluciones existentes para el agarre y manipulación de objetos, característica muy importante en robots humanoides, y si explicarán sus características y ventajas de la integración en el robot TEO, considerando también el estado de desarrollo en el que se encuentra actualmente el robot.

2.2.1. Proyecto Handle

Figura 4. Mano del proyecto Handle sosteniendo un objeto.

El proyecto *Handle* [2] es un ambicioso proyecto de investigación y desarrollo a gran escala, con la colaboración de universidades de seis países europeos: Francia, Reino Unido, España, Portugal, Suecia y Alemania. La Universidad Pierre y Marie Curie de Paris es la coordinadora de las nueve instituciones que forman el consorcio que lleva este proyecto, entre las cuales se encuentra la Universidad Carlos III de Madrid.

Los objetivos de este proyecto son:

- Caracterización de las posibilidades de manipulación de objetos.

[1] La imagen contiene los cables que suministra el fabricante. Se puede observar que las etiquetas que incluyen (CAN 1 y 2) discrepan en la numeración de los nodos con la configuración del software de la propia tarjeta (CAN 0 y 1).

- Aprendizaje e imitación de las estrategias humanas en las operaciones de manejo.
- Mejora de aptitudes mediante" *babbling*".
- Manipulación diestra y autónoma.
- Futuras manos artificiales como dispositivos conectables.

Las asombrosas capacidades que ofrecería una mano de este estilo, con semejante nivel de sensorización son, sin duda, las deseadas en un robot humanoide, pues el nivel de interacción con objetos podría ser equiparable al de un humano, con el impacto en la integración social del robot que esto supone, incluso para tareas asistenciales.

Sin embargo, la integración de esta mano en TEO no es posible debido tanto al estado actual de desarrollo del humanoide, cuyo diseño actual no sería apto para manejar tantos grados de libertad, como de la propia mano, cuyo proyecto, de duración limitada, figura en estos momentos como finalizado.

2.2.2. Mano similar a la humana de bajo coste

Existen muchas soluciones para el agarre de objetos con varios dedos y formas similares a la mano humana, con un número de grados de libertad contenido y sensores de posición que, si bien no tienen todas las características de habilidad y control de una mano humana, sí que pueden permitir el agarre y manejo de distintos objetos.

Sin ir más lejos, en la propia Universidad [3], se ha desarrollado una mano de bajo coste que está compuesta por cuatro dedos, con ocho grados de libertad, ocho actuadores y control de posición de los servomotores mediante Arduino UNO.

No obstante, y aunque la electrónica sería reemplazable por otra más fácilmente integrable, tanto a nivel de inserción física como de comunicaciones con el ordenador de TEO destinado a la manipulación, debido a su carácter aún de prototipo, que conlleva pequeños problemas mecánicos en cuando a agarre de objetos con fuerza o deslizamiento en la transmisión, achacable también al diseño para producción de bajo coste, no hacen de este modelo una alternativa para aplicar en este momento.

Sin embargo, queda patente que este modelo de mano, o uno similar, es la evolución natural del sistema de agarre y manipulación de objetos que se está utilizando ahora, presentado en el siguiente subapartado, y cuyo diseño electrónico y de software se

trata en el presente trabajo, por ofrecer más posibilidades en el control del dispositivo y en la manipulación de objetos.

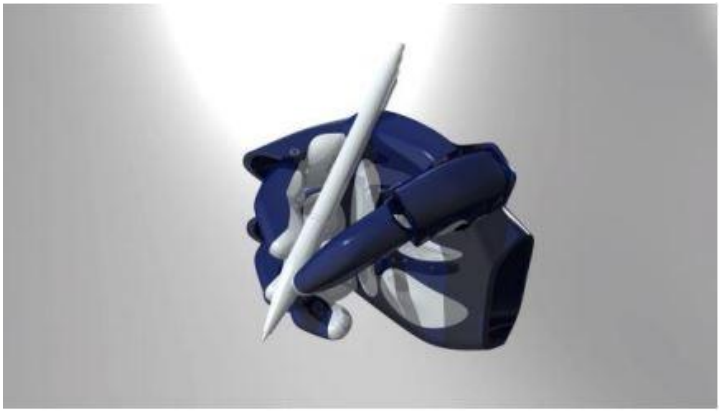


Figura 5. Simulación de manipulación de un bolígrafo por la mano similar a la humana de bajo coste.

Así, partiendo de esta base, y una vez que esté el robot TEO en un estado de desarrollo más avanzado, donde las tareas de locomoción, que son las principales por poner en peligro la integridad del robot en caso de fallo, y se asegure el correcto funcionamiento de la instalación de manipulación actual, se podría actualizar a una nueva versión de esta misma mano de bajo coste desarrollada en la Universidad con los pequeños fallos mecánicos corregidos y diseñando una electrónica de control y software adecuados.

2.2.3. FetchHand: mano robótica subactuada

La subactuación en robótica consiste en el control de varios grados de libertad mediante un menor número de actuadores. En el caso de la mano FetchHand, del fabricante Lacquey, un único motor (actuador) es capaz de mover adecuadamente tres dedos, cada uno compuesto por dos falanges, sumando un total de seis grados de libertad. Este efecto se logra mediante un buen diseño mecánico que reparte la fuerza del motor entre los dedos y consigue mantener un agarre firme de objetos.

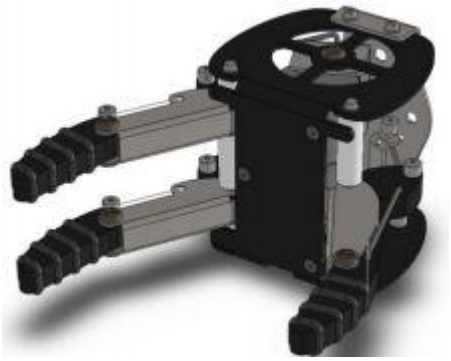


Figura 6. FetchHand, mano robótica subactuada.

El uso de la subactuación conlleva varias ventajas: reducción del peso del sistema al necesitar solamente un actuador y no hacer falta sensores, reducción en la complejidad del control y adaptabilidad del agarre de la mano, distribuyendo además las fuerzas de contacto. En la figura 7 se puede estudiar el proceso de agarre de esta mano con mayor detalle.

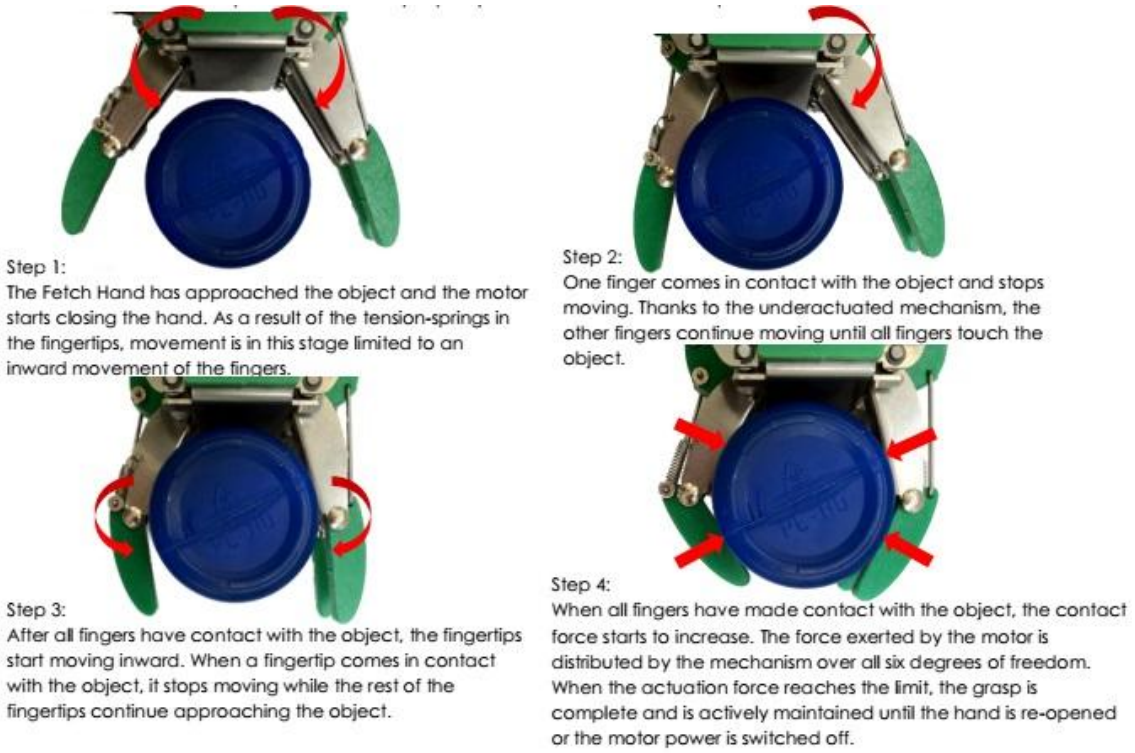


Figura 7. Proceso de agarre de un objeto por la mano FetchHand de Lacquey.

El robot humanoide TEO incorpora dos FetchHand Lite, un modelo algo más ligero que el original al reducir el uso del acero inoxidable para la estructura. También posee partes de plástico (POM), y estrías de goma en las falanges para un agarre superior.

Incluye, además, una pequeña placa electrónica montada en el interior de la caja que forma la mano, junto al motor, que según el fabricante ejerce el control de fuerza y protección contra sobrecarga. Esto es lo que permite un amplio rango válido de alimentación, de entre 12 V a 30 V. También indica que el consumo máximo es de 0,35A, y que valores superiores a esta corriente y 12V no implicarán un movimiento más rápido ni un agarre más fuerte [4].

Por otra parte, el consumo real medido en las pruebas es mucho menor que el indicado, siendo el consumo máximo medido de 0,12 A para el caso que más demanda produce, que es la alimentación a 12 V.

Esta mano, al ser una solución muy robusta, fiable y sencilla, de un coste inferior al de otras soluciones[2], que se adapta su forma para agarrar distintos objetos, y, sobre todo, que requiere poco control para conseguir agarrar y sostener objetos con éxito, hacen de esta mano una buena solución para que haya sido incorporada a TEO, según el estado de desarrollo del mismo, si bien podría sustituirse en un futuro, como se ha comentado anteriormente, por otro dispositivo que requiera un mayor control.

2.3. Opciones de diseño para el control de la mano FetchHand

En este apartado se resumen las consideraciones que han llevado durante el desarrollo del presente trabajo a un diseño de una placa electrónica basada en un microcontrolador integrado en la misma, planteando algunas alternativas que podrían haberse tomado.

2.3.1. Control mediante Arduino

Arduino es una plataforma de hardware libre en forma de pequeñas placas electrónicas que integran, generalmente, un microcontrolador Atmel, y que están listas para usarse, pues incluye cabezales para conectar directamente cables, conexión USB y entrada de alimentación externa, si es necesaria. Además, proporciona un sencillo entorno de desarrollo para programar (IDE), y un lenguaje basado en C con un gran número de librerías que se pueden utilizar, además de que tiene una gran comunidad de usuarios que realizan aportaciones, tanto de software como módulos de ampliación del hardware.

Estos módulos de ampliación, que se presentan normalmente como otra placa con forma análoga al Arduino al que amplía, suelen tener todos los pines de este en la misma posición por un lado, y cabezales de acceso a los mismos por el otro, de manera que se siga teniendo acceso a los mismos, se denominan escudos (*shields*).

Figura 8. Ejemplo de expansión de una tarjeta Arduino UNO mediante un escudo.

[2] El coste actual del dispositivo es de 905€, impuestos indirectos no incluidos.

Aunque el coste de la realización del control de la mano sería bajo en cuanto a la parte del microcontrolador, sería necesario encontrar un módulo *shiled* de expansión de la misma que permitiera el control de las acciones mediante el bus CAN, además de otro para el motor. No obstante, esta solución, aunque sencilla en cuanto a montaje, no sería válida respecto a la integración, al resultar un dispositivo compuesto por estas placas, unas encima de otras: Arduino, escudo CAN y escudo de control de motor.

2.3.2. Control utilizando placa mbed LPC1768.

Esta pequeña placa [5], la cual va a ser implementada en varias partes de TEO para obtener la lectura de sensores, implementa un microcontrolador basado en ARM Cortex-M3 a 96 MHz. Incluye 512 KB de memoria FLASH, 32KB de RAM y una gran cantidad de interfaces, como son Ethernet, USB Host/Device, 2xSPI, 2xI2C, 3xUART, **CAN**, 6xPWM, 6xADC y puertos de entrada y salida de propósito general.

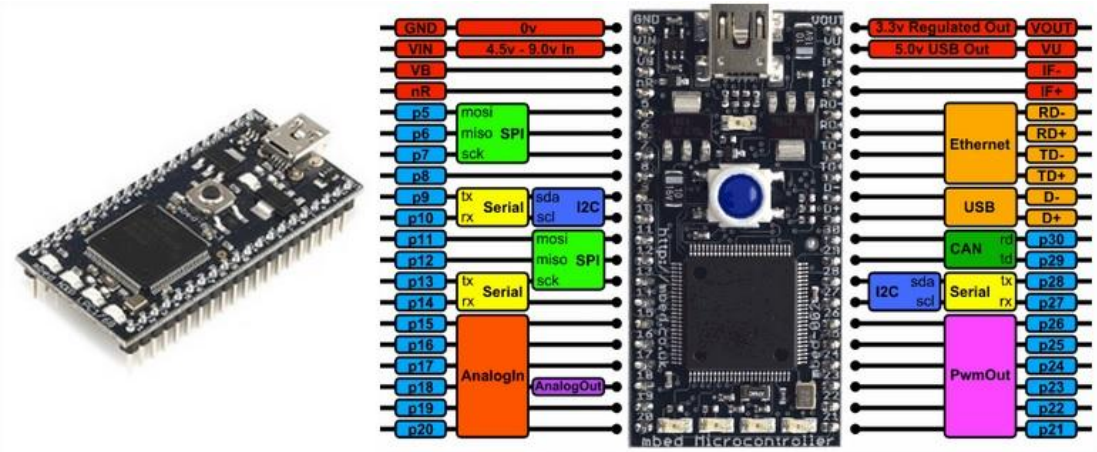


Figura 9. Placa MBED PC1768.

Comparando con la propuesta del subapartado anterior, la del diseño cono Arduino, este dispositivo equivaldría a una placa Arduino más las funciones del escudo CAN. Sin embargo, el propósito de esta placa es la de ser integrada en otra, por lo que se seguiría teniendo que diseñar una PCB para soportarla, alimentarla, proporcionar los conectores necesarios para el bus CAN e controlar el motor, desperdiciando la mayor parte de capacidades de la placa MBED. Este razonamiento lleva directamente a la solución del siguiente subapartado.

2.3.3. Control mediante una placa específica

Según lo expuesto anteriormente en este apartado 2.3, este trabajo realiza el control de la mano FetchHand mediante un circuito electrónico impreso diseñado específicamente a tal efecto, y cuyo diseño se trata en este documento.

Esta solución permite ajustar mucho más las especificaciones del dispositivo, disponiendo solo de las características que sean necesarias, además de que, al estar todos los componentes en una misma placa, reducen el tamaño de la misma y mejoran su integración dentro del sistema.

2.4. Bus de comunicaciones CAN

En este apartado se introduce el origen y algunas características del bus utilizado para el control, CAN, así como algunos aspectos del funcionamiento del mismo.

CAN (Controller Area Network) [6] [7] [8] es un protocolo de comunicación serie desarrollado por Bosch (Robert Bosch GmbH) de tipo asíncrono y *half-duplex* (puede enviar y recibir, pero no simultáneamente), basado en una topología bus para la transmisión de mensajes en entornos distribuidos. Fue ideado originalmente en 1983 para su aplicación en automóviles, por lo que se diseñó para ser altamente inmune al ruido.

2.4.1. Características del bus CAN

El modo de transmisión de este protocolo permite que un mismo mensaje sea recibido por todos los nodos, que según su configuración de filtro lo aceptarán o no, pues está orientado hacia el mensaje en lugar de hacia el nodo. Cada mensaje contiene un , el cual es usado para arbitraje en caso de colisión por intento de envío de varios mensajes simultáneos por dos o más nodos. Esta forma de operación permite también la existencia de varios nodos que actúen como maestro dentro de una misma red, y cualquier nodo puede escribir en ella, siempre que esté libre. También se puede realizar una petición de información a otro nodo mediante el campo RTR (Remote Transmission Request).

Con esta forma de operación del protocolo, sus principales características son :

- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción por multidifusión (multicast) con sincronización de tiempos.
- Sistema robusto en cuanto a consistencia de datos.
- Sistema multimaestro.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas
- Distinción entre errores temporales y fallas permanentes de los nodos de la

red, y desconexión autónoma de nodos defectuosos.

2.4.2. Medio físico del bus CAN

El medio más común por el que se transmite la información es un par trenzado. Uno de los cable corresponde al nivel alto, CAN H, y otro al bajo, CAN L. Estas señales son en realidad una única señal, transmitida en forma de diferencia de tensión entre este par, lo que otorga una alta inmunidad contra el ruido. Los valores que se pueden enviar son dominante, con el valor lógico '0', o recesivo, con el valor lógico '1'.

La impedancia entre estos cables viene definida por el estándar ISO 11898-2 con un valor de 120 Ω. Estas resistencias deben colocarse en los finales de la línea para delimitarla, eliminar los reflejos de señal al final del bus y asegurar unos niveles de tensión correctos.

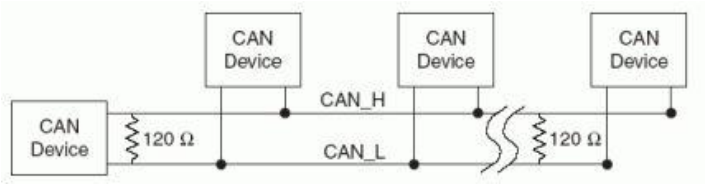


Figura 10. Esquema de conexión en una red CAN.

Las máximas longitudes de línea permitidas varían según la velocidad de transferencia. algunas de ellas son:

- 40 metros a 1 Mbps
- 100 metros a 500 kbps
- 200 metros a 250 kbps
- 500 metros a 125 kbps
- 6 kilómetros a 10 kbps

2.4.3. Formato de mensajes

La especificación 2.0 de Bosch [9] para el bus CAN contempla una caso A, que es compatible con la antigua especificación 1.2 de identificador estándar de 11 bits, y una parte B en la que se extiende el identificador a 29 bits.

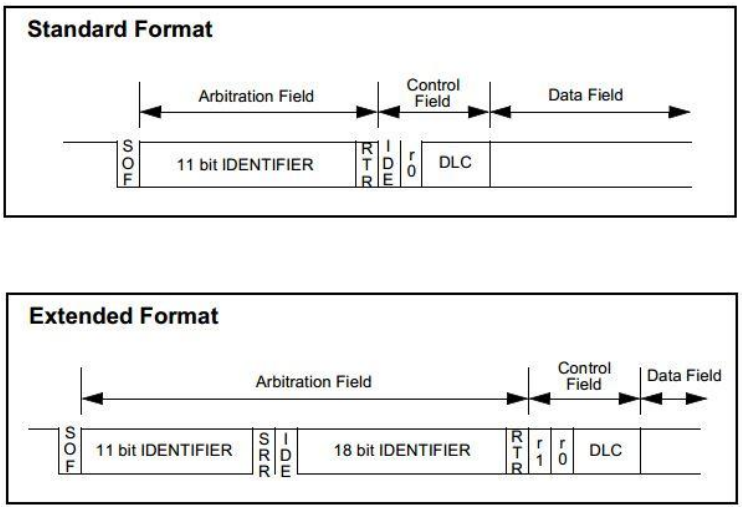


Figura 11. Formato de trama de un mensaje CAN.

Los campos que se pueden ver en la figura 11 son los que componen el mensaje CAN, que son:

- El **campo de arbitraje**, que contiene el identificador (si se trata de la especificación 2.0 B está divido en 11 bits por una parte y 18 por otra, separados por el bit de petición de transmisión remota para el formato extendido, SRR, y el bit de indicación de trama extendida, IDE).
- El campo de **datos**, que puede contener hasta 8 bytes.
- El campo de **control**, que incluye dos bits reservados y 4 bits más para el **DLC**, que indica la cantidad de bytes de datos que se envían.
- El campo **CRC**, que contiene un *checksum* de 15 bits.
- El campo **ACK**, durante el cual se confirma la recepción de un mensaje.

