UNIVERSIDAD CARLOS III

Escuela Politécnica Superior

Departamento de Ingeniería de Sistemas y Automática

DESARROLLO DE UN PROTOTIPO DE SENSOR TÁCTIL DE TRES EJES

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA INDUSTRIAL ELECTRÓNICA INDUSTRIAL

Autor: Raúl Martín Delgado.

Tutor: Carlos Pérez Martínez.

Director: Santiago Martinez de la Casa Díaz.

25 de septiembre de 2009

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

A mi familia y amigos. Y a todas las personas que me han ayudado.

Capítulo 1

Introducción.

Este proyecto está contenido dentro de uno de mayor envergadura que es el diseño de el robot humanoide RH-2 en el Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid. El objetivo subyacente de este proyecto fin de carrera es la consecución del título de Ingeniero Técnico Industrial. Este Departamento tiene una composición multidisciplinar e internacional. Las enseñanzas que imparte pertenecen a las titulaciones de Ingeniería Industrial, Ingeniería Informática e Ingeniería de Telecomunicaciones, así como a las titulaciones de Ingenierías Técnicas Industrial. Cuenta con laboratorios[1] de Robótica, Automatización, Neumática, Sistemas de Control, Autómatas Programables y laboratorio de Visión.
El Laboratorio de Robótica RoboticsLab[2] es uno de los pioneros de la Robótica y Automatización, tanto a nivel nacional como internacional. Desde principios de los años 80 los miembros del grupo investigador llevan abordando de forma ininterrumpida proyectos de investigación y desarrollo en colaboración con empresas e instituciones nacionales y Europeas. Estas acciones se han caracterizado por un alto nivel científicotecnológico.

El RH-2 aparece como el sucesor del RH-1 en la serie RH. El robot debe ser capaz de realizar diversas tareas tanto en el sector industrial como en el de servicios, así como trabajar por sí solo e interaccionar con los seres humanos. El humanoide debe ser lo más parecido al hombre, por ello, sus características fundamentales son:

- Locomoción bípeda.
- Posesión de dos brazos manipuladores.
- Cuerpo donde ubicar todos los elementos de control comunicación y baterías.
- Una cabeza provista de sensores. La vista y el oido.
- Toma de decisiones autónoma.

[1] Ver web de referencia http://www.uc3m.es/portal/page/portal/dpto_ing_sistemas_automatica/equipamiento_docente

[2] Ver web de referencia http://roboticslab.uc3m.es/roboticslab/

- Independencia de la red eléctrica, es decir, autonomía para trabajar sin cables durante tiempo limitado.

- Desarrollo de funciones en entornos que no han sido específicamente diseñados para robots. Desde viviendas hasta oficinas, locales comerciales, escuelas, hospitales, ambientes exteriores, etc. Por ello el robot deberá ser capaz de caminar sobre superficies dispares, ya sean planas, inclinadas e incluso con pequeños relieves, también siendo capaz de subir y bajar escaleras.

Las tareas a realizar por el robot son básicamente de tres tipos:

- Manipulación, operaciones de montaje y transporte de objetos, tanto individualmente como en cooperación con otros robots o humanos.

- Funciones de seguridad, inspecciones o vigilancia.

- Funciones de atención al público e interacción con personar.

1.1. Objetivos.

Los robots, tanto robots industriales como robots humanoides necesitan conocer información de sí mismos y del entorno que les rodea.
Los sensores propioceptivos son aquellos que se usan para la medición del estado interno del robot, posición angular de cada uno de los grados de libertad, carga de las baterías... Mientras que los sensores exteroceptivos se refiere a la percepción de aspectos externos al robot; por ejemplo temperatura, presión, localización de objetos.

El proyecto que se expone a continuación tiene como objetivo el desarrollo de un sensor exteroceptivo para la ayuda al equilibrio de un robot humanoide. Los humanos podemos mantener el equilibrio gracias a tres sentidos, estos son la vista, "el oído"[3] y el tacto[43]. El tacto del pie con el suelo es muy importante a la hora de mantener el equilibrio, por ejemplo con una sola pierna. Así, se pretende crear un sensor bioinspirado con dos funciones. La medida de las fuerzas de reacción del suelo contra la suela del pie del robot humanoide RH-2 (CARHU) y la absorción de pequeñas irregularidades del terreno como lo haría una zapatilla de un ser humano o el propio pie humano. Esta medida se basará en crear una matriz de sensores en el que cada una de las celdas sensoras proporcionará un vector de desplazamiento o de fuerza (Ver figura 1.1) así se conseguirá una especie de tacto.
A modo de resumen los objetivos son:

- Creación de un prototipo de array táctil 2×2 basado en el Efecto Hall para la medida de fuerzas en tres direcciones orientado a la medida de éstas en la suela del pie de un robot humanoide.

[3]La audición es manejada por el tímpano, martillo, estribo y el caracol (coclea). El equilibrio es controlado por el ampula, el utriculo, el saculo y los conductos semicirculares.

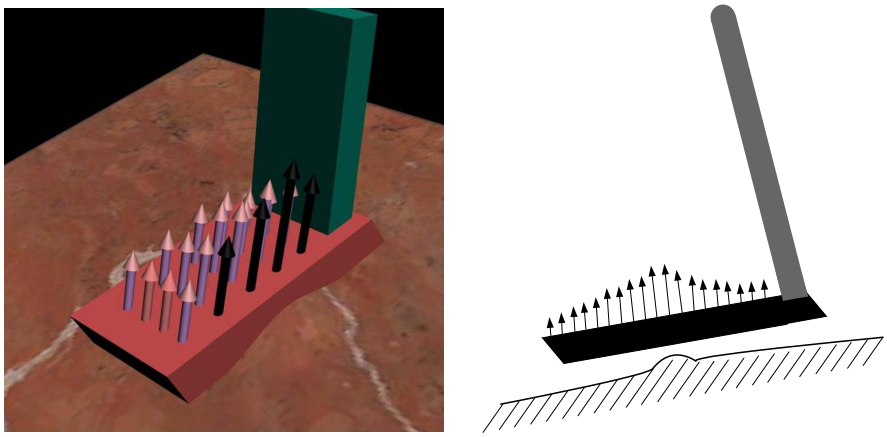


Figura 1.1: Vectores que indican la fuerza de reacción del suelo contra la planta del pié en una superficies no plana.

- Diseño del algoritmo de medida.
- Multiplexión de las señales del array para disminuir el número de puertos I/O.
- Diseño de la electrónica de acondicionamiento de señal.

- Envío de la información táctil a un PC para poder ser tratada posteriormente.

1.2. Motivación.

Desde principios del Siglo XX el ser humano ha estado buscando la forma de crear una réplica artificial de sí mismo, es decir, la creación de un humanoide. Para este propósito ha sido necesaria la colaboración de muchas ramas del conocimiento, en un principio solo ingenieriles, como la electrónica y la mecánica pero a medida que estos diseños han ido avanzando también han tenido cabida otras disciplinas como la psicología o el arte.

Básicamente un robot humanoide se compone de tres grandes bloques de desarrollo, estos son la adquisición de información del exterior, el procesamiento de la información y toma de decisiones y por último la actuación con el entorno. Tres ramas de la ingeniería trabajan en estos bloques, la electrónica, la informática y la mecánica. No se puede decir que cada uno de ellos sea independiente de los demás, pero si es cierto que en la adquisición de información del exterior tiene mayor protagonismo la electrónica, igual ocurre en el procesamiento de la información donde es la informática y en la actuación con el exterior, la mecánica. Conseguir mejoras en cualquiera de las anteriores secciones dará como resultado una mejora del producto final.

En los últimos años los avances en la capacidad de computación han dado como resultado grandes avances como la capacidad de procesar imágenes en tiempo real, etc. Pero aún así esa elevada capacidad de proceso carece de utilidad si no existe información que procesar, esta información es la procedente de los sensores.

En la actualidad se están realizando grandes investigaciones sobre la caminata de los robots basados en el ZMP[4], que se calcula mediante sensores fuerza par situados en el tobillo. También se están desarrollando arrays táctiles que permiten medir las presiones resultantes del contacto entre el suelo y la suela del pie de forma normal[5]. Pero por el momento no se ha acometido la labor de medir estas fuerzas en tres dimensiones de modo que se pueda tener una medida mucho más rica en información y que posibilite la implementación de algoritmos de control más complejos que impliquen una mejor caminata y una mayor adaptabilidad de la misma.

1.3. Descripción del libro.

El libro comienza realizando un breve repaso por el proyecto RH de la universidad, después se hace una comparación entre el sistema sensorial humano y los receptores táctiles artificiales. Se realiza una breve descripción de los principales sensores y arrays táctiles.

En el capítulo 3 se desarrollan las herramientas teóricas que van a ser necesarias para el desarrollo del proyecto. Éstas son las herramientas de posicionamiento en el plano y en el espacio (Triangulación y trilateración.), herramientas de traslación a de un sistema de coordenadas mediante matriz de traslación, estudio del campo magnético generado por un imán partiendo de la ley de Biot-Savart y análisis del mismo para estudiar su evolución, algoritmos de búsqueda en arrays de datos con éstos ordenados.

En el capítulo 4 se procede a describir lo que ha sido el desarrollo práctico, es decir, la construcción del prototipo del cual trata el proyecto. Desde la elección del microcontrolador hasta la elección del sensor efecto Hall, distribución física de los sensores, acondicionamiento de señal, filtros, multiplexión, alimentación. También se describen las comunicaciones entre el prototipo y un PC para el envío de información.

En el capítulo 5 se realizan pruebas de tiempos de cómputo y espacio necesario de cada uno de los diferentes algoritmos de medida desarrollados para elegir el más rápido, también se realizan pruebas de medida aplicando al prototipo incrementos de posición conocidos se comprueban los valores devueltos por el sensor y se obtienen gráficas del movimiento.

En el último capítulo se comprueba que los objetivos han sido satisfechos, se valora cualitativamente el trabajo realizado y se proponen posibles mejoras de prototipo.

[4] *Zero Moment Point*. Punto de Momento Cero.
[5] Perpendicular a la superficie.

Capítulo 2

Estado del arte.

El objetivo que se ha propuesto ha sido la medición de las fuerzas que actúan sobre la suela del pie del robot humanoide, este cálculo consiste en la medición de las deformaciones en los objetos causadas por esas fuerzas. Es por eso que se va a hacer una revisión de que tipo de sensores y soluciones se han utilizado hasta le fecha comenzando con una pequeña introducción al robot humanoide de la Univesidad Carlos III de Madrid. Tras esto una pequeña introducción en el tema de la medición de fuerzas y la bioinspiración de los arrays táctiles y por último se presentará una recopilación de sensores táctiles diseñados hasta la fecha.

2.1. El proyecto RH.

El proyecto RH consta de dos prototipos construidos, el RH-0 y el RH-1, en la figura 2.1 se pueden observar imágenes y esquemáticos de estos. Sin embargo, el RH-1 sólo representa la modificación de algunos elementos hardware manteniendo el tamaño y los grados de dibertad, un total de 21 excluyendo los dos GDL[1] de la cámara. Cada GDL se compone de un motor, encoder, driver y reductora, lo que le perite proporcionar el par necesario.

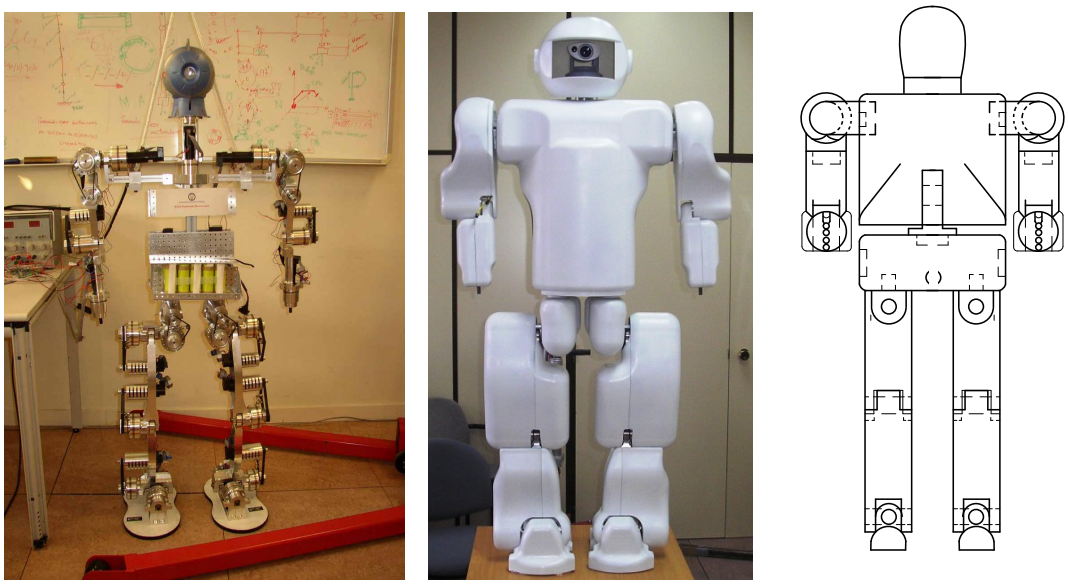
2.1.1. RH-2, el nuevo miembro de la familia RH.

Este nuevo robot dispondrá de 24 GDL distribuidos como se muestra en la tabla 2.1, un peso de $60Kg$ y será capaz de caminar a una velocidad de $1Km/h$ así como transportar objetos de hasta 2Kg de peso. Las principales modificaciones[6] que se han realizado con respecto del prototipo anterior RH-1 son:

- Se has incluido 3 GDL más, uno en cada brazo, concretamente en el codo en el eje transversal y otro en el tronco en el eje sagital.
- Ha aumentado la altura de $1{,}2m$ hasta $1{,}60m$.

[1] Como abreviatura de Grados De Libertad se suele emplear GDL, o DOF, del inglés Degrees Of Freedom

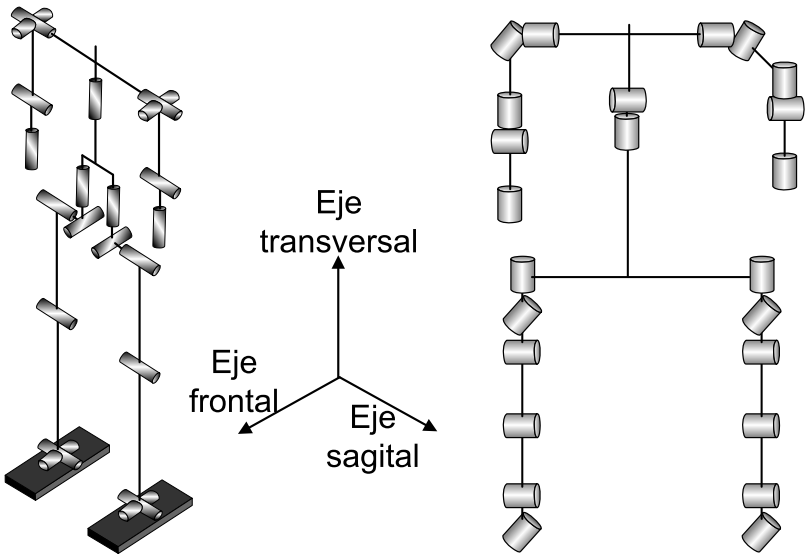
- Otros cambios como la sustitución de los sensores de sincronismo por encoders absolutos, diferentes unidades de control, diferentes drivers, diferentes motores...



(a) RH-0 (b) RH-1 (c) RH-2

Figura 2.1: Prototipos del proyecto RH.

En la figura 2.2 se muestran los modelos cinemáticos para los robots RH-1 y RH-2



(a) RH-1 (b) RH-2

Figura 2.2: Esquemas cinemáticos.

GDL	Número	Eje de movimiento
Piernas	**12 GDL**	
Cadera	$3\,(\times 2)$	▪ Sagital. ▪ Frontal. ▪ Transversal.
Rodilla	$1\,(\times 2)$	▪ Sagital.
Tobillo	$2\,(\times 2)$	▪ Sagital. ▪ Frontal.
Brazos	**8 GDL**	
Hombros	$2\,(\times 2)$	▪ Sagital. ▪ Frontal.
Codo	$2\,(\times 2)$	▪ Sagital. ▪ Transversal.
Muñeca	$1\,(\times 2)$	▪ Transversal.
Tronco	**2 GDL**	
Tronco	2	▪ Sagital. ▪ Transversal.

Cuadro 2.1: Distribución de los grados de libertad. RH-2.

2.2. Breve comparación entre el sistema sensorial humano y los receptores táctiles artificiales.

2.2.1. El sistema sensorial humano.

El sistema humano dispone de una serie de mecanismos nerviosos que se encargan de recoger la información sensorial generada en los diferentes receptores del cuerpo. Estos mecanismos denominados sentidos somáticos, son capaces de detectar y estimar, entre otras, las sensaciones táctiles y de posición en el tejido corporal[29].
Los sentidos somáticos del sistema humano, se pueden encuadrar en tres grandes grupos:

- Sentidos Mecanorreceptores: Captan efectos mecánicos, como los receptores del tacto de la piel (Tacto, Presión, Vibración) y de Posición: Estática, de Movimiento.

- Sentidos Termorreceptores: Captan la temperatura, incluyen las sensaciones de Frío y Calor.

- Sentidos del Dolor: Incluyen las sensaciones que dañen los tejidos del cuerpo.

Las principales características de los sentidos mecanorreceptores del ser humano son:

- Determinadas zonas de la piel son más sensibles que otras.
- Es posible localizar un estímulo táctil en una determinada zona de la piel.
- El sistema receptor humano es más sensible a las bajas frecuencias de un estímulo vibratorio.
- La sensibilidad al tacto es la capacidad para discriminar dos puntos próximos. Se han obtenido datos experimentales en [12] que establecen una resolución de 1,2 a 2,4 mm. entre los centros de las celdas sensitivas más elementales de la piel.
- Es posible diferenciar curvaturas de objetos en función del tacto. En [20], H. Liu concluye en que el sistema humano es capaz de discriminar, por ejemplo, dos esferas cuyos radios de curvatura se diferencien en un 10 %.

Existe una diferencia entre la respuesta al estímulo en zonas de piel glabra (Sin vello, recepción activa) y las zonas con vello (recepción pasiva). La piel glabra, es el órgano que realiza la exploración táctil en el sistema humano. La piel con vello tiene una respuesta más pasiva a la recepción.

2.2.2. Los receptores táctiles artificiales.

La tendencia ha sido la de compensar las deficiencias de medida en tiempo real de los sensores táctiles mediante el empleo de sistemas externos de visión. Esta solución, si bien en determinadas aplicaciones ha producido buenos resultados, implica un aumento de los sistemas electrónicos, y de control, que en muchos casos resulta inviable para la tarea que se quiere realizar.

Según el comportamiento físico, los sensores táctiles artificiales se pueden clasificar en: Sensores Piezoeléctricos, Capacitivos, Resistivos, Celdas de Carga, Galgas Extensiométricas, De Efecto Magnético, Acelerómetros, Biopotenciales, Sensores Basados en Análisis de Color, etc.
En la tabla 2.2 se muestra que tipo de sensores biológicos y artificiales miden cada uno de los estímulos característicos.

Estímulo y características	**Sensor biológico**	**Sensor artificial**
Estímulo y características	Sensor biológico	Sensor artificial
Fuerza: Medida de presión aplicada por una zona del dispositivo de agarre. Puede ser medida con algún dispositivo externo al robot.	Terminaciones nerviosas, Merkel, Ruffini, Paccini.	Galgas, Piezo eléctricos, Celdas de Carga.
Tacto: Medida continua de las fuerzas ejercidas en un array. Proporciona información muy difusa sobre el contacto.	Terminaciones nerviosas,	Resistivos, Capacitivos.

Continua en la página siguiente.

Cuadro 2.2 – Continuación de la pagina anterior.

Estímulo y características	Sensor biológico	Sensor artificial
	Merkel, Ruffini.	
Contacto: Medida de simples contactos, debida a las fuerzas ejercidas en pocos puntos de la superficie. Permite determinar la distribución de las fuerzas de contacto e identificar características del objeto.	Meissner, Órgano terminal del pelo, Paccini.	Resistivos, Capacitivos, Magnetoresistor.
Deslizamiento: Medida de la mínima fuerza que hay que ejercer para agarrar un objeto produciendo la suficiente fricción para detenerlo entre las superficies de contacto.	Meissner, Órgano terminal del pelo.	Acelerómetros, Micrófonos, Rodadores.
Posición: Medida de la situación de los puntos más característicos del objeto en las superficies táctiles.	Meissner, Merkel.	Resistivos, Capacitivos, Ópticos.

Cuadro 2.2: Sensores biológicos y artificiales que miden cada uno de los estímulos táctiles.

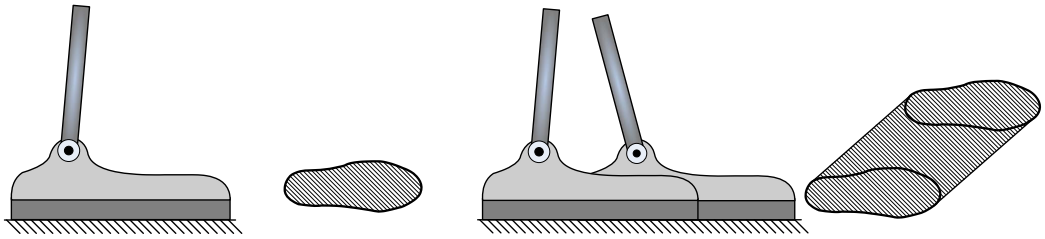
Los sensores, se puede definir en cuanto a sus características estáticas (describen la actuación del sensor en régimen permanente o con cambios lentos del estímulo) y dinámicas (en régimen transitorio).

- Caraterísticas estáticas:
  - Campo o rango de medida: Es dominio en la magnitud medida en el que puede aplicarse el sensor.
  - Resolución: Es el cociente entre la longitud de una arista y el número de sensores en ella.
  - Precisión: Desviación máxima entre valor real y el medido.
  - Repetibilidad: Desviación máxima entre valores de salida al repetir varias veces la misma medida.
  - Alinealidad: Máxima desviación entre la respuesta real y la lineal.
  - Sensibilidad: Variación de salida por unidad de magnitud de entrada.
  - Ruido: Desviación de la salida por el efecto del ruido asociado al sensor.
  - Histéresis: Variación del crecimiento o disminución progresiva de la medida.
- Características dinámicas:

- Velocidad de respuesta: El sensor debe responder a los cambios de la variable a medir en un tiempo mínimo. Lo ideal sería que la respuesta fuera instantánea.
- Respuesta frecuencial.
- Estabilidad y derivas.

2.3. Criterio de estabilidad ZMP.

El punto de momento cero es un concepto muy importante en la resolución del movimiento de un robot bípedo, como es el caso de los humanoides. Mantener la estabilidad dinámica no es tarea fácil, ya que el torso del robot tiene más masa e inercia que las piernas, las cuales tienen que soportar todo el peso. El punto de momento cero es aquél en el que la componente tangencial del momento resultante de la inercia, la fuerza de la gravedad y las fuerzas externas es cero como se afirma en [17].
Según la teoría de estabilidad de Miomir Vukobratović [42], para asegurar la estabillidad del movimiento y que no se produzcan pares de vuelco, el ZMP[2] tiene que estar en todo momento dentro del área efectiva de estabilidad. Este área depende del apoyo del robot, que puede ser simple o doble. Durante el apoyo simple, el área efectiva de estabilidad (Donde tiene que estar la proyección del ZMP) comprende la suela del pie como se muestra en la figura 2.3a. En el apoyo doble, el área de estabilidad corresponde al polígono de soporte entre los dos pies fig 2.3b.

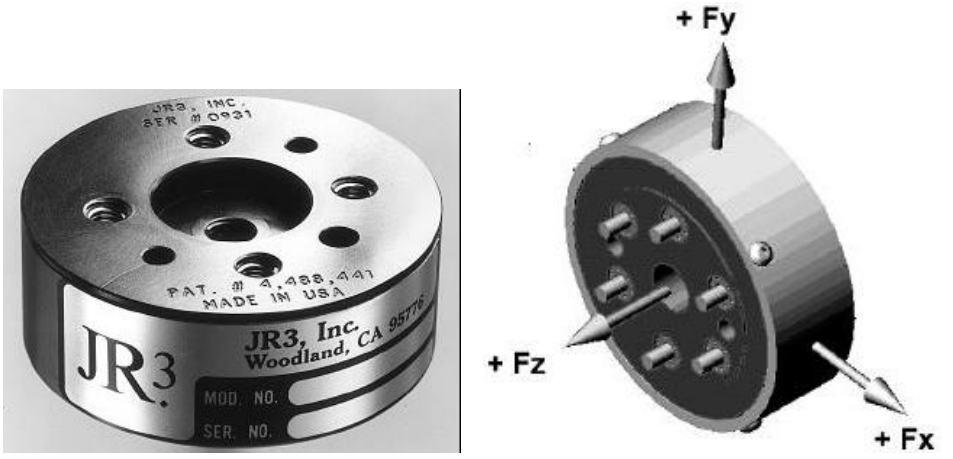


(a) Apoyo simple. (b) Apoyo doble.

Figura 2.3: Área efectiva de estabilidad.

Los problemas de control han sido estudiados por muchos investigadores mediante algoritmos de control basados en el ZMP.
Para calcular el ZMP se pueden emplear sensores comerciales fuerza-par como el que se muestra en la figura 2.4 situado en el tobillo proporcionan información sobre las componentes de las fuerzas en los tres ejes y los pares actuando en el tobillo durante el contacto del pié con el suelo.

[2]Zero Moment Point. Punto de momento cero.



(a) Sensor de fuerza par.

(b) Distribución de los ejes en el sensor de fuerza par.

Figura 2.4: Sensores fuerza-par.

2.4. Arrays táctiles para medida de fuerzas normales a la superficie.

Se trata de un tipo especial de sensores de fuerza que permiten obtener una imagen de las fuerzas ejercidas en una zona de un plano. Pero en realidad únicamente se componen de un conjunto de sensores, dependiendo de que sensores unitarios se utilicen dará lugar a diferentes arrays táctiles. En la actualidad se utilizan sensores unitarios con las más dispares técnicas de medida y fabricación. Cada uno de los sensores unitarios se posicionan formando una matriz o malla. Esta característica permite conocer la fuerza aplicada en multitud de puntos, siendo posible reconocer formas en los objetos que se está manipulando o pisando. Este tipo de dispositivos suelen montarse en las pinzas de los brazos de robot. Cada uno de los sensores de fuerza que componen la matriz suele ser una almohadilla de elastómero, que al comprimirse cambia su resistencia eléctrica de manera proporcional a la fuerza aplicada. Midiendo esa resistencia se puede obtener la información acerca de la fuerza, pero como ya se ha indicado, existen gran cantidad de técnicas de medida. En los siguientes apartados se mostrarán los arrays táctiles creados hasta le fecha. La resolución de este tipo de sensores vendrá dada lógicamente por las dimensiones de la matriz de sensores y por las limitaciones de cuan pequeños pueden fabricarse cada uno de los sensores unitarios que componen la matriz.

2.4.1. Array táctil de sensores resistivos. Goma resistiva.

Este tipo de sensores consisten en una mezcla de caucho con compuestos de carbono que une dos placas conductoras. Al contrario que en los sensores capacitivos, en los que el material que separa las dos placas es un aislante o dieléctrico, en este caso el caucho es conductor debido a las partículas de carbono. Ante una presión el caucho se deforma

aumentando así la superficie de contacto y disminuyendo la resistencia. Midiendo esta variación será posible obtener un valor de las fuerzas aplicadas.

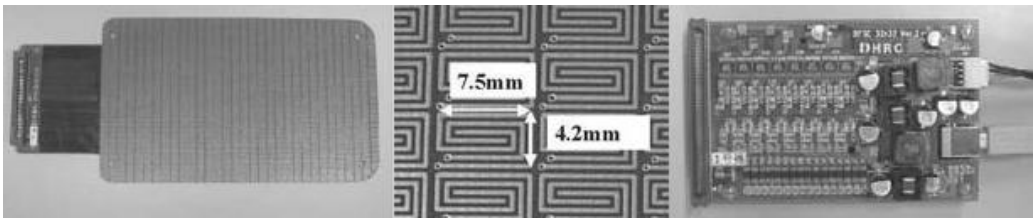
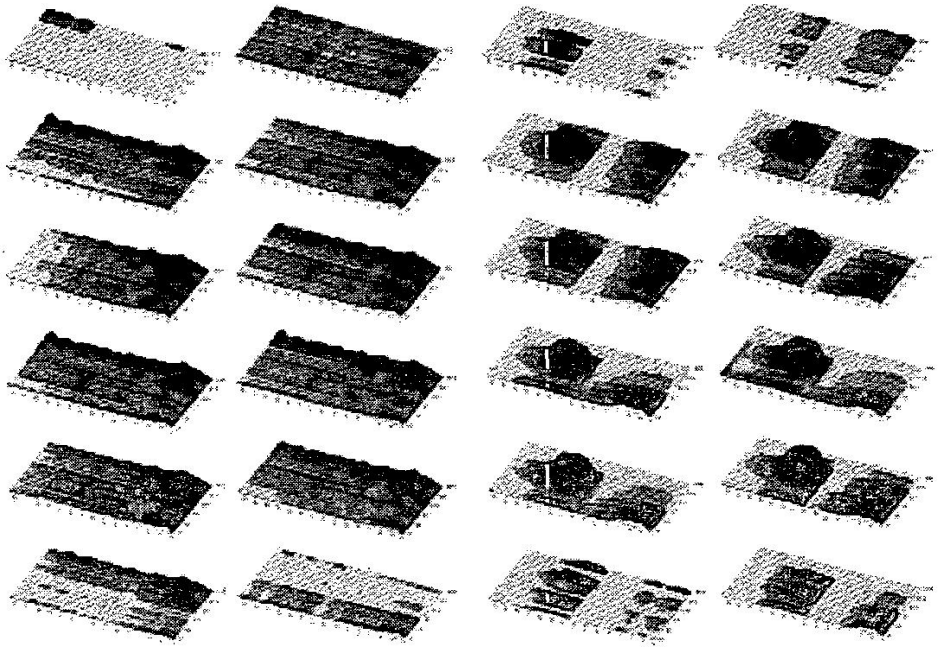


Figura 2.5: De izquierda a derecha: Suela del pié del robot HRP2-DHRC, vista cercana del electrodo del sensor y placa de entradas salidas.

Sensores de este tipo se encuentran en el robot humanoide HRP2-DHRC [16] dispone de una matriz de 32×32 sensores para medir la presión en la suela del pié. Cada uno de los sensores con unas dimensiones de $4{,}2 \times 7{,}0mm$ mide fuerzas de entre $0{,}25$ y $20N$ siendo la frecuencia de escaneo teórica de 1Khz.



(a) Sobre un suelo plano. (b) Sobre un objeto redondo y estrecho de metal

Figura 2.6: Resultados experimentales de la distribución de fuerzas durante la caminata.

En la figura 2.6 se observa la respuesta del sensor ante la caminata sobre distintas superficies. Este tipo de sensores permiten la utilización de un sistema de análisis de imágenes para determinar que tipo de suelo u objeto se está pisando.

La gran ventaja de estos arrays respecto a los capacitivos es que no son tan sensibles a las fuerzas de cizalla (Las tangentes a la superficie de contacto) sin embargo continúan

teniendo el mismo problema respecto a que únicamente pueden medir la componente de la fuerza normal a la superficie.

2.4.2. Array táctil de sensores capacitivos.

Este tipo de sensores tiene la ventaja de la simplicidad de construcción y la buena caracterización. Básicamente la respuesta es lineal. Como se observa en la figura 2.7

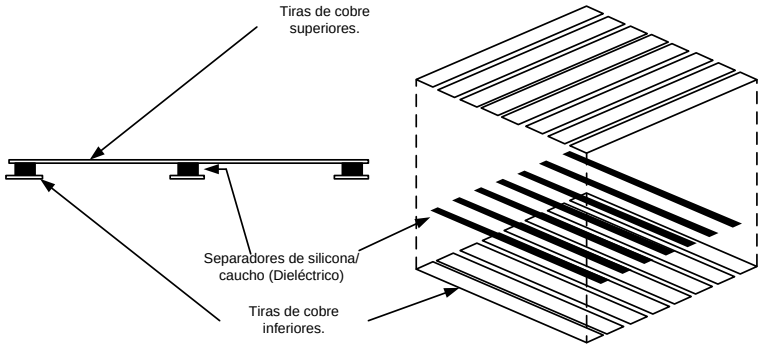


Figura 2.7: Array capacitivo

Como se muestra en [36], el sensor está compuesto de dos capas cruzadas de líneas de cobre y separadas por unas finas líneas de caucho. Cuando la fuerza es aplicada sobre la superficie, la distancia entre las dos tiras de cobre disminuye, lo que hace que la capacidad del condensador formado entre las tiras aumente. Mediante la medida de la capacidad que se produce en cada cruce de tiras conductoras se puede medir la distribución espacial de la presión sobre el sensor. La resolución de cada elemento sensor puede ser de hasta $0{,}1g$. El sensor forma una fina capa que puede ser fácilmente unida a dedos, manos, pies...
Para conseguir una medida correcta es necesario realizar una calibración. Para ello se mide la capacidad de los condensadores a presión atmosférica, no todos tienen la misma capacidad debido a las variaciones en las pistas y en los cables. Esto es lo que se llama Tara a cero "Zero Tare". Después se introduce el sensor en una cámara a presión y se mide la capacidad de cada condensador. De este modo se puede calcular la ganancia de cada condensador y la matriz de ganancias necesaria para hacer que todos los condensadores tengan la misma ganancia y así conseguir una respuesta uniforme.
$G_{ij} = \frac{P}{V_{ij}^{P} - V_{ij}^{baseline}}$
Donde:
$\boldsymbol{P}$ es la presión de test.
$\boldsymbol{V_{ij}^{p}}$ es la tensión del condensador ij a la presión de test P
$\boldsymbol{V_{ij}^{baseline}}$ es la tensión del condensador ij a la presión atmosférica
La componente tangente de las fuerzas no solo introduce un error considerable en la medida sino que puede llegar a dañar el array.

Pressure Profile Systems, Inc (PPS)[32] es uno de los fabricantes de este tipo de sensores con distribuidores en América del Norte, Asia y Europa. Algunas de las características generales de los sensores que fabrica son:

- Pueden medir presiones de hasta 2000 psi ahora bien, únicamente ofrecen calibraciones para medidas de presión de 700 psi como máximo.

- En sensores de medida de bajas presiones se consiguen resoluciones de $0{,}01$ psi.

- Frecuencia de medida elemento a elemento de $10khz$ en los arrays táctiles.

- Por lo general estos sensores reciben una calibración en el laboratorio y no es necesario realizarla de nuevo, pero es posible hacerlo si se desea.

- El tamaño mínimo de un elemento del array táctil es de $1mm \times 2mm$ y está protegido frente a la humedad de modo que no habrá que temer el efecto negativo que tiene la humedad en los sensores capacitivos.

- Aunque resisten la inundación durante unos instantes, esta podría dañar la capa de interconexión de los electrodos y no es recomendable. PPS suministra unos revestimientos resistentes al agua para evitar estos problemas.

Cabe destacar que PPS no solo proporciona el array táctil sino toda la electrónica asociada a la lectura de los datos. Además aceptan la realización de diseños "customizados" a petición del cliente.

2.4.3. Sensor de reconocimiento de pendiente.

El dispositivo que se muestra a continuación es una de las implementaciones más sencillas de un sensor táctil con únicamente tres elementos sensores, tres FSR[3]. Como se muestra en los artículos [38] y [37], se utilizan tres sensores con una colocación específica que permite el reconocimiento activo de la pendiente del suelo. El sensor táctil implementado en la suela del robot adquiere la distribución del suelo mediante mediante los tres sensores colocados en forma triangular como se muestra en la figura 2.8.

La información obtenida de los tres sensores de fuerza se utiliza para estimar la orientación de la pendiente del suelo una vez que se tiene un punto de contacto y mueve el pié para intentar mantenerlo paralelo a la superficie del suelo. El pie propuesto puede detectar pendientes del suelo durante la caminata para asistir a la misma y al equilibrio del robot en diferentes suelos. Puede detectar Suelo horizontal,pendientes ascendente, descendente, a izquierda y a derecha.

[3]Force Sensitive Resistor.

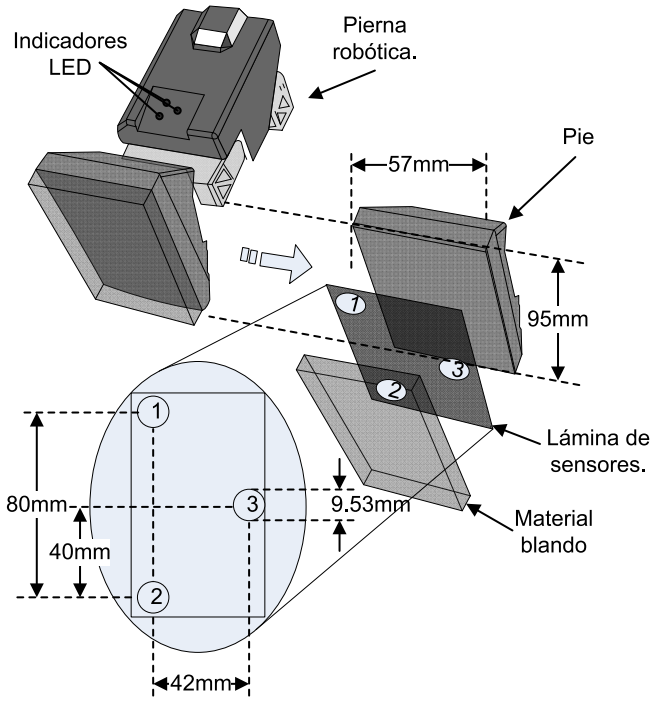
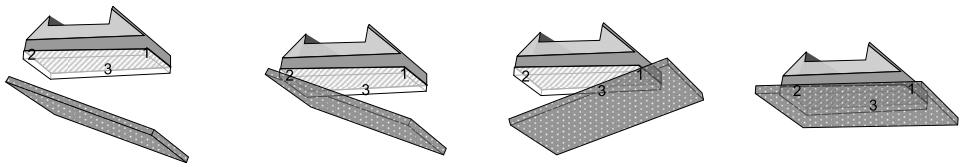


Figura 2.8: Despiece de un sensor táctil para la medida activa de la pendiente del suelo.

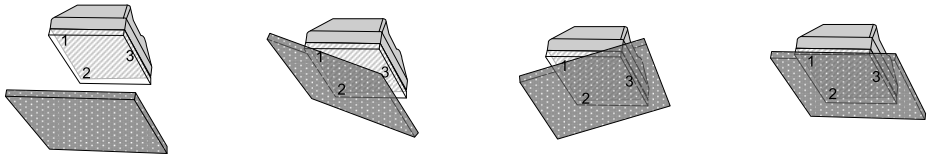


(a) No se detectan fuerzas en ninguno de los tres sensores.

(b) Fuerza del sensor 2>3>1, "Tocando en la parte trasera"

(c) Fuerza del sensor 1>3>2 "Tocando en la parte delantera"

(d) Fuerza del sensor 2≈3≈1, "Tocando El suelo plano"



(e) No se detectan fuerzas en ninguno de los tres sensores.

(f) Fuerza del sensor (1≈2)>3, "Tocando en la parte izquierda"

(g) Fuerza del sensor (1≈2)<3, "Tocando en la parte derecha"

(h) Fuerza del sensor 2≈3≈1, "Tocando El suelo plano"

Figura 2.9: Metodología de reconocimiento de la pendiente del suelo.

El funcionamiento del sensor se muestra en el conjunto de figuras 2.9 que a continuación se explica de manera más pormenorizada:

Pendiente descendente ver figura 2.9b se detecta cuando en la pisada el sensor 2 detecta la mayor fuerza, es decir $S2 > S3 > S1$.

Pendiente ascendente ver figura 2.9c se detecta cuando el sensor 1 detecta una fuerza mayor al resto de sensores, $S1 > S3 > S2$.

Pendiente descendente a la derecha ver figura 2.9f ocurre cuando la fuerza es detectada en los sensores 1 y 2 y es mayor que la detectada en el sensor 3, $(S1 \approx S2) > S3$.

Pendiente descendente a la izquierda ver figura 2.9g ocurre cuando la fuerza que detecta el sensor 3 es mayor que la detectada por los otros dos sensores, $S3 > (S2 \approx S1)$.

Pisada paralela al suelo ver figuras 2.9d y 2.9h Es detectada cuando los tres sensores detectan una fuerza aproximadamente igual.

No se está realizando la pisada ver figuras 2.9a y 2.9e Es detectada cuando ninguno de los tres sensores detecta fuerza alguna.

Los sensores FSR también se pueden emplear para realizar medidas de fuerza par, en este caso se pierde el objetivo de un array tactil pues no se pretende cubrir toda la suela del pie sino únicamente tomar medidas en cuatro puntos, ver figura 2.10 para calcular el ZMP según indica Genichiro Kinoshita en [19].

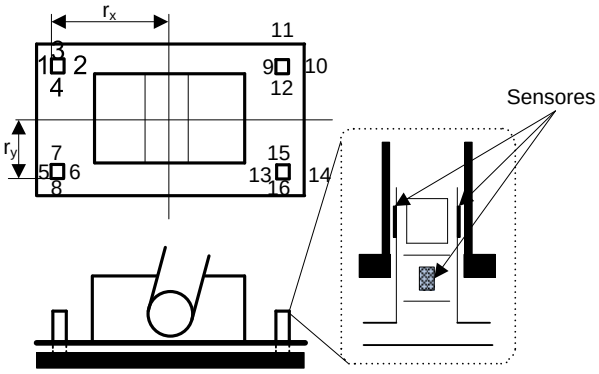


Figura 2.10: Colocación de los sensores para la medida de fuerzas

La medida se realiza mediante cuatro dispositivos, cada uno detecta las componentes de fuerza en dos ejes. Cada dispositivo utiliza cuatro galgas extensiométricas para medir la deformación. Los extensímetros $S_{4(i-1)+1}$, $S_{4(i-1)+2}$ detectan las deformaciones a lo largo del eje x, $S_{4(i-1)+3}$, $S_{4(i-1)+4}$ detectan las deformaciones a lo largo del eje y del dispositivo, para $i = 1, \ldots, 4$. r_x y r_y representan las distancia entre el centro del dispositivo a lo largo del eje x y a lo largo del eje y respectivamente. La posición del punto ZMP queda represenetada por (p_{0x}, p_{0y}, p_{0z}) como se muestra a continuación.

$$F_x = (S_1 + S_5 + S_9 + S_{13})/4 \tag{2.1}$$

$$F_y = (S_4 + S_8 + S_{12} + S_{16})/4 \tag{2.2}$$

$$F_z = (S_1 + S_8 + S_9 + S_{15})/4 \tag{2.3}$$

$$M_x = (S_3 - S_7 + S_{11} - S_{15})/(4r_y) \tag{2.4}$$

$$M_y = (S_1 + S_5 - S_9 - S_{13})/(4r_x) \tag{2.5}$$

$$M_z = (S_3 + S_{16})/2r_x - (S_5 + S_{10})/(2r_y) \tag{2.6}$$

$$p_{0x} = -\frac{M_y}{f_z},\ \ p_{0y} = -\frac{M_x}{f_z},\ \ p_{0z} = 0 \tag{2.7}$$

2.5. Arrays táctiles para la medida de fuerzas en tres dimensiones.

2.5.1. Celda de carga para la medida de fuerzas en tres direcciones.

A continuación se presenta un sensor táctil diseñado por Fuji Electric Corporate [33] utilizando únicamente silicio como material mecánico, con ello se consiguen las siguientes ventajas:

- Un sensor táctil que utiliza cristal de silicio como material mecánico tiene la propiedad de una buena linealidad y muy poca histéresis dado que el cristal de silicio es un buen material elástico.
- Es posible conseguir una densidad de integración de sensores muy elevada mediante la tecnología de fabricación de semiconductores
- El dispositivo sensor al completo (Transductor más adquisición y procesamiento de señal.) se puede fabricar en un tamaño muy reducido si los circuitos integrados para el procesamiento de señal se fabrican en la misma oblea que el sensor mediante la tecnología de semiconductores.

Principio físico de la medida.

Como se muestra en la figura 2.11, cuando se aplica una fuerza en cualquiera de las direcciones de los ejes Fx Fy Fz en uno de los extremos de la viga de carga en forma de anillo mientras el otro extremo permanece fijo, la distribución del estrés, es decir, la deformación de la viga curva se puede definir por las siguientes fórmulas:

Cuando se aplica una fuerza en el eje x. Ecuación 2.8.

$$\sigma_\theta = -\frac{3}{4} \cdot \frac{R^2}{b \cdot t^2} \cdot F_x \cdot \frac{t}{R - t} \cdot \cos\theta \tag{2.8}$$

Cuando se aplica una fuerza en el eje y. Ecuación 2.9.

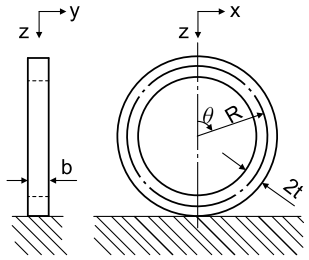
$$\sigma_\theta = -\frac{1}{2} \cdot \frac{R \cdot b}{I} \cdot F_y \cdot \cos\theta \tag{2.9}$$

Cuando se palica en el eje z. Ecuación 2.10.

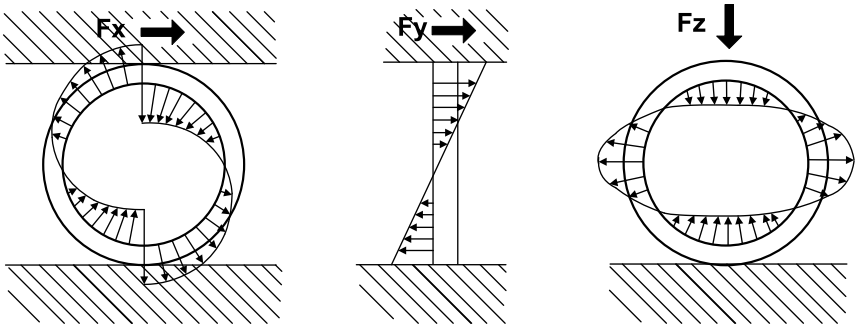
$$\sigma_\theta = \frac{3}{4} \cdot \frac{R}{b \cdot t^2} \cdot F_z \cdot \left(\frac{2}{\pi} - \sin\theta\right) \tag{2.10}$$

Donde todos los parámetros se pueden observar en la figura 2.11a. y significan lo siguiente:

- b : Espesor de la viga curva.
- 2t: Ancho de la viga curva.
- R : Radio de la viga curva.
- θ : Ángulo con origen cero en la linea vertical.
- I : Segundo momento de inercia.



(a) Coordenadas y dimensiones.



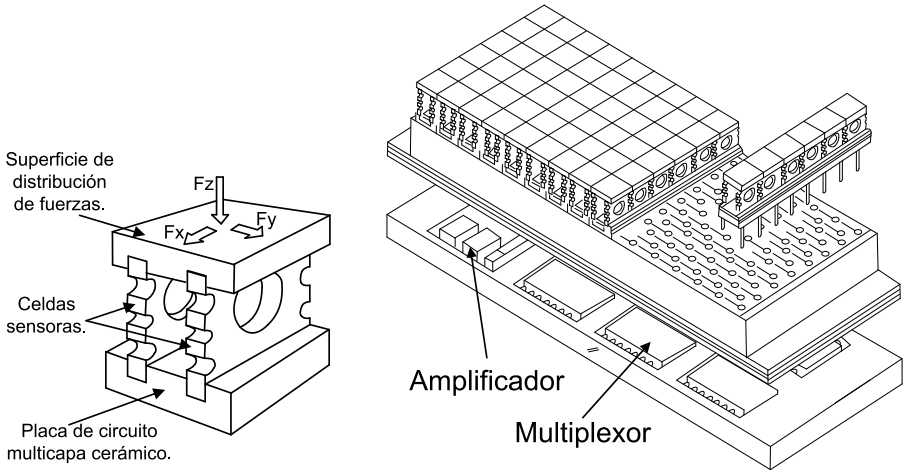
(b) Fuerza horizontal en el eje x. (c) Fuerza horizontal en el eje y. (d) Fuerza vertical.

Figura 2.11: Distribución de la deformación / Estrés circunferencial.

Para realizar la medida se emplean doce galgas extensiométricas fabricadas también mediante tecnología de semiconductores para medir la distribución de la deformación. Cada cuatro extensiómetros forma un Puente de Wheatstone, y tres puentes de Wheatstone son capaces de medir las tres componentes Fx, Fy, Fz. Por tanto el sensor puede medir las tres componentes Fx, Fy, Fz de la fuerza aplicada sin interferencias entre ellas.

La estructura del módulo del sensor táctil se muestra en la figura 2.12a, en concreto se muestran dos celdas, sobre estas una superficie cuadrada de material recibidor de la fuerza para distribuirla en las dos celdas. La parte inferior de la celda está unida mediante dos surcos a la placa del circuito. Las líneas de alimentación y de señal son transmitidas a través de la placa de circuito cerámica multicapa

El array está montado sobre una placa de circuito multicapa de fibra de vidrio reforzada mediante epoxy, cada unidad sensora está unida a la PCB mediante pines y sockets, el array del ejemplo tiene un tamaño de 15×6 módulos sensores. En la otra cara de de la placa de fibra de vidrio se encuentran los multiplexores para seleccionar el módulo sensor



(a) Estructura de un módulo sensor táctil, compuesto de dos celdas sensoras.

(b) Construcción de un array de sensores táctiles.

Figura 2.12: Construcción de un sensor táctil.

con el que se va a medir un amplificador. Estos datos pasan a una unidad de procesado que convierte la medida analógica de los puentes de Wheatstone en valores digitales, rectifica la no linealidad de los sensores y compensa los errores ocasionados por la temperatura. Una vez realizado el procesado de los datos, estos pasan a una unidad de reconocimiento que se encargará de generar la información táctil. Ya sea crear una imagen táctil, calcular momentos, centro de presión...

2.5.2. Medida de fuerzas en tres dimensiones mediante sensores piezorresistivos.

Este sensor táctil, desarrollado por la universidad de Hon Kong [23] consta de cinco capas estructurales: Una superficie de goma, columnas de concentración de fuerza, array sensor, una base de protección y una placa de circuito, como se muestra en la figura 2.13.Una goma de 2mm de espesor se emplea para absorber el impacto y proteger los dispositivos internos. La capa de goma está pegada a la parte superior de las columnas de concentración de fuerza mediante un pegamento en gel de secado lento que es un excelente aislante eléctrico. Las columnas de concentración de fuerza está fabricadas de silicio y posicionadas en el centro de la celda sensora.

Las celdas sensoras tiene una membrana cuadrada "con forma de E" de silicio fabricada mediante tecnología de semiconductores. La presión inducida por las fuerzas aplicadas en la membrana es medida mediante tres grupos de piezorresistores integrados en la membrana de silicio. Mediante Análisis de Elementos Finitos (FEA)[4] de la membrana "con forma de E" se determinan las posiciones óptimas de los piezorresistores para obtener una

[4]FEA: Finite-Element Analysis.

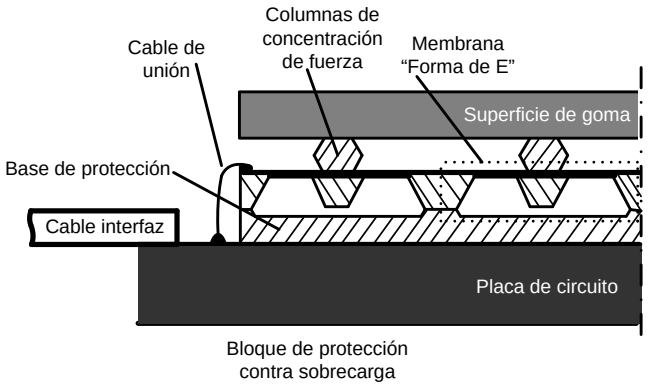
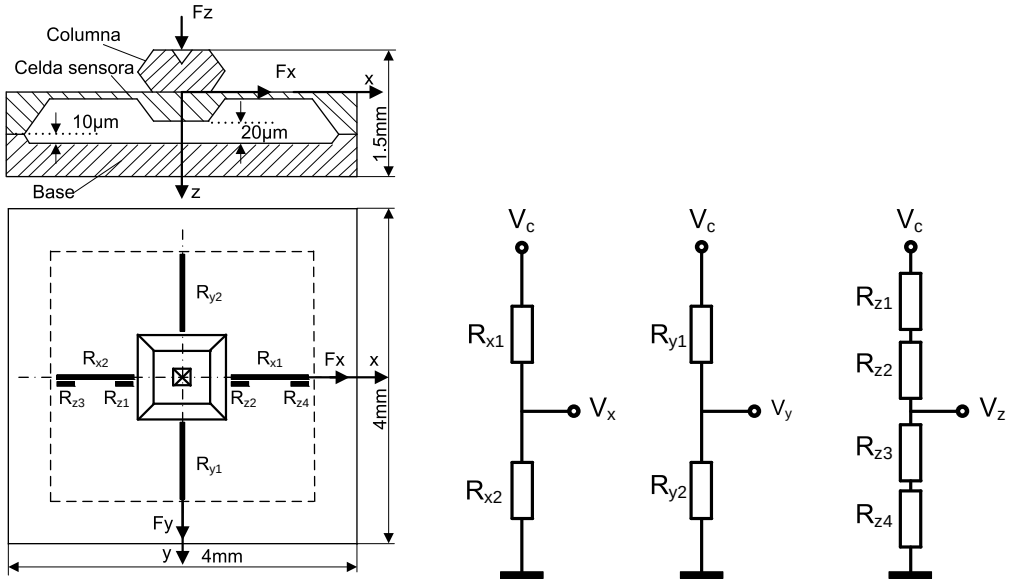


Figura 2.13: Sección de una célula del array tactil.

salida óptima de las fuerzas en los tres ejes Fx,Fy,Fz. El bloque de protección permite un desplazamiento en el eje z máximo de $20\mu m$ lo que equivale a una fuerza de $10N$.



(a) Localización de los sensores.

(b) Diagrama eléctrico para la medida en cada uno de los ejes.

Figura 2.14: Estructura mecánica y eléctrica.

