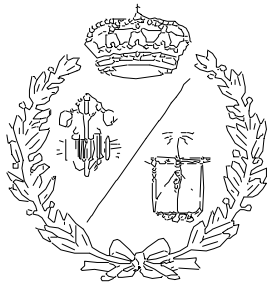
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Carrera

CONTROL DE TRAYECTORIA DE UN ROBOT MÓVIL EN ADA 95 SOBRE MaRTE OS

Para acceder al Titulo de

INGENIERO TÉCNICO INDUSTRIAL
ESPECIALIDAD ELECTRÓNICA INDUSTRIAL

Autor: Bernardo Ruíz Abascal

Octubre-2005

Quisiera mostrar mi agradecimiento a Michael y a Mario por ofrecerme la posibilidad de realizar este proyecto en el Grupo de Computadores y Tiempo Real. También agradecer al resto de personas del departamento su colaboración así como a otros alumnos que han realizado sus proyectos paralelamente a este.

Finalmente agradecer a mi familia, amigos y compañeros de carrera toda su dedicación, comprensión y apoyo.

RESUMEN

La Robocup es una competición de fútbol para equipos formados por robots que nació en 1996 con la intención de incentivar el interés por la ciencia y la tecnología. Desde el Departamento de Computadores y Tiempo Real de la Universidad de Cantabria se ha visto en este fenómeno una forma de experimentar los desarrollos y avances conseguidos en la tecnología de software de tiempo real.

En este proyecto se plantea el desarrollo de un controlador de trayectoria de un robot móvil que podría participar en la Robocup en el futuro. Dicho controlador se implementa sobre una plataforma PC, la cual utiliza como sistema operativo MaRTE OS. Se trata de un sistema operativo de tiempo real, fruto de una tesis doctoral desarrollada dentro del Grupo de Computadores y Tiempo Real , que sigue el estándar POSIX.1 (“Portable Operating System Interface”), y que está especialmente concebido para ser utilizado en pequeños sistemas empotrados.

El robot dispone de un chasis con dos ruedas independientes, motores de corriente continua, codificadores de posición magnéticos y un circuito que permite proporcionar en cada momento la potencia necesaria a los motores dependiendo de la señal de control que llega desde el PC.

La programación se realiza en el lenguaje Ada 95, lenguaje que soporta concurrencia, exclusión mutua y ofrece posibilidades de gestión de tiempo, proporcionando un código legible, modular y fiable. La aplicación desarrollada parte de unos datos de entrada como son la velocidad lineal, la velocidad angular y el tiempo de movimiento, los compara con los datos proporcionados por los codificadores de posición y calcula y envía la señal de control necesaria en cada momento a los motores. La comunicación entre el PC y los motores (trayectoria directa de control) y entre los codificadores de posición y el PC (trayectoria de realimentación) se realiza a través del puerto paralelo.

Finalmente se realiza un montaje de todos los elementos mencionados con anterioridad y se desarrolla una demostración consistente en la ejecución de segmentos de trayectorias básicas.

ÍNDICE

1 MEMORIA

1.1 Introducción y objetivos.

1.1.1 Qué es la Robocup.

Tratando a la robótica como instrumento para el entretenimiento y la educación, se encuentra el campeonato de mundial de fútbol para robots, la llamada Robocup. Esta competición nació en 1996 con la intención de incentivar el interés por la ciencia y la tecnología. Presenta un entorno dinámico, de tiempo real y distribuido donde se pueden investigar distintas tecnologías en generación de comportamiento autónomo. Por ello supone un escenario desafiante para la investigación en robótica, inteligencia artificial, etc. con el aliciente de la competición y la vistosidad. Cada año se celebra este evento que enfrenta a equipos de distintas nacionalidades los cuales pueden competir en alguna de las diferentes categorías que existen en esta Robocup. Una de estas categorías enfrenta a equipos de robots móviles de pequeño tamaño, en la que compiten equipos formados por cinco robots cada uno, sobre un tablero de ping-pong con una pelota de golf de color naranja y ayudados por una cámara cenital. El procesamiento de imágenes lo realiza un PC externo, para determinar la posición y la actuación que deberán llevar a cabo los robots en función de esa posición. Por ello, el robot sólo se ocupa de actuar, mientras la toma de decisiones es responsabilidad del PC. Existen otras categorías que van desde la que enfrenta a equipos implementados en un simulador hasta llegar a categorías como la que enfrentan a humanoides o robots bípedos), robots que intentan reproducir total o parcialmente la forma y el comportamiento cinemático del ser humano.[15]

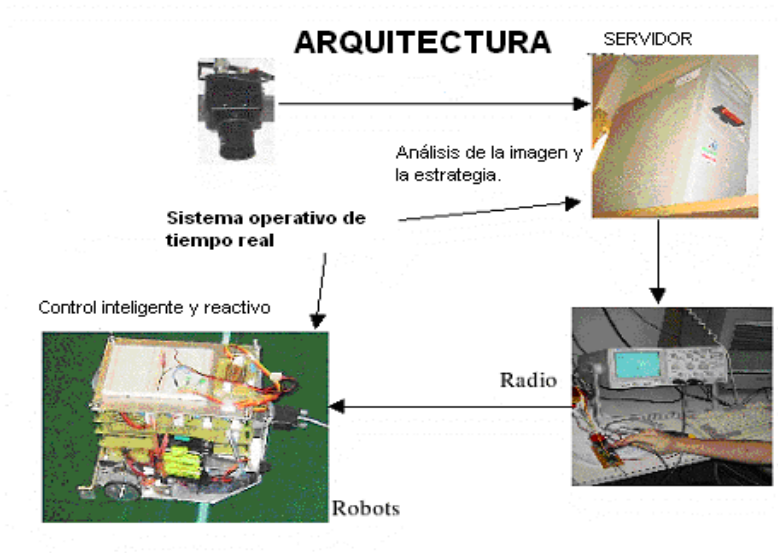


Figura 1: Descripción del entorno de desarrollo de la robocup.

La competición cuenta con una reglamentación verdaderamente estricta que en muchos de sus aspectos es bastante similar a las competiciones habituales de fútbol pero que se adapta a las condiciones particulares de sus competidores.[16].

Un partido se disputa entre dos equipos de robots formado como máximo por cinco robots incluido el portero. A cada equipo se le asignará un color antes del comienzo del partido (azul o amarillo). Es posible realizar cambios durante el partido, incluso del portero, siendo necesario para ello la no disputa de la pelota en esos momentos.

Los robots deben ser totalmente autónomos ya que sólo se permite la intervención del programador durante los descansos o los tiempos muertos.

Se permite la comunicación inalámbrica pero es necesario comunicar la frecuencia y el tipo de comunicación (no se permite la comunicación wireless bluetooth).

Las dimensiones de los competidores están estrictamente acotadas. El robot debe caber en un cilindro de 180 mm de diámetro mientras que en ningún caso su altura debe superar los 225 mm.

Un aspecto de especial importancia es el periodo de autonomía de los robots. Los partidos están tienen dos periodos de tiempo cada uno de ellos de 10 minutos. En medio se establece un descanso como máximo de 10 minutos.

Además de estar acotadas las dimensiones de los participantes también se establecen unas medidas determinadas para el terreno de juego y para el esférico. El campo es rectangular con una longitud de 4900 mm y un ancho de 3400 mm. La superficie de juego está constituida por un fieltro o alfombra de color verde sobre una superficie rígida. El esférico es de color naranja pesando aproximadamente 46 gramos y con un diámetro de 43 mm.

Como se puede apreciar esta competición cuenta con una gran cantidad de reglas y aspectos con los que hay que contar. En los párrafos anteriores simplemente hemos hecho hincapié en los aspectos más generales de esta competición pero es necesario advertir que la reglamentación es bastante más extensa.

Por este motivo es necesario inferir que la disputa de esta competición conlleva la utilización de una gran cantidad de recursos tanto económicos como de recursos humanos y de programación.

Como dijimos al principio se trata de una competición derivada de la investigación y cuyo objetivo último es el incentivar el interés por la tecnología. Por ello los equipos que acuden a dicha competición son mayoritariamente universidades que intentan demostrar allí todo su potencial humano e investigador al resto del mundo.

Una de las mejores competidoras es la Universidad de Cornell [18] que destina a la investigación en estos prototipos sumas verdaderamente sorprendentes. El diseño de dichos prototipos se realiza minuciosamente comparando distintos

componentes estructurales, ruedas, motores, sensores, hardware y software hasta llegar a conseguir el mejor robot. Para ello cuentan con equipos de trabajo que se especializan en cada una de las partes del robot.

Desde el Departamento de Computadores y Tiempo Real [17] de la Universidad de Cantabria se ha hecho una apuesta por este fenómeno viendo en su filosofía una forma de plasmar sus avances en el desarrollo de software. La implementación de las aplicaciones que se desarrollarán para el diseño de estos robots serán realizadas utilizando el sistema operativo de tiempo real MaRTE OS que ha sido desarrollado en este departamento.

Este sistema operativo es de código abierto y está en continuo desarrollo. Está pensado principalmente para aplicaciones empotradas en entornos industriales aunque actualmente su mayor uso se da en el desarrollo de proyectos de investigación, tanto en la Universidad de Cantabria como en universidades del resto de España (Politécnica de Valencia, Universidad de Vigo, Universidad de Málaga...) y del extranjero (Universidad de York).

No obstante es necesario remarcar que el objetivo de este departamento no es el de acudir a la Robocup ya que es evidente el potencial de otras Universidades. Este pudiera llegar a ser el objetivo a medio o largo plazo. De momento es una vía de investigación y desarrollo que está dando sus primeros pasos, y permite probar diferentes avances en la tecnología de software de tiempo real.

1.1.2 Antecedentes.

Durante el continuo desarrollo del sistema operativo MaRTE OS se trató de demostrar la posibilidad de migración de este a otras arquitecturas que no fueran la Intel. Para ello se utilizó un robot comercial, el soccerbot, un robot que contaba con puerto paralelo, puertos serie, LCD, entradas digitales y analógicas, cámara digital,

sensores de posición por infrarrojos , motores DC, dispositivo de golpeo del esférico...