2.5. YARP

En este apartado se darán unas indicaciones muy breves de lo que es este componente y su utilidad para controlar al robot TEO. El diseño del software de este trabajo, tal y como se explicará en esta memoria, está escrito para que pueda ser fácilmente integrable en esta plataforma, pues es la herramienta que se está empleando.

YARP (Yet Another Robot Platform) [10] es, como dice la misma plataforma, un pequeño *middleware* para robots humanoides y más.

Se trata de un pequeño software que actúa de interfaz, compuesto por un conjunto de librerías, protocolos y herramientas para abstraer el manejo de dispositivos y facilitar la comunicación entre ellos.

Consta de tres componentes:

- libYARP_OS: ofrece la comunicación básica interactuando con el sistema operativo. Es totalmente multiplataforma y también multiarquitectura.
- libYARP_sig: está orientado al procesamiento de grandes datos, como imágenes y audio.
- libYARP_dev: sirve como interfaz para el control de distintos tipos de dispositivos, como motores, multimedia como cámaras digitales, etc. Provee una serie de clases genéricas para la gestión de estos tipos de dispositivos (por ejemplo, para control de motores se incluye una clase *yarp::dev::IPositionControl*, que tiene como funciones miembro parar, mover a una posición, seleccionar velocidad y otras muchas otras).

Este *middleware* tiene las ventajas de ser ligero, multiplataforma y de fácil uso. Se emplea para asociar a una clase una cadena de caracteres y poder controlar dispositivos mediante la línea de comandos tanto desde el ordenador en el que se está ejecutando como desde otro remoto conectado en red.

3. Hardware diseñado y pruebas de componentes.

En este capítulo se describirá el diseño desde el punto de vista del hardware (bloques funcionales necesarios, componentes seleccionados, consideraciones tomadas y algunos procedimientos seguidos), así como las pruebas realizadas para verificar el funcionamiento de cada subsistema.

3.1. Bloques del diseño

Teniendo en cuenta que la alimentación disponible es de 36 V de corriente continua, la comunicación se realiza mediante el bus CAN, y la mano FetchHand se controla mediante dos terminales sobre los que se puede aplicar una diferencia máxima de tensión de 30 V (valor a partir el cual la FetchHand puede sufrir daños) y una mínima de 12 V para abrir o cerrar (cambiando la polaridad de la misma), se ha diseñado un sistema capaz de aplicar una diferencia de tensión en esos terminales de unos 25 V, de manera que siempre quede protegida, y que consta de los siguientes bloques principales:

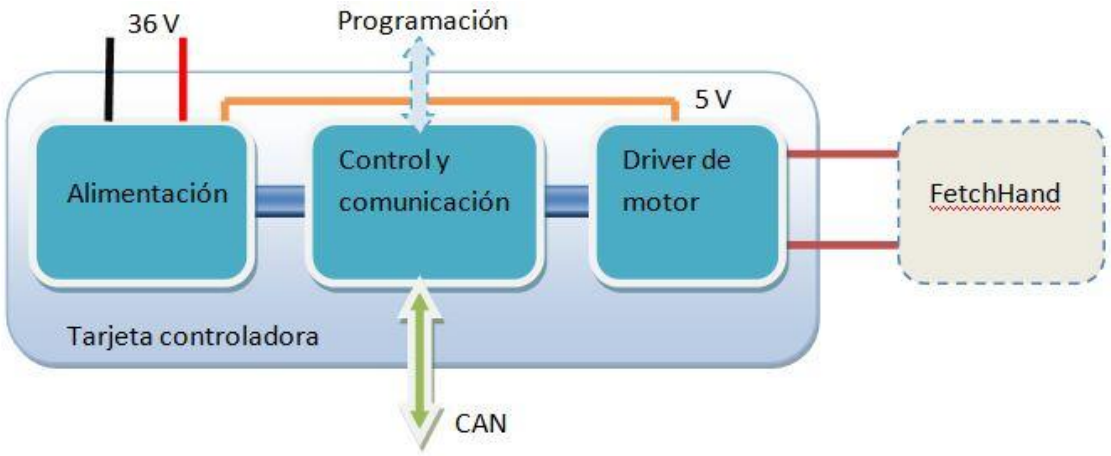


Figura 12. Diagrama de bloques de la tarjeta controladora.

Así, el bloque *Alimentación*, deberá realizar dos tareas:

- Rebajar la tensión de 36V a los 5V necesarios para el control de la lógica del bloque *Control y comunicación* y utilizada también en *Driver de motor*. Se debe tener en cuenta una eficiencia energética y una temperatura aceptable, por lo que, como se explica más adelante se incorporará un regulador conmutado.

- Incorporará un pequeño regulador que rebajará 10 V la tensión de potencia del bloque *Driver de motor* con el fin de alimentar la mano con una tensión por debajo de 30 V y someter a dicho bloque a una exigencia menor.

- Se añadirá un diodo supresor de transitorios (TVS) de 36 V como protección adicional para el circuito.

El bloque de *Control y comunicación* estará compuesto, principalmente, por un microcontrolador, encargado de generar las señales adecuadas en función de las órdenes enviadas por la red CAN. También incluirá un transceptor para poder comunicarse por la red CAN, indicaciores luminosos que ayuden a la identificación de fallos y un conector dedicado para la programación del micro dentro del circuito (ICSP, In-circuit serial programming).

Por último, el bloque de *Driver de motor* incluirá un puente H que transmitirá la potencia a la FetchHand según las instrucciones del microcontrolador, haciendo que esta se quede inactiva, abra o cierre.

3.2. Selección de componentes

En esta sección se detallan las características y se justifica la elección de los componentes principales incluidos en el diseño, como pueden ser los circuitos integrados y conectores, así como alguna observación sobre los componentes auxiliares necesarios para su correcto acondicionamiento.

Dichos componentes son, según los bloques funcionales antes mencionados:

- Alimentación: Regulador LM2575, diodo zéner 1N5347B.

- Control y comunicación: microcontrolador PIC18F2580, transceptor MCP2551.

- Driver de motor: puente H L293D.

3.2.1. Microcontrolador PIC18F2580

Este microcontrolador de la compañía Microchip [11], disponible tanto en encapsulados de orificio pasante como de montaje superficial, que se ha seleccionado por lo siguiente:

- Incluye módulo ECAN, por lo que se programa directamente y se evita una comunicación mediante otro protocolo con el transceptor. De esta manera, el transceptor solo debe realizar la adaptación de niveles.
- Experiencia en el Departamento con este microcontrolador para otros diseños, poseyendo la herramienta de programación y depuración MPLAB ICD2.
- Descarga gratuita del entorno de programación MPLAB IDE y compilador C18, que incluye librerías para facilitar el desarrollo en esta plataforma. Si bien la

utilización del ICD2 no es compatible con las herramientas de software más modernas de Microchip, MPLAB X IDE y MPLAB XC Compiler, las herramientas mencionadas son más que suficientes para el correcto desarrollo del proyecto. [3]

Las características más relevantes de este microcontrolador se encuentran resumidas en la Tabla 1:

Features	PIC18F2480	PIC18F2580	PIC18F4480	PIC18F4580
Operating Frequency	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz	DC – 40 MHz
Program Memory (Bytes)	16384	32768	16384	32768
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	1	1	1	1
Enhanced Capture/ Compare/PWM Modules	0	0	1	1
ECAN Module	1	1	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Parallel Communications (PSP)	No	No	Yes	Yes
10-bit Analog-to-Digital Module	8 Input Channels	8 Input Channels	11 Input Channels	11 Input Channels
Comparators	0	0	2	2
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), $\overline{MCLR}$ (optional), WDT
Programmable High/Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled	75 Instructions; 83 with Extended Instruction Set enabled
Packages	28-pin SPDIP 28-pin SOIC 28-pin QFN	28-pin SPDIP 28-pin SOIC 28-pin QFN	40-pin PDIP 44-pin QFN 44-pin TQFP	40-pin PDIP 44-pin QFN 44-pin TQFP

Tabla 1. Características del microcontorlador PIC18F2580 y comparativa con otros de su misma familia.

Respecto a los módulos mencionados, a continuación se amplía muy brevemente la información sobre el módulo ECAN y el CCP:

- ECAN: el módulo incluido tiene las siguientes características:

[3] En caso de ser necesaria la utilización del nuevo software, siempre se podría actualizar el programador, por ejemplo, a un PICkit 3, con un coste aproximado de 30€, y con mayor poder de depuración que el ICD2.

- Permite una tasa de transferencia de hasta 1 Mbps
- Cumple la normativa 2.0B que especifica Bosch
- Totalmente retrocompatible con los módulos CAN de los PIC18XXX8
- Tres modos de operación: *Legacy* (modo 0)*, Enhanced Legacy* (modo 1) *y FIFO* (modo 2)
- Tres buffers dedicados a la transmisión con prioridades
- Dos buffers dedicados a la recepción
- Seis buffers programables de transmisión/recepción
- Tres máscaras de aceptación de 29 bits completos
- Dieciséis máscaras de aceptación de 29 bits completos asignables dinámicamente
- Soporta filtrado de byte de datos de DeviceNet ™
- Tratamiento automático de tramas remotas
- Características avanzadas de tratamiento de errores

- CCP: este módulo es un periférico integrado que puede capturar, comparar o generar señales PWM:

- Modo captura: cuando se detecta una evento en el pin de CCP, se guarda el valor del temporizador asociado y se activa el *flag* de interrupción. Se puede configurar para que se detecte un evento por el pin de una de estas cuatro maneras: cada flanco de subida, cada flanco de bajada, al cuarto flanco de subida o al decimosexto flanco de subida.
- Modo comparar: permite cambiar el valor del pin CCP cuando el registro del CCP y el del temporizador asociado tienen el mismo valor, haciendo que el pin , según cómo se haya configurado, tenga uno de estos cuatro valores: nivel alto, nivel bajo, cambio de bajo a alto y viceversa según el valor en el que estaba o mantenerse sin cambios respecto al lazo de entrada y salida.
- Modo PWM (*Pulse Width Modulation*, modulación por ancho de pulso): permite generar señales periódicas de periodo configurable que pueden tener solo los valores alto y bajo. La proporción entre el tiempo que esta señal se encuentra a nivel alto y el tiempo que dura un periodo se denomina ciclo de trabajo. Es útil para simular salidas analógicas de manera digital, y es el modo por el que se comenta este periférico, pues se puede utilizar para aplicarlo a la entrada del puente H y controlar así la tensión media que le llega a la FetchHand, teniendo como fin menos par en el motor y menos consumo del mismo.

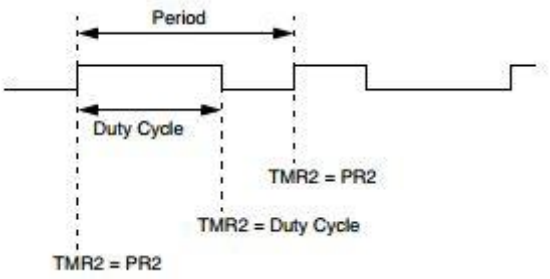


Figura 13. Onda PWM.

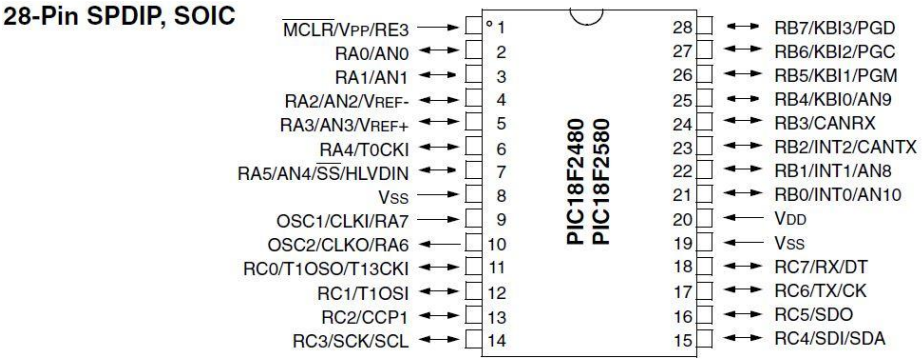


Figura 14. Patillaje del microcontrolador con el encapsulado seleccionado

Por último, cabe destacar el oscilador que servirá como señal de reloj para le microcontrolador. Se ha elegido un pequeño cristal de cuarzo de 20 MHz de la marca QANTEK por ser pequeño, con una tolerancia de frecuencia de ±10ppm y una estabilidad de frecuencia de ±20ppm. Esto cumple con las especificaciones CAN de Bosch para la versión 2.0 [9], donde se puede leer:

> *In order to increase the maximum oscillator tolerance from the 0.5% currently possible to 1.5%, the following modifications, which are upwards compatible to the existing CAN specification, are necessary:*
>
> [...]
>
> *This modifications allow a maximum oscillator tolerance of 1.58% and the use of a ceramic resonator at a bus speed of up to 125 Kbits/second. For the full bus speed range of the CAN protocol, still a quartz oscillator is required. The compatibility of the enhanced and the existing protocol is maintained, as long as:*
>
> *[7] CAN controllers with the enhanced and existing protocols, used in one and the same network, have all to be provided with a quartz oscillator.*

Los dos condensadores que se necesitan para el oscilador se han elegido de 15 pF, de acuerdo con la siguiente tabla del manual del microcontrolador:

Osc Type	Crystal Freq	Typical Capacitor Values Tested:	
		C1	C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	1 MHz	33 pF	33 pF
	4 MHz	27 pF	27 pF
HS	4 MHz	27 pF	27 pF
	8 MHz	22 pF	22 pF
	20 MHz	15 pF	15 pF

Tabla 2. Guía de selección de condensadores según el cristal utilizado.

3.2.2. Transceptor MCP2551

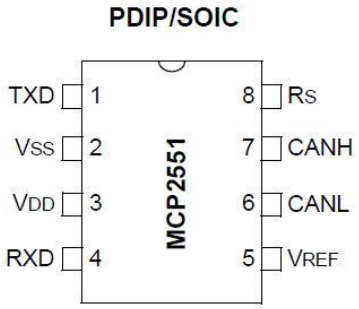


Figura 15. Patillaje del transceptor utilizado.

Este circuito integrado [12] adapta los niveles del módulo ECAN del microcontrolador a los estándares para la capa física de la red CAN, según la norma ISO-11898, que tiene como características más relevantes, entre otras:

- Detección de fallos en la tierra (estado dominante permanente en pin TXD)
- Protección contra cortocircuito y transitorios de alto voltaje
- Apagado automático por temperatura
- Soporte para hasta 112 nodos
- Alta inmunidad al ruido
- Posibilidad de control de emisiones de radiofrecuencia
- Soporte para operar hasta 1 Mbps

A continuación se enumeran las funciones de cada pin del circuito integrado:

- **Pin 1, TXD**: entrada de transmisión de datos desde el microcontrolador.
- **Pin 2, Vss**: conexión a la masa del circuito.
- **Pin 3, Vdd**: conexión a la alimentación, 5 V.

- **Pin 4, RXD**: salida de transmisión datos hacia el microcontrolador.
- **Pin 5, Vref**: salida de tensión de refencia (Vdd/s).
- **Pin 6, CAN-L**: conexión de entrada/salida a la línea de nivel bajo del bus.
- **Pin 7, CAN-H**: conexión de entrada/salida a la línea de nivel alto del bus.
- **Pin 8, Rs**: selección de velocidad de transmisión según el valor de la resistencia que se coloque entre el pin y masa.

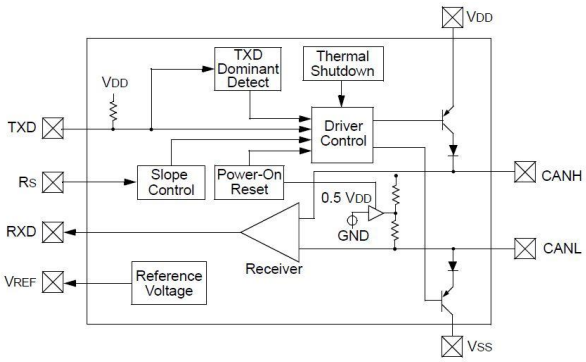


Figura 16. Diagrama de bloques del transceptor.

Pin Number	Pin Name	Pin Function
1	TXD	Transmit Data Input
2	VSS	Ground
3	VDD	Supply Voltage
4	RXD	Receive Data Output
5	VREF	Reference Output Voltage
6	CANL	CAN Low-Level Voltage I/O
7	CANH	CAN High-Level Voltage I/O
8	RS	Slope-Control Input

Tabla 3. Numeración y función del patillaje del transceptor.

Respecto al pin 8, se puede configurar el transceptor en uno de estos tres modos:

- **Modo *High-Speed***, que tiene unas pendientes de transición lo más rápidas posibles para poder trabajar a máxima velocidad. Se consigue conectando el pin directamente a masa, y será el modo utilizado, pues la red de TEO trabaja a 1 Mbps.
- **Modo *Slew-Rate***, el cual es útil para reducir el nivel de emisiones electromagnéticas (EMI) controlando las pendientes de transición, con la consecuencia de disminuir la máxima tasa de transferencia. Los valores típicos de la pendiente según la resistencia colocada se pueden ver en la figura 17.

- **Modo *Standby***, en el que se configura el transceptor en un estado de bajo consumo apagando la parte transmisora y haciendo que la parte receptora opere con poca corriente (menor tasa de transmisión). Así, el microcontrolador, que aún es capaz de monitorizar mensajes entrantes, puede volver a configurarlo en un modo de consumo normal, aunque podría perder el primer mensaje si la tasa de transferencia es elevada. Este modo de bajo consumo se configura aplicando un nivel alto a la patilla 8.

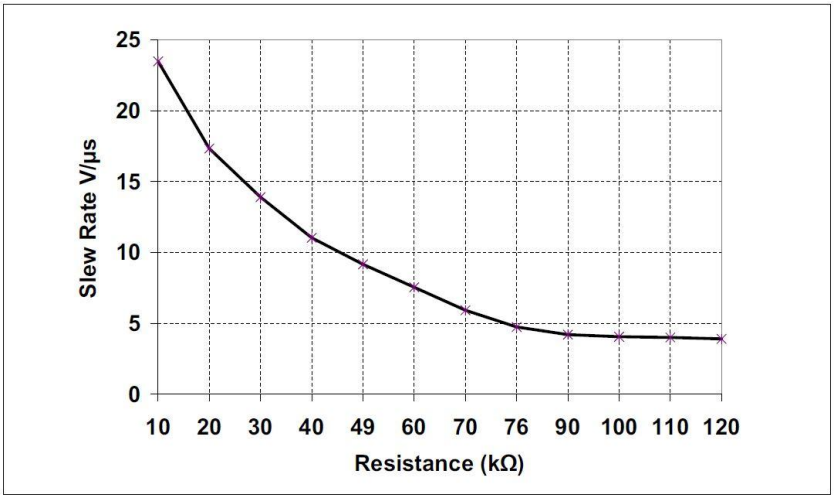


Figura 17. Tasa de cambio en función de la resistencia montada.

3.2.3. Regulador LM2575

Este circuito integrado [13] contiene algunos de los componentes que formarían parte de un convertidor DC/DC conmutado reductor (*buck*) con el fin de rebajar los 36 V de la alimentación del robot hasta 5 V, y cuyo esquema general se puede ver en la figura 18, como sería el cableado, el circuito de disparo y el de control. Este último se encarga de generar la frecuencia de la onda periódica de control, que para este integrado es de 52 kHz, y mediante un lazo de control, ajusta el ciclo de trabajo (véase la explicación del módulo CCP del microcontrolador dentro del subapartado 3.2.1).

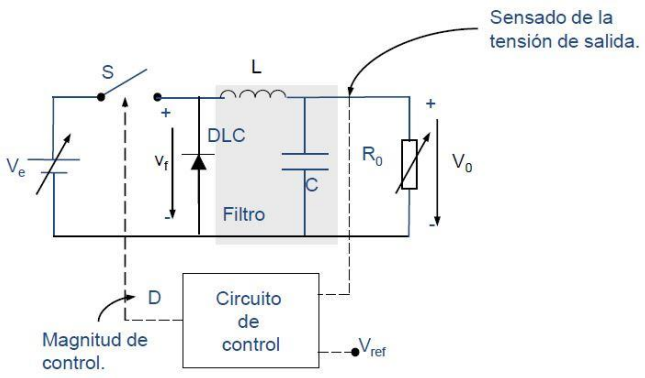


Figura 18. Esquema de convertidor reductor sin aislamiento.