La localización de los piezorresistores en la membrana se muestra en la figura 2.14a. Ante una fuerza en el eje z, las deformaciones en las zonas de la membrana en que se encuentran R_{x1}, R_{x2}, R_{y1} y R_{y2} de modo que las salidas V_x y V_y permanecerá constante ver figura 2.14b. Sin embargo, ante la carga de F_z, R_{z1} y R_{z2} bajo un esfuerzo de compresión y R_{z3} y R_{z4} bajo esfuerzo de de tracción. Así R_{z1} y R_{z2} disminuyen mientras que R_{z3} y R_{z4} aumentan su valor bajo la aplicación de una fuerza Fz. Cuando se aplica cuna fuerza en la dirección del eje x F_x en la celda sensora, R_{x1} se comprime mientras que R_{x2} se elonga y la salida V_x varía con el estímulo Fx. El mismo proceso ocurre con V_y. Ni V_x ni V_y se ven afectadas por las variaciones de la fuerza F_z pues R_{x1}, R_{x2}, R_{y1} y R_{y2} equidistan

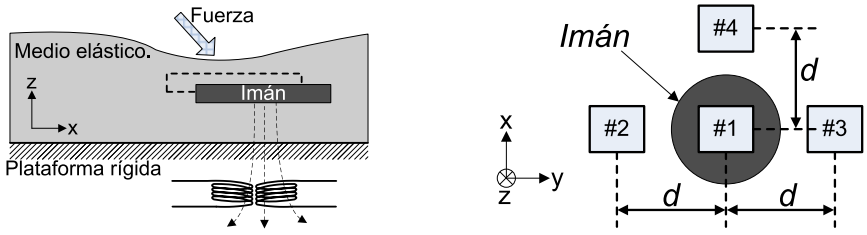
del centro del sensor y se compensan entre ellas. Tampoco V_z se ve afectado por fuerzas F_x o F_y, pues en el primer caso se compensan y en el segundo al ser los piezorresistores direccionales no se ven afectados por las fuerzas F_y.

2.5.3. Sensor táctil de tres ejes basado en la inducción electromagnética.

Satoru Takenawa, investigador del departamento de Ingeniería Mecánica de la Universidad de Kobe, Japón, presenta en [39] un sensor para la medida de fuerzas en tres direcciones basado en el campo magnético. La estructura de este sensor es simple, consiste en un imán permanente y cuatro bobinas encapsuladas. El principio está basado en la inducción electromagnética y se puede implementar con un consumo de corriente muy reducido dado que sólo se utilizan elementos pasivos.
La estructura del sensor se muestra en la figura 2.15a y b. La tensión generada por cada bobina es proporcional al vector fuerza aplicado arbitrariamente el cual deforma el cuerpo elástico y produce un desplazamiento del imán. La amplitud de la tensión V_{out}viene dada por la fórmula

$$V_{out} = -NS\frac{\partial B_z}{\partial t} \tag{2.11}$$

donde N y S son el número de espiras de chip inductor por unidad de longitud y el area de una espira respectivamente.



(a) Sección de una celda sensora.

(b) Situación de bobinas e imán en una celda.

Figura 2.15: Estructura de un sensor.

El sensor la información del vector de fuerza utilizando elementos pasivos es por eso que este sensor tiene un consumo muy bajo y aunque está pensado para ser utilizado en las puntas de los dedos de los robots es extensible a muchas otras zonas del cuerpo debido como ya se ha dicho a su bajo consumo y al bajo coste.

Capítulo 3

Desarrollo teórico.

Se pretende realizar la medida de las fuerzas que actúan sobre la suela del pié del robot humanoide midiendo la deformación que sufre una plancha de materia elástico. Ésta tiene un doble objetivo, por un lado ser el soporte o medio físico donde se puede realizar la medida y por el otro actuar como amortiguación en la caminata absorbiendo pequeñas irregularidades de terreno.
En el capítulo anterior se han encontrado varios sistemas de medida para realizar medidas similares (Pressure Profile Systems, Inc. e Inaba Instries Inc...). Aunque las principales limitaciones que estos han tenido son que algunos solo miden fueras perpendiculares a la superficie en la que se encuentran, y otros que sí son capaces de medir fuerzas en los tres ejes no soportan las fuerzas a las que se ve sometida la suela del pie de un robot humanoide. Es por eso que se ha decidido realizar el diseño de un nuevo sistema sensor.

En general, la estructura de un sensor completo se compone de lo siguiente:

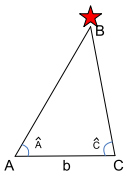
- Controlador del transductor. Se encarga de que se consiga la actuación que el usuario quiere.
- Transductor. Convierte las variaciones de una magnitud eléctrica (señal) en variaciones de una física.
- Acondicionamiento de la señal. Si existe, realiza la función de modificar la señal entregada al transductor para obtener una señal adecuada (amplificación, modulación, etc.). Con el avance de la electrónica digital, cada vez los circuitos acondicionadores son más sencillos.

La idea inicial es emplear tres sensores de campo magnético y un objeto que genere un campo magnético. Así, el el objeto que genera el campo magnético será el testigo que se mueve solidario a las deformaciones del material elástico y los sensores fijos. Al proporcionar estos una tensión de salida proporcional a la densidad de flujo magnético que atraviesa su superficie y estando la densidad de flujo relacionada con la distancia a la que se encuentra el imán. La salida de estos sensores estaría vinculada con la distancia al testigo, combinando las tres distancias proporcionadas por los sensores se podría dar una posición del testigo (Coordenadas xyz del testigo). Si previamente se han medido las

coordenadas del testigo en reposo se podrá conocer la dirección, sentido y magnitud del desplazamiento mediante la resta de estas dos posiciones y de ahí los mismos datos de fuerza si aplicamos la ley de Hooke a un material elástico. La fuente de campo magnético será un imán y los sensores de campo magnético sensores de Efecto Hall.

3.1. De la triangulación a la trilateración.

3.1.1. La Triangulación.



La triangulación consiste en el uso de la trigonometría para la medida de distancias y la determinación de la posición de puntos [44]. Así, trabajando en un plano y conociendo la posición de dos puntos y el ángulo que forman con un tercero se puede conocer la posición de ese tercer punto. Por desgracia, la medida de ángulos es algo que no es posible de forma directa mediante sensores hall. Esto nos lleva un paso más allá y a la búsqueda de otro método para la determinación de la posición de un objeto. La trilateración.

3.1.2. La trilateración en el plano (2D).

La trilateración, también conocida como trilateralización consiste en la determinación de la posición de un objeto en función de las distancias a puntos conocidos [45]. También se basa en la geometría de triángulos pero esta vez la medida se realiza mediante distancias en lugar de ángulos, valores que sería posible medir con un sensor Hall. Supongamos que se dispone de dos puntos $O_1 = (0, 0)$ y $O_2 = (0, y_1)$ y dos distancias de cada punto al objeto que se quiere popsicionar, r_1 y r_2 conocidas. Entonces podemos decir que el punto deseado se encuentra en el lugar geométrico de los puntos del plano que cumple las siguientes dos condiciones: dista una distancia r_1 de O_1 y una distancia r_2 de O_2. Esto se puede interpretar tanto matemática como gráficamente mediante la intersección de dos circunferencias.

$$r_1^2 = x^2 + y^2 \tag{3.1}$$

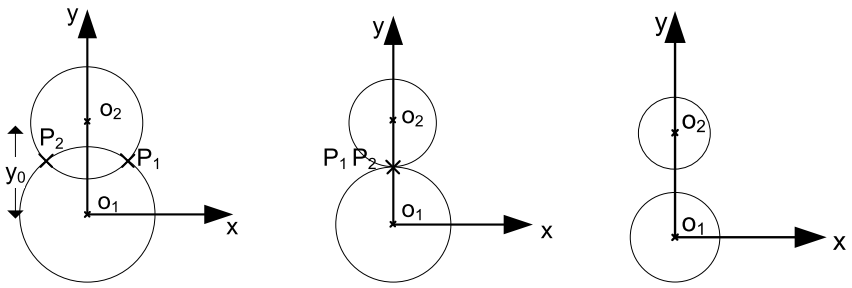
$$r_2^2 = x^2 + (y - y_0)^2 \tag{3.2}$$

Las ecuaciones 3.1 y 3.2 representan matemáticamente la posición y medida de dos sensores en un plano. Son dos circunferencias cuyo centro representa la posición del sensor y el radio la medida.

De la resolución de este sistema (Intersección de dos circunferencias) da lugar a tres posibles casos.

- **Caso1.** $|y_0| < |r_1+r_2|$ además $|r_1-r_2| < |y_0|$ las soluciones son dos puntos posibles, siendo P_1 y P_2, para discriminar uno de ellos se pueden seguir dos estrategias, la primera de ellas una estrategia de restricción física, por ejemplo, el punto solo puede situarse en lo puntos del plano cuya coordenada $x > 0$, la otra opción en emplear un tercer punto de medición.

- **Caso2.** $r_1 + r_2 = y_0$ Sólo se cortan en un punto.



(a) Caso 1. (b) Caso 2. (c) Caso 3.

Figura 3.1: Posibles soluciones del sistema ecuación 3.1 y ecuación 3.2.

$$y = \frac{r_1^2 - r_2^2 + y_0^2}{2 \cdot y_0} \tag{3.3}$$

$$x = \pm\sqrt{r_1^2 - y^2} \tag{3.4}$$

Las ecuaciones 3.3 y 3.4 representan la solución al sistema de ecuaciones 3.1 y 3.2 para los casos 1 y 2.

- **Caso3.** $r_1 + r_2 < y_0$ ó $|r_1 - r_2| > y_0$ Las circunferencias no se cortan y no hay solución.

Si las medidas realizadas son correctas es imposible que aparezca el caso 3 pues implica que se han medido distancias diferentes desde cada uno de los puntos de referencia. Mediante la técnica anterior es posible realizar el posicionamiento de un punto en un plano, pero que ocurre si se quiere posicionar un punto en el espacio.

3.1.3. La trilateración en el espacio (3D).

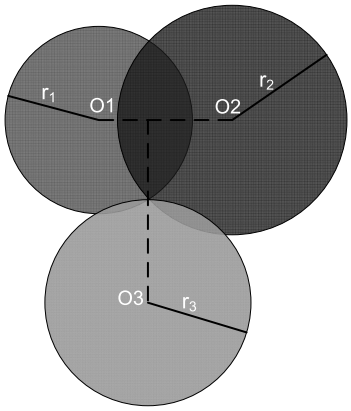


Figura 3.2: Trilateración 3D tres esferas.

En este caso la trilateración emplea la posición conocida de 3 puntos de referencia, es decir, la posición de los tres sensores efecto hall y se pretende conocer la posición del imán, para ello cada uno de los sensores proporcionará una distancia, con un sensor situado en O_1 y una distancia r_1, las posibles posiciones del imán se reducen al lugar geométrico de los puntos del espacio que dista del primer sensor una distancia r_1. Esto es una esfera de radio r_1 con centro en O_1.
Con el segundo sensor hall se obtienen los mismos resultados, así pues se consigue una segunda esfera con centro en O_2 y de radio r_2. De la intersección de las dos esferas anteriores se pueden dar tres casos: que las dos esferas se corten, que solo se toquen en un punto o que no lleguen a tocarse en ningún punto.

- **Caso1.** Las dos esferas se cortan generando así una circunferencia en el espacio, de ese modo la posición del testigo se limita de una esfera a una circunferencia. Es el caso general y con el que se seguirá trabajando. Esto ocurre cuando $|\overrightarrow{O_2} - \overrightarrow{O_1}| < |r_1 - r_2|$ además $|r_1 - r_2| = |\overrightarrow{O_2} - \overrightarrow{O_1}|$.

- **Caso2.** Las dos esferas se tocan en un punto, así la posición del testigo queda totalmente determinada. Esto solo ocurre cuando $|\overrightarrow{O_2} - \overrightarrow{O_1}| = |r_1 + r_2|$ o que $|r_1 - r_2| = |\overrightarrow{O_2} - \overrightarrow{O_1}|$. Es decir, que la distancia entre los centros de las dos esferas es igual a la suma de los radios de las mismas (tangente exterior), o que la deferencia entre los centros es igual a la suma de los radios (tangente interior).

- **Caso3.** Las dos esferas nunca se cortan. Este ultimo caso ocurre si $|\overrightarrow{O_2} - \overrightarrow{O_1}| > |r_1 + r_2|$ ó $|r_1 - r_2| > |\overrightarrow{O_2} - \overrightarrow{O_1}|$, es decir, que están demasiado lejos para cortarse o que una de las esferas está contenida dentro de la otra.

El caso general ocurre cuando las dos esferas se cortan. En este caso la posición del testigo ha quedado reducida a una circunferencia. Mediante la utilización de un tercer sensor situado en O_3 obtendremos una distancia r_3 y con ello una tercera esfera de centro O_3 y radio r_3. La intersección entre la circunferencia resultante de los dos puntos anteriores con la nueva esfera situada en O_3 puede diferenciarse en tres posibles casos, aunque nos centraremos en el caso en que se cortan en dos puntos. De ese modo la localización del testigo se ha reducido a dos posibles posiciones bien diferenciadas entre si, ver figura 3.3. En ese caso, por las restricciones físicas de la medida se podría desechar uno de los puntos por no estar situado dentro del campo de movimiento posible. A continuación se presenta

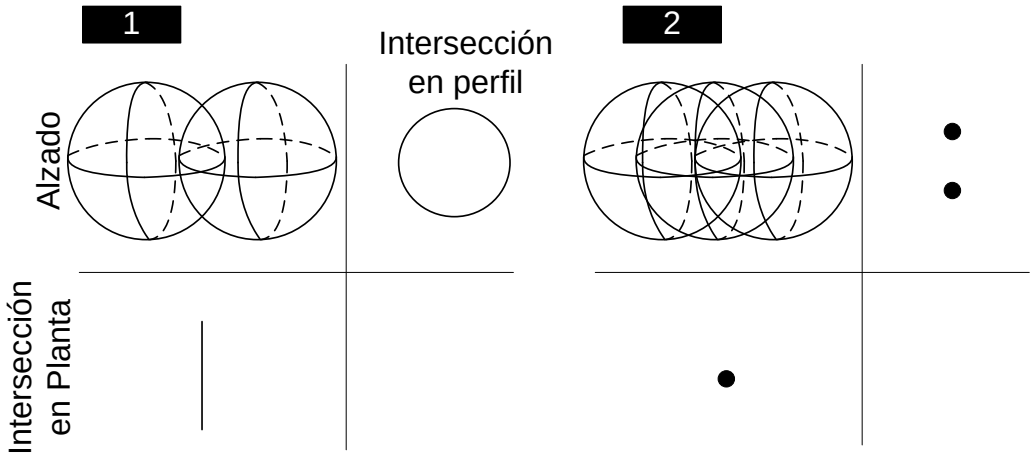


Figura 3.3: A la izquierda la intersección de 3 esferas genera una circunferencia en el espacio. A la derecha, la intersección con una tercera esfera genera dos puntos en el espacio.

la representación matemática de la posición de los tres sensores y sus medidas devueltas. Que se modela con tres esferas.

$$r_1^2 = x^2 + y^2 + z^2 \tag{3.5}$$

$$r_2^2 = (x - O_{2x})^2 + y^2 + z^2 \tag{3.6}$$

$$r_3^2 = (x - O_{3x})^2 + (y - O_{3y})^2 + z^2 \tag{3.7}$$

Donde:
$r_1, r_2, r_3 \rightarrow$ Radio de cada una de las esferas. Medida devuelta por los sensores.
$O_1 = (0, 0, 0) \rightarrow$ Centro de la esfera 1. Posición del sensor 1.
$O_2 = (O_{2x}, 0, 0) \rightarrow$ Centro de la esfera 2. Posición del sensor 2.
$O_3 = (O_{3x}, O_{3y}, 0) \rightarrow$ Centro de la esfera 3. Posición del sensor 3.

La solución del sistema formado por las ecuaciones 3.5 a 3.7 tiene como resultado la posición del testigo.

$$x = \frac{r_1^2 - r_2^2 + O_{2x}^2}{2 \cdot O_{2x}} \tag{3.8}$$

Al sustituir esta solución en la ecuación 3.5 se obtiene la ecuación de una circunferencia (ecuación 3.9), que es la solución a la intersección de las esferas representadas por las ecuaciones 3.5 y 3.6.

$$x^2 + y^2 = r_1 - \frac{(r_1^2 - r_2^2 + O_{2x}^2)^2}{4 \cdot O_{2x}^2} \tag{3.9}$$

Valiéndonos de la tercera esfera (ecuación 3.7) se pueden obtener las coordenadas x e y del testigo (ecuaciones 3.10 y 3.11).

$$y = \frac{r_1^2 - r_3^2 + O_{3x}^2 + O_{3y}^2}{2 \cdot O_{3y}^2} - \frac{O_{3x}}{O_{3y}} \cdot x \tag{3.10}$$

$$z = \sqrt{r_1^2 - x^2 - y^2} \tag{3.11}$$

Esta solución es correcta siempre que se parta de las condiciones para las que se ha calculado, es decir, que las tres esferas tengan su centro en el plano $z = 0$, de manera más específica, que el centro de la esfera 1 se encuentre en el origen $O_1 = (0, 0, 0)$, que el centro de la esfera 2 se encuentre a lo largo del eje $x =_2= (O_{2x}, 0, 0)$ y que el centro de la esfera 3 se encuentre sobre el plano $z = 0 =_3= (O_{3x}, O_{3y}, 0)$. Para conseguirlo se pueden aplicar sucesivas transformaciones en el espacio (Rotaciones y traslaciones). Cabe destacar que al estar trabajando con esferas no van a ser necesarias las transformaciones de rotación, únicamente de traslación para desplazar los centros a O_1 O_2 y O_3

3.1.4. Traslación de un sistema de coordenadas.

La traslación de un sistema de coordenadas con respecto a otro se representa mediante la suma de vectores [21]. Si el sistema UVW coincide inicialmente con el sistema XYZ y se traslada el sistema UVW una distancia $\overrightarrow{d}$ con respecto de XYZ entonces los ejes de UVW permanecen paralelos a los de XYZ.
Sea un punto P situado inicialmente en el sistema de coordenadas UVW con coordenadas:

$$P_{UVW} = u \cdot i_u + v \cdot j_v + w \cdot k_w \tag{3.12}$$

Después de trasladar el sistema UVW un vector $\overrightarrow{d} = (d_1 \cdot i_x + d_2 \cdot j_y + d_3 \cdot K_z)$ con respecto al sistema XYZ. Las coordenadas del punto P con respecto de UVW se mantienen mientras que con respecto de XYZ se convierten en:

$$P_{XYZ} = (u + d_1)i_x + (v + d_2)j_y + (w + d_3)k_z \tag{3.13}$$

Esta igualdad puede ser expresada también de forma matricial.

$$\begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} \tag{3.14}$$

A la izquierda de la igualdad se representan las coordenadas del punto P en es sistema UVW. Así se puede expresar:

$$P_{XYZ} = T_r \cdot P_{UVW} \tag{3.15}$$

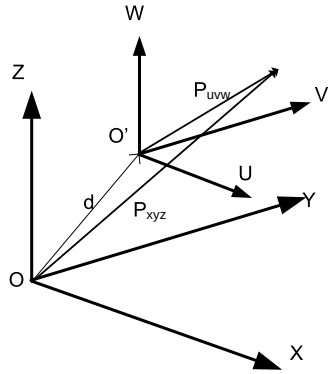


Figura 3.4: Traslación de un sistema de coordenadas UVW respecto de un sistema fijo XYZ

De este modo se puede conocer la posición del testigo respecto del sistema de referencia deseado en lugar del sistema de referencia utilizado para simplificar las operaciones matemáticas. Lo que permite que en el caso de tener un conjunto sensor con un número n de celdas sensoras, conocida la posición y orientación de cada sistema de referencia de la celda y la posición del testigo con respecto a la celda, es posible conocer la posición de los n testigos con respecto a un único sistema de referencia común para todas las celdas.

3.2. Estudio del campo magnético generado por un imán.

Los sensores efecto hall miden la densidad de flujo magnético B perpendicular a su plano, es por eso que se pretende encontrar una ecuación que muestre la densidad de flujo magnético B en un punto arbitrario del espacio. La unidad del sistema internacional es el Tesla $[T]$

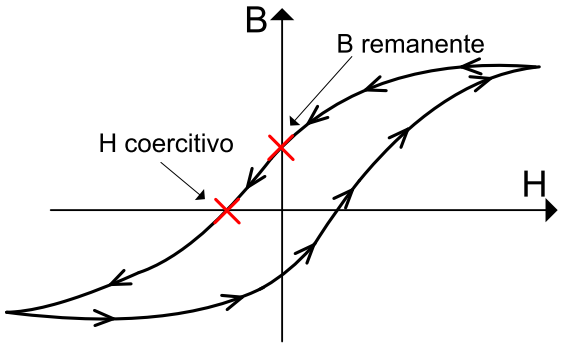


Figura 3.5: Curva B-H de un imán.

Como se observa en la figura 3.5, la remanencia es una medida de la densidad de flujo magnético que permanece en el imán tras su magnetización [40]. Su unidad del Sistema Internacional es el Tesla. La relación $Tesla$-$Gauss$ se define con la ecuación 3.16

$$1T = 10^4 Gauss \tag{3.16}$$

La coercitividad se define como la intensidad de campo necesaria para volver a desmagnetizar un imán. La unidad para medir la intensidad de campo magnético es el $[A/m]$ (Amperio por metro).

El campo magnético generado por un imán con momento magnético m en cualquier punto del espacio no es trivial pero se puede estudiar como el campo magnético generado por un solenoide por el que circula una corriente equivalente I_{eq} como se muestra en [40] con la siguiente relación:

$$I_{eq} = \frac{m}{\pi \cdot a^2} \tag{3.17}$$

Donde a es el radio del imán.

3.2.1. Campo magnético generado por una espira de corriente en cualquier punto del espacio.

El objetivo final es calcular el campo magnético generado por un solenoide, para ello se parte del campo generado por un elemento del solenoide, es decir, por la espira. Ver figura 3.6.

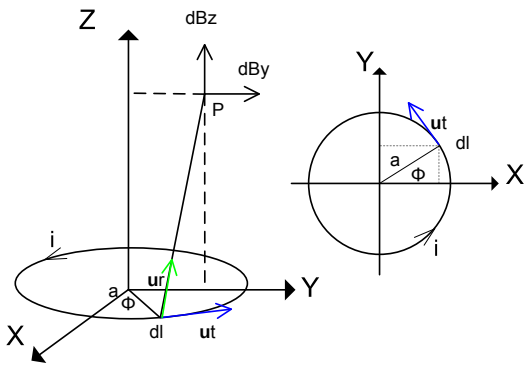


Figura 3.6: Espira de corriente por la que circula una corriente i.

La ley de Biot-Savart en el caso de corrientes distribuidas en volúmenes afirma que la contribución al campo magnético de cada elemento de longitud de la distribución, viene dado por:

$$B = \frac{\mu \cdot i}{4 \cdot \pi} \cdot \oint \frac{u_t \times u_r}{r^3}\, dl \tag{3.18}$$

Donde μ_0 es la permeabilidad del vacío
$mu_0 = 4\pi \cdot 10^-7[N \cdot A^-2]$ conocida también como constante magnética, dl es el elemento de corriente al que u_t es tangente y u_r es el vector unitario señalando la dirección del punto en el que se quiere calcula el campo magnético y r la distancia al punto P. Todo ello en unidades del sistema internacional.

Dada la simetría de la espira, en cualquier punto del espacio se observa que el campo magnético B tiene simetría axial, es decir, $B_x = 0$ y será posible mediante desplazamientos y giros situar el punto P en $(0, y, z)$, es decir, en el plano YZ colocando la espira en el plano XY con centro en $(0, 0, 0)$.
A continuación se expresarán r u_r u_t en función de parámetros que se puedes observar en la figura 3.6

$$r = \sqrt{a^2 + y^2 + z^2 - 2 \cdot a \cdot y \cdot \sin\theta} \tag{3.19}$$

$$u_r = \frac{-a \cdot \cos\theta\, \overrightarrow{i} + (y - a \cdot \sin\theta)\, \overrightarrow{j} + z\, \overrightarrow{k}}{r} \tag{3.20}$$

$$u_t = -\sin\theta\, \overrightarrow{i} + \cos\theta\, \overrightarrow{j} \tag{3.21}$$

Donde:
$a \rightarrow$ Radio de la espira de corriente.
$\theta \rightarrow$ Ángulo que forma el eje x con el segmento que une el origen y dl.

Calculando el producto vectorial y teniendo en cuenta que $dl = a \cdot d\theta$ resultan las tres componentes del campo magnético en el punto P.

$$B_x = \frac{\mu_0 \cdot i \cdot a \cdot z}{4 \cdot \pi} \cdot \int_0^{2\pi} \frac{\cos\theta}{r^3}\, d\theta \tag{3.22}$$

$$B_y = \frac{\mu_0 \cdot i \cdot a \cdot z}{4 \cdot \pi} \cdot \int_0^{2\pi} \frac{\sin\theta}{r^3}\, d\theta \tag{3.23}$$

$$B_z = \frac{\mu_0 \cdot i \cdot a}{4 \cdot \pi} \cdot \int_0^{2\pi} \frac{a - y \cdot \sin\theta}{r^3}\, d\theta \tag{3.24}$$

Anteriormente ya se ha dicho que existe una simetría axial en el problema y que por lo tanto el campo magnético en la dirección del eje x es 0.
De modo que solo existen dos componentes del campo, una en la dirección del eje radial B_y y otra en la dirección del eje central B_z. Sustituyendo r (ver la ecuación 3.19) por su valor en las ecuaciones 3.23 y 3.24 se obtiene:

$$B_y = \frac{\mu_0 \cdot i \cdot a \cdot z}{2\pi} \cdot \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{\sin\theta}{(a^2 + z^2 + y^2 - 2 \cdot a \cdot y \cdot sin\theta)^{3/2}}\, d\theta \tag{3.25}$$

$$B_z = \frac{\mu_0 \cdot i \cdot a}{2\pi} \cdot \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{a - y \cdot \sin\theta}{(a^2 + z^2 + y^2 - 2 \cdot a \cdot y \cdot sin\theta)^{3/2}}\, d\theta \tag{3.26}$$

Para simplificar la representación de estas integrales designaremos $b = \frac{a^2+z^2+y^2}{2 \cdot a \cdot y}$ y se hace el cambio de variable $\theta = \frac{\pi}{2} - \varphi$; entonces $d\theta = -d\varphi$. Así B_y B_z quedan como sigue:

$$B_y = \frac{\mu_0 \cdot i \cdot a \cdot z}{2\pi} \cdot \frac{1}{(2a \cdot y)^{3/2}} \cdot \int_{\pi}^{0} \frac{\cos\varphi}{(b - \cos\varphi)^{3/2}} \cdot (-d\varphi)$$

$$B_y = -\frac{\mu_0 \cdot i \cdot a \cdot z}{2\pi \cdot (2a \cdot y)^{3/2}} \cdot \int_{0}^{\pi} \frac{-\cos\varphi}{(b - \cos\varphi)^{3/2}} \cdot (-d\varphi) \tag{3.27}$$

$$B_z = \frac{\mu_0 \cdot i \cdot a}{2\pi} \cdot \frac{1}{(2a \cdot y)^{3/2}} \cdot \int_{\pi}^{0} \frac{a - y \cdot \cos\varphi}{(b - \cos\varphi)^{3/2}} \cdot (-d\varphi)$$

$$B_z = \frac{\mu_0 \cdot i \cdot a}{2\pi \cdot (2a \cdot y)^{3/2}} \cdot \left(a \cdot \int_0^{\pi} \frac{d\varphi}{(b - \cos\varphi)^3/2} + y \cdot \int_0^{\pi} \frac{-\cos\varphi \cdot d\varphi}{(b - cos\varphi)^3/2} \right) \tag{3.28}$$

Las integrales de la Ecuación 3.27 y 3.28 se pueden expresar en función de integrales elípticas de primera y segunda especie.
Siendo $K_{(m)}$ la integral elíptica de primera especie y $E_{(m)}$ la integral elíptica de segunda especie definidas como:

$$K_{(m)} = \int_0^{\frac{\pi}{2}} \frac{d\phi}{\sqrt{1 - m \cdot \sin^2\phi}} \tag{3.29}$$

$$E_{(m)} = \int_0^{\frac{\pi}{2}} \sqrt{1 - m \cdot \sin^2\phi} \cdot d\phi \tag{3.30}$$

Y expresando las integrales 3.27 y 3.28 en función de las integrales elípticas anteriores queda:

$$\int_0^{\pi} \frac{d\varphi}{(b - \cos\varphi)^{3/2}} = \frac{m}{2 - 2m} \cdot \sqrt{2m} \cdot E_{(m)} \tag{3.31}$$

$$\int_0^{\pi} \frac{-\cos\varphi \cdot d\varphi}{(b - \cos\varphi)^{3/2}} = \sqrt{2m} \cdot K_{(m)} - \frac{2 - m}{2 - 2m} \cdot \sqrt{2m} \cdot E_{(m)} \tag{3.32}$$

Con $m = \frac{2}{1+b}$
Finalmente el campo magnético en cualquier punto del espacio en sus tres direcciones se puede definir como:

$$B_x = 0 \tag{3.33}$$

$$B_y = \frac{\mu_0 \cdot i \cdot a}{2\pi \cdot (2a \cdot y)^{3/2}} \cdot z \cdot \left(-\sqrt{2m} \cdot K_{(m)} + \frac{2-m}{2-2m} \cdot \sqrt{2m} \cdot E_{(m)} \right) \tag{3.34}$$

$$B_z = \frac{\mu_0 \cdot i \cdot a}{2\pi \cdot (2a \cdot y)^{3/2}} \cdot \left(a \cdot \frac{m}{2-2m} \cdot \sqrt{2m} \cdot E_{(m)} + y \cdot \sqrt{2m} \cdot K_{(m)} - y \cdot \frac{2-m}{2-2m} \cdot \sqrt{2m} \cdot E_{(m)} \right) \tag{3.35}$$

Con $m = \frac{2}{1+b} = \frac{4a \cdot y}{a^2+z^2+y^2+2a \cdot y}$ Se ha realizado todo el desarrollo en el vacío, pero lo cierto es que entre el imán y el sensor no va a haber vacío, va a haber latex, cuando se estudie el comportamiento magnético del latex, es decir, si es un material paramagnético (no magnéticos), cuya permeabilidad relativa μ_r es aproximadamente 1 (se comportan como el vacío) o diamagnético, cuya $\mu_r < 1$ es necesario multiplicar todas las ecuaciones por μ_r en el caso de que no sea un material paramagnético.
Por otro lado se ha calculado el campo magnético tanto para el eje y como para el eje z sin embargo la disposición del sensor hall solo permite la medida de B_z por lo que a partir de ahora se continuará trabajando únicamente con esa componente del campo magnético.

3.2.2. Campo magnético generado por un imán o un solenoide en cualquier punto del espacio.

Para calcular el campo magnético creado por un solenoide en cualquier punto del espacio se parte de la fórmula de cálculo del campo magnético generado por una espira en cualquier punto del espacio y se integra a lo largo de toda la longitud del solenoide. Es por eso que en el apartado 3.2.1 se ha considerado necesario realizar el cálculo del campo generado por la espira.

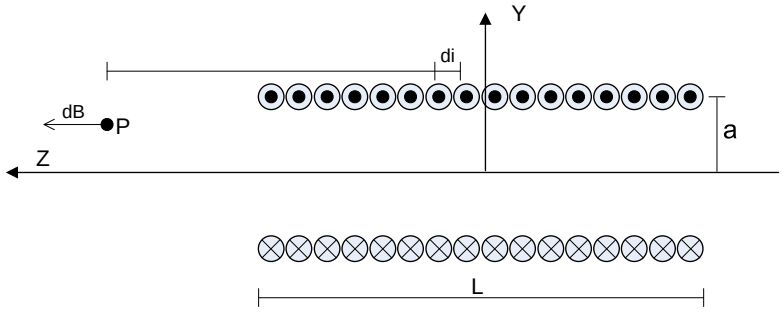


Figura 3.7: Campo generado en la dirección del eje z por un solenoide en un punto P.

El campo magnético dB_z debido a un elemento de corriente dI_{eq} es el siguiente a partir de la ecuación 3.26

$$dB_z = \frac{\mu_0 \cdot dI_{eq} \cdot a}{2\pi} \cdot \left(\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{a - y \cdot \sin\theta}{(a^2 + z^2 + y^2 - 2a \cdot y \cdot \sin\theta)^{3/2}} \cdot d\theta \right) \tag{3.36}$$

Si se hace el cambio de variable $dI_{eq} = \frac{I_{eq}}{L} \cdot dz$ y se aplica la igualdad de la ecuación 3.17.

$$dB_z = \frac{\mu_0 \cdot m}{2\pi^2 \cdot L \cdot a} \cdot \left(\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{a - y \cdot \sin\theta}{(a^2 + z^2 + y^2 - 2a \cdot y \cdot \sin\theta)^{3/2}} \cdot d\theta \right) \cdot dz \tag{3.37}$$

Integrando a lo largo de toda la longitud del solenoide queda:

$$B_z = \frac{\mu_0 \cdot m}{2\pi^2 \cdot L \cdot a} \cdot \int_{z-L/2}^{z+L/2} \left(\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{a - y \cdot \sin\theta}{(a^2 + z^2 + y^2 - 2a \cdot y \cdot \sin\theta)^{3/2}} \cdot d\theta \right) \cdot dz \tag{3.38}$$

Estas integrales se hacen de manera numérica con $MatLab^{®}$ o cualquier otro lenguaje de programación pues son integrales elípticas.

3.2.3. Análisis del campo magnético.

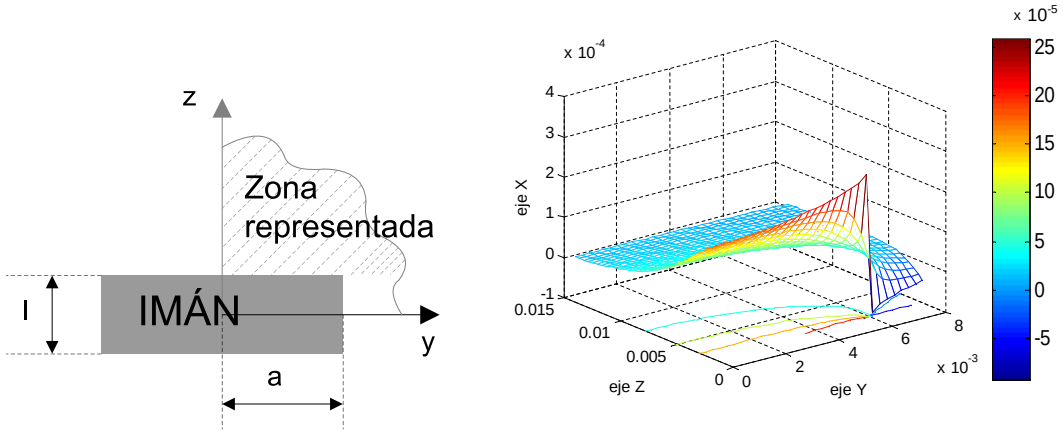
Una vez obtenida la ecuación matemática para el campo magnético es posible representarlo en una gráfica y realizar diversos análisis. Para ello se han creado varias funciones en $MatLab^{®}$ que es posible consultar en los Anexos.

La función que se va a emplear a su vez llama al resto de funciones creadas. De este modo, dados los parámetros del imán representa el campo magnético en una zona del espacio. $function\ grafic\_3D\_mag\_z(tamano\_z, tamano\_y, mu0, m, l, a, numero\_de\_muestras)$[1]

- $tamano\_z \rightarrow$ Es el tamaño que va a tener la representación del campo magnético medida desde el extremo del imán $[m]$.
- $tamano\_y \rightarrow$ Tamaño que va a tener la representación del campo magnético desde el eje del imán hacia el exterior de forma radial $[m]$.
- $mu0 \rightarrow$ Permeabilidad del vacío $4\pi \cdot 10^{-7}$ $[T.m/A]$ ó $[N/A^2]$.
- $m \rightarrow$ Momento magnético del imán $[A * m^2]$.
- $l \rightarrow$ Longitud del imán $[m]$.
- $a \rightarrow$ Radio del imán $[m]$.

De modo que el imán se va a simular teniendo un radio de $6mm$, una longitud de $4mm$ y una corriente equivalente de $2A$, por lo que tendrá un momento magnético de $2{,}262 \cdot 10^{-4}$ $[A * m^2]$ (ver ecuación 3.17) y se pretende representar hasta una distancia del plano del imán de $12mm$ y una distancia al eje de $8mm$. Realizando la llamada a la función con los parámetros que se muestran a continuación se obtiene el gráfico de la figura 3.8b. Se deben puntualizar algunos aspectos en la figura 3.8 como la colocación de los ejes, la zona representada y la magnitud medida.

[1] Se ha decidido no emplear la letra ñ en el código para evitar cualquier problema de compatibilidad.



(a) Vista lateral del imán, definición de la zona representada en la gráfica

(b) Representación del módulo del campo magnético en la dirección del eje Z para puntos del plano $y - z$ cercanos al imán.

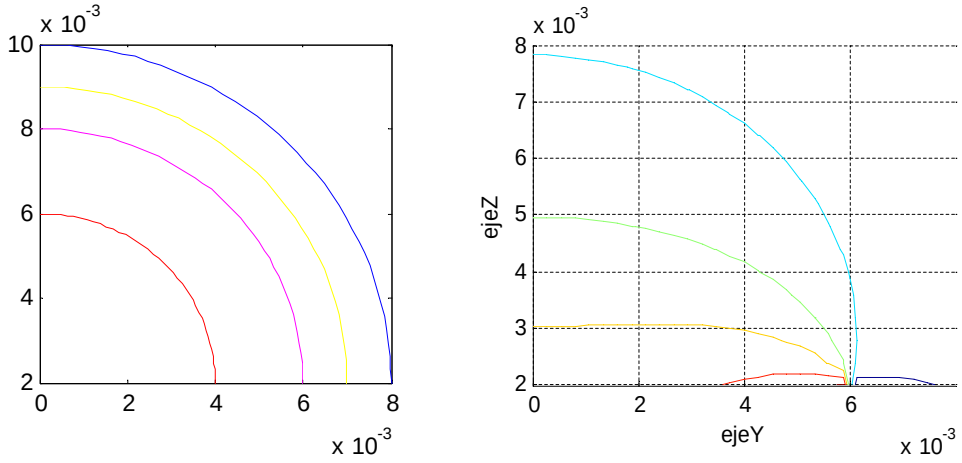
Figura 3.8: Representación del campo magnético en la dirección del eje z

- Colocación de los ejes: En la figura 3.8a se indica que el eje y tiene dirección radial al imán y origen en el centro del imán, mientras que el eje z tiene la dirección del eje del imán y origen en la mitad de la longitud del mismo.
- Zona representada: Anteriormente se ha propuesto representar un área de $8mm \times 12mm$, en la figura 3.8b se observa que el campo se ha representado en el eje y $8mm$ y en el eje z desde $2mm$ hasta $14mm$ dado que la distancia de $2mm$ del origen es donde finaliza el imán.
- La magnitud medida es el módulo del campo magnético en la dirección del eje z. Y está representada en el eje x. Es decir, $B_{(x,y)} : \mathbb{R}^2 \rightarrow \mathbb{R}$ La función campo magnético creada va de $\mathbb{R}^2$ a $\mathbb{R}$

Las curvas de nivel son en este caso la proyección sobre el plano yz de la traza de la superficie que se obtiene del corte de la superficie con el plano $x = k$ donde k es una constante, al conjunto de estas líneas se le conoce como mapa de contorno. En la figura 3.9b se han obtenido las curvas de nivel de la gráfica 3.8b. En la figura 3.9a aparecen las curvas de nivel de una función con una característica peculiar, y es que sus curvas de nivel son circunferencias concéntricas en ese caso es posible, aplicando la técnica de medida descrita en el apartado 3.1.3 Trilateración en el espacio 3D. Sin embargo, las curvas de nivel que proporciona el análisis del campo magnético generado por un imán no tienen esa característica y por lo tanto no permiten la aplicación de la trilateración.

3.3. Estrategia de medida.

Del análisis de las curvas de nivel realizado en el apartado 3.2.3 se ha descubierto y al mismo tiempo demostrado que la estrategia de medida que se ha propuesto en un primer momento (Trilateracilón en el espacio 3D) no es válida Dado que el campo magnético



(a) Curvas de nivel deseadas (b) Curvas de nivel obtenidas.

Figura 3.9: Curvas de nivel.

no tiene una simetría esférica. Sin embargo, sí tiene simetría cilíndrica de modo que es necesario diseñar una nueva estrategia de medida basada en la trilateración en el plano. Para ello se han tenido en cuenta las siguientes consideraciones:

- El campo magnético generado por un imán en el espacio se obtiene con un gran coste computacional.
- El campo magnético generado por un imán ha quedado bien definido en los apartados anteriores.
- Se está trabajando en un espacio tridimensional.
- El algoritmo se va a implementar en un PIC y por tanto existen limitaciones respecto a potencia de cálculo.

La idea es generar en un PC una matriz de m filas por n columnas de modo que en cada posición se guardará el valor del campo magnético generado por el imán en ese punto. Donde m estará relacionado con la distancia perpendicular al plano del imán y n con la distancia perpendicular al eje del imán.

Con la obtención de esta matriz se ha evitado tener que realizar los cálculos de las integrales 3.28 y 3.35 cada vez que el programa necesite calcular el campo magnético en un punto.

La medida consistirá así en un algoritmo de optimización ciega. Se tienen tres sensores hall en un mismo plano, y en posiciones conocidas. Al tomar una medida estos sensores proporcionan el campo magnético en cada uno de los tres puntos. Obtenida la medida del campo magnético se pretenden resolver dos incógnitas: la distancia del plano de sensores al plano del imán, y la distancia de cada uno de los sensores al eje del imán. Se propone parte de la primera solución posible, y es que la distancia al plano del imán sea máxima.

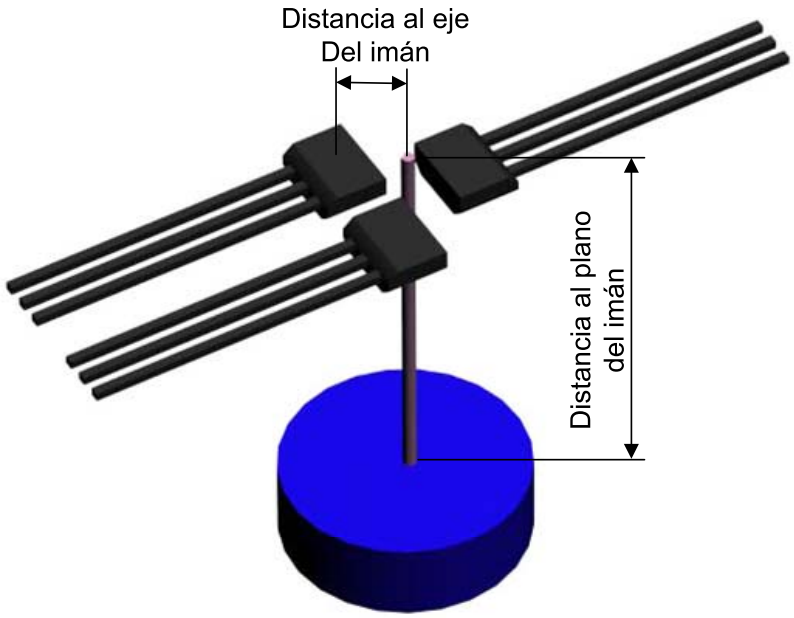
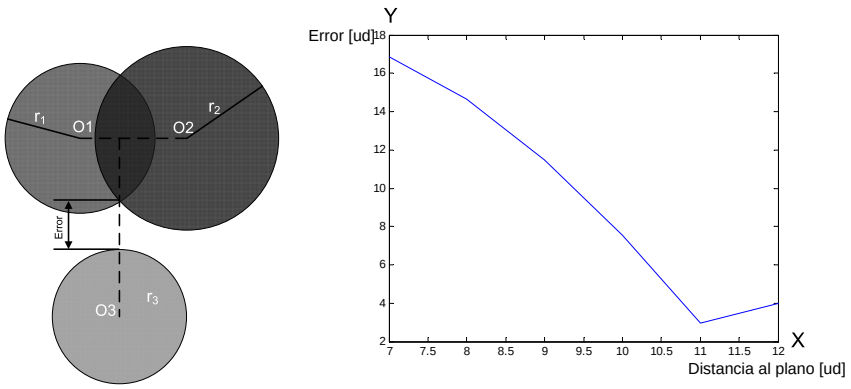


Figura 3.10: Distribución física del imán y los sensores efecto Hall. Incógnitas de medida.

Esto restringe la búsqueda de datos en la matriz a una sola fila, la fila m. En esta fila para cada valor de campo magnético medido corresponderá con una distancia al eje, es decir, una posición en la fila entre 1 y n, ó entre 0 y $(n-1)$ según el lenguaje de programación empleado. A estas alturas se dispone de un plano (el plano que dista una distancia m del plano del imán), tres puntos, que son los puntos en los que se sitúan cada uno de los sensores de Efecto Hall y tres distancias, calculadas mediante la lectura de la matriz en la fila m. De modo que con dos de los sensores y sus dos respectivas distancias establecemos la posición del eje del imán, y con el tercer sensor y tercera distancia se medirá el error cometido. En el mejor de los casos, el eje del imán distará del sensor tres una distancia R_3, entonces el error será cero, pero eso es algo que por norma general no va a ocurrir. El modo de proceder es realizar los pasos anteriores ahora con una distancia al plano del imán de $(m-1)$ y calcular el error. Volver a repetirlo con una distancia $(m-2)\ldots$ y así hasta una distancia 1. En definitiva, realizando una abstracción sería recorrer la matriz por cada una de sus filas desde m hasta 1, comprobando el error que devuelve al cotejar los datos proporcionados por los tres sensores con la información que se encuentra en cada fila de la matriz.

Comparando cada uno de los errores se observa una característica muy importante que en una gráfica se puede apreciar de forma clara(Ver figura 3.11). Si se representa en el eje x la distancia al plano del imán y en el eje y el módulo del error calculado anteriormente se observa un mínimo relativo de la función entre 1 y m. En el mínimo el error cometido habrá sido mínimo valga la redundancia y se puede tomar como medida válida. De forma que queda totalmente definida la distancia al plano del imán y la posición del eje del mismo. Esta estrategia de medida ha sido simulada mediante $MatLab^{®}$ y funciona correctamente, la resolución de la medida es, como cabe esperar directamente proporcional a



(a) Representación del error en la medida

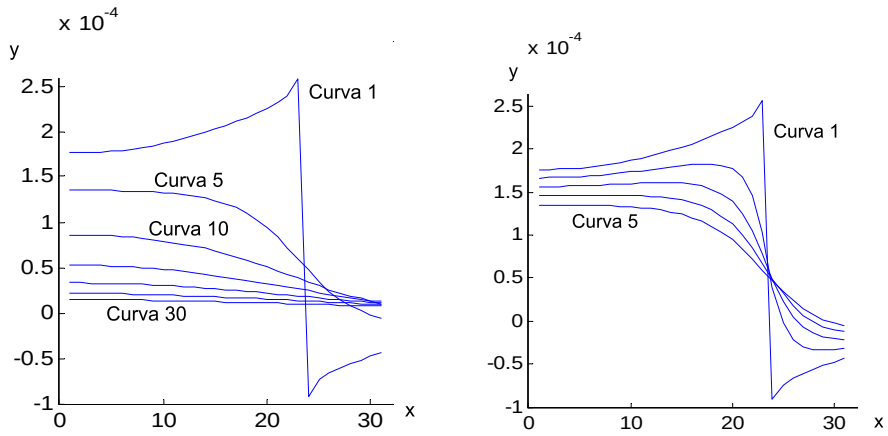
(b) Gráfica de la evolución del error en la medida con respecto a la distancia al plano del imán para un punto concreto.

Figura 3.11: Evolución del error.

la resolución con la que se hayan tomado las muestras al crear la matriz de datos sobre la que luego se realizan las lecturas. Pero existe una limitación, si la matriz de datos ocupa demasiado espacio, la búsqueda de datos dentro de ella conllevará un elevado coste computacional y eso va a repercutir directamente en la frecuencia máxima de medida a la que podrá trabajar el sensor. Por ello hay que buscar un equilibrio entre resolución y frecuencia máxima.

Ya se ha estudiado como evoluciona el error en la medida conforme nos desplazamos en planos paralelos al plano del imán. Pero también sería necesario estudiar la evolución del campo magnético al alejarse del eje del imán en cada uno de los planos. Realizar este análisis es lo mismo que analizar la evolución de las filas de la matriz creada, y la mejor manera de realizarlo es crear una gráfica (Ver figura 3.12). En esta gráfica se relacionará la distancia al eje del imán, que irá representada en el eje x, con el módulo del campo magnético en la dirección del eje z (Recordemos que solo se mide el campo magnético en el eje z), que irá representado en el eje y. Se realiza esta representación para distintas distancias al plano del imán con lo que se obtiene una familia de curvas. La curva 1 es la que está más cercana al plano del imán y la curva 30 la más alejada.

Se observa en la figura 3.12a que el campo magnético decrece a medida que el sensor se aleja del eje del imán, pero esto no se cumple en todos los casos. Ya en la figura 3.12a se observa, y en la 3.12b con más claridad que para distancias cercanas al plano del imán el campo magnético crece conforme nos acercamos al borde cilíndrico del imán, decrece bruscamente en las proximidades del borde (tanto por un lado como por el otro) para después volver a crecer suavemente. Esa es una de las peculiaridades del campo magnético generado por un imán y lo cierto es que para realizar la medida no es nada favorable, aún así se puede despreciar esta última característica si se restringe la zona de movimiento del sensor a zonas suficientemente alejadas del plano del imán. De este modo se puede afirmar que el módulo del campo magnético es decreciente conforme el punto se aleja



(a) Para todo el espacio representado. (b) En la zona cercana al imán.

Figura 3.12: Evolución del campo magnético con la distancia al eje del imán para distintas distancias al plano del imán.

del eje del imán y de esa forma los datos en las filas de la matriz están ordenados de mayor a menor. Este dato parece no tener mayor trascendencia pero lo cierto es que si los valores están ordenados existen estrategias que van a permitir el ahorro de mucho tiempo de procesamiento, es decir, el algoritmo será más rápido.

3.3.1. Búsqueda binaria o dicotómica.

Los datos de campo magnético están ordenados por filas en la matriz de mayor a menor. Esto es cierto a partir de una determinada distancia al plano del imán. De modo que para distancias mayores podremos emplear un algoritmo de búsqueda binaria o dicotómica [8] con una cierta particularidad. La búsqueda binaria consiste en dividir el array por su elemento medio en dos subarrays más pequeños, y comparar el elemento con el valor buscado. Si coinciden, la búsqueda se termina. Si el elemento es menor, debe estar en el primer subarray, y si es mayor está en el segundo.
Este algoritmo implica un coste computacional mucho menor, digamos que en el peor de los casos habrá que realizar $\log_2(n)$ operaciones donde n es el tamaño del array en el que se realiza la búsqueda. La particularidad es que no se pretende realizar lo que se conoce como una búsqueda en el sentido de que teniendo por ejemplo el $array$ $\{9, 7, 6, 5, 3, 2, 1\}$ se busca el número 4 y la respuesta sería evidente, no existe. Sin embargo no es esto lo que se pretende. Dado que los valores guardados en las filas de la matriz se corresponden con una función continua en el intervalo representado. Empleando el Teorema de Bolzano que dice:

> "Si una función es continua en un intervalo cerrado y acotado y en los extremos del mismo ésta toma valores con signos opuestos, entonces existe al menos una raíz de la función en el interior del intervalo."
> "Sea f una función continua en un intervalo cerrado $[a, b]$ tal que $f_{(c)} = 0$." [3]

Y por extención el teorema del valor intermedio que establece que:

> "Si una función es continua en un intervalo, la función toma todos los valores intermedios comprendidos entre los valores de la función en los extremos del intervalo."
> "Sea f una función continua en un intervalo cerrado $[a, b]$ y supongamos que $f_{(a)} > f_{(b)}$. Entonces para cada z tal que $f_{(a)} > z > f_{(b)}$, existe al menos un x dentro de (a, b) tal que $f_{(x)} = z$."
> La misma conclusión se obtiene para el caso que $f_{(a)} < f_{(b)}$.
> "Si además se añade que la función es decreciente en el intervalo abierto (a, b), entonces se puede afirmar que existe un único punto x dentro de (a, b) tal que $f_{(x)} = z$."

Por tanto, continuando con el ejemplo del $array$ $\{9, 7, 6, 5, 3, 2, 1\}$, si se decide buscar el número 4 teniendo en cuenta las consideraciones anteriores la respuesta será que se encuentra entre la posición 4 y la posición 5 del array. Esta va a ser la forma de proceder.

Capítulo 4

Desarrollo práctico.

4.1. Introducción.

Una vez realizado el estudio teórico del problema y simuladas las posibles soluciones, es momento de proceder con la parte práctica de la solución. En este capítulo se pretende dar una descripción de los componentes hardware y software empleados así como los motivos que llevaron a elegir estos componentes (Hardware) y estrategias (Software) y no otros en su lugar.
Lo que se pretende es que a partir de unas medidas del campo magnético se determine la posición de esta fuente. Para ello se van a emplear teres bloques principales. Sensores de campo magnético, electrónica de acondicionamiento y un software de medida. Cada uno de estos bloques se desarrollará de manera más pormenorizada en las siguientes secciones.

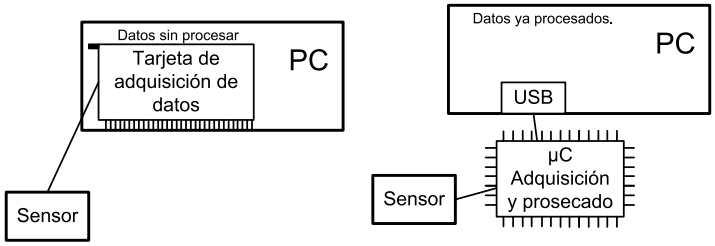
4.1.1. Arquitectura (Centralizada/Distribuida).

Lo primero que se debe definir va a ser la arquitectura del sistema. Centralizada o distribuida.

El robot humanoide RH-2 dispone de dos unidades de computación SBC Gemini 5.25" (core 2 duo). Una de ellas para el control de la parte inferior (caminante) y la otra para el control de la parte superior (extremidades superiores y visión).

Se puede emplear un sistema centralizado(ver Fig 4.1a.), es decir, que el proceso de las señales del sensor lo realice el mismo microprocesador que realiza el control del robot, para ello se emplea una tarjeta de adquisición de señal con la que se adquieren los datos "crudos" del sensor y que necesitan de un procesado bastante costoso hablando en términos de tiempo para obtener los datos finales de medida. O por el contrario se puede emplear un sistema distribuido (ver Fig 4.1b.), en el que un microprocesador externo o microcontrolador se encargue de realizar todo el procesamiento de los datos. Comunicándose con el microprocesador principal proporcionándole los datos finales de la medida.
Este microcontrolador está dedicado exclusivamente a la lectura del sensor por lo que el tiempo de proceso únicamente influye en la frecuencia de la lectura pero nunca impide o

dificulta la ejecución de otros procesos.