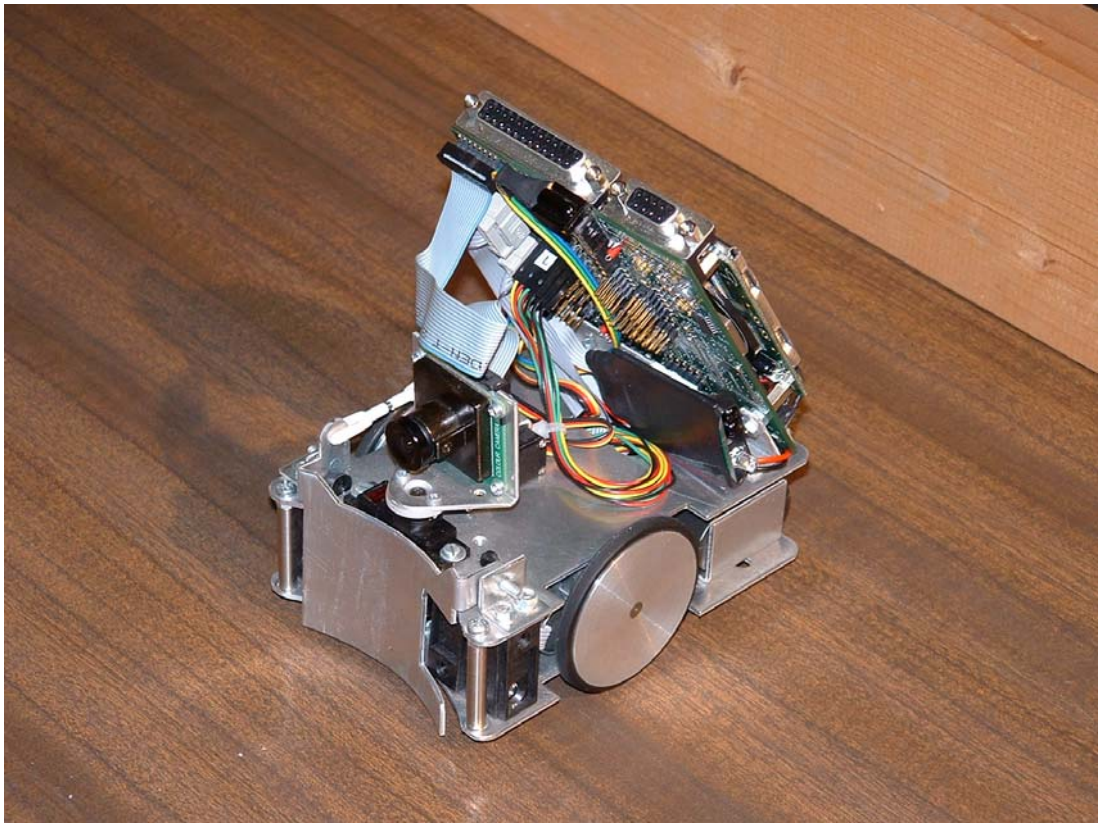


Figura 2: Soccerbot.

El control de todos estos dispostivos y accesorios se lleva a cabo gracias al controlador eyebot dirigido por el microcontrolador 68332 de 32 bits, perteneciente a la familia 68300 de controladores empotrados de Motorola, compatible con la familia 68000.

Aunque el proyecto se finalizó con éxito[19] la dificultad de poner al día dos sistemas de desarrollo, uno para PC y otro para el microcontrolador 68332, con escasos recursos de personal, han llevado al grupo de investigación a decidirse por una única plataforma de desarrollo, en concreto la basada en PCs.

Por ello, se tratará de construir robots utilizando para la etapa de control plataformas de tipo PC de la familia 386 o superior, la cual estará gobernada por

MaRTE OS. Mientras que con el soccerbot la configuración de elementos sensores y actuadores estaba ya definida de antemano ahora se irá configurando y desarrollando el robot desde sus inicios con el fin de intentar buscar una óptima elección de componentes para llegar a la consecución de un robot lo más competitivo posible.

1.1.3 Objetivos de este proyecto.

El objetivo general del proyecto es poner a punto una plataforma de robot móvil basada en procesadores de tipo PC, con sistema operativo MaRTE OS y con la finalidad de servir como base para la experimentación práctica de tecnologías avanzadas de software de tiempo real.

Más en concreto, la plataforma base que se desarrolla en este proyecto, va encaminada a conseguir la ejecución de trayectorias básicas por parte del robot:

- Se tratará de conseguir dotar al robot de una movilidad lo más alta posible dentro de su entorno.
- Las trayectorias conseguidas deberán ser tanto rectilíneas como curvilíneas, hacia delante y hacia atrás.
- Se podrán realizar a distintas velocidades.
- Se tendrá en todo momento un control de la velocidad y el espacio recorrido gracias a la realimentación del sistema de control y al uso de la temporización adecuada.
- Se intentará dejar preparada la programación para la implementación de un control superior que proporcione los datos de las trayectorias a realizar.
- Finalmente se realizará una demostración para así analizar los resultados obtenidos.

Se intentará demostrar también la viabilidad del sistema operativo MaRTE para el control de sistemas empotrados de tiempo real de pequeño tamaño, así como las

posibilidades que ofrece el lenguaje de programación Ada 95 para este tipo de sistemas.

Para la consecución de estos objetivos será necesario el estudio previo de todos los componentes estructurales y de hardware de los que se dispone para optar por las configuraciones que en cada momento aparezcan como más oportunas.

1.1.4 Organización de la memoria

En el primer capítulo hemos definido los objetivos de este proyecto haciendo referencia tanto a la idea de la que nace como a los antecedentes habidos.

En el segundo capítulo se hace una descripción de los sistemas de tiempo real, centrándose posteriormente en el caso particular de MaRTE OS. Asímismo se plantean las ventajas del uso del lenguaje Ada 95.

En el tercer capítulo se hace una descripción de los elementos hardware que constituyen el prototipo, explicando cuáles han sido las distintas opciones de configuración de algunos de ellos y justificando las elecciones realizadas.

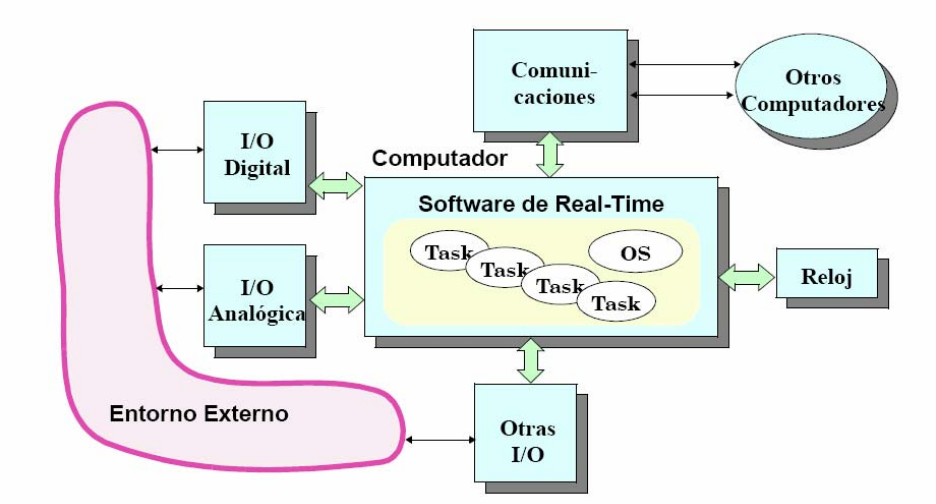
El capítulo 4 se centra en la programación software realizada, plasmándose los cálculos realizados y los resultados obtenidos.

En el capítulo 5 se plantean las conclusiones a las que se ha llegado, estableciéndose las posibles mejoras futuras, algunas de las cuales serán detalladas.

1.2 Introducción a los sistemas de tiempo real.

1.2.1 Introducción.

Existen sistemas computacionales fuertemente relacionados con el entorno que les rodea y con el que se encuentran en constante interacción. Ejemplos de este tipo de sistemas son los sistemas de adquisición de datos y los controladores industriales, los cuales están compuestos por diversos elementos (actuadores, sensores, unidades de cómputo, redes de interconexión, etc.) que deben trabajar de forma conjunta y coordinada. Debido a la naturaleza cambiante del entorno con el que interactuan, junto con la necesidad de coordinación entre sus componentes, los resultados obtenidos sólo podrán ser considerados válidos cuando, además de ser correctos desde el punto de vista lógico, hayan sido generados a tiempo. Resultará, por tanto, preciso imponer restricciones temporales cuyo cumplimiento garantice el correcto funcionamiento del sistema. Este tipo de sistemas, capaces de realizar tareas y responder a eventos asíncronos externos dentro de unos plazos temporales determinados son los denominados “Sistemas de Tiempo Real”.[2].



2. Figura 3: Aspecto general de un sistema de tiempo real.

Para que sea posible la predecibilidad temporal del sistema completo, todas las partes que le componen deberán de presentar un comportamiento predecible. En consecuencia, en el caso de utilizar un sistema operativo, los servicios que éste proporcione a las aplicaciones deberán presentar tiempos de respuesta acotados, de forma que así sea capaz de garantizar los requerimientos temporales de los procesos bajo su control. Mientras que en un sistema operativo de tiempo compartido, como Unix , lo importante es proporcionar a los usuarios unos buenos tiempos de respuesta promedios , la clave en los sistemas operativos de tiempo real será garantizar los requerimientos temporales incluso en el peor caso posible; el tiempo de respuesta promedio pasa así a un segundo plano.[1].

1.2.1.1 Clasificacion de los sistemas de tiempo real.

- Según su criticidad:

  a)Tiempo real estricto/críticos (*hard real-time*): Todas las acciones deben ocurrir dentro del plazo especificado. Ejemplo: *control de vuelo.*

  b)Tiempo real flexible/acrítico (*soft real-time*):Se pueden incumplir plazos de vez en cuando.El valor/utilidad de la respuesta decrece con el tiempo en que se sobrepasen los plazos. Ejemplo: *adquisición de datos.*

  c)Tiempo real firme (*firm real-time*):Se pueden perder plazos ocasionalmente. Una respuesta tardía no tiene valor. Ejemplo: *sistemas multimedia*.

- Según la activación de las tareas:

  a)Sistemas dirigidos por tiempo: su mecanismo de activación es el reloj.

  b)Sistemas dirigidos por eventos: su mecanismo de activación son señales externas, tanto software como hardware (interrupciones).

- Según la ubicación de la aplicación:

  a)Centralizados: Sólo existe una CPU, la cual lleva a cabo todo el procesamiento y las repuestas asociadas a él.

b)Distribuidos: El sistema se compone de nodos, cada uno de los cuales posee una memoria y uno o más procesadores. Se tiene que establecer una forma de comunicación bien definida entre los nodos.

1.2.2 Sistemas empotrados de tiempo real.

Los sistemas empotrados se caracterizan porque en ellos el computador constituye una parte más de un sistema mayor en el que se encuentra altamente integrado y en el que se dedica a realizar una función (o un pequeño conjunto de ellas). Las aplicaciones tradicionales de este tipo de sistemas incluirían sistemas de control en aviones, trenes, nudos de telecomunicaciones, motores de automóviles, procesos industriales, teléfonos móviles, etc. Muchos sistemas empotrados son también sistemas de tiempo real, por su fuerte interacción con el entorno.