De esta forma, para completar el regulador, son necesarios cuatro componentes, dos condensadores, una bobina y un diodo.

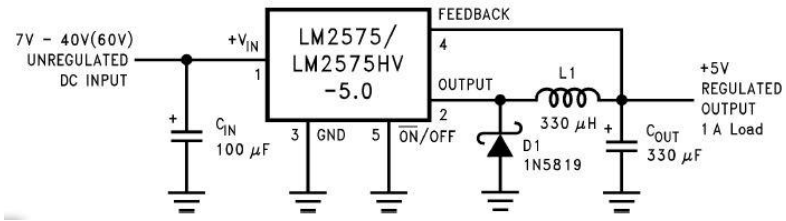


Figura 19. Conexiones del convertidor utilizado.

Algunas características de este dispositivo son:

- Salida fija de 5V
- Salida de corriente de hasta 1A.
- Voltaje de entrada de hasta 40V
- Solo requiere cuatro componentes externos.
- Oscilador interno con frecuencia fija de 52 kHz
- Capacidad de apagado TTL, modo *Standby* de bajo consumo
- Alta eficiencia
- Utiliza inductores estándares de fácilmente disponibles
- Protecciones de limitación de corriente y apagado por temperatura

Las hojas de características de este integrado facilitan el uso del mismo incluyendo una pequeña guía de selección de los componentes auxiliares, y cuyo resultado de seguirla para una Vin=36V y Vout=5V, con una corriente de salida de 0,6A es el siguiente:

- Condensador de entrada: 100µF, 50V.
- Condensador de salida de 330µF.
- Diodo Schottky MBR350 (de la lista de recomendados por el fabricante), de orificio pasante.
- Bobina de 470µH, seleccionada según la figura 20, y con características similares a las de la lista de recomendadas por el fabricante .

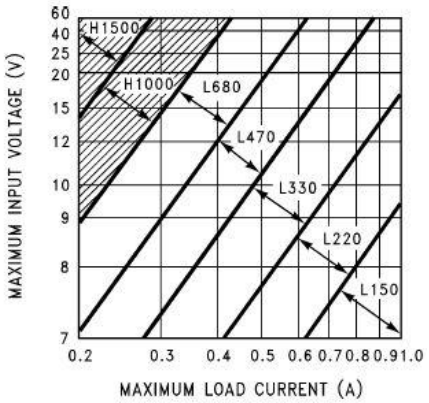


Figura 20. Selección del inductor del convertidor.

Para la elección de estos componentes se han tenido en cuenta los valores anteriormente citados, pero la corriente, que es una magnitud variable, se ha sobredimensionado para poder aguantar mayores corrientes, aunque lo esperado sería el convertidor tuviese que suministrar alrededor de 0,03A. Por ello, en la sección 3.5, se verifica que el inductor seleccionado, que es el componente que podría dar problemas por elegirlo con una corriente sobredimensionada, también sea válido para el suministro de esta corriente.

Respecto al encapsulado elegido, el TO-220 (ver figura 21) provee una buena relación entre capacidad de disipación térmica, dimensiones, facilidad de montaje y ruteado de la placa (al ser de orificio pasante es más flexible en cuanto a la distribución y ruteado por ambas capas). La disposición para colocarlo sería con el cuerpo pegado a la placa, en contacto con un plano de masa que haría de disipador.

Figura 21. Encapsulado TO-220 de 5 patillas del convertidor LM2575.

Las temperaturas, según los datos del fabricante, para un montaje sin disipador (no se tienen valores reales de la disipación por contacto con el plano de masa para calcularlos) serían, según la corriente que se especifica en la tabla 4:

							Montaje superficial		Agujero pasante	
Parámetros	Iload (A)	Ta (ºC)	Vo(V)	Vin(V)	Iq(mA)	Vsat(V)	Rja(ºC/W)	Rjc(ºC/W)	Rja(ºC/W)	Rjc(ºC/W)
	0,5	29	5	36	11	1,3	70	5	65	5

		Montaje superficial	Agujero pasante
Resultados	Pd(W)	Tj sin disip (ºC)	Tj sin disip (ºC)
	0,48627778	63,03944444	60,60805556

Tabla 4. Temperaturas del LM2575 sin disipador para 0,5A y 29ºC en el entorno

Como se puede observar, la temperatura de este componente no implica ningún problema, ya que es razonable al ser sometido a un nivel muy contenido de exigencia, menor que el expuesto en la tabla, y con una plano de masa que actuará como disipador.

3.2.4. Diodo zéner

Se ha elegido el diodo zéner 1N5347B con características principales [14] de tensión zéner nominal de 10V (máxima de 10,5 V y mínima de 9,5 V , medidas con una corriente de prueba de 125mA) y hasta 5W de potencia, siendo capaz de soportar típicamente hasta 475mA de corriente, con una temperatura máxima de operación de 150°C.

El valor de corriente está muy por encima del máximo consumido por la FetchHand (140mA para 12 V), y las derivas de tensión no son preocupantes, puesto que se necesita rebajar la tensión desde 36 V a, como valor máximo, 30 V, por lo que se cumple el propósito de manera holgada hasta en la peor situación.

La forma de disponerlo para que actúe de la manera deseada es la que se puede ver en la figura 22.

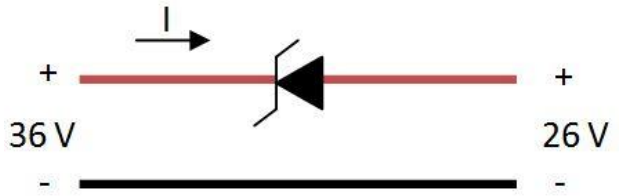


Figura 22. Esquema de disposición del zéner de 10 V para reducir tensión.

3.2.5. Puente H L293D

El circuito integrado L293D se corresponde con cuatro medios puente (cuatro canales), con una entrada digital por cada medio puente y una entrada de habilitación por cada dos medios puentes. La letra "D" indica que esta versión incluye los diodos de protección, necesarios para controlar cargas inductivas [15] [16].

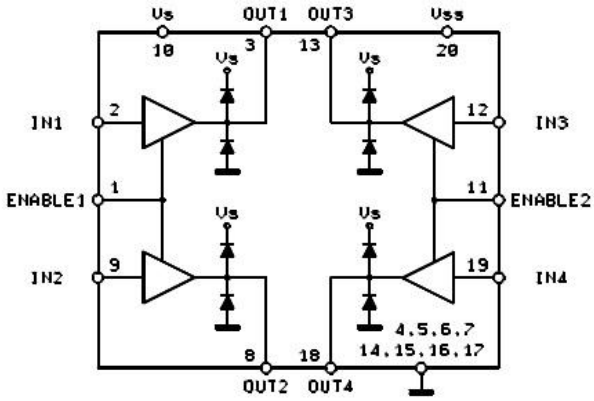


Figura 23. Diagrama de bloques del L293D.

Un puente H, explicado de forma sencilla, es un circuito inversor de dos ramas en el que cada rama tiene dos elementos que se comportan como interruptor. La finalidad es poder alimentar la carga, colocada entre las dos ramas y en medio de los dos interruptores, haciendo que la corriente circule en un sentido de la carga o en el contrario, lo que, en el caso que se está tratando aquí, el motor giraría en un sentido u otro. Esto se consigue controlando apropiadamente los interruptores, como se puede apreciar en la figura 24, la cual es muy ilustrativa.

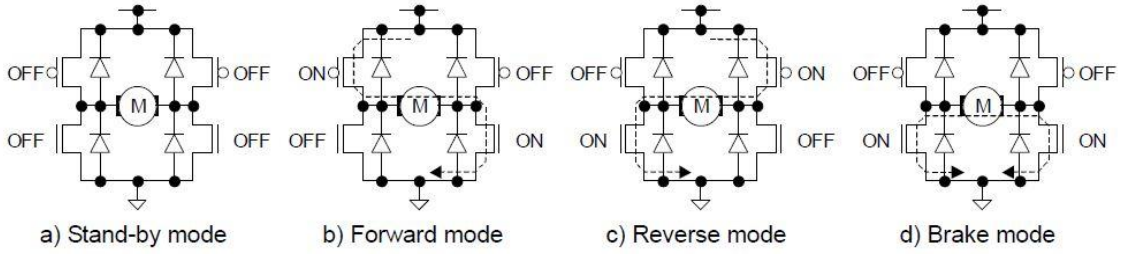


Figura 24. Funcionamiento de un puente H para controlar un motor.

El integrado elegido, como se ha comentado, dispone de de una entrada de habilitación por cada puente. Esto es útil para asegurarse de mantener las dos ramas del puente apagadas y, por ejemplo, realizar un control por PWM por esta patilla (se recuerda que el microcontrolador utilizado solo posee una salida PWM). Además, al ser medios puentes independientes (si se mantiene la habilitación a nivel alto para que la señal de control no sea ignorada), permite el control de hasta cuatro motores en un solo sentido.

En cuanto a las características del integrado, estas son algunas de ellas:

- Amplio rango de tensiones de entrada : 4,5 V - 36 V

- Entrada separada para la alimentación de la lógica (soporta hasta 36V, pero se recomienda que esté entre 4,5 V y 7 V)
- Corriente de salida de hasta 600 mA por canal
- Corriente de pico de hasta 1,2 A por canal
- Nivel '0' del voltaje de entrada de 1,5 V (alta inmunidad al ruido)
- Protección contra sobretemperatura
- Diodos supresores de transitorios inductivos incorporados

En cuanto al patillaje para el encapsulado PDIP, la distribución se puede ver en la figura 25 y las funciones resumidas son las siguientes:

- **Pin 16, Vss:** Tensión de alimentación
- **Pin 8, Vs**: Tensión de alimentación a la carga (motor)
- **Pin2 (INPUT 1), Pin6 (INPUT 2)**: con nivel alto se obtiene una tensión cercana a Vs el **pin 3 (OUTPUT 1)** y en el **pin 6 (OUTPUT 2)**, respectivamente.
- **Pin 10 (INPUT 3), Pin 15 (INPUT 2)**: con nivel alto se obtiene una tensión cercana a Vs el **pin 11 (OUTPUT 3)** y en el **pin 14 (OUTPUT 4)**, respectivamente.
- **Pin1 (ENABLE 1), Pin2 (ENABLE 2)**: cuando están a nivel bajo, las salidas OUTPUT 1 y 2 por un lado (asociadas a ENABLE 1) y las salidas OUTPUT 3 y 4 por otro lado (asociadas a ENABLE 2) permanecerán a nivel bajo, independientemente del nivel de tensión que tengan en ese momento las patillas INPUT correspondientes
- Las patillas de **GND** deben ir conectadas a masa para retorno de corriente y disipación de calor.

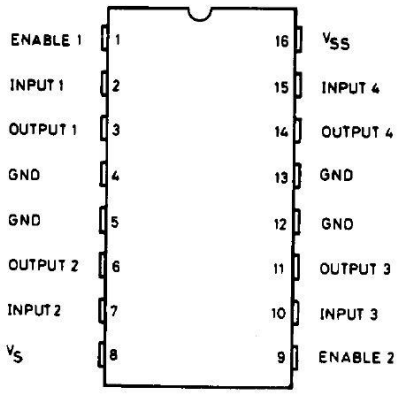


Figura 25. Patillaje del integrado L293D.

La elección de este puente H se debe a la facilidad de uso, de montaje y diseño de las pistas en la PCB al ser de orificio pasante y, sobre todo, por la extensísima cantidad de proyectos donde se incluye, siendo un circuito integrado muy probado.

De esta manera, y para aprovechar la inclusión de dos puentes H completos en él, se va a utilizar como dos puentes en paralelo uniendo las dos patillas de ENABLE, las señales de INPUT 1 y 4 , las de INPUT 2 y 3, las de OUTPUT 1 y 4 y las de OUTPUT 2 y 3, con el fin de que actúe como un único puente pero se reparta por todo el integrado la corriente y la temperatura, de manera que se aproveche completamente.

Como observación final de este circuito, y a diferencia de otros *drivers* de motor que tienen bombas de carga, las patillas OUTPUT experimentarán una caída de tensión cuando estén a nivel alto con un valor típico de 1,4V por debajo de la tensión de Vs, que para el caso que nos atañe es una ventaja, puesto que se busca disminuir la tensión de alimentación de la FetchHand.

3.2.6. Conectores

A continuación se explican los conectores que se utilizarán en el diseño y su justificación:

- Alimentación de la placa: conector Micro-Fit acodado de 4 pines en dos columnas. Se has seleccionado por ser un conector robusto que solo puede encajar de una manera.

- Conexión de la placa con la FetchHand: mismo conector que el de la alimentación de la placa.

- Cable de datos (señales CAN): conector Micro-MaTch de 10 pines. Se utilizará por compatibilidad, ya que la placa más cercana a la que se conectará para incorporarse en la red CAN tiene ese conector.

- Programación: conector MicroMaTch de 6 pines. Se utilizará en lugar del RJ-12 (conector análogo al RJ-11, que es el de las rosetas telefónicas, pero con seis cables en lugar de cuatro) por ser más pequeño que este.

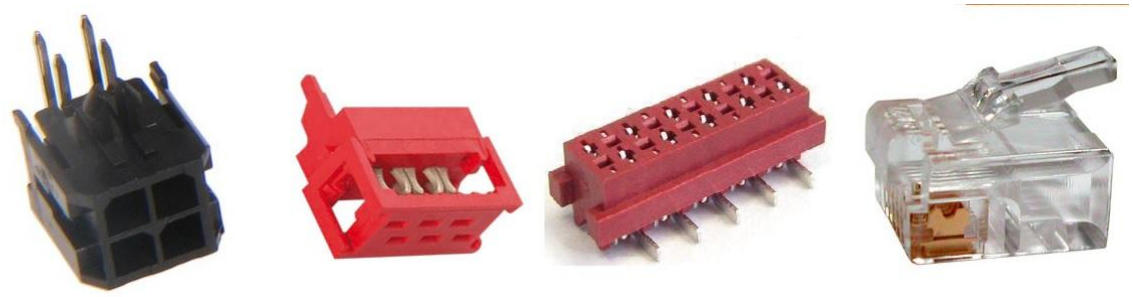


Figura 26.Conectores. De izquierda a derecha: Micro-Fit, Micro-Match (6 pines), Micro-MaTch (10 pines), RJ-12.

3.3. Diseño de circuito: esquemático

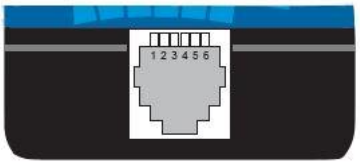
En este apartado se describen las consideraciones tenidas en cuenta en el momento de diseñar el circuito, en la parte del esquemático, que es la representación de las conexiones eléctricas que existen entre los componentes. Este esquemático más adelante se utilizará para producir un *layout*, que contiene la forma de la placa, las huellas donde se montarán los componentes y las pistas que los unen con la disposición con la que luego se fabricará.

Así, el esquemático se corresponde a un nivel más abstracto, que es necesario para generar la placa, y contiene información solo de las conexiones entre componentes, que luego se podrán realizar de una manera u otra en el *layout*, que se explicará en el siguiente apartado.

En líneas generales, se han conectado los componentes que se citaron anteriormente, añadiendo condensadores de desacoplo, tres diodos LED conectados al microcontrolador (por lo que se puede controlar su iluminación por software) como indicadores luminosos, la resistencia de 120 Ω con un *jumper* como terminación del bus CAN, una resistencia de *pull-down* con un *jumper* por si es necesario asegurarse de que la señal de habilitación del puente H se encuentre a nivel bajo y algunas resistencias para el circuito de programación y reset.

El circuito de programación está destinado a dar un correcto acceso a los pines necesarios para la programación (PGD, PGD y MCLR) sin necesidad de tener que desoldarlo de la placa, mediante el modo ya mencionado anteriormente ICSP. Para ello, se ha realizado un cable con un extremo RJ-12 para la conexión con el MPLAB ICD2 [17] [18] y, en el otro extremo, el conector Micro-MaTch que irá a la PCB,

siguiendo las indicaciones de la figura 27 y se ha realizado un esquema análogo al de la figura 28.



Pin	Signal
Not Used	1
PGC	2
PGD	3
Ground	4
VDD	5
MCLR/VPP	6

Modular Connector Pin	Microcontroller Pin
6	Not Used
5	PGC
4	PGD
3	Ground
2	VDD
1	MCLR/VPP

Figura 27. Conexión del cable del ICD2 en sus extremos.

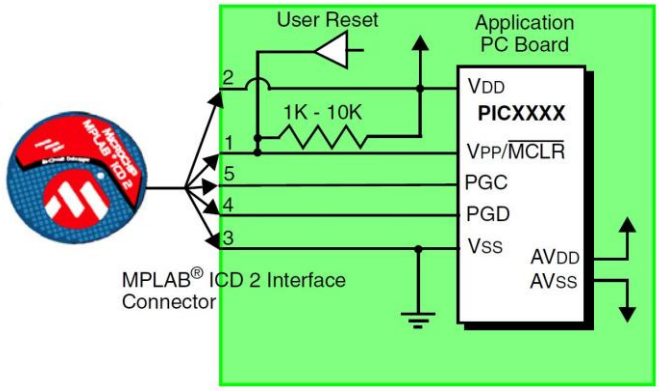


Figura 28. Circuito de conexión con el ICD2.

Además, se ha colocado un pequeño circuito de reset que incluye un pulsador, y se han tenido otras indicaciones del fabricante, como:

- No colocar resistencias de *pull-up* en PGC y PGD, ya que estas líneas poseen unas resistencias de *pull-dow*n de 4,7 kΩ dentro del depurador ICD2.
- No colocar condensadores en PGC y PGD, ya que estos evitarían rápidas transiciones en las señales de datos y de de reloj durante las operaciones de programación y depuración.
- No colocar condensadores en la señal MCLR que manda el programador, ya que estos evitarían transiciones rápidas a Vpp.
- No colocar diodos entre PGC y PGD, ya que provocarían que no hubiera comunicación bidireccional entre el dispositivo y el depurador.

Todo esto se ha incluido en el esquemático diseñado con la herramienta *OrCAD Capture*® del paquete *OrCAD 10.0*, el cual es una versión algo anticuada, pero que cumple perfectamente con las características exigidas y es la misma versión en unos casos, y en otros una versión compatible, con las versiones de *OrCAD* que se encuentran instaladas en la Universidad.

En la representación del circuito mediante el citado programa informático, se han utilizado las librerías de símbolos que incluye el mismo, especialmente la de conectores, pues si únicamente se necesita representar, y no simular, se pueden obtener fácilmente representaciones de circuitos integrados, aunque sea necesario modificar los símbolos (submenú del botón derecho del ratón). También se han incluido los símbolos que proporcionaba Microchip® [19] para el microcontrolador y el transceptor (ha sido necesaria la herramienta gratuita *Ultralibrarian Reader*, la cual es capaz de exportar unos ficheros genéricos descargados de la web del fabricante a muy diversos formatos compatibles con distintos programas de diseño electrónico, entre los cuales se encontraba *OrCAD*.

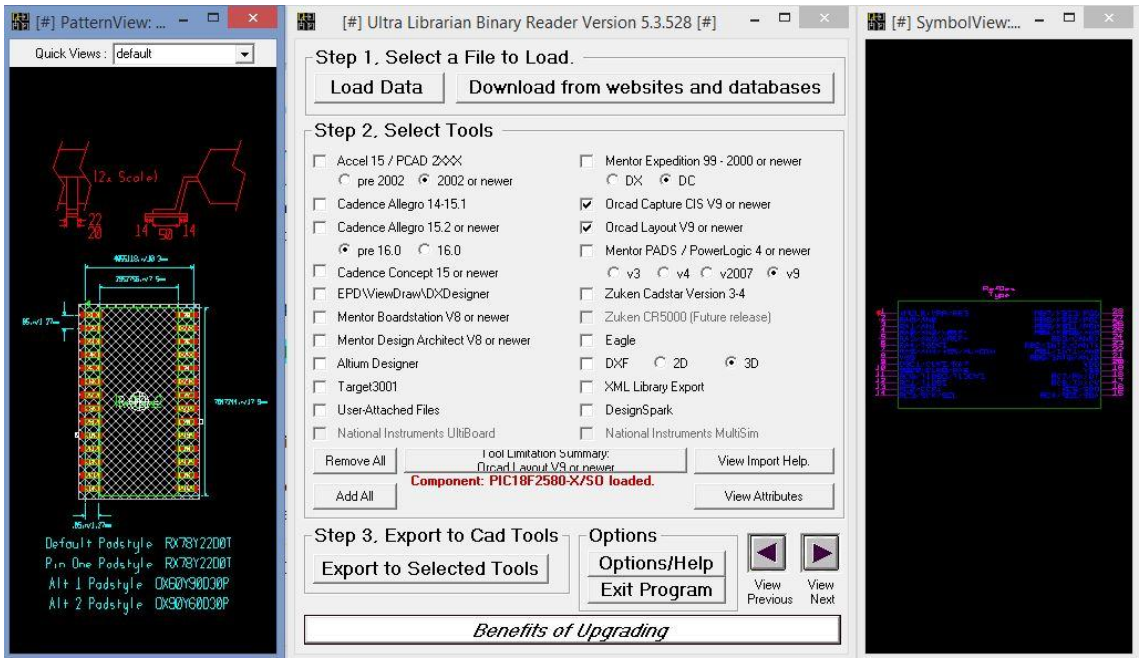


Figura 29. Captura de *Ultralibrarian Reader*.