(a) Sistema centralizado. (b) Sistema distribuido.

Figura 4.1: Opciones de sistema centralizado o distribuido.

Un sistema distribuido es una agrupación de procesadores independientes, es decir, autónomos comunicados entre si de tal forma que a la vista del usuario actúan como una única máquina.

Ventajas	Inconvenientes
Un sistema distribuido difunde la carga de trabajo entre distintos microprocesadores. Se permite que los recursos se sitúen coherentemente donde deben estar, es decir, en el caso de la implementación del sensor que nos ocupa la lectura y procesamiento se realizará a escasos centímetros de la localización del sensor y no a una distancia de 1m, como sería el caso de la arquitectura centralizada.	La pérdida de comunicación o saturación de la red de comunicación puede saturar el sistema.

Cuadro 4.1: Ventajas e inconvenientes de los sistemas distribuidos

En el cuadro 4.1 se presentan las ventajas e inconvenientes de un sistema distribuido y se decide emplear ese sistema. Por ello los cálculos necesarios para la medida de los sensores recaerán en un microcontrolador totalmente independiente de las cpu. Ya se ha indicado anteriormente que se trata de unos cálculos un tanto pesados por lo que a continuación se buscarán microcontroladores que cumplan unas ciertas especificaciones.

4.2. Elección del sensor Hall.

A la hora de elegir el sensor de efecto Hall aparecen dos grupos bien diferenciados, aquellos que tienen una salida digital PWM y los que proporcionan una salida analógica.

Dentro de estos últimos además existen dos tipos, los que se puede programar su ganancia, offset, ajuste de sensibilidad con los incrementos de temperatura y los que no.
Antes de pormenorizar cada uno de los tres tipos de sensor Hall expuesto anteriormente se muestra un esquema con los bloques funcionales que componen por regla general un sensor Hall (Ver Fig 4.2). Los CI's sensor Hall se componen además del sensor hall por unos elementos de compensación de temperatura, compensación de los escuerzos mecánicos y amplificación de señal. Ya que el sensor Hall es muy dependiente de la temperatura y de los esfuerzos mecánicos.

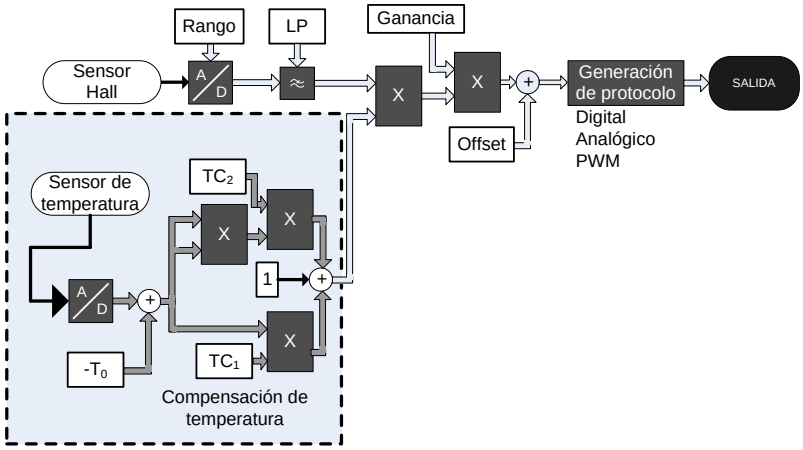
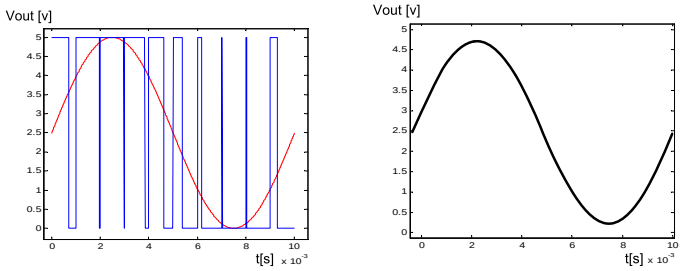


Figura 4.2: Diagrama de bloques de un CI sensor Hall.

4.2.1. Sensores Hall con salida PWM.

Un ejemplo de estos sensores es el TLE4998 de Infineon. El sensor proporciona una señal digital PWM la cual es adecuada para decodificar directamente por cualquier unidad de medida del ciclo de trabajo de una onda rectangular, normalmente un timer o una entrada de interrupción de un microcontrolador. Además es posible acoplar un filtro paso bajo (Ver Fig 4.3), lo que permite emplear un conversor A/D para realizar la medida.



(a) Onda PWM. (b) Filtrado de la onda PWM.

Figura 4.3: Ejemplo de demodulación de onda PWM mediante filtro paso bajo.

El proceso de funcionamiento de estos sensores es el siguiente:

- Se lee el flujo magnético mediante una célula Efecto Hall.
- La señal de la célula Efecto Hall es convertida a una señal digital.
- Un filtro paso bajo programable reduce el ruido.
- Se mide la temperatura y se convierte a una señal digital.
- La compensación de la temperatura se realiza mediante un polinomio de segundo orden.
- Se procesa digitalmente la señal basándose en una sensibilidad determinada y en un origen de campo cero.
- El valor de salida está limitado mediante limitadores digitales.
- El valor de salida final es transferido a un módulo PWM.
- El ciclo de trabajo de salida es proporcional a un valor de salida de 12 bits.
- Algunos valores como el rango de medida, y el modo de polaridad pueden ser programados para ajustarse a las necesidades.

La etapa de salida es un drenador abierto conectando la salida únicamente a nivel bajo, de modo que el nivel alto se obtiene únicamente mediante con una resistencia de pull-up externa. Este tipo de salida tiene la ventaja de que la resistencia de pull-up se puede conectar a tensiones más bajas, por ejemplo 3.3V.

4.2.2. Sensor Hall programable con salida analógica.

Un ejemplo de estos sensores es el MLX90215 de Melexis. Es un sensor de Efecto Hall lineal y programable que emplea tecnología CMOS. Posee una corrección de error activa que elimina totalmente el error de offset asociado a las celdas Efecto Hall. La respuesta ante el campo magnético del sensor puede ser totalmente programada a través de la salida Vout en operación normal. Cuando se programa para una sensibilidad convencional la tensión de salida aumenta a medida que un campo magnético Sur[1], se aplica a la cara marcada del sensor. De igual forma la tensión de salida decrece al aplicar un campo Norte. El sensor tiene un desplazamiento en la sensibilidad de $\pm 1\,\%$.

La programación se realiza a través del pin de salida mediante un cambio en las tensiones de alimentación, cuando ésta crece entre 13 y $18V$. Cuando la tensión de alimentación es de $5V$ el sensor se comporta normalmente. Cuando la tensión de alimentación crece por encima de $13V$ la salida se comporta como una entrada y carga una palabra de 31bits sin una señal de reloj dedicada. La palabra se almacena temporalmente en RAM, estos datos pueden ser guardados tantas veces como se desee pero se trata de una memoria volátil, únicamente cuando la tensión de alimentación crezca por encima de $18V$ los datos

[1] campo sur equivale a campo entrante, campo norte equivale a campo saliente.

almacenados en RAM se guardarán en memoria ROM con un número limitado de escrituras.
Mediante los 32 bits se puede programar el offset, la ganancia, y elegir si la pendiente de la ganancia es directa o inversa[2].

4.2.3. Sensor Hall con salida analógica.

Un ejemplo de estos sensores es el A1322 de Allegro La familia A132 de sensores Efecto Hall tienen una sensibilidad optimizada y estabilizada con respecto a la temperatura. Estos sensores radiométricos de Efecto Hall proporcionan una tensión de salida proporcional a campo magnético aplicado. Tienen una tensión de salida a campo 0 Gauss del $50\,\%$ de la tensión de alimentación Vcc y una sensibilidad de entre 2,5 y $5mV/G$. Cada dispositivo integra un elemento Hall con compensación de temperatura, un amplificador con baja impedancia de salida y un sistema de cancelación de offset con un reloj de alta frecuencia que permita frecuencias de muestreo superiores que resultan en mayor precisión y mayor rapidez en la capacidad del procesamiento de la señal. Esta técnica produce un dispositivo con una tensión de salida muy estable e inmune al estrés mecánico. El amplificador dentro del chip elimina los problemas derivados de las señales analógicas de bajo nivel. La precisión en la salida se obtiene mediante un ajuste de ganancia y offset interno en fábrica al final de la línea de producción.

De los tres sensores lo primero que cabe decir es que se va a descartar la utilización de sensores con salida PWM pues ya desde un principio se pretende emplear el conversor A/D del microcontrolador para tomar la medida. Entre los otros dos restantes se decide emplear el sensor Hall de ganancia no programable por su sencillez y sobre todo para evitar la necesidad de comprar un programador de sensores Hall. Como conclusión el sensor Hall elegido es el Allegro A1322.

4.3. Distribución física de los sensores y características del imán.

En este apartado se pretende determinar la posición relativa de tres sensores Hall teniendo en cuenta unas condiciones previamente establecidas como son el desplazamiento que va a sufrir el imán, también llamado testigo en algunos apartados de este documento y las dimensiones del sensor Hall escogido. Una vez establecida la posición de los tres sensores se continuará con la elección del imán, el material del que está compuesto, las dimensiones y un dato muy importante derivado de las dimensiones, la relación de aspecto, identificada como A.R. (Del inglés Aspect Ratio). Elegido el imán se debe determinar cual va a ser su posición de reposo de modo que nos permita mayor sensibilidad en la medida. Cabe destacar que la solución a este problema no es única y por ello lo que se ha hecho ha

[2]Ganancia directa implica que con campo magnético entrante la tensión aumentará y con campo saliente disminuirá, ganancia inversa implica un comportamiento exactamente al revés.

sido estudiar el problema, proponer una solución y comprobar que cumple las especificaciones. En el caso de que no se cumplan observar que modificaciones pueden hacerse para cumplir las condiciones restantes. Por lo tanto sería injusto determinar este sistema como un diseño ensayo-error, más bien es un ensayo-mejora.

4.3.1. Características del imán.

En el momento de elegir un imán la primera pregunta que se hace es el material de que está compuesto, ferrita, neodimio o samario. [1] [9]

Imán de ferrita: Las llamadas ferritas duras se emplean en imanes permanentes y tienen una gran remanencia después de su magnetización proporcionan un flujo magnético en función del volumen moderado. Están compuestas por aleaciones de ferrita cobalto y bario. Las ferritas se producen a menudo en forma de polvo, con el cual se pueden producir piezas de gran resistencia y dureza, previamente moldeadas por presión y luego calentadas, hasta llegar a la temperatura de fusión, en un proceso conocido como sinterización.

Imán de neodimio: Los imanes de Neodimio están compuestos por Tierras Raras, aleación Nd. Fe. B mediante sinterizado. Son imanes con una gran tendencia a la corrosión por lo que están protegidos exteriormente con un baño estándar de níquel, plata o zinc, plata habitualmente. La temperatura máxima de su utilización es de unos $80^{\text{o}}\ C$ para la aleación estándar y hasta los $200^{\text{o}}\ C$ para la calidad EH. Son imanes de gran potencia, aprox. 6 veces más que la Ferrita Anisotrópica.

Imán de Samario: Los imanes de samario-cobalto son una aleación por sinterizado de hierro, Samario, Cobalto, y adiciones de otros elementos. Pueden trabajar a una temperatura máxima de $350^{\text{o}}\ C$. Su mayor resistencia mecánica no hace imprescindible un recubrimiento superficial. Tienen una potencia ligeramente inferior a los imanes de neodimio.

Las condiciones primordiales y que puede resultar muy evidente es que el imán no debe saturar el sensor Hall en ningún momento y que su peso sea el menor posible. En un principio se pensó en utilizar imanes de Neodimio por tener para un mismo campo magnético menor tamaño, aunque lo cierto es que el tamaño del imán también quedará determinado por el tamaño de los sensores Hall y resulta que un imán de Neodimio de esas características satura los sensores Hall de modo que se han elegido finalmente imanes de ferrita.

En la figura 3.12 se representa la evolución del campo magnético con la distancia al eje del imán para distintas distancias al plano del imán.

1. Se observa que conforme la distancia al eje del imán disminuye, la variación del campo magnético en el eje z con respecto a este desplazamiento disminuye $\frac{dB_z}{dr} \rightarrow 0$ y decir esto es como afirmar que la sensibilidad del sensor con respecto a estos desplazamientos en la zona indicada es muy pequeña, por ello habrá que evitar esa zona de movimiento.

2. Del estudio anterior también se ha concluido que resulta conveniente que los sensores se encuentren siempre dentro del la proyección del área de imán, ¿Por qué? Pues pura y simplemente porque fuera de esta zona el campo magnético empieza a variar de una forma más brusca y ocurre que para dos distancias al plano del imán diferentes se obtiene el mismo valor de campo magnético, esa es una situación no deseada pues el algoritmo de medida propuesto se basa en comprobaciones del campo en distintas distancias al plano del imán y si hay puntos que coinciden, podría ocurrir que las soluciones sean múltiples.

3. Por último hay que tener en cuenta que el sensor no debe situarse nunca muy cerca del plano del imán pues como se observa en la figura 3.12 ocurre que en las cercanías del plano del imán el campo magnético deja de ser decreciente, entiéndase decreciente con que decrece conforme se mantiene fijo en el plano paralelo al plano del imán y se desplaza aumentando la distancia al eje del imán. Si el campo no es decreciente no se puede emplear el algoritmo de búsqueda por ello es otra zona de movimiento a evitar.

Por ello el diseño hará que tras la posible excursión máxima del imán, este aún se sitúe con su eje a una distancia mínima de 0.7mm de cada uno de los sensores. Así resulta el sistema de la figura 4.4

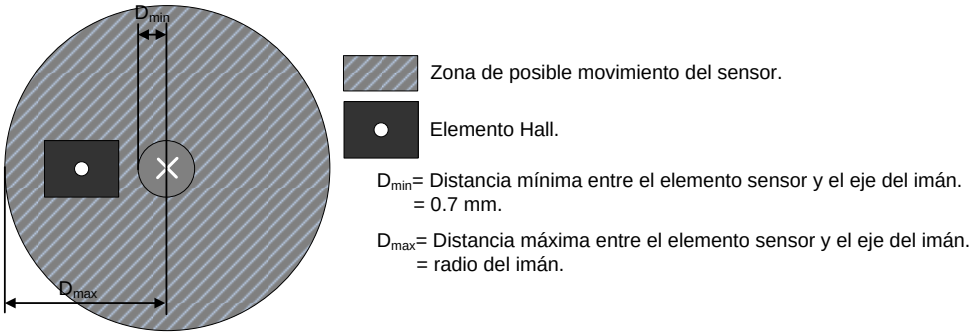


Figura 4.4: Zona de posible movimiento de un elemento sensor en función de las dimensiones del imán.

4.3.2. Distribución física de los sensores.

Se dispone de tres sensores, que proporcionan la lectura del campo magnético en tres puntos, de modo que se colocarán formando un triángulo. Entonces. ¿En que punto debe situarse el imán en el equilibrio? A simple vista se podría decir que da lo mismo mientras que se cumplan las condiciones 1, 2 y 3 del apartado 4.3.1, es decir, que esté por el medio, entre los tres sensores. Pero eso no sería del todo cierto. Los puntos notables del triángulo son cuatro, a saber circuncentro, incentro, ortocentro y baricentro [25].

El circuncentro es el punto en el que se cortan las tres mediatrices y es el centro de la circunferencia circunscrita, y por tanto el punto que equidista a los tres vértices.

El incentro de un triángulo es el punto en que se cortan las tres bisectrices y es el centro de la circunferencia inscrita, por ello es el punto que equidista a los tres lados del triángulo.

El ortocentro es el punto en el que se cortan las tres alturas.

El baricentro es el punto de corte de las tres medianas y coincide con el centro de gravedad del triángulo considerando que su densidad es uniforme.

Como todos los datos de campo magnético tienen que guardarse en una tabla, lo lógico es que una única tabla se pueda emplear para los tres sensores y ocupe el espacio mínimo. Para ilustrar esta idea se tomará el ejemplo de que la distancia al centro del imán varía entre $1mm$ y $3mm$ para el sensor 1, la del sensor 2 varía entre $2mm$ y $3mm$ y la del sensor3 entre $3mm$ y $5mm$, entonces debemos guardar una tabla en el micro una tabla que contemple los valores entre $1mm$ y $5mm$, si tomamos 256 muestras, la sensibilidad será de $\frac{5mm}{256muestras} \simeq 0{,}02mm/muestra$ o viéndolo de otra forma, si se desea una sensibilidad de 0.05mm/muestra, entonces el número de muestras que se han de guardar en la tabla sería de $\frac{5mm}{0{,}05mm/escalon} = 100$ muestras.
Ahora suponer que se sitúa el imán en el circuncentro del triángulo que forman los tres sensores, entonces, para los tres sensores, las distancias variarán por ejemplo entre $2mm$ y $4mm$ por ello tomando las mismas muestras(256) $\frac{2mm}{256muestras} \simeq 0{,}008mm/muestra$ queda una sensibilidad mayor. Por otro lado si se desea tener igual que en el apartado anterior una sensibilidad de $0{,}05mm/muestra$, entonces el número de muestras a guardar en la tabla se reduciría de 100 a $\frac{2mm}{0{,}05mm/muestra} = 40$ muestras. Es decir, ocupando menos memoria se consigue la misma sensibilidad y mayor velocidad ya que como se ha razonado en apartados anteriores buscar en una tabla más pequeña conlleva un menor coste computacional.

Debido a las limitaciones que nos imponen las dimensiones de los sensores OHN3150U que son para los que se ha diseñado inicialmente y con las máximas excursiones supuestas, de $\pm 1{,}5mm$ no va a existir el problema de la mínima distancia al eje. Y la distribución se realizará posicionándolos de modo que formen un triángulo isósceles como se muestra en la figura 4.5.

Esta configuración genera un triángulo isósceles cuyo circuncentro se encontrará en la intersección entre:
La mediatriz del segmento que une los sensores 1 y 2, es decir P_1 y P_2 y la mediatriz del segmento que une los sensores 1 y 3 (P_1 y P_3)
La primera mediatriz es trivial pues se trata de la recta

$$x = \frac{A}{2} \tag{4.1}$$

La segunda mediatriz es la recta perpendicular a la recta 4.2 y que pasa por el punto medio del segmento$\overline{P_1P_3}$, este punto medio[3] es $\left(\frac{B}{2}, \frac{C}{2}\right)$. Por tanto un vector paralelo a la recta 4.2

[3] A, B y C identificadas en la figura 4.5.

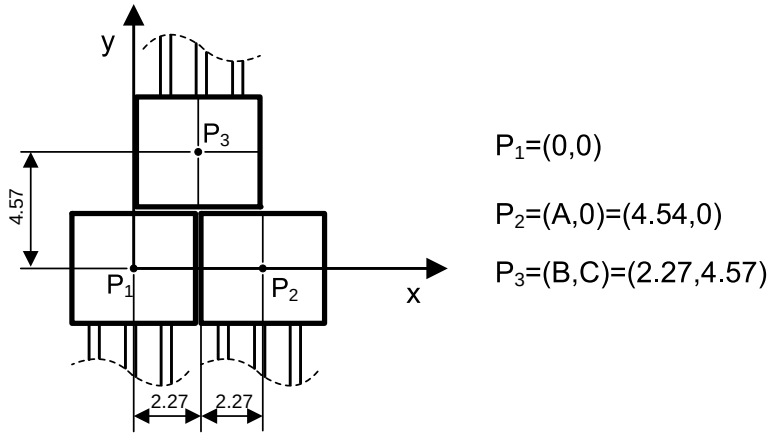


Figura 4.5: Situación de los sensores Hall.

puede ser $\vec{r_1} = \vec{i} + \frac{C}{B} \cdot \vec{j}$, un vector perpendicular $\vec{r_1}$ puede ser $\vec{r_2} = d \cdot \vec{i} + e \cdot \vec{j}$.

$$y = \frac{C}{B} \cdot x \tag{4.2}$$

Para que los vectores $\vec{r_1}$ y $\vec{r_2}$ sean perpendiculares se debe cumplir que el producto escalar de los mismos sea 0, es decir, $\vec{r_1} \cdot \vec{r_2} = 0 \rightarrow |\vec{r_1}| \cdot |r_2| \cdot \cos\alpha = 0$ porque perpendiculares implica $\alpha = 90^{\text{o}}$, entonces $\vec{r_1} \cdot \vec{r_2} = \left(1, \frac{C}{B}\right) \cdot (d, e) = d + e \cdot \frac{C}{B}$ Por tanto la pendiente de la recta perpendicular a la recta 4.2 será $m_2 = \frac{e}{d} = \frac{-B}{C}$. Se halla la ecuación de la mediatriz 4.3 mediante la ecuación de la recta punto pendiente $y - y_0 = m_2 \cdot (x - x_0)$ con $x_0 = \frac{B}{2}$, $y_0 = \frac{C}{2}$

$$y = \frac{-B}{C} \cdot x + \frac{B^2 + C^2}{2 \cdot C} \tag{4.3}$$

De modo que la posición del circuncentro es la intersección de las rectas 4.1 y 4.3:

$$x_{circuncentro} = \frac{A}{2} = 2{,}27mm \tag{4.4}$$

$$y_{circuncentro} = \frac{B^2 - B \cdot A + C^2}{2 \cdot C} = 1{,}72mm \tag{4.5}$$

Y así, el imán queda colocado según la figura 4.6

4.3.3. Comprobación de distancias máximas y mínimas.

Se debe comprobar que la distancia mínima sea mayor que la que se ha especificado $(0{,}7mm)$ y la máxima menor que el radio del imán pues los sensores siempre deberán estar sobre el área cubierta por el imán.
La expresión del módulo de la distancia entre dos puntos es:

$$|r| = \sqrt{(x - x_0)^2 + (y - y_0)^2} \tag{4.6}$$

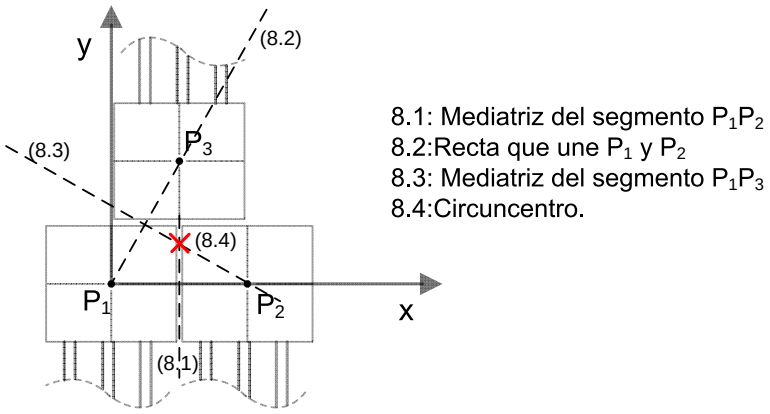


Figura 4.6: Posición del imán relativa a los tres sensores. Circuncentro.

Distancias al sensor 1

$$d_{eq} = \sqrt{\left(\frac{A}{2}\right)^2 + \left(\frac{B^2 - B \cdot A + C^2}{2 \cdot C}\right)^2} = 2{,}85mm$$

$$d_{max} = \sqrt{\left(\frac{A}{2} + \Delta x\right)^2 + \left(\frac{B^2 - B \cdot A + C^2}{2 \cdot C} + \Delta y\right)^2} = 4{,}96mm$$

$$d_{min} = \sqrt{\left(\frac{A}{2} - \Delta x\right)^2 + \left(\frac{B^2 - B \cdot A + C^2}{2 \cdot C} - \Delta y\right)^2} = 0{,}80mm$$

Con $\Delta x = \Delta y = 1{,}5mm$, d_{max} es la distancia máxima entre el sensor y el eje del imán d_{min} distancia mínima entre el sensor y el eje del imán.
Distancias al sensor 2: Idem al sensor 1 por simetría del triángulo isósceles.
Distancias al sensor 3:

$$d_{eq} = \sqrt{\left(B - \frac{A}{2}\right)^2 + \left(C - \frac{B^2 - B \cdot A + C^2}{2 \cdot C}\right)^2} = 2{,}85mm$$

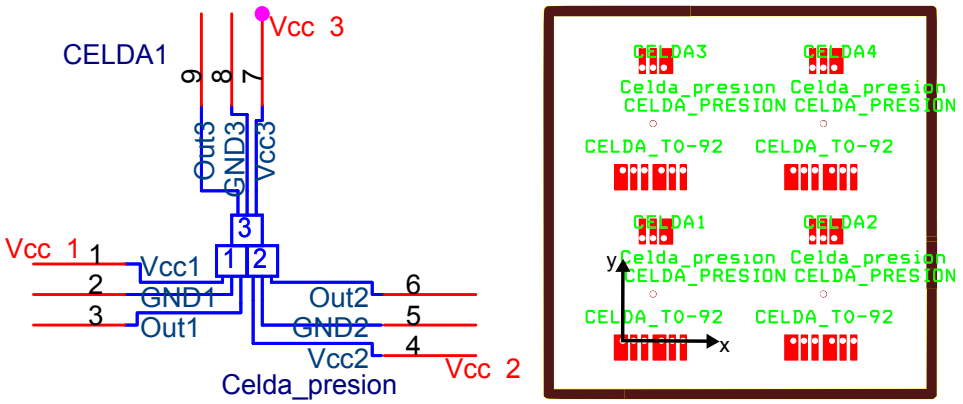
$$d_{max} = \sqrt{\left(B - \frac{A}{2} + \Delta x\right)^2 + \left(C - \frac{B^2 - B \cdot A + C^2}{2 \cdot C} + \Delta y\right)^2} = 4{,}60mm$$

$$d_{min} = \sqrt{B - \left(\frac{A}{2} - \Delta x\right)^2 + \left(C - \frac{B^2 - B \cdot A + C^2}{2 \cdot C} - \Delta y\right)^2} = 1{,}35mm$$

Se puede comprobar que se cumple que la distancia mínima es menor que $0{,}7mm$. En resumen se empleará un con un diámetro mayor a $b_{max} \cdot 2$, es decir, mayor de $10mm$, buscando en fabricantes de imanes se han encontrado imanes de ferrita de dimensiones $\left\{\begin{matrix} \phi & = & 12mm \\ L & = & 4mm \end{matrix}\right\}$.

4.4. Diseño de una placa prototipo con 4 celdas sensoras.

Una vez que se han determinado las posiciones de los elementos sensores, ver Fig 4.5. y la posición del imán, ver Ecuaciones 4.1 y 4.3, la generación de un prototipo de 2×2 celdas es posible, para ello se van a emplear Herramientas de Cadente Design Systems, Inc. Tanto Orcad Capture como Orcad Layout. El primero se emplea para crear una celda compuesta de tres sensores Hall y sus conexiones electricas, replicando ésta cuatro veces se tienen todos los elementos necesarios para crear el prototipo 2×2. El segundo programa, Orcad Layout se emplea para asignar una huella de componente a la celda sensora y posicionarla en la placa de circuito impreso [2] [5].



(a) Componente Celda Hall en Orcad Capture. (b) Layout de un prototipo 2×2 mediante Orcad Layout.

Figura 4.7: Diseño de un prototipo 2×2.

Para situar los taladros en la huella del componente se han tenido en cuenta las dimensiones del sensor hall que se va a emplear Ver Fig 4.8. Conociendo la posición en la que se debe encontrar el elemento Hall y las coordenadas de cada taladro respecto del sensor hall se pueden conocer las coordenadas absolutas de cada taladro.

Por ejemplo, suponer que el centro del elemento Hall se sitúa en coordenadas (0,0), entonces, las posiciones de los taladros son los siguientes:

$$t_1 = (0 - 1{,}27, 0 - 3{,}75) = (-1{,}27, -3{,}75)$$

$$t_2 = (0, 0 - 3{,}75) = (0, -3{,}75)$$

$$t_3 = (0 + 1{,}27, 0 - 3{,}75) = (1{,}27, -3{,}75)$$

Siendo T_i la coordenadas del taladro para la pata i. En la tabla 4.3 se muestran las coordenadas de cada uno de los taladros que forman el prototipo de la celda 2×2.

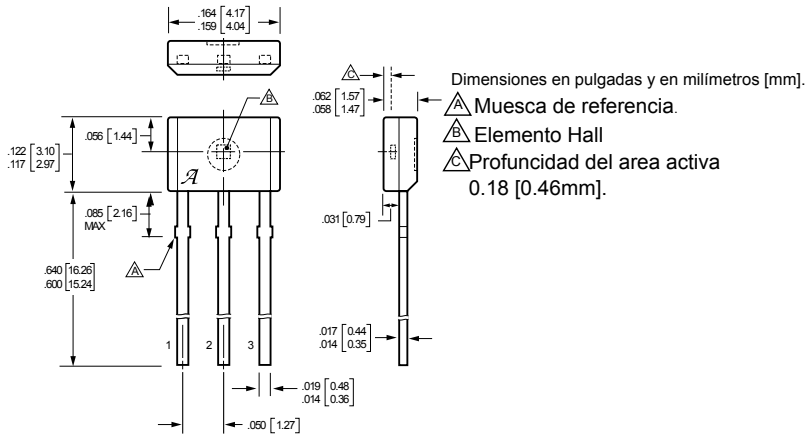


Figura 4.8: Dimensiones del sensor efecto Hall Allegro A1321.

CELDA 1-1				CELDA 1-2 Sumando 20mm a la coordenada x de la celda 1-1		
	X	Y			X	Y
A1	8,73	6,25		A1	28,73	6,25
A2	10	6,25		A2	30	6,25
A3	11,27	6,25		A3	31,27	6,25
B1	13,27	6,25		B1	33,27	6,25
B2	14,54	6,25		B2	34,54	6,25
B3	15,81	6,25		B3	35,81	6,25
C1	13,54	18,29		C1	33,54	18,29
C2	12,27	18,29		C2	32,27	18,29
C3	11	18,29		C3	31	18,29
Centro	12,27	11,72		Centro	32,27	11,72
CELDA 2-1 Sumando 20mm a la coordenada y de la celda 1-1				CELDA 2-2 Sumando 20mm a la coordenada x de la celda 2-1		
	X	Y			X	Y
A1	8,73	26,25		A1	28,73	26,25
A2	10	26,25		A2	30	26,25
A3	11,27	26,25		A3	31,27	26,25
B1	13,27	26,25		B1	33,27	26,25
B2	14,54	26,25		B2	34,54	26,25
B3	15,81	26,25		B3	35,81	26,25
C1	13,54	38,29		C1	33,54	38,29
C2	12,27	38,29		C2	32,27	38,29
C3	11	38,29		C3	31	38,29
Centro	12,27	31,72		Centro	32,27	31,72

Cuadro 4.3: coordenadas xy de cada uno de los 36 taladros para el posicionamiento de los sensores Hall en la placa de circuito impreso.

Calculados los puntos de taladro de una celda, el resto de celdas se obtienen por un simple desplazamiento de puntos, este desplazamiento será de 2cm pues es la distancia a la que dos imanes no tienen influencia entre sí.

4.5. Electrónica asociada a la adquisición de datos.

4.5.1. Multiplexación.

El microcontrolador PIC32MX460F512L tiene 16 entradas analógicas, para una celda 2x2 se necesitarían únicamente 12 entradas analógicas, por tanto una opción es conectarlo directamente a las entradas del conversor A/D del microcontrolador, pero de hacer esto había que incluir el acondicionamiento de señal para 12 señales. Se empleará la multiplexación ver Fig 4.9. con dos objetivos: El primero minimizar la necesidad de electrónica de acondicionamiento. El segundo, poder seleccionar cuál es la celda que se quiere medir en un preciso momento.

1. Minimizar la necesidad de electrónica: Si se conecta una etapa de multiplexación, la cual reduce las 12 señales a 3, en lugar de emplear 12 circuitos de acondicionamiento de señal será suficiente con emplear 3, ahora bien, si el acondicionamiento de señal necesita de algún filtro paso bajo se debe tener en cuenta este factor a la hora de diseñar la frecuencia de corte del filtro.

2. Selección de la celda a medir: Mediante los multiplexores es posible con dos señales posicionarse en la matriz de sensores, con una señal indicando la fila, y otra la columna, en el caso del prototipo 2x2 se trata de únicamente dos señales de un bit.

Para la implementación se han buscado multiplexores analógicos[4] 4-1, pero no se han encontrado en el distribuidor habitual, de modo que se han empleado multiplexores 8-1 74HC4051N en un encapsulado DIL100 de 16 pines.

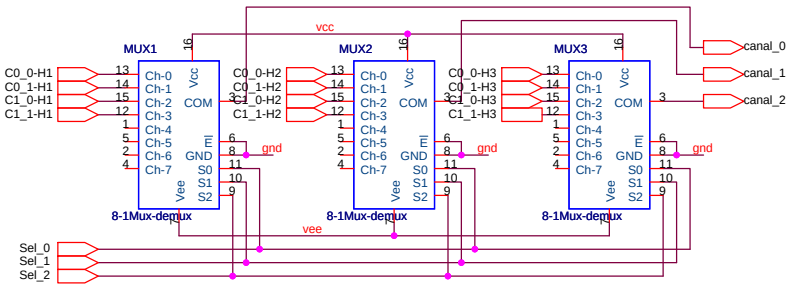


Figura 4.9: Diseño del circuito de multiplexión 12-3. La nomenclatura C1_0-H2 se refiere al sensor Hall 2 de la fila 1 y columna 0.

[4]Cuando se habla de multiplexores analógicos, estos multiplexores, al actuar de manera similar a un interruptor analógico se pueden emplear como multiplexores o como demultiplexores indistintamente.

Los canales de selección han de ir conectados a los puertos de salida del PIC. Para ello habrá que encontrar tres pines libres. Esto no parece ser un problema ya que el microcontrolador que se ha utilizado PIC32MX460F512L se suministra en un encapsulado 100-TQFP, es decir, un encapsulado de 100 pines de los cuales 83 corresponden a Pines de entrada/salida. Aún así, muchos puertos están multiplexados con otras funciones, es por eso que se debe comprobar que pines de que puertos están siendo utilizados y cuales aún quedan libres.

Función	Nº pin encapsulado	Nombre	Multiplexado con:	Puerto	Numero
Conversor Analógico/ Digital	25	AN0	PGD1/EMUD1/CN2/RB0	Puerto B	0
	24	AN1	PGC1/EMUC1/CN3/RB1	Puerto B	1
	23	AN2	C2IN-/CN4/RB2	Puerto B No utilizado pues está roto.	2
	22	AN3	C2IN+/CN5/RB3	Puerto B	3
Comunicación serie RS-232	50	U2Tx	PMA8/CN18/RF5	Puerto F	5
	49	U2Rx	PMA9/CN17/RF4	Puerto F	4
Comunicación serie USB	57	D+	RG2	Puerto G	2
	56	D-	RG3	Puerto G	3
Control de multiplexión de celdas Hall.	72	RD0	SDO1/OC1/INT0	Puerto D	0
	76	RD1	OC2	Puerto D	1
	77	RD2	OC3	Puerto D	2

Cuadro 4.5: Pines de entrada/salida en uso en el microcontrolador.

Como se muestra en la tabla 4.5 los pines usados para controlar la multiplexión de las señales de entrada son los puertos RD0 RD1 y RD2, se ha decidido emplear estos tres porque en la placa de desarrollo estos tres pines están conectados a tres LED SMD que sirven para saber en todo momento cual de las cuatro celdas es la que se está direccionando. Esta información visual únicamente será útil cuando se ejecute el código paso a paso (Step by step.) pues en funcionamiento normal es tal la velocidad a la que se hace la lectura que el ojo humano no es capaz de distinguir el encendido y el apagado de los LED. Únicamente se observa el LED encendido con una luminosidad determinada igual que si estuviera alimentado con una onda PWM.

En la tabla 4.7 se observa que el circuito de multiplexión implementado es capaz de direccional 8 celdas de medida, sin embargo el prototipo fabricado únicamente dispone de cuatro celdas por lo que solo se utilizan las cuatro primeras.

RD2	RD1	RD0		Direccionando
0	0	0	$\longrightarrow$	Celda 0_0.
0	0	1	$\longrightarrow$	Celda 0_1.
0	1	0	$\longrightarrow$	Celda 1_0.
0	1	1	$\longrightarrow$	Celda 1_1.
1	0	0	$\longrightarrow$	Sin implementar.
1	0	1	$\longrightarrow$	Sin implementar.
1	1	0	$\longrightarrow$	Sin implementar.
1	1	1	$\longrightarrow$	Sin implementar.

Cuadro 4.7: Celdas direccionadas en función de las tres señales de control.

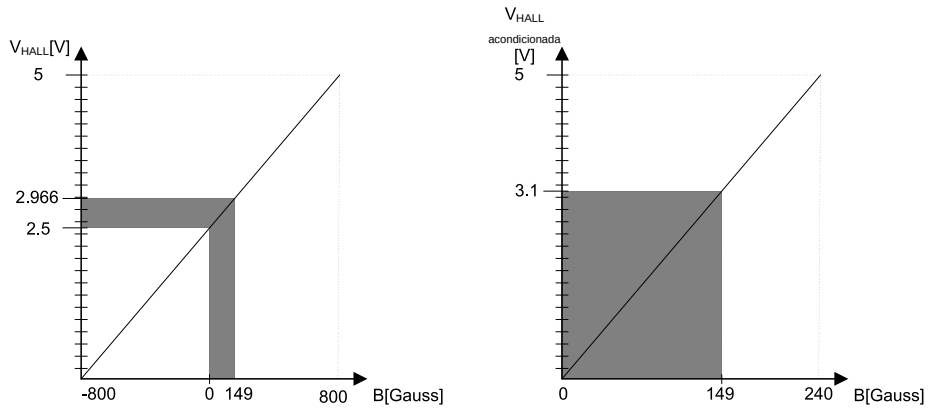
4.5.2. Acondicionamiento de señal.

Para la lectura de la señal se dispone de un conversor A/D de 10 bits, este conversor puede medir una tensión máxima de $3{,}3V$. Está configurado para la medida de tensiones entre 0 y $3{,}3V$. Los registros encargados de configurar el conversor A/D de esta forma son:

- AD1PCFG Se le pasa una máscara para saber que pines pertenecientes al puerto actúan como entradas analógicas o como pines I/O digitales. 0 configura como entradas analógicas.
- AD1CON1 Inicia la conversión.
- AD1CSSL Controla las funciones de escaneo de canales.
- AD1CON2 Utiliza MUXA y establece las referencias a V_{DD} y V_{SS}, es decir, $3{,}3V$ y $0V$.
- AD1CON3 Selecciona el reloj de conversión.

Con esta configuración para una tensión de $0V$ corresponderá un código de salida $00.0000.0000_2$ y para $3{,}3V$ el código será $11.1111.1111_2$. Para aprovechar todo el rango de medida del conversor A/D se debe eliminar el offset y amplificar la señal (Ver Fig 4.11.) en este orden y no en el contrario, pues de amplificar la señal antes de eliminar el offset saturaría los amplificadores en la primera etapa.

En la Fig 4.10.A se observa que el sensor Efecto Hall tiene un offset de $2{,}5V$, es decir, para un campo magnético incidente de 0Gauss, la tensión de salida es de $2{,}5V$. La forma de eliminarlo es realizando una resta de tensión, de forma que , esto se puede conseguir con un amplificador operacional en configuración sumador-restador. A la hora de tomar la tensión de referencia para la resta existen varias posibilidades, emplear como tensión de referencia V_{cc}, es decir, los $5V$ o emplear un zener para fijar la tensión de referencia. Se ha optado la segunda opción utilizando un Zener de precisión de $2{,}5V$.



(a) Antes del acondicionamiento. (b) Después del acondicionamiento.

Figura 4.10: Gráfica campo magnético contra tensión de salida.

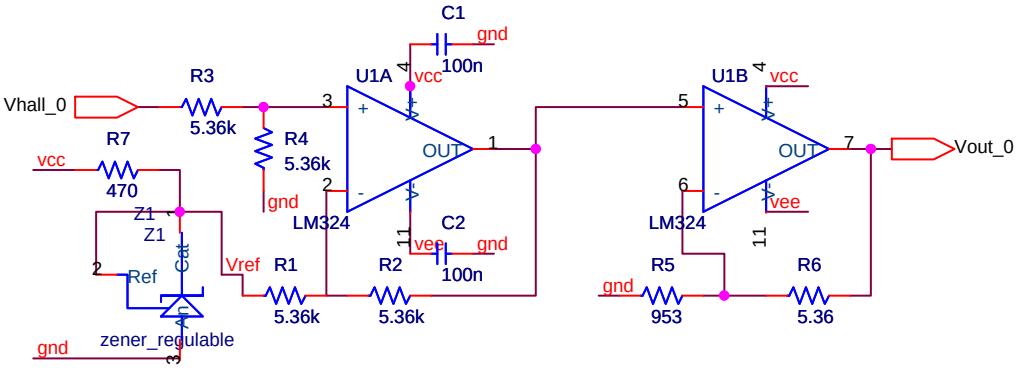


Figura 4.11: Acondicionamiento de señal, eliminación de offset y amplificación de señal.

Estos diodos zener de precisión son reguladores de tensión que pueden proporcionar tensiones desde $2{,}5V$ hasta $36V$ de modo que al desear una resta de tensión de $2{,}5V$ se configura el zener para proporcionar esta tensión y así todas las resistencias del circuito tendrán el mismo valor con lo que se simplifica tanto la compra como el montaje de los comonentes. La resistencia R_7 se diseña de modo que la corriente mínima del zener en inversa sea siempre mayor que 0.
Las resistencias R_1,R_2,R_3 y R_4 se calculan para que definidas las dos entradas, $V_{ref} = 2{,}5V$ fijada por el zener y V_{hall} la tensión variable de entrada, V_{hall} sufra una resta de $2{,}5V$ sin ninguna ganancia. La ecuación que rige el comportamiento de un amplificador operacional en modo sumador restador es la siguiente:

$$V_{in} = V_{hall} \cdot \frac{R_4}{R_3 + R_4} \cdot \frac{R_1 + R_2}{R_1} - V_{ref} \cdot \frac{R_2}{R_1} \tag{4.7}$$

Para conseguir la resta de 2.5V, el segundo término $V_{ref} \cdot \frac{R_2}{R_1} = 2{,}5V$ por tanto $R_1 = R_2$, para que V_{hall} permanezca sin ganancia, el primer término $V_{hall} \cdot \frac{R_4}{R_3+R_4} \cdot \frac{R_1+R_2}{R_1} = V_{hall}$ y por tanto $R_3 = R_4$ una solución simple y muy conveniente por unificar valores es hacer $R_1 = R_2 = R_3 = R_4 = 5{,}36K\Omega$

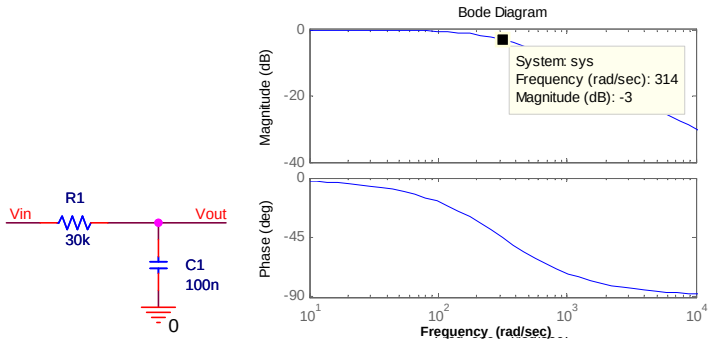
Respecto a la etapa de amplificación se pretende que una tensión entre 0V y 0.466V alcance valores de 3.1V para emplear el máximo rango de medida del conversor A/D. Resulta trivial calcular que la ganacia de amplificación debe ser:

$$G = \frac{V_{out}}{V_{in}} = \frac{3{,}1V}{0{,}466V} = 6{,}652V/V \tag{4.8}$$

La ecuación que describe el comportamiento del amplificador operacional como amplificador no inversor es $V_{out} = V_{in} \cdot \left(1 + \frac{R_6}{R_5}\right)$
Así, los valores de las resistencias son cualquier pareja que cumpla que $\frac{R_6}{R_5} = 5{,}652$. Se van a emplear resistencias con una tolerancia del $0{,}5\,\%$ de modo que $R_6 = 5{,}36K\Omega$ y $R_5 = 0{,}953K\Omega$.

4.5.3. Filtro Anti-alising.

El alising es el efecto que ocurre cuando una señal es muestreada a una frecuencia inferior la Frecuencia de Nyquist[5]. En este caso, dos señales continuas diferentes pueden tornarse indistinguibles cuando son muestreadas.
Está demostrado que para evitar el alising, la frecuencia de la señal continua f_{max} no debe ser superior a la mitad de la frecuencia de muestreo $f_{muestreo}/2$.
Por otro lado al analizar la señal proporcionada por los sensores hall se ha observado que ésta tenía un ruido de amplitud $100mV$ en un conjunto de frecuencias a partir de los $50Hz$ hasta los $22Khz$ por lo que se ha decidido implementar un filtro paso bajo (Ver Fig 4.12.A) de primer orden con frecuencia de corte $50Hz$



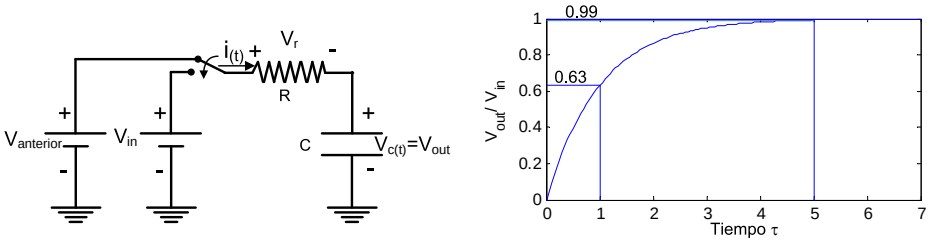
(a) Implementación de un filtro paso bajo de primer orden

(b) Diagrama de bode del filtro paso bajo.

Figura 4.12: Filtro paso bajo.

El filtro paso bajo de la figura 4.12 es necesario para eliminar el ruido observado en las señales, pero es momento de preguntarse que es lo que ocurre durante la multiplexión.

[5]La frecuencia de Nyquist es dos veces la frecuencia de la onda que se desea muestrear.



(a) Circuito que representa el transitorio en una multiplexión.

(b) Gráfica de un transitorio RC.

Figura 4.13: Transitorio en un circuito RC.

Al cambiar de un canal a otro, el filtro de la figura 4.12A se puede analizar como un circuito en régimen transitorio (Ver figura 4.13A), es decir, la carga o descarga de un condensador a través de una resistencia, y es por eso que tras realizar un cambio de canal no se puede proceder a tomar la muestra para la conversión analógico a digital sino que hay que esperar a que termine el transitorio. Para saber cuando termina el transitorio se debe analizar el circuito.
La ecuación que rige el comportamiento del circuito es:

$$V_{in} = R \cdot i_{(t)} + Vc(t) \tag{4.9}$$

De manera general, la corriente que circula por un condensador queda definida con:

$$i_{v(t)} = C\frac{dV_{C(t)}}{dt} \tag{4.10}$$

Sustituyendo 4.10 en 4.9 queda lo siguiente:

$$V_{in} = RC \cdot \frac{dV_{C(t)}}{dt} + V_{C(t)} \longrightarrow \frac{1}{V_{in} - V_{C(t)}} \cdot dV_{C(t)} = \frac{1}{RC} \cdot dt \tag{4.11}$$

Integrando la expresión refec:transitorio-sin-integral teniendo en cuenta las condiciones iniciales, en $t = 0$, $V_{C(t)} = V_{anterior}$ y en $t = t'$, $V_{C(t)} = V_{out}$

$$\int_{V_{anterior}}^{V_{out}} \frac{1}{V_{in} - V_{C(t)}}\, dV_{C(t)} = \int_0^t \frac{1}{RC}\, dt \tag{4.12}$$

Realizando la integración y tomando la exponencial se obtiene la relación entre la tensión de salida y la tensión de entrada con respecto al tiempo.

$$V_{out} = Vin \cdot \left(1 - e^{\frac{-t}{RC}}\right) + V_{anterior} \cdot e^{\frac{-t}{RC}} \tag{4.13}$$

De la ecuación 4.13 se observa que la salida está compuesta por dos términos, el primero $Vin \cdot \left(1 - e^{\frac{-t}{RC}}\right)$ que es la entrada actual y el segundo $V_{anterior} \cdot e^{\frac{-t}{RC}}$ que es la entrada anterior al transitorio. Si $t = 0^+$, es decir, cero por la derecha se produce el muestreo, la medida que se toma $V_{out} = V_{in} \cdot (1-1) + V_{anterior} \cdot 1$ no es la señal deseada. Será necesario esperar un tiempo a que el transitorio termine. Este tiempo como norma general se suele

tomar $5 \cdot \tau$ (Ver fig 4.13), siendo τ la constante de tiempo $\tau = R \cdot C$. Esto es porque para un tiempo de $t = 5 \cdot \tau$:

$$V_{out} = Vin \cdot \left(1 - e^{-5}\right) + V_{anterior} \cdot e^{-5} \text{ , es decir, } V_{out} = 0{,}99V_{in} + 0{,}007V_{anterior} \quad (4.14)$$

Es por todo esto que tras ralizar un cambio de canal hay que programar una espera de $5 \cdot \tau$ segundos antes de realizar la lectura del valor analógico.

4.5.4. Alimentación. Electrónica de potencia.

La electrónica de acondicionamiento y medida empleada necesita de tensiones simétricas para operar correctamente. Es por eso que se opta por dotar al diseño de un convertidor continua-continua. Este convertidor debe tener una entrada de $5V$ y una salida simétrica de $\pm 5V$. Para conseguir una tensión simétrica es necesario conseguir masas aisladas.
Un convertidor apropiado para esta tarea es el convertidor $FLYBACK$, que es un reductor elevador con aislamiento.

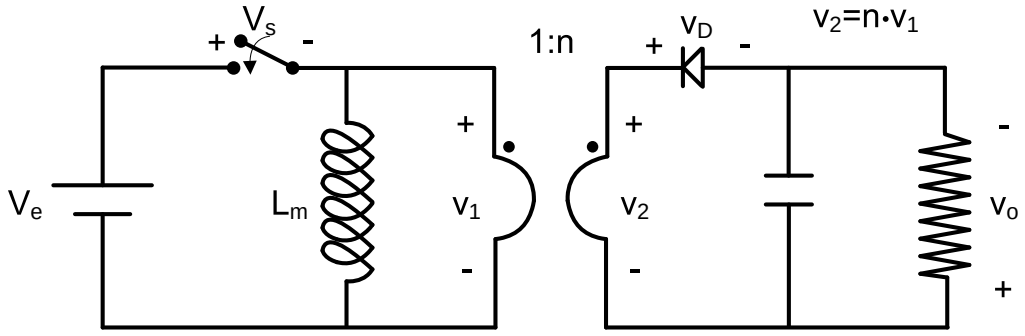
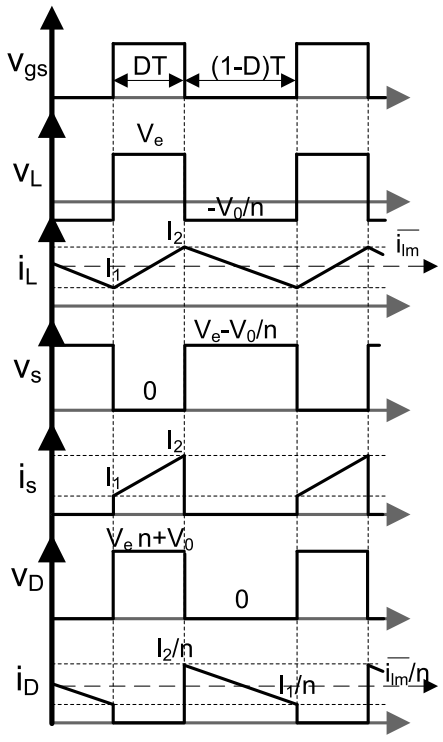
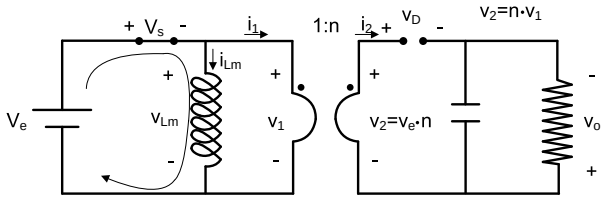


Figura 4.14: Diagrama de un convertidor reductor elevador FLYBACK.



Se distinguen dos estados del circuito T_{ON} y T_{OFF}

Durante el tiempo T_{ON}



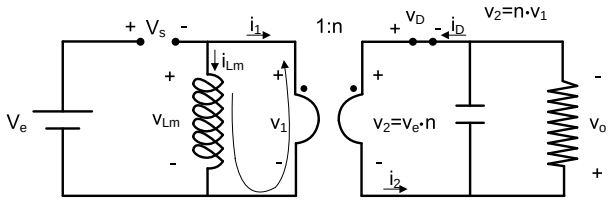
$v_{lm} = V_e$ $\quad i_{Lm} = I_1 + \frac{V_e}{L} \cdot t$

$v_D = v_2 - (V_0) = V_e \cdot n + V_0$ $\quad i_D = 0$

$v_s = 0$ $\quad i_s = i_{Lm} + i_1 = i_{Lm}$

$i_1 = 0 \; ; i_2 = 0$

Durante el tiempo T_{OFF}



$v_{Lm} = v_1 = \frac{-V_0}{n}$ $\quad i_{Lm} = I_2 - \frac{V_0}{n \cdot L} \cdot t$

$v_D = 0$ $\quad i_D = i_2 = \frac{i_{Lm}}{n}$

$v_s = V_e - v_{Lm} = V_e + \frac{V_0}{n}$ $\quad i_s = 0$

$v_1 = \frac{v_2}{n}$, $v_2 = -V_0$ $\quad i_2 = \frac{i_{Lm}}{n}$

- Relación de conversión de tensión.

$$\bar{v_L} = 0 \rightarrow V_e \cdot D \cdot \not{T} = \frac{V_0}{n} \cdot (1-D) \cdot \not{T} \rightarrow V_e \cdot D = \frac{V_0}{n} - \frac{V_0}{n} \cdot D \rightarrow V_0 \cdot \frac{(1-D)}{n} = V_e \cdot D$$

$$V_0 = V_e \cdot \frac{D}{1-D} \cdot n \tag{4.15}$$

Donde D es el ciclo de trabajo.