La utilización de los sistemas empotrados está extendiéndose paulatinamente a otras muchas aplicaciones a medida que los avances tecnológicos permiten disponer de procesadores baratos cada vez más potentes y de memorias de menor coste y mayor escala de integración. Así, ya es habitual encontrarse con computadores empotrados en productos como televisores, juguetes, reproductores de discos DVD, pequeños electrodomésticos, etc. Sin embargo, la mayor parte de los sistemas empotrados ven limitadas sus prestaciones por razones de tamaño, peso, consumo o coste. Por las citadas razones, lo normal es que no dispongan de un terminal de propósito general para interfaz con el usuario, ni de sistema de ficheros o dispositivos de almacenamiento magnéticos, y que tengan un procesador y capacidad de memoria muy reducidos.

Paralelamente a la mejora de los componentes hardware se están desarrollando aplicaciones empotradas mediante el uso de lenguajes de alto nivel y sistemas operativos que proveen servicios específicos para entornos de tiempo real.

1.2.2.1 Necesidad de un estándar: POSIX (Portable Operating System Interface).

En un principio, fueron los fabricantes de los sistemas hardware los que gradualmente procedieron a incorporar algunos de estos conceptos en sus sistemas de desarrollo específicos. La consecuencia de esta acción descoordinada y casi siempre competitiva, resultó en un considerable esfuerzo para transferir o adaptar el software a las diferentes plataformas hardware de sus controladores industriales, que por su naturaleza son en general poco homogéneos. La solución a esta problemática se está buscando a nivel internacional mediante la elaboración de normas de estandarización, que permiten independizar los diferentes aspectos del diseño del sistema, consiguiendo con ello una mayor modularidad, reusabilidad y, en definitiva, la reducción del tiempo de diseño y de los costos.

Entre las normas internacionales de estandarización destaca el estándar POSIX (conocido en el ámbito internacional con la referencia ISO/IEC-9945), cuyo objetivo es permitir la portabilidad de aplicaciones a nivel de código fuente entre diferentes sistemas operativos.

El estándar POSIX está gozando de gran aceptación entre los fabricantes. Así en los últimos años han ido apareciendo numerosos sistemas operativos de tiempo real conformes (en mayor o menor medida) con el estándar POSIX. La mayoría de ellos, entre los que se encuentran los más utilizados por la industria a nivel mundial, son sistemas propietarios tales como VxWorks, pSOSystem , QNX o LynxOS. Pero también este estándar es seguido por gran cantidad de sistemas operativos distribuidos bajo políticas de libre distribución y código libre, entre ellos que cabe citar MiThOS, RTEMS , RT-Linux y S.Ha.R.K. [1].

Otro importante conjunto de estándares en el área de las aplicaciones de tiempo real está constituido por los lenguajes de programación como C++ y Ada 95. Ada 95 soporta los aspectos más avanzados de la ingeniería software para sistemas de tiempo real, como son la programación concurrente con tiempos de respuesta predecibles, estrategias sofisticadas de planificación de tareas y programación orientada a objetos. Un lenguaje de programación que ha experimentado un

importante avance en el campo de los sistemas empotrados es el Java debido principalmente a la portabilidad a nivel binario que proporciona. Existe actualmente una especificación Java para sistemas de tiempo real.

1.2.3 Algoritmos de planificación.

Como se expuso anteriormente, un sistema de tiempo real se caracteriza por la fuerte interacción que mantiene con el entorno que le rodea, debiendo responder a los diferentes eventos generados por este en unos plazos de tiempo preestablecidos. La naturaleza compleja y concurrente del entorno, conduce a la utilización de arquitecturas de software concurrente para los sistemas de tiempo real. En este tipo de arquitecturas, el sistema global se divide en un conjunto de actividades concurrentes, que denominaremos tareas, cada una de ellas encargada de responder a un determinado evento o conjunto de eventos generados por el entorno. El software producido siguiendo este modelo será más próximo al sistema real, resultando por tanto mucho más sencillo y comprensible.

Puesto que el número máximo de tareas que es posible ejecutar simultáneamente en un computador es limitado (como máximo igual al número de procesadores), es necesario definir un conjunto de reglas que permitan determinar qué tarea o tareas deben ser ejecutadas en cada momento.

a)Ejecutivo cíclico:

El algoritmo de planificación históricamente más utilizado en sistemas empotrados de tiempo real es el “ejecutivo cíclico”, el cual utiliza una tabla o “plan estático”, que es recorrido cíclicamente para indicar los instantes en que cada tarea debe tomar y abandonar la CPU. Las principales ventajas de este algoritmo son la simplicidad y la predecibilidad absoluta de los instantes de ejecución. Las principales desventajas son la falta de flexibilidad y la ineficiencia para gestionar eventos aperiódicos.

b)Prioridades:

Los sistemas operativos de tiempo real utilizan algoritmos de planificación en tiempo de ejecución basados en prioridades, en los que los problemas de contención

de recursos se resuelven en el instante que se producen, eliminándose la necesidad de que exista un plan de ejecución previo. En dichos algoritmos, a cada tarea le es asignada una prioridad y, en función de ella, se resuelven los posibles conflictos de utilización del procesador y de los demás recursos del sistema.

Dependiendo de si la prioridad de las tareas es constante o cambia en función del estado del sistema los algoritmos de planificación en tiempo de ejecución se dice que están basados en prioridades estáticas o dinámicas.

b.1) fijas: la mayor parte de los sistemas operativos de tiempo real usan prioridades fijas (estándar POSIX, lenguaje Ada95...). Sus principales ventajas son que permiten un alto aprovechamiento de la CPU, poseen una implementación sencilla y que cuentan con una consolidada base teórica (RMA: ("Rate Monotonic Analysis") y DMA: ("Deadline Monotonic Analysis")), que permiten determinar si un conjunto de tareas es planificable así como realizar una asignación óptima de prioridades (a menor plazo de ejecución mayor prioridad).

b.2) dinámicas: se adaptan a entornos dinámicos en los que la carga del sistema no puede ser conocida de antemano, permitiendo aprovechar al máximo la potencia del procesador. Se sustentan sobre algoritmos como el EDF ("Earliest Deadline First") y el LLF ("Least Laxity First").

Además, sea cual sea el tipo de prioridades, puede diferenciarse entre políticas expulsoras y no expulsoras: en una política de planificación no expulsora la tarea en posesión del procesador no verá detenida su ejecución hasta que así lo desee o bien hasta que se bloquee al tratar de acceder a algún recurso no disponible en ese momento. Por el contrario, en una política expulsora una tarea perderá la posesión del procesador en el momento en que aparezca otra tarea de mayor prioridad lista para ejecutar. El planificador no expulsor es más sencillo de implementar, aunque a cambio, presenta la importante desventaja de que su uso puede provocar retrasos importantes en la ejecución de las tareas más prioritarias.

c)Reparto proporcional:

Existe otro tipo de algoritmo de planificación no basado en prioridades que se utiliza principalmente en el campo de las comunicaciones y de las aplicaciones

multimedia (muestreo de voz, adquisición de imágenes, reproducción de vídeo, etc.). En el tipo de aplicaciones mencionadas la carga de trabajo es variable o no se conoce de antemano. Estos algoritmos no buscan imponer requisitos temporales estrictos sino que persiguen el reparto proporcional de los recursos del sistema entre las distintas tareas o procesos que lo componen permitiendo que a cada tarea se le concede el porcentaje de tiempo de procesador solicitado.

1.2.4 Protocolos de sincronización.

Además de tener en cuenta el reparto del uso del procesador (algoritmos de planificación) es necesario establecer unos criterios que regulen el uso concurrente de ciertos dispositivos (almacenamiento, I/O...) y de estructuras de datos compartidas por varias tareas.

Es necesario asegurar la exclusión mutua de manera que mientras una tarea este haciendo uso de una estructura de datos las demás que quieran usarlo permanezcan bloqueadas a la espera de que sea liberado.

Sin embargo esto puede provocar que una tarea de alta prioridad permanezca excesivo tiempo bloqueada produciéndose retrasos en la ejecución de la misma (inversión de prioridad no acotada). Para resolver este problema y conseguir una inversión de prioridad acotada por la duración, generalmente pequeña, de las secciones críticas se recurre a los protocolos de sincronización , los cuales modifican temporalmente las propiedades de planificación de las tareas.

Existen varios protocolos de sincronización tanto para prioridades fijas (“Techo de Prioridad Inmediato”, “Herencia Básica de Prioridad” y “Priority Ceiling Protocol”) como para prioridades dinámicas (“Monitor Centralizado”, “Stack Resource Protocol” (SRP)…).

1.2.5 Sistema Operativo MaRTE OS.

El origen y desarrollo del sistema MaRTE OS provienen del resultado de una tesis doctoral que se desarrolló dentro del Grupo de Computadores y Tiempo Real, adscrito al Departamento de Electrónica y Computadores de la Universidad de Cantabria. Esta tesis doctoral *"Planificación de Tareas en Sistemas Operativos de tiempo Real Estricto para Aplicaciones Empotradas"* propone una interfaz para sistemas empotrados bajo la cual sea posible desarrollar aplicaciones en las que se puedan definir internamente algoritmos de planificación de tareas, utilizando para su implementación las bases establecidas en el estándar POSIX de sistemas operativos de tiempo real. El hecho de cumplir la integración y compatibilidad con este estándar POSIX hace que se facilite el uso de la interfaz para programadores familiarizados con este estándar. La sucesiva aprobación, primero del estándar POSIX 1003.1b en el año 1993 (extensiones de tiempo real), también conocido como POSIX.4, y posteriormente del estándar POSIX 1003.1c en el año 1995 (extensión de '*threads*'), también llamado POSIX.4a, supusieron que ya se pudiera desarrollar aplicaciones con requisitos de tiempo real sobre sistemas operativos POSIX. Actualmente estos estándares están integrados en la versión POSIX.1 de 2003.

Pero un sistema que implemente el estándar POSIX de manera completa es un sistema demasiado grande para su uso en pequeños sistemas empotrados y por ello se aprobó en el año 1997 otro estándar, POSIX.13, en el cual se definieron cuatro subconjuntos de servicios del sistema operativo (*"perfiles de entornos de aplicación"*) para que distintas plataformas pudieran usar distintos perfiles. Sus diferencias se encuentran en la presencia o no presencia de un sistema de ficheros complejo y jerárquico, y en la utilización o no utilización de múltiples procesos. El estándar POSIX.13 se revisó en el año 2003 para adaptarlo a la versión POSIX.1 de ese mismo

año.