También han sido muy utilizadas las etiquetas de red, una función que permite la conexión de dos cables con una etiqueta con el mismo nombre de etiqueta de red, sin aparecer conectados gráficamente por ningún hilo, pudiendo así tener una idea más clara y evitar las confusiones que producen los excesivos cruces y las zonas con alta densidad de cableado. Algunos ejemplos de señales para las que se han utilizado estas

etiquetas son: GND (masa del circuito), Vrobot (36 V de alimentación del robot TEO), V5 (alimentación de 5 V que proporciona el regulador instalado), etc.

Una vez representado el circuito, se puede asociar a cada componente su respectiva huella (se explicará qué es en el próximo apartado) y generar la *netlist*, archivo que contiene la información de las señales que circulan por las distintas conexiones entre componentes y cómo se encuentran estos conectados. Este archivo será necesario para la creación del *layout.*

Finalmente, el esquemático del circuito diseñado para el control de la FetchHand a través del bus CAN se puede ver en la figura 30.

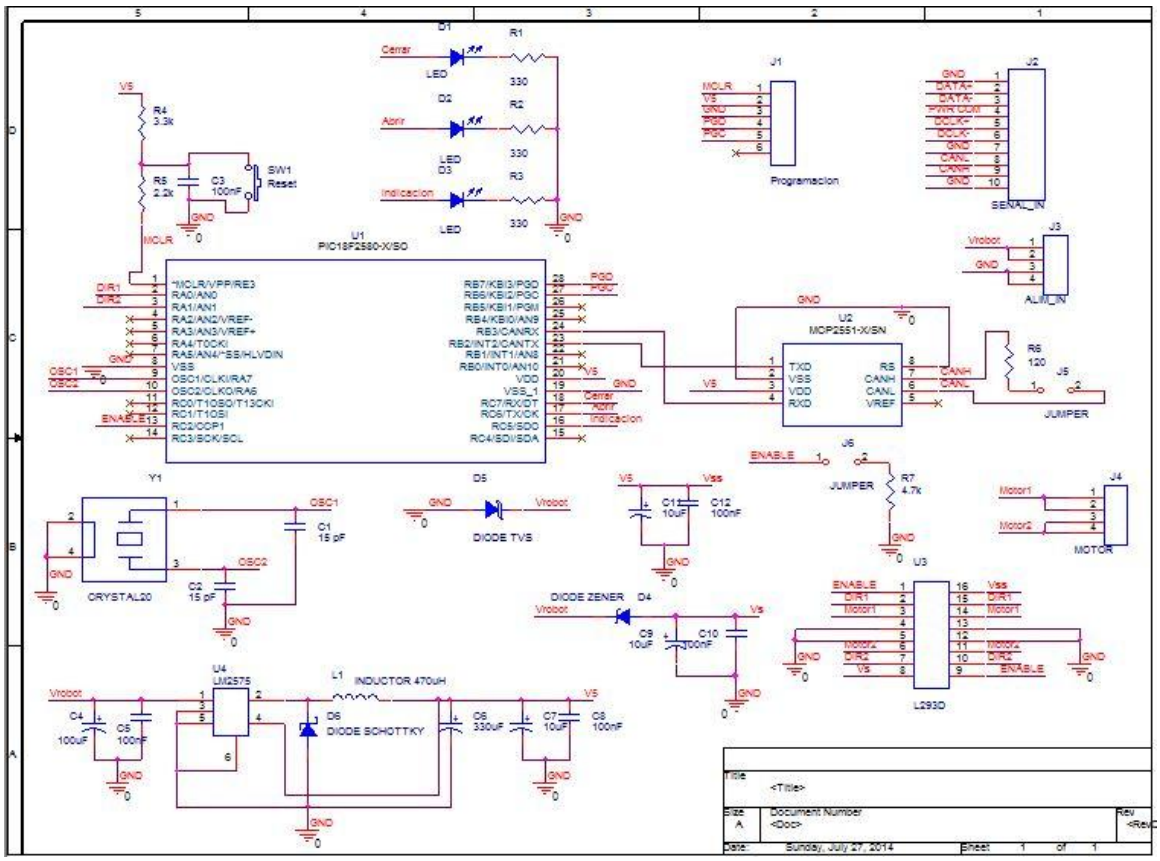


Figura 30. Esquemático del diseño.

3.4. Diseño de circuito: *layout* de PCB

Este apartado trata sobre la creación del *layout* de la placa [20] que contendrá las pistas y los componentes soldados, con las dimensiones exactas, y que es el documento que sirve de "plano" para finalmente fabricarla. Se introducirán algunas cuestiones generales sobre el diseño de PCB, así como una pequeña descripción del proceso seguido para esta en particular, utilizando la herramienta *OrCAD Layout Plus®*.

Para comenzar, se comentará la principal información que se recoge en el *layout*, que es:

- Tamaño y forma de la placa.

- Disposición de las huellas de los componentes, así como el tipo (orificio pasante o montaje superficial), la forma y tamaño de estas.

- Disposición y ancho de las distintas pistas, que conectan adecuadamente las huellas de los componentes, según el esquemático. Algunas pistas, como las de potencia, serán más anchas que las de señal. Incluso alguna, como podría ser la que corresponde a la señal GND, podría tratarse de un plano que, además, actuaría como disipador térmico para componentes que lo necesiten.

- Disposición de la máscara de soldadura (normalmente conocida por su término inglés, *solder mask*). Esta es la capa es un lacado que se aplica para proteger al circuito contra la corrosión o posibles cortocircuitos (tanto durante la soldadura como en fallos posteriores). Se puede ver en casi todos los circuitos y, aunque puede ser de varios colores, muchas veces es verde. Esta información suele estar incluida en las huellas de cada componente, e indica en qué lugares **no se debe colocar** esta máscara, como pueden ser los contactos eléctricos entre las patillas de los componentes y la huella.

- Serigrafía que permita la identificación de componentes o zonas, bien para el montaje o bien para el uso correcto del dispositivo. También podría incluir logotipos o nombres de empresas o instituciones que la han fabricado.

Toda esta información se encuentra en distintas capas, cada una con una función. De esta manera, hay una capa destinada a la serigrafía del lado de arriba, y otra para la del lado de abajo; otras dos para incluir las pistas y otras zonas de material conductor, como las que hacen contacto con los componentes al soldarlos, tanto en la capa de arriba (*Top*) como en la de abajo (*Bottom*); dos capas más para el *solder mask*, tanto arriba como abajo; otras capas con información para fabricación; otra con información de los taladros; otras si se la placa tiene más capas físicas, en lugar de utilizar solo una o los dos lados; etc.

Las huellas contienen la información relativa a cómo se une la placa y sus pistas al componente o, mejor dicho, al encapsulado del componente, y la demás información relativa a él, como pueden ser los taladros, la máscara de soldadura, la serigrafía, e incluso información sobre el ensamblado. Esto quiere decir que un mismo componente se puede presentar en varios encapsulado, como, por ejemplo, el microcontrolador

elegido, el cual se puede obtener de tres formas diferentes de 28 patillas: en formato de agujero pasante SPDIP, cuyas patillas se encuentran en dos de los lados más largos, son largas y se inserta en la placa, por lo que su huella contendrá taladros de diámetro y separación apropiada, en formato de montaje superficial SOIC, donde las patillas también se encuentran en dos lados a lo largo del componente, pero salen de manera que apoyan sobre la superficie donde se encuentra el componente, y otro encapsulado, el QFN, que en vez de ser alargado es cuadrado, más pequeño, los pines están en sus cuatro lados y son de más difícil acceso, pues el contacto está debajo del componente.

Por otra parte, componentes distintos, aunque tengan una funcionalidad dispar, pueden tener el mismo encapsulado, pudiendo reaprovechar huellas que se tengan hechas y utilizar las que se encuentren en librerías, como las que incluye el propio OrCAD.

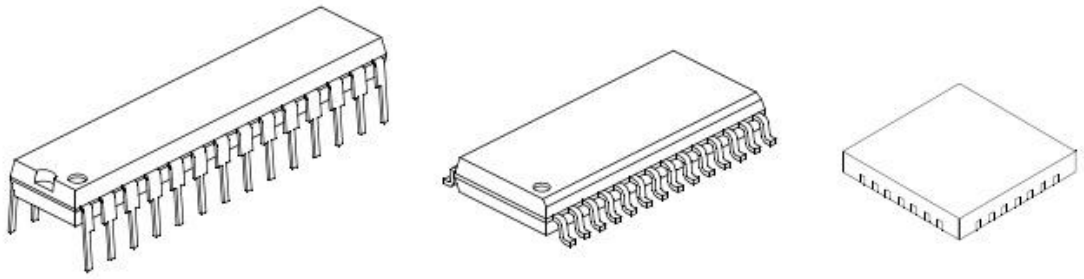


Figura 31. Representación de los distintos encapsulados del PIC18F2580

Cada componente en OrCAD tiene asociada una huella, según se ha hecho en la parte del diseño correspondiente a la representación del esquemático, y se ha incluido en la *netlist*, aunque esta huella se puede cambiar más adelante en *OrCAD Layout Plus* o regenerando la *netlist* en *OrCAD Capture* por si se debiese actualizar debido a algún error.

Tanto en el diseño o modificación de una huella es posible incluir lo que se denominan obstáculos, como puede ser el borde de un componente, el borde de la placa, un plano en la capa *Top* que se asocie a la señal de masa, una pequeña área de cobre para modificar el contacto de una zona específica y otros más, que posibilitan el correcto diseño de la huella y la placa.

Además de la herramienta de creación y edición de obstáculos, existe otra relativa a los *pads*, es decir, al contacto entre los pines del componente y la PCB. Aquí es donde se definen todos los parámetros de este, como son el área de cobre necesaria para un correcto contacto entre la PCB y el componente y que se pueda soldar aceptablemente

bien, la forma del mismo (rectangular, cuadrado, ovalado, circular...), el taladro (si es de orificio pasante), cómo es el *solder mask* (forma y área), si existe toda esta información en un lado de la PCB o en los dos (en los componentes de orificio pasante se definen estos parámetros para los lados de arriba y abajo de la placa), y otros parámetros menos utilizados.

Teniendo estas nociones básicas se puede explicar, de forma resumida y a grandes rasgos, cómo ha sido el proceso seguido para el diseño de la PCB en *Orcad Layout Plus*, y que se encuentra en las siguientes líneas.

En primer lugar, se han importado las huellas de que proporcionó Microchip para el microcontrolador y el transceptor desde *Ultralibrarian Reader* a una librería nueva en *Layout Plus* (opción de importar .min creando un .llb). Dentro de la opción *Library Manager*, donde se pueden ver y editar tanto las librerías como las huellas que contienen, se importa la librería que se acaba de crear. Para el resto de componentes ha sido necesario verificar cuáles de las huellas que incluye OrCAD eran válidas, identificarlas, y asignarlas en *OrCAD Capture*, dentro de propiedades de componente.

Las huellas que no eran aplicables directamente se han modificado para su correcta aplicación. Para ello, se escoge una huella como base, cuanto más parecida a la deseada mejor (por ejemplo, si se requiere una huella para montaje superficial se escogerá de este tipo, o se procurará que la huella escogida tenga un número y disposición de pines parecido al deseado, siendo una magnitud muy importante la separación entre estos). En las modificaciones se cambian los obstáculos relativos al borde del componente, el borde en la serigrafía y en la capa de ensamblado, el tamaño y forma de propiedades del cobre, taladros máscaras de soldadura de los *pads* y número de estos, y la disposición de dichos *pads* para que se ajuste a la descripción del encapsulado que facilita el fabricante (las dimensiones pueden ser dadas en milímetros o, normalmente, en unidades anglosajonas, como son los mils, que corresponden a la milésima parte de una pulgada, y que es la unidad más utilizada en elctrónica).

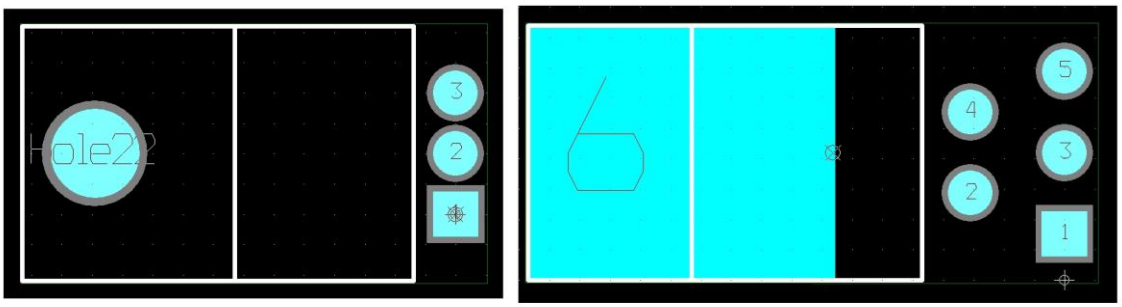


Figura 32. Muestra de huella modificada. A la izquierda, la huella del TO-220 de las librerías de OrCAD. A la derecha, la huella modificada (el orificio del tornillo se encuentra en una capa no visible en *Library Manager*).

Una vez asociados los componentes en el esquemático y generada la *netlist* (en las opciones de generación se marca *run ECO* y las medidas en pulgadas), se procede al diseño de la PCB. Para ello, se crea un nuevo diseño, introduciendo en los archivos que solicita el asistente la *_default.tch* para el primer campo, que es la plantilla estándar que se utilizará para medidas en pulgadas, y el archivo con extensión *.mnl* que corresponde a la *netlist*, cambiándole el nombre al tercer campo, que se ha rellenado automáticamente, y que es nuestro archivo de trabajo con extensión *.max* (por ejemplo, *placaNueva.max*). Se debe dejar la opción de *AUTOECO* activada.

Figura 33. Menú de creación de una placa nueva.

Por otra parte, como el primer paso una vez que se tienen todas las huellas es la definición del borde de la placa, y para este diseño se ha considerado que lo mejor es

que se adapte al borde del lado de la mano donde se va a colocar, se ha importado la geometría de dicho lado. Este proceso ha consistido en la generación de un plano con un software de diseño mecánico 3D, como Catia o SolidEdge, a partir del archivo con la información 3D con extensión *.step* que muy amablemente facilitó el fabricante, Lacquey, y su posterior exportación a un archivo *.DXF*.

Este archivo se puede importar desde *OrCAD Layout Plus* desde *File>Import>DXF to Layout*. Aquí se selecciona el archivo a importar, el nombre y ruta del archivo de salida con formato *.max*, la ubicación del fichero *maxdxf.ini*, el cual debe ser editado como se indica en el propio fichero para configurar la conversión de las unidades del archivo a importar (mm) a las de la placa donde se utilizará (pulgadas), escribiendo un valor numérico de factor de conversión que indica dicho fichero (para este caso *UNITS_DIVISOR=25.4*). El último campo se deja por defecto, con el *default.tch* de plantilla para pulgadas y se hace clic en *Translate*.

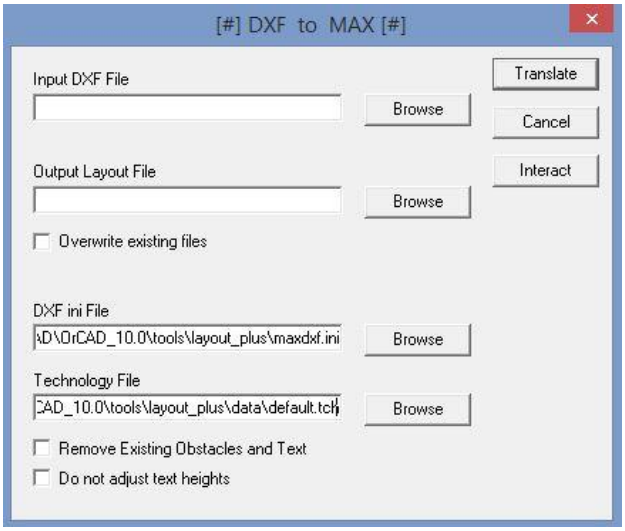


Figura 34. Menú de importación de un archivo .DXF.

Una vez realizada la importación a un formato de *OrCAD*, se abre de nuevo el fichero *.max* de la PCB, y en el menú *File>Load...* se escoge la opción de *Merge board (*.max)*. Siguiendo este proceso se ha conseguido importar la geometría como un componente (ver figura 35), pero para convertir el perímetro de este a un obstáculo de tipo de borde de placa (*Board outline*) no se ha encontrado ninguna opción a tal efecto, aunque sí se ha encontrado para versiones más modernas [21]. Así, la mejor opción para ello ha sido dibujar, con el borde importado como guía, el obstáculo de borde de placa (también se ha seguido este proceso para crear el borde de los planos de masa de la capa de arriba y abajo, solo que dejando una pequeña distancia con la línea del borde exterior).

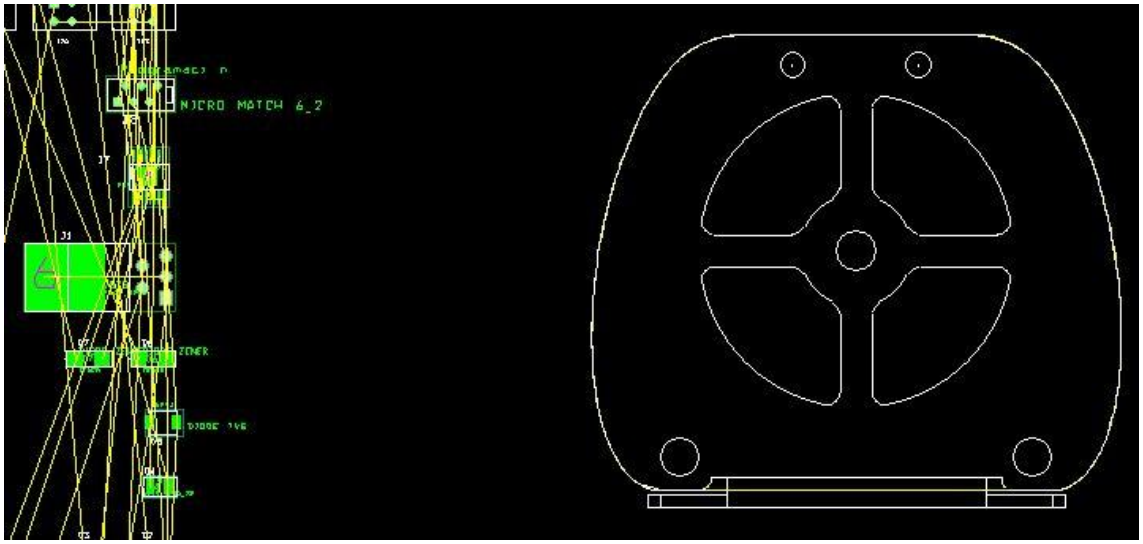


Figura 35. Geometría de la mano importada para trazar el borde.

Se aprovecha que está todavía el dibujo importado para colocar unos componentes que representan los taladros de la placa, por donde se introducirán los pasadores que la sujetarán a la mano de la siguiente forma: con la *Component Tool* seleccionada, se crea un nuevo componente en *Tool>Componente>New...* y se le asigna la huella del taladro (creada previamente en *Library Manager*). Se aconseja, una vez ubicado este taladro en su lugar correcto, marcar las opciones de *Fixed, Locked, Do not rename, Not in netlist* y desmarcar *Route enabled* para asegurar que no se realiza contacto con pistas, que la posición no se puede cambiar y que no desaparezca el taladro si se regenera la *netlist*.