Se ha decidido utilizar un convertidor DC-DC de la marca XP Power [31] modelo IA0505S en un encapsulado Single In Package

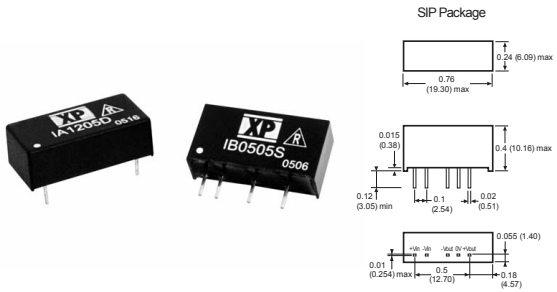
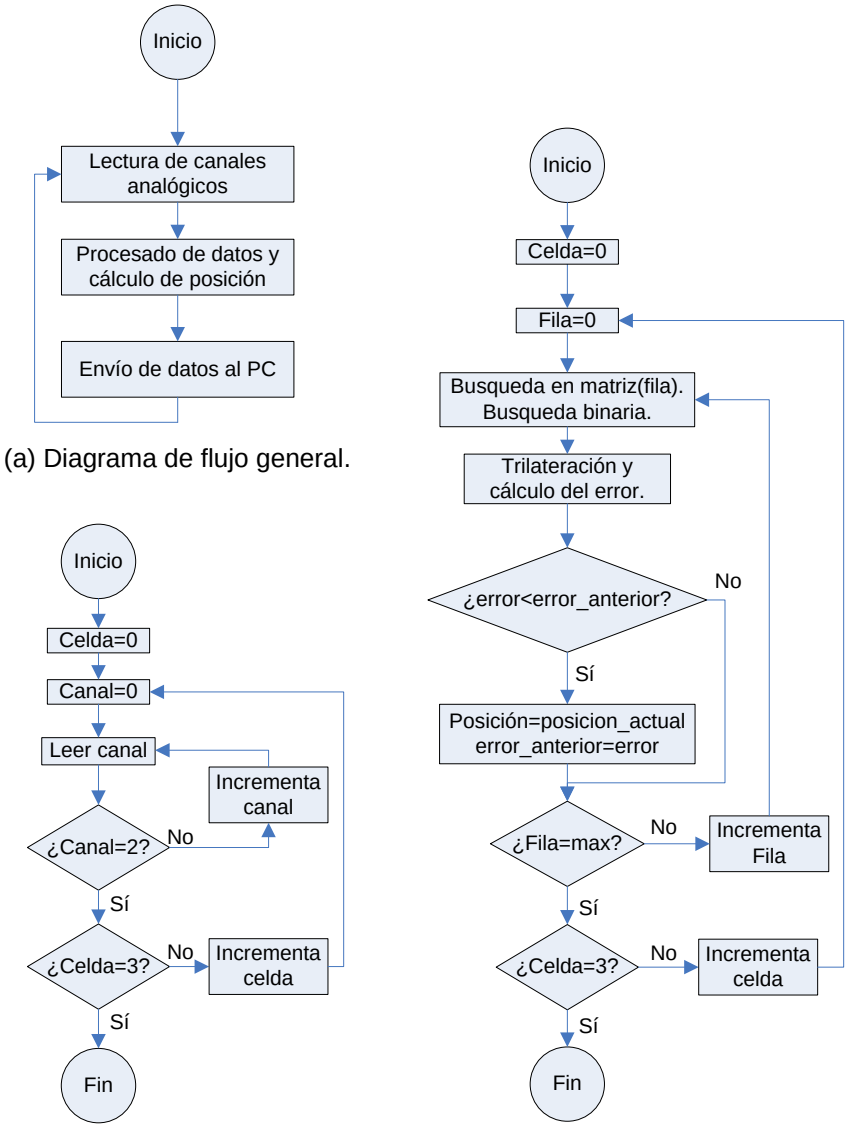


Figura 4.15: Convertidor de potencia DC-DC para montaje sobre PCB.

Este convertidor tiene una tensión de entrada de 5V y una tensión de salida de $\pm 5V$ pudiendo entregar una corriente máxima de $\pm 100mA$ con una eficiencia del $70\,\%$ y un consumo de corriente en reposo de 25mA. La refrigeración la lleva a cabo la libre convección del aire.

4.6. Firmware de medida.

El firmware de medida grabado en el PIC se compone de tres partes principales: Multiplexión y medida de las entradas analógicas, cálculo de la posición mediante búsqueda binaria y trilateración y por último el envío de los datos de las posiciones al PC.



(a) Diagrama de flujo general.

(b) Diagrama de flujo lectura.

(c) Diagrama de flujo cálculo.

Figura 4.16: Diagramas de flujo del Firmware del PIC.

El módulo de multiplexión y medida trabaja de la manera siguiente: Mediante tres bits de salida selecciona la celda en la cual se va a realizar la medida. Teniendo tres bits de selección podría seleccionar hasta 8 celdas, pero en este prototipo sólo se han implementado 4 de ellas. Una vez seleccionada la celda a medir se realiza una espera para que el filtro R-C pueda alcanzar su valor estacionario y a partir de ese momento se lee secuencialmente cada una de las tres entradas analógicas. Se repite este proceso con cada una de las celdas.

E l proceso de cálculo de la posición se puede dividir en dos partes. La primera parte es una búsqueda binaria como se explica en el apartado 3.3.1. Esta búsqueda binaria se aplica a los tres sensores de una celda sobre la primera fila de una matriz en la que está guardado el valor del campo magnético en cada uno de los puntos, ver apartado 3.2.3. Con esto se consigue una distancia al plano del imán y tres distancias al eje del imán. Con las dos primeras distancias al plano del imán y con una herramienta matemática llamada trilateración se calcula una posible posición del imán en el pllano. Con esta posición y la tercera distancia se calcula el error en la medida. Repitiendo este proceso para todas las filas de la matriz se tomatá como posición correcta la fila de la matriz que proporcione un menor error, y de esa fila se calcularán las posiciones XYZ del menor error.

Una vez realizada la medida solo queda enviarla al PC meidante una comunicación RS-232 para poder visualizar esa posición en una gráfica o realizar cálculos de distribución de fuerzas etc.

4.7. Elección del microcontrolador.

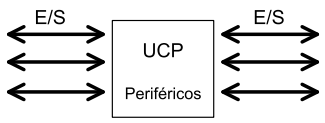
Existe una gran variedad de microcontroladores. La mejor manera de clasificarlos tal vez sea en microcontroladores de 8, 16 ó 32 bits. Aunque los microcontroladores de 32 bits ofrecen unas prestaciones muy superiores, la realidad es que los microcontroladores de 8 bits dominan el mercado. La razón de esta tendencia es que los microcontroladores 8 bits son apropiados para la inmensa mayoría de las aplicaciones, lo que hace innecesario emplear microcontroladores más potentes y por lo tanto más caros.

Inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en la actualidad se impone la arquitectura Harvard.

La arquitectura de von Neumann es una familia de arquitecturas de computadoras que utilizan el mismo dispositivo de almacenamiento tanto para las instrucciones como para los datos.
La arquitectura Harvard emplea dos dispositivos de almacenamiento independientes. Una que contiene sólo instrucciones y otra, sólo datos. Cada una de ellas dispone de sus propios buses y por ello es posible realizar operaciones de lectura/escritura en ambas memorias al mismo tiempo.

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el conjunto de instrucciones que ejecuta la aplicación. Otra parte de memoria es del tipo RAM volátil, y se destina a guardar las variables y los datos. Por lo general, el tipo de memoria ROM suele ser memoria flash pues se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar, es programable en el circuito, más rápida que la E^2PROM tolerando más ciclos de escritura/borrado.

Para la elección final del microcontrolador será necesario conocer: Número de entradas y salidas necesarias. Necesidad de conversores Analógico digital (A/D) o digital analógico (D/A) además de número de canales, resolución y frecuencia de muestreo. Puertos de comunicaciones necesarios ya sean RS-232, USB, CAN bus, I2C, SPI, LIN Ethernet... Cantidad mínima de memoria de programa y de datos. Frecuencia de proceso. Juego de instrucciones RISC/CISC.



Cada fabricante de microcontroladores oferta un gran número de modelos diferentes, desde los más sencillos de 8bits hasta los más potentes de 32bits. Dentro de cada arquitectura es posible seleccionar la capacidad y tipo de memorias, el número de líneas E/S, la velocidad máxima de funcionamiento y otros periféricos. Es por ello que la elección del microcontrolador va a condicionar todo el diseño e influirá en el rendimiento del mismo.

Se pretende utilizar un microcontrolador de la familia Microchip, los motivos son los siguientes:

- Experiencia previa con los microcontroladores más básicos de este fabricante.
- Ya se dispone de herramientas de desarrollo como programadores y depuradores para microcontroladores de este fabricante.
- Dispone de gran variedad de productos.
- Precio de los mismos.
- Resulta sencillo encontrar documentación sobre estos microcontroladores (sobre todo de los más antiguos) Así como códigos de ejemplo y aplicaciones varias.
- Disponiblidad de IDE (Integrated development enviroment) Entorno de desarrollo con compilador para C entre otros.

A continuación se presenta la tabla 4.10 con distintos modelos de microcontroladores PIC de microchip [24]

La condición indispensable es que el microcontrolador disponga de comunicación USB al menos como dispositivo, con esas especificaciones Microchip ofrece un total de

[1]Ksps: Kilo samples per second (Miles de muestras por segundo).

	PIC18F87J50	PIC24FJ256GB110	PIC32MX460F512L
Arquitectura	8bits	16bits	32bits
Memoria de programa (flash)KB	128	256	512
EEPROM	0	0	0
ROM	3904	16384	32768
CPU Mhz	48	32	80
CPU MIPS	12	16	80
Total Pines	80	100	100
Pines E/S	65	84	85
Encapsulado	80/TQFP	100/TQFP	100/TQFP
Timers	2×8bit 3×16bit	5×16bit	5×16bit 1×32bit
Canales A/D (Resolucion) Frecuencia de muestreo	12 (10bit) 100Ksps[1]	16 (10bit) 500Ksps	16 (10bit) 1000Ksps
Comunicaciones	2×A/E/USART 2×SSP(SPI/I2C)	4×UART 3×SPI	2×UART 2×SPI 2×I2C USB

Cuadro 4.10: comparación de tres microcontroladores de Microchip de tres familias deferentes.

45 microcontroladores en sus tres familias PIC18 (8bist) PIC24 (16bits) y PIC32 (32bits) en la tabla 4.10 se muestra una comparación del microcontrolador con más recursos de cada familia.
Respecto al resto de recursos son los siguientes:

- Conversor A/D: El tiempo de medida será menor en tanto en cuanto el tempo de conversión del conversor A/D sea pequeño. Por lo que interesa un conversor A/D rápido.

- Velocidad de la UCP: La obtención de la medida real a partir de los datos leídos con el conversor A/D requiere de un elevado coste computacional, si se quiere que el tiempo de computación sea reducido se debe trabajar con un micro que funcione a una frecuencia elevada, siendo más específicos, que realice un número de operaciones por segundo elevadas.

- Cantidad de memoria: Para realizar la medida se debe disponer de una parametrización del campo generado por el imán en memoria, será una constante. De manera más específica una matriz constante y se puede almacenar tanto en memoria de programa (ROM)como en memoria de datos (RAM) el espacio ocupado por esta tabla suponiendo que se van a utilizar variables tipo short, es decir 16bits y un tamaño de tabla de 100 elementos por cien elementos sería de 100×100×2Bytes=20000Bytes. En aras de conseguir una mayor velocidad es preferible que esta cantidad de datos se pueda guardar en memoria RAM.

Por todo lo anterior el micro elegido para realizar la implementación del prototipo ha sido el PIC32MX460F512L.

4.7.1. Características del PIC32MX460F512L y placas de desarrollo.

Se trata de un microcontrolador de 100 pines en encapsulado 100/TQFP, es decir, de montaje superficial. Para simplificar el desarrollo y disponer de un programador y un debugger se ha adquirido la placa de desarrollo PIC32 USB Starter Borad (Ver Fig 4.18). Se trata de una placa que incorpora el PIC citado, tres pulsadores, tres LEDs, conexión USB para actuar como Dispositivo o Host y un segundo PIC, esta vez un 18F4550 con conexión USB que se encarga de las funciones de programar y depurar. Puede programar y verificar los 512KB de memoria Flash en 9 segundos. Por último dispone de un conector [Hirose: FX10A-120P/12-SV1(71)] desde el cual se tiene acceso a todos los pines de entrada salida.
Para poder acceder a las señales del microcontrolador se ha adquirido también la placa PIC32 I/O Expansion Board. (Ver figura 4.17) que se conecta con la starter Board mediante el conector citado anteriormente. La placa de expansión proporciona acceso total a las señales del microcontrolador y conectores para métodos de programación y depuración alternativos tales como JTAG o ICSP[6].

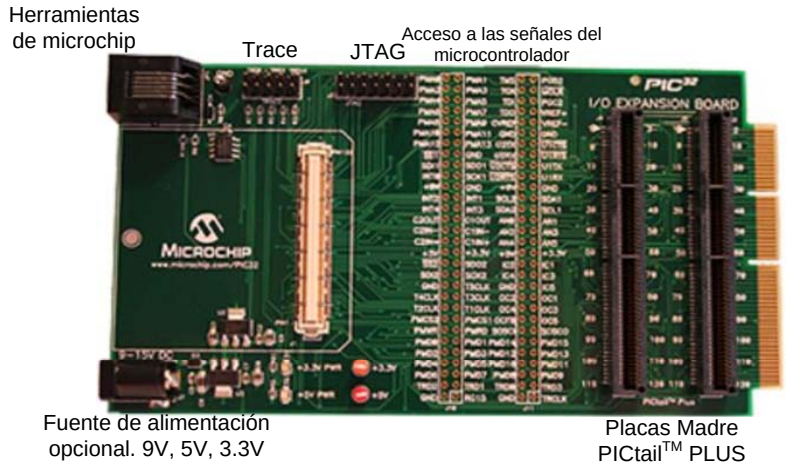


Figura 4.17: Placa de Expansión. PIC32 I/O Expansion Board.

[6]In Circuit Serial Programming

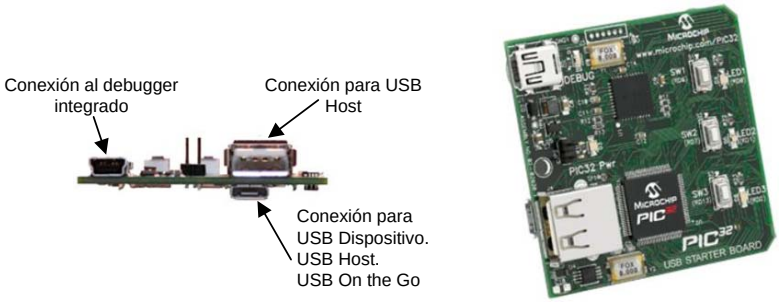


Figura 4.18: Placa de desarrollo PIC32 USB Starter Board.

4.7.2. Programación de un microcontolador PIC32.

A continuación se presenta una pequeña guia de programación del PIC32MX460F512L, para más informción referirse al datasheet del fabricante [24], también puede resultar de gran ayuda el libro [14] que sirve de guí a para programar este tipo de microcontroladores partiendo de un nivel de conocimientos sobre programacion en C y microcontroladores casi nulos.

Configuración del entorno de desarrollo.

Requerimientos del PC:

Para utilizar la placa de desarrollo PIC32 USB starter board son necesarios los siguientes requerimientos:

- Sistema PC-compatible
- Puerto USB disponible o Hub USB alimentado.
- Sistema operativo Microsoft Windows XP®o Windows Vista®. (Este sistema no ha sido probado en Windows NT®ni en Windows 2000®)

Descarga de software:

- MPLAB IDE versión 8.10 o posterior.
- MPLAB C Compiler for PIC32 v1.03 - Student Edition o posterior. Nota: Hay que estar registrado para realizar la descarga.
- Pila para USB dispositivo y Host embebido para PIC32.

Instalar MPLAB IDE:

Localizar el archivo MPLAB IDE descargado en el paso anterior. Relizar doble clic para descomprimirlo y guardarlo en el disco duro. Ejecutar el archivo resultante Install_ MPLAB_Vxxx.exe para comenzar la instalación de MPLAB IDE. Una vez instalado reiniciar el sistema.
Nota: Si se realiza una instalación personalizada, verificar que la casilla “PIC32MX Satrter

Kit" esté seleccionada en la ventana select features durante la instalación.

Si se prefiere utilizar una instalación de MPLAB ya existente (Versión 8.10 o superior), verificar que aparece "PIC32MX Starter Kit" en el menú Debugger-Tools. Si no aparece o está en gris, ejecutar el instalador de MPLAB elegir la opción Modificar y selecionar la casilla "PIC32MX Satrter Kit" como se muestra en la figura de la izquierda.

Instalar el compilador MPLAB C32 C Compiler:

Localizar el archivo MPLAB C Compiler descargado anteriormente y ejecutarlo aceptando todos los pasos.

Instalar los archivos PIC32 USB Starter Board Demo:

Localizar los archivos Pila para USB dispositivo y Host embebido para PIC32 descargados anteriormente,descomprimir el archivo "pic32mx_usb_xxxx.zip" y comenzar el proceso de instalción.

Ha terminado la instalación de software.

Conexión del hardware:

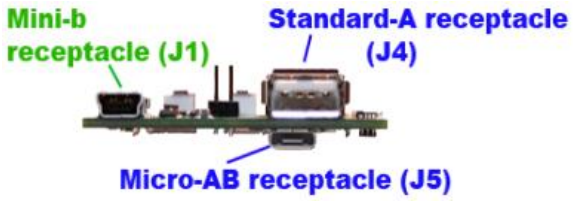


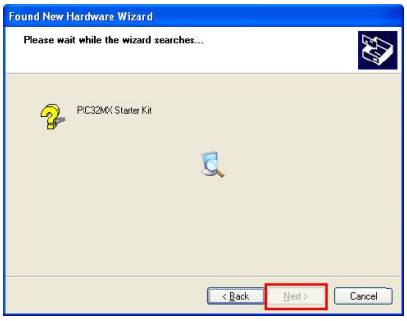
Figura 4.19: Perfil de la placa PIC32 USB Starter Board.

- Localizar el cable USB Mini-b A
- Conectar un extremo al PC
- Conectar el otro extremo al conector J1 de la placa USB Starter Board etiquetado con la palabra "Debug". Esta conexión USB se utiliza exclusivamente para el probramador/depurador y no es accesible para el usuario desarrollando aplicaciones USB, ver figura 4.19.
- Una vez conectada, el LED verde D3 se ilumina, loque indica que la placa está alimentada.

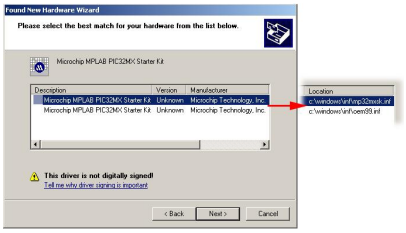
Instalación de drivers:

Cuando la placa se conecta al PC via USB aparece el cuadro de nuevo hardware encontrado. Si se pegunta se desea permitir que windows se conecte a internet para encontrar los driver desmarcar la casilla. Cuando se pregunte si instalar los drivers automáticamente o seleccionarlo de una lista o ubicación especificada, seleccionar “Instalar software automáticamente” y hacer clic en siguiente.

La siguiente ventana le guía a trevés del proceso de búsqueda del driver, este proceso puede tomar varios minutos. Cuando haya terminado haga clic en siguiente. En el caso de que windows no encuentre el driver, este se ubica en C:\Program Files\Microchip\MPLAB IDE\PIC32MXSKit\Drivers\



Si se da a elegir entre varios drivers elegir el archivo OEMxxx.inf



Si aparece el diálogo que indica que no tiene el logo de windows hacer clic en continuar de todas formas.

La última ventana indica que la instalación del software para el Starter Kit ha sido completada, haga clic en finalizar.

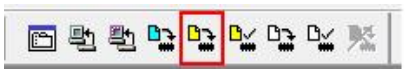
Comenzar un nuevo proyecto:

Una vez instalados los programas anteriores se deben seguir los siguientes pasos para crear un nuevo proyecto:

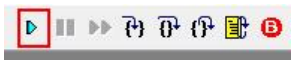
- Ejecutar MPLAB IDE haciendo doble clic en el icono del escritorio. Seleccionar Project->Project wizard, en el diálogo de presentación hacer clir en siguiente, en este cuadro seleccionar del menú desplegable el PIC a programas, en nuestro caso el "PIC32MX460F512L" y hacer clic en siguiente.
- Dado que se va a programar en C se elegirá el compilador Microchip PIC32 C-Compiler Toolsuite del menú desplegable Active Toolsuite y se hará clic en siguiente.
- En el espacio en blanco bajo la etiqueta Create New Project File se hará clic en Browse y se elige la ruta de destino y el nombre de archivo y se hace clic en siguiente.
- Aparece una ventana en la que se pueden añadir los archivos que se desee al proyecto, pero como estamos empezando aún no existe ninguno. Hacer clic en siguiente y después en finalizar.
- A la izquierda de la pantalla aparece un panel en el que se pueden añadir los archivos .c .h .a que componene el proyecto.
- Seleccionar la herramienta de programación, para ello nos vamos al menú Debuger->Select Tool->PIC32Starter Kit

Botón Make all para compilar el proyecto.

Botón Load all memories, para cargar el archivo .hex en el starter kit.



Botón Play para iniciar la ejecución del código.



Habilitación de instrucciones para la mejora del rendimiento.

Configuración de estados de espera para memoria flash:

Por defecto, el bus del nucleo y el bus de la memoria está desacoplado anadiendo a este último un número de tiempos de espera, correspondiente a siete ciclos de reloj, este valor se puede modificar meiante el registro CHECON, concretamente mediante los bits PFMWS, por defecto está configurado en 7 pero se puede disminuir, existe una función perteneciente a la librería de periféricos que configura este valor de forma óptima:

```
SYSTEMConfigWaitStatesAndPB(freq);
```

Donde freq es la frecuencia a la que trabaja el microcontrolador, en nuestro caso 80.000.000L hz

Habilitar cache:

La caché es una memoria muy pequeña y muy rápida diferente a la flash, se activa mediante la función definida en la librería de periféricos:

```
CheKseg0Cache=n();
```

Habilitar instruction Pre-Fetch:

Cuando se activa el Pre-Fetch se leen bloques de instrucciones de cuatro en cuatro, si el código se ejecuta secuencialmente, las siguientes tres instrucciones se ejecutarán con un estado de espera correspondiente a cero.
Se puede activar mediante el siguente macro definido en la librería de periféricos.

```
mCheConfigure(CHECON | 0x03);
```

Se pueden unificar todas las mejoras de rendimiento anteriores mediante una única función definida en la librería de periféricos:

```
SYSTEMConfigPerformance(80000000L);
```

Configuración de puertos como entradas o salidas.

Los puertos, PORT A, PORT B, PORT C pueden actuar como entradas o salidas digitales, y en particular en puerto PORT B está multiplexago con las 16 entradas del conversor analógico digital.

Virtual Address	Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BF88_6000	TRISA	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	TRISA15	TRISA14	—	—	—	TRISA10	TRISA9	—
		7:0	TRISA<7:0>							

Cuadro 4.22: TRISA SFR

Virtual Address	Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BF80_9060	AD1PCFG	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	PCFG15	PCFG14	PCFG13	PCFG12	PCFG11	PCFG10	PCFG9	PCFG8
		7:0	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0

Cuadro 4.23: AD1PCFG SFR

El registro TRISx configura los pines del puerto x para trabajar como entradas o salidas. Un 1 implica entrada, un 0 salida. Por ejemplo:
TRISA=0xFF; los 16[7] pines del puerto A se comportan como entrada.
TRISA=0x00; los 16 pines del puerto A se comportan como salida.
TRISA=0x01; el pin 0 del puerto A se comporta como entrada, el resto como salida.

Caso particular es el puerto B que además de configurarlo como entrada o salida digital hay que configurarlo para saber si está seleccionado como entrada al converdor A/D o como entrada-salida digital, de ello se encarga el registro AD1PCFG, un 0 inplica que el pin está conectado al conversor A/D y un 1 que será entrada o salida digital según se configure mediante TRISB, por ejemplo:
AD1PCFG=0xFF; los 16 pines del puerto B conectados al módulo digital.
AD1PCFG=0x00; los 16 pines del puerto B conectados al módulo analógico.
AD1PCFG=0x01; el pin 0 del puerto B está conectado al módulo digital y el resto al analógico.

Configuración de un timer.

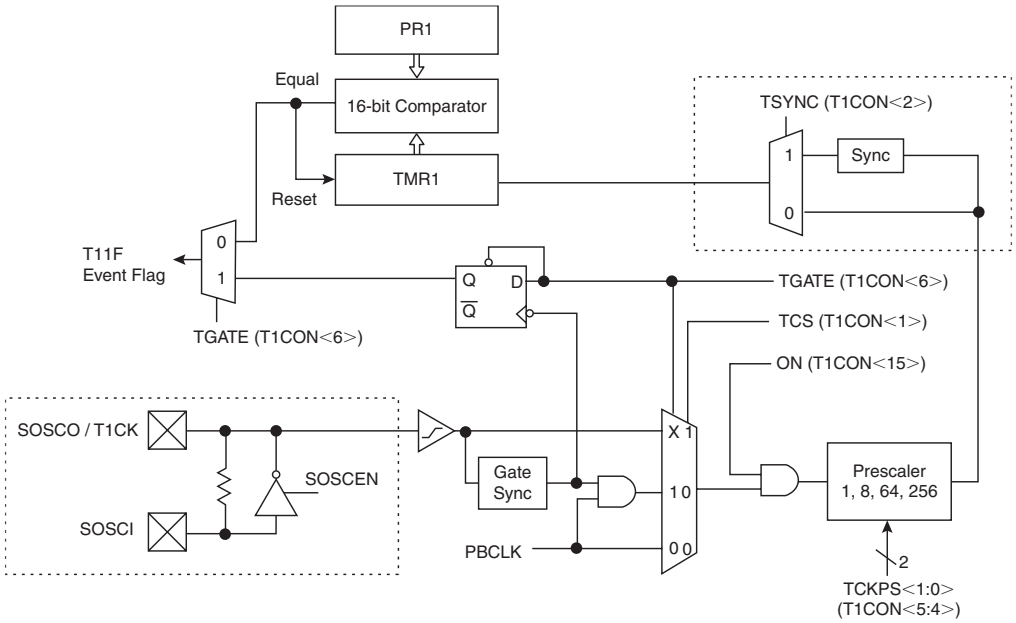


Figura 4.20: Diagrama de bloques del Timer 1 16bits.

[7]En realidad no son los 16, si se observa detenidamente la tabla 4.22 del puerto A que no son configurables como entrada o salida pues son pines con funciones especiales.

Virtual Address	Name		Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
BF80_0600	T1CON	31:24	—	—	—	—	—	—	—	—
		23:16	—	—	—	—	—	—	—	—
		15:8	ON	FRZ	SIDL	TMWDIS	TMWIP	—	—	—
		7:0	TGATE	—	TCKPS<1:0>		—	TSYNC	TCS	—

Cuadro 4.24: T1CON Timer 1 Control Register.

Este microcontrolador dispone de 5 tiemrs de 16 bits y 1 timer de 32 bits. Para la configuración de un timer los registros que entran en juego son:

- TMRx contiene el valor del contador de 16 bits del temporizador x.
- TxCON contiene el control, activación y modo de operación del timer x.
- PRx se puede usar para producir un reset periódico del timer (No requerido en el proyecto).

A continuación se mostrará un ejemplo de funcionamiento para el Timer 1, ver figuras 4.20 y 4.24 Se puede realizar una puesta a cero del temporizador mediante:
TMR1=0;
T1CON=0x8000;
Lo que implica:
TON = 1, el timer está activado TCKPS<1:0>= 00 , TGATE = 0 , TSYNC = 0 El reloj de periféricos directamente como reloj del timer. TCS = 0 El reloj principal sirve como fuente de reloj.

Configuración del conversor analógico/digital y realización de una medida.

El PIC utilizado en este proyecto ofrece hasta 16 pines, (ver figura 4.21) que pueden ser empleados como entradas analógicas, dispone de dos multiplexores en los que seleccionar las entradas. La salida del conversor A/D es de 10bits, con un fondo de escala de 3,3V.

Los pasos para la configuración del conversor A/D son:

- Seleccionar mediante el registro AD1PCFG que pines del puerto B van a ser entradas analógicas, ver tabla 4.23.
- Configurar el registro AD1CON1 para realizar una conversión con tiempo de muestreo automático o manual, AD1CON1=0 tiempo de muestreo manual. AD1CON1=0x00E0 tiempo de muestreo automático.
- Configurar el registro AD1CON2=0 para que las tensiones de referencia para el conversor A/D sean Vss y Vdd.
- Configurar el registro AD1CSSL=0 para que no se produzca un escaneo de canales.
- Configurar el registro AD1CON3 de modo que el tiempo dem uestreo sea al menos 75ns.
- Encender el módulo conversor mediante la instrucción AD1CON1bits.AD=N=1;

Los pasos para realizar una medida analógica una vez configurado el conversor A/D son:

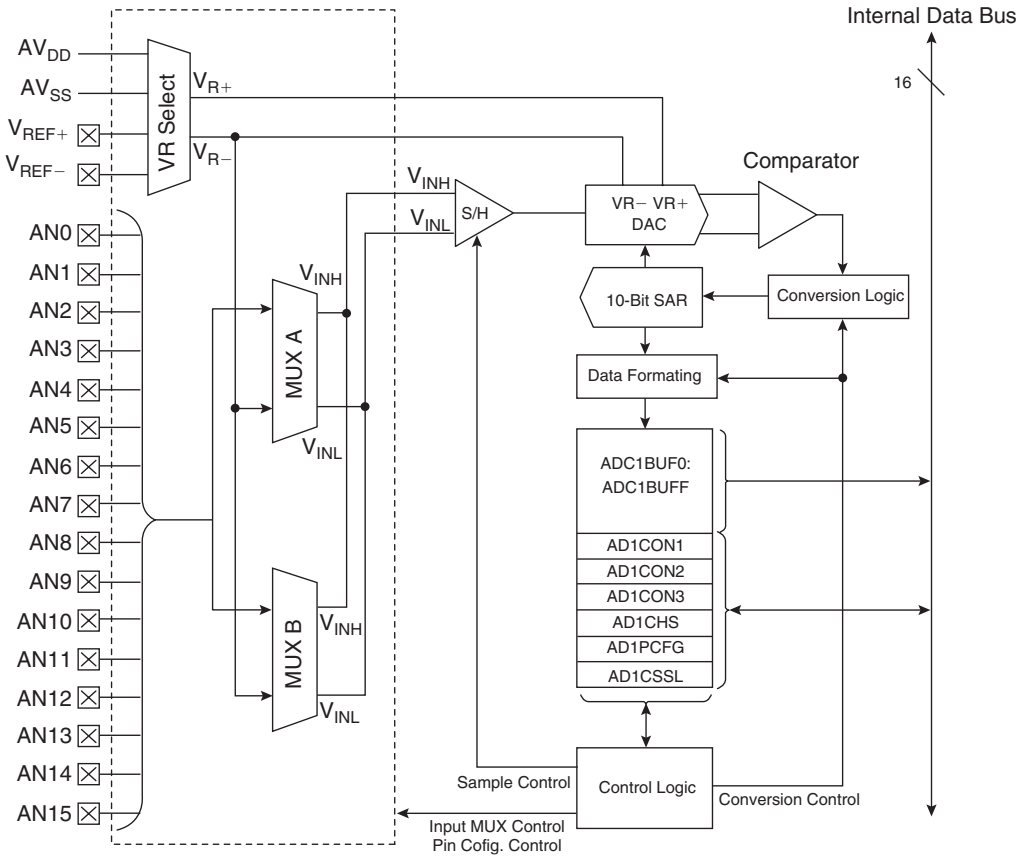


Figura 4.21: Diagrama de bloques conversor A/D 10bits alta velocidad.

- Seleccionar la entrada que se va a medir mediante AD1CHSbits.CH0SA
- Comenzar el muestreo mediante AD1CON1bits.SAMP=1, en el caso de realizar una medida con tiempo de muestreo automático saltarse los siguientes dos pasos.
- Crear una espera activa de al menos 75ms para que de tiempo a realizar un muestreo correcto.
- Comenzar la conversión mediante AD1CON1bits.SAMP=0
- Esperar a que termine la conversión, la conversión habrá terminado cuando AD1CON1bits.DONE tenga un valor distinto de cero.
- Leer el registro ADC1BUFF0, en ese registro se encuentra el valor leido.

4.8. Comunicación.

Una vez resuelta la medida, es necesario que el microcontrolador se conecte a un PC mediante alguno de los periféricos de entrada/salida que incorpora el PC. Es decir, por USB o por RS-232.

4.8.1. Comunicación serie USB.

El protocolo USB [4] esta implementado por un modelo en capas (similar al modelo OSI para redes).
La Capa de Interfaz USB provee la conectividad física, las señalización y la comunicación de paquetes entre el host y el dispositivo.
La capa de dispositivo USB es la lógica del software de sistema para las operaciones de protocolo, como la enumeración. En el dispositivo, en ésta capa se encuentra el Endpoint 0 para las operaciones de configuración.
La capa de función provee una función al dispositivo mediante un cliente Software (ó driver) en el host.
En esta capa se comunica el cliente software con la interfaz formada por varios pipe's para realizar transacciones de interrupción, isócronas y de volumen.

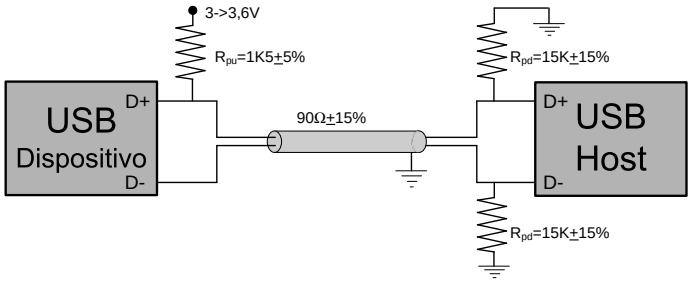
En la tabla 4.26 se muestran los distintos tipos de comunicación USB posibles.

Tipo de transferencia→	Control	Bulk	Interrupción	Isócrona
Uso típico	Identificación y configuración.	Impresoras, scaneres, discos externos	Ratón, teclado.	Audio y video en Streaming.
Bits/milisegundo high speed	15872(treinta y un transacciones de 64 Bytes/microframe)	53248 (trece transacciones de 512 bytes/microframe)	24576 (tres transacciones de 1024 Bytes/microframe)	24576 (tres transacciones de 1024 Bytes/microframe)
Bytes/milisegundo full speed	832 (trece transacciones de 64 Bytes/frame)	1216 (diecinueve transacciones de 64 Bytes/frame)	64 (una transacción de 64-byte/frame)	1023 (una transacción de 1023-byte/frame)
Bytes/milisegundo low speed	24 (tres transacciones de 64 Bytes)	No permitido	0,8 (8 Bytes en 10 milisegundos)	No permitido
Ancho de banda reservado	10 % a low/full speed, 20 % a high speed (minimo)	ninguno	90 a low/full speed, 80 a high speed (maximo)	
Corrección de errores	Sí	Sí	Sí	No
Tiempo de entrega de mensaje garantizado	No	No	No	Sí
Latencia garantizada. Tiempo máximo entre transferencias.	No	No	Sí	Sí

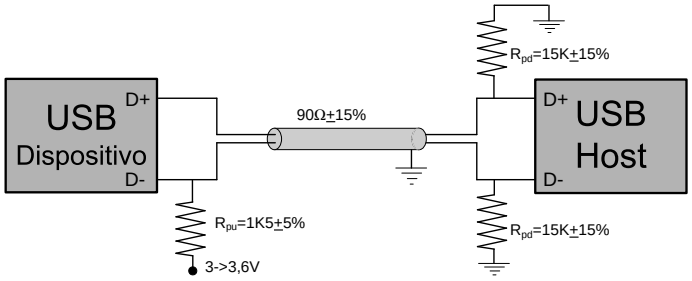
Cuadro 4.26: Tipos de transferencia USB.

Identificación de la velocidad.

El Host reconoce si el dispositivo Usb es Low Speed ó Full Speed a través de la resistencia de pull up conectada al terminal D+(Full speed) ver figura 4.22a o D-(Low Speed) ver figura 4.22b del dispositivo (Device).



(a) Conexión para detección de velocidad full speed 12MB/s.



(b) Conexión para detección de velocidad low speed 1.5MB/s.

Figura 4.22: Detección de full speed/ low speed en una conexión USB.

La norma USB es un tanto extensa así como la configuración del dispositivo de modo que se obviará toda la capa de formato USB (Ver figura 4.23) ya que se ha realizado a partir de un ejemplo que proporciona Microchip, para más información se puede consultar la norma USB en www.usb.org

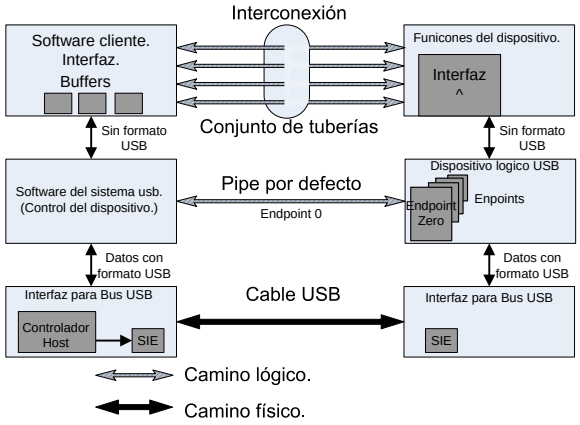


Figura 4.23: Capas del protocolo USB 2.0

Protocolo.

En la comunicación USB sólo se va a utilizar un mensaje que es aquel que el PC envía al PIC la petición de la posición de las cuatro celdas.

Mensaje PC→PIC.

Cabecera	Longitud	Mensaje
P	1	[0]='D'

Cuadro 4.27: Mensaje enviado de PC a PIC via USB

Mensaje Respuesta PC→Matlab.

Cabecera	Longitud	Mensaje
V	13	[0]='D'
		[1]=Posición_0_0X
		[2]=Posición_0_0Y
		[3]=Posición_0_0Z
		[4]=Posición_0_1X
		[5]=Posición_0_1Y
		[6]=Posición_0_1Z
		[7]=Posición_1_0X
		[8]=Posición_1_0Y
		[9]=Posición_1_0Z
		[10]=Posición_1_1X
		[11]=Posición_1_1Y
		[12]=Posición_1_1Z

Cuadro 4.28: Mensaje de respuesta de PIC a PC devolviendo la posición xyz de las cuatro celdas del prototipo.

4.8.2. Comunicación serie RS232.

El RS232 es un protocolo de comunicación serie orientado a caracteres, es decir, un protocolo donde toda la información es enviada por un solo canal bit a bit (un canal para enviar información y otro para recibirla), y donde lo que se envían son caracteres. Por ejemplo, si queremos enviar el número 123, primero tendremos que enviar el carácter 1, seguidamente el 2 y para finalizar el 3, y no el byte que represente el número 123.
Este protocolo está diseñado para distancias cortas, de unos 15 metros mas o menos, y se puede trabajar de forma asíncrona o síncrona y con tipos de canal simplex, halfduplex y fullduplex.

Una conexión RS232 está definida por un cable desde un dispositivo al otro. Hay 25 conexiones en la especificación completa pero en la mayoría de los casos se utilizan menos de la mitad. Los conectores mas utilizados son los DB9 y los DB25 (4.24).

- GND: Valor a 0V. DB-9:5
- Tx: Línea de datos del transmisor al receptor. DB-9:3
- Rx: Línea de datos del receptor al transmisor. DB-9:2
- DTR: Línea por donde el receptor informa al transmisor que está vivo y bien. DB-9:4
- DSR: Línea por donde el transmisor informa al receptor que está vivo y bien. DB-9:6
- RTS: Línea en la que el transmisor indica que quiere enviar algo al receptor. DB-9:7
- CTS: Línea en la que se informa que el receptor está preparado para recibir datos. DB-9:8
- DCD: Línea por la que el receptor informa al transmisor que tiene una portadora entrante. DB-9:1
- RI: Línea en la que se indica que se ha detectado una portadora. DB-9:9

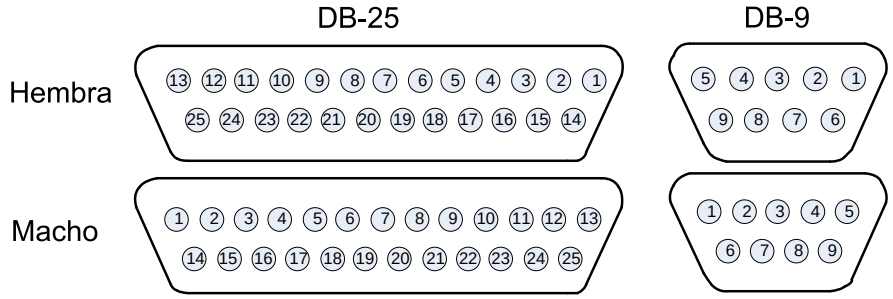


Figura 4.24: Conectores macho y hembra DB-25 y DB-9.

Se ha implementado una comunicación PC→PIC mediante RS-232, para realizar esta comunicación el modelo de PIC utilizado ya dispone de una UART hardware, pero esta UART trabaja con señales TTL, es decir $0 \rightarrow 5V$ mientras que la comunicación RS-232 utiliza niveles de $+12V \rightarrow -12V$, para ello se necesita configurar la UART del PIC, crear una capa física que adapte las señales TTL a niveles RS232 y por ultimo emplear en el PC algún tipo de software para comunicarse por el puerto serie.

Protocolo.

En la comunicación se emplean varios mensajes que serán definidos a continuación:
La cabecera P indica que el mensaje va al PIC, la longitud es el número de bytes que se compone Mensaje.
La cabecera M indica que el mensaje va dirigido al PC, en conctreto a Matlab.

1. Petición del valor de los tres sensores de la primera celda.

   Mensaje Matlab→PIC.

Cabecera	Longitud	Mensaje
P	1	[0]='A'

Cuadro 4.29: Mensaje enviado de PC a PIC.

Mensaje Respuesta PIC→Matlab.

Cabecera	Longitud	Mensaje
M	7	[0]='A'
		[1]=Sensor0 H
		[2]=Sensor0 L
		[3]=Sensor1 H
		[4]=Sensor1 L
		[5]=Sensor2 H
		[6]=Sensor2 L

Cuadro 4.30: Mensaje de respuesta de PIC a PC.

Se divide en bytes la parte alta de la variable de 16 bits y la parte baja. Sensor 1, sensor 2 y sensor 3 están definidos como en la figura 4.7.A.

2. Petición de la posición de la primera celda.

   Mensaje Matlab→PIC

Cabecera	Longitud	Mensaje
P	1	[0]='B'

Cuadro 4.31: Mensaje enviado de PC a PIC de petición de la posición xyz del imán en la primera celda.

Mensaje Respuesta PIC→Matlab

Cabecera	Longitud	Mensaje
M	4	[0]='B'
		[1]=PosicionX
		[2]=PosicionY
		[3]=PosicionZ

Cuadro 4.32: Mensaje de respuesta de PIC a PC devolviendo la posición xyz del imán en la primera celda.

3. Petición del valor de los tres sensores de la primera celda y de la posición.

Esta función de comunicación es un tanto redundante pero se va a utilizar para comprobar que los cálculos realizados con el código implementado en el PIC coinciden con los realizados con el código implementado en Matlab®.

Mensaje Matlab→PIC.

Cabecera	Longitud	Mensaje
P	1	[0]='C'

Cuadro 4.33: Mensaje enviado de PC a PIC de petición del valor leido por el conversor A/D en cada uno de los sensores y la posición del imán.

Mensaje Respuesta PIC→Matlab.

Cabecera	Longitud	Mensaje
M	4	[0]='C'
		[1]=Sensor0 H
		[2]=Sensor0 L
		[3]=Sensor1 H
		[4]=Sensor1 L
		[5]=Sensor2 H
		[6]=Sensor2 L
		[7]=PosicionX
		[8]=PosicionY
		[9]=PosicionZ

Cuadro 4.34: Mensaje de respuesta de PIC a PC devolviendo la posición xyz del imán en la primera celda y el valor de tensión de cada uno de los sensores.

4. Petición de la posición de las cuatro celdas.

Mensaje Matlab→PIC.

Cabecera	Longitud	Mensaje
P	1	[0]='D'

Cuadro 4.35: Mensaje enviado de PC a PIC de petición de la posición xyz de las cuatro celdas del prototipo.

Mensaje Respuesta PIC→Matlab.

Cabecera	Longitud	Mensaje
M	13	[0]='D'
		[1]=Posición_0_0X
		[2]=Posición_0_0Y
		[3]=Posición_0_0Z
		[4]=Posición_0_1X
		[5]=Posición_0_1Y
		[6]=Posición_0_1Z
		[7]=Posición_1_0X
		[8]=Posición_1_0Y
		[9]=Posición_1_0Z
		[10]=Posición_1_1X
		[11]=Posición_1_1Y
		[12]=Posición_1_1Z

Cuadro 4.36: Mensaje de respuesta de PIC a PC devolviendo la posición xyz de las cuatro celdas del prototipo.

Los coeficientes 00 01 10 11 muestran el primer digito la fila y el segundo la columna, de este modo se accede a una celda de las cuatro como se muestra en la figura 4.7.B.

5. Petición del valor de los sensores Hall de las cuatro celdas.

Mensaje Matlab→PIC.

Cabecera	Longitud	Mensaje
P	1	[0]='E'

Cuadro 4.37: Mensaje enviado de PC a PIC de petición del valor de todos los sensores Hall del prototipo.

Mensaje Respuesta PIC→Matlab.

Cabecera	Longitud	Mensaje
M	25	[0]='E'
		[1]=Sensor_0_0_0 H
		[2]=Sensor_0_0_0 L
		[3]=Sensor_0_0_1 H
		[4]=Sensor_0_0_1 L
		[5]=Sensor_0_0_2 H
		[6]=Sensor_0_0_2 L
		[7]=Sensor_0_1_0 H
		[8]=Sensor_0_1_0 L
		[9]=Sensor_0_1_1 H
		[10]=Sensor_0_1_1 L
		[11]=Sensor_0_1_2 H
		[12]=Sensor_0_1_2 L
		[13]=Sensor_1_0_0 H
		[14]=Sensor_1_0_0 L
		[15]=Sensor_1_0_1 H
		[16]=Sensor_1_0_1 L
		[17]=Sensor_1_0_2 H
		[18]=Sensor_1_0_2 L
		[19]=Sensor_1_1_0 H
		[20]=Sensor_1_1_0 L
		[21]=Sensor_1_1_1 H
		[22]=Sensor_1_1_1 L
		[23]=Sensor_1_1_2 H
		[24]=Sensor_1_1_2 L

Cuadro 4.38: Mensaje de respuesta de PIC a PC devolviendo el valor de todos los sensores Hall del prototipo.

Firmware.

El firmware del PIC configura la UART 2 para tener una velocidad de 19200bps, una velocidad lenta en principio pero que a medida que se realicen pruebas varias se irá incrementando hasta su máximo 115200. Se configura para trabajar con 8 bits de datos, sin paridad, un bit de parada y sin control de flujo por hardware.

El PIC pone el puerto serie a la escucha de modo que cuando recibe un mensaje correcto (El primer Byte es P), el segundo la variable tamaño, es el número extra de bytes que deberá recibir, tras recibir el segundo byte recibirá tantos como tamaño indique, todo esto en un tiempo de timeout.

Una vez recibido el mensaje se comprueba su significado, el significado se ha implementado en el tercer byte transmitido, se dispone de cinco tipos de mensaje, A, B, ... E. Comprobando en que caso se encuentra el pic enviará la información requerida. El acceso a esa información se realiza mediante funciones interfaz pues se encuentran en diferentes módulos.

Adaptación de niveles.

El integrado MAX232 (Ver figura 4.25) soluciona la conexión necesaria para lograr comunicación entre el puerto serie de un PC y cualquier otro circuito, con funcionamiento en base a señales de nivel TTL/CMOS. Transforma los niveles TTL a los del estándar RS-232 y niveles RS-232 a TTL, es decir, es un circuito integrado que convierte los niveles de las líneas de un puerto serie RS232 a niveles TTL y viceversa. El circuito integrado posee dos conversores de nivel TTL a RS232 y otros dos que, a la inversa, convierten de RS232 a TTL.
Estos conversores son suficientes para manejar las cuatro señales más utilizadas del puerto serie de la PC, que son TX, RX, RTS y CTS. TX es la señal de transmisión de datos, RX es la de recepción, y RTS y CTS se utilizan para establecer el protocolo para el envío y recepción de los datos. En este caso no hay control de flujo por hardware por lo que únicamente se emplean dos de las señales y dos de los conversores.

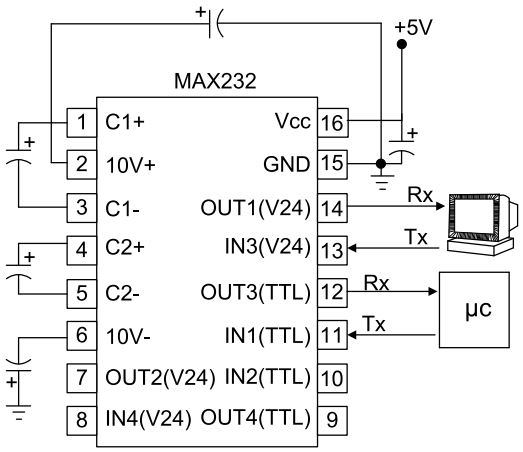


Figura 4.25: Integrado MAX232 de adaptación de niveles TTL (0 5V) a niveles (-12V +12V) compatibles con el puerto RS232 del PC.

Lo interesante es que sólo necesita una alimentación de $5V$, ya que genera, internamente, las tensiones que son necesarias para el estándar RS232. Otros integrados que manejan las líneas RS232 requieren dos voltajes, $+12V$ y $-12V$. La obtención de estas tensiones se realiza mediante lo que se conoce como una bomba de carga mediante los 4 condensadores externos.

Endianness: Big-endian y Little-endian.

Se debe definir como se almacenan y se envían los datos de más de un byte [41]. Por ejemplo, si se quiere almacenar en memoria una variable $unsigned\ short$ de 16 bits que

tiene el valor $255_{10} = 0\text{x}00, 0\text{x}00, 0\text{x}FF, 0\text{x}FF$
Se podría almacenar en memoria de dos modos distintos:

- Comenzando por el byte más significativo MSB $0\text{x}00, 0\text{x}00, 0\text{x}FF, 0\text{x}FF$
- Comenzando por el byte menos significativo LSB $0\text{x}FF, 0\text{x}FF, 0\text{x}00, 0\text{x}00$

El primer formato es Big-endian y el segundo Little-endian.

Big-endian fue promovido por Motorola y quiere decir desde el más grande al final exactamente como se suelen representar los números de manera natural (Decimal).

Little-endian fue promovido por Intel y quiere decir desde el pequeño al final. En este sistema se comienza almacenando el Byte menos significativo, siendo el byte con más peso el último en ser almacenado.

Hay arquitecturas que permiten escoger la "endianness" como por ejemplo DEC Alpha, PA-RISC, IA64, MIPS y ARM PowerPC y se conocen com sistemas "bi-endian" o "middle-endian".

En las comunicaciones de este proyecto cuando es necesario enviar variables que ocupan más de un Byte se emplea el formato Big-endian es decir, primero el Byte más significativo.

Software en el PC para la comunicación.

Se ha decidido programar la comunicación mediante Matlab ya que ofrece una ventaja sobre otros lenguajes de programación y es que permite la presentación de gráficas con gran facilidad. Se han creado varios programas, entre otros uno que muestra una esfera moviéndose en el espacio según la posición del imán, y otro que muestra una gráfica 2D con tres representaciones que son el desplazamiento en cada uno de los ejes. Para más información sobre estos programas ver el anexo.

Capítulo 5

Análisis del sensor.

En este capítulo se pretende comprobar la funcionalidad del sensor diseñado, para ello se va a proceder a realizar unas medidas predeterminadas y observar la salida del sensor.

5.1. Pruebas de rendimiento.

Se van a realizar pruebas de tiempo de ejecución de las dos rutinas principales que componen el código, la conversión analógica digital y la búsqueda de datos en una matriz.

Conversión A/D.

Se va a realizar una medida de tiempo empleado para realizar la conversión analógica con distintos códigos, para comprobar cual es el más eficiente. Para ello se empleará el Timer2 que incorpora el PIC configurado con su registro T2CON$= 0\text{x}8000$ y el microcontrolador trabajando a una frecuencia de 80Mhz. Con la configuración aplicada al timer, el periodo $T_{ck} = \frac{1}{80Mhz} = 1{,}25 \cdot 10^{-8}s = 12{,}5ns$

- **Modo de lectura 1. Tiempo manual de muestreo.**

Se pretende realizar la conversión del valor analógico de un canal con muestreo manual, este proceso implica que primero se realiza un muestreo, y cuando este muestro termina hay que iniciar manualmente la conversión. Para realizar esta medida hay que configurar los registros del microcontrolador correspondientes al conversor AD de la siguiente manera:
AD1PCFG: Máscara de bits para el puerto B, 0 Configura como entrada analógica, 1 como digital.
AD1CON1=0 Secuencia de control de conversión manual.
AD1CSSL=0 El escaneo de canales no se requiere.
AD1CON2=0 Usa MUXA Las referencias positiva y negativa son tomadas de AVdd/AVss, es decir $3{,}3V$ y $0V$.
AD1CON3=0x1F02 Define el tiempo de muestreo. Tad=(2+1)*2*Tpb ->Tad=6*27ns >75ns.
AD1CON1bits.ADON=1; //Enciende el conversor.
Los resultados son los siguientes:

Tiempo: 447 unidades Timer2= 5.59us
Memoria de Programa: 824 Bytes.
Memoria de Datos: 1553 Bytes.

- **Modo de lectura 2. Tiempo de muestreo automático.**

En este caso se configura el conversor de modo que una vez terminado el muestreo comience automáticamente la conversión. Las únicas modificaciones se realizan en el registro: AD1CON1=0x00E0 Inicia la conversión automáticamente después del muestreo. Los resultados son los siguientes:
Tiempo: 447 unidades Timer2= 5.59us
Memoria de Programa 808 Bytes.
Memoria de datos 1553 Bytes.

	Modo1	Modo2
Tiempo [us]	5.59	5.59
Memoria de Programa [Bytes]	824	808
Memoria de Datos [Bytes]	1553	1553
Necesita Timer	Si	No

Cuadro 5.1: Comparación entre el modo 1 y el Modo 2 de conversión.

Es normal que salga el mismo tiempo trabajando en dos modos diferentes. La mayor parte del tiempo es la que se emplea en realizar el muestreo (Sample) y ese tiempo está controlado por el programador, en el modo 1 el tiempo se controlaba mediante el timer 1 y en el modo 2 ese tiempo se controla internamente mediante el número de Tad, El periodo Tad es un múltiplo del periodo Tpb que sería el periodo del bus de periféricos.