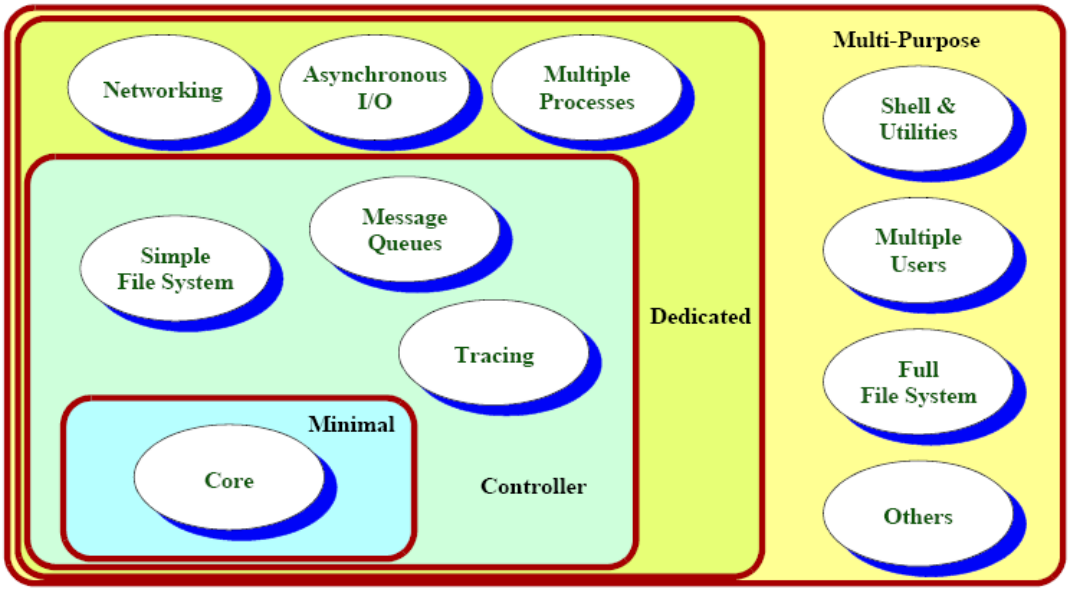


Figura 4: perfiles de aplicación

De todos ellos, el perfil que interesa para la intenciones de desarrollo de MaRTE OS es el *"Sistema de Tiempo Real Mínimo"*, usado en aplicaciones empotradas pequeñas, que soportan unos servicios POSIX mínimos para reducir el tamaño de las aplicaciones desarrolladas bajo este perfil. Las características que lo diferencian del resto de los perfiles se encuentran en el hecho de que no requiere el soporte para múltiples procesos ni tampoco un complejo sistema de ficheros jerárquico, eliminándose así la mayor parte del tamaño y la complejidad del POSIX completo.

En el desarrollo de la citada tesis doctoral, una vez situada la base que conforma el estándar POSIX para la implementación de esta interfaz para definir los algoritmos de planificación, se llega a la conclusión de que no se encuentra ningún sistema operativo POSIX apropiado para poder realizar la necesaria implementación de la interfaz mencionada, de modo que se diseña e implementa un nuevo sistema operativo de tiempo real bautizado como MaRTE OS (*"**M**inim**a**l **R**eal-**T**ime Operating System for **E**mbedded Applications"*), sobre el cual se va a desarrollar toda la base de este proyecto.

El código de MaRTE se ha realizado bajo las condiciones de libre distribución de GPL ("***G****NU's* ***P****ublic* ***L****icense*"). Este uso de las condiciones GPL de libre *software* hace del MaRTE OS un sistema operativo de código abierto, de gran utilidad para docencia y futuras investigaciones en sistemas de tiempo real.

Esas condiciones establecidas para el proyecto GNU suponen que el propósito de la distribución sea la utilidad general, pero este hecho de ser *software* libre implica que no se puede ofrecer ninguna garantía de su correcto funcionamiento, incluso las implícitas '*garantía mercantil*' o la '*garantía de ajuste a un propósito particular*'. La libre distribución hace que se rompa con la tendencia de los sistemas propietarios que realmente siguen el estándar POSIX de manera completa, pero que no ofrecen la posibilidad de tener acceso al código del sistema operativo.

1.2.5.1 Principales características de MaRTE OS.

El sistema operativo MaRTE OS se caracteriza principalmente por ser un sistema operativo diseñado bajo las normas del subconjunto mínimo definido en el POSIX.13, y a raíz de esta característica primordial, se tienen otra serie de características asociadas, como:

- Está pensado para aplicaciones principalmente estáticas.
- Presenta tiempos de respuesta acotados en todos sus servicios.
- Existe un sólo espacio de direcciones de memoria, compartido tanto por el núcleo como por la aplicación.
- Se permite la ejecución de aplicaciones desarrolladas tanto en lenguaje Ada como en lenguaje C.
- Incluye gestión de memoria dinámica simplificada.
- Es portable para el uso en distintas arquitecturas.
- Presenta un '*núcleo monolítico*', con la mayor parte de las llamadas al sistema utilizando accesos a secciones críticas.
- Toma la forma de una librería para ser enlazada con la aplicación.

Como se comentó en el apartado anterior, MaRTE OS sigue las directrices marcadas por el estándar POSIX.13 para *"Sistemas de Tiempo Real Mínimo"*, referido a pequeños sistemas empotrados, y es por ello que se tienen que soportar una serie de servicios para cumplir con las necesidades impuestas por el estándar. Algunos de ellos son los siguientes:

- Soporte para ejecución concurrente.
- Planificación de threads.
- Sincronización entre threads.
- Señales.
- Acceso a dispositivos.
- Servicios de temporización.
- Paso de mensajes.
- Servicios de configuración.
- Gestión de memoria dinámica.

Además también se soportan algunos servicios no necesarios para pequeños sistemas empotrados, pero que deben ser incluidos en el sistema MaRTE para mantener la posible compatibilidad con el resto de los perfiles:

- Bloqueo de memoria.
- Entrada/Salida sincronizada y no sincronizada.
- Objetos de memoria compartida.

El sistema así engendrado, bajo el desarrollo propio de la tesis, nos ofrece un núcleo ("*kernel"*) que da soporte a dos interfaces POSIX, la original, para el POSIX-C y también para POSIX-Ada, pudiéndose ejecutar aplicaciones empotradas desarrolladas tanto con '*threads*' en C como con tareas en Ada.

A continuación, se puede observar la arquitectura con la cual se trabaja cuando se ejecuta una aplicación en el sistema operativo MaRTE OS, tanto para las aplicaciones en Ada como para las desarrolladas en lenguaje C. Entre ambas se tiene la diferencia de la capa intermedia que hace de interfaz de unión entre la aplicación y la parte más interna del núcleo del sistema operativo.[7].

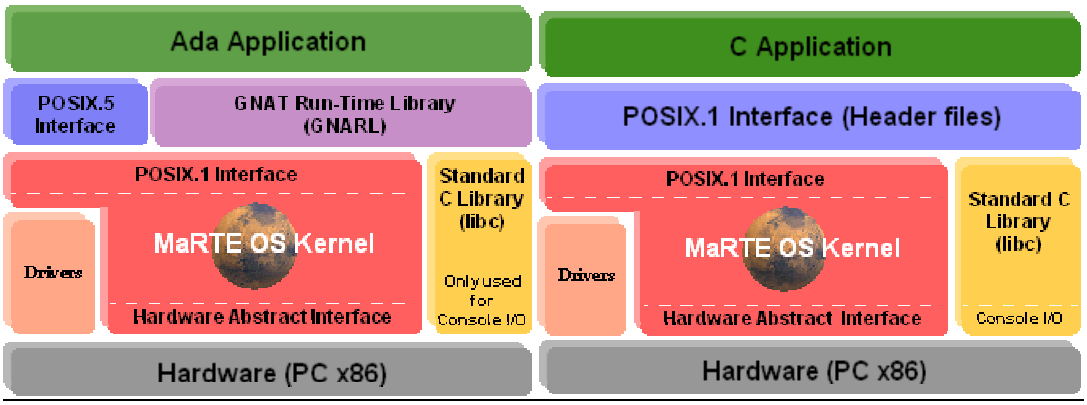


Figura 5: Arquitectura para aplicaciones en Ada y en C.

1.2.5.2 Creacion carga y depuración de las aplicaciones sobre MaRTE OS.

Lo habitual en los sistemas empotrados es que el desarrollo de las aplicaciones se realice en un entorno cruzado formado por dos computadores: el computador empotrado sobre el que se ejecutarán las aplicaciones y el equipo de desarrollo, generalmente más potente y con un sistema operativo de propósito general, en el que se dispone de un conjunto de herramientas que posibilitan la creación, carga en el equipo de ejecución y depuración de las aplicaciones.

En este sentido MaRTE OS no consiste únicamente en un núcleo en el que se implementa la funcionalidad incluida en el subconjunto mínimo del POSIX (concurrencia, sincronización, temporización, etc), sino que además consta de una serie de aplicaciones y servicios que posibilitan la creación, carga y depuración de aplicaciones.

Para realizar el entorno de creación, carga y desarrollo de las aplicaciones del sistema MaRTE OS se utiliza un equipo con Linux bajo un entorno cruzado especifico, con los compiladores GNAT y GCC y una serie de archivos de órdenes (*scripts*) desarrollados en lenguaje Perl que facilitan la generación y ejecución de dichas aplicaciones.

En este equipo con Linux se encuentra el núcleo del sistema MaRTE OS además de una serie de librerías, que son enlazadas de manera conjunta con el código de la aplicación que pretende crearse, tras haberse hecho la compilación correcta del código.

Las aplicaciones se generan en el equipo Linux, pero han de ser cargadas de modo remoto desde el sistema empotrado, y para ello se usa una utilidad especial que carga la aplicación a través de la red de comunicaciones. Esta utilidad se conoce como ***'NetBoot'***, y se caracteriza por realizar un arranque remoto de una aplicación dentro de una red Ethernet, configurándose para ello diferentes parámetros de manera inicial.

Aparte también se tiene una comunicación entre ambos equipos establecida a través del puerto serie, utilizada para realizar la depuración de la aplicación en tiempo de ejecución a través de la utilidad GDB, cargada desde el equipo Linux sobre el que se ha desarrollado la aplicación en primera instancia.

Actualmente, el sistema MaRTE OS es un producto perfectamente útil para el desarrollo de aplicaciones industriales empotradas, y también es una gran herramienta para el estudio e investigación de los sistemas operativos de tiempo real.

1.2.6 El lenguaje Ada 95.

El lenguaje de programación Ada 95 es un descendiente directo de Ada 83, el cual fue desarrollado por iniciativa del Departamento de Defensa de los EE.UU. (DoD) para ser utilizado en sistemas empotrados.[1].