Cuando se ha realizado esto ya es posible eliminar el componente importado que representa el plano del lado de la mano, de manera que sea más fácil visualmente colocar los componentes.

En la colocación de los componentes se han tenido en cuenta las siguientes consideraciones:

- Agrupación de los componentes principales con los que necesita para funcionar correctamente como, por ejemplo, condensadores de desacoplo, bobinas, diodos, oscilador, etc.
- Disposición de estos grupos de manera que queden lo suficientemente cerca de los grupos con los que se deben conectar, aunque no demasiado para no concentrar calor en exceso, que no afecte demasiado el ruido de otros componentes (por ejemplo, el transceptor) y se pueda soldar bien.

- Se deben colocar los componentes de manera que el ruteado de las pistas sea posible y cuando más directo mejor, colocando componentes en fila cuando si se puede (por ejemplo, se puede observar estos en la disposición de varios condensadores).
- Los conectores deben estar cerca del borde de la placa.

En el ruteado de las pistas se ha realizado con estas premisas:

- Se deben evitar giros de 90º en las pistas para disminuir el ruido eléctrico.
- Las pistas de potencia deben ser más anchas que las de señal.
- Se debe intentar que las pistas salgan o lleguen en dirección radial a los *pads* circulares.
- No deben quedar zonas con necesidad de señal de masa aisladas (para diseño con planos de masa).

Una vez trazadas correctamente todas las pistas, se dibujan los contornos de los planos de masa (se ha dispuesto uno en la capa de arriba y otro en la de abajo) y se asocia con esa señal en las propiedades del mismo. Como una de las patillas de GND del transceptor quedaba aislada, y no era solucionable cambiando la orientación del componente, se ha insertado una vía, que es un componente que permite la conexión entre dos capas mediante un taladro metalizado, análogo a los que se encuentran en las huellas de los componentes de orificio pasante.

También se ha ajustado el *thermal relief*, que es la unión entre un *pad* que se tiene que conectar a masa con el plano de masa, para que exista un buen contacto pero sea fácil soldar (si el contacto fuese completo, al estar conectado un plano, la excelente disipación térmica haría que se necesitase mucho calor para soldar).

Para terminar, se añadió un obstáculo de tipo área de cobre para mejorar el contacto entre la parte disipadora del regulador DC/DC para eliminar el comentado *thermal relief*, pues en este caso no existe soldadura (el componente y la placa disponen de un taladro para mantener el circuito integrado en contacto).

Así, a continuación (figura 36), se puede apreciar la vista global del *layout* final:

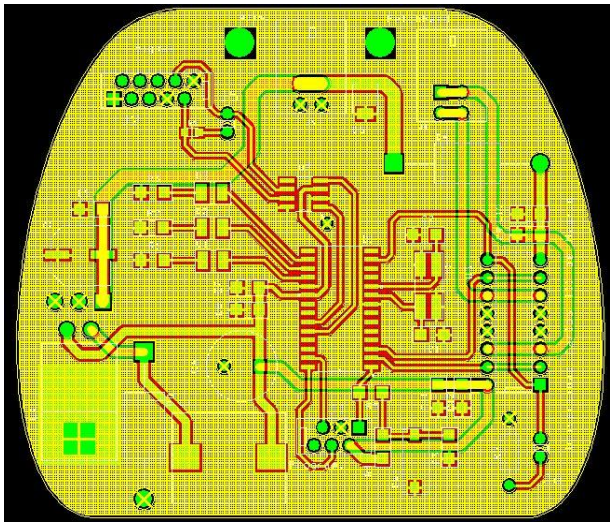


Figura 36. Vista de la capa global de la PCB creada.

3.5. Prueba de convertidor DC/DC (LM2575)

En este apartado se resumirán las pruebas de verificación realizadas para corroborar el correcto funcionamiento del circuito de alimentación, que deberá proveer 5 V a su salida a partir de los 36 V que le llegan de la alimentación general del robot, y que se corresponde con el circuito integrado LM2575 y sus componentes necesarios para operar, citados en el apartado "Selección de componentes" (apartado 3.2.3.).

A tal efecto, se ha realizado el montaje de este circuito en una *protoboard* (placa de pruebas con orificios para insertar los componentes, donde los orificios de una misma columna están conectados entre sí), y determinar si la tensión de salida es estable para distintas demandas de corriente.

Primeramente, se ha comprobado el que el condensador de la salida del circuito sea de un valor adecuado, asegurando que la onda de 5 V sea lo más estable posible. Para ello, se probó el circuito con distintas resistencias de potencia, con lo que se determinó que un condensador a la salida de 330 µF suministraba una tensión muy estable, cuyo rizado no se podía distinguir del ruido del osciloscopio, mientras que no ocurría lo mismo con un condensador de 100 µF o dos en paralelo (equivalentes a uno de 200 µF), que hacían que la tensión fluctuase ligeramente. Por tanto, el condensador finalmente elegido ha sido el de 330 µF, y el correcto funcionamiento ha quedado verificado.

Es interesante observar que este circuito, al ser un reductor de tensión y conservar la misma potencia a la entrada y a la salida del mismo (exceptuando pérdidas), necesita consumir poca corriente aunque a la salida se le demande más, como se puede ver en la siguiente tabla:

Carga aplicada(Ω)	Consumo carga (A)	Consumo del integrado (A)
25,6	0,195	0,04
40,3	0,124	0,03
5,6	0,89	0,02

Tabla 5. Consumo del LM2575 según la demanda de corriente de la carga.

3.6. Prueba del puente H descartado (A3950)

En las primeras etapas del diseño se consideró la utilización del circuito integrado A3950 [22] de Allegro MicroSystems, principalmente por soportar hasta 36 V, tener una entrada de habilitación para poder aplicar PWM en caso de que fuese necesario y ser un componente de montaje superficial de una tamaño muy pequeño.

Otras características de este componente son:

- Protección contra sobretemperatura
- Protecciones contra cortocircuitos
- Protección contra sobrecorriente
- Salida para monitorizar si existe un fallo
- Una única entrada de alimentación, tanto para potencia como para la lógica
- Patilla para controlar la entrada del circuito a un modo de bajo consumo (*Sleep*)
- Una patilla para controlar sentido de giro y otra para la selección de modo *slow decay* o *fast decay*, que están relacionados con un frenado rápido o que permita al motor girar libremente hasta que se pare, respectivamente.

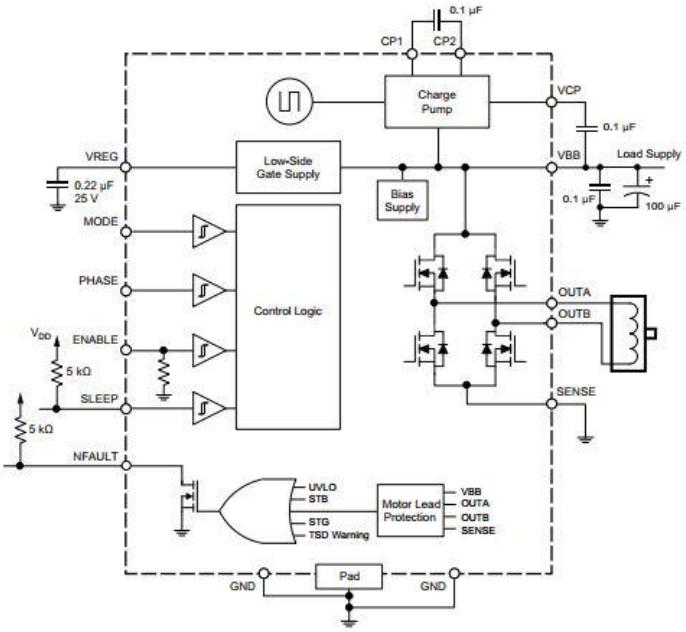


Figura 37. Diagrama de bloques del A3950

El montaje que interesaba para este trabajo no precisaba de una resistencia para medir la corriente, por lo que la patilla correspondiente se podía conectar directamente a masa y, respecto al control, se mantendría la patilla *MODE* a nivel bajo, se seleccionaría el sentido de giro mediante el pin *PHASE*, se habilitaría el dispositivo con la patilla *ENABLE* y se monitorizarían los fallos mediante el pin NFAULT, salida que es de tipo drenador abierto, por lo que necesita de una resistencia de *pull-up* para definir bien los estados alto y bajo (el dispositivo indica con un nivel bajo un fallo cualquier tipo de fallo detectado y, al ser drenador abierto, deja como indeterminado la indicación de correcto funcionamiento, por lo que es necesaria la resistencia *pull-up*, que fija esa indeterminación como un nivel alto) y cuyo estado se podría reflejar mediante un LED. También se controlaría la patilla de *SLEEP* para disminuir el consumo cuando no se estuviese aportando energía a la mano, pudiendo poner la patilla a nivel bajo (modo de bajo consumo activo) cuando el tiempo de inactividad fuese mayor que el prefijado. El patillaje y la función de cada pin se encuentra en la tabla 6.

Name	Number		Description
	EU	LP	
NFAULT	15	1	Fault output, open drain
MODE	16	2	Logic input
PHASE	1	3	Logic input for direction control
GND	2, 12	4,13	Ground
SLEEP	3	5	Logic input
ENABLE	4	6	Logic input
OUTA	6	7	DMOS full-bridge output A
SENSE	7	8	Power return
VBB	8	9	Load supply voltage
OUTB	9	10	DMOS full-bridge output B
CP1	10	11	Charge pump capacitor terminal
CP2	11	12	Charge pump capacitor terminal
VCP	13	14	Reservoir capacitor terminal
VREG	14	15	Regulator decoupling terminal
NC	5	16	No connection
Pad	–	–	Exposed pad for thermal dissipation connect to GND pins

Tabla 6. Patillaje del A3950. El encapsulado utilizado es el LP.

Para las pruebas del componente, al existir solo en formato de soldadura superficial, ha sido necesario diseñar una pequeña placa adaptadora, la cual permitiese la inserción del circuito integrado A3950 en la placa de puntos de la figura 38 (placa de desarrollo que, a diferencia de la *protoboard*, requiere soldadura y permite unas conexiones más robustas que esta, siendo más cercana a un diseño final) para facilitar el conexionado con los componentes auxiliares (casi todos condensadores, como se pudo observar más arriba, en la figura 37, más otros de desacoplo de la alimentación) y su integración un microcontrolador para controlarlo. Al tratarse de una prueba rápida y no disponer aún del PIC18F2580, además de por facilidad de programación al ser un entorno sencillo y estar más familiarizado con él y ser muy fácil la comunicación

con el ordenador se ha optado por utilizar una de las conocidas placas Arduino, concretamente el modelo Leonardo, del cual ya se disponía.

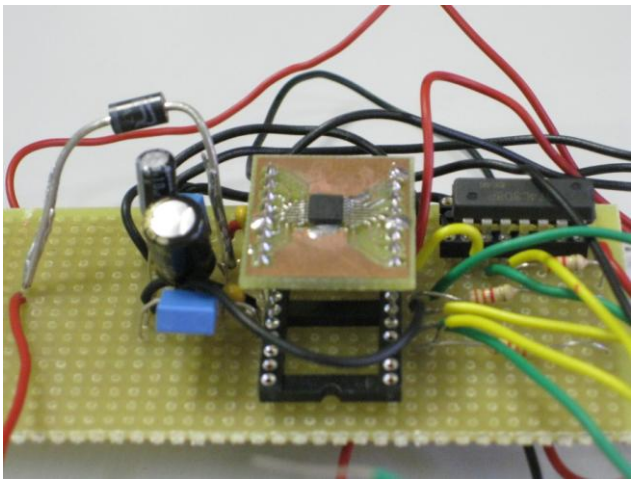


Figura 38. Placa adaptadora del A3950 montada en el circuito de prueba.

Arduino, como se ha explicado en el capítulo de "Estado del arte", sección 2.3.1, es una serie de pequeñas placas de hardware libre que incluyen un microcontrolador, entrada de alimentación externa, conexión USB, cabezales para conectar directamente los cables a los pines y otras funcionalidades, que son muy útiles para realizar proyectos de manera fácil, como prototipos rápidos, y con una gran documentación debido al extenso uso del mismo, incluso para pequeños proyectos de aficionados, que además tiene un bajo precio y un entorno de desarrollo de software (IDE) propio muy ligero y sencillo de utilizar.

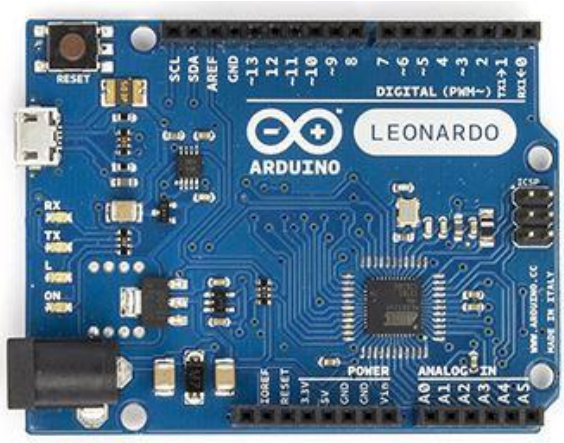


Figura 39. Vista frontal de la placa Arduino Leonardo

Por otra parte, la elección de esta placa para las pruebas permite la programación del dispositivo como un banco de pruebas, de manera que se ha diseñado el programa para ser controlada por USB mediante el ordenador, con los botones del mismo, al

estilo de "coche teledirigido", para mandar fácil y rápidamente al sistema las acciones deseadas.

Para poder comandar la placa remotamente de la forma descrita, utilizando las teclas de un ordenador, se ha utilizado Processing, que es otro entorno de desarrollo multiplataforma y un lenguaje de programación, por poseer las siguientes funcionalidades:

- Lenguaje de programación prácticamente idéntico al utilizado en Arduino (la plataforma Arduino está basada en Processing)
- Permite detectar fácilmente pulsalciones de teclas en un ordenador, muy útil para desarrollar rápidamente el programa de control mediante teclas.
- Comunicación serie con Arduino fácil de utilizar y robusta, con capacidad para que, por ejemplo, una acción se ejecute solo mientras se esté apretando una tecla sin que existan problemas en el buffer de recepción de Arduino, cargando en la placa el programa que se incluye en el entorno de Arduino como ejemplo: *StandardFirmata*.
- Al estar muy enfocado en representaciones visuales, posibilidad de implementar un pequeño panel de control donde se recoja el estado de la placa Arduino.

Los comandos remotos de Arduino se pueden insertar en el mismo código del ordenador, con una sintaxis similar a la del propio Arduino al utilizar la librería de Arduino para Processing [23], y la comunicación serie, la generación de formas y distintas geometrías y los rellenos de color de estas con las funciones básicas de Processing.

De esta manera, cargando en Arduino el ejemplo citado *StandardFirmata* (dentro de Archivo>Ejemplos>Firmata), y programando rápidamente en Processing un pequeño código que represente un panel de control y monitorice los pines de Arduino o los controle según las teclas pulsadas en el ordenador, se obtuvo la interfaz que se puede ver en la figura 40 y la figura 41 y las siguientes funciones:

- Control de avance del motor en un sentido u otro, según se esté pulsando la tecla 1 o 2. Si no se está pulsando ninguna el motor debe permanecer parado.
- Visualización en pantalla del sentido de giro en el panel de control.
- Control de avance en un sentido de giro aplicando PWM con las teclas 8 y 9. Control del valor del PWM aplicado con las teclas + y - .

- Visualización en pantalla del valor de PWM y si se está aplicando para algún sentido de giro.
- Visualización por pantalla del estado *Sleep* del A3950.
- Visualización por pantalla del estado de la patilla del A3950 correspondiente a *NFAULT* para diagnosticar problemas.

Figura 40. Control de motor con el A3950 mediante Processing. Motor parado, en *Sleep* y sin fallo.

Figura 41. Control de motor con el A3950 mediante Processing. Motor en marcha, fallo activado, modo normal.

En la prueba realizada, para proteger la FetchHand de posibles contratiempos y garantizar un posterior funcionamiento correcto, se utilizó un pequeño motor de corriente continua del que se disponía con un consumo de 220 mA para 24 V y 190 mA para 12 V, magnitudes mayores que en la FetchHand, y que sirven para tener la certeza de que funcionará con el diseño final al estar el sistema sobredimensionado.

Estas pruebas han revelado algunos problemas al intentar controlar el motor, que provocaban reinicios en Arduino o incluso malfuncionamiento del integrado en algunos casos de mayor exigencia (se cortocircuitaba el pin CP1 con masa), además de que en el periodo de activación de Arduino, que es bastante largo, la alta impedancia del pin es interpretada como nivel alto, con lo que se activa el motor.

También se ha programado directamente en Arduino una secuencia que correspondía al giro en un sentido durante unos segundos, frenado, giro en el otro sentido y parada lenta. Aunque cabe destacar que el modo de frenado rápido consigue una parada del motor en el acto, después de varios ciclos del código programado el controlador de motor provocaba un reinicio en Arduino, por lo que se ha decidido probar un circuito integrado con un uso mucho más extendido, como el L293D.

3.7. Prueba del puente H finalmente introducido (L293D)

Este componente se ha amontado sobre una *protoboard* y, primeramente, se ha programado un ejemplo en Arduino equivalente al hecho con el A3950, pero adaptándolo a la forma de control de este circuito (el sentido de giro se selecciona mediante dos patillas, y el frenado más rápido se puede indicar controlando las dos ramas del puente a nivel alto. En este caso no se ha observado ningún problema de reinicio, a pesar de que se han realizado más de treinta ciclos.

También se ha adaptado el programa de Processing para controlar el motor a través del ordenador, recortando algunas funcionalidades, como el control del modo *Sleep*, que en este circuito no existe, y la monitorización de fallo, que tampoco implementa (ver figura 42). De esta manera, se ha podido controlar a voluntad el motor mediante a través del ordenador de manera satisfactoria.

Así, una vez comprobado el funcionamiento correcto con el motor de prueba, se ha conectado la FetchHand a 24V ,y después de verificar el funcionamiento correcto con el programa de Processing, se ha realizado la misma prueba en conjunción con el diodo zéner para rebajar tensión conectando la alimentación a 36 V, según se describe en el apartado siguiente.

Los resultados han sido satisfactorios, y se ha observado que el control por PWM, si bien queda fuera del propósito inicial del fabricante de la FetchHand, podría servir para controlar la fuerza del agarre o disminuir en alrededor de 20 mA el consumo de la mano, aunque su modo de control principal sería mediante señales constantes.

De esta manera, se ha seleccionado el L293D para controlar la FetchHand por:

- Mayor fiabilidad en las pruebas que el A3950, que es la razón principal
- Uso mucho más extendido
- Control más sencillo
- Mayor facilidad de montaje, con posibilidad de montarse en un zócalo por si es necesario cambiarlo (el A3950 tiene un *pad* de disipación térmica en su base, más difícil de soldar con herramientas básicas)
- Posibilidad de unir sus cuatro medios puentes dos a dos para obtener un equivalente a dos puentes H en paralelo, introduciendo un nivel de exigencia aún más bajo en el componente
- La señal de habilitación permanece a nivel bajo durante el arranque de Arduino (pin en alta impedancia)

Figura 42. Control de motor con el L293D mediante Processing.

3.8. Prueba del diodo zéner para rebajar tensión

Se ha incorporado este diodo al montaje de la prueba del L293D y se ha comprobado, primero alimentando a 26 V (tensión dentro del rango de admisión de la FetchHand, por si el diodo no funcionaba correctamente) para alimentar al puente con 16 V y, una vez que se ha verificado un funcionamiento correcto, se ha ascendido hasta 36,1 V, de manera que se han medido 26,1 V a la entrada del puente H cuando este suministraba potencia al motor.

Se ha observado que, cuando el puente no está trabajando (señal de habilitación a nivel bajo), la tensión a la salida del zéner (entrada del puente) puede ascender hasta 29,7 V, aunque este hecho no es preocupante, pues está dentro de los rangos admisibles y la transición entre el estado de apagado del diodo y el funcionamiento en

zona zéner no introduce ninguna sobretensión en forma de pico a la entrada de la mano.

3.9. Prueba del microcontrolador

Para esta prueba, y sucesivas con el sistema completo, se ha montado el bloque de control y comunicación (microcontrolador, LEDs de estado, transceptor, resistencia terminal del bus CAN y circuito de programación) sobre una placa de puntos, aunque en este caso se ha colocado el conector RJ-11 de 6 pines para la programación en lugar del Micro-MaTch, al no existir limitaciones de espacio y ser más sencillo de montar, y cables conectados con clemas al circuito soldados según la tabla 7 a un DB-9 para probarlo conectándolo a la red por este otro tipo de conexión.