El espacio ahorrado en memoria de programa no es un factor decisivo, 16 palabras no es algo a tener en cuenta. Sin embargo, dado que lo único que se busca es rapidez mediante el ahorro de algunas instrucciones y se ha visto que el tiempo tardado es el mismo.

- **Test de lectura 3. Lectura de 9 entradas analógicas secuencialmente en modo manual.**

Básicamente lo que hacemos es reutilizar la función empleada en el test 1 dentro de un bucle for durante 9 veces, recorriendo así del canal analógico 0 al 8.
Los resultados son los siguientes:
Tiempo : 4491 unidades Timer2= 56.14us
Memoria de Programa 840 Bytes
Memoria de Datos 1565 Bytes

- **Test 4 lectura de las 9 entradas analógicas secuencialmente en modo automático.**

Igual que el test 3 pero empleando las funciones de muestreo automático.
Los resultados son:
Tiempo empleado. 4459 unidades Timer2= 55.73us
Memoria de Programa 824 Bytes.
Memoria de Datos 1565 Bytes.

- **Test 5 Realiza 9 conversiones analógicas dentro de la misma función.**

En este test se pretenden eliminar la múltiples llamadas a una función para así evitar el proceso de guardar en la pila distintos registros como el PC[1] y otros. Como contrapartida es posible que el programa ocupe más espacio en memoria. Se creará otra función que devuelva el valor de los 9 canales analógicos. Hay un pequeño problema, y es que las funciones en C solo pueden devolver una variable, y no se pueden usar variables globales pues desde main no se puede acceder al módulo analog. Por eso se debe crear una función interfaz. Lo que va a hacer esta función interfaz es que le pasamos un puntero al array que queremos que nos vuelque los datos del conversor analógico- digital. Y desde el módulo analog los copia. Se comprueba después si se ahorra algo te tiempo de proceso. Pues lo que se desea es optimizar el programa dado que es bastante exigente hablando en términos de tiempo.
Los resultados son:
Tiempo empleado 4615 unidades Timer2= 57.69us
Memoria de Programa: 866 Bytes.
Memoria de Datos: 1576 Bytes.

- **Test 6 Realiza 9 conversiones analógicas dentro de la misma función.**

Este test se basa en la misma teoría que el pero esta vez empleando el modo de tiempo de muestréo automático. A la vista de los Tiempo Los resultados son:
Tiempo empleado. 4675 unidades Timer2= 58.44us
Memoria de programa: 850 Bytes. Memoria de datos: 1576 Bytes.

- **Test 7 Lectura de 9 canales en modo manual sin función interfaz.**

Se pretende eliminar la necesidad de funciones interfaz de modo que la función de lectura ya sea capaz de almacenar los datos sin necesidad de un buffer pasándole un puntero a la zona de memoria.
Los resultados son:
Tiempo empleado 3871 unidades Timer2= 48.39us
Memoria de programa: 836 Bytes.
Memoria de datos: 1564 Bytes.

[1] Program Counter.

- **Test 8 Lectura de 9 canales en modo automático sin función interfaz.**

Se trata del mismo test que el 7, pero esta vez con tiempo de muestreo automático.
Tiempo empleado: 3819 unidades Timer2= 47.71us
Memoria de programa: 820 Bytes.
Memoria de datos: 1564 Bytes.

	Test3	Test4	Test5	Test6	Test7	Test8
Tiempo total [us]	56.14 (100 %)	55.73 (99.29 %)	57.69 (102.76 %)	58.44 (104.10 %)	48.39 (86.19 %)	47.71 (85.04 %)
Tiempo de conversión [us]	—	—	48.68	47.84	—	—
Memoria de Programa [Bytes]	840	824	866	850	836	820
Memoria de Datos [Bytes]	1565	1565	1576	1576	1564	1564

Cuadro 5.2: Comparación entre distintos modos de lectura de 9 canales del conversor AD.

Comprobando los tiempos se concluye que el mejor sistema de medida es el empleado en el Test 8 y será el que se utilice de ahora en adelante.

Búsqueda de datos en matriz.

Se van a crear un código de búsqueda básico, y a partir del mismo se empezarán a realizar modificaciones para intentar optimizarlo, aunque en este caso ya incluso antes de ver los primeros resultados cabe esperar que las mejoras que podamos realizar (mejorar algunos us) no sean muy grandes dada la cantidad de tiempo que ya tarda de por si el algoritmo de búsqueda. También se va a comparar la búsqueda binaria o dicotómica con un búsqueda lineal, de esta forma se observará que se ahorra mucho tiempo de proceso con una búsqueda binaria.

Todos los test se basarán en la realización de tres búsquedas consecutiva, ya que una célula de medida se compone de 3 sensores hall. Los tiempos se miden meidante Timer 2 con la misma configuración que en el apartado anterior, de modo que una unidad del Timer 2 equivale a $T_{ck} = 12{,}5ns$.

- **Test 1.**

Crear una función de búsqueda binaria, ver cuanto tarda en hacer una llamada y hacer tres llamadas a la misma.
Una llamada(Un valor):
Tiempo: 1071 unidades Timer2= 13.38us.
Memoria programa: 946 Bytes.

Memoria datos:1565 Bytes.

Tres llamadas(Tres valores):
Tiempo:3439 unidades Timer2=42.99us
Memoria programa:976 Bytes.
Memoria datos: 1568 Bytes.

- **Test 2. Algoritmo_de_búsqueda_Busca_los_3_valores_sin_punteros.**

Lo que se pretende en este test es evitar realizar tres llamadas a la función de búsqueda ya que para buscar una célula entera hay que buscar los valores de los tres sensores, entonces la idea es crear una función que nos permita buscar los tres valores y los devuelva en una sola llamada y sin emplear punteros.
Una llamada (Tres valores):
Tiempo: 3319 unidades Timer2=41.88us.
Memoria programa: 977 Bytes.
Memoria datos:1567 Bytes.

- **Test 3. Algoritmo_de_búsqueda_Busca_los_3_valores_con_punteros.**

Se basa en la misma filosofía que el test 2 pero emplearemos directamente los punteros y no variables intermedias.
una sola llamada y sin emplear punteros.
Tiempo: 4099 unidades Timer2=51.24us.
Memoria programa: 1003 Bytes.
Memoria datos:1567 Bytes.

Conclusión del test 3, Parece necesario analizar por que una función que utiliza punteros ha tardado más tiempo que una que no los utiliza. La causa de este comportamiento es que cada vez que se accede a una zona de memoria apuntada por un puntero hay que realizar una suma que sería la dirección de memoria que contiene el puntero más el índice del elemento al que se pretende apuntar. Esto carga el microprocesador con gran cantidad de sumas que aunque cada una de ellas no supone un gran incremento en el tiempo de ejecución, este valor multiplicado por el número de accesos al final pasan factura.

- **Test 4.**

Test 4 Viendo los problemas que presenta el test 3 se van a intentar solucionar mediante el incremento del puntero una sola vez y guardándolo. Aún así se aumenta la carga de proceso, así que no se seguirá por este camino de investigación.

Tiempo: 3719 unidades Timer2=46.49us.
Memoria programa: 990 Bytes.
Memoria datos:1567 Bytes.

- **Test 5 Búsqueda lineal.**

Se va a programar una búsqueda sin optimización alguna, simplemente teniendo en cuenta que los valores están ordenados de mayor a menor. Lo que hará es ir avanzando por el array de uno en uno hasta llegar al intervalo en que se encuentra el valor buscado.

Tiempo: 5631 unidades Timer2=70.38us. Puede ser un tiempo medio, dado que depende enormemente de la posición en la que se encuentre, si se encuentra en posiciones iniciales del array el tiempo será mucho menor, y si se encuentra cerca del final, el tiempo será mayor.
Memoria programa: 957 Bytes.
Memoria datos:1567 Bytes.

	Test1	Test2	Test3	Test4	Test5
Tiempo [us]	42.99 (100 %)	41.88 (96.51 %)	51.24 (119.19 %)	46.49 (108.14 %)	70.38 (163.74 %)
Memoria de Programa [Bytes]	976 (100 %)	977 (100.10 %)	1003 (102.77 %)	990 (101.43 %)	957 (98.05 %)
Memoria de Datos [Bytes]	1568 (100 %)	1567 (99.94 %)	1567 (99.94 %)	1567 (99.94 %)	1567 (99.94 %)

Cuadro 5.3: Comparación de algoritmos de búsqueda.

Comprobando los tiempos se concluye que el mejor sistema de busqueda es el empleado en el Test 2 y será el que se utilice de ahora en adelante.

5.2. Pruebas de funcionamiento.

Las siguientes gráficas corresponden a las pruebas realizadas para el sensor de posición/fuerza diseñado. De modo que ante cada una de las perturbaciones que se indican se muestran tres trazas. La traza roja indica la coordenada x en la que se encuentra el centro del imán, la traza verde la coordenada y, por último la traza azul indica la coordenada z.

Se va a proceder a deformar el sensor en cada una de las tres direcciones del espacio cartesiano y también medidas compuestas. Todas estas medidas se han realizado de manera manual, siendo los elementos de medida un pie de rey, folio, bolígrafo y vista ortogonal del ojo humano. A pesar del precario sistema de medición el sensor ha proporcionado unos resultados aceptables.

- **Desplazamiento único en z.**

En reposo, la coordenada z tiene un valor de $18{,}7mm$. Se ejerce una fuerza de modo que la coordenada z disminuya hasta $15mm$, esto supone un $\Delta z = -3{,}7mm$. Los resultados se muestran en la figura 5.1a.

Se observa que está midiendo un mínimo de 11.75mm en lugar de 15mm. Las coordenadas x e y deberían mantenerse constantes pero también se observa que varía del siguiente modo:
La coordenada x varía entre 0 y $-1{,}07mm$.
La coordenada y varía entre $5{,}20$ y $3{,}60mm$.

- **Desplazamiento único en x.**

En esta prueba se realiza un desplazamiento en el eje x intentando mantener constante la distancia y y la distancia z. El desplazamiento en x, $\Delta x = +3mm$ se muestra en la figura 5.1b.

Se observa el desplazamiento deseado en el eje x. x varía entre $-0{,}53$ y $3{,}47mm$, es decir, $\Delta x = 4{,}1mm$ Pero también se observan desplazamientos no deseados en los ejes y y z:
y varía entre $2{,}80$ y $4{,}67mm$.
z varía entre $18{,}69$ y $19{,}40mm$.

- **Desplazamiento único en y.**

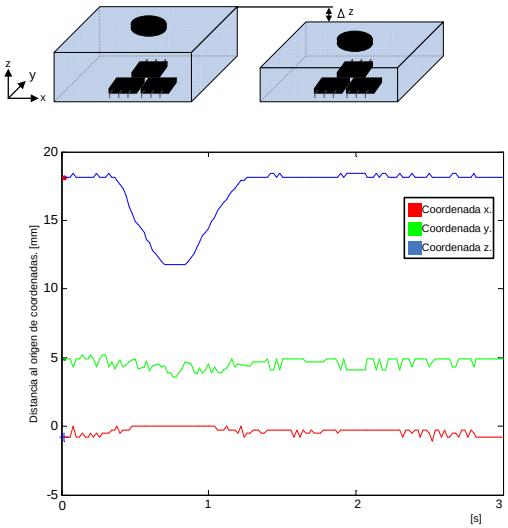
Incremento de y $\Delta y = 3mm$. Se realiza un desplazamiento en el eje y manteniendo constantes las coordenadas x y z aunque debido a las restricciones físicas del material esto no es $100\,\%$ posible. Y al realizar el desplazamiento en y la espuma obliga a realizar un pequeño desplazamiento en z. Se muestra en la figura 5.1c.

Se observa una variación deseada en el eje y, este varía entre $4{,}00$ y $9{,}21mm$, es decir, $\Delta y = 5{,}21mm$. También se observan variaciones no deseadas de x y z.
x varía entre $-2{,}4$ y $-0{,}5$.
z varía entre $16{,}02$ y $18{,}96$.

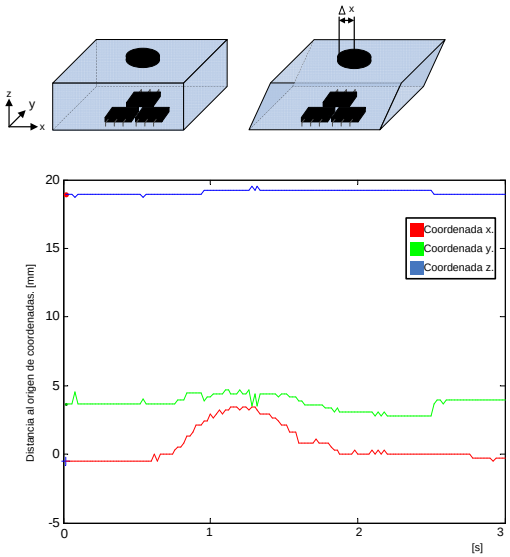
- **Desplazamiento conjunto en x y.**

Resulta imposible medir con el calibre un desplazamiento en xy, pero se ha realizado de manera aproximada para tener una gráfica cualitativa, más que cuantitativa. Ver figura 5.1d.

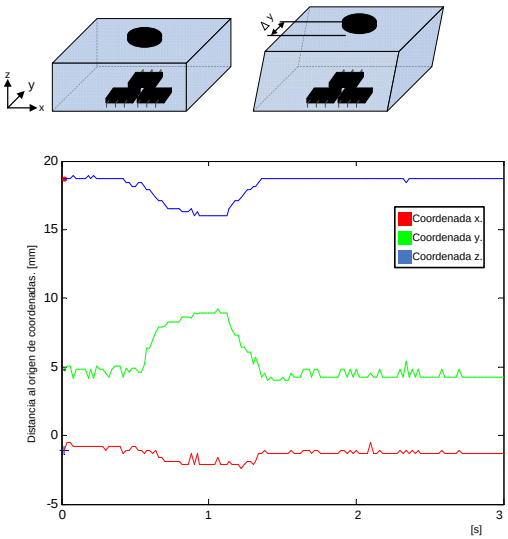
Se realiza un incremento de x y un incremento de y, en la gráfica se observa una respuesta correcta ante ese tipo de entradas.



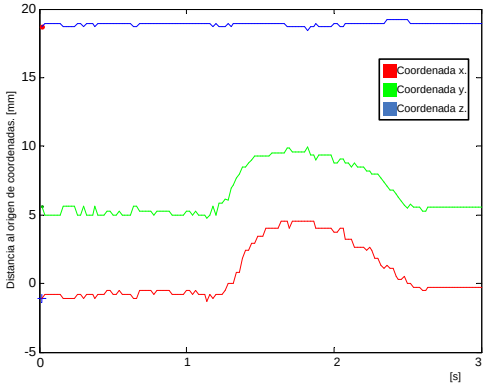
(a) Respuesta del sensor ante un desplazamiento en z.



(b) Respuesta del sensor ante un desplazamiento en x.



(c) Respuesta del sensor ante un desplazamiento en y.



(d) Respuesta del sensor ante un desplazamiento en xy.

Figura 5.1: Gráficas de las pruebas realizadas para una celda sensora de desplazamientos en distintos ejes.

Capítulo 6

Conclusión

6.1. Resumen y conclusión.

El sensor está formado por una matriz de celdas compuestas de tres sensores de efecto Hall y un imán permanente. Para sostener cada par de componentes imán-sensor, un material elástico, que realiza las funciones de interfaz entre los componentes y medio de soporte para realizar la medida.

El objetivo del sensor es medir el movimiento de un plano de contacto respecto de otro inicialmente paralelo. Se pretende obtener una imagen de los diferenciales de fuerzas de contacto en las tres direcciones del espacio cartesiano. Ésto no sólo permitirá registrar irregularidades o contornos de lo que presione el sensor, sino direcciones de las fuerzas aplicadas, posibles cambios de material, deslizamientos, etc. El origen de este sensor es una solución para una de las primeras posibles aplicaciones, un pie para un robot humanoide. El pie realiza un contacto con el suelo cada vez que realiza una pisada y un material elástico en la suela absorberá el contacto. Este fue el origen del sensor, la necesidad de medir las deformaciones del material elástico para poder mejorar los algoritmos de control del robot. Con el sensor se puede conocer cuan irregular es el suelo, qué porción del área del pie realiza el contacto, una información complementaria para localizar el punto de equilibrio debido a la mejor información de la distribución de fuerzas, si el contacto está resbalando por la no medida de fuerzas tangenciales esperadas, etc.

El funcionamiento del sensor desarrollado, consiste en la medición en cada celda, o unidad de medida, del movimiento relativo de un imán, o chivato, provocado por la aparición de una fuerza sobre el material flexible que lo sustenta. La variación del campo magnético es registrado por tres sensores Hall ubicados en el plano fijo y rígido del sensor. El diámetro del imán debe ser tal que cubra los tres sensores Hall, los cuales están dispuestos de tal manera que las áreas de detección están localizadas sobre los vértices de un triángulo equilátero, cuyo centro coincide con el del imán. Aprovechando la linealidad del campo con esta disposición, se puede calcular la posición del imán con respecto a los sensores mediante trilateración. Para realizar la medida, se ha simulado el campo magnético generado por el imán y guardado los datos en una matriz. Con el valor de los tres sensores Hall

y buscando en la matriz de datos simulada anteriormente se reduce el problema de 3D a 2D. Empleando la trilateración en el espacio bidimensional (en un plano que se consigue gracias a la búsqueda en una solo fila de la matriz de datos) ocurre que las tres esferas, de radio el valor de cada sensor Hall, no se cortan en un punto exacto; a esta diferencia la llamamos error. Minimizando el error se resuelve la posición de manera numérica.

Existen arrays de sensores táctiles y algunos construidos sobre un sustrato elástico, pero la mayoría están basados en sensores piezoeléctricos, variación de la capacidad, sensores resistivos y muchos sólo miden la fuerza normal a la superficie.

6.2. Posibles mejoras.

Se ha desarrollado un sensor de fuerza en tres dimensiones basado en tres sensores Hall. Los resultados mostrados en el apartado anterior muestran que el funcionamiento es aceptable teniendo en cuenta los precarios métodos de fabricación que se han empleado y el método de comprobación. Sin embargo, son muchas las mejoras que pueden realizarse para conseguir mejorar esta idea.

El principal error en la medida se produce debido a las irregularidades del imán. El imán ha sido modelado como un solenoide de las mismas dimensiones que el imán por el que circula un corriente i_{eq} como se muestra en la ecuación 3.17 pero los imanes de propósito general (ferretería) que se han comprado, puesto que no han sido fabricados para este tipo de tareas no presentan un campo magnético tan similar al modelado. Esto puede haber ocurrido por las temperaturas de almacenamiento, o por haber sido almacenados junto con otros imanes que hayan modificado la estructura interna del imán. La mejora propuesta es la utilización de imanes especialmente diseñados para este propósito con un proceso de fabricación y transporte más cuidado. Otra solución sería emplear directamente bobinas en lugar de imanes, el problema es que estos elementos deberían ir alimentados y complicaría el diseño.

También se debe trabajar en el diseño de un soporte físico flexible que permita trabajar con fuerzas equivalentes a las de la pisada de un robot. Pues de momento este soporte físico es una lámina de espuma.

Se deberá trabajar en un método para colocar los sensores hall en su posición con precisión. En el prototipo se ha diseñado una PCB cono Orcad Layout para determinar la posición de los taladros, estos taladros se han realizado de manera automática con precisiones de $0{,}1mm$ pero al soldar cada uno de estos sensores a la PCB se ha hecho a mano y no garantiza la exactitud de los mismos.

Bibliografía

[1] S.L. Aiman GZ. http://www.aimangz.com, 2009.

[2] Mª Auxilio Recasens Bellver and José González Calabuig. *"OrCAD CAPTURE y LAYOUT V.9.2"*. T H O M S O N.

[3] Bombal, Marin, and Vera. *"Problemas de Análisis matemático: Cálculo Diferencial"*. ed. AC, 1988.

[4] Universal Serial Bus. http://www.usb.org/developers, 2009.

[5] OrCAD CADENCE. http://www.cadence.com/products/orcad, 2009.

[6] Andrés Cano Sánchez. *"Diseño de la arquitectura hardware del Robot Humanoide RH-2"*. PFC UC3M, 2008.

[7] Z. Chu, P.M. Sarro, and S. Middelhoek. "Silicon three-axial tactile sensor". *Sensors and Actuators A 54 (1996) 505-510*, 1996.

[8] Algoritmia. Algoritmos de búsqueda. http://www.algoritmia.net/articles.php?id=32, 2009.

[9] La BOUTIQUE del IMAN. http://www.laboutiquedeliman.com, 2009.

[10] Universidad del Pais Vasco. http://www.sc.ehu.es/sbweb/fisica/elecmagnet/magnetico/cMagnetico.html, 2009.

[11] R. L. Feller, C. K. Lau, C. R. Wagner, D. P. Perrin, and R. D. Howe. "The Effect of Force Feedback on Remote Palpation". *IEEE International Conference on Robotics & Automation New Orleans*, 2004.

[12] A.W. Goodwin. "Extracting Object Shape from Nerve Fiber Responses". *Neural Aspects of Tactile Sensation. J.W. Moeley (Editor), Elsevier Science B.V. pp 55-87.*, 1998.

[13] Kensuke Harada, Shuuji Kajita, Kenji Kaneko, and Hirohisa Hirukawa. "ZMP Analysis for Arm/Leg Coordination.". *Intl. Conference on Intelligent Robots and Systems*, 2003.

[14] Lucio Di Jasio. *Programming 32-bit Microcontroles in C. Exploring the PIC32*. ELSEVIER, 2008.

[15] JR3. http://www.jr3.com/about.html, 2009.

[16] S. Kagami, K. Nishiwaki, J. Kunffner, S. Thompson, J. Chestnutt, M. Stilman, and P. Michel. "Humanoid HRP2-DHRC for Autonomous and Interactive Behavior". *Robotics Research, STAR 28, pp. 103 117. Springer-Verlag Berlin Heidelberg*, 2007.

[17] Hyung Joo Kim, Emily Horn, YuJiang Xiang, Karim Abdel Malek, and Jsabir S. Arora. "Dynamic Motion Prediction of Gait and Lifting".

[18] Kunnyun Kim, Kang Ryeol Lee, Dae Sung Lee, Nam-Kyu Cho, Won Hyo Kim, Kwang-Bum Park, Hyo-Derk Park, Yong Kook Kim, Yon.Kyu Park, and Jong-Ho Kim. "A silicon-based flexible tactile sensor for ubiquitous robot companion applications". *Journal of Phisics: Conference Series 34 399-403 International MEMS Conference*, 2006.

[19] Genichiro Kinoshita, Tomonori Kimura, and Makoto Shimojo. "Dynamic Sensing Experiments of Reaction Force Distributions on the Sole of a Walking Humanoid Robot". *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada*, 2003.

[20] H Liu, Meusel, and Hirzinger. "A tactile sensing system for the DLR Three-Finger Robot Hand". *ISMCR, 1995, pp 91-96*, 1995.

[21] Rafael Iñigo Madrigal and Enric Vidal Idiarte. *"Robots industriales manipuladores"*. Edicions UPC, 2002.

[22] T. Matsunaga, K. Totsu, M. Esashi, and Y. Haga. "Tactile Display for 2-D and 3-D Shape Expression Usinf SMA Micro Actuators". *IEEE EMBS Special Topics Conference on Microtechnologies in Medicine and Biology Kahuku, Oahu, Hawaii*, 2005.

[23] Tao Mei, Wen J. Li, Yu Ge, Yong Chen, Lin Ni, and Ming Ho Chan. "An integrated MEMS three-dimensional tactile sensor with large force range". *Sensors and Actuators 80 (2000) 155-162*, 1999.

[24] Microchip. http://www.microchip.com, 2009.

[25] Domínguez Muro and Mariano. *Trigonometría activa: 2 BUP*. Ediciones Universidad Salamanca (ed.), 1985.

[26] Masahiro Ohka, Hiroaki Kobayashi, and Yasunga Mitsuya. "Sensing Characteristics of an Optical Three-axis Tactile Sensor Mounted on a Multi-fingered Rrobotic Hand". *IEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[27] Masahiro Ohka, Yasunga Mitsuya, Isamu Higashioka, and Hisanoir Kabeshita. "An Experimental Optical Three-axis Tactile Sensor". *Robotica volume 23, pp 456-465 Cambridge University Press*, 2005.

[28] Masahiro Ohka, Yasunga Mitsuya, Yasuuaki Matsunaga, and Shuichi Takeuchi. "Sensig characteristics off an optical three-axis tactile sensor under combines loading". *Robotica volume 22, pp. 213-221 Cambridge University Press*, 2004.

[29] J.L. Pedreño-Molina, A.Guerrero-González, and J. López-Coronado. "Estudio de los sensores táctiles artificiales aplicados a la robótica de agarre". *Universidad Politécnica de Cartagena*.

[30] Pedro Plaza Merino. *"Diseño y realización del control de un convertidor CC/CC basado en el microcontrolador PIC "*. PFC UC3M, 2007.

[31] XP Power. http://www.xppower.com/, 2009.

[32] INC Pressure Profile Systems. http://www.pressureprofile.com/case-study-esoles, 2009.

[33] S. Sagisawa, T. Shinokura, and M. Kobayashi. "Three Direction Sensing Tactile Sensor". *F u j i E l e c t r i c Corporate Kesearch and Development. LTD Kanagawa, Japan*.

[34] Raúl Sanz Díaz. *"Diseño de sistema de sensorización basado en microprocesador PIC para simulador de vuelo : helicóptero AS-350-B2"*. PFC UC3M, 2005.

[35] Ravinder S.Dahiya, Maurizion Valle, Giorgio Metta, and Leandro Lorenzelli. "Tactile Sensing Arrays foor Humanoid Robots". *ROBOTCUP*, 2004.

[36] Jae S. Son, Mark R. Cutkosky, and Robert D. Howel. "Comparison of Contact Sensor Localization Abilities During Manipulation". *Harvard University, Division of Applied Sciences Pierce Hall, Cambridge, Massachusetts, 02138 USA*.

[37] Kitti Suwanratchatamanee and Mitsuru Matsumoto. "A Simple Tactile Sensing Foot for Humanoid Robot Active Ground Slope Recognition". *IEEE International Conference on Mechatronics. Malaga, Spain, April 2009.*, 2009.

[38] Kitti Suwanratchatamanee, Ryo Saegusa, Miitsuharu Matsumoto, and Shuji Hashimoto. "A simple tactile sensor system for robot manipulator and objet edge shape recognition". *Publicación desconocida*, 2007.

[39] Satooru Takenawa. "A Soft Three-Axis Tactile Sensor based on Electromagnetic Induction". *IEEE International Conference on Mechatronics. Malaga, Spain.*, 2009.

[40] Paul A. Tipler. *"Física" Tomo 2*. Editorial Reverte, S.A.

[41] Dr. William T. Verts. "An Essay on Endian Order". 1996.

[42] M. Vukobratovic and D. Juricic. "Contribution to The Synthesis of Biped Gait". 1969.

[43] Wikipedia. http://es.wikipedia.org/wiki/equilibriocepción, 2009.

[44] Wikipedia. http://es.wikipedia.org/wiki/triangulación, 2009.

[45] Wikipedia. http://es.wikipedia.org/wiki/trilateración, 2009.

[46] Hanafiah Yussof, Masahiro Ohka, Hiroaki Kobayashi, Jumpei Takata, Mitsuhiro Yamano, and Yasuo Nasu. "Development of an Optical Three-Axis Tactile Sensor for Objet Handing Tasks in Humanoid Robot Navigation System". *Studies in Computational Intelligence (SCI) 76, 43-51*, 2007.

ANEXO

6.3. Código.

Código en C para PIC32MX460F512L.

Funciones contenidas en el archivo “analog.c”

```
1  //************************************
2  //analog.c                            *
3  //************************************
4  #include "analog_variables_y_definiciones.h"
5  /*************************************************************
6   * Función: void initADC(int analog_mask)
7   *  Esta funcion inicializa el periférico conversor analógico digital.
8   *
9   * Pre condiciones: Los pines que se quieran emplear como variables
        analógicas previamente deben ser configurados como entradas
        mediante el registro TRISB.
10  *
11  * Entradas: int analog_mask: Máscara de 16 bits, un 0 configura el
        pin del puerto B como analógico mientras que un 1 lo hace como
        digital.
12  *
13  * Salidas: Ninguna.
14  *
15  * Otros efectos: Determina el modo de realizar las lecturas
        analógicas.
16  *
17  * Funcionamiento: analog_mask indica cuales de los pines del puerto B
        se emplean como entradas analógicas, esta inicialización del
        conversor analógico digital requiere dos pasos para realizar la
        medida, uno es el muestreo y otro la conversión. Las referencias
        para el conversor A/D son GND y Vdd, es decir, que el conversor
        A/D va a medir entre 0V y 3.3V.
18  *
19  * Nota: Si se emplea el puerto B como entrada / salida digital se
        debe saber que este sólo soporta un máximo de 3.3V y no 5V.
20  *************************************************************/
21 /*
22 void initADC_manual(int analog_mask)     // Analog mask dirá cuales de
       los pines del puerto B se emplean como entradas digitales y cuales
       se emplearán como entradas analógicas
23     // Un 0 lo configura como entrada analógica
```

```
24      // Un 1 lo configura como entrada digital
25      //NOTA: Estas entradas aunque se empleen como entradas digitales
            solo soportan 3.3V máximo.
26 {
27 AD1PCFG=analog_mask;
28 AD1CON1=0;   //Secuencia de control de conversión manual.
29 AD1CSSL=0;   //El escaneo de canales no se requiere
30 AD1CON2=0;   //Usa MUXA Las referencias positiva y negativa son tomadas
       de AVss/AVdd
31 AD1CON3=0x1F02; //Tad=(2+1)*2*Tpb -> Tad=6*27ns > 75ns
32 AD1CON1bits.ADON=1; //Enciende el conversor.
33 }
34 */
35
36
37 /*********************************************************
38  * Función: void initADC_automatic_sampling_timing (int analog_mask)
39  *  Esta funcion inicializa el periférico conversor analógico digital.
40  *
41  * Pre condiciones: Los pines que se quieran emplear como variables
       analógicas previamente deben ser conficurados como entradas
       mediante el registro TRISB.
42  *
43  * Entradas: int analog_mask: Máscara de 16 bits, un 0 configura el
       pin del puerto B como analógico mientras que un 1 lo hace como
       digital.
44  *
45  * Salidas: Ninguna.
46  *
47  * Otros efectos: Determina el modo de realizar las lecturas
       analógicas.
48  *
49  * Funcionamiento: analog_mask indica cuales de los pines del puerto B
       se emplean como entradas analógicas, esta inicialización del
       conversor analógico digital REQUIERE UN SOLO PASO PARA REALIZAR LA
       MEDIDA PUES EL TIEMPO DE MUESTREO ES AUTOMÁTICO Y UNA VEZ HA
       TERMINADO COMIENZA AUTOMÁTICAMENTE LA CONVERSIÓN. Las referencias
       para el conversor A/D son GND y Vdd, es decir, que el conversor
       A/D va a medir entre 0V y 3.3V.
50  *
51  * Nota: Esta función se diferencia de void initADC(int analog_mask en
       que con la función actual la lectura de conversor A/D se realiza
       en un solo paso mientras que con la anterior se realiza en dos.
52  *********************************************************/
53  void initADC_automatic_sampling_timing (int analog_mask)
54 {
55 AD1PCFG=analog_mask;
56 AD1CON1=0x00E0; //Conversion automatica después del muestreo.
57 AD1CSSL=0;      //El escaneo de canales no se requiere
58 AD1CON2=0;      //Usa MUXA Las referencias positiva y negativa son
       tomadas de AVss/AVdd
59 AD1CON3=0x1F02;
60 //Este es el original AD1CON3=0x1F3F;    //  02->Tad=(2+1)*2*Tpb ->
       Tad=6*27ns > 75ns
```

```
61                                          //  1F--> 31*Tad.
62 AD1CON1bits.ADON=1; //Enciende el conversor.
63 }
64
65
66 /*********************************************************
67  * Función: unsigned char lee_analog_manual(unsigned char canal)
68  *  Esta funcion realiza una lectura analógica.
69  *
70  * Pre condiciones: El conversor A/D debe haber sido inicializado con
       la función void initADC_manual(int analog_mask) y el canal que se
       pretende leer debe haber sido configurado como entrada analógica.
71  *
72  * Entradas: unsigned char canal:Indica el canal que se quiere leer.
73  *
74  * Salidas: ADC1BUF0>>2: Que es la variable que proporciona el
       conversor A/D tras la lectura pero con 8 bits en lugar de 10.
75  *
76  * Otros efectos: Utiliza e inicializa el Timer 1.
77  *
78  * Funcionamiento: Esta función realiza una lectura simple del valor
       del conversor A/D teniendo como parámetros de entrada el canal que
       se quiere leer y devuelve la variable en la que se va a guardar.
79  *
80  * Nota: Esta función devuelve una variable de 8 bits, los 8 bits más
       significativos. Así, para una tensión de 0V la salida será 0x00 y
       para una tensión de 3.3V la salida será 0xFF.
81  *********************************************************/
82 /*
83 unsigned char lee_analog_manual(unsigned char canal)
84 {
85 AD1CHSbits.CH0SA=canal;      //Selecciona el canal del que se va a
       hacer la conversión.
86 AD1CON1bits.SAMP=1;          //Comienza el muestreo
87 T1CON=0x8000;                //Configura el tiempo de muestreo, lo
       único que hace es encender el timer con un preescalado de 1:1, y la
       frecuencia la del bus de periféricos que en un principio es de
       74Mhz/2.
88 TMR1=0;                     //Pone a 0 la cuenta del timer
89 while (TMR1<100);           //En el momento que la cuenta del timer
       sea de 100 saldrá del bucle y habrá pasado el tiempo de muestreo.
90 AD1CON1bits.SAMP=0;         //Comienza la conversión
91 while (!AD1CON1bits.DONE); //
92 return(ADC1BUF0>>2);        //Cuando ha acabado la conversión devuelve
       el valor del registro.
93                             //Es como un shift right El >>2 hace que
                                   el valor del A/D pase a ser una
                                   variable de 8 bits.
94 }
95 */
96
97
98 /*********************************************************
99  * Función: unsigned char lee_analog_automatic_sampling_timing(canal)
```

```
100  *  Esta funcion realiza una lectura analógica.
101  *
102  * Pre condiciones: El conversor A/D debe haber sido inicializado con
       la función void initADC_automatic_sampling_timing (int
       analog_mask) y el canal que se pretende leer debe haber sido
       configurado como entrada analógica.
103  *
104  * Entradas: unsigned char canal:Indica el canal que se quiere leer.
105  *
106  * Salidas: ADC1BUF0>>2: Que es la variable que proporciona el
       conversor A/D tras la lectura pero con 8 bits en lugar de 10.
107  *
108  * Otros efectos: Ninguno.
109  *
110  * Funcionamiento: Esta función realiza una lectura del valor del
       conversor A/D teniendo como parámetros de entrada el canal que se
       quiere leer y devuelve la variable en la que se va a guardar.
111  *
112  * Nota: La diferencia de esta función con la función unsigned char
       lee_analog_manual(unsigned char canal) es que la actual no emplea
       el timer 1 para controlar el tiempo de muestreo.
113  ************************************************************/
114 /*
115 unsigned char lee_analog_automatic_sampling_timing(unsigned char canal)
116 {
117 AD1CHSbits.CH0SA=canal;       //Selecciona el canal de entrada
118 AD1CON1bits.SAMP=1;           //Comienza el muestreo
119 while (!AD1CON1bits.DONE);    //Espera a que la conversión esté
       completada.
120 return(ADC1BUF0>>2);          //Cuando ha acabado la conversión devuelve
       el valor del registro.
121                               //Es como un shift right El >>2 hace que
                                     el valor del A/D pase a ser una
                                     variable de 8 bits.
122 }
123 */
124
125
126 /************************************************************
127  * Función: void
       lee_9_canales_analogicos_manualmente_dejandolo_en_buffer(void)
128  *  Esta funcion realiza la lectura analógica de 9 canales.
129  *
130  * Pre condiciones: El conversor A/D debe haber sido inicializado con
       la función void initADC(int analog_mask) y los 9 canales que  se
       pretende leer AN0....AN8 debe haber sido configurado como entradas
       analógicas.
131  *
132  * Entradas: Ninguna.
133  *  La función implícitamente ya tiene definido que se van a leer los
       canales AN0...AN8
134  *
135  * Salidas: Ninguna.
```

```
136  *  Los valores leidos se guardan en una variable global static char
         buffer_conversiones[9] dentro del módulo para acceder después a
         esta información se empleará una función interfaz.
137  *
138  * Otros efectos: Utiliza e inicializa el timer 1
139  *
140  * Funcionamiento: Esta función realiza 9 lecturas del valor del
         conversor A/D sin tener que realizar 9 llamadas a la función, lo
         que evita el proceso de guardar valores en la pila, stack etc etc.
141  *
142  * Nota: Ninguna
143  ************************************************************/
144 /*
145 void lee_9_ch_analog_man_deja_buffer(void)
146 {
147 unsigned char canal;
148 T1CON=0x8000;                   //Configura el tiempo de muestreo, lo
       único que hace es encender el timer con un preescalado de 1:1, y la
       frecuencia la del bus de periféricos que en un principio es de
       74Mhz/2.
149 for (canal=0;canal<9;canal++)
150     {
151     AD1CHSbits.CH0SA=canal;     //Selecciona el canal del que se va a
           hacer la conversión.
152     AD1CON1bits.SAMP=1;         //Comienza el muestreo
153     TMR1=0;                     //Pone a 0 la cuenta del timer
154     while (TMR1<100);           //En el momento que la cuenta del
           timer sea de 100 saldrá del bucle y habrá pasado el tiempo de
           muestreo.
155     AD1CON1bits.SAMP=0;         //Comienza la conversión
156     while (!AD1CON1bits.DONE);  //espera a que haya terminado la
           conversión.
157     buffer_conversiones[canal]=(ADC1BUF0>>2);  //Cuando ha acabado la
           conversión el valor se pasa manualmente al buffer de
           conversiones que nos hemos creado.
158                                 //Es como un shift right El >>2 hace
                                       que el valor del A/D pase a ser una
                                       variable de 8 bits.
159     }
160 //Una vez leidos los 9 canales esta funcion termina y tenemos que
       recuperar los datos del buffer. Esto ya lo hace otra función, la
       funcion interfaz.
161 }
162 */
163
164
165 /************************************************************
166  * Función: void lee_9_ch_analog_automatico_deja_buffer(void)
167  *  Esta funcion realiza la lectura analógica de 9 canales.
168  *
169  * Pre condiciones: El conversor A/D debe haber sido inicializado con
         la función void initADC_automatic_sampling_timing (int
         analog_mask) y los 9 canales que   se pretende leer AN0....AN8
         debe haber sido configurado como entradas analógicas.
```

```
170  *
171  * Entradas: Ninguna.
172  *  La función implícitamente ya tiene definido que se van a leer los
         canales AN0...AN8
173  *
174  * Salidas: Ninguna.
175  *  Los valores leidos se guardan en una variable global static char
         buffer_conversiones[9] dentro del módulo para acceder después a
         esta información se empleará una función interfaz.
176  *
177  * Otros efectos: Ninguno.
178  *
179  * Funcionamiento: Esta función realiza 9 lecturas del valor del
         conversor A/D sin tener que realizar 9 llamadas a la función, lo
         que evita el proceso de guardar valores en la pila, stack etc etc.
180  *
181  * Nota: La diferencia entre la función actual y la función void
         lee_9_canales_analogicos_manualmente_dejandolo_en_buffer(void) es
         que la actual no utiliza el timer 1 para definir el tiempo de
         muestreo
182 ************************************************************/
183 /*
184 void lee_9_ch_analog_automatico_deja_buffer(void)
185 {
186 unsigned char canal;
187 for(canal=0;canal<9;canal++)
188     {
189     AD1CHSbits.CH0SA=canal;       //Selecciona el canal de entrada
190     AD1CON1bits.SAMP=1;           //Comienza el muestreo
191     while (!AD1CON1bits.DONE);    //Espera a que la conversión esté
            completada.
192     buffer_conversiones[canal]=(ADC1BUF0>>2);  //Cuando ha acabado la
            conversión devuelve el valor del registro.
193                                   //Es como un shift right El >>2 hace
                                        que el valor del A/D pase a ser una
                                        variable de 8 bits.
194     }
195 //Una vez leidos los 9 canales esta funcion termina y tenemos que
      recuperar los datos del buffer. Esto ya lo hace otra función, la
      funcion interfaz.
196 }
197 */
198
199
200 /************************************************************
201  * Función: void lee_y_devuelve_9_ch_analog_manual(unsigned char*
         datos_analogicos)
202  *  Esta funcion realiza la lectura analógica de 9 canales.
203  *
204  * Pre condiciones: El conversor A/D debe haber sido inicializado con
         la función void initADC(int analog_mask) y los 9 canales que se
         pretende leer AN0....AN8 debe haber sido configurado como entradas
         analógicas.
205  *
```

```
206  * Entradas: unsigned char* datos_analogicos: Puntero a la zona de
         memoria donde se guardan los datos de la lectura.
207  *
208  * Salidas: Se pasa la variable datos_analogicos por referencia y esa
         es la zona de memoria donde se guardan las lecturas.
209  *
210  * Otros efectos: Ninguno.
211  *
212  * Funcionamiento: Esta función realiza 9 lecturas del valor del
         conversor A/D sin tener que realizar 9 llamadas a la función, lo
         que evita el proceso de guardar valores en la pila, stack etc etc.
         Además tampoco necesita una función interfaz para devolver los
         datos ya que utiliza un puntero.
213  *
214  * Nota: Ninguna.
215  ************************************************************/
216 /*
217 void lee_y_devuelve_9_ch_analog_manual(unsigned char* datos_analogicos)
218 {
219 unsigned char canal;
220 T1CON=0x8000;                    //Configura el tiempo de muestreo, lo
       único que hace es encender el timer con un preescalado de 1:1, y la
       frecuencia la del bus de periféricos que en un principio es de
       74Mhz/2.
221 for (canal=0;canal<9;canal++)
222     {
223     AD1CHSbits.CH0SA=canal;      //Selecciona el canal del que se va a
           hacer la conversión.
224     AD1CON1bits.SAMP=1;          //Comienza el muestreo
225     TMR1=0;                      //Pone a 0 la cuenta del timer
226     while (TMR1<100);            //En el momento que la cuenta del
           timer sea de 100 saldrá del bucle y habrá pasado el tiempo de
           muestreo.
227     AD1CON1bits.SAMP=0;          //Comienza la conversión
228     while (!AD1CON1bits.DONE);   //espera a que haya terminado la
           conversión.
229     datos_analogicos[canal]=(ADC1BUF0>>2);  //Cuando ha acabado la
           conversión el valor se a la zona de memoria apuntada pol el
           puntero.
230                                  //Es como un shift right El >>2 hace
                                         que el valor del A/D pase a ser una
                                         variable de 8 bits.
231     }
232 }
233 */
234
235
236 /************************************************************
237  * Función: void lee_y_devuelve_9_ch_analog_automatico(unsigned char*
         datos_analogicos)
238  *  Esta funcion realiza la lectura analógica de 9 canales.
239  *
240  * Pre condiciones: El conversor A/D debe haber sido inicializado con
         la función void initADC_automatic_sampling_timing (int
```

```
        analog_mask) y los 9 canales que  se pretende leer AN0....AN8
        debe haber sido configurado como entradas analógicas.
241   *
242   * Entradas: unsigned char* datos_analogicos: Puntero a la zona de
        memoria donde se guardan los datos de la lectura.
243   *
244   * Salidas: Se pasa la variable datos_analogicos por referencia y esa
        es la zona de memoria donde se guardan las lecturas.
245   *
246   * Otros efectos: Ninguno.
247   *
248   * Funcionamiento: Esta función realiza 9 lecturas del valor del
        conversor A/D sin tener que realizar 9 llamadas a la función, lo
        que evita el proceso de guardar valores en la pila, stack etc etc.
        Además tampoco necesita una función interfaz para devolver los
        datos ya que utiliza un puntero.
249   *
250   * Nota: La diferencia entre la función actual y la función void
        lee_y_devuelve_9_ch_analog_manual(unsigned char* datos_analogicos)
        es que la actual no utiliza el timer 1 para definir el tiempo de
        muestreo.
251   *************************************************************/
252 void lee_y_devuelve_9_ch_analog_automatico(unsigned char*
      datos_analogicos)
253 {
254 unsigned char canal;
255 for(canal=0;canal<9;canal++)
256     {
257     AD1CHSbits.CH0SA=canal;       //Selecciona el canal de entrada
258     AD1CON1bits.SAMP=1;           //Comienza el muestreo
259     while (!AD1CON1bits.DONE);    //Espera a que la conversión esté
          completada.
260     datos_analogicos[canal]=(ADC1BUF0>>2);  //Cuando ha acabado la
          conversión devuelve el valor del registro.
261                                   //Es como un shift right El >>2 hace
                                        que el valor del A/D pase a ser una
                                        variable de 8 bits.
262     }
263 }
264
265
266 /*************************************************************
267   * Función: void
        lee_y_devuelve_3_ch_analog_automatico_con_10_bits(unsigned short*
        datos_analogicos)
268   *  Esta funcion realiza la lectura analógica de 3 canales con toda la
        resolución que permite el conversor A/D.
269   *
270   * Pre condiciones: El conversor A/D debe haber sido inicializado con
        la función void initADC_automatic_sampling_timing (int
        analog_mask) y los 3 canales que  se pretende leer AN0, AN1 y
        AN3!!! deben haber sido configurado como entradas analógicas.
271   *
```

```
272  * Entradas: unsigned short* datos_analogicos: Puntero a la zona de
         memoria donde se guardan los datos de la lectura.
273  *
274  * Salidas: Se pasa la variable datos_analogicos por referencia y esa
         es la zona de memoria donde se guardan las lecturas.
275  *
276  * Otros efectos: A estas alturas el canal AN2 se ha roto, ha sido
         causa de un problema de masas al conectar un convertido DC-DC d 1W.
277  *
278  * Funcionamiento: Esta función realiza 3 lecturas del valor del
         conversor A/D sin tener que realizar 3 llamadas a la función, lo
         que evita el proceso de guardar valores en la pila, stack etc etc.
         Además tampoco necesita una función interfaz para devolver los
         datos ya que utiliza un puntero.
279  *
280  * Nota: Se ha creado esta función porque los datos a leer se van a
         multiplexar en grupos de 3.
281  ************************************************************/
282 void lee_y_devuelve_3_ch_analog_automatico_con_10_bits(unsigned short*
        datos_analogicos)
283 {
284 unsigned char canal;
285 for(canal=0;canal<3;canal++)
286     {
287     if (canal==2)
288         {AD1CHSbits.CH0SA=3;}        //En el caso del canal 2 como está
                roto se emplea el canal 3.
289     else
290         {AD1CHSbits.CH0SA=canal;}    //Selecciona el canal de entrada
291     AD1CON1bits.SAMP=1;              //Comienza el muestreo
292     while (!AD1CON1bits.DONE);       //Espera a que la conversión esté
            completada.
293     datos_analogicos[canal]=(ADC1BUF0);  //Cuando ha acabado la
            conversión devuelve el valor del registro.
294                                      //Esta vez se devuelven todos los
                                             bits que proporciona el
                                             conversor A/D.
295     }
296 }
297
298
299 /*************************************************************
300  * Función: void
         lee_y_devuelve_3_ch_analog_automatico_con_10_bits_realizando_media_
         para_eliminar_rizado(unsigned short* datos_analogicos)
301  *  Esta funcion realiza la lectura analógica de 3 canales con toda la
         resolución que permite el conversor A/D.
302  *
303  * Pre condiciones: El conversor A/D debe haber sido inicializado con
         la función void initADC_automatic_sampling_timing (int
         analog_mask) y los 3 canales que  se pretende leer AN0, AN1 y
         AN3!!! deben haber sido configurado como entradas analógicas.
304  *
```

```
305  * Entradas: unsigned short* datos_analogicos: Puntero a la zona de
         memoria donde se guardan los datos de la lectura.
306  *
307  * Salidas: Se pasa la variable datos_analogicos por referencia y esa
         es la zona de memoria donde se guardan las lecturas.
308  *
309  * Otros efectos: A estas alturas el canal AN2 se ha roto, ha sido
         causa de un problema de masas al conectar un convertido DC-DC d 1W.
310  *
311  * Funcionamiento: Esta función realiza 3 lecturas del valor del
         conversor A/D sin tener que realizar 3 llamadas a la función, lo
         que evita el proceso de guardar valores en la pila, stack etc etc.
         Además tampoco necesita una función interfaz para devolver los
         datos ya que utiliza un puntero.
312  *
313  * Nota: Además repite estas tres lecturas durente "Num_iteraciones"
         veces y luego calcula la media, esto es lo mismo que un filtro
         paso bajo.
314  ************************************************************/
315 void
        lee_y_devuelve_3_ch_analog_automatico_con_10_bits_realizando_media_
        para_eliminar_rizado(unsigned short* datos_analogicos)
316 {
317 unsigned char canal;
318 unsigned char iter,Num_iteraciones;
319 unsigned short datos_sin_media[]={0,0,0};
320 Num_iteraciones=1;//Es el numero de veces que leemos para hacer la
        media.
321 for(iter=0;iter<Num_iteraciones;iter++)
322     {
323     for(canal=0;canal<3;canal++)
324         {
325         if (canal==2)
326             {AD1CHSbits.CH0SA=3;}       //En el caso del canal 2 como
                    está roto se emplea el canal 3.
327         else
328             {AD1CHSbits.CH0SA=canal;}   //Selecciona el canal de entrada
329
330         AD1CON1bits.SAMP=1;             //Comienza el muestreo
331         while (!AD1CON1bits.DONE);      //Espera a que la conversión
                esté completada.
332         datos_sin_media[canal]=datos_sin_media[canal]+(ADC1BUF0);
                //Cuando ha acabado la conversión devuelve el valor del
                registro.
333                                         //Esta vez se devuelven todos
                                              los bits que proporciona el
                                              conversor A/D.
334         }
335     }
336 for(canal=0;canal<3;canal++)
337     {
338     datos_analogicos[canal]=datos_sin_media[canal]/Num_iteraciones;
            //Esto es básicamente hacer la media.
339     }
```

```
340 }
341
342
343
344
345
346   //-------------------------------------------------------//
347  //FUNCIONES INTERFAZ                                      //
348 //-------------------------------------------------------//
349
350
351 /*************************************************************
352  * Función: void lee_buffer_conversiones_analogico(unsigned char*
         buffer)
353  *
354  * Pre condiciones: Alguna de las siguientes funciones ha de haber
         sido llamada anteriormente:
355  *  void lee_9_ch_analog_man_deja_buffer(void)
356  *  void lee_9_ch_analog_automatico_deja_buffer(void)
357  *
358  * Entradas: unsigned char* buffer: Puntero a la zona de memoria en
         otro módulo en el que se quieren copiar los datos.
359  *
360  * Salidas: Se pasa la variable buffer por referencia y esa es la zona
         de memoria donde se copian los datos.
361  *
362  * Otros efectos: Ninguno
363  *
364  * Funcionamiento: Esta función hace accesible los datos analógicos
         para las funciones que realizan la lectura y no emplean punteros.
365  *
366  * Nota: Además repite estas tres lecturas durente "Num_iteraciones"
         veces y luego calcula la media, esto es lo mismo que un filtro
         paso bajo.
367  *************************************************************/
368 /*
369 void lee_buffer_conversiones_analogico(unsigned char* buffer)    //Esta
       funcion lee el buffer creado por software.
370 {
371 unsigned char indice;
372 for(indice=0;indice<9;indice++)
373      {
374      buffer[indice]=buffer_conversiones[indice];
375      }
376 }
377 */
```

Contenido del archivo “analog.h” que contiene los prototipos de funciones.

```
1 //************************************
2 //analog.h                           *
3 //************************************
4 //Variables globales
5 //- - - - - - - - - - - - - - - - - -
6 //- - - - - - - - - - - - - - - - - - -
```

```
7 //Prototipos de funciones
8 void initADC_manual(int analog_mask);
9 void initADC_automatic_sampling_timing (int analog_mask);
10 unsigned char lee_analog_manual(unsigned char canal);
11 unsigned char lee_analog_automatic_sampling_timing(unsigned char
      canal);
12 void lee_9_ch_analog_man_deja_buffer(void); //lee 9 canales analogicos
      manualmente dejandolo en un buffer
13 void lee_9_ch_analog_automatico_deja_buffer(void);////lee 9 canales
      analogicos con muestreo automatico dejandolo en un buffer
14 void lee_y_devuelve_9_ch_analog_manual(unsigned char*
      datos_analogicos); //Lee los 9 canales anaógicos manualmente y los
      devuelve a la zona de memoria apuntada por el puntero.
15 void lee_y_devuelve_9_ch_analog_automatico(unsigned char*
      datos_analogicos); //Lee los 9 canales anaógicos con tiempo de
      muestreo automático y los devuelve a la zona de memoria apuntada
      por el puntero.
16 void lee_y_devuelve_3_ch_analog_automatico_con_10_bits(unsigned short*
      datos_analogicos); //Lee los 3 canales analógicos con tiempo de
      muestreo automático y los devuelve a la zona de memoria apuntada
      por el puntero. Esta es la unica función que trabaja con todos los
      bits del conversor AD (CON LOS 10 bits)
17 void
      lee_y_devuelve_3_ch_analog_automatico_con_10_bits_realizando_media
      _para_eliminar_rizado(unsigned short* datos_analogicos);
18
19 //- - - - - - - - - - - - - - - - - - - - -
20 //Funciones interfaz
21 void lee_buffer_conversiones_analogico(unsigned char* buffer); //Esta
      funcion lee el buffer creado por software.
```

Variables y definiciones contenidas en el archivo “analog_variables_y_definiciones.h”

```
1 //************************************
2 //analog_variables_y_definiciones.h    *
3 //************************************
4 #include <p32mx460f512l.h>
5 #include "analog.h"
6
7 //Definimos variables globales
8 //static char buffer_conversiones[9];
9 //
```