A principio de la década de los 70, el DoD usaba gran cantidad de software, en su mayor parte en aplicaciones empotradas, que había sido desarrollado por compañías independientes que utilizaban una enorme variedad de lenguajes de programación. El rápido progreso de la tecnología en el campo de los dispositivos electrónicos que tuvo lugar en aquella época parecía indicar que sería posible abordar aplicaciones software mucho más ambiciosas y complejas, pero la realidad era que los grandes programas no se finalizaban dentro de los plazos programados, su costo final excedía enormemente lo inicialmente presupuestado, contenían muchos errores y no satisfacían a los clientes ya que rara vez cubrían todos sus requerimientos.

Como respuesta a esta crisis, el Departamento de Defensa estudió las características que debía reunir un lenguaje de programación, principalmente enfocado a los sistemas empotrados, de forma que se redujesen los costos de desarrollo y mantenimiento del software, descubriendo que ninguno de los lenguajes existentes cumplía los requisitos deseados. Por esta razón lanzó un concurso para el diseño de un nuevo lenguaje de programación que permitiera aplicar de forma eficiente los principios de la ingeniería software.

El proceso culminó en 1983 con la generación del estándar ANSI para el lenguaje de programación Ada. En el año 1988, y de nuevo por iniciativa del Departamento de Defensa de los EE.UU., se puso en marcha el proyecto Ada 9X, con el que se pretendía revisar el lenguaje Ada 83 para adaptarle a los nuevos paradigmas de programación, principalmente a la “programación orientada a objetos”. La revisión fue provocada en gran medida por el hecho de que el lenguaje, inicialmente pensado para sistemas empotrados, estaba siendo utilizado en muchas otras áreas de aplicación. El proceso de revisión finalizó en 1995 con un estándar ISO que describe el manual de referencia del lenguaje Ada 95. Recientemente

fueron publicadas las correcciones técnicas del estándar anteriormente mencionado constituyendo el “Manual de Referencia Ada Consolidado”.

1.2.6.1 Principales características

Modularidad y abstracción:

Los módulos o componentes software son implementados en Ada mediante el concepto de paquete. Los paquetes Ada permiten un alto nivel de abstracción y ocultamiento de la información (la claves para la escritura de software reusable) imponiendo una clara distinción entre las partes visibles para el resto de la aplicación, descritas en la interfaz del paquete, y las dependientes de la implementación que se hayan contenidas en su cuerpo. La existencia del concepto de paquete genérico, el cual permite independizar los algoritmos de los tipos de datos sobre los que operan, facilita aún más el proceso de construcción de código reutilizable.

La abstracción de datos se obtiene mediante la utilización conjunta de los paquetes y de la característica del lenguaje que permite al programador definir sus propios tipos de datos junto con los operadores asociados. Los nuevos tipos así creados serán tratados por el compilador igual que se tratará a los tipos predefinidos por el lenguaje.

Los paquetes constituyen unidades de compilación independiente. Esto facilita el desarrollo de grandes aplicaciones, puesto que los módulos pueden ser compilados y probados de forma independiente por los distintos equipos de trabajo sin necesidad de disponer aún de la aplicación final. Además la estructura del lenguaje permite conocer en tiempo de compilación las dependencias entre los distintos módulos sin que sea necesaria la intervención del programador como ocurre en otros lenguajes (como el C y el uso de los “makefiles”). Esto permite automatizar el proceso de compilación de grandes aplicaciones, con lo que se puede optimizar el número de recompilaciones a realizar tras una modificación y se evitan los errores que la intervención del programador puede provocar.

Fiabilidad:

Una de las características más relevantes del lenguaje Ada es la fiabilidad del código generado, conseguida gracias a que las reglas del lenguaje permiten detectar numerosos errores en tiempo de compilación. La mayor parte de los errores detectados durante la fase de compilación lo son gracias a la propiedad de tipología estricta del lenguaje. Dicha propiedad obliga al programador a hacer explícita cualquier conversión de tipos que desee realizar, suponiendo un error de compilación la utilización de un dato de tipo indebido en una expresión, como parámetro de un procedimiento o función o en cualquier otra construcción del lenguaje.

A pesar de las características del lenguaje y del cuidado diseño de la aplicación, es inevitable que en ocasiones se produzcan errores durante la ejecución del programa. Por ejemplo que como resultado de una operación aritmética a un dato le sea asignado un valor fuera de rango, que se pretenda realizar una división por cero o que se acceda a un "array" con un valor incorrecto del índice. En estas circunstancias, lo normal en la mayoría de los lenguajes de programación es que la aplicación finalice con un mensaje más o menos explícito, o incluso que el error no sea ni siquiera detectado. Por el contrario en Ada el error sería detectado y notificado a la aplicación mediante la generación de una excepción, la cual puede ser tratada de forma que se ejecuten las acciones correctoras pertinentes. Gracias a esta propiedad del lenguaje Ada, gran parte de los errores no detectados durante la fase de compilación, pueden ser detectados en tiempo de ejecución antes de que produzcan resultados catastróficos o impredecibles.

Las excepciones en Ada son el sistema preferido para la notificación de errores ya que constituyen un mecanismo muy potente que evita la necesidad de incluir numerosos chequeos en el código, además de permitir la separación del tratamiento de los errores del punto donde se producen, con la consiguiente mejora en la legibilidad del código. El mecanismo de excepciones ha demostrado ampliamente su validez, como lo prueba el hecho de que otros lenguajes posteriores al Ada, como son el Java o el C++, han adoptado y mejorado también este mecanismo de propagación de errores.

Servicios de tiempo real:

Otra característica destacable la constituye el hecho de que la concurrencia esté directamente soportada por el lenguaje, existiendo primitivas que permiten definir las actividades concurrentes, denominadas tareas, así como sus interacciones [2]. La planificación de las tareas se realiza mediante una política expulsora basada en prioridades con orden FIFO dentro de la misma prioridad. Se definen dos formas de interacción entre tareas: el envío de mensajes o "rendezvous" y la utilización de objetos protegidos. Estos últimos permiten el acceso exclusivo a datos compartidos mediante la utilización del protocolo de techo de prioridad inmediato. Tanto para la política, como para los mecanismos de sincronización elegidos se dispone de una amplia base teórica, lo que permite programar aplicaciones con tiempos de respuesta predecibles. Además este lenguaje ofrece paquetes en su implementación que facilitan notablemente la temporización como son el "*ada.calendar"* y el "*ada.real_time"* junto con tipos específicos que permiten definir instantes determinados (*time)* y espacios temporales (*Time_Span, Duration*).

Podemos concluir que Ada soporta los aspectos más avanzados de la ingeniería software, rama de la ingeniería que persigue la reducción de los costos de desarrollo y mantenimiento de las aplicaciones mediante el aumento de la fiabilidad del código, la reusabilidad de componentes y el establecimiento de métodos de trabajo disciplinados permitiendo reducir los costes de desarrollo de las aplicaciones . Ada 95 es un lenguaje apropiado para ser utilizado en muchos tipos de aplicaciones, aunque en la actualidad su más amplia utilización tiene lugar en aplicaciones empotradas de tiempo real con requisitos de alta seguridad, especialmente en el campo aerospacial. Un ejemplo de utilización de Ada 95 es la programación del Airbus 380 que podemos ver en la siguiente figura.

Figura 6: Imagen del Airbus 380 .

1.3 Descripción del hardware y de los elementos constructivos utilizados.

1.3.1 Chasis.

En el desasrrollo del presente proyecto se ha utilizado el chasis "Soccer Robo Body Set". Se trata de una plataforma especialmente diseñada para la realización de prototipos de robots de fútbol.

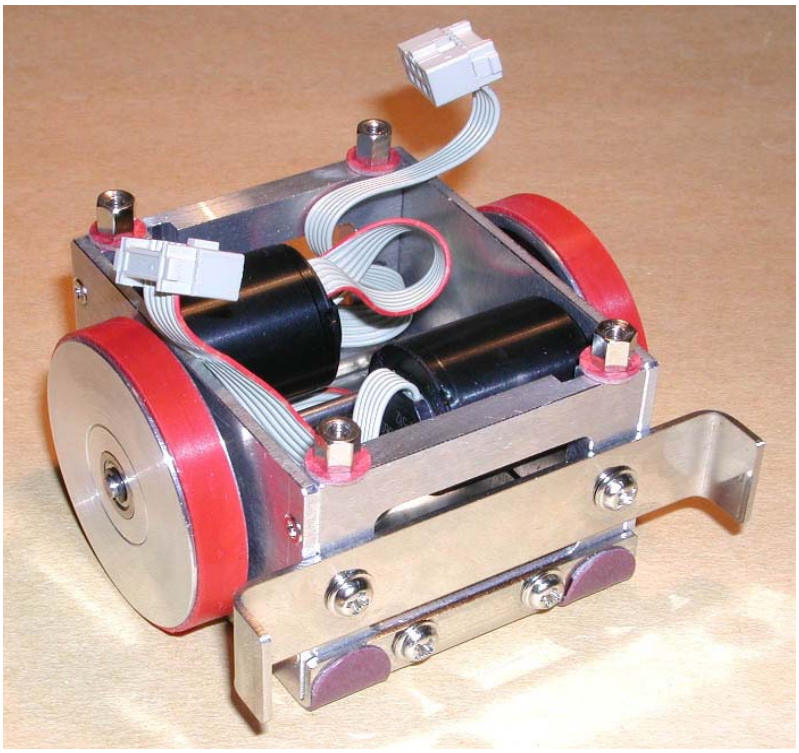


figura 7: Chasis Soccer Robo Body Set.

Su diseño y construcción satisfacen completamente las normativas establecidas por el más importante órgano regulador , la FIRA (Federation of International Robot-Soccer Association).[11].