Pin del conector	Función
2	CAN L
3	GND
7	CAN H

Tabla 7. Conexionado de señales CAN a conector DB-9.

Se han realizado también las conexiones con el circuito de alimentación y el puente H, tal y como se observa en la figura 43, y se ha comprobado el funcionamiento del microcontrolador cargando un pequeño programa de parpadeo de uno de los LEDs instalados cada segundo, con el fin de verificar la posibilidad de transferir programas correctamente y que el reloj estuviese bien configurado (el parpadeo cada segundo está configurado para un reloj de 20 MHz, como el instalado, y si no fuese correcto se obtendría un parpadeo diferente al programado).

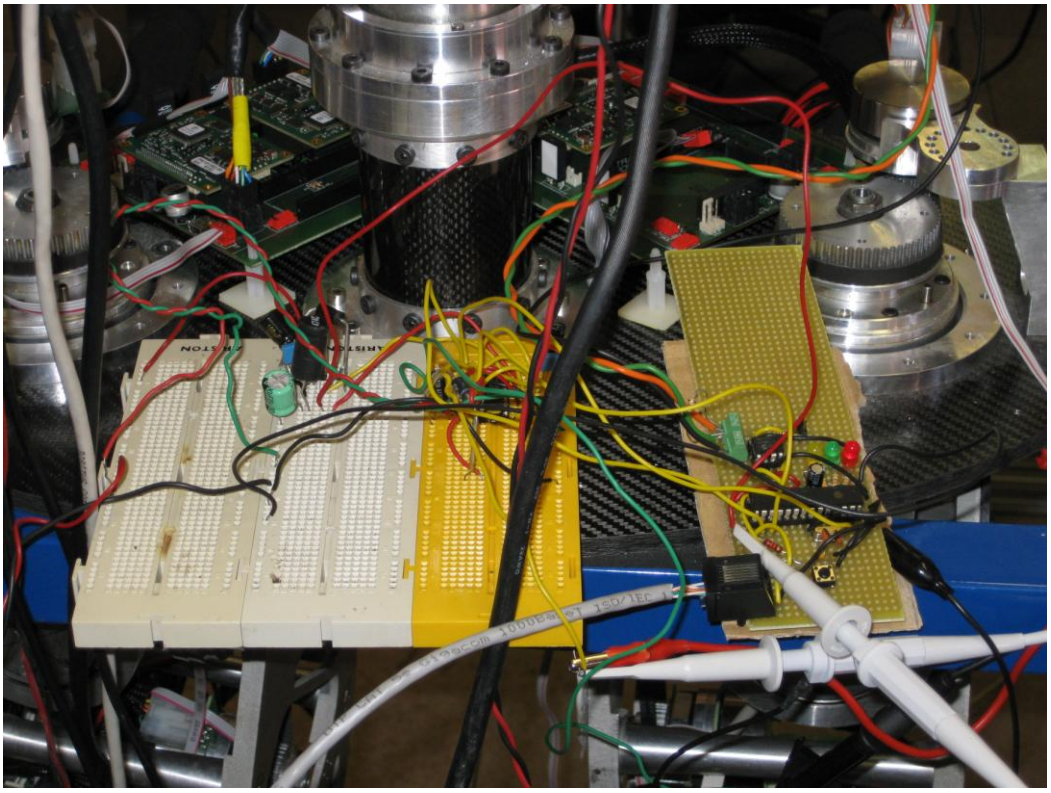


Figura 43. Montaje de sistema de pruebas provisional. La placa de puntos de la derecha corresponde al bloque de control y comunicación, la *protoboard* amarilla al puente H diodo zéner y la blanca a su izquierda es contiene el regulador a 5 V.

El resto de pruebas del sistema completo, en especial las referidas al CAN, están directamente relacionadas con el software, tanto del PIC como del ordenador que envía las órdenes, por lo que estarán contenidas en el siguiente capítulo, referido a la programación en ambos sistemas.

4. Software desarrollado y pruebas del sistema completo

En este capítulo se describe el desarrollo del software en el lado del microcontorlador PIC y en el lado del PC, especialmente buscando una buena integración con el sistema de control de otros elementos de la red preparándolo para poder implementarse en YARP, las pruebas realizadas y las correcciones hechas para un funcionamiento correcto del dispositivo y de la comunicación.

4.1. Programación del PIC18F2580

Como se comentó en el apartado del capítulo anterior "Selección de componentes", se ha utilizado el entorno de desarrollo MPLAB® IDE, en su versión 8.92 (la última antes de MPLAB® X IDE, no compatible con el depurador ICD2). Se instaló también el compilador C18 para el lenguaje C, herramienta proporcionada también por Microchip, y la *application note* AN878 [24] de la misma empresa.

Las notas de aplicación de Microchip pueden contener información sobre tecnologías que aplican a sus dispositivos (por ejemplo, diferencias entre un módulo incorporado en antiguos microcontroladores y el nuevo), sobre temas más generales (por ejemplo, descripción del bus CAN), o librerías descargables para incorporar a un proyecto y notas sobre su uso, como es el caso de esta AN878.

La **AN878** provee una serie de funciones para el manejo del módulo ECAN del microcontrolador, con muchas opciones de configuración en tiempo de compilación. Consta de tres archivos de código: **ECAN.def**, donde están las citadas opciones de configuración, **ECAN.h**, que contiene las cabeceras de las funciones que componen la librería y **ECAN.c**, archivo donde se encuentra el código de dichas funciones.

Lo más importante respecto a la configuración, y que debe editarse en el archivo .def, es la configuración de la tasa de trasferencia. Para la red de TEO la tasa es de 1 Mbit/s, y es necesario ajustar varios parámetros, dependientes de esta tasa y del reloj del microcontrolador (20 MHz), que corresponden con la duración de cuatro intervalos en términos de una unidad denominada *Time Quanta* (TQ) y otros parámetros de corrección. La duración de los cuatro segmentos se denomina tiempo de bit nominal, y debe estar compuesto por un número de TQ de entre 8 y 25.

Los parámetros a configurar son los siguientes:

- **Synchronization Segment** (**sync**):segmento que se utiliza para configurar los distintos nodos del bus con los flancos de bajada de la trama. Su duración es de un TQ.
- **Propagation Time Segment** (**prop**): segmento que sirve para compensar los retrasos por la propagación a través de la red. Su valor es programable entre 1 y 8 TQ.
- **Phase Buffer Segment 1**(**phase1**): segmento que se encuentra a la izquierda del punto de muestreo y compensa errores. Se puede alargar momentáneamente para resincronizar Su duración es de 1 a 8 TQ.
- **Phase Buffer Segment 2**(**phase2**): segmento que está a la derecha del punto de muestreo y se puede acortar momentáneamente para resincronizar. Su valor mínimo es 2 TQ y su máximo 8 TQ, pues debe dejar un tiempo para el procesado de información.
- **Synchronization Jump Width** (**SJW**): parámetro de corrección para ajustar la sincronización de la Phase 1 y 2. Su duración puede ser de 1 a 4 TQ.
- **Baud Rate Preescaler** (**BRP**): preescalado con el que se obtiene el TQ. Su valor mínimo es 1, e indica que el TQ se corresponde con el doble del periodo del oscilador.

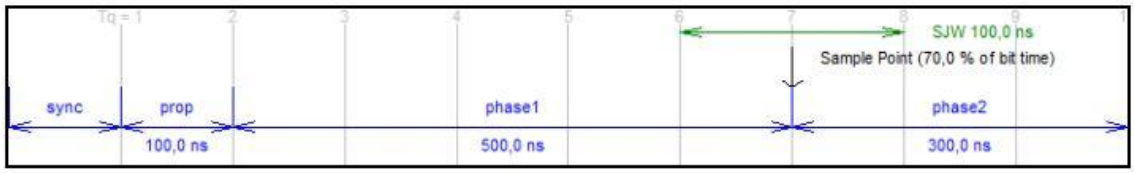


Figura 44. Representación de la distribución de tiempos configurada en el bit nominal.

Utilizando la herramienta gratuita *MBTime*, que genera un informe con una solución para los datos de reloj y tasa de transferencia deseada, se han calculado los parámetros citados y se han representado en la figura 44 los parámetros obtenidos, cuyos valores son:

- BRP=1
- Prop=1
- Phase1=5
- Phase2=3
- SJW=1

También se indicará en el .def el modo en el que se dejará el módulo ECAN tras la configuración, que será el normal, y el modo de funcionamiento, que será el *Legacy*

(utilización solo de dos buffers, pues incluso solo sería necesario uno). El ajuste de los filtros y máscaras, si bien se podría hacer aquí, se hará mediante una función que no es de esta librería, pues la de la AN878 contiene errores, y que además se utilizará para que solo se acepte una ID, como se explicará más adelante. Las funciones de configuración que se acaban de describir se ejecutarían en el código con la rutina *ECANInitialize()*.

De esta librería también se utilizará la función *ECANReceiveMessage(&IDOrden, Orden, &OrdenLon, &OrdenFlags)*, cuyos parámetros son las variables donde se almacenará el mensaje de la siguiente forma:

- IDOrden es una variable de tipo *unsigned long*, la cual almacenará el identificador del mensaje recibido.
- Orden es un vector de 8 elementos de tipo *BYTE*, que almacenarán el mensaje (un mensaje CAN puede constar de hasta ocho bytes en el campo de datos).
- OrdenLon es la variable que almacena el número de bytes de válidos del mensaje. Para este caso, siempre le llegará que son válidos 2 datos, por lo que indicaría que solo deberían procesar los dos primeros elementos del vector que contiene el mensaje.
- OrdenFlags guarda la información relativa a datos del mensaje, como el tipo de ID (estándar o extendida), si es una petición remota de transmisión (RTR, Remote Transmision Request) u otros indicadores de mensaje. Para este caso contendrá siempre ceros, ya que es un flujo de información normal con ID estándar.

Por otra parte, se han utilizado también tres librerías incluidas con el compilador C18, además de la que corresponde al modelo del PIC utilizado, que son:

- delays.h: incluye varias funciones para hacer pausas según un número determinado de ciclos de instrucción. La frecuencia de instrucciones es la cuarta parte que la frecuencia del oscilador instalado, pues tarda cuatro ciclos de reloj por instrucción. Se ha utilizado para implementar una rutina que introduzca retardos en milisegundos, cuyo prototipo es *void delayms(int t).*
- pwm.h: aquí se encuentran las funciones que facilitan la configuración del módulo CCP para su operación en modo PWM. Se han utilizado las funciones *void OpenPWM1(char period)* y *void SetDCPWM1(unsigned int dutycycle)*.Estas funciones se utilizan:

- *void OpenPWM1(char period)*: Configura y abre el PWM, y escribe en el registro del timer 2 PR2 el valor que se le pasa como argumento (ver figura 45). Como se explicará en breve, el valor será 255 para poder obtener los 10 bits de resolución.
- *void SetDCPWM1(unsigned int dutycycle)*: configura el valor del PWM con los 10 bits menos significativos del argumento *dutycycle* cuyo intervalo de validez, en decimal, es de 0 a 1023. Para el valor 0 se obtiene una salida constante a nivel bajo.
- También se han creado dos funciones para apagar y encender el PWM, que son *void stopPWM(void)* y *void startPWM(void)*, que colocan a 0 o a 1, respectivamente, el bit 2 y el 3 (el bit menos significativo es el 0) del registro CCP1CON (CCP1CON=0b00001100 si está activado, y todos a 0 si está desactivado). Cuando está desactivado el CCP, la salida por la patilla se corresponde con lo que esté guardado en el bit del registro de salida LAT LATCbits.LATC2.

- timers.h: contiene las funciones para configurar e inicializar los timers. Se ha utilizado para configurar el timer 2 con la función OpenTimer2(TIMER_INT_OFF & T2_PS_1_16), donde se ha indicado en el argumento que las interrupciones están deshabilitadas y el preescalado del reloj es de 16, para obtener una frecuencia de PWM baja.

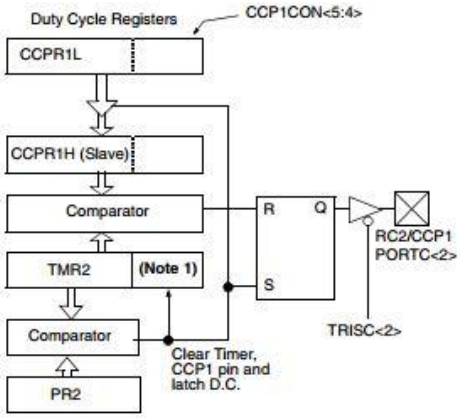


Figura 45. Diagrama de bloque simplificado del PWM del PIC

Con todo esto, a continuación se explica el código cargado al microcontrolador según se ha escrito en un sencillo programa, el cual se ha guardado en un archivo main.c:

- Primeramente, se incluyen las librerías a utilizar y se configura el microcontrolador mediante las sentencias [25] [26] [27] *#pragma config*, donde se le indica el tipo de oscilador, que es de alta frecuencia *(#pragma config OSC = HS*), se desactiva, el

watchdog y el modo de programación de bajo voltaje (WDT = OFF, LVP = OFF, tal y como indica el manual del ICD2), además del activar el MCLR para poder utilizar el circuito de reset (MCLR=ON). Aquí se pueden ver las opciones configuradas, a excepción de las configuraciones de varias opciones de protección contra escritura:

```
#pragma config OSC = HS, FCMEN = OFF, IESO = OFF
#pragma config PWRT = ON, BOREN = OFF, BORV = 1
#pragma config WDT = OFF, WDTPS = 32768
#pragma config PBADEN = OFF, LPT1OSC = OFF, MCLRE = ON
#pragma config STVREN = OFF, LVP = OFF, XINST = OFF
```

- Se asignan unos nombres a los registros que controlan los pines mediante varios *#define* para un uso más cómodo y claro en el control de las salidas, y se incluyen los prototipos de las funciones anteriormente citadas, cuyas implementaciones se encuentran al final del archivo.

- Se inicia la rutina principal *main*, se declaran las variables a utilizar, y se configuran los puertos de salida del control de puente H como 0 para las direcciones y 1 para la señal de habilitación. La señal de habilitación se debe configurar a 1 para permitir una salida digital a nivel alto cuando se desactive el PWM en el caso de valor máximo, 1023, ya que con PWM contiene algo de ruido. El valor 0 se puede obtener sin problemas con *SetDCPWM1(0).*

- Se inicializa el PWM con el siguiente código:

```
OpenPWM1(255);
OpenTimer2(TIMER_INT_OFF & T2_PS_1_16);
SetDCPWM1(0);
```

La primera línea es, como se ha comentado anteriormente, el valor del registro PR2, que se utilizará para definir el periodo del PWM. Este es un registro de 8 bits que utiliza dos bits adicionales del preescalado para crear la base de tiempo de 10 bits. El valor configurado debe ser 255 para obtener la resolución máxima porque:

$$T_{on} = [DC \times PRE] \times Tosc$$
$$T_{pwm} = [(PR2+1) \times 4 \times PRE] \times Tosc$$

Con la limitación de los 10 bits del registro, que son 1024 valores, DCmax= (PR2+1) x 4, y si se otorgan valores mayores al DC que el máximo fijado, entonces Ton > Tpwm, con lo que se mantendrá a nivel alto de manera continua [28].

La segunda instrucción sirve para iniciar el temporizador 2 con un preescalado de 16 lo que, en conjunción con el valor de PR2=255 resulta una frecuencia de 1,22 kHz. Preescalados menores (el resto de posibles valores es 4 y 1) proporcionarían valores demasiado altos, de 4,88 kHz y 19,75 kHz.

La tercera línea simplemente configura la salida a nivel bajo como valor inicial.

- Se inicializa el módulo CAN con la función anteriormente descrita *ECANInitialize()* ,y se configuran todos los filtros y máscaras para que los buffers de recepción solo acepten mensajes con una ID (en las pruebas se ha utilizado 0x640). Para ello, se ha escrito una función, *void setId(int ID)*, la cual asigna el identificador a los registros correspondientes, que son RXFnSIDH (parte alta) y RXFnSIDL (parte baja), donde n son valores de 0 a 5 (el 0 1 el 1 corresponden al buffer 0, y los restantes al buffer 1).

El identificador estándar que se está utilizando es de 11 bits, y se distribuyen los 8 más significativos en el registro de la parte alta, y los 3 bits menos significativos del ID en los más significativos, como se puede ver en la figura 46. Los bits restantes del registro de la parte baja se corresponderían con bits de identificador extendido, que no se está usando, y en el bit 3 se escribe 0, pues es el que indica si es Id estándar o extendido (valor 0 para estándar).

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDEN(2)	—	EID17	EID16
bit 7							bit 0

Figura 46. Registros correspondientes a un filtro. Arriba RXFnSIDH, abajo RXFnSIDL.

Los valores de los bits de las máscaras, en los registros RXMnSIDH y RXMnSIDL (n es el número del buffer, que para el modo que se está utilizando, *Legacy,* tiene solo valores 0 y 1), se configuran todos a 1, pues así se indica que es necesario comparar todos los bits de los filtros con los del ID del mensaje recibido, de manera que solo permita la recepción de mensajes con ID exactamente igual al configurado.

- En el bucle infinito que contiene el código que estará ejecutando constantemente en el microprocesador se espera a que haya un mensaje disponible para leer (que haya pasado los filtros) con la función ECANReceiveMessage(&IDOrden,Orden,&OrdenLon,&OrdenFlags) y, según la información contenida en los dos bytes válidos del mensaje Orden[0] y Orden[1], se abrirá o cerrará la FetchHand con el valor de PWM que contenga el mensaje, indicando además el sentido de la acción mediante los LEDs a tal efecto, o se detendrá la

actuación del motor. Como el valor del PWM tiene una resolución de 10 bits, se ha codificado la misma de la siguiente manera:

- La orden 0xA0 indica que el sentido es abrir, 0xC0 es para cerrar y 0xF0 para par el motor. La información para estas acciones se encuentra en Orden[0], en los cuatro bits más significativos, y se corresponde con el valor en hexadecimal "A", "C" o "F". Por tanto, esta información indicará la acción a realizar.
- El valor del PWM está en Orden[1] y los dos bits menos significativos de Orden[0], y es el ciclo de trabajo que se escribe para controlar la habilitación del puente H en las órdenes de tipo abrir y cerrar.

Aunque la codificación de la información podría ser distinta, esta es fácil de entender y de realizar enmascarando y desplazando bits. Así, por ejemplo, se ha separado la información de esta manera:

```
OrdenBits=Orden[0] & 0b11110000;

PWMBits=((unsigned int)Orden[1]<<2)|(Orden[0] & 0b00000011);
```

La primera línea guarda en OrdenBits, que es la variable de tipo BYTE que se utiliza para decidir la acción a realizar, el valor contenido en Orden[0], pero con los cuatro bits menos significativos con valor 0.

La segunda línea escribe en la variable de tipo entero PWMBits los bits de Orden[1] desplazados dos posiciones a la izquierda y coloca los dos últimos bits menos significativos de Orden[0] en los bits menos significativos de PWMBits.

Se puede resumir la funcionalidad del código en el sencillo flujograma de la figura 47.

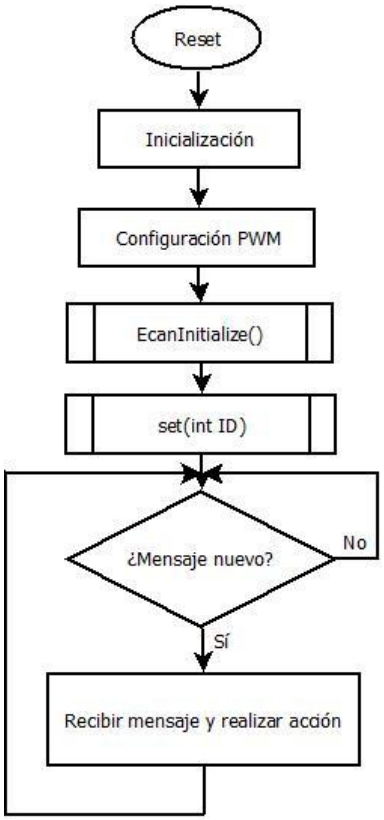


Figura 47. Diagrama de flujo del código del microcontrolador.