Funciones contenidas en el archivo “calculo.c”

```
1 //************************************
2 //calculo.c                           *
3 //************************************
4 #include "calculo_variables_y_definiciones.h"
5
6 /*********************************************************
7  * Función: unsigned char buscar_en_matriz(unsigned char valor_leido,
      unsigned char distancia )
8  *  Existe una variable global que es un array bidimensional, esta
      función busca el valor "valor_leido" en la fila "distancia"
9  *
```

```
10  * Pre condiciones: Ninguna
11  *
12  * Entradas: Valor leido: Es el valor que se pretende buscar en la
        tabla/Matriz/Vector.
13  *           Distancia: Indica en que fila de la matriz se va a
        realizar  la búsqueda. Fisicamente también indica la distancia en
        la dirección perpendicular a la base del imán.
14  *
15  * Salidas: indice: Es el número de elemento en el que se ha
        encontrado el valor (el más próximo al valor pues recordemos los
        valores  son continuos. Si el valor no se encuentra dentro del
        rango  del vector, entonces diremos que el valor no ha sido
        encontrado. Esto se indica dándole a la variable indice el valor
        255.
16  *
17  * Otros efectos: Ninguno
18  *
19  * Funcionamiento: Lo que esta función hace es buscar un valor dentro
        de una matriz, en realidad, hablando con propiedad únicamente lo
        busca dentro de un vector perteneciente a la matriz, el vector es
        el numero de fila de la matriz indicado por la variable distancia.
20  *
21  * Nota: Emplea búsqueda binaria
22  ************************************************************/
23 /*
24 unsigned char buscar_en_matriz(unsigned char valor_leido, unsigned
       char distancia )
25 {
26 unsigned char encontrado=0;
27 unsigned char maximo=(MAX_columnas-1);  //Por el momento son 31
       columnas, de 0 a 30.
28 unsigned char minimo=0;
29 unsigned char indice;
30 if (valor_leido>Espacio_de_busqueda[distancia][minimo]||valor_leido<
       Espacio_de_busqueda[distancia][maximo]) //El valor no está
       contenido en el array y devolveremos indice=255 que indicará que no
       se ha encontrado, en caso contrario se enviará indice igual a la
       posición en la que se encuentre el valor buscado.
31 {
32 indice=255;
33 }
34 else
35 {
36      while(!encontrado)
37          {
38          indice=(maximo+minimo)/2;
39          if (Espacio_de_busqueda[distancia][indice]>valor_leido)
40              {
41              minimo=indice;
42              }
43          else
44              {
45              maximo=indice;
46              }
```

```
47          if (minimo+1>=maximo)
48              {
49              encontrado=1;
50              }
51          }
52      if((Espacio_de_busqueda[distancia][minimo]-valor_leido)<(valor_leido-
            Espacio_de_busqueda[distancia][maximo]))
53          indice=minimo;
54      else
55          indice=maximo;
56  }
57  return(indice);
58  }
59  */
60
61
62  /*********************************************************
63   * Función: void buscar_en_matriz_3_datos_sin_punteros(unsigned char*
         valor_leido_array, unsigned char distancia, unsigned char*
         indice_array )
64   *  Existe una variable global que es un array bidimensional, esta
         función busca el valor "valor_leido_array" en la fila "distancia"
65   * Pre condiciones: Ninguna
66   *
67   * Entradas: valor_leido_array: puntro al inicio de un array que
         contiene los tres valores que se pretenden buscar en la
         tabla/Matriz/Vector.
68   *           distancia: Indica en que fila de la matriz se va a
         realizar la búsqueda. Fisicamente también indica la distancia en
         la dirección perpendicular a la base del imán.
69   *
70   * Salidas: indice_array: Puntero a un vector donde se guardarán las
         posiciones encontradas en la matriz.
71   *
72   * Otros efectos: Ninguno
73   *
74   * Funcionamiento: El funcionamiento es parecido al de la función
         anterior buscar_en_matriz(unsigned char valor_leido, unsigned char
         distancia ), unicamente cambian algunos parámetros.
75   *
76   * Nota: Emplea busqueda binaria
77   *********************************************************/
78  /*
79  void buscar_en_matriz_3_datos_sin_punteros(unsigned char*
        valor_leido_array, unsigned char distancia, unsigned char*
        indice_array )
80  {
81  unsigned char sensor;           //esta variable se utiliza para
        recorrer los tres sensores
82  unsigned char encontrado;
83  unsigned char maximo;           //Por el momento son 31 columnas, de 0
        a 30.
84  unsigned char minimo;
85  unsigned char valor_leido,indice;
```

```
86 for (sensor=0;sensor<3;sensor++)
87     {
88     maximo=(MAX_columnas-1);      //Por el momento son 31 columnas, de 0
           a 30.
89     minimo=0;
90 encontrado=0;
91     valor_leido=valor_leido_array[sensor];
92     if
           (valor_leido>Espacio_de_busqueda[distancia][minimo]||valor_leido<
           Espacio_de_busqueda[distancia][maximo]) //El valor no está
           contenido en el array y devolveremos indice=255 que indicará
           que no se ha encontrado, en caso contrario se enviará indice
           igual a la posición en la que se encuentre el valor buscado.
93         {
94         indice=255;
95         }
96     else
97         {
98         while(!encontrado)
99             {
100            indice=(maximo+minimo)/2;
101            if (Espacio_de_busqueda[distancia][indice]>valor_leido)
102                {
103                minimo=indice;
104                }
105            else
106                {
107                maximo=indice;
108                }
109            if (minimo+1>=maximo)
110                {
111                encontrado=1;
112                }
113            }
114        if((Espacio_de_busqueda[distancia][minimo]-valor_leido)<(valor_leido-
               Espacio_de_busqueda[distancia][maximo]))
115            indice=minimo;
116        else
117            indice=maximo;
118        }
119    indice_array[sensor]=indice;
120    }
121
122 }
123 */
124
125
126 /*********************************************************
127  * Función: void buscar_en_matriz_3_datos_con_punteros(unsigned char*
       valor_leido, unsigned char distancia, unsigned char* indice)
128  *  Existe una variable global que es un array bidimensional, esta
       función busca el valor "valor_leido" en la fila "distancia"
129  *
130  * Pre condiciones: Ninguna
```

```
131  *
132  * Entradas: valor_leido: puntro al inicio de un array que contiene
         los tres valores que se pretenden buscar en la tabla/Matriz/Vector.
133  *           distancia: Indica en que fila de la matriz se va a
         realizar la búsqueda. Fisicamente también indica la distancia en
         la dirección perpendicular a la base del imán.
134  *
135  * Salidas: indice: Puntero a un vector donde se guardarán las
         posiciones encontradas en la matriz.
136  *
137  * Otros efectos: Ninguno
138  *
139  * Funcionamiento: El funcionamiento es parecido al de la función
         anterior buscar_en_matriz(unsigned char valor_leido, unsigned char
         distancia ), unicamente cambia que en este caso se trabaja
         directamente con los punteros.
140  *
141  * Nota: Emplea busqueda binaria
142  ************************************************************/
143 /*
144 void buscar_en_matriz_3_datos_con_punteros(unsigned char* valor_leido,
        unsigned char distancia, unsigned char* indice)
145 {
146 unsigned char sensor;
147 unsigned char encontrado;
148 unsigned char maximo;
149 unsigned char minimo;
150 for(sensor=0;sensor<3;sensor++)
151     {
152     maximo=(MAX_columnas-1);  //Por el momento son 31 columnas, de 0 a
          30.
153     minimo=0;
154     encontrado=0;
155
156     if (valor_leido[sensor]>Espacio_de_busqueda[distancia][minimo]||
          valor_leido[sensor]<Espacio_de_busqueda[distancia][maximo])
          //El valor no está contenido en el array y devolveremos
          indice=255 que indicará que no se ha encontrado, en caso
          contrario se enviará indice igual a la posición en la que se
          encuentre el valor buscado.
157     {
158     indice[sensor]=255;
159     }
160     else
161         {
162         while(!encontrado)
163             {
164             indice[sensor]=(maximo+minimo)/2;
165             if (Espacio_de_busqueda[distancia][indice[sensor]]>
                    valor_leido[sensor])
166                 {
167                 minimo=indice[sensor];
168                 }
169             else
```

```
170                     {
171                     maximo=indice[sensor];
172                     }
173                 if (minimo+1>=maximo)
174                     {
175                     encontrado=1;
176                     }
177                 }
178             if((Espacio_de_busqueda[distancia][minimo]-valor_leido[sensor])<
                    (valor_leido[sensor]-Espacio_de_busqueda[distancia][maximo]))
179                 indice[sensor]=minimo;
180             else
181                 indice[sensor]=maximo;
182             }
183         }
184 }
185 */
186
187
188 /*********************************************************
189  * Función: void
        buscar_en_matriz_3_datos_con_punteros_optimizado(unsigned char*
        valor_leido, unsigned char distancia, unsigned char* indice)
190  *
191  *  Opimización de la función
        anterior(buscar_en_matriz_3_datos_con_punteros)
192  *
193  *********************************************************/
194 /*
195 void buscar_en_matriz_3_datos_con_punteros_optimizado(unsigned char*
        valor_leido, unsigned char distancia, unsigned char* indice)
196 {
197 unsigned char sensor;
198 unsigned char encontrado;
199 unsigned char maximo;
200 unsigned char minimo;
201 for(sensor=0;sensor<3;sensor++)
202     {
203     if (sensor>0)
204     {
205     valor_leido++;
206     indice++;
207     }
208     maximo=(MAX_columnas-1); //Por el momento son 31 columnas, de 0 a
            30.
209     minimo=0;
210     encontrado=0;
211
212     if
            (*valor_leido>Espacio_de_busqueda[distancia][minimo]||*valor_leido<
            Espacio_de_busqueda[distancia][maximo]) //El valor no está
            contenido en el array y devolveremos indice=255 que indicará
            que no se ha encontrado, en caso contrario se enviará indice
            igual a la posición en la que se encuentre el valor buscado.
```

```
213     {
214     *indice=255;
215     }
216     else
217         {
218         while(!encontrado)
219             {
220             *indice=(maximo+minimo)/2;
221             if
                    (Espacio_de_busqueda[distancia][*indice]>valor_leido[sensor])
222                 {
223                 minimo=*indice;
224                 }
225             else
226                 {
227                 maximo=*indice;
228                 }
229             if (minimo+1>=maximo)
230                 {
231                 encontrado=1;
232                 }
233             }
234         if((Espacio_de_busqueda[distancia][minimo]-*valor_leido)<
                (*valor_leido-Espacio_de_busqueda[distancia][maximo]))
235             *indice=minimo;
236         else
237             *indice=maximo;
238         }
239     }
240 }
241 */
242
243
244 /*************************************************************
245  * Función: void
        buscar_en_matriz_3_datos_Totalmente_desoptimizado(unsigned char*
        valor_leido_array, unsigned char distancia, unsigned char*
        indice_array )
246  *
247  *  Esta es una función de búsqueda sin ninguna optimización que se ha
        creado para comparar los tiempos de búsqueda entre diferentes
        funciones.
248  *
249  *************************************************************/
250 /*
251 void buscar_en_matriz_3_datos_Totalmente_desoptimizado(unsigned char*
        valor_leido_array, unsigned char distancia, unsigned char*
        indice_array )
252 {
253 unsigned char sensor;                   //esta variable se utiliza
        para recorrer los tres sensores
254 unsigned char encontrado;
255 unsigned char maximo=(MAX_columnas-1);
256 unsigned char valor_leido,indice;
```

```
257 for (sensor=0;sensor<3;sensor++)
258     {
259     encontrado=0;                           //Inicializamos la variable
            que permite salir del bucle de búsqueda.
260     indice=0;
261     valor_leido=valor_leido_array[sensor];
262
263     if (valor_leido>Espacio_de_busqueda[distancia][0]||valor_leido<
            Espacio_de_busqueda[distancia][maximo]) //El valor no está
            contenido en el array y devolveremos indice=255 que indicará
            que no se ha encontrado, en caso contrario se enviará indice
            igual a la posición en la que se encuentre el valor buscado.
264         {
265         indice=255;                         //Esto indica que el valor no
                se encuentra dentro del rango del array
266         }
267     else
268         {
269         while(!encontrado)
270             {
271             indice++;
272             if (valor_leido>=Espacio_de_busqueda[distancia][indice])
273                 {
274                 encontrado=1;               //Significa que lo hemos
                        encontrado, ahora queda saber si está mas cerca de
                        un extremo o de otro.
275                 }
276             }
277         if((Espacio_de_busqueda[distancia][indice]-valor_leido)<
                (valor_leido-Espacio_de_busqueda[distancia][(indice-1)]))
278         {   //Indice se queda como está, se podría haber omitido.
279         }
280         else
281             {
282             indice--;                       //El índice es el valor
                    anterior.
283             }
284         }
285     indice_array[sensor]=indice;
286     }
287 }
288 */
289
290
291 /*************************************************************
292  * Función:              void
        buscar_en_matriz_3_datos_sin_punteros_variables_short(unsigned
        short* valor_leido_array, unsigned char distancia, unsigned char*
        indice_array )
293  *  Al realizar las lecturas de los valores anaógicos con toda la
        resolución del conversor A/D se necesitan funciones que trabajen
        con variables unsigned short.
294  * Pre condiciones: Ninguna
295  *
```

```
296  * Entradas: distancia: Es el número de fila dentro de la matriz por
         la que se realiza la búsqueda.
297  *           Espacio_de_busqueda: Es uma matriz, es variable golobal
         del módulo, así que no hay que pasársela.
298  *           valor_leido_array: (x3) Puntero al valor que lee el
         conversor A/D y que hay que buscar en la matriz.
299  *
300  * Salidas: indice_array: Puntero a un vector donde se guardarán las
         posiciones encontradas en la matriz.
301  *
302  * Otros efectos: Ninguno
303  *
304  * Funcionamiento: Esta funcion busca en una fila, es llamada tantas
         veces como filas tiene la matriz.
305  *  Esta función se ha creado a partir de
         buscar_en_matriz_3_datos_sin_punteros pero con la necesidad de que
         la matriz de búsqueda estuviera compuesta de variables short (De
         16 bits). Las variables distancia e indice_array es suficiente con
         que sean de 8 bits porque distancia hace referencia al número de
         filas y indice_array al número de columnas y ninguna de ellas
         supera los 255 (para ser másа precisos hay 101 y 101).
306  *
307  * Nota: COMPROBACIÓN DE ERROR EN LA SALIDA:
308  *  Si indice_array[0]=255 entonces habrá ocurrido que alguno de los
         tres valores no ha podido ser encontrado en esa fila de la matriz.
309  *  Emplea búsqueda binaria
310  ************************************************************/
311
312 void buscar_en_matriz_3_datos_sin_punteros_variables_short(unsigned
       short* valor_leido_array, unsigned char distancia, unsigned char*
       indice_array )
313 {
314 unsigned char sensor;                  //esta variable se utiliza para
       recorrer los tres sensores
315 unsigned char encontrado;
316 unsigned char NO_EXISTE=0;
317 unsigned char maximo;                  //Actualmente son 101 columnas, de
       0 a 100
318 unsigned char minimo;
319 unsigned short valor_leido;
320 unsigned char indice;
321 for (sensor=0;sensor<3;sensor++)
322     {
323     if(!NO_EXISTE)
324         {
325         //maximo=(MAX_columnas-1);  //Por el momento son 101 columnas
              de 0 a 100.
326         //minimo=0;
327         maximo=100;
328         minimo=0;
329         encontrado=0;
330         valor_leido=valor_leido_array[sensor];
331         if (valor_leido>Espacio_de_busqueda[distancia][minimo]||
              valor_leido<Espacio_de_busqueda[distancia][maximo]) //El
```

```
            valor no está contenido en el array y devolveremos
            indice=255 que indicará que no se ha encontrado, en caso
            contrario se enviará indice igual a la posición en la que
            se encuentre el valor buscado.
332             {
333             indice_array[0]=255;
334             NO_EXISTE=1;
335             }
336         else
337             {
338             while(!encontrado)
339                 {
340                 indice=(maximo+minimo)/2;
341                 if (Espacio_de_busqueda[distancia][indice]>valor_leido)
342                     {
343                     minimo=indice;
344                     }
345                 else
346                     {
347                     maximo=indice;
348                     }
349                 if (minimo+1>=maximo)
350                     {
351                     encontrado=1;
352                     }
353                 }
354             if((Espacio_de_busqueda[distancia][minimo]-valor_leido)<
                    (valor_leido-Espacio_de_busqueda[distancia][maximo]))
355     //Esta condición lo que hace es interpolando mediante una linea
        recta aproximar a cual de los dos putos está más cerca (al
        izquierdo o al derecho).
356                 indice=minimo;
357             else
358                 indice=maximo;
359
360             indice_array[sensor]=indice;    //En el caso de que se
                    encuentre se guarda la posición
361             }
362         }
363     }
364 }
365
366
367 /*********************************************************
368  * Función: float Buscar_posiciones_y_errores(unsigned char* radios,
        unsigned char* posicion)
369  *
370  * Pre condiciones: Anteriormente es necesario llamar a alguna de las
        funciones de búsqueda que es la que nos devuelve los radios.
371  *
372  * Entradas: radios: Puntero con la distancia del imán a cada uno de
        los sensores, circunferencias que se van a intersecar.
373  *
```

```
374  * Salidas: posicion: Puntero a un vector donde se guardarán las
         coordenadas xyz calculadas.
375  *          Return de la funcion(error): Error cometido en la medida.
376  *
377  * Otros efectos: Ninguno
378  *
379  * Funcionamiento: Esta función conoce la posición de los tres
         sensores respecto de un sistema de referencia fijo con centro en
         el sensor 1, la posición de los sensores es [SENSOR_1 (0,0)]
         [SENSOR_2 (4.54,0)] [SENSOR3 (2.27,4.54)]. Como parametros de
         entrada se le pasan tres distancias o radios y como parámetros de
         salidaa devuelve el error de la medida.
380  *
381  * Nota: COMPROBACIÓN DE ERROR EN LA SALIDA:
382  *  Si la salida es 200, entonces significa que no cortan, eso se hace
         así porque lo he elegido yo (así no tengo que emplear otra
         variable)
383  ************************************************************/
384 float Buscar_posiciones_y_errores(unsigned char* radios, unsigned
       char* posicion)
385 {
386 float error;
387 float distancia_x,distancia_y;
388 float test_float;
389 int test_int=30;
390 test_float=X_02;
391 test_int=test_int-test_float;
392 if((radios[1]-radios[0]>X_02)||(radios[0]-radios[1]>
       X_02)||(radios[0]+radios[1]<X_02))
393     //Esto significa que las circunferencias no se cortan.
394     //Entonces lo que tenemos que indicar es que el error es 200 y así
           quedará registrado que no se cortan.
395     error=200;
396 else
397     {
398     //En el caso de llegar a esta parte del código es porque las dos
           circunferencias se cortan
399     distancia_x=(pow(radios[0],2)-pow(radios[1],2)+pow(X_02,2))/(2*X_02);
400     //distancia_x=(radios[1]^2-radios[2]^2+X_02^2)/(2*X_02);
401     distancia_y=sqrt(pow(radios[0],2)-pow(distancia_x,2));
402     //distancia_y=sqrt(radios[1]^2-distancia_x^2);
403     /*Se ha calculado el punto en el que se encuentra el imán, ahora
           entra en juego el tercer sensor para determinar el error
           cometido.*/
404     error=sqrt(pow(X_03-distancia_x,2)+pow(Y_03-distancia_y,2))-radios[2];
405     /*Un error de -9 no implica que sea menor error que un error de 3.
           Con esto quiero decir que lo que nos interesa del error es el
           valor absoluto, no he encontrado en la librería math.h ninguna
           función que realice el valor absoluto así que habrá que
           programarla.*/
406     if(error<0)
407         {
408         error=-error;    //Cambia de signo el error.
409         }
```

```
410     posicion[0]=distancia_x;     //Devolviendo la posición.
411     posicion[1]=distancia_y;     //Devolviendo la posición.
412     }
413 return(error);  //Devolviendo el error.
414 }
415
416
417 /*********************************************************
418  * Función: void posicion_del_sesnsor(unsigned short*
        valor_leido_array, unsigned char* posicion_XYZ )
419  *
420  * Pre condiciones: Haber realizado la lectura del canal analógico.
421  *
422  * Entradas: valor_leido_array:Es un array de tres valores de entrada,
        son los valores leidos por el conversor A/D de tres sensores de
        una celda.
423  *
424  * Salidas: Posicion_XYZ: [SALIDA en unidades array]Es un valor que
        devuelve esta función, fruto de la fila que proporciona menos
        error en la medida (posicion Z) y las posiciones x e y que
        corresponden con esa fila.
425  *
426  * Otros efectos: Ninguno
427  *
428  * Funcionamiento: Esta es la función a la que se llama una vez se han
        leido los valores de tres sensores y lo que hace es llamar
        sucesivas veces a una función que busca en una fila y devuelve el
        error supuesto en la medida, la llama tantas veces como filas
        tiene la matríz y elige la medida que devuelve menor error.
429  *
430  * Nota:
431  *********************************************************/
432 void posicion_del_sesnsor(unsigned short* valor_leido_array, unsigned
      char* posicion_XYZ )
433 {
434 unsigned char filas=MAX_filas-1;
435 unsigned char indice=0;
436 unsigned char posicion_XY_temporal[2];
437 unsigned char posicion_XY[2];
438 unsigned char posicion_Z;
439 unsigned char distancia_eje_array[3];
440 float error_anterior=200;
441 float error;
442
443 while(indice<=filas)
444     {
445     buscar_en_matriz_3_datos_sin_punteros_variables_short(valor_leido_array,indice,
          distancia_eje_array );
446     //Comprobación de error, decíamos que si
          distancia_eje_array[0]==255 entonces la busqueda no ha dado
          resultados.
447     if(distancia_eje_array[0]!=255) //Por tanto si es distinta de 255
          es lógico que prosigamos con el cálculo.
```

```
448         {//Aqui hay que introducir la funcion buscar posiciones., seguir por aquí
449         error=Buscar_posiciones_y_errores(distancia_eje_array, posicion_XY_temporal);
450         if(error<error_anterior)
451             {   //Se ha reducido el error.
452             error_anterior=error;                    //Se actualiza el error;
453             posicion_Z=indice;                       //Se actualiza la posición z del imán.
454             posicion_XY[0]=posicion_XY_temporal[0]; //Se actualiza la posición x del imán.
455             posicion_XY[1]=posicion_XY_temporal[1]; //Se actualiza la posición y del imán.
456             }
457         }
458     indice++;
459     }
460     //Terminado el bucle ya se tienen las coordenadas del imán con el menor error posible. A continuación se procede a pasarlas a la función superior.
461 posicion_XYZ[0]=posicion_XY[0]; //X
462 posicion_XYZ[1]=posicion_XY[1]; //Y
463 posicion_XYZ[2]=posicion_Z;     //Z
464 }
```

Contenido del archivo “calculo.h” que contiene los prototipos de funciones.

```
1  //************************************
2  //calculo.h                          *
3  //************************************
4  //Prototipos de las funciones
5  unsigned char buscar_en_matriz(unsigned char valor_leido, unsigned char distancia );
6  void buscar_en_matriz_3_datos_sin_punteros(unsigned char* valor_leido_array, unsigned char distancia, unsigned char* indice_array);
7  void buscar_en_matriz_3_datos_con_punteros(unsigned char* valor_leido, unsigned char distancia, unsigned char* indice);
8  void buscar_en_matriz_3_datos_con_punteros_optimizado(unsigned char* valor_leido, unsigned char distancia, unsigned char* indice);
9  void buscar_en_matriz_3_datos_Totalmente_desoptimizado(unsigned char* valor_leido_array, unsigned char distancia, unsigned char* indice_array );
10 void buscar_en_matriz_3_datos_sin_punteros_variables_short(unsigned short* valor_leido_array, unsigned char distancia, unsigned char* indice_array );
11 float Buscar_posiciones_y_errores(unsigned char* radios, unsigned char* posicion);
12 void posicion_del_sesnsor(unsigned short* valor_leido_array, unsigned char* posicion_XYZ );
13 //Funciones interfaz
```

Variables y definiciones contenidas en el archivo “calculo_variables_y_definiciones.h”

```
1  //************************************
```

```
2 //calculo_variables_y_definiciones.h    *
3 //************************************
4 #include <p32mx460f512l.h>
5 #include "calculo.h"
6 #include <math.h>
7 //Definiciones
8 //Constantes del tamaño y número de divisiones de la zona de búsqueda.
9 #define MAX_filas 101
10 #define MAX_columnas 101
11
12 //Constantes que definen las posiciones de los tres sensores respecto
      a un sistema de coordenadas fijo en el sensor 1 son.
13 //Para realizar esta operacion conocemos la posición de  los tres
      sensores en mm pero tenemos que pasarlas a unidades de matriz, esto
      es que 25mm son 100 divisiones, entonces para pasara de mm a
      divisiones hay que dividir entre 25 y multiplicar por 100 y dividir
      entre 25
14     //Estos valores hay que actualizarlos si cambiamos
          Bz!!!!!!!!!!!!!!!!!!!!!!!!!!
15 #define X_02 18.16
16 #define X_03 9.08
17 #define Y_03 18.16
18
19 //Declaración de constantes
20
21 const unsigned short Espacio_de_busqueda[MAX_filas][MAX_columnas]=
22 {
23 /*Aquí están definiddos cada uno de los elementos de un array
      bidimensional de 100 filas por 100 columnas, no se ha incluido en
      la documentación impresa pues no aporta información a la vista.
24 Si se quiere ver por curiosidad o análisis, referirse al código
      firmware del PIC en formato electrónico.*/
25 };
26
27 //Declaración de variables globles
```

Funciones contenidas en el archivo “uart_basic.c”

```
1 //************************************
2 //uart_basic.c                        *
3 //************************************
4 #include "uart_basic_variables_y_definiciones.h"
5 #include "main.h"
6 //**************************************
7 //void inicializa_rs232_y_optimiza()     *
8 //**************************************
9 //Optimiza el sistema, inicializa la UART2 y envía una cabecera de
      texto
10 // para nuestra tranquilidad y poder comprobar que existe comunicación.
11 void inicializa_rs232_y_optimiza()
12 {
13     int pbClk;
14     pbClk=SYSTEMConfig(SYS_FREQ, SYS_CFG_WAIT_STATES | SYS_CFG_PCACHE);
15     OpenUART2(UART_EN, UART_RX_ENABLE | UART_TX_ENABLE,
          pbClk/16/DESIRED_BAUDRATE-1);    // calculate actual BAUD
```

```
        generate value.
16      putsUART2("*** UART Inicializada ***\r\n");
17      putsUART2("*** Este mensaje se escribe para poder observar algo en
            el Hyper teminal de windows al hacer las pruebas  ***\r\n");
18  }
19
20
21  /*************************************************************
22   * Función: void funcion_espejo_para_puerto_rs232(int
         numero_repeticiones)
23   *
24   * Pre condiciones: Inicializar el puerto serie.
25   *
26   * Entradas: numero_repeticiones: El número de veces que la función va
         a actuar como espejo.
27   *
28   * Salidas: Ninguna
29   *
30   * Otros efectos: Ninguno
31   *
32   * Funcionamiento: Cuando llega un caracter por el puerto serie lo
         devuelve, es lo que se conode cono un echo.
33   *
34   * Nota: Esta función se ha utilizaro para hacer las pruebas iniciales
         con la UART y el puerto serie.
35   *************************************************************/
36  void funcion_espejo_para_puerto_rs232(int numero_repeticiones)
37  //Esta función devuelve exactamente lo mismo que recibe, realiza esta
        acción tantas veces como parámetro de entrada.
38  {
39     unsigned char data[3];
40  data[1]='\n';
41  data[2]='\n';
42  for(;numero_repeticiones>0;numero_repeticiones--)
43      {
44      while(!DataRdyUART2()); /* Escribe datos en la UART2 */
45          {}
46      data[0] = (char)ReadUART2(); /* Lee la UART2. */
47      while(BusyUART2()); /* Espera a que la UART2 haya terminado de
            transmitir. */
48          {}
49      putcUART2(data[0]); /* Escribe el dato dos veces. */
50      putcUART2(data[0]);
51      }
52  }
53
54
55  /*************************************************************
56   * Función: unsigned char lectura_basica_con_timeout(unsigned char
         *dato)
57   *
58   * Pre condiciones: Inicializar el puerto serie.
59   *
60   * Entradas: Ninguna
```

```
61  *
62  * Salidas: unsigned char *dato: Puntero a la zona de memoria donde se
        guarda el byte.
63  *
64  * Otros efectos: Ninguno
65  *
66  * Funcionamiento: Lee un Byte, es decir, pone el puero a la escucha,
        pero durante un tiempo limitado definido por timeout.
67  *
68  * Nota:
69  ************************************************************/
70 unsigned char lectura_basica_con_timeout(unsigned char *dato)
71 {
72 int tiempo;
73 unsigned char timeout=0;
74 T3CON=0x8000;
75 tiempo=0;
76 TMR3=0;
77
78 while((!DataRdyUART2())&&(!timeout)) /* Espera a recibir un dato en la
       UART2 o a que salte el timeout. Queremos un timeout de 1s*/
79      {
80      if (TMR3>40000)
81           {
82           TMR3=0;
83           tiempo++;
84           if(tiempo==5000)
85                {
86                timeout=255;
87                }
88           }
89      }
90 if(timeout==0)        //No ha saltado por timeout
91      {
92      dato[0] = (char)ReadUART2();
93      return 0;         //Implica lectura
94      }
95 return 255;           //Implica timeout
96 }
97
98
99 /*************************************************************
100  * Función: void funcion_espejo_para_puerto_rs232_con_timeout(int
        numero_repeticiones)
101  *
102  * Pre condiciones: Inicializar el puerto serie.
103  *
104  * Entradas: Ninguna
105  *
106  * Salidas: numero_repeticiones: El número total de bytes que se van a
        reflejar.
107  *
108  * Otros efectos: Ninguno
109  *
```

```
110  * Funcionamiento: Cuando llega un caracter por el puerto serie lo
         devuelve, es lo que se conode cono un echo. Además tiene un
         timeout.
111  *
112  * Nota: Similar a la función void
         funcion_espejo_para_puerto_rs232(int numero_repeticiones) pero con
         timeout.
113  ************************************************************/
114 void funcion_espejo_para_puerto_rs232_con_timeout(int
       numero_repeticiones)
115 {
116    unsigned char data[3];
117 data[1]='\n';
118 data[2]='\n';
119 for(;numero_repeticiones>0;numero_repeticiones--)
120     {
121 //Lectura
122     if(lectura_basica_con_timeout(data)==0) //Se ha leido correctamente
123         {//Escritura
124         while(BusyUART2()); /* Wait till the UART transmitter is free.
               */
125         {}
126         putcUART2(data[0]); /* Write data into Tx. Parece que el
               primero nunca se escribe. */
127         putcUART2(data[0]);
128         }
129     }
130 }
131
132
133 /************************************************************
134  * Función: unsigned char leer_UART(unsigned char *mensaje_entrante)
135  *
136  * Pre condiciones: Inicializar el puerto serie.
137  *
138  * Entradas: Ninguna
139  *
140  * Salidas: unsigned char *mensaje_entrante: Puntero a la zona de
         memoria donde se guardará el mensaje.
141  *          return(resultado):
142  *           0 -> La lectura se ha realizado de forma correcta.
143  *           1 -> Ha habido error de timeout en la recepción del
         mensaje.
144  *           2 -> Timeout en la lectura de la cabecera 'P'.
145  *           3 -> Timeout en la lectura del Nº de Bytes.
146  *           4 -> La cabecera es distinta de 'P'.
147  * Otros efectos: Ninguno
148  *
149  * Funcionamiento: Esta función permite leer los datos del PUERTO
         serie con el protocolo que hemos acordado, es decir,
         destinatario->longitud->mensaje
150  *
151  * Nota:
152  ************************************************************/
```

```
153 unsigned char leer_UART(unsigned char *mensaje_entrante)
154 {
155 unsigned char temporal;
156 unsigned char timeout=0;
157 unsigned char leyendo=0;
158 unsigned char resultado;
159 if(lectura_basica_con_timeout(&temporal)==0)            //Implica que
      se ha leido correctamente el Byte 1
160     {
161     if(temporal=='P')    //Cabecera correcta,
162         {
163         if(lectura_basica_con_timeout(&temporal)==0)     //Se ha leido
              correctamente el Byte 2
164             {    //Ahora mismo en temporal se encuentra el número de
                  Bytes de datos que se deben leer.
165 //
166 //IMPORTANTE
167 //Como lo estamos mandando mediante el terminal, los números se
      interpretan como
168 // caracteres asci de modo que debemos adaptarlo. RESTANDO 48 ENTRE 0
      Y 9
169
170 //temporal=temporal-48;
171
172
173             while((leyendo<temporal)&&(!timeout))
174                 {
175                 timeout=lectura_basica_con_timeout(&mensaje_entrante[leyendo]);
176                 leyendo++;
177                 }
178             if(timeout!=0)
179                 {//Ha habido error de timeout en la recepción del
                      mensaje
180                 resultado=1;
181                 }
182             else
183                 {// La lectura se ha realizado de forma correcta
184                 resultado=0;
185                 }
186             }
187         else
188             {//Timeout en la lectura del Nº de Bytes
189             resultado=3;
190             }
191         }
192     else
193         {//La cabecera es distinta de 'P'
194         resultado=4;
195         }
196     }
197 else
198     {//Timeout en la lectura de la cabecera 'P'
199     resultado=2;
200     }
```

```
201 return resultado;
202 }
203
204
205 /*************************************************************
206  * Función: void test_respuesta_ante_entrada_protocolo()
207  *
208  * Pre condiciones: Inicializar el puerto serie.
209  *
210  * Entradas: Ninguna
211  *           Solo envía lo que recibe, pues es la respuesta ente
       entrada protocolo en modo espejo.
212  *
213  * Salidas: Ninguna.
214  *
215  * Otros efectos: Ninguno
216  *
217  * Funcionamiento: Esta función permite leer los datos del PUERTO
       serie con el protocolo que hemos acordado, es decir,
       destinatario->longitud->mensaje
218  *
219  * Nota: Funciona en el modo espejo, es decir, envía un echo.
220  *************************************************************/
221 void test_respuesta_ante_entrada_protocolo()
222 {
223 unsigned char mensaje[20];
224 unsigned char resultado;
225 resultado=leer_UART(mensaje);
226
227 switch(resultado)
228     {
229     case 0:
230     putsUART2("Protocolo␣correcto\r\n");
231     putsUART2("El␣primer␣Byte␣es:␣");
232     putcUART2(mensaje[0]); /* Write data into Tx. */
233     putsUART2("\r\n");
234     break;
235     case 1:
236     putsUART2("Ha␣habido␣error␣de␣timeout␣en␣la␣recepción␣del␣
          mensaje.\r\n");
237     break;
238     case 2:
239     putsUART2("Timeout␣en␣la␣lectura␣de␣la␣cabecera␣'P'.\r\n");
240     break;
241     case 3:
242     putsUART2("Timeout␣en␣la␣lectura␣del␣Nº␣de␣Bytes.\r\n");
243     break;
244     case 4:
245     putsUART2("La␣cabecera␣es␣distinta␣de␣'P'.\r\n");
246     break;
247     default:
248     putsUART2("WTF␣Error␣no␣contemplado\r\n");
249     break;
250     }
```

```
251 }
252
253
254 /*************************************************************
255  * Función: void comunicacion_serie()
256  *
257  * Pre condiciones: Inicializar el puerto serie.
258  *
259  * Entradas: Ninguna
260  *           Emplea funciones inteface para acceder a los datos.
261  *
262  * Salidas: Ninguna.
263  *
264  * Otros efectos: Ninguno
265  *
266  * Funcionamiento: Esta función lee, recibe el mensaje y dependiendo
       de lo que le pidan pues se lo envía mediante una función interfaz.
       Es necesario hacerlo mediante una función interfaz porque se
       encuentra en un módulo diferente al moduo en el que se encuentran
       los datos.
267  *  A-> Manda los valores de las lecturas analógicas de una celda.
268  *  B-> Valor de la posición de la primera celda una vez se ha
       procesado.
269  *  C-> tanto el valor de los sensores analógicos como el valor de la
       posición para la primera celda.
270  *  D-> Posición una vez procesado de las cuatro celdas.
271  *  E-> Valor de los sensores Hall de las cuatro celdas.
272  *
273  * Nota:
274  *************************************************************/
275 void comunicacion_serie()
276 {
277 unsigned char mensaje[20];
278 unsigned char mensaje_a_enviar[27];
279 unsigned char manejador_de_error;
280 unsigned char indice_envio;
281 manejador_de_error=leer_UART(mensaje);
282     //En estos momentos la comprobación de errores es
283 if (manejador_de_error==0)
284     { //El protocolo parece correcto, vamos a hacer un case sobre que
          mensaje ha entrado
285     switch(mensaje[0])
286         {
287         case 'A':   //El mensaje es A, entonces me pide que le mande
              los valores de las lecturas anallógicas.
288         {
289         mensaje_a_enviar[0]='M';
290         mensaje_a_enviar[1]=0x07;
291         mensaje_a_enviar[2]='A';
292         get_valores_analogicos(&(mensaje_a_enviar[3]));
293
294         //Es cierto que se podría utilizar un bucle for para enviar
              todos estos caracteres, pero no se ha hecho.
295         putcUART2(mensaje_a_enviar[0]); /* Write data into Tx. */
```

```
296         putcUART2(mensaje_a_enviar[1]); /* Write data into Tx. */
297         putcUART2(mensaje_a_enviar[2]); /* Write data into Tx. */
298         putcUART2(mensaje_a_enviar[3]); /* Write data into Tx. */
299         putcUART2(mensaje_a_enviar[4]); /* Write data into Tx. */
300         putcUART2(mensaje_a_enviar[5]); /* Write data into Tx. */
301         putcUART2(mensaje_a_enviar[6]); /* Write data into Tx. */
302         putcUART2(mensaje_a_enviar[7]); /* Write data into Tx. */
303         putcUART2(mensaje_a_enviar[8]); /* Write data into Tx. */
304         }
305         break;
306         case 'B':   //El mensaje B nos está pidiendo el valor de la
                posición una vez se ha procesado
307         {
308         mensaje_a_enviar[0]='M';
309         mensaje_a_enviar[1]=0x04;
310         mensaje_a_enviar[2]='B';
311         get_coordenadas_del_iman(&(mensaje_a_enviar[3]));
312         putcUART2(mensaje_a_enviar[0]); /* Write data into Tx. */
313         putcUART2(mensaje_a_enviar[1]); /* Write data into Tx. */
314         putcUART2(mensaje_a_enviar[2]); /* Write data into Tx. */
315         putcUART2(mensaje_a_enviar[3]); /* Write data into Tx. */
316         putcUART2(mensaje_a_enviar[4]); /* Write data into Tx. */
317         putcUART2(mensaje_a_enviar[5]); /* Write data into Tx. */
318         }
319         break;
320         case 'C':   //El mensaje C nos pide tanto el valor de los
                sensores analogicos como el valor de la posicion, esta es
                una forma de comprobar que da el mismo resultado
                calculándolo mediante Matlab que con el codigo introducido
                en el pic.
321         {
322         mensaje_a_enviar[0]='M';
323         mensaje_a_enviar[1]=0x0A;   //Esto son 10 en decimal
324         mensaje_a_enviar[2]='C';
325         get_valores_analogicos(&(mensaje_a_enviar[3]));      //Rellena
                6 Bytes, del 3 al 8
326         get_coordenadas_del_iman(&(mensaje_a_enviar[9]));    //Rellena
                3 Bytes, del 9 al 11
327         //En este caso el envio lo vamos a hacer con un bucle for
328         for(indice_envio=0;indice_envio<12;indice_envio++)  //Sería lo
                mismo que poner <=11, pero me ha parecido correcto poner lo
                que he puesto.
329             {
330             putcUART2(mensaje_a_enviar[indice_envio]); /* Write data
                    into Tx. */
331             }
332         }
333         break;
334         case 'D':   //El mensaje D nos pide el valor de la posición
                una vez procesado de las cuatro cledas.
335                     //Para obtenerlo se va a emplear una función
                            interfaz específica.
336         {
337         mensaje_a_enviar[0]='M';
```

```
338         mensaje_a_enviar[1]=0x0D;    //Esto son 13 en decimal
339         mensaje_a_enviar[2]='D';     //Echo del mensaje
340         get_coordenadas_del_iman_4_celdas(&(mensaje_a_enviar[3])); //
                Rellena 12 Bytes con el valor xyz de las cuatro celdas.
341         for(indice_envio=0;indice_envio<15;indice_envio++) //Sería lo
                mismo que poner <=14, pero me ha parecido correcto poner lo
                que he puesto.
342             {
343             putcUART2(mensaje_a_enviar[indice_envio]); /* Write data
                    into Tx. */
344             }
345         }
346         break;
347         case 'E':   //Petición del valor de los sensores Hall de las
                cuatro celdas.
348         {
349         mensaje_a_enviar[0]='M';
350         mensaje_a_enviar[1]=0x19;    //Esto son 25 en decimal
351         mensaje_a_enviar[2]='E';     //Echo del mensaje
352         get_valores_analogicos_4_celdas(&(mensaje_a_enviar[3]));
                //Rellena 24 Bytes a enviar.
353         for(indice_envio=0;indice_envio<27;indice_envio++) //Sería lo
                mismo que poner <=26, pero me ha parecido correcto poner lo
                que he puesto.
354             {
355             putcUART2(mensaje_a_enviar[indice_envio]); /* Write data
                    into Tx. */
356             }
357         }
358         break;
359
360         default:
361         putsUART2("Desconocido\r\n");
362         break;
363         }
364     }
365 }
```

Contenido del archivo “uart_basic.h” que contiene los prototipos de funciones.

```
1  //*************************************
2  //uart_basic.h                        *
3  //*************************************
4  //Prototipos de funcion
5  void inicializa_rs232_y_optimiza();
6  void funcion_espejo_para_puerto_rs232(int numero_repeticiones);
7  unsigned char lectura_basica_con_timeout(unsigned char *dato);
8  void funcion_espejo_para_puerto_rs232_con_timeout(int
       numero_repeticiones);
9  unsigned char leer_UART(unsigned char *mensaje_entrante);
10 void test_respuesta_ante_entrada_protocolo();//Tener cuidado con el
       numero de caracteres si está en asci o en numero natural.
11 void comunicacion_serie();
```

Variables y definiciones contenidas en el archivo “uart_basic_variables_y_definiciones.h”

```
1 //************************************
2 //uart_basic_variables_y_definiciones.h*
3 //************************************
4 #include "uart_basic.h"
5 #include <plib.h>
6
7 //Definiciones
8 #define SYS_FREQ              (80000000L)
9 #define DESIRED_BAUDRATE      (19200)       // El BaudRate deseado
```

Funciones contenidas en el archivo “main.c”

```
1 //************************************
2 //main.c                              *
3 //************************************
4 // Configuration Bit settings
5 // SYSCLK = 80 MHz (8MHz Crystal/ FPLLIDIV * FPLLMUL / FPLLODIV)
6 // PBCLK = 40 MHz
7 // Primary Osc w/PLL (XT+,HS+,EC+PLL)
8 // WDT OFF
9 // Other options are don't care
10 //
11 #pragma config FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FPLLODIV = DIV_1,
     FWDTEN = OFF
12 #pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV_1
13
14 #include <p32mx460f512l.h>
15 #include "uart_basic.h"
16 #include "analog.h"
17 #include "calculo.h"
18 #include "main_variables_y_definiciones.h"
19
20
21 int main()
22 {
23 unsigned char celda,resultado;
24 inicializa_rs232_y_optimiza();
25 initADC_automatic_sampling_timing (0xFFF0); //Inicializa el conversor
     AD en modo de muestreo automático. El '0xFFF8 es
     1111.1111.1111.1000' implica que los puertos RB0 RB1 y RB2 se
     emplean como entradas analógicas.
26 TRISB=0xFFFF;        //Configura el puerto B para que RB0 RB1 RB3 sean
     entradas, y para facilidad que todas sean entradas.
27 TRISD=0xFFF8;    //Configura el Puerto D para que RD0 RD1 RD2 sean
     salidas, estas serán las salidas que controlan la multiplexión y
     además están conectados a 3 LED en la placa de desarrollo y hacen
     posible visualizar que canal se está leyendo en cada momento.
28 transformacion.numero=1021;
29 celda=0;
30 //Se configura el timer 2 para la recta de carga del condensador que
     actua como filtro paso bajo 5*tau que son 0.09s de momento.
31 T2CON=0x8070;        //Arranca timer 2 con un preescalado de 256.
32 while(1)
33     {
34
```

```
35      for (celda=0;celda<4;celda++)    //Va a recorrer las cuatro celdas
            haciendo la conversion A/D
36          {
37          PORTDbits.RD0=(celda&0x01);       //Se asigna el bit 0 al puerto
                RD0. Realizando una xor con 0x01, es decir 0000.0001.
38          if((celda&0x02)==0x02)
39              PORTDbits.RD1=1;          //Se asigna el bit 1 al puerto
                    RD1. Realizando una xor con 0x02, es decir 0000.0010.
40          else
41              PORTDbits.RD1=0;
42          if((celda&0x04)==0x04)
43              PORTDbits.RD2=1;          //Se asigna el bit 2 al puerto
                    RD2. Realizando una xor con 0x04, es decir 0000.0100.
44          else
45              PORTDbits.RD2=0;
46          TMR2=0;
47          //Configurar T2CON como 1000.0000.0111.0000
48          //                          _          ___
49          //                          |           |_Prescale 1:256
50          //                          |_Encinede el timer 2
51          //La frecuencia del bus de periféricos es 80Mhz/2=40Mhz
52          //Tras es preescalado, la frecuencia del timer 2 es
                ft2=40Mhz/256=156250Hz
53          //Y el periodo del timer 2 es 6.4us (microsegundos)
54          //Si se requiere un retardo de 0.09s entonces
55          //La cuenta del timer debe se 14062
56          while (TMR2<14062)
57              {}  //Espera de 5 Tau.
58          lee_y_devuelve_3_ch_analog_automatico_con_10_bits_realizando_media_
                para_eliminar_rizado(&valor_leido_hall[celda*3]); //Realiza
                la conversión de los 9 canales y le pasamos un puntero, en
                esa zona de memoria guardará los datos.
59
60          }
61
62      for (celda=0;celda<4;celda++)    //Ahora este bucle también recorre
            las cuatro celdas, pero replicar cuatro veces la búsqueda de
            datos.
63          {
64          posicion_del_sesnsor(&valor_leido_hall[celda*3],
                &posicion_del_iman_XYZ[celda*3]);
65          }
66
67      comunicacion_serie();
68      //test_respuesta_ante_entrada_protocolo();
69      //funcion_espejo_para_puerto_rs232_con_timeout(9);
70      //funcion_espejo_para_puerto_rs232(9);
71      }
72  return 0;
73      }
74
75  //**************************
76  //**************************
77  //**    FUNCIONES INTERFAZ    **
```

```
78 //**************************
79 //**************************
80
81 // void get_valores_analogicos(unsigned char *array_hall)
82 // Función interfaz que se emplea para pasar al módulo de
       comunicaciones el valor leido en los tres sensores hall de la
       primera celda.
83 void get_valores_analogicos(unsigned char *array_hall)
84 {
85 transformacion.numero=valor_leido_hall[0];
86      array_hall[0]=transformacion.caracter[1];
87      array_hall[1]=transformacion.caracter[0];
88 transformacion.numero=valor_leido_hall[1];
89      array_hall[2]=transformacion.caracter[1];
90      array_hall[3]=transformacion.caracter[0];
91 transformacion.numero=valor_leido_hall[2];
92      array_hall[4]=transformacion.caracter[1];
93      array_hall[5]=transformacion.caracter[0];
94 }
95
96
97 // void get_coordenadas_del_iman(unsigned char *array_posiciones)
98 // Función interfaz que se emplea para pasar al módulo de
       comunicaciones las coordenadas xyz del imán de la primera celda.
99 void get_coordenadas_del_iman(unsigned char *array_posiciones)
100 {
101 array_posiciones[0]=posicion_del_iman_XYZ[0];
102 array_posiciones[1]=posicion_del_iman_XYZ[1];
103 array_posiciones[2]=posicion_del_iman_XYZ[2];
104 }    //No utilizamos un bucle for porque así es más rápido, la
       diferencia no es abismal, pero es más rápido.
105
106
107 // void get_coordenadas_del_iman_4_celdas(unsigned char
       *array_posiciones)
108 // Función interfaz que se emplea para pasar al módulo de
       comunicaciones
109 // las coordenadas xyz de las cuatro celdas.
110 void get_coordenadas_del_iman_4_celdas(unsigned char *array_posiciones)
111 {
112      unsigned char indice_copia;
113      for(indice_copia=0;indice_copia<12;indice_copia++)
114          {
115          array_posiciones[indice_copia]=posicion_del_iman_XYZ[indice_copia];
116          }
117 }
118
119
120 // void get_valores_analogicos_4_celdas(unsigned char *array_hall)
121 // Función interfaz que se emplea para pasar al módulo de
       comunicaciones el valor leido en los tres sensores hall en cada una
       de las cuatro celdas.
122 void get_valores_analogicos_4_celdas(unsigned char *array_hall)
123 {
```

```
124     unsigned char indice_copia;
125     for(indice_copia=0;indice_copia<12;indice_copia++)
126         {
127         transformacion.numero=valor_leido_hall[indice_copia];
128             array_hall[indice_copia*2]=transformacion.caracter[1];
129             array_hall[(indice_copia*2)+1]=transformacion.caracter[0];
130         }
131 }
```

Contenido del archivo “main.h” que contiene los prototipos de funciones.

```
1  //************************************
2  //main.h                             *
3  //************************************
4  //Prototipos de funcion
5
6  //Funciones interfaz
7  void get_valores_analogicos(unsigned char *array_hall);
8  void get_coordenadas_del_iman(unsigned char *array_posiciones);
9  void get_coordenadas_del_iman_4_celdas(unsigned char
       *array_posiciones);
10 void get_valores_analogicos_4_celdas(unsigned char *array_hall);
```

Variables y definiciones contenidas en el archivo “main_vaariables_y_definiciones.h”

```
1  //************************************
2  //main_variables_y_definiciones.h     *
3  //************************************
4  //main_variables_y_definiciones
5  #include "main.h"
6
7  //Variables globales
8  static unsigned short valor_leido_hall[12];
9  static unsigned char posicion_del_iman_XYZ[12];
10
11 //Union globales
12 union datos
13     {
14     unsigned short numero;
15     unsigned char caracter[2];
16     };
17 union datos transformacion;
```

Código en Matlab utilizado para análisis

grafic_3D_mag_z

Funcionamiento:
Introduciendo todos los parámetros muestra por pantalla una matriz de valores de campos magnéticos en distintos puntos y con la función meshc se hace una gráfica en 3 dimensiones con curvas de nivel. Se basa en la función “campo_magnetico_z” que calcula el campo magnético en la dirección del eje z en el punto dado.

Entradas:

- tamano_z: Es el tamaño que va a tener la representación del campo magnético medida desde el extremo del imán $[m]$.
- tamano_y: Tamaño que va a tener la representación del campo magnético desde el centro del imán hacia el exterior de forma radial $[m]$.
- mu0: Permeabilidad del vacío $[T \cdot m/A]$ $[N/A^2]$.
- m: Momento magnético del imán $[A \cdot m^2]$.
- l: Longitud del imán $[m]$.
- a: Radio del imán $[m]$

```
%%Crea y muestra por pantalla el campo con la funcion mesh
function
    Bz=grafic_3D_mag_z(tamano_z,tamano_y,mu0,m,l,a,numero_de_iteraciones)
z_grid=l/2:tamano_z/numero_de_iteraciones:(tamano_z+l/2); %%Eso que
    parece un l es una l (letra) de longitud
y_grid=0:tamano_y/numero_de_iteraciones:tamano_y;
iteraciones_z=size(z_grid);
iteraciones_y=size(y_grid);

indice_z=1;

for indice_z=1:1:iteraciones_z(2),%Recorre las filas fila1 fila2 fila3
    indice_y=1;
    for indice_y=1:1:iteraciones_y(2),%se mueve cambiando de columnas.
        %Aqui ya tenemos un indice_z y un indice_y por lo que ya
            podemos asignar el valor a la matriz.
        Bz(indice_z,indice_y)=campo_magnetico_z(mu0,m,l,a,y_grid(indice_y),z_grid(in
    end
end
[X Y]=meshgrid(y_grid,z_grid);

meshc(X,Y,Bz)
xlabel('ejeY')
ylabel('ejeZ')
zlabel('ejeX')
Bz
```

campo_magnetico_z.

Funcionamiento:
Proporciona el valor del campo magnético en el punto definido por las coordenadas y,z. Dado que hemos hecho que el punto esté en el plano x=0. Esta función multiplica las constantes necesarias por el valor devuelto por la función “integral_solenoide”

Entradas:

- mu0: Permeabilidad del vacío $[T \cdot m/A]$ $[N/A^2]$.
- m: Momento magnético del imán $[A \cdot m^2]$.
- l: Longitud del imán.
- a: Radio del imán.
- y: Coordenada y del punto sobre el que se va a calcular el campo magnético.
- z: Coordenada z del punto en el que se va a calcular el campo magnético.

```
1 %Esta funcion calcula el cámpo magnético en la direccion del eje z en
      cualquier punto del espacio.
2 function Bz=campo_magnetico_z(mu0,m,l,a,y,z)
3 Bz=((mu0*m)/(2*pi^2*a*l))*integral_solenoide(l,a,y,z);
```

integral_solenoide.