Sus características más notables son:

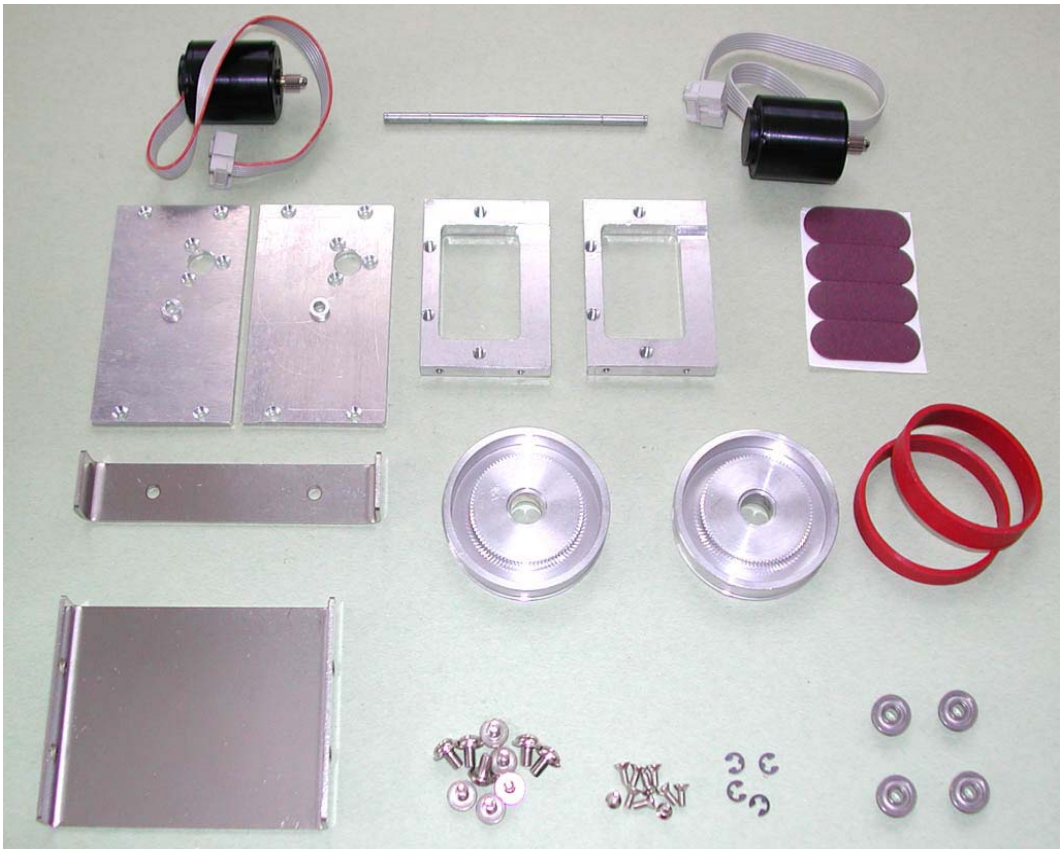


figura 8: Componentes del chasis.

- Motores de corriente continua con codificadores de posición (encoders) magnéticos.
- Dos ruedas independientes y engomadas para mejorar la tracción.
- Cuatro tiras deslizantes en la parte inferior para disminuir el rozamiento con el suelo.

- Cuenta con engranajes de alta precisión que permiten reducir la velocidad de salida del motor gracias a la relación de transformación 7.5:1. Esto se consigue a través de un engranaje interior a la rueda de 90 dientes y al piñón de 12 dientes a la salida del motor.

1.3.2 Motores.

Se utilizan motores de corriente continua. Su funcionamiento se basa en la acción de campos magnéticos opuestos que hacen girar el rotor (eje interno) en el sentido opuesto al del estator (imán externo o bobina).

Para modificar su velocidad se recurre a variar la tensión de alimentación. Pero de esta forma además de una reducción de velocidad conseguiremos una reducción del par de rotación (fuerza) lo cual puede perjudicar el uso del motor en determinadas condiciones de baja velocidad. Para mejorar este aspecto es deseable el uso de técnicas de control tales como la PWM (Pulse Width Modulation) o la PFM (Pulse Frecuency Modulation), que modifica la intensidad que se aporta al motor.

En el chasis Soccer Robo Body Set se han montado dos micromotores DC (Precious Metal Commutation) de la marca Faulhaber[9], concretamente el modelo 2224006 SR cuyos aspectos más relevantes pasamos a detallar a continuación:

- Tensión nominal : 6V.
- Potencia de salida (rotor): 4.55 W.
- Rendimiento: 80%.
- Velocidad sin carga: 8200 rpm.
- Consumo en vacío: 29 mA..
- Par de arranque: 16.9 mN*m.
-Relación entre velocidad y par : 387 rpm/mN*m.

Otras características reseñables y que son de gran importancia para la aplicación concreta a la que van a ser destinados estos motores son un reducido peso

(46 g), una baja inercia del rotor (2.7 g*cm^2) que le permiten alcanzar aceleraciones angulares máximas ciertamente elevadas (78 rad/seg^2).

Además el fabricante nos ofrece unas recomendaciones de utilización del motor:

- Velocidad máxima: 8000 rpm.
- Par máximo: 5mN*m.
- Consumo máximo: 1.2 A.

1.3.3 Encoders.

La función principal de los encoders es la de conocer la posición y/o velocidad de los motores. Por lo tanto está claro que juegan un papel fundamental en la realimentación del control de lazo cerrado ya que ellos mismos son quienes implementan dicho lazo de realimentación hacia la etapa de control.

El control mediante el uso de encoders aunque es más caro que el realizado por otros transductores (potenciómetros, tacogeneradores, transductores capacitivos) tiene múltiples ventajas ya que su implementación requiere un espacio muy reducido permitiendo controlar tanto posición como velocidad de una manera bastante precisa. Además, la creciente digitalización de la información hace de los encoders una herramienta muy adecuada para facilitar información a un PC.

Se ha utilizado el modelo IE2-512 de la casa Faulhaber[9]. Se trata de un encoder magnético que se encuentra unido al mismo motor, recibiendo de este último señales de flujo magnético .

Sus principales caracteristicas son las siguientes:

- Proporciona 512 pulsos por cada revolución del motor, así que teteniendo en cuenta la relación de transformación 7.5:1 el encoder nos señala cada vuelta de las ruedas con 3840 pulsos.
- La salida se efectúa por dos canales distintos.
- Se alimenta con una tensión de 5 V de continua.
- Tiene un consumo típico de 6 mA y máximo de 12 mA.
- Se trata de un circuito compatible TTL y CMOS.

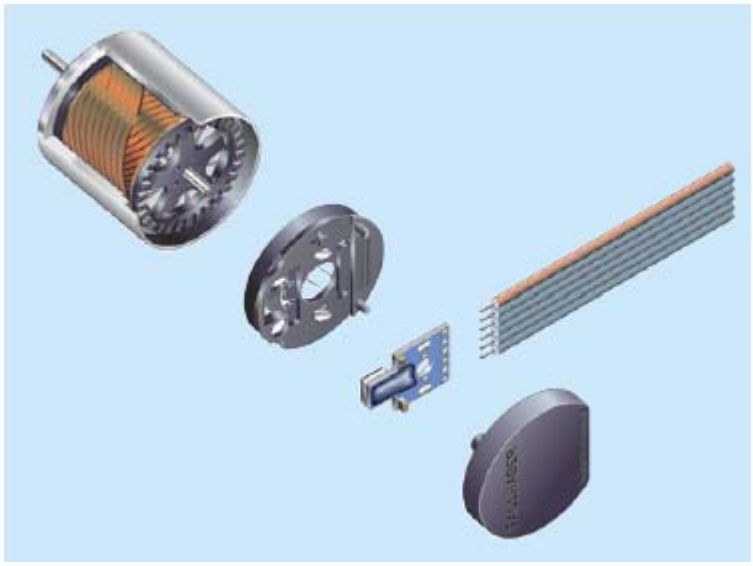


figura 9: estructura del motor junto al encoder.

Dentro de la serie IE2 se trata del encoder que más pulsos devuelve por cada vuelta del motor. En la siguiente figura podemos apreciar la comparación entre dos modelos, uno que proporciona 16 pulsos por revolución y otro que genera 512, siendo evidente que el último se aproxima mucho mejor al valor ideal.

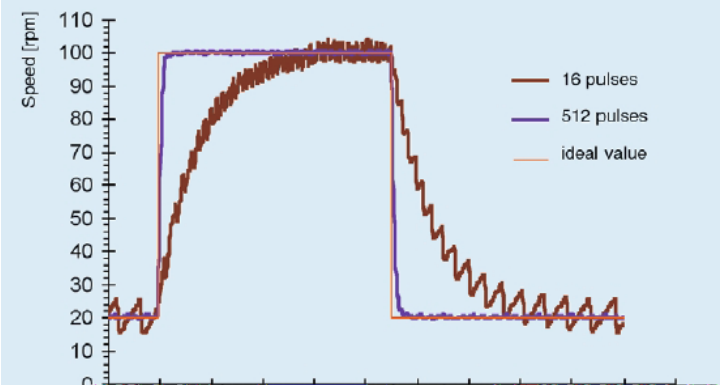


figura 10: Comparación entre encoders de distinta precisión.

De esta característica y de las anteriormente citadas cabe destacar la idoneidad del modelo utilizado debido a su elevada precisión., su reducido tamaño, un consumo limitado y a la producción de señales perfectamente adaptadas para una etapa de control basada en un PC.

Entrando más en detalle en las señales que genera este encoder se trata de dos ondas cuadradas de la misma frecuencia pero desfasadas ¼ de periodo. Señalar que la frecuencia de ambas es la misma para una misma velocidad de rotación del motor pero que aumentará o disminuirá en proporción a como lo haga la velocidad del motor según la siguiente relación:

$$\text{Velocidad motor (rpm)} = (f_{encoder}(HZ)*60)/512.$$

Si nos fijamos en los siguientes diagramas podemos ver la utilidad de la existencia de dos señales. Dicha ventaja radica en que podemos conocer con relativa facilidad el sentido de giro en cada momento. Simplemente hay que fijarse en una de las ondas cuadradas, más concretamente en el momento en que se produce un flanco,

por ejemplo ascendente. En ese preciso instante de tiempo la otra señal estará tomando un valor determinado (0 V ó 5V). Dependiendo del valor que tome esta última el motor estará rotando en un sentido u otro.

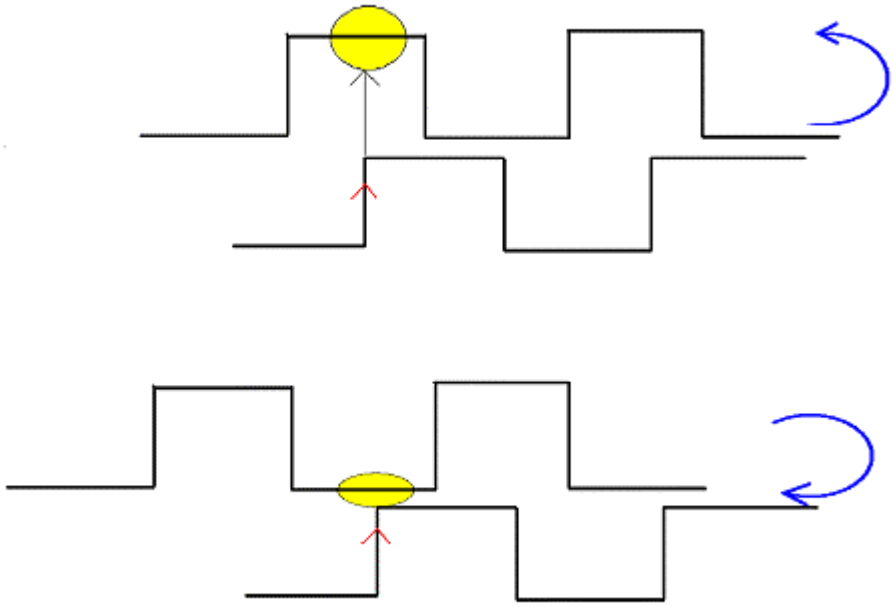


figura 11 : Establecimiento del sentido de giro a partir de las señales desfasadas.