4.2. Programación en PC

Además de la programación del propio dispositivo, ya descrita en el apartado anterior, se ha creado código para su ejecución en el ordenador destinado a la manipulación de TEO, utilizando la API (*Application Programming Interface*, interfaz de programación de aplicaciones) que provee el fabricante de la tarjeta HICO.CAN-miniPCI , que es el módulo que se utiliza para comunicar dicho ordenador con Linux con el resto de dispositivos de manipulación integrados en la red CAN.

Mediante el propio compilador GCC (GNU Compiler Collection) que integra la distribución Debian instalada, se han creado varios pequeños programas de prueba para testar el funcionamiento del bus y del código del microcontrolador de la mano, con una pequeña interfaz de texto en la terminal del sistema operativo, como se describirá en el siguiente apartado.

De esta manera, con el fin último de poder integrar correctamente un código de control de la FetchHand con el sistema de control de movimiento mediante YARP, se ha decidido implementar una clase en C++ llamada *FetchHand*, que contenga todo lo necesario para abrir y mantener una comunicación a través del bus CAN con los dispositivos reales instalados en el robot TEO.

Para esta labor, y para el resto de pequeños programas de prueba que corrían bajo la plataforma Linux del ordenador integrado, ha resultado muy útil el manual del driver de Linux [29] que ponía a disposición Emtrion, el fabricante de la tarjeta CAN, especialmente los ejemplos que incluye. Para que la utilización de esta tarjeta sea posible, tal y como indica dicho manual, es necesario tener instalado el driver en forma de módulo de kernel (LKM, Loadable Kernel Module), válido para kernel 2.6.32, y que se encontraba instalado por investigadores del Departamento, así como incluir la cabecera de la API, "hico_api.h".

Para poder empezar a comunicar, en todo programa que se cree con el único fin de enviar mensajes y que vaya a utilizar esta tarjeta, se deben seguir una serie de pasos:

1. Incluir las librerías que se van a utilizar y, ya en la rutina principal, declarar todas las variables que se vayan a utilizar y la estructura del mensaje. Los campos que componen el mensaje descritos según esta API son:

- msg.**id**: ID o campo de arbitraje de CAN (el ID también lleva aparejada una prioridad).
- msg.**data[0:7]**: bytes de datos del mensaje.
- msg.**dlc**: indicación de longitud de datos válidos del mensaje.
- msg.**ff**: indica, si es un 0, que se utiliza el identificador estándar de 11 bits. Si es un 1, por el contrario, el extendido, de 29 bits.
- msg.**rtr**: *flag* de indicación de petición de transmisión remota.
- msg.**node**: número de nodo que ha recibido el mensaje
- msg.**iopin**: estado del pin IO del nodo, si el CAN es tolerante a fallos.

Los campos que se han escrito con valores distintos de cero son los tres primeros, ya que los tres últimos son características no utilizadas y el cuarto campo, referente al tipo de identificador, requiere que sea 0 para configurar el identificador estándar utilizado. En cuanto al campo de identificador, se escribirá en él la ID en hexadecimal del nodo al que va dirigido el mensaje.

2. Es necesario abrir uno de los dos nodos (*can0* y *can1*) de los que dispone la tarjeta CAN del PC, especificándolo de una forma similar al siguiente ejemplo, si se quiere poder leer y escribir:

```
canFd = open("/dev/can1", O_RDWR);
if(canFd<0){
        err(1, "could not open can node");
}
```

```
else
        printf("CAN node opened \n");
```

No obstante, después de este paso, es necesario configurar la velocidad de transmisión, que se recuerda que es de 1 Mbps, e iniciar el nodo, de manera similar a como se indica en estas líneas:

```
/* Set bitrate */
val=BITRATE_1000k;
ret=ioctl(canFd,IOC_SET_BITRATE,&val);
if(ret!=0){
        err(1, "could not set bitrate");
}
else
printf("bitrate set \n");
/* Start the node */
ret=ioctl(canFd,IOC_START);
if(ret!=0){
        err(1, "IOC_START");
}
else
        printf("CAN node started: %d \n",canFd);
```

3. Se recaban los datos a enviar, se escriben los campos del mensaje necesarios de la manera correcta (se recuerda que, para la aplicación de este trabajo, se están grabando los datos del valor del PWM y del tipo de acción en dos bytes, según la codificación explicada en el apartado anterior). Un ejemplo de cómo se enviaría una instrucción de abrir con un determinado valor de PWM introducido con anterioridad por el usuario se podría ver en el siguiente fragmento de código:

```
msg.data[1]=PWMval>>2;
msg.data[0]=0xA0 + (PWMval & 0b0000000000000011);
/* Write the message to the CAN bus */
 ret=write(canFd,&msg,sizeof(struct can_msg));
 if(ret!=sizeof(msg)){
        err(1, "Failed to send message");
 }
 else
        printf("message sent (%d): open \n",ret);
 printf("message code sent: Order= %X, PWM= %d : \n",
msg.data[0]&0b11110000,    (msg.data[1]<<2)+(msg.data[0] & 0b00000011));
```

4. Por último, una vez terminada la tarea, quedaría cerrar el nodo de esta manera:

```
close(canFd);
```

En la realización de la clase *FetchHand*, cuya documentación se incluye en los anexos, se han tomado estas consideraciones. A continuación, se indica de forma muy resumida de qué se compone esta clase:

- Posee dos miembros privados: uno referente a la ID (_id) y otro al identificador que se le asigna a un nodo cuando se abre (_canFd).
- Constructor parametrizado, para instanciar un objeto de esta clase asignándole una ID y el nodo CAN de la tarjeta HICO.CAN a la que está conectado. También realiza todas las labores de configuración descritas en el paso 2.
- Constructor vacío, que realiza las mismas acciones que el parametrizado pero con valores por defecto de _id=0x00 y dispositivo "/dev/can0".
- Función miembro *int FetchHand::action(int pwmVal)*, que recibe un valor de PWM en el rango [-1023,1023], ajusta el rango del PWM y su significado (el valor cero para el motor, valores positivos abren la mano según el valor del PWM introducido y negativos la cierran), escribe los datos del mensaje (paso 1), y lo envía (paso 3).
- Tiene otros miembros para ver o cambiar el ID y el destructor, el cual cierra el nodo (paso 4).

Esto facilita enormemente la implementación del código en YARP, pues el personal del Departamento ha podido reutilizar una clase anterior ya hecha en YARP de tipo de control de posición e insertar el código de esta clase con muy leves modificaciones, consiguiendo poder controlar la mano mediante la terminal del sistema del ordenador de TEO (previo lanzamiento del servidor de YARP) o incluso de otro ordenador conectado en red.

4.3. Test realizados

En este punto se tratan de forma breve algunas de las pequeñas pruebas apoyadas en software que se han realizado para la comprobación del funcionamiento y detección de errores.

4.3.1. Test del PIC en modo *Loopback*

Como pequeña y primera comprobación de que el programa cargado en el microcontrolador debería funcionar comandado por el ordenador, y con el fin de detectar errores en la recepción de mensajes, se han introducido algunas líneas de código consistentes en la composición de un mensaje con identificador estándar de tipo "abrir" que posteriormente se ha mandado por CAN (utilizando la función a tal efecto, y muy similar a la de recibir, de la *application note* AN878), con el módulo ECAN configurado para trabajar en modo *loopback* en lugar de en modo normal a

través de las opciones del archivo .def, de manera que se han copiado los datos del buffer de salida al de entrada, la mano se ha abierto y se ha encendido el LED asociado abrirse, comprobando así que la programación relativa a la recepción de mensajes era correcta.

4.3.2 Pruebas de emisión de mensajes de la tarjeta HICO.CAN mediante sus dos nodos

Conectando los dos nodos de la tarjeta entre sí mediante un cable con conectores DB-9 hembra con una resistencia de 120 Ω soldada entre sus cables CAN_H y CAN_L, se ha comprobado el correcto funcionamiento de la emisión de mensajes, utilizando el nodo 0 para enviar y el nodo 1 para mostrar por pantalla los datos enviados.

Los datos que se han enviado han sido del tipo "0xAB", "0xCB", "0xFF" y otros similares (parecidos a las instrucciones de enviar órdenes que se han descrito, aunque solamente un byte, sin PWM, pues al principio se ha probado sin este tipo de control y con otra codificación), mediante un pequeño programa que presentaba un menú en la termina para elegir el mensaje a enviar de una lista.

Mediante otro programa, configurado para el nodo 1, se ha comprobado que los mensajes enviados, que se mostraban por terminal en este segundo programa, llegaban correctamente.

También ha servido para saber que, si se conseguían enviar, una luz verde se iluminaba brevemente una vez, mientras que si la conexión era defectuosa o no existía resistencia permanecía una luz roja el intentar enviar el mensaje hasta reiniciar el programa o conseguir una conexión válida. Este último caso de error también se aplica los nodos han sido configurados para velocidades de transferencia distintas.

4.3.3. Prueba de control de la FetchHand mediante el PC y CAN.

En esta prueba se ha realizado un programa para ejecutar en el PC con el objetivo de verificar el funcionamiento casi completo del software (no se utilizaba aún la clase FetchHand), permitiendo al usuario decidir entre las siguientes opciones: 1:abrir, 2:cerrar, 3:parar, 4:cambiar ID, 5:salir. Si la opción era una de las dos primeras se solicitaba un valor de PWM y se mandaba, si era la tercera solo se mandaba el comando de parar, y la cuarta solicitaba un nuevo ID con el que mandar los siguientes mensajes.

Esta prueba ha sido muy útil para comprobar el funcionamiento general del sistema, tanto del control de apertura y cierre, como del procesamiento y ejecución correcta

del valor del PWM, y del filtro, ya que si se cambiaba la ID por otra que no fuese la configurada (0x640) la mano dejaba de responder, pero se volvía a poner la mano realizaba las acciones ordenadas de nuevo. En la figura 48 se puede ver el menú que desplegaba el programa.

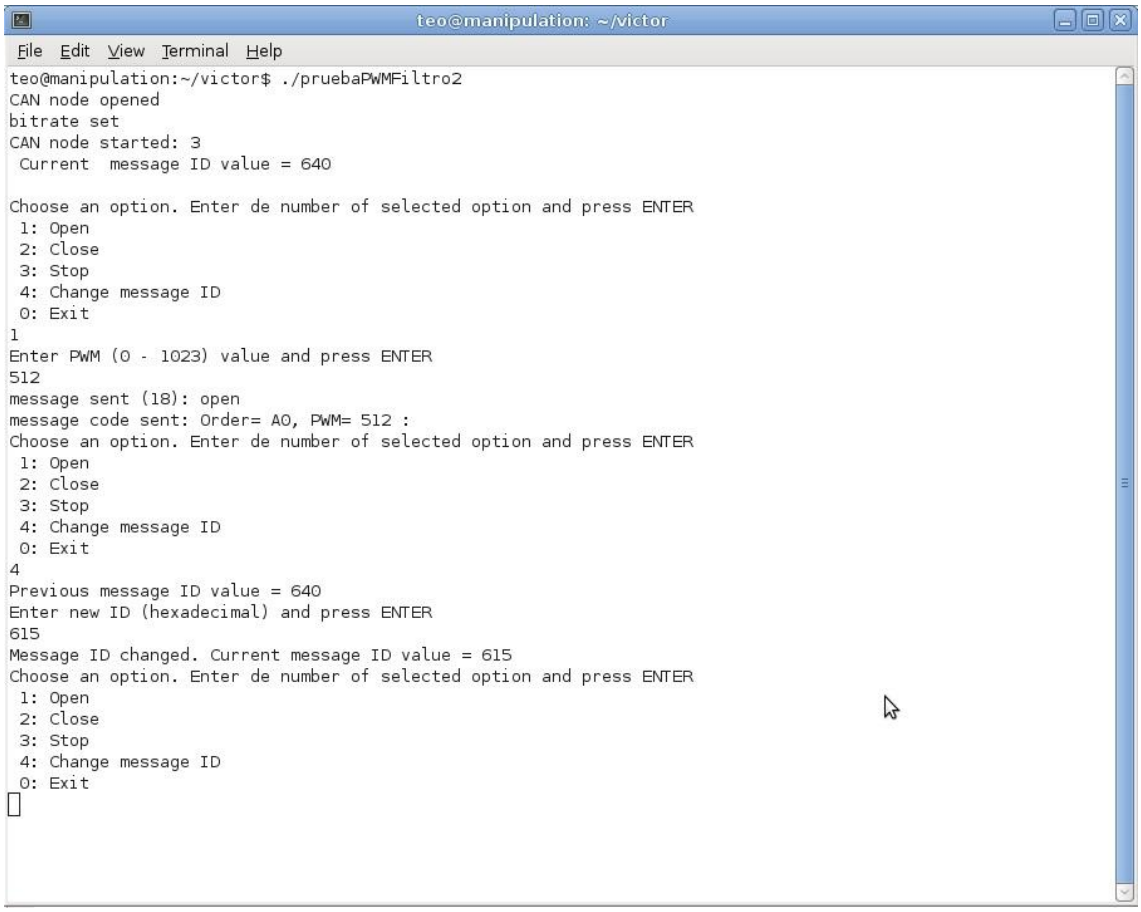


Figura 48. Visualización de la terminal con un pequeño programa de prueba para el control de la mano .

4.3.4. Prueba y representación gráfica del control por PWM

Una vez implementada la clase *FetchHan*d se ha escrito un pequeño programa que aumentaba los valores del PWM al pulsar una tecla con el fin de, por una parte, probar el funcionamiento correcto de la clase y, por otra, obtener información relativa al voltaje que recibe la mano entre sus bornes en función del valor de PWM y sentido de giro (abrir con valores positivos o cerrar con valores negativos). El resultado se encuentra representado en la figura 49, donde se puede observar que existe una gran linealidad entre ambas magnitudes, si bien se puede perder un poco alrededor de cero y de los extremos. También cabe destacar que no existe movimiento por debajo de un valor de PWM de alrededor de 340 sobre 1023.

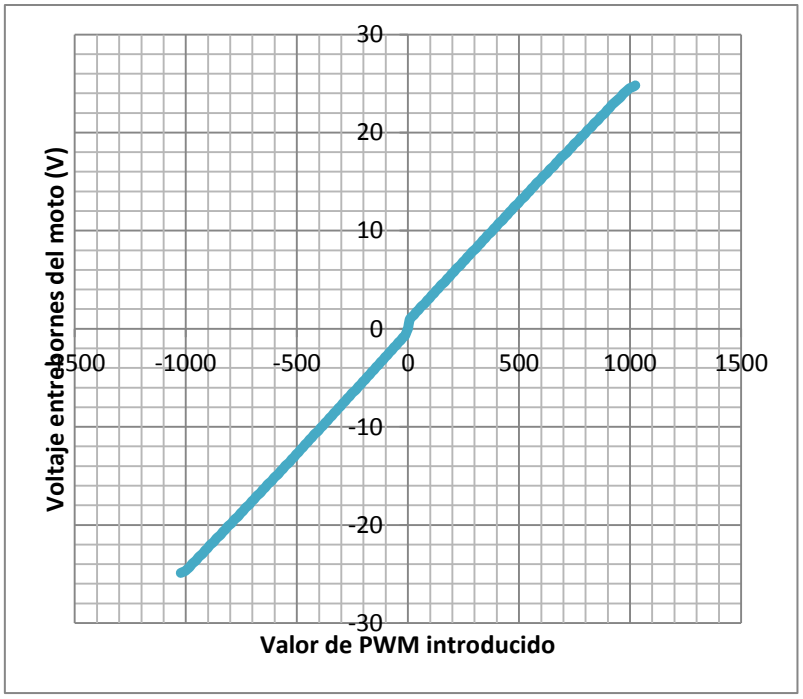


Figura 49. Gráfica de voltaje en el motor en función del ciclo de trabajo.

Figura 50.Captura de osciloscopio con PWM de 517 para abrir. Las señales, de arriba a abajo: habilitación (PWM), alimentación 5 V, salida puente 1, salida puente 2.

5. Conclusiones y futuras mejoras

El trabajo realizado ha cumplido los objetivos fijados en cuanto a:

- Diseño de la electrónica de control de forma que la placa se adapte a la forma de la mano, debiendo acoplarse en uno de sus lados.
- La electrónica, según las pruebas realizadas en cada bloque y las del sistema completo en montajes de prueba, es fiable. Para ello, se ha tenido que rechazar un componente para el control del motor, y se ha sustituido por otro con mayores referencias y una respuesta mejor en las pruebas. Aun así, en caso de que se presentasen problemas en dicho componente, el encapsulado en el que se presenta el incluido en el diseño final (orificio pasante, se puede montar con zócalo), distinto de como se puede encontrar el componente rechazado (montaje superficial), hace posible su reemplazo de manera sencilla.
- El software de control integrado en la mano es simple y funcional, así como la manera en la que se ha presentado al Departamento la parte que ejecuta el ordenador de control que incorpora TEO, que es en forma de clase autocontenida, habiendo hecho posible su integración rápida en YARP para facilitar el control de la mano por el bus CAN.

Otros objetivos paralelos que se han alcanzado, propios de un proyecto académico como es este, es la capacidad para utilizar las herramientas de software que han sido necesarias para el desarrollo del trabajo, como son los conocimientos adquiridos para el diseño mediante OrCAD, los adquiridos para la utilización del entorno MPLAB de Microchip y el depurador ICD2, y que se pueden extrapolar a otras herramientas similares, la mejora en el manejo del osciloscopio del laboratorio y la familiarización con un protocolo de transmisión de datos como es CAN.

Por otra parte, quedaría corroborar que la electrónica diseñada puede aguantar los ciclos de funcionamiento de la mano, así como utilizaciones prolongadas del dispositivo.

En cuanto a mejoras en el diseño se propone, una vez que se ha comprobado la fiabilidad del actual diseño, la sustitución de algunos de los componentes más voluminosos por otros equivalentes, como podría ser el caso de algunos condensadores, la bobina o el puente H L293D, del cual existe también una versión de soldadura superficial con un área y una altura menor y, en general, la sustitución de los componentes de mayor tamaño por otros con un encapsulado más compacto.

El mejor aprovechamiento del espacio en la placa debido a la sustitución de estos componentes se podría utilizar para la implantación de un convertidor DC/DC para el sensor de fuerza incorporado en la muñeca, evitando así la necesidad de encontrar otro lugar cercano para la instalación del mismo.

Referencias

[1] J. García, "Análisis de la arquitectura hardware modular y descentralizada del robot humanoide TEO," Universidad Carlos III de Madrid, Leganés, Tesis de Máster 2014.

[2] Handle Project. [Online]. http://www.handle-project.eu/

[3] J. Rodríguez, "Diseño e implementación de una mano robótica de bajo coste," Universidad Carlos III de Madrid, Leganés, Trabajo de Fin de Grado 2014.

[4] Lacquey. Lacquey documents. [Online]. http://www.lacquey.nl/node/10442

[5] mbed. mbed LPC1768. [Online]. https://mbed.org/platforms/mbed-LPC1768/

[6] A. López, "Diseño y desarrollo de un módulo de conexión a CANopen de un sensor comercial de fuerza/par," Universidad Carlos III de Madrid, Leganés, Proyecto Fin de Carrera 2011.

[7] J. Fortes, "Diseño e implementación encoder absoluto CANopen," Universidad Carlos III de Madrid, Leganés, Proyecto Fin de Carrera 2009.

[8] kvaser. The CAN Bus Protocol. [Online]. http://www.kvaser.com/about-can/the-can-protocol/

[9] Robert Bosch GmbH, *'BOSCH Controller Area Network protocol specification*. Stuttgart: Revision 2.0, 1991.