Funcionamiento:
Proporciona el valor de la integral:

$$\int_{z-L/2}^{z+L/2} \left[\int_{-\pi/2}^{\pi/2} \frac{a - y \cdot \sin\theta}{(a^2 + z^2 + y^2 - 2 \cdot a \cdot y \cdot \sin\theta)^{3/2}} \cdot d\theta \right] \cdot dz$$

Integrando por rectángulos. Se basa en el valor de retorno de la función “integral_espira”

- l: Longitud del imán.
- a: Radio del imán.
- y: Coordenada y del punto sobre el que se va a calcular el campo magnético.
- z: Coordenada z del punto en el que se va a calcular el campo magnético.

```
1 function integral=integral_solenoide(l,a,y,z)
2     %l es la longitud del solenoido (longitud del iman)
3     %a es el radio del solenoide (radio del imán)
4     %(y,z) son las coordenadas espaciales del punto, colocando
          apropiadamente
5     %los ejes hemos conseguido hacer x=0 y dada la simetria radial del
6     %problema el campo en x se hace 0
7 z_vector=(z-l/2):(l/100):(z+l/2);
8 temp=size(z_vector); %esto nos da las dimensiones del vector y por lo
      %%anto el número de vece que tendremos que sumar.
9 integral=0;
10 for indice=1:1:temp(2),
11     integral=integral+integral_espira(a,y,z_vector(indice))*(l/100);
          %Se multiplica por el ancho
12 end
```

integral_espira

Funcionamiento:
Proporciona el valor de la integral:

$$\int_{-\pi/2}^{\pi/2} \frac{a - y \cdot \sin\theta}{(a^2 + z^2 + y^2 - 2 \cdot a \cdot y \cdot \sin\theta)^{3/2}} \cdot d\theta$$

Entradas:

- a: Radio del la espira imaginaria por la que circula un diferencial de corriente.
- y: Coordenada y del punto sobre el que se va a calcular el campo magnético.
- z: No es la coordenada Z del punto, sino la distancia en la dirección del eje z entre el punto y el diferencial de corriente.

```
1 %esta es a función que hace la integral numérica del campo magnético
      en la espira.
2
3 function integral=integral_espira(a,y,z)
4
5 teta=-pi/2:0.005:pi/2;
6 funcion_a_integrar=(a-y*sin(teta))./((a^2+z^2+y^2-2*a*y*sin(teta)).^(3/2));
7 %aqui el codigo de integracion por cuadrados(rectangulos)
8 integral=0;
9 temp=size(teta); %esto nos da las dimensiones del vector y por lo
      tanto el número de vece que tendremos que sumar.
10 for indice=1:1:temp(2),
11     integral=integral+funcion_a_integrar(indice)*0.005;
12 end
```

Script Comprobación_de_algoritmo.m

```
1 %Lo primero es calcular la matriz que estará en una LUT
2 %%%%%
3 %Hacemos lo siguiente, nosotros le damos una posición, cone eso se
      calcula el campo magnético medido por cada sensor, luego con esso
      valores buscamos en la tabla el menor eror y calculamos la
      posición, si este razonamiento es correcto, entonces devería
      devolvernos unos valores de posiciones parecidos a los que nosotros
      le introducimos.
4 %%%%%
5 %Definiciones
6 mu0=4*pi*10^-7;
7 m=0.2306;
8 a=0.006;   %6mm de radio
9 l=0.004;   %4mm de grosor del iman
10 %PARÁMETROS DE ENTRADA
11 %la posicion del iman para esta simulaicon va a ser x=2mm y=2.5mm z=3mm
12 ImanX_real=0.004;
```

```
13 ImanY_real=0.0033;
14 ImanZ_real=0.0033;
15
16
17
18 %La matriz ya esta en el workspace, de otro modo tardaría mucho en
       calcularse.
19 %Bz=grafic_3D_mag_z(0.010,0.006,mu0,m,l,a)
20 [Bz1 Bz2 Bz3]=campo_medido(ImanX_real,ImanY_real,ImanZ_real,mu0,m,l,a)
21 [Imanx_medido Imany_medido
       Imanz_medido]=simulacion_algoritmo_medida(Bz1,Bz2,Bz3,Bz)
```

```
 1 %BIEN
 2 %Esta función nos calcula el cámpo magnético que van a leer los
       sensores dependiendo del punto en el que se sitúe el imán.
 3 function [Bz1 Bz2
       Bz3]=campo_medido(ImanX_real,ImanY_real,ImanZ_real,mu0,m,l,a)
 4
 5 %distancia Z es Imanz_real para todos los sensores.
 6 %Pero con las distancias ImanX_real,ImanY_real tendremos que hayar el
       módulo de la suma pues esa será la distancia al eje del imán que es
       distaia y real.
 7
 8 %Para el sensor Bz1 la distancia al eje es:  porqueestá situado en
       (0,0)
 9 distancia_y_real=sqrt(ImanX_real^2+ImanY_real^2);
10 Bz1=campo_magnetico_z(mu0,m,l,a,distancia_y_real,ImanZ_real+l/2);
11
12 %Para el sensor Bz2, la distancia al eje del imán es algo más
       complicada, es el módulo del vector que une los dos puntos (eje del
       imán y sensor)
13 vector_x=ImanX_real-0.00454;  %el número es la posición x del sensor 2
       habrá que cambiarlo si cambian los parámetros constructivos.
14 vector_y=ImanY_real-0; %%el número es la posición y del sensor 2 habrá
       que cambiarlo si cambian los parámetros constructivos.
15 distancia_y_real=sqrt(vector_x^2+vector_y^2);
16 Bz2=campo_magnetico_z(mu0,m,l,a,distancia_y_real,ImanZ_real+l/2);
17
18 %Para el sensor Bz3, la distancia al eje del imán es más complicada
       aún (no es para tanto), es el módulo del vector que une los dos
       puntos (eje del imán y sensor)
19 vector_x=ImanX_real-0.00227;  %el número es la posición x del sensor 3
       habrá que cambiarlo si cambian los parámetros constructivos.
20 vector_y=ImanY_real-0.00457; %%el número es la posición y del sensor 3
       habrá que cambiarlo si cambian los parámetros constructivos.
21 distancia_y_real=sqrt(vector_x^2+vector_y^2);
22 Bz3=campo_magnetico_z(mu0,m,l,a,distancia_y_real,ImanZ_real+l/2);
```

```
 1 function [Imanx_medido Imany_medido
       Imanz_medido]=simulacion_algoritmo_medida(Bz1,Bz2,Bz3,Bz)
 2 %empezamos las iteraciones para cada uno de los planos Z, Recuerda que
       cada plano z va a ser una fila de la matriz Bz. Cada fila se aleja
       una (unidad de distancia vertical) del imán y cada columna se aleja
```

```
    una (unidad de distancia horizontal), es decir, del eje de rotación
    del imán
3
4 temp=size(Bz);
5 error_anterior=255;
6 for Plano_z=38:1:temp(1),%ponemos a partir de z=38 porque es a partir
    de la distancia que en este caso los datos están ordenados de mayor
    a menor y el algoritmo de búsqueda puede funcionar de forma
    eiciente.
7     sprintf('Estamos haciendo un corte por el plano o fila %f
        ',Plano_z)
8     [radios no_encontrado]=encontrar_radios(Bz1, Bz2, Bz3,
        Bz(Plano_z,:));%Bz(Plano_z,:) lo que hace es pasarle toda la
        fila de valores de campo magnético.
9     if(no_encontrado==0) %Si se ha encontrado el valor en esta zona de
        la tabla intentaremos
10        [error distancia_x distancia_y
            no_cortan]=calculo_de_posicion(radios);
11        sprintf('Da un error aproximado de la medida es %f\n',error)
12        if(no_cortan==0)
13            if (error<error_anterior) %si ha disminuido el error ese
                punto será más válido que el anterior
14                error_anterior=error;
15                Imanx_medido=distancia_x;
16                Imany_medido=distancia_y;
17                Imanz_medido=Plano_z;
18                %NOTA, recuerda que Imanx_medido, Imany_medi.... aún
                    no está en metros, unicamente está en unidades de
                    matriz(es decir filas columnas, eso luego lo
                    transformaremos)
19            else
20            end
21        end
22    end
23 end
24 %Ahora hacemos la conversion a metros de la medida, para ello debemos
    saber que el incremento de cada unidad de la matriz en el plano
    horixontal es de 0.006m/256
25 Imanx_medido=Imanx_medido*(0.006/256);
26 Imany_medido=Imany_medido*(0.006/256);
27 %Y en el plano vertical es de 0.010m/256
28 Imanz_medido=Imanz_medido*(0.010/256);
29 error_anterior
```

```
1 %Tenemos una fila de valores ordenados de mayor a menor y tenemos que
    buscar en ese array los tres valores Bz1 Bz2 Bz3
2 function [radios no_encontrado]=encontrar_radios(Bz1, Bz2, Bz3,
    Bz_fila)
3 temp=size(Bz_fila);
4 no_encontrado=0;
5 Bz=[Bz1 Bz2 Bz3]; %metemos los tres valores del campo en un array, así
    el código queda algo más elegnate.
6 sensor=1;
```

```
radios(3)=0;
while(sensor<=3&&no_encontrado~=1),
    Inferior=1;
    Superior=temp(2);
    final=0;
    if (Bz(sensor)>Bz_fila(1)||Bz(sensor)<Bz_fila(temp(2))) %Es decir,
        si es mayor que la primera o menor que la ultima ocurre que no
        está contenido ese valor, entonces diremos que no es posible
        que se encuentre a esa altura Z y no hacemos más cálculos para
        esa fila de datos.
        no_encontrado=1;
    else
        while(final~=1),
            %fix lo que hace es truncar al entero más próximo a cero
            if Bz(sensor)>Bz_fila(fix((Inferior+Superior)/2)) %Eso
                quiere decir que va a esta más a la izquierda en el
                vector.
                %Inferior se queda como está
                Superior=fix((Inferior+Superior)/2);
            else  %Eso quiere decir que va a estar más a la derecha en
                el vector.
                Inferior=fix((Inferior+Superior)/2);
                %Superior se queda como está
            end
            if (Superior==Inferior+1) %ya hemos acotado donde se
                encuentra.
                final=1;
                if
                    (Bz_fila(Inferior)-Bz(sensor)<Bz(sensor)-Bz_fila(Superior))
                    radios(sensor)=Inferior;
                else
                    radios(sensor)=Superior;
                end
            end
        end
    end
    sensor=sensor+1;  %Ya hemos tratado un sensor, ahora tratamos el
        siguiente.
end
```

```
%Esta funcion lo que hace es calcular la posición del imán aproximada
    y el error de la medida partiendo de la distancia del imán a cada
    uno de los sensores proyectada en el plano XY a partir de tres
    puntos conocidos y tres distancias, Sería como la trilateracion
    pero en el plano.
function [error distancia_x distancia_y
    no_cortan]=calculo_de_posicion(radios)
%Los centros de cada una de las circunferencias son los siguientes:
    %X01=0;
    %Y01=0;
    %X02=4.54mm*(256ud/6mm)
    %Y02=0;
    %X03=2.27mm*(256ud/6mm)
```

```
9       %Y03=4.57mm*(256ud/6mm)
10      %IMPORTANTE IMPORTANTE IMPORTANTE!!!!!!!!!!
11 X02=189.87;
12 X03=96.85;
13 Y03=194.99;
14 %---
15 no_cortan=0;
16 %Así la intersección de las circunferencias 1 y 2 nos da un punto X Y
       válido y otro fuera del la zona de movimiento
17 if(radios(2)-radios(1)>X02||radios(1)-radios(2)>X02||radios(1)+radios(2)<X02)
18      %Esto significa que las circunferencias no se cortan.
19      no_cortan=1;
20      %A estas variables hay que darlas algún valor para que podamos
           salir de la función, pero da igual el valor que las demos pues
           no se van a usar
21      error=999;
22      distancia_x=999;
23      distancia_y=999;
24 else
25      distancia_x=(radios(1)^2-radios(2)^2+X02^2)/(2*X02);
26      distancia_y=sqrt(radios(1)^2-distancia_x^2);
27      error=abs(sqrt((X03-distancia_x)^2+(Y03-distancia_y)^2)-radios(3));
28 end
```

Función para analizar a partir de que distancia al plano del imán los datos están ordenados.

```
1 %Analizamos la matriz para saber a partir de que fila se cumple que
      los datos están ordenados de mayor a menor
2 function ordenacion=analisis(Bz)
3 tamanio=size(Bz);
4 fila=1;
5 for fila=1:1:tamanio(1),
6       ordenacion(fila)=1; %Está ordenado hasta que se demuestre lo
           contrario
7       for columna=1:1:tamanio(2)-1,
8           if (Bz(fila,columna)<Bz(fila, columna+1))
9               ordenacion(fila)=0;  %está desordenado
10          end
11      end
12      if(fila~=1)
13          if(ordenacion(fila)==1&&ordenacion(fila-1)==0)
14              sprintf('A␣partir␣de␣la␣distancia␣%f␣metros␣los␣datos␣
                   están␣ordenados',0.01/tamanio(1)*fila)
15          end
16      end
17 end
18 ejex=[0:0.01/tamanio(1):0.01/tamanio(1)*(tamanio(1)-1)];
19 plot(ejex,ordenacion)
```

6.4. Funciones en Matlab para la comunicación serie RS-232.

Abrir puerto COMx.

Abre el puerto COM y devuelve el manejador del puerto. Se configura para trabajar con 8 bits de datos, sin paridad un bit de parada, sin control de flujo y una velocidad de 19200bps, estos parámetros deben ser los mismos que se han cargado en el firmware del PIC.

```
function SerPIC=abrir_puerto_com4()
%Apertura del puerto
SerPIC = serial('COM4');
set(SerPIC,'BaudRate',19200);      %anterior 9600
set(SerPIC,'DataBits',8);
set(SerPIC,'Parity','none');
set(SerPIC,'StopBits',1);
set(SerPIC,'FlowControl','none');
fopen(SerPIC);
end
```

Cerrar puerto COMx

```
function cerrar_puerto(SerPIC)
fclose(SerPIC);
delete(SerPIC)
clear SerPIC
end
```

Mensaje A. Petición del valor de los tres sensores de la primera celda.

```
%Esta funcion lee los datos analogicos del PIC via RS232 Tantas veces
    coomo se le indica por parametro, NOTA, antes hay que abrir el
    puerto. y despues cerrarlo.
function dataloger_RS232(numero_de_muestras,SerPIC)
%Apertura del puerto. Esto ya lo hace otra funcion.
%Mensaje de petición de datos.
msg=['P',1,'A']; %'P1A';
            %Este es el mensaje con:    CABECERA:    P
            %                           Longit:      1
            %                           Comando:     A
for(indice=1:1:numero_de_muestras)
    fprintf(SerPIC,'%s',msg);
    pause(0.05);
    q=fread(SerPIC,9);
    if indice~=1
        sensoranterior1=sensor1;
        sensoranterior2=sensor2;
        sensoranterior3=sensor3;
    end
        sensor1=q(4)*256+q(5);
        sensor2=q(6)*256+q(7);
```

```
20          sensor3=q(8)*256+q(9);
21
22      if indice==1
23          clf
24          hold off;
25          subplot(2,2,1),plot(indice,sensor1)
26          subplot(2,2,2),plot(indice,sensor2)
27          subplot(2,2,3),plot(indice,sensor3)
28      else
29          hold on;
30          arrayx(1)=indice-1;
31          arrayx(2)=indice;
32          %Imprimios en la grafica 1
33          arrayy(1)=sensoranterior1;
34          arrayy(2)=sensor1;
35          subplot(2,2,1),plot(arrayx,arrayy)
36          hold on
37          %Imprimimos en la grafica 2
38          arrayy(1)=sensoranterior2;
39          arrayy(2)=sensor2;
40          subplot(2,2,2),plot(arrayx,arrayy)
41          %Imprimimos en la grafica 3
42          hold on
43          arrayy(1)=sensoranterior2;
44          arrayy(2)=sensor2;
45          subplot(2,2,3),plot(arrayx,arrayy)
46      end
47
48 end
49 %Se cierra el puerto. Esto lo hace otra funcion.
50 end
```

Mensaje B. Petición de la posición de la primera celda.

```
1 %Esta función pide al PIC que le envie la posición del imán XYZ y
      represent
2 function dataloger_posicion_RS232_posicion(numero_de_muestras,SerPIC)
3 %Apertura del puerto. Esto ya lo hace otra funcion.
4 %Mensaje de petición de datos.
5 msg=['P',1,'B'];%'P1B';
6               %Este es el mensaje con:    CABECERA:    P
7               %                           Longit:      1
8               %                           Comando:     B
9 for(indice=1:1:numero_de_muestras)
10      fprintf(SerPIC,'%s',msg);
11      pause(0.05);
12      q=fread(SerPIC,6);
13      posicionx=q(4);
14      posiciony=q(5);
15      posicionz=q(6);
16      if indice~=1
17          hold on
18          array_tiempo(1)=indice-1;
19          array_tiempo(2)=indice;
```

```
20          arrayx(1)=posicionanteriorx;
21          arrayx(2)=posicionx;
22          arrayy(1)=posicionanteriory;
23          arrayy(2)=posiciony;
24          arrayz(1)=posicionanteriorz;
25          arrayz(2)=posicionz;
26          plot(array_tiempo,arrayx,'r',array_tiempo,arrayy,'g',
                array_tiempo,arrayz,'b')
27          posicionanteriorx=posicionx;
28          posicionanteriory=posiciony;
29          posicionanteriorz=posicionz;
30      else
31          clf
32          hold off
33          plot(indice,posicionx,'+',indice,posiciony,'-',indice,posicionz,'.')
34          posicionanteriorx=posicionx;
35          posicionanteriory=posiciony;
36          posicionanteriorz=posicionz;
37      end
38  end
39  end
```

Mensaje C. Petición del valor de los tres sensores de la primera celda y de la posición.

Mensaje D. Petición de la posición de las cuatro celdas.

```
1  %Esta función pide al PIC que le envie la posición del imán XYZ de las
       cuatro celdas y representa.
2  function
       dataloger_posicion_RS232_posicion_de_4_celdas(numero_de_muestras,SerPIC)
3  %Apertura del puerto. Esto ya lo hace otra funcion.
4  %Mensaje de petición de datos.
5  msg=['P',1,'D'];%'PID';
6               %Este es el mensaje con:     CABECERA:    P
7               %                            Longit:      1
8               %                            Comando:     D
9  for(indice=1:1:numero_de_muestras)
10     if indice~=1
11         posicion_anterior=posicion;
12     end
13
14     fprintf(SerPIC,'%s',msg);
15     pause(0.05);
16     q=fread(SerPIC,15);
17     for(indice_posicion=1:1:12)
18         posicion(indice_posicion)=q(indice_posicion+3);
19     end
20     posicion
21     if indice==1
22         posicion_anterior=posicion;
23     end
24
25     for(indice_celda=1:1:4)
26         if indice~=1
```

```
27              hold on
28              for(indice_coordenada=1:1:3)
29                  indice_temporal=(indice_celda-1)*3+indice_coordenada;
30
31                  array_coordenada(indice_coordenada,1)=posicion_anterior
                        ((indice_celda-1)*3+indice_coordenada);
32                  array_coordenada(indice_coordenada,2)=posicion((indice_celda-1)*
                        3+indice_coordenada);
33                  array_tiempo(1)=indice-1;
34                  array_tiempo(2)=indice;
35              end
36              subplot(2,2,indice_celda)
37              plot(array_tiempo,array_coordenada(1,:),'r',array_tiempo,
                    array_coordenada(2,:),'g',array_tiempo,array_coordenada(3,:),'b')
38
39              legend('Relacion a coordenada x','relacion a coordenada
                    y','relacion a coordenada z')
40              switch indice_celda
41                  case 1
42                      title ('Gráfica de la celda 0\_0');
43                  case 2
44                      title ('Gráfica de la celda 0\_1');
45                  case 3
46                      title ('Gráfica de la celda 1\_0');
47                  otherwise
48                      title ('Gráfica de la celda 1\_1');
49              end
50          else
51              clf
52              hold off
53              %plot(indice,posicionx,'+',indice,posiciony,'-',
                    indice,posicionz,'.')
54          end
55      end
56      if indice~=1
57
58          posicion;
59          array_coordenada;
60          posicion_anterior;
61      end
62  end
63  end
```

Mensaje E. Petición del valor de los sensores Hall de las cuatro celdas.

```
1  %Funcion que muestra por pantalla las tensiones de los tres sensores
       hall de cada celda.
2  function dataloger_valor_hall_de_4_celdas(numero_de_muestras,SerPIC)
3  %Apertura del puerto. Esto ya lo hace otra funcion.
4  %Mensaje de petición de datos.
5  msg=['P',1,'E'];%'PlD';
6               %Este es el mensaje con:    CABECERA:    P
7               %                           Longit:      1
8               %                           Comando:     E
```

```
for(indice=1:1:numero_de_muestras)
    if indice~=1
        valor_anterior=valor;
    end
    fprintf(SerPIC,'%s',msg);
    pause(0.05);
    q=fread(SerPIC,27);
    for(indice_valor=1:1:12)
        valor(indice_valor)=q((indice_valor-1)*2+3+1)*255+q((indice_valor-1)*
            2+3+2) %El dato estaba dividido en dos bytes, de esta forma
            de vuelve a tener el valor original.
    end
    if indice==1
        valor_anterior=valor;
    end

    for(indice_celda=1:1:4)
        if indice~=1
            hold on
            for(indice_sensor=1:1:3)
                array_sensor(indice_sensor,1)=valor_anterior((indice_celda-1)*
                    3+indice_sensor);
                array_sensor(indice_sensor,2)=valor((indice_celda-1)*3+
                    indice_sensor);
                array_tiempo(1)=indice-1;
                array_tiempo(2)=indice;
            end
            subplot(2,2,indice_celda)
            plot(array_tiempo,array_sensor(1,:),'r',array_tiempo,array_sensor(2,:),
                'g',array_tiempo,array_sensor(3,:),'b')
            switch indice_celda
                case 1
                    title('Gráfica de la celda 0\_0');
                case 2
                    title('Gráfica de la celda 0\_1');
                case 3
                    title('Gráfica de la celda 1\_0');
                otherwise
                    title('Gráfica de la celda 1\_1');
            end
        else
            clf
            hold off
            %plot(indice,posicionx,'+',indice,posiciony,'-',
                indice,posicionz,'.')
        end
    end
end
end
```

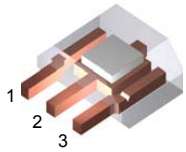
6.5. Hojas de características.

A1321/A1322/A1323

Ratiometric Linear Hall Effect Sensor for High-Temperature Operation

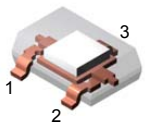
Package UA, 3-pin SIP

1. VCC
2. GND
3. VOUT



Package LH, 3-pin Surface Mount

1. VCC
2. VOUT
3. GND



ABSOLUTE MAXIMUM RATINGS

Supply Voltage, V_{CC} ..... **8 V***
Reverse-Battery Voltage, V_{RCC} ..... **–0.1 V**
Reverse-Output Voltage, V_{ROUT} ..... **–0.1 V**
Output Sink Current, I_{OUT} ..... **10 mA**
Operating Temperature
Ambient, T_A, Range E ..... **–40°C to 85°C**
Ambient, T_A, Range L ..... **–40°C to 150°C**
Maximum Junction, $T_{J(max)}$ ..... **165°C**
Storage Temperature, T_S ..... **–65°C to 170°C**

*Additional current draw may be observed at voltages above the minimum supply Zener clamp voltage, $V_{Z(min)}$, due to the Zener diode turning on.

Allegro MicroSystems, Inc.
www.allegromicro.com
Allegro MicroSystems, Inc. 115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

The A132X family of linear Hall-effect sensors are optimized, sensitive, and temperature-stable. These ratiometric Hall-effect sensors provide a voltage output that is proportional to the applied magnetic field. The A132X family has a quiescent output voltage that is 50% of the supply voltage and output sensitivity options of 2.5mV/G, 3.125mV/G, and 5mV/G. The features of this family of devices are ideal for use in the harsh environments found in automotive and industrial linear and rotary position sensing systems.

Each device has a BiCMOS monolithic circuit which integrates a Hall element, improved temperature-compensating circuitry to reduce the intrinsic sensitivity drift of the Hall element, a small-signal high-gain amplifier, and a rail-to-rail low-impedance output stage.

A proprietary dynamic offset cancellation technique, with an internal high-frequency clock, reduces the residual offset voltage normally caused by device overmolding, temperature dependencies, and thermal stress. The high frequency clock allows for a greater sampling rate, which results in higher accuracy and faster signal processing capability. This technique produces devices that have an extremely stable quiescent output voltage, are immune to mechanical stress, and have precise recoverability after temperature cycling. Having the Hall element and an amplifier on a single chip minimizes many problems normally associated with low-level analog signals.

Output precision is obtained by internal gain and offset trim adjustments made at end-of-line during the manufacturing process.

The A132X family is provided in a 3-pin single in-line package (UA) and a 3-pin surface mount package (LH).

Features and Benefits

- Temperature-stable quiescent output voltage
- Precise recoverability after temperature cycling
- Output voltage proportional to magnetic flux density
- Ratiometric rail-to-rail output
- Improved sensitivity
- 4.5 to 5.5 V operation
- Immunity to mechanical stress
- Solid-state reliability
- Robust EMC protection

Use the following complete part numbers when ordering:

Part Number	Package	Ambient	Sensitivity, Typ.
A1321EUA	SIP	–40°C to 85°C	5.000 mV/G
A1321ELH	Surface Mount		
A1321LUA	SIP	–40°C to 150°C	
A1321LLH	Surface Mount		
A1322EUA	SIP	–40°C to 85°C	3.125 mV/G
A1322ELH	Surface Mount		
A1322LUA	SIP	–40°C to 150°C	
A1322LLH	Surface Mount		
A1323EUA	SIP	–40°C to 85°C	2.500 mV/G
A1323ELH	Surface Mount		
A1323LUA	SIP	–40°C to 150°C	
A1323LLH	Surface Mount		

A1321/A1322/A1323
Ratiometric Linear Hall Effect Sensor for High-Temperature Operation

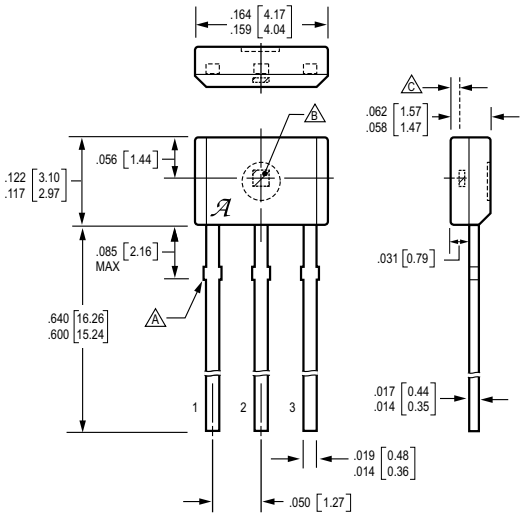
MAGNETIC CHARACTERISTICS[1,2] over operating temperature range, T_A; V_{CC} = 5 V, I_{out} = –1 mA; unless otherwise noted

Characteristics	Symbol	Test Condition	Min	Typ[3]	Max	Units[4]
Sensitivity[5]	Sens	A1321; T_A = 25ºC	4.750	5.000	5.250	mV/G
		A1322; T_A = 25ºC	2.969	3.125	3.281	mV/G
		A1323; T_A = 25ºC	2.375	2.500	2.625	mV/G
Delta $V_{out(q)}$ as a function of temperature	$V_{out(q)(\Delta T)}$	Defined in terms of magnetic flux density, B	–	–	±10	G
Ratiometry, $V_{out(q)}$	$V_{out(q)(\Delta V)}$		–	–	±1.5	%
Ratiometry, Sens	$\Delta Sens_{(\Delta V)}$		–	–	±1.5	%
Positive Linearity	Lin+		–	–	±1.5	%
Negative Linearity	Lin–		–	–	±1.5	%
Symmetry	Sym		–	–	±1.5	%
UA Package						
Delta Sens at T_A = max[5]	$\Delta Sens_{(TAmax)}$	From hot to room temperature	–2.5	–	7.5	%
Delta Sens at T_A = min[5]	$\Delta Sens_{(TAmin)}$	From cold to room temperature	–6	–	4	%
Sensitivity Drift[6]	$Sens_{Drift}$	T_A = 25°C; after temperature cycling and over time	–	1	2	%
LH Package						
Delta Sens at T_A = max[5]	$\Delta Sens_{(TAmax)}$	From hot to room temperature	–5	–	5	%
Delta Sens at T_A = min[5]	$\Delta Sens_{(TAmin)}$	From cold to room temperature	–3.5	–	8.5	%
Sensitivity Drift[6]	$Sens_{Drift}$	T_A = 25°C; after temperature cycling and over time	–	0.328	2	%

[1] Additional information on chracteristics is provided in the section Characteristics Definitions, on the next page.
[2] Negative current is defined as conventional current coming out of (sourced from) the specified device terminal.
[3] Typical data is at T_A = 25°C, except for ΔSens, and at *x.x* Sens. Typical data are for initial design estimations only, and assume optimum manufacturing and application conditions. Performance may vary for individual units, within the specified maximum and minimum limits. In addition, the typical values vary with gain.
[4] 10 G = 1 millitesla.
[5] After 150ºC pre-bake and factory programming.
[6] Sensitivity drift is the amount of recovery with time.

A1321/A1322/A1323
Ratiometric Linear Hall Effect Sensor for High-Temperature Operation

Package UA, 3-Pin; (TO-92)



Dimensions in inches
Metric dimensions (mm) in brackets, for reference only
Dimensions without tolerances are basic
A Dambar removal protrusion
B Hall element
C Active Area Depth .018 [0.46]

LM124
LM224 - LM324

LOW POWER QUAD OPERATIONAL AMPLIFIERS

- WIDE GAIN BANDWIDTH : 1.3MHz
- INPUT COMMON-MODE VOLTAGE RANGE INCLUDES GROUND
- LARGE VOLTAGE GAIN : 100dB
- VERY LOW SUPPLY CURRENT/AMPLI : 375µA
- LOW INPUT BIAS CURRENT : 20nA
- LOW INPUT OFFSET VOLTAGE : 5mV max. (for more accurate applications, use the equivalent parts LM124A-LM224A-LM324A which feature 3mV max.)
- LOW INPUT OFFSET CURRENT : 2nA
- WIDE POWER SUPPLY RANGE : SINGLE SUPPLY : +3V TO +30V DUAL SUPPLIES : ±1.5V TO ±15V

N
DIP14
(Plastic Package**)**

D
SO14
(Plastic Micropackage**)**

P
TSSOP14
(Thin Shrink Small Outline Package)

DESCRIPTION

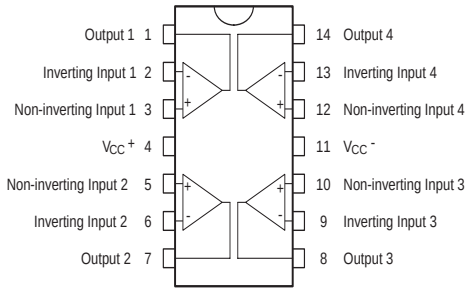
These circuits consist of four independent, high gain, internally frequency compensated operational amplifiers. They operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

ORDER CODE

Part Number	Temperature Range	Package		
		N	D	P
LM124	-55°C, +125°C	•	•	•
LM224	-40°C, +105°C	•	•	•
LM324	0°C, +70°C	•	•	•
Example : LM224N				

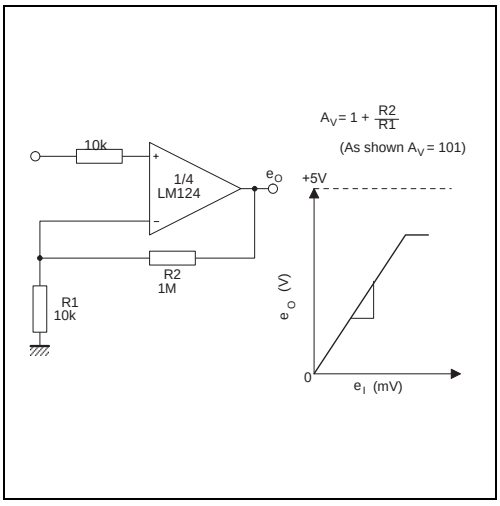
N = Dual in Line Package (DIP)
D = Small Outline Package (SO) - also available in Tape & Reel (DT)
P = Thin Shrink Small Outline Package (TSSOP) - only available in Tape &Reel (PT)

PIN CONNECTIONS (top view)

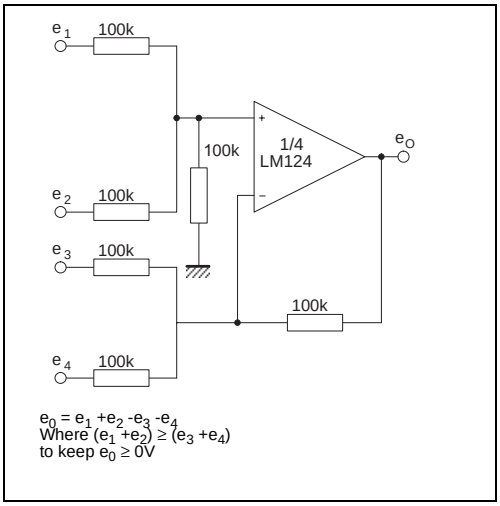


TYPICAL SINGLE - SUPPLY APPLICATIONS

NON-INVERTING DC GAIN

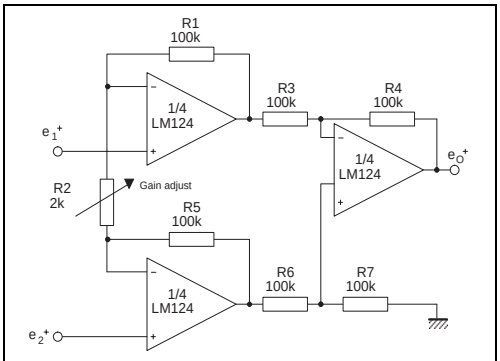


DC SUMMING AMPLIFIER



$e_0 = e_1 + e_2 - e_3 - e_4$
Where $(e_1 + e_2) \geq (e_3 + e_4)$
to keep $e_0 \geq 0V$

HIGH INPUT Z ADJUSTABLE GAIN DC INSTRUMENTATION AMPLIFIER

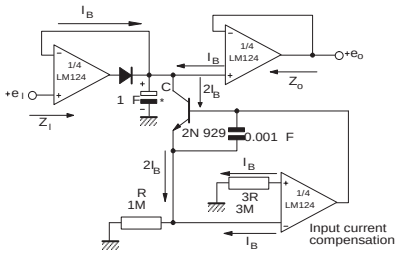


if R1 = R5 and R3 = R4 = R6 = R7

$$e_0 = \left[1 + \frac{2R_1}{R_2}\right](e_2 - e_1)$$

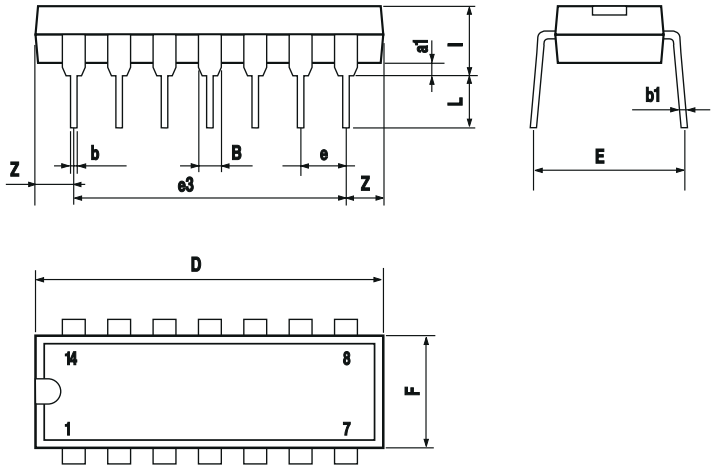
As shown $e_0 = 101\ (e_2 - e_1)$.

LOW DRIFT PEAK DETECTOR



* Polycarbonate or polyethylene

PACKAGE MECHANICAL DATA
14 PINS - PLASTIC DIP



Dimensions	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
a1	0.51			0.020		
B	1.39		1.65	0.055		0.065
b		0.5			0.020	
b1		0.25			0.010	
D			20			0.787
E		8.5			0.335	
e		2.54			0.100	
e3		15.24			0.600	
F			7.1			0.280
i			5.1			0.201
L		3.3			0.130	
Z	1.27		2.54	0.050		0.100

TEXAS INSTRUMENTS

Data sheet acquired from Harris Semiconductor
SCHS122A

CD54HC4051, CD74HC4051, CD74HCT4051, CD74HC4052, CD74HCT4052, CD74HC4053, CD74HCT4053

High Speed CMOS Logic Analog Multiplexers/Demultiplexers

November 1997 - Revised April 1999

Features

- **Wide Analog Input Voltage Range . . . . . ±5V Max**
- **Low "On" Resistance**
  - **70Ω Typical (V_{CC} - V_{EE} = 4.5V)**
  - **40Ω Typical (V_{CC} - V_{EE} = 9V)**
- **Low Crosstalk between Switches**
- **Fast Switching and Propagation Speeds**
- **"Break-Before-Make" Switching**
- **Wide Operating Temperature Range . . -55°C to 125°C**
- **CD54HC/CD74HC Types**
  - **Operation Control Voltage . . . . . 2V to 6V**
  - **Switch Voltage . . . . . 0V to 10V**
  - **High Noise Immunity . . . N_{IL} = 30%, N_{IH} = 30% of V_{CC}, V_{CC} = 5V**
- **CD54HCT/CD74HCT Types**
  - **Operation Control Voltage . . . . . 4.5V to 5.5V**
  - **Switch Voltage . . . . . 0V to 10V**
  - **Direct LSTTL Input Logic Compatibility . . . V_{IL} = 0.8V Max, V_{IH} = 2V Min**
  - **CMOS Input Compatibility . . . . . $I_I \leq 1\mu A$ at V_{OL}, V_{OH}**

Description

These devices are digitally controlled analog switches which utilize silicon gate CMOS technology to achieve operating speeds similar to LSTTL with the low power consumption of standard CMOS integrated circuits.

These analog multiplexers/demultiplexers control analog voltages that may vary across the voltage supply range (i.e. V_{CC} to V_{EE}). They are bidirectional switches thus allowing any analog input to be used as an output and visa-versa. The switches have low "on" resistance and low "off" leakages. In addition, all three devices have an enable control which, when high, disables all switches to their "off" state.

Ordering Information

PART NUMBER	TEMP. RANGE (°C)	PACKAGE	PKG. NO.
CD54HC4051F	-55 to 125	16 Ld CERDIP	F16.3
CD74HC4051E	-55 to 125	16 Ld PDIP	E16.3
CD74HC4052E	-55 to 125	16 Ld PDIP	E16.3
CD74HC4053E	-55 to 125	16 Ld PDIP	E16.3
CD74HCT4051E	-55 to 125	16 Ld PDIP	E16.3
CD74HCT4052E	-55 to 125	16 Ld PDIP	E16.3
CD74HCT4053E	-55 to 125	16 Ld PDIP	E16.3
CD74HC4051M	-55 to 125	16 Ld SOIC	M16.15
CD74HC4052M	-55 to 125	16 Ld SOIC	M16.15
CD74HC4053M	-55 to 125	16 Ld SOIC	M16.15
CD74HCT4051M	-55 to 125	16 Ld SOIC	M16.15
CD74HCT4052M	-55 to 125	16 Ld SOIC	M16.15
CD74HCT4053M	-55 to 125	16 Ld SOIC	M16.15
CD74HCT4053PW	-55 to 125	16 Ld TSSOP	
CD74HCT4052SM	-55 to 125	16 Ld SSOP	M16.15A

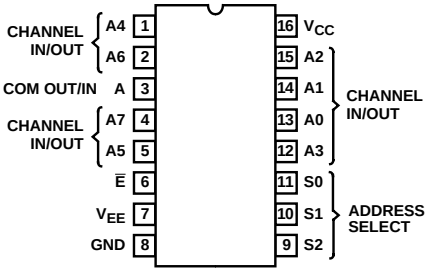
NOTES:

1. When ordering, use the entire part number. Add the suffix 96 to obtain the variant in the tape and reel. For the TSSOP package only, add the suffix R to obtain the variant in the tape and reel.
2. Wafer or die is available which meets all electrical specifications. Please contact your local sales office or Harris customer service for ordering information.

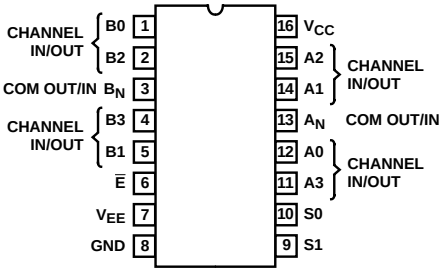
CD54HC4051, CD74HC4051, 52, 53; CD74HCT4051, 52, 53

Pinouts

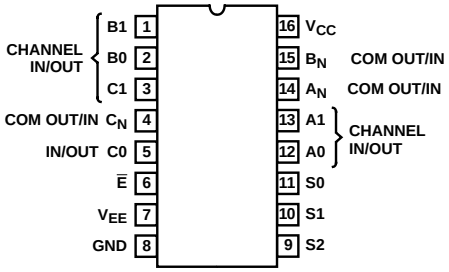
CD54HC4051
CD74HC4051, CD74HCT4051
(CERDIP, PDIP, SOIC)
TOP VIEW



CD74HC4052, CD74HCT4052
(PDIP, SOIC)
TOP VIEW

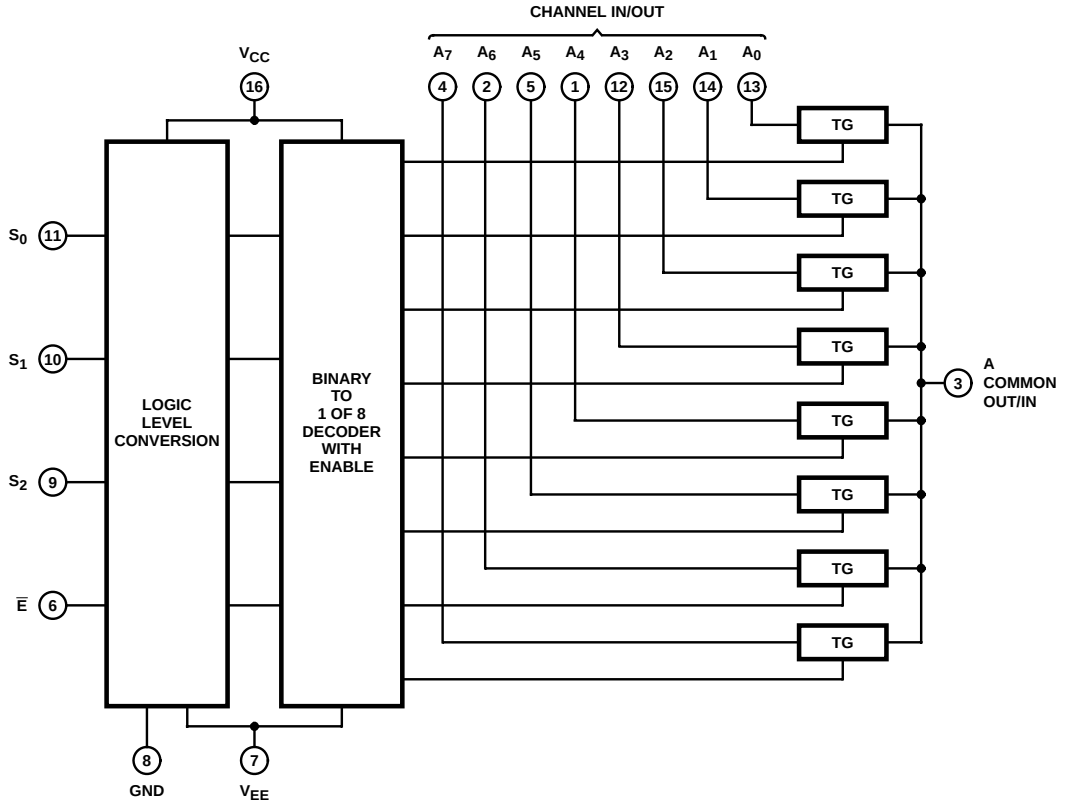


CD74HC4053, CD74HCT4053
(PDIP, SOIC, TSSOP)
TOP VIEW



CD54HC4051, CD74HC4051, 52, 53; CD74HCT4051, 52, 53

Functional Diagram of HC/HCT4051



TRUTH TABLE
CD54/74HC/HCT4051

INPUT STATES				"ON" CHANNELS
ENABLE	S_2	S_1	S_0	
L	L	L	L	A0
L	L	L	H	A1
L	L	H	L	A2
L	L	H	H	A3
L	H	L	L	A4
L	H	L	H	A5
L	H	H	L	A6
L	H	H	H	A7
H	X	X	X	None

X = Don't care

CD54HC4051, CD74HC4051, 52, 53; CD74HCT4051, 52, 53

Absolute Maximum Ratings (Note 3)

DC Supply Voltage, V_{CC} - V_{EE} . . . . . -0.5V to 10.5V
DC Supply Voltage, V_{CC} . . . . . -0.5V to +7V
DC Supply Voltage, V_{EE} . . . . . +0.5V to -7V
DC Input Diode Current, I_{IK}
For V_I < -0.5V or V_I > V_{CC} + 0.5V . . . . . ±20mA
DC Switch Diode Current, I_{OK}
For V_I < V_{EE} -0.5V or V_I > V_{CC} + 0.5V . . . . . ±20mA
DC Switch Current, (Note 2)
For V_I > V_{EE} -0.5V or V_I < V_{CC} + 0.5V . . . . . ±25mA
DC V_{CC} or Ground Current, I_{CC} . . . . . ±50mA
DC V_{EE} Current, I_{EE} . . . . . -20mA

Thermal Information

Thermal Resistance (Typical, Note 4)	θ_{JA} ($^oC/W$)	θ_{JC} ($^oC/W$)
PDIP Package	90	N/A
SOIC Package	160	N/A
CERDIP Package	130	55
TSSOP Package	149	35

Maximum Junction Temperature . . . . . 150^oC
Maximum Storage Temperature Range . . . . . -65^oC to 150^oC
Maximum Lead Temperature (Soldering 10s) . . . . . 300^oC

Recommended Operating Conditions

For maximum reliability, nominal operating conditions should be selected so that operation is always within the following ranges

PARAMETER	MIN	MAX	UNITS
Supply Voltage Range (For T_A = Full Package Temperature Range), V_{CC} (Note 5)			
CD54/74HC Types	2	6	V
CD54/74HCT Types	4.5	5.5	V
Supply Voltage Range (For T_A = Full Package Temperature Range), V_{CC} - V_{EE}			
CD54/74HC Types, CD54/74HCT Types (See Figure 1)	2	10	V
Supply Voltage Range (For T_A = Full Package Temperature Range), V_{EE} (Note 5)			
CD54/74HC Types, CD54/74HCT Types (See Figure 2)	0	-6	V
DC Input Control Voltage, V_I	GND	V_{CC}	V
Analog Switch I/O Voltage, V_{IS}	V_{EE}	V_{CC}	V
Operating Temperature, T_A	-55	125	^oC
Input Rise and Fall Times, t_r, t_f			
2V	0	1000	ns
4.5V	0	500	ns
6V	0	400	ns

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTES:

3. All voltages referenced to GND unless otherwise specified.
4. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.
5. In certain applications, the external load resistor current may include both V_{CC} and signal line components. To avoid drawing V_{CC} current when switch current flows into the transmission gate inputs, the voltage drop across the bidirectional switch must not exceed 0.6V (calculated from r_{ON} values shown in Electrical Specifications table). No V_{CC} current will flow through R_L if the switch current flows into terminal 3 on the HC/HCT4051; terminals 3 and 13 on the HC/HCT4052; terminals 4, 14 and 15 on the HC/HCT4053.

Recommended Operating Area as a Function of Supply Voltages

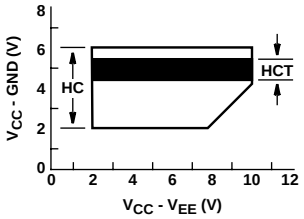


FIGURE 1.

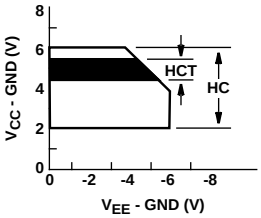


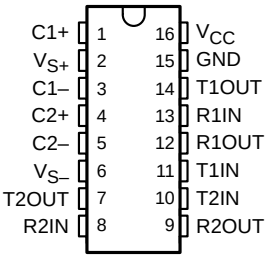
FIGURE 2.

MAX232, MAX232I
DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047I – FEBRUARY 1989 – REVISED OCTOBER 2002

- **Meet or Exceed TIA/EIA-232-F and ITU Recommendation V.28**
- **Operate With Single 5-V Power Supply**
- **Operate Up to 120 kbit/s**
- **Two Drivers and Two Receivers**
- **±30-V Input Levels**
- **Low Supply Current . . . 8 mA Typical**
- **Designed to be Interchangeable With Maxim MAX232**
- **ESD Protection Exceeds JESD 22 – 2000-V Human-Body Model (A114-A)**
- **Applications**
  - **TIA/EIA-232-F**
  - **Battery-Powered Systems**
  - **Terminals**
  - **Modems**
  - **Computers**

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)



description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	PDIP (N)	Tube	MAX232N	MAX232N
	SOIC (D)	Tube	MAX232D	MAX232
		Tape and reel	MAX232DR	
	SOIC (DW)	Tube	MAX232DW	MAX232
		Tape and reel	MAX232DWR	
	SOP (NS)	Tape and reel	MAX232NSR	MAX232
–40°C to 85°C	PDIP (N)	Tube	MAX232IN	MAX232IN
	SOIC (D)	Tube	MAX232ID	MAX232I
		Tape and reel	MAX232IDR	
	SOIC (DW)	Tube	MAX232IDW	MAX232I
		Tape and reel	MAX232IDWR	

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

MAX232, MAX232I
DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047I – FEBRUARY 1989 – REVISED OCTOBER 2002

Function Tables

EACH DRIVER

INPUT TIN	OUTPUT TOUT
L	H
H	L

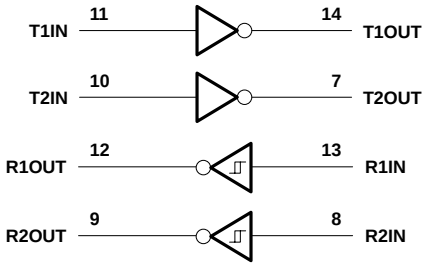
H = high level, L = low level

EACH RECEIVER

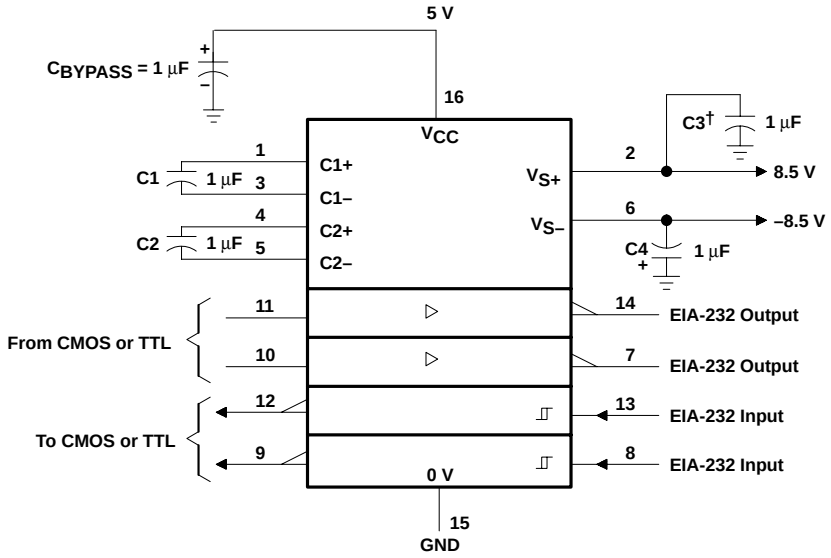
INPUT RIN	OUTPUT ROUT
L	H
H	L

H = high level, L = low level

logic diagram (positive logic)



APPLICATION INFORMATION



† C3 can be connected to V_{CC} or GND.

Figure 4. Typical Operating Circuit

D-Subminiature Shielded I/O PCB Receptacle

48201

Female, Right Angle Through Hole

Features and Benefits
- Beveled metal pins provide positive PCB retention force
- Metal shell provides EMI/RFI shielding
- Intermateable with industry standard receptacles

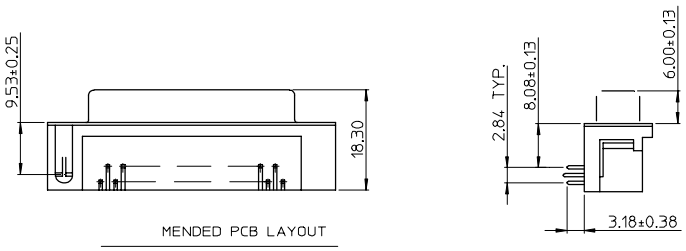
Reference Information
Product Specification: PS-48201-001
Packaging: Tray
Mates With: 48202
Designed In: Millimeters

Electrical
Voltage: 300V DC
Current: 1.0A
Contact Resistance: 20 milliohms max.
Dielectric Withstanding Voltage: 500V AC
Insulation Resistance: 1000 Megohms min.

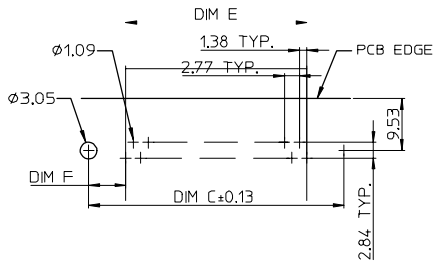
Mechanical
Contact Retention Force: 22.24N
Durability: Gold Flash—600 cycles
30µ" Gold—1000 cycles

Physical
Housing: Glass-filled polyester, UL 94V-0
Contact: Phosphor Bronze
Plating: Solder Tail Area—Tin
Underplating: 50µ" Nickel
PCB Thickness: 1.60mm (.062")
Operating Temperature: -55 to +85˚C

Circuits	Order No.	Lock to Mating Part	Plating Contact	Lead-free
9	48201-6041	4-40 Round Inner Thread Screws Included	30µ" Gold	Yes
	48201-6046	4-40 Hex Internal Thread Screws Included		
	48201-6043	4-40 Hex Internal Thread Screws Assemble		
	48201-6062	Screw Hole, Threaded	Gold Flash	
15	48201-6158	M3 Inner Thread Screws Included	15µ" Gold	
	48201-6157	M2.6 Inner Thread Screws Included		
	48201-6156	4-40 Hex Internal Thread Screws Included		
	48201-6155	M3 Inner Thread Screws Assemble		
	48201-6154	M2.6 Inner Thread Screws Assemble		
	48201-6153	4-40 Hex Internal Thread Screws Assemble		
	48201-6152	Screw Hole, Threaded		
	48201-6166	4-40 Hex Internal Thread Screws Included	Gold Flash	
25	48201-6367	M2.6 Inner Thread Screws Included	Gold Flash	
	48201-6366	4-40 Hex Internal Thread Screws Included		
	48201-6365	M3 Inner Thread Screws Assemble		
	48201-6364	M2.6 Inner Thread Screws Assemble		
	48201-6363	4-40 Hex Internal Thread Screws Assemble		
	48201-6362	Screw Hole, Thread		
	48201-6368	M3 Inner Thread Screws Included		

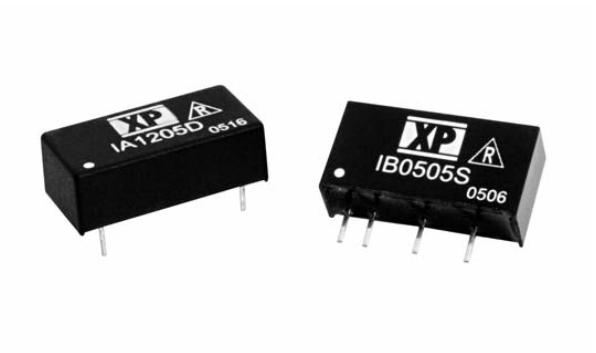


MENDED PCB LAYOUT



1 Watt
IA/IB Series

xppower.com



- Single & Dual Output
- SIP or DIP Package
- Industry Standard Pinout
- 1000 VDC Isolation
- Short Circuit Protection
- -40 °C to +85 °C Operation
- MTBF >2 MHrs

DC-DC

Specification

Input

Input Voltage Range	• Nominal ±10%
Input Reflected Ripple	• 20 mA rms
Input Reverse Voltage Protection	• None

Output

Output Voltage	• See table
Minimum Load	• None[7]
Line Regulation	• 1.2%/1% Δ Vin
Load Regulation	• ±10% 20-100% load change (3.3 V models ±20%)
Setpoint Accuracy	• ±3%
Ripple & Noise	• 60 mV pk-pk 20 MHz bandwidth
Temperature Coefficient	• 0.02%/°C
Short Circuit Protection	• 1 s max
Maximum Capacitive Load	• 100 µF

General

Efficiency	• 75% typical
Isolation Voltage	• 1000 VDC minimum
Isolation Resistance	• $10^9\,\Omega$
Isolation Capacitance	• 60 pF typical
Switching Frequency	• Variable
MTBF	• >2 MHrs to MIL-STD-217F

Environmental

Operating Temperature	• -40 °C to +85 °C
Storage Temperature	• -40 °C to +125 °C
Case Temperature	• 100 °C max
Cooling	• Convection-cooled

Notes

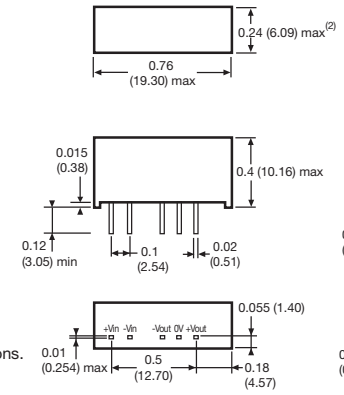
1. Replace 'S' in model number with 'D' for DIP package.
2. SIP 48 Vin models, dimension is 0.28 (7.20) max.
3. DIP 48 Vin models, dimension is 0.27 (6.88) max.
4. Outputs power-trade.
5. All dimensions in inches (mm).
6. For 48 V models a 10 µF capacitor is required between +Vin and -Vin pins.
7. Operation at no load will not damage unit but it may not meet all specifications.
8. IB Series has no 0V pin. Use -Vout and +Vout pins for output.