1.3.3.1 Principio de funcionamiento del encoder.

Este encoder está fundamentado en un principio magnético por el cual pequeñas diferencias de flujo magnético son convertidas en voltajes por un sensor magnético especialmente desarrollado para este cometido. Es importante resaltar que se convierten diferencias de flujo con lo cual el sistema resultante es altamente insensible a las interferencias de otros campos.

Las señales a la salida del sensor son bastante débiles así que deben ser amplificadas. Una vez amplificadas pasarán por una red de resistencias donde se produce un efecto de multiplicación obteniéndose a la salida 16 señales desfasadas entre sí que todavía poseen carácter sinusoidal.

El siguiente escalón está formado por unos comparadores digitales a cuya salida las señales son ya cuadradas. Finalmente un circuito lógico facilita dos ondas cuadradas compatibles TTL y CMOS listas para ser analizadas por una computadora o similar.

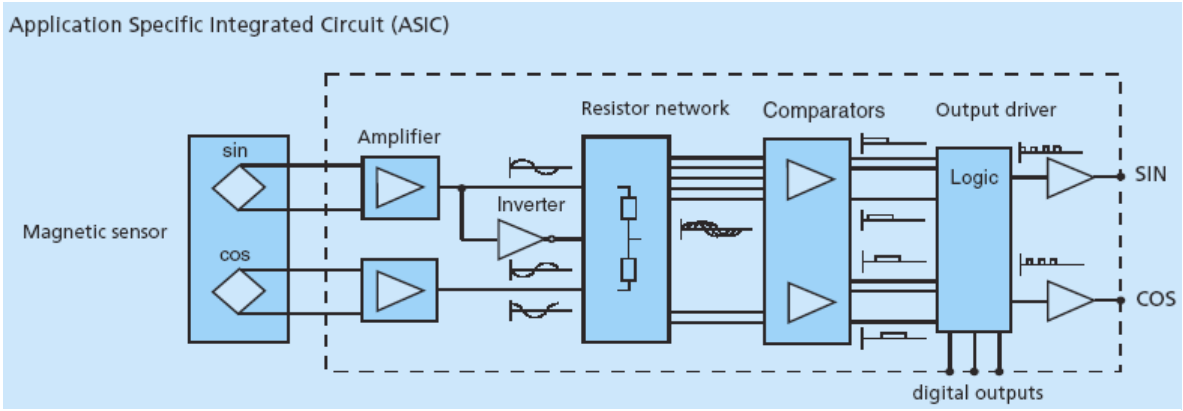


figura 12: Diseño electrónico del encoder.

1.3.4 Controlador.

1.3.4.1 Estándar pc104.

En este punto se dará una breve explicación sobre el estándar pc104, para darle una visión global al lector de las características y cualidades que posee este estándar hardware.

La arquitectura pc104 es un estándar utilizado en aplicaciones donde se quiere utilizar una plataforma PC de dimensiones reducidas y consumo mínimo. La

necesidad en la investigación y la industria de sistemas informáticos complejos en situaciones específicas han fomentado nuevos estándares para evitar que se hayan de implementar nuevos sistemas para cada uno de los casos. Estos estándares se engloban en los denominados sistemas empotrados, o más conocidos por su denominación inglesa “EMBEDDED SYSTEM”.

Entre las características que debe cumplir un sistema empotrado se encuentran la robustez (resistencia a los golpes u excesos de temperatura), minimización del tamaño y peso, bajo consumo y modularidad de los componentes (remodelación de sistema sin necesidad de grandes cambios).

Debido a la venta masiva de ordenadores con la arquitectura PC se ha generado una gran cantidad de software y hardware para este sistema El estándar pc104 se ha desarrollado específicamente para añadir la compatibilidad de la arquitectura PC en aplicaciones empotradas.

Aunque, a diferencia de los PC de sobremesa, los empotrados están pensados para una alta integración. PC/104 difiere de sistemas PC estándar en lo siguiente:

- Las tarjetas PC104 (90 x 96mm) son mucho más pequeñas que las tarjetas ISA. Se apilan una sobre otra mediante conectores pin /socket, lo que elimina la necesidad de placas base, backplane o chasis.

.

- Se reducen los requerimientos de alimentación (1-2 Watios por módulo) y manejo de señal (4mA) y minimiza la circuiteria.

- Además de esto, los sistemas PC104 están diseñados para ser más robustos que los sistemas PC.

Estos sistemas pueden ser programados con las mismas herramientas utilizadas con las PCs: APIs, compiladores, depuradores, herramientas de desarrollo, sistemas operativos y utilidades, lo que reduce la necesidad de conocimientos y el costo de un desarrollo personalizado.

Gracias a la compatibilidad PC se pueden utilizar sistemas de comunicaciones y almacenamiento ya diseñados. Añadiendo que los desarrolladores tienen experiencia en la arquitectura PC, se pueden reducir costes por el decremento del tiempo de diseño.

Aunque en la actualidad nos encontramos con procesadores muy potentes (K7, pentium3, etc.) lo habitual es encontrar procesador 486 o pentium en las arquitecturas pc104 que, comparados con los microcontroladores que se utilizan en la industria (intel 80C51 por ejemplo) poseen un poder de cálculo muy superior.

Los módulos pc104 se pueden conectar uno sobre el otro formando una pila de diferentes placas con propósito específico (CPU, tarjeta de adquisición, ethernet). Este sistema de expansión vertical permite una reducción de volumen en el sistema final por adquirir apariencia de cubo. Existen diversas formas de utilizar módulos PC/104:

- Un modulo PC/104 puede ser utilizado como un sistema independiente (stand–alone)
- Módulos PC/104 pueden ser añadidos como parte de otro sistema (cPCI, VME, etc)
- Múltiples módulos PC/104 pueden ser apilados entre sí para crear un sistema.

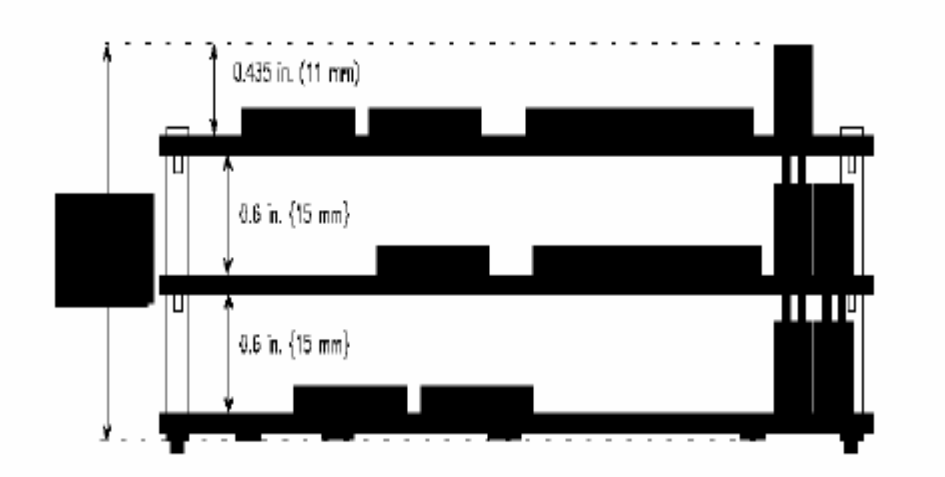


figura 13: Apilamiento de placas PC 104.

Los módulos PC/104 se encuentran comercialmente disponibles para un amplio rango de funciones, incluyendo:

- Tarjetas CPU compatibles con PC completas .
- Entradas / Salidas analógicas y digitales .
- Video: VGA, LCD, EL, Frame Grabbers .
- Redes: Ethernet, CAN bus, ARCNET .
- Controladores : FDD, IDE HDD, SCSI .

El bus pc104 es una versión compacta del bus (IEEE P996 PC PC/AT) optimizado para aplicaciones empotradas.

Existen dos tipos de módulos pc104 según su bus:

- Bus PC (8 bits) con un único conector de 64 pins.
- Bus PC/AT (16 bits) con dos conectores 64+40 (104 pins).

Debemos señalar la existencia de un nuevo estándar derivado. La especificación PC/104-plus define la adición de PCI a PC/104 incluyendo detalles de los conectores. El nuevo conector tiene 120 pines con espacio de 2mm.

PC/104-plus ha llegado justo a tiempo para controladores de vídeo, procesadores y otros dispositivos de alto rendimiento, manteniendo la compatibilidad hacia atrás con PC/104.

Appendix B. PC/104 Bus Signal Assignments

Pin Number	J1/P1 Row A	J1/P1 Row B	J2/P2 Row C[1]	J2/P2 Row D[1]
0	--	--	0V	0V
1	IOCHCHK*	0V	SBHE*	MEMCS16*
2	SD7	RESETDRV	LA23	IOCS16*
3	SD6	+5V	LA22	IRQ10
4	SD5	IRQ9	LA21	IRQ11
5	SD4	-5V	LA20	IRQ12
6	SD3	DRQ2	LA19	IRQ15
7	SD2	-12V	LA18	IRQ14
8	SD1	ENDXFR*	LA17	DACK0*
9	SD0	+12V	MEMR*	DRQ0
10	IOCHRDY	(KEY)[2]	MEMW*	DACK5*
11	AEN	SMEMW*	SD8	DRQ5
12	SA19	SMEMR*	SD9	DACK6*
13	SA18	IOW*	SD10	DRQ6
14	SA17	IOR*	SD11	DACK7*
15	SA16	DACK3*	SD12	DRQ7
16	SA15	DRQ3	SD13	+5V
17	SA14	DACK1*	SD14	MASTER*
18	SA13	DRQ1	SD15	0V
19	SA12	REFRESH*	(KEY)[2]	0V
20	SA11	SYSCLK	--	--
21	SA10	IRQ7	--	--
22	SA9	IRQ6	--	--
23	SA8	IRQ5	--	--
24	SA7	IRQ4	--	--
25	SA6	IRQ3	--	--
26	SA5	DACK2*	--	--
27	SA4	TC	--	--
28	SA3	BALE	--	--
29	SA2	+5V	--	--
30	SA1	OSC	--	--
31	SA0	0V	--	--
32	0V	0V	--	--

figura 14: Descripción de las señales del bus PC/104.