[10] YARP. YARP. [Online]. http://wiki.icub.org/yarp/

[11] Microchip. (2006) PIC18F2580 datasheet. [Online]. http://ww1.microchip.com/downloads/en/DeviceDoc/39637b.pdf

[12] Microchip. MCP2551 datasheet. [Online]. https://www.sparkfun.com/datasheets/DevTools/Arduino/MCP2551.pdf

[13] Texas Instruments. (1999, revised 2013, may) LM2575 datasheet. [Online]. http://www.ti.com/lit/ds/symlink/lm1575.pdf

[14] On Semiconductor. (2013, November) 1N5333B datasheet. [Online].

http://www.onsemi.com/pub_link/Collateral/1N5333B-D.PDF

[15] Texas Instruments. (1986, revised 2004) L293D datasheet. [Online]. http://www.ti.com/lit/ds/symlink/l293d.pdf

[16] STMicroelectronics. L293D datasheet. [Online]. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000059.pdf

[17] Microchip. (2004) Using MPLAB ICD2. [Online]. http://ww1.microchip.com/downloads/en/devicedoc/51265e.pdf

[18] Microchip. (2005) MPLAB ICD2 In-Circuit Debugger User's Guide. [Online]. http://ww1.microchip.com/downloads/en/devicedoc/51331b.pdf

[19] Microchip. CAD/CAE Symbols. [Online]. http://www.microchip.com/pagehandler/en-us/devtools/cad-cae-symbols.html

[20] ASROB. Tutoriales de electrónica. Robot rastreador/velocista. [Online]. http://asrob.uc3m.es/index.php/Robots_Rastreadores_/_Velocistas

[21] parsysEDA. OrCAD How-To - Import DXF Tutorial Cadence OrCAD Allegro. [Online]. http://www.youtube.com/watch?v=wTegtW9bPCs

[22] Allegro microSystems. A3950: DMOS Full-Bridge Motor Driver. [Online]. http://www.allegromicro.com/en/Products/Motor-Driver-And-Interface-ICs/Brush-DC-Motor-Drivers/A3950.aspx

[23] Arduino and Processing. [Online]. http://playground.arduino.cc/interfacing/processing

[24] Microchip. (2003) AN878 PIC18C ECAN 'C' Routines. [Online]. http://ww1.microchip.com/downloads/en/AppNotes/00878a.pdf

[25] Tutorial MPLAB C18 Desde 0. [Online]. http://www.ucontrol.com.ar/forosmf/tutoriales-guias-y-cursos-en-ucontrol/tutorial-mplab-c18-desde-0/

[26] Microchip, "PIC18F Peripheral Library Help Document," Manual de ayuda.

[27] Microchip, "MPLAB C18 Getting Started," Manual de ayuda 2005.

[28] Tutoriales PIC: Modulación PWM (Pulse Width Modulation). [Online]. http://picfernalia.blogspot.com.es/2012/06/modulacion-pwm-pulse-width-modulation.html

[29] Emtrion. (2009) Linux CAN driver manual. [Online]. http://www.emtrion.com/doc/hcanpci_linux_v1535en_edi1878.pdf

Anexo

A.1. Diseño electrónico

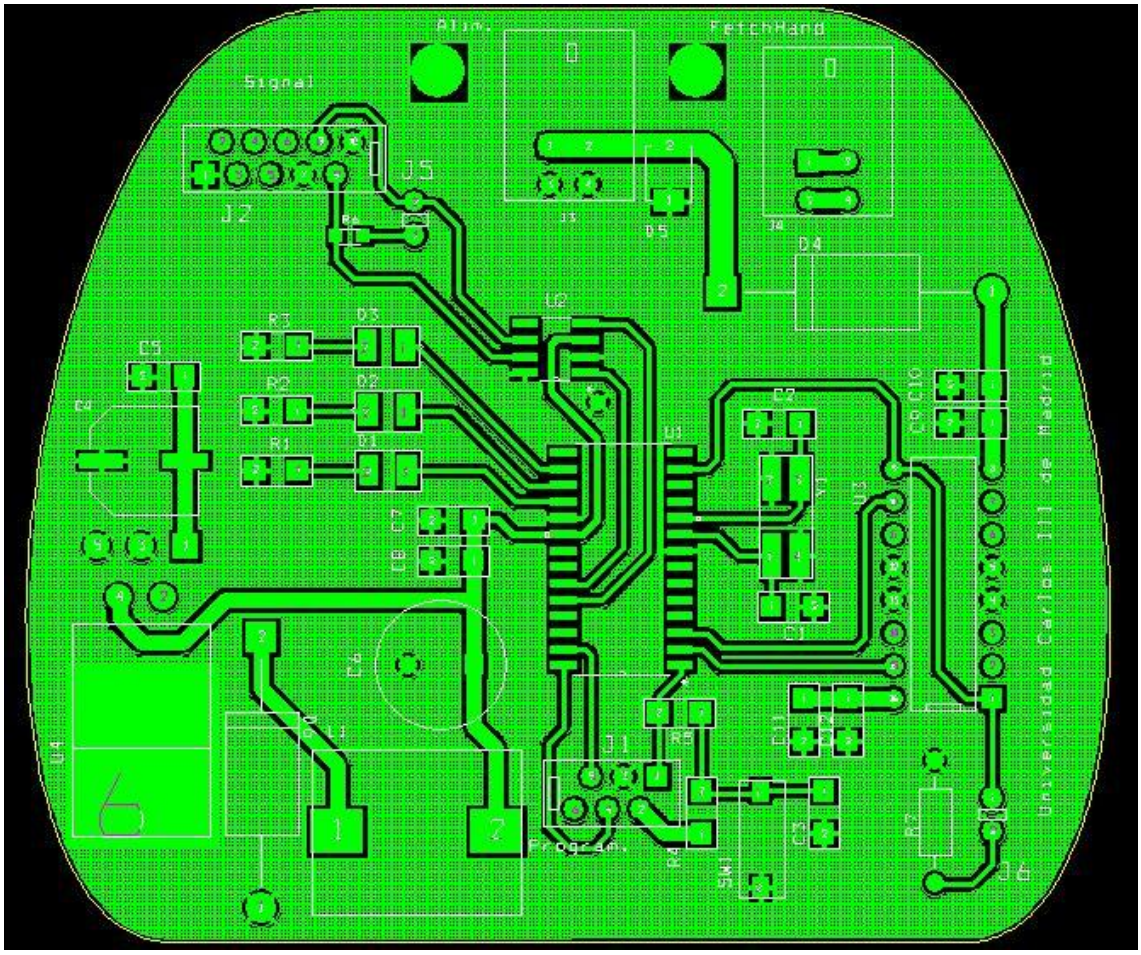


Figura 51. Capa TOP del diseño.

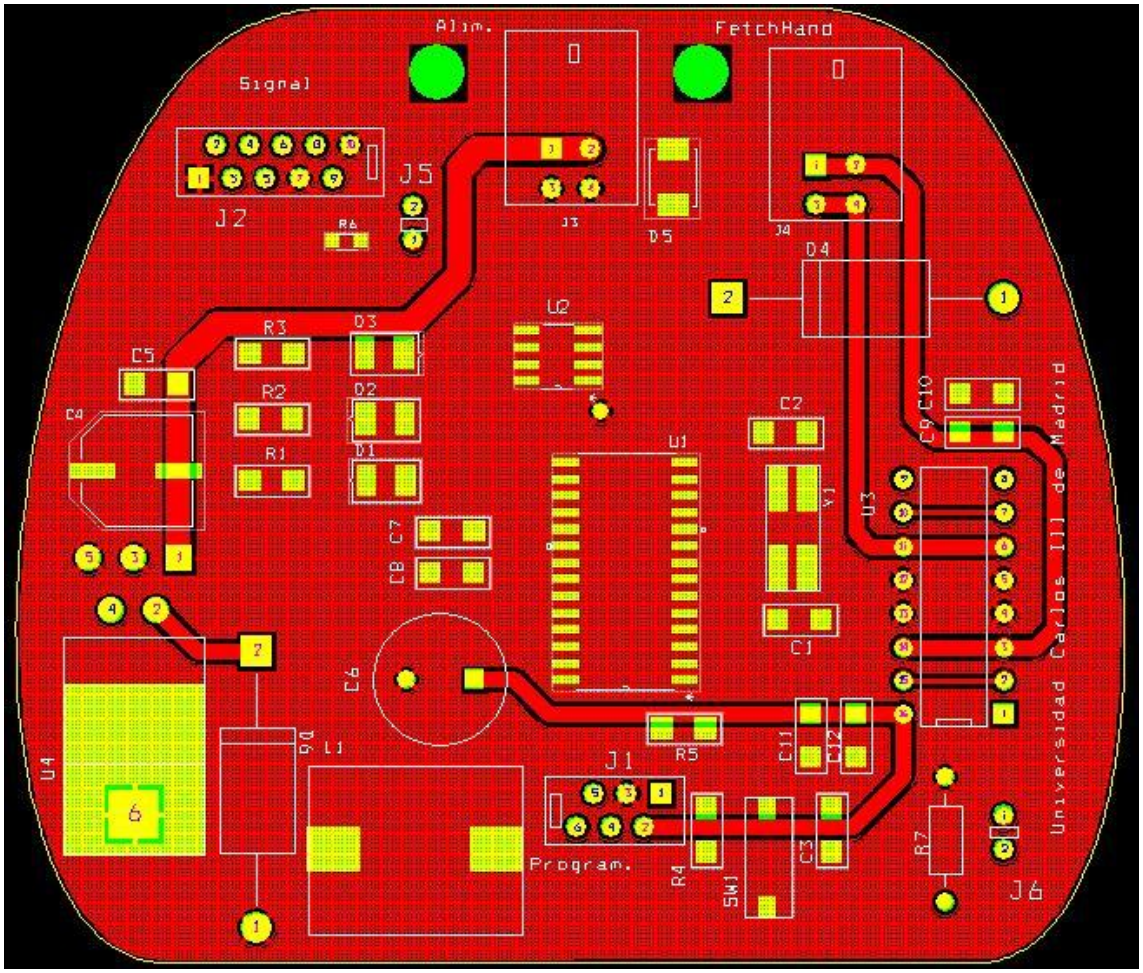


Figura 52.Capa BOTTOM del diseño

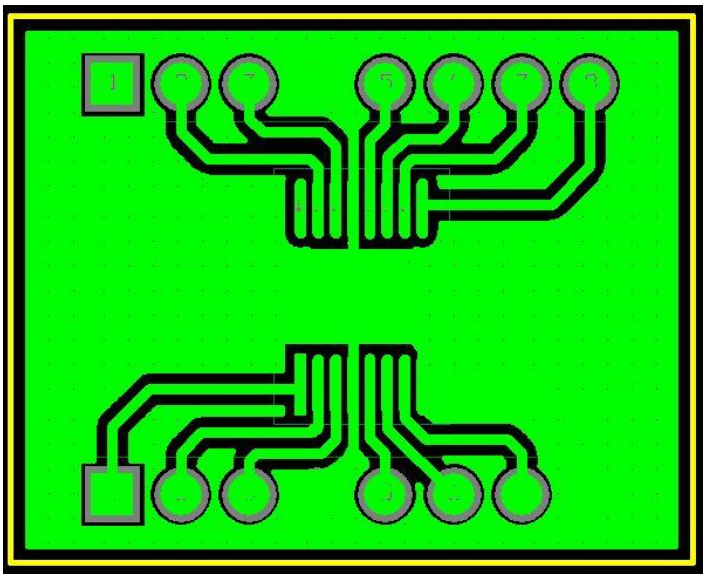


Figura 53. Capa TOP del adaptador del A3950.

A.2. Presupuesto para el diseño y fabricación de tres placas

Componente	Cantidad	Descripción	Precio unitario(€)	Coste
Microcontorlador PIC18F2580	3	Microcontrolador	4,81	14,43
Transceiver MC2551	3	Comunicación	0,9	2,7
Convertidor LM2575	3	Alimentación 5V	1,79	5,37
Puente H (driver) L293D	3	Motor	2,69	8,07
Bobina 470 uH	3	Alimentación 5V	1,042	3,126
Diodo Schottky	3	Alimentación 5V	0,459	1,377
Diodo zéner 10 V, 5 W	3	Motor	0,34	1,02
Diodo TVS (SMBJ) 36V	3	Protección	0,257	0,771
Pulsador	3	Reset	0,25	0,75
Cristal (20MHz)	3	Oscilador	0,98	2,94
LED rojo	3	Indicador	0,194	0,582
LED azul	3	Indicador	0,494	1,482
LED verde	3	Indicador	0,195	0,585
Jumper	6	CAN y enable	0,43	2,58
Resistencia 120	3	CAN	0,018	0,054
Resistencia 3,3k	3	Reset	0,53	1,59
Resistencia 2,2k	3	Reset	0,53	1,59
Resistencia 330	9	Indicador	0,45	4,05
Resistencia 4,7k	3	Pull-down enable	0,05	0,15
Condensador 15 pF	6	Oscilador	0,178	1,068
Condensador 100nF, 50V	15	Filtros	0,074	1,11
Condensador 10uF, 50V	9	Filtros	0,239	2,151
Condensador 100 uF, 50V	3	Alimentación 5V	0,482	1,446
Condensador 330uF,10V	3	Alimentación 5V	0,35	1,05
Conector Micro-Fit	6	Alimentación y motor	1,38	8,28
Conector Micro-MaTch 6	3	Programación	0,56	1,68
Conector Micro-MaTch 10	3	Datos	0,63	1,89
Fabricación PCB[4]	3		42,85	178,55
			Impuestos:	21%
			Total:	**303,03482**

Tabla 8. Presupuesto y lista de componentes para montar 3 PCB (una por mano más otra de repuesto).

[4] El precio total de la fabricación de las placas es la suma de los coste unitarios más un coste fijo de 50€ correspondiente a la creación del modelo.

Coste laboral

Nombre	Categoría	Horas empleadas	Coste horario(€)	Coste(€)
V. Sanz	Ingeniero	250	18	4500
J. García	Ingeniero Sénior	27	26	702
			Total:	**5202**

Tabla 9. Coste laboral del proyecto.

Coste total

Coste laboral(€)	5202
Coste material(€)	303,04
Total (€):	**5505,04**

Tabla 10. Coste total del diseño y fabricación de 3 tarjetas.

A.3. Planificación

A continuación se adjunta un diagrama de Gantt que representa una estimación de la organización del proyecto en cuanto a duración de las tareas necesarias, dejando como reserva (se podría decir que es el buffer del proyecto) hasta el día 23 de septiembre. Al ser un caso muy particular, en el que la realización de las tareas depende de la misma persona, las tareas, casi todas las tareas tienen una predecesora que debe estar acabada, mostrándose una disposición de las tareas una detrás de otra. Así, el siguiente diagrama se debe tomar solo como una estimación inicial y para observar la clasificación de tareas.

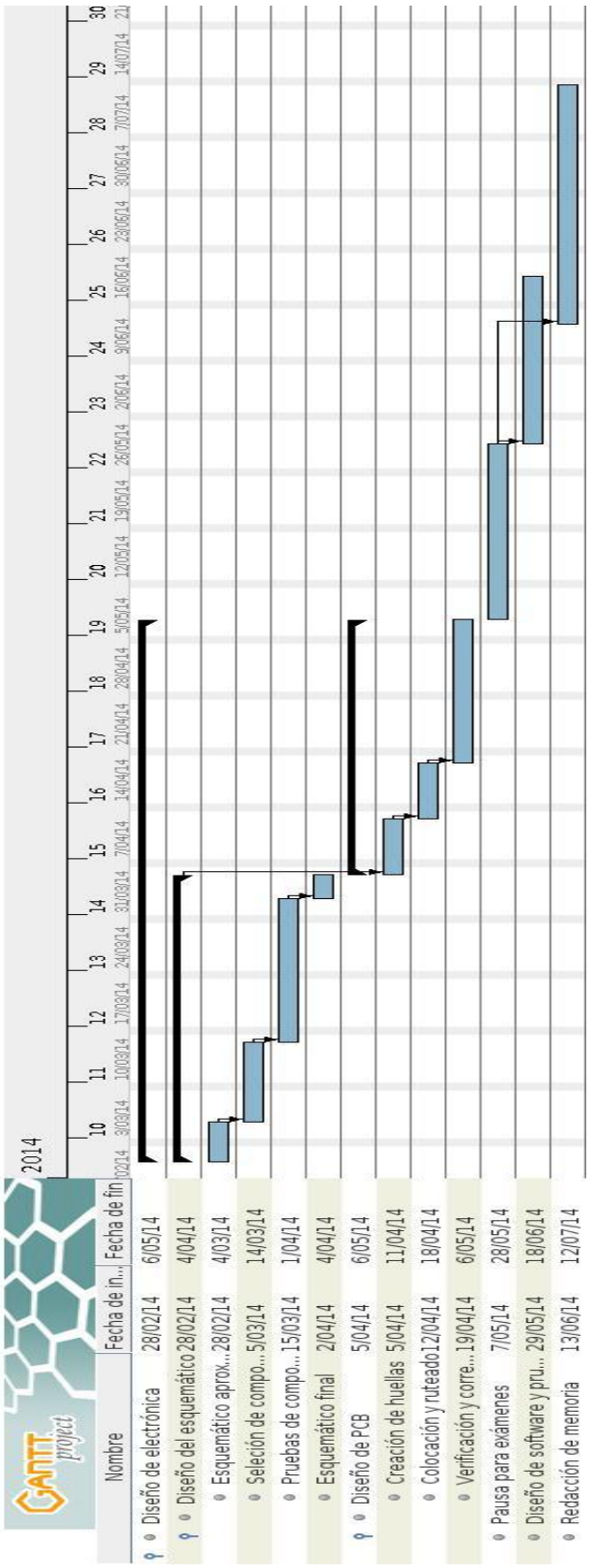


Figura 54.Diagrama de Gantt aproximado del proyecto.

A.4. Clase FetchHand

Class Documentation

FetchHand Class Reference

This class represents a TEO hand, which is a **FetchHand**, made by Lacquey (Robot Grasping Solution).

```
#include <FetchHand.h>
```

Public Member Functions

- **FetchHand** ()
  Empty constructor.
- **FetchHand** (int id, const string &canNode)
  Parametrized constructor.
- **~FetchHand** ()
  Destructor.
- return The acceptance ID number int **getId** ()
  Get hand ID.
- void **setId** (int id)
  Set hand ID.
- bool **action** (int pwmVal)
  Commands hand by PWM.

Detailed Description

This class represents a TEO hand, which is a **FetchHand**, made by Lacquey (Robot Grasping Soulution).

Constructor & Destructor Documentation

FetchHand::FetchHand ()

Empty constructor.

This constructor assings _id=0 and the CAN node 0 "/dev/can0".

FetchHand::FetchHand (int id, const string & canNode)

Parametrized constructor.

Parameters:

id	integer which represents hand acceptance ID.
canNode	is the node of miniPCI CAN.

FetchHand::~FetchHand ()

Destructor.

Closes CAN node

Member Function Documentation

bool FetchHand::action (int pwmVal)

Commands hand by PWM.

Parameters:

pwmVal	an integer from -1023 to 1023. Sign + is for open and - for close. PWM resolution: 10 bits.

Returns:

The result of action. Returns false if an error occurred.

int FetchHand::getId ()

Get hand ID.

Gets the hand acceptance ID.

void FetchHand::setId (int id)

Set hand ID.

Sets the hand acceptance ID.

Parameters:

id	integer which represents hand acceptance ID.

A.5. Bit timing. Reporte de MBTime

Bit Time Calculations for the Microchip CAN

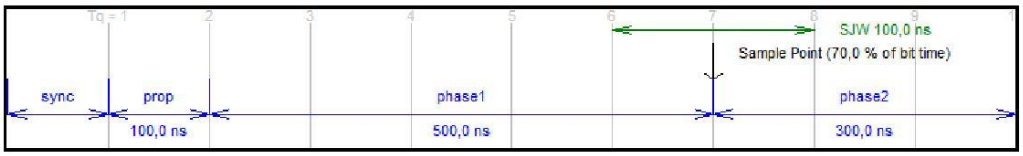
Setup Criteria	
Oscilator Frequency	20,000 MHz
Target CAN Bus Baud Rate	1,000 Mbps

Selected Options	
BRP-1 (Baud Rate Prescaler)	0
Tq (Time Quanta)	100,000 ns
Number of Time Quanta	10
% Error of Target Baud Rate	0,0 %

Bit Timing Setup in Tq	
Propagation Delay	1
Phase Segment 1	5
Phase Segment 2	3
Syncronization Jump Width (SJW)	1

Multiple bit sampling is off. Wakeup filter is off.

Bit Timing Diagram



Configuration Register Setup (PIC18/MCP251X) (neoVI blue/green, ValueCAN 2)		
Register	Binary	Hexadecimal
CNF1/BRGCON1	b'00000000'	0x00
CNF2/BRGCON2	b'10100000'	0xA0
CNF3/BRGCON3	b'00000010'	0x02

Configuration Register Setup (dsPIC33F,PIC24H,dsPIC30) (neoVI Red/Fire/Test/Yellow/ECU, ValueCAN 3)		
Register	Binary	Hexadecimal
CICFG1	b'0000000000000000'	0x0000
CICFG2	b'0000001010100000'	0x02A0

Generated 9:01:04 pm 06/29/2014
Microchip CAN Bit Timing Calculator by Intrepid Control Systems, Inc. (www.intrepidcs.com)