Input Voltage	Output Voltage	Output Current[4]	IA Model Number[1]	Output Voltage	Output Current	IB Model Number
3.3 VDC	±5.0 V	±100 mA	IA0305S			
5 VDC	±3.3 V	±151 mA	IA0503S	3.3 V	303 mA	IB0503S
	±5.0 V	±100 mA	IA0505S†	5.0 V	200 mA	IB0505S
	±9.0 V	±55 mA	IA0509S†	9.0 V	111 mA	IB0509S
	±12.0 V	±42 mA	IA0512S†	12.0 V	84 mA	IB0512S
	±15.0 V	±33 mA	IA0515S†	15.0 V	66 mA	IB0515S
	±24.0 V	±21 mA	IA0524S	24.0 V	42 mA	IB0524S
12 VDC	±3.3 V	±151 mA	IA1203S	3.3 V	303 mA	IB1203S
	±5.0 V	±100 mA	IA1205S†	5.0 V	200 mA	IB1205S
	±9.0 V	±55 mA	IA1209S†	9.0 V	111 mA	IB1209S
	±12.0 V	±42 mA	IA1212S†	12.0 V	84 mA	IB1212S
	±15.0 V	±33 mA	IA1215S†	15.0 V	66 mA	IB1215S
	±24.0 V	±21 mA	IA1224S	24.0 V	42 mA	IB1224S
24 VDC	±3.3 V	±151 mA	IA2403S	3.3 V	303 mA	IB2403S
	±5.0 V	±100 mA	IA2405S†	5.0 V	200 mA	IB2405S
	±9.0 V	±55 mA	IA2409S	9.0 V	111 mA	IB2409S
	±12.0 V	±42 mA	IA2412S†	12.0 V	84 mA	IB2412S
	±15.0 V	±33 mA	IA2415S†	15.0 V	66 mA	IB2415S
	±24.0 V	±21 mA	IA2424S	24.0 V	42 mA	IB2424S
48 VDC	±3.3 V	±151 mA	IA4803S	3.3 V	303 mA	IB4803S
	±5.0 V	±100 mA	IA4805S†	5.0 V	200 mA	IB4805S
	±9.0 V	±55 mA	IA4809S	9.0 V	111 mA	IB4809S
	±12.0 V	±42 mA	IA4812S†	12.0 V	84 mA	IB4812S
	±15.0 V	±33 mA	IA4815S	15.0 V	66 mA	IB4815S
	±24.0 V	±21 mA	IA4824S	24.0V	42 mA	IB4824S

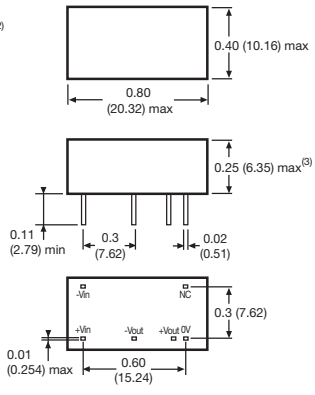
† Available from Farnell. See pages 204-206.

Mechanical Details

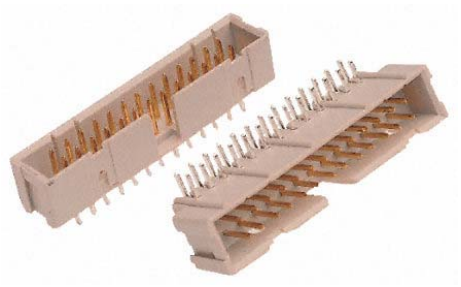
SIP Package



DIP Package



Term macho IDC recto encap 20vías. 33.3mm



Atributo	Valor
Tipo	Cable a placa
Género	Conector macho para PCB
Montaje	Agujero pasante
Orientación	Recto
Número de contactos	20
Método de terminación	Conector IDC
Temperaturas de funcionamientode	- 45 a 105 °C
Long.	33,3mm
Chapado de los contactos	Oro

Detalles.

Conectores macho recubiertos para placa

Molduras de poliéster con homologación UL94-V con contactos de aleación de cobre chapados en oro
Conexiones de clavija por soldadura en rejilla de 2,54 mm adecuadas para PCB estándar de 1,6 mm de grosor
Conectores macho compatibles con conectores hembra polarizados por resalte y universales tipo BT224

Descripción de la gama

El sistema de conectores Speedbloc® comprende tres gamas de conectores por desplazamiento del aislante; conectores hembra aéreos con conectores macho de montaje en PCB, conectores de borde y DIL, conectores de transición, conectores macho y hembra D-Sub. Se utilizan junto a Speedbloc®. El cable plano IDC permite el montaje rápido, simple y económico, al tiempo que posibilita al usuario añadir o ampliar los sistemas de cable plano existentes a través de herramientas IDC universales. Para obtener más información, consulte cada uno de los productos y las tablas de datos adicionales. Como ayuda en la fabricación de PCB, utilice transferencias resistentes a ataques químicos de un solo símbolo. Consulte la sección Diseñode PCB y Accesorios

Descripción del Grupo de Productos

El sistema de conectores Speedbloc® comprende gamas de conectores por desplazamiento del aislante; conectores hembra aéreos con conectores macho de montaje en PCB, conectores de borde y DIL, conectores de transición, conectores macho y hembra D-Sub. Se utilizan junto a Speedbloc®. El cable plano IDC permite el montaje rápido, simple y económico, al tiempo que posibilita al usuario añadir o ampliar los sistemas de cable plano existentes a través de herramientas IDC universales. Para obtener información adicional, consulte cada producto. Como ayuda en la fabricación de PCB, utilice transferencias resistentes a ataques químicos de un solo símbolo. Consulte la sección Diseñode PCB y Accesorios

MICROCHIP

PIC32MX3XX/4XX

High-Performance 80 MHz MIPS-Based 32-bit Flash Microcontroller 64/100-Pin General Purpose and USB

High-Performance 32-bit RISC CPU:

- MIPS32® M4K™ 32-bit Core with 5-Stage Pipeline
- 80 MHz Maximum Frequency
- 1.56 DMIPS/MHz (Dhrystone 2.1) Performance at 0 Wait State Flash Access
- Single-Cycle Multiply and High-Performance Divide Unit
- MIPS16e™ Mode for Up to 40% Smaller Code Size
- User and Kernel Modes to Enable Robust Embedded System
- Two Sets of 32 Core Register Files (32-bit) to Reduce Interrupt Latency
- Prefetch Cache Module to Speed Execution from Flash

Microcontroller Features:

- Operating Voltage Range of 2.3V to 3.6V
- 32K to 512K Flash Memory (plus an additional 12KB of Boot Flash)
- 8K to 32K SRAM Memory
- Pin-Compatible with Most PIC24/dsPIC® Devices
- Multiple Power Management Modes
- Multiple Interrupt Vectors with Individually Programmable Priority
- Fail-Safe Clock Monitor Mode
- Configurable Watchdog Timer with On-Chip Low-Power RC Oscillator for Reliable Operation

Peripheral Features:

- Atomic SET, CLEAR and INVERT Operation on Select Peripheral Registers
- Up to 4-Channel Hardware DMA Controller with Automatic Data Size Detection
- USB 2.0 Compliant Full Speed Device and On-The-Go (OTG) Controller
- USB has a Dedicated DMA Channel
- 10 MHz to 40 MHz Crystal Oscillator
- Internal 8 MHz and 32 kHz Oscillators
- Separate PLLs for CPU and USB Clocks
- Two I^2C™ Modules
- Two UART Modules with:
  - RS-232, RS-485 and LIN 1.2 support
  - IrDA® with On-Chip Hardware Encoder and Decoder
- Parallel Master and Slave Port (PMP/PSP) with 8-bit and 16-bit Data and Up to 16 Address Lines
- Hardware Real-Time Clock/Calendar (RTCC)
- Five 16-bit Timers/Counters (two 16-bit pairs combine to create two 32-bit timers)
- Five Capture Inputs
- Five Compare/PWM Outputs
- Five External Interrupt Pins
- High-Speed I/O Pins Capable of Toggling at Up to 80 MHz
- High-Current Sink/Source (18 mA/18 mA) on All I/O Pins
- Configurable Open-Drain Output on Digital I/O Pins

Debug Features:

- Two Programming and Debugging Interfaces:
  - 2-Wire Interface with Unintrusive Access and Real-time Data Exchange with Application
  - 4-wire MIPS Standard Enhanced JTAG interface
- Unintrusive Hardware-Based Instruction Trace
- IEEE Std 1149.2 Compatible (JTAG) Boundary Scan

Analog Features:

- Up to 16-Channel 10-bit Analog-to-Digital Converter:
  - 500 ksps Conversion Rate
  - Conversion Available During Sleep, Idle
- Two Analog Comparators
- 5.5V Tolerant Input Pins (digital pins only)

PIC32MX3XX/4XX

Pin Diagram (100-Pin USB)

100-Pin TQFP (USB)

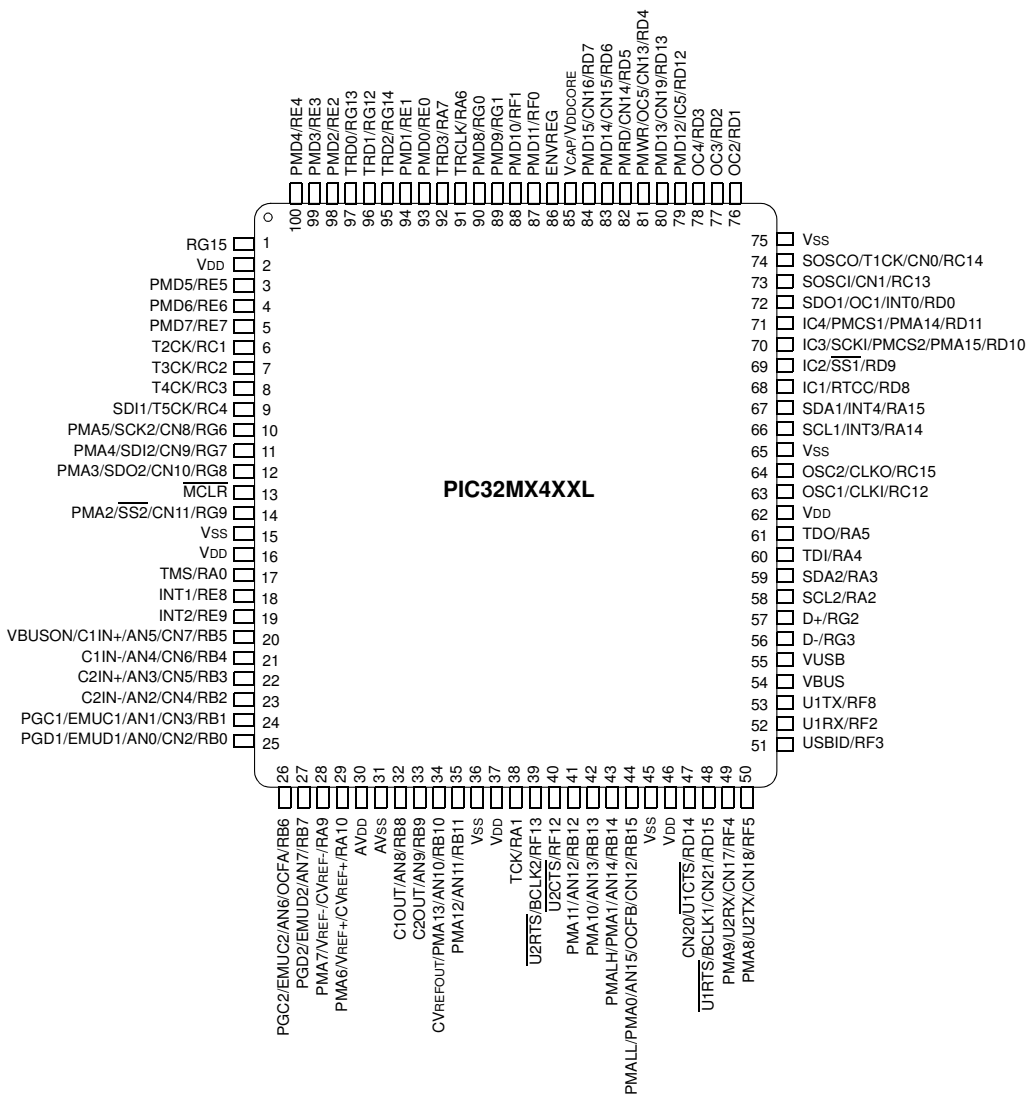
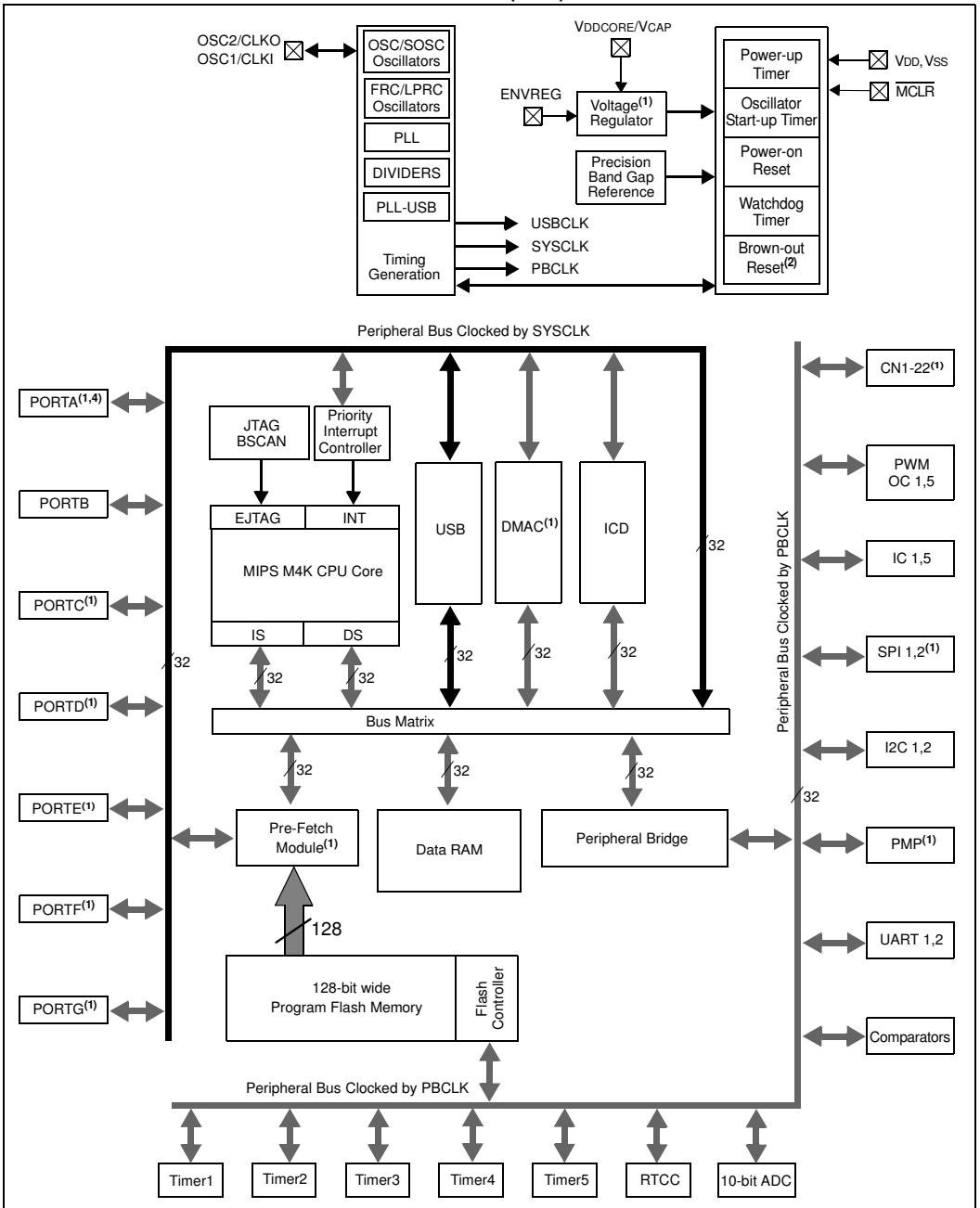


TABLE 1-3: DEVICE FEATURES FOR THE PIC32MX4XXFXXX USB FAMILY

Features	PIC32MX420F032H	PIC32MX440F128H	PIC32MX440F256H	PIC32MX440F512H	PIC32MX440F128L	PIC32MX460F256L	PIC32MX460F512L
Operating Frequency	DC – 40 MHz	DC – 80 MHz					
Program Memory (Bytes)	32K	128K	256K	512K	128K	256K	512K
Data Memory (Bytes)	8K	32K	32K	32K	32K	32K	32K
Interrupt Sources / Vectors	95 / 63						
I/O Ports	Ports B, C, D, E, F, G				Ports A, B, C, D, E, F, G		
Total I/O Pins	51				83		
DMA Channels	0 + 2 USB	4 + 2 USB					
Timers:							
Total number (16-bit)	5						
32-bit (from paired 16-bit timers)	2						
32-bit Core Timer	1						
Input Capture Channels	5						
Output Compare/PWM Channels	5						
Input Change Interrupt Notification	19				22		
Serial Communications:							
Enhanced UART	2						
SPI (3-wire/4-wire)	1				2		
I^2C™	2						
Parallel Communications (PMP/PSP)	Yes, 8-bit only				Yes, 8-bit/16-bit		
JTAG Boundary Scan	Yes						
JTAG Debug and Program	Yes						
ICSP 2-wire Debug and Program	Yes						
Instruction Trace	No						Yes
Hardware Break Points:	6 Instruction, 2 Data						
10-bit Analog-to-Digital Module (input channels)	16						
Analog Comparators	2						
Internal LDO	Yes						
Resets (and Delays)	POR, BOR, $\overline{\text{MCLR}}$, WDT, SWR (Software Reset), CM (Configuration Bit Mismatch) (PWRT, OST, PLL Lock)						
Instruction Support	MIPS32 Enhanced Architecture (Release 2) MIPS16e™ Code Compression						
Packages	64-pin TQFP				100-pin TQFP		

FIGURE 1-2: PIC32MX4XX BLOCK DIAGRAM (USB)



Note 1: Not all pins or features are implemented on all device pinout configurations. See Table 1-5 for I/O port pin descriptions.

2: Some features are not available on certain device variants.

3: BOR functionality is provided when the on-board voltage regulator is enabled.

4: PORTA is not present on 64 pin devices

MICROCHIP — A Leading Provider of Microcontrollers & Analog Semiconductors

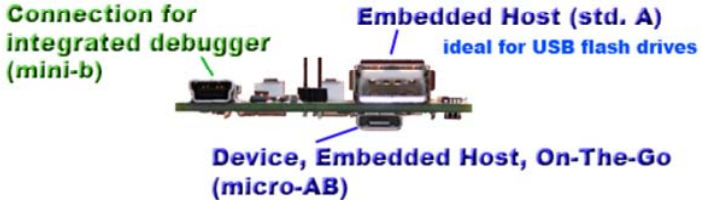
PIC32 USB Starter Board

USB Starter Board

(Part Number: DM320003)

The PIC32 USB Starter Board provides the easiest and lowest cost method to experience the USB On-The-Go family of PIC32 microcontrollers. Users can develop USB embedded host, device, dual-role, or On-The-Go applications by combining this board with Microchip's free USB software (On-The-Go support provided in a future software release).

The USB Starter Board has the same form factor and expansion connector as the PIC32 Starter Kit (#DM320001). New in MPLAB 8.10 is support for the 32-bit edition of Microsoft Windows Vista, with support for the 64-bit edition currently in development for a future release. Windows XP and Windows 2000 are also supported.



DEVELOP. PROGRAM. RUN.

Write Code
- Free MPLAB IDE v8.10 or newer
- Free MPLAB C32 C Compiler with no code size limit**
- Example project files for HID-class device mode, MSD-class host mode (see note below), CDC-class device mode. Additional projects will be available in the future.

Program Memory
- Integrated programmer - all you need is the included USB Cable
- Program and verify a full 512k image in under 9 seconds

Debug
- Integrated debugger - all you need is the included USB cable
- Full C source based debugging in MPLAB IDE

Execute Code
- PIC32 running at 80 MHz with 512K Flash, 32K RAM, 4 ch. DMA + 2 ch. DMA dedicated to USB
- USB powered board
- USB connectors, user switches, and LEDs

Customize
- Expansion connector enables addition of Microchip's PIC32 expansion boards or create your own
- Majority of MCU signals are present on the connector [Hirose: FX10A-120P/12-SV1(71)]

Example Software:
Example Projects for Micrium µC/OS-II using the PIC32 USB Starter Board, MPLAB IDE, and MPLAB C Compiler for PIC32

Box Contents:
- PIC32 USB Starter Board
- Standard A to mini B cable for debugger
- Standard A to micro B cable for USB application development
- Quick start card directing users to web based instructions for software download and installation. Note that no software is provided in the box.

The software installation and setup instructions are located here.

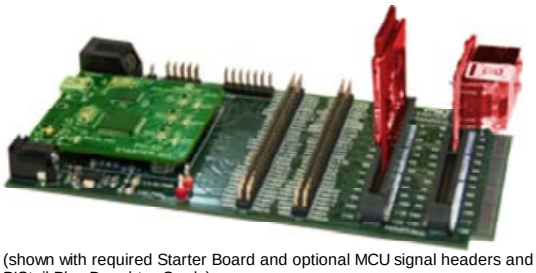
**Starting with version 1.03 the MPLAB C Compiler for PIC32 Student Edition has no code size limit and full functionality for 60 days. After 60 days, optimization levels -O2, -O3, -Os are not available.

Tutorial Videos:
Installing PIC32 Starter Kit on Windows Vista 32-bit edition (5 mins)
PIC32 Development Board Options (3 mins)

Note: The PIC32 USB Starter Board runs on Microsoft Windows XP, Windows 2000, and Windows Vista (32-bit) support when used with MPLAB IDE versions 8.10 and newer. (view an installation tutorial). Support for Windows Vista 64-bit is planned.

MICROCHIP
A Leading Provider of Microcontrollers & Analog Semiconductors

PIC32 I/O Expansion Board

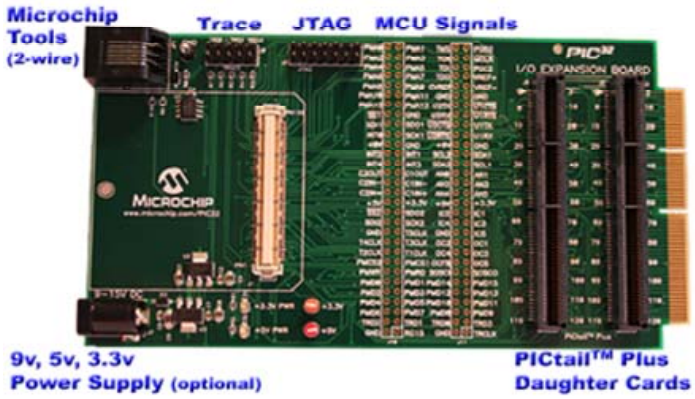


I/O Expansion Board

(Part Number: DM320002)

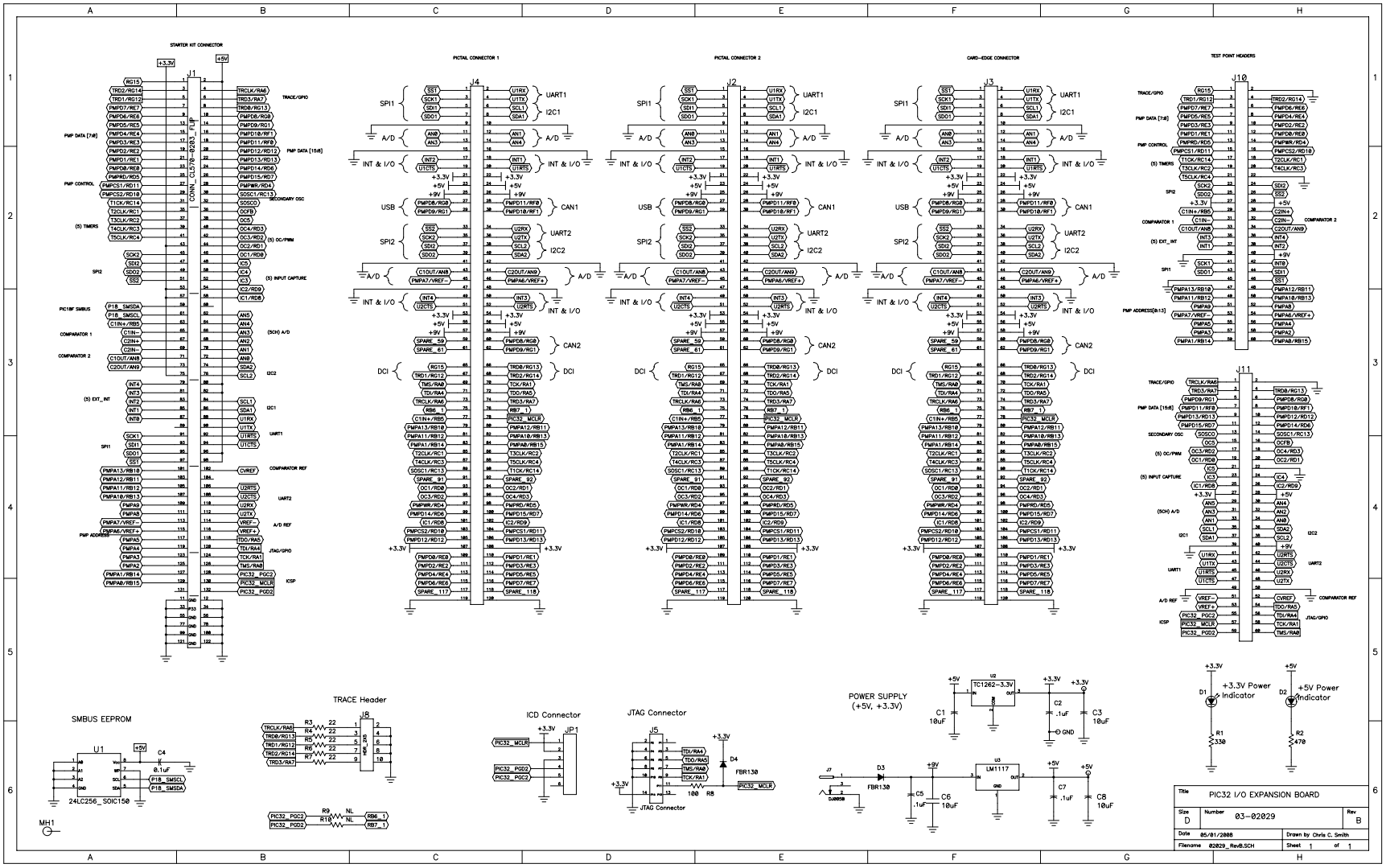
(shown with required Starter Board and optional MCU signal headers and PICtail Plus Daughter Cards)

The PIC32 I/O Expansion Board provides Starter Kit and Starter Board users with full access to MCU signals, additional debug headers, and connection of PICtail Plus daughter cards. MCU signals are available for attaching prototype circuits or monitoring signals with logic probes. Headers are provided for connecting JTAG tools or Microchip tools using the 2-wire (ICSP) interface.

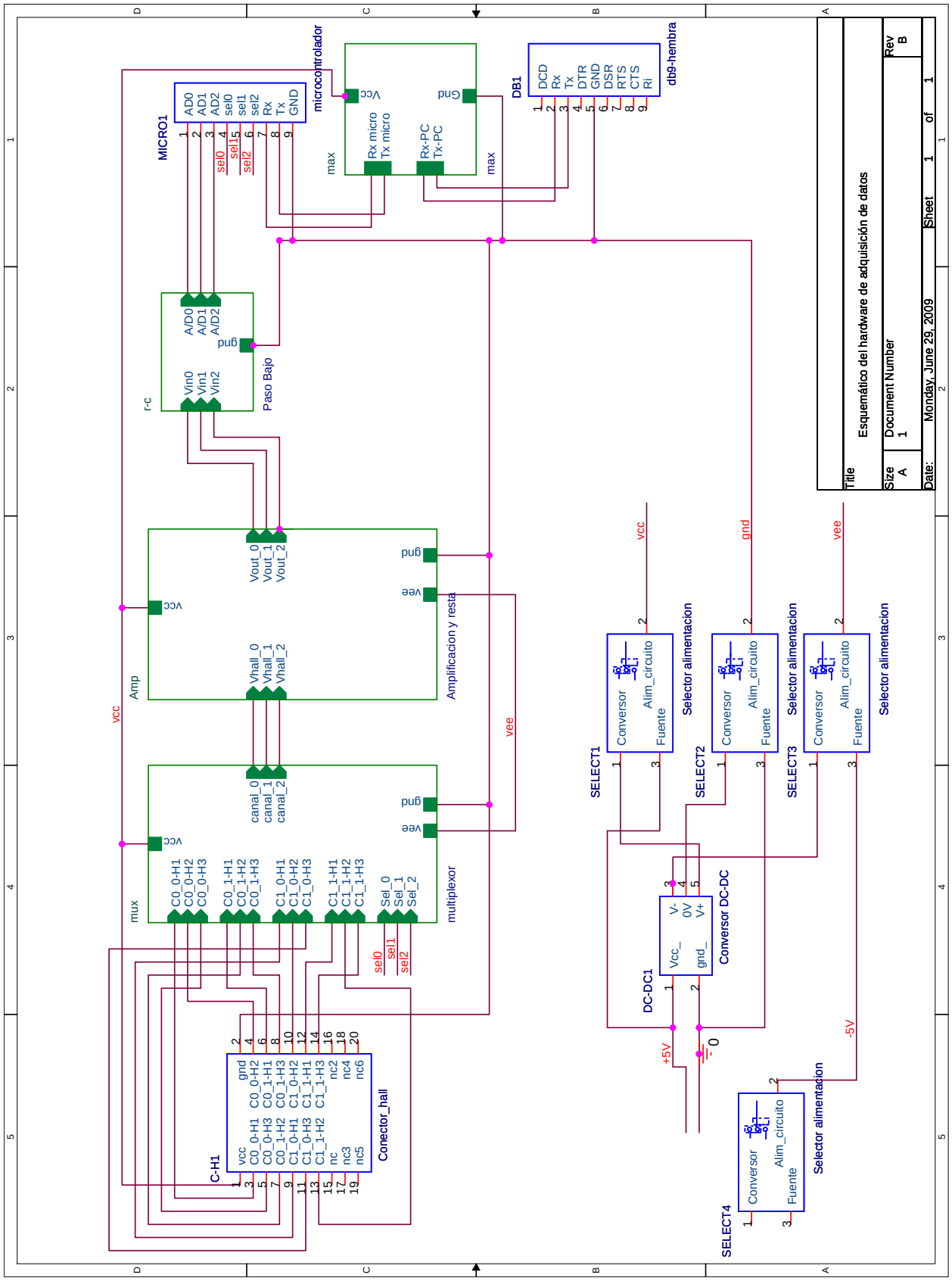


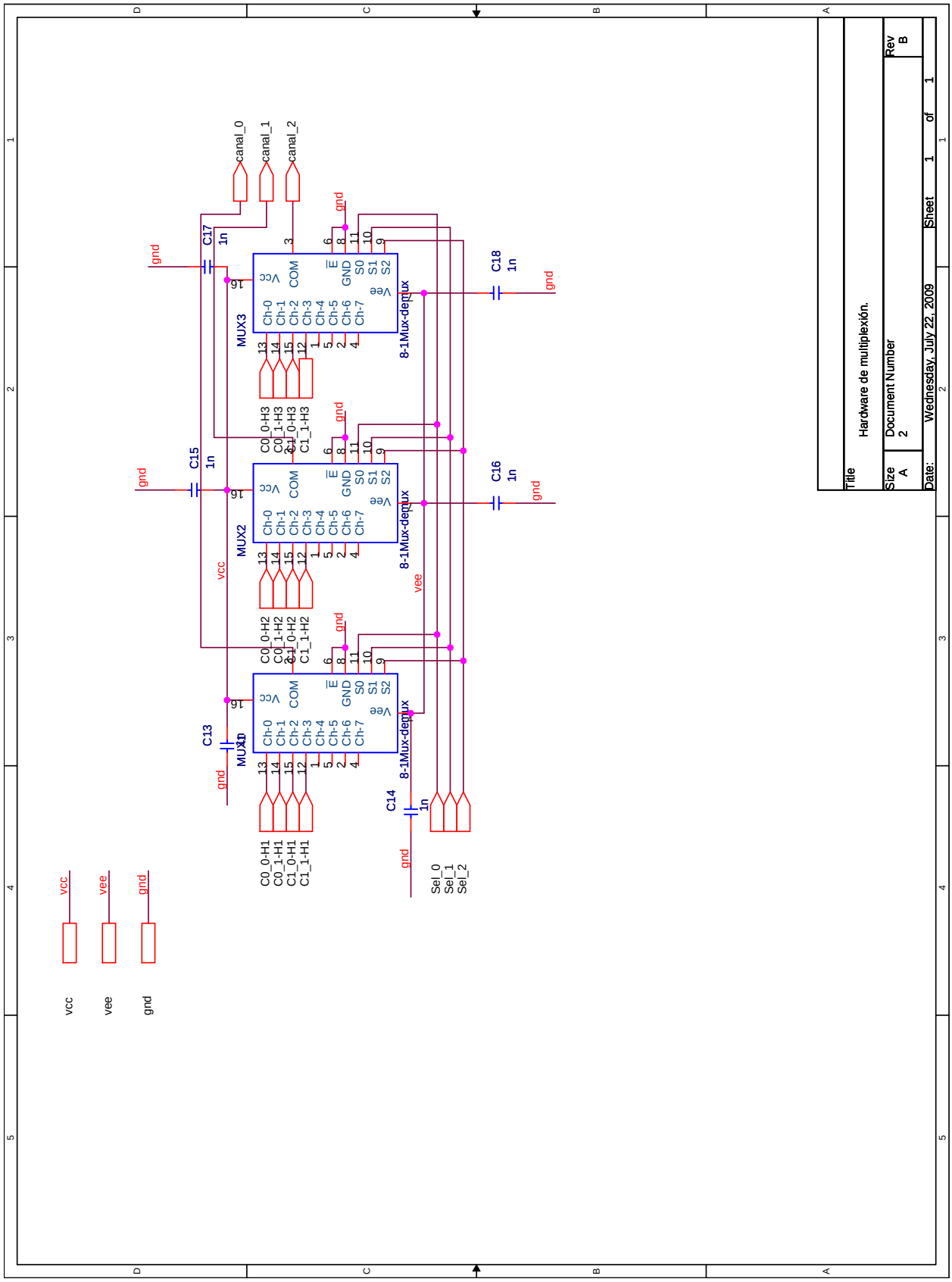
A PIC32 Starter Kit (#DM320001) or PIC32 USB Starter Board (#DM320003) is required for running application code. The I/O Expansion Board does not have an MCU.

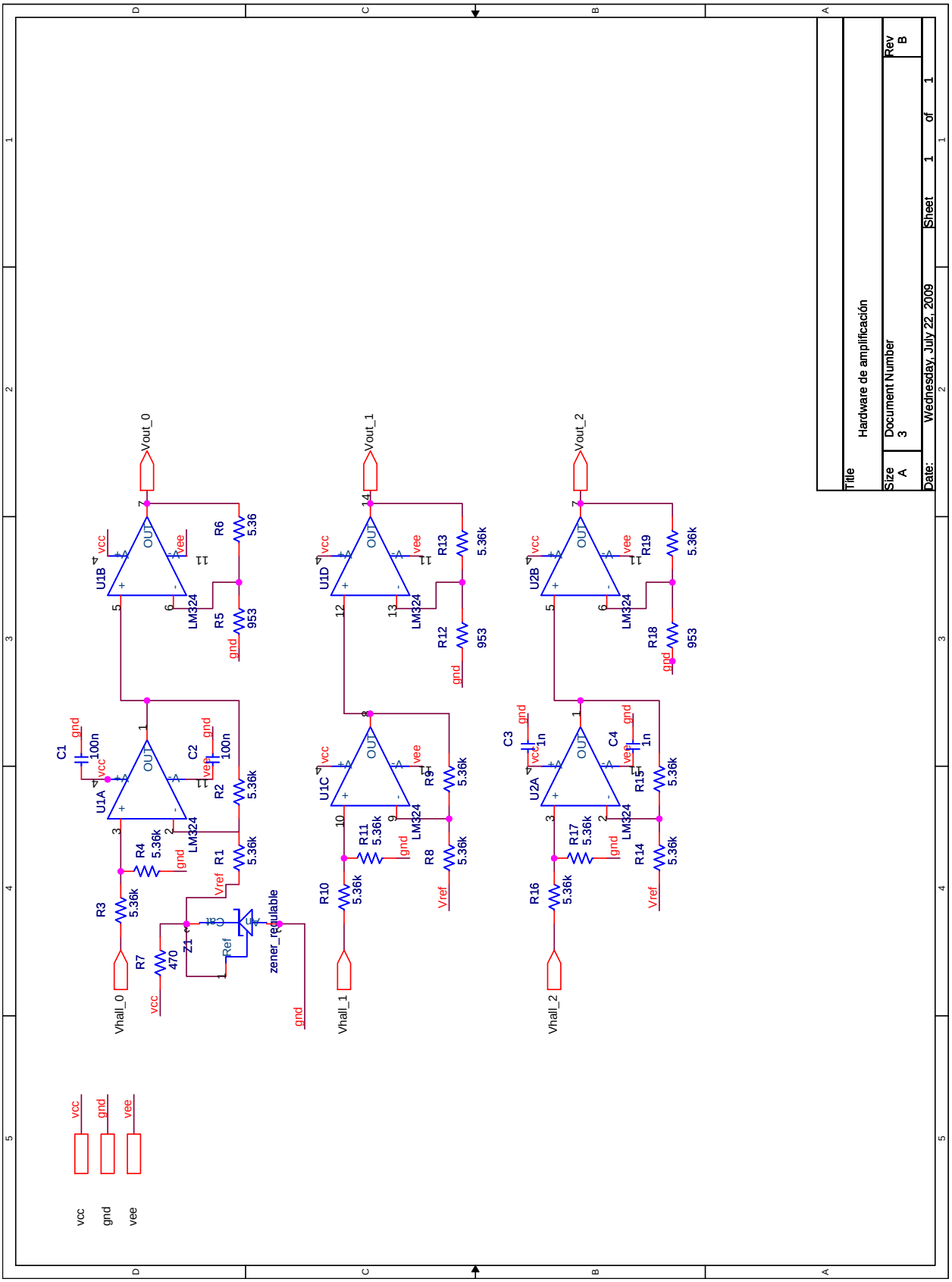
PIC32 Starter Boards can provide power to the I/O Expansion Board. The amount of power is determined by the the drive capability of the USB port connected to the Starter Board's debugger at connector J1. If additional power is required, connect an optional 9V power supply (#AC16203) to the I/O Expansion Board. This is the same power supply used with the Explorer 16 Development Board.

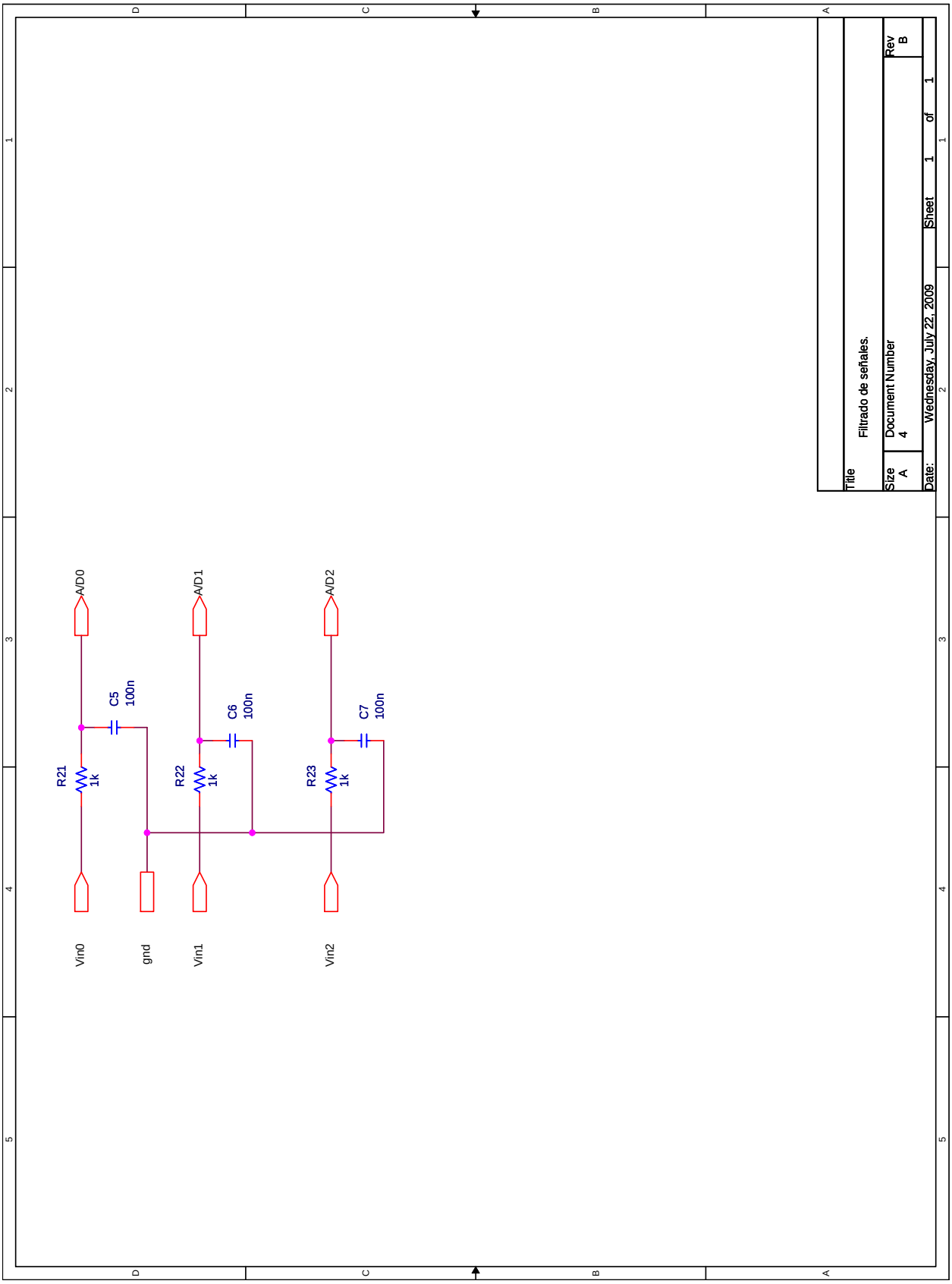

STARTER KIT CONNECTOR
PICTAIL CONNECTOR 1
PICTAIL CONNECTOR 2
CARD-EDGE CONNECTOR
TEST POINT HEADERS
SMBUS EEPROM
TRACE Header
ICD Connector
JTAG Connector
POWER SUPPLY (+5V, +3.3V)
+3.3V Power Indicator
+5V Power Indicator
PIC32 I/O EXPANSION BOARD
Number 03-02029
Rev B
Date 05/01/2008
Drawn by Chris C. Smith
Filename 02029_RevB.SCH
Sheet 1 of 1

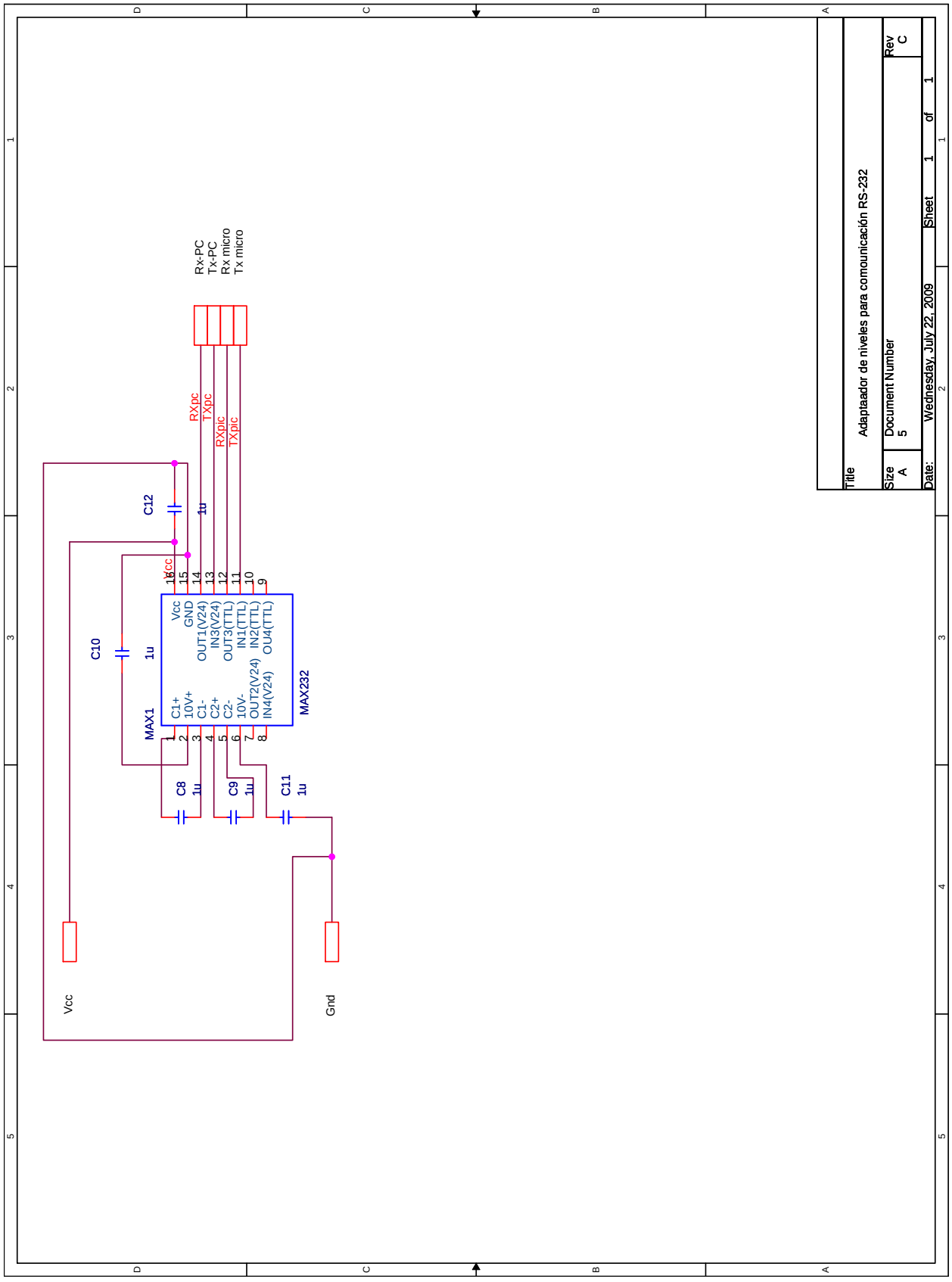
6.6. Diseño en Orcad del circuito.

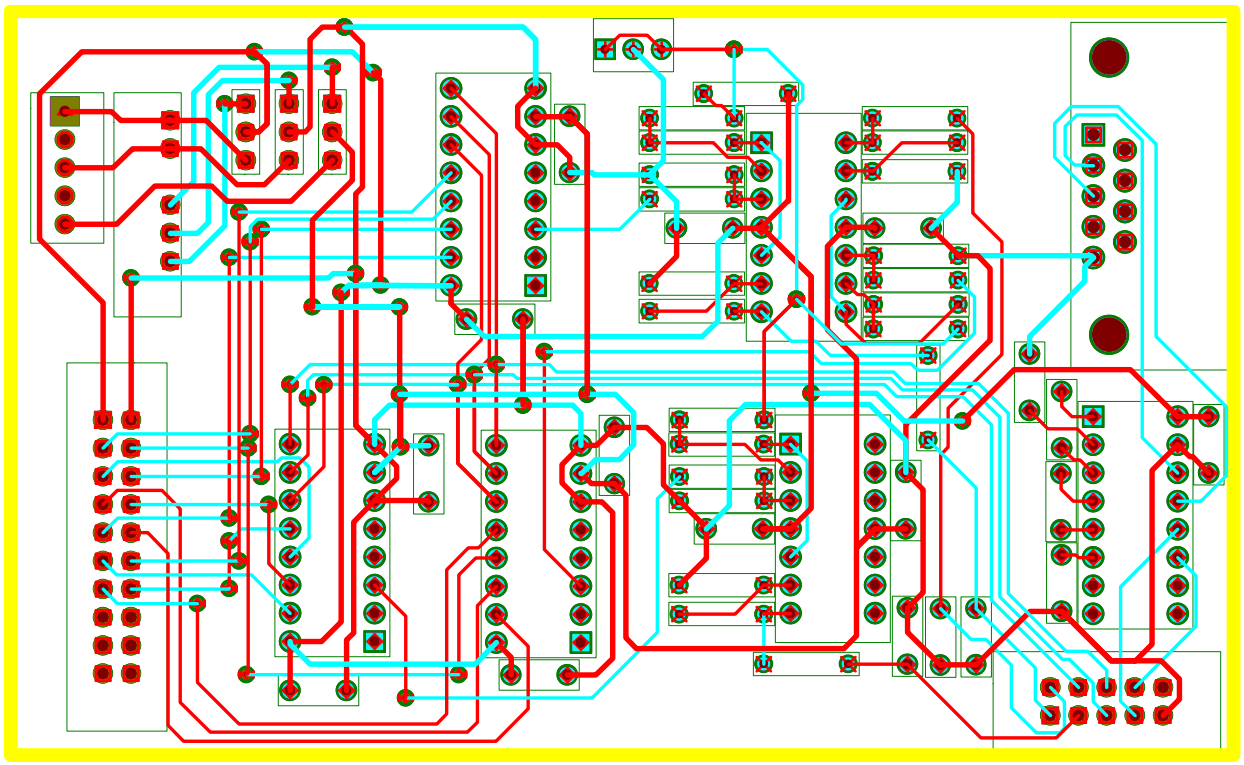



canal_0
canal_1
canal_2
C17 1n
C18 1n
C15 1n
C16 1n
C13 1n
C14 1n
MUX3
MUX2
MUX1
8-1Mux-demux
C0_0-H3
C0_1-H3
C1_0-H3
C1_1-H3
C0_0-H2
C0_1-H2
C1_0-H2
C1_1-H2
C0_0-H1
C0_1-H1
C1_0-H1
C1_1-H1
Sel_0
Sel_1
Sel_2
vcc
vee
gnd
Title
Hardware de multiplexión.
Size A
Document Number 2
Rev B
Date: Wednesday, July 22, 2009
Sheet 1 of 1


Title
Hardware de amplificación
Size
A
Document Number
3
Rev
B
Date:
Wednesday, July 22, 2009
Sheet
1
of
1
Vout_0
Vout_1
Vout_2
U1A
U1B
U1C
U1D
U2A
U2B
LM324
OUT
vcc
vee
gnd
C1
C2
C3
C4
100n
1n
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
5.36k
953
470
Vref
Vhall_0
Vhall_1
Vhall_2
Z1
Ref
zener_regulable


Title
Filtrado de señales.
Size
A
Document Number
4
Rev
B
Date:
Wednesday, July 22, 2009
Sheet
1
of
1
A/D0
A/D1
A/D2
C5
100n
C6
100n
C7
100n
R21
1k
R22
1k
R23
1k
Vin0
gnd
Vin1
Vin2


Rx-PC
Tx-PC
Rx micro
Tx micro
RXpc
TXpc
RXpic
TXpic
C12
1u
C10
1u
C8
1u
C9
1u
C11
1u
MAX1
MAX232
Vcc
GND
OUT1(V24)
IN3(V24)
OUT3(TTL)
IN1(TTL)
IN2(TTL)
OU4(TTL)
C1+
10V+
C1-
C2+
C2-
10V-
OUT2(V24)
IN4(V24)
Vcc
Gnd
Title
Adaptaador de niveles para comounicación RS-232
Size
A
Document Number
5
Rev
C
Date:
Wednesday, July 22, 2009
Sheet
1
of
1



6.7. Layout del circuito.

DRILL CHART				
SYM	DIAM	TOL	QTY	NOTE
⋈	0.381 mm		43	
×	0.711 mm		44	
+	0.787 mm		3	
◇	0.864 mm		128	
○	0.965 mm		4	
⊠	0.965 mm		44	
⊞	1.067 mm		9	
△	1.400 mm		1	
⧉	3.048 mm		2	
TOTAL			278	