1.3.4.2 Descripción de la placa base.

En la realización del presente proyecto se ha utilizado una placa base con la especificación PC/104. Se trata del módulo PCM-3347 F de la compañía Advantech[13]. Esta placa proporciona la mayor parte de los servicios de un PC en un tamaño realmente reducido (90 * 96 mm con un peso de103 g.). Pasamos a realizar una descripción de sus aspectos más importantes:

- Cuenta con una CPU Elite a 133 MHz compatible con la arquitectura 486.
- Memoria SDRAM de 32 MB.
- Consumo típico de 1.62 A.
- 4 puertos serie y 1 puerto paralelo.
- Permite la conexión de disco duro y disquetera.
- Permite conexión de tarjetas de memoria del tipo Compact Flash
- Conectores para LCD y VGA.
- Tarjeta de ethernet RTL 8100 B 10/100 Mbps.
- Posibilidad de conectar dispositivos como ratón o teclado.
- Soporta protección por perro guardián.

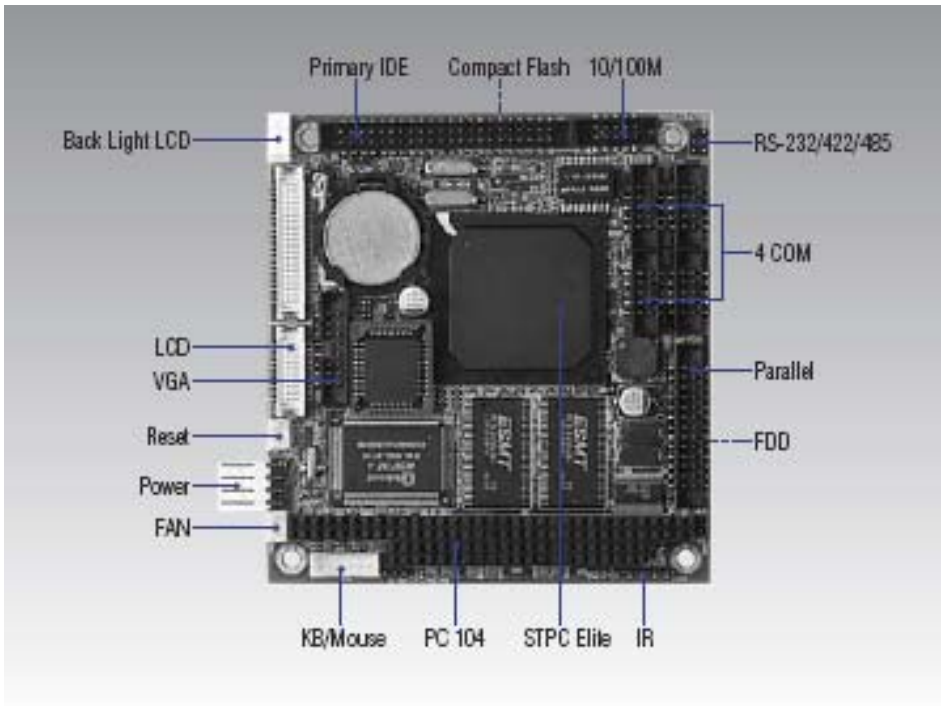


figura 15: Descripción PCM 3347 F.

1.3.5 Etapa de potencia.

En apartados anteriores hemos señalado las características principales de los motores y encoders. Asimismo se ha descrito a grandes rasgos el controlador.

El controlador es incapaz por sí mismo de suministrar la potencia necesaria a los motores para que puedan realizar su cometido. Se hace pues imprescindible una etapa intermedia que permita recibir las consignas de control por un lado y la alimentación por el otro de tal manera que a los motores llegue una señal que sea resultado de la combinación de las dos primeras.

Para tal cometido en la realización de este proyecto se ha utilizado el chip de potencia MMC (Motor Mind C) de la compañía Solutions Cubed. Se trata de un integrado de 40 pines con unas prestaciones que le hacen ser apropiado para su uso en pequeños robots. Algunas de estas prestaciones son:

- Facilita el control de velocidad y dirección de uno o dos motores de corriente continua. En el caso de dos motores el control se produce de manera independiente sobre cada uno de ellos.
- Está diseñado para alimentar motores cuyas tensiones nominales pueden ir desde los 6 V DC hasta los 35 V DC.
- Permite consumos de 1.5 A para cada uno de los motores pudiendo llegar a los 2 A. mediante la adopción de métodos de disipación.
- Ofrece la posibilidad de elegir entre modos diferentes de operación: modo serie, análogo y R/C. Estas tres configuraciones del MMC suponen tres maneras bien diferenciadas de generar y comunicar la señal de control desde el computador hacia el MMC. Más adelante se realizará una descripción más detallada de cada uno de ellos escogiendo el más apropiado para nuestros objetivos.

- Detecta picos elevados de corrientes y sobrecalentamiento, protegiendo de esta forma a los motores.

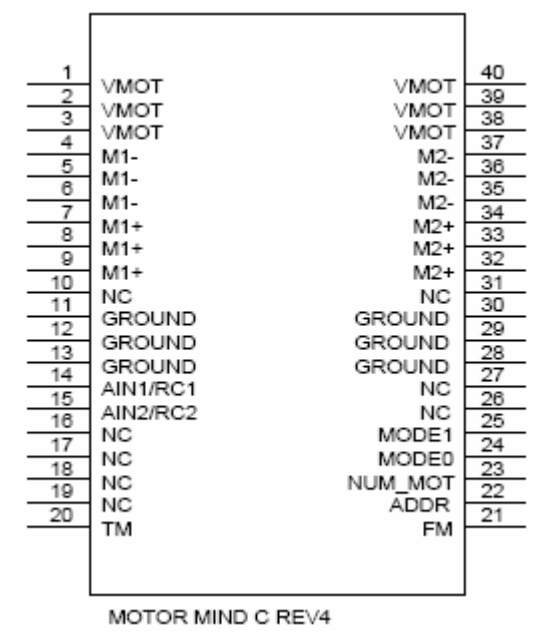


figura 16: Patillaje del MMC.

1.3.5.1 Principio de funcionamiento.

Este circuito permite realizar el control de la velocidad de los motores gracias al uso de la técnica PWM (Pulse Width Modulation).

Está formado por un microcontrolador PIC 16F73 y por dos puentes H de tiristores. Dependiendo de la señal de control a la entrada el microcontrolador adelantará o retrasará más o menos el disparo de los elementos de potencia (tiristores), variando de esta manera el ciclo útil de la señal de tensión que actúa sobre los motores.

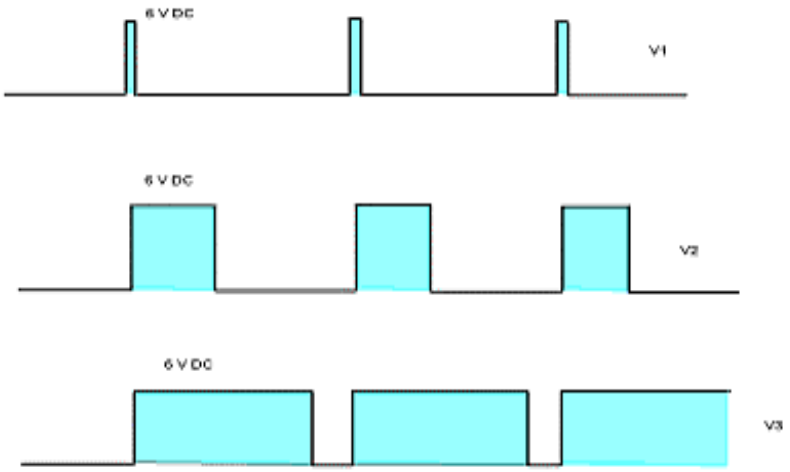


figura 17: La tension promedio aplicada varía con el ancho de pulso.

A los motores llegará una tensión cuadrada de amplitud máxima 6 V DC y frecuencia 1.2 KHz. Dependiendo del ciclo útil obtendremos una intensidad que podemos variar a voluntad. Todo esto se refiere a un mismo sentido de giro. Para que el motor cambie de dirección el MMC invertirá la polaridad de la tensión y modificará el ciclo útil como ya se comentó anteriormente.

1.3.5.2 Modos de funcionamiento.

Cada uno de las siguientes configuraciones permite el control independiente de velocidad y dirección en ambos motores.

MODO SERIE:

Su nombre deriva de la interfaz que se precisa para conectarse con el computador. Para esta conexión el MMC dispone de dos pines TTL, uno para la entrada de datos (FM) y otro para la salida de los mismos (TM).

La conexión establecida entre el PC y el chip de potencia es de tipo Master-Slave, siendo bidireccional pero no simultánea. Los datos que se manejan son de tamaño byte. Para regular este tránsito de datos se utiliza un protocolo específico de este circuito, MMCCP (Motor Mind C Communication Protocol) que define una serie de comandos y registros sobre los que es posible efectuar operaciones de lectura en unos caso, escritura en otros o ambas.

En este modo es posible elegir entre dos velocidades de transmisión de datos (9.6 Kbps ó 38.4 Kbps). También podemos modificar la frecuencia de la señal PWM que actúa sobre los motores eligiendo entre el valor estándar de 1.2 KHz ó el de 19.2 KHz. Sólo será posible esta elección configurando el MMC en el modo serie.

El control PWM que se realiza en este caso es de 10 bits lo cual supone que tendremos 1024 ‘escalones’ de velocidad en cada uno de los sentidos de rotación.

Aunque ofrece prestaciones que no obtendremos con las demás posibles configuraciones la complejidad del protocolo MMCCP limita notablemente las posibilidades de elección del modo serie.

MODO ANÁLOGO:

Consiste en suministrar una tensión entre 0 V y 5V al circuito MMC. Una tensión de 0V representa velocidad máxima en un sentido , 5V máxima en el sentido opuesto y 2.5 V mantienen el motor en reposo.

El MMC posee internamente un ADC (convertidor analógico digital) de 8 bits así que tendremos 256 “escalones” de velocidad para pasar de máxima velocidad en un sentido a máxima velocidad en el sentido opuesto, lo que representa un escalón de velocidad cada 19.53 mV . Internamente el controlador realiza la siguiente operación:

$$PWM = ((Vin/19.53\ mV) - 128) * 8$$

Para realizar esta configuración es preciso contar con una tarjeta con salida analógica, como es el caso de la PCM 3712 de Advantech.

MODO R/C:

Este procedimiento es el mismo que se usa habitualmente en el control de servomotores. Se trata de aplicar al MMC una señal cuadrada de 5V con un periodo aproximado de 20 ms (50 HZ). No es recomendable que se sobrepasen los 25 ms.

El ciclo útil de dicha señal deberá oscilar entre 1 ms y 2 ms. Con 1 ms se produce máxima velocidad en un sentido, con 2 ms máxima en el sentido opuesto mientras que 1.5 ms suponen el reposo del motor. Internamente el controlador realiza la siguiente operación:

$$PWM = (Tpulse\text{-}width/2.44us) - 615) * 6$$

Para la utilización de esta configuración es necesario contar con herramientas que permitan una adecuada temporización así como con una interfaz digital que comunique el computador con la etapa de potencia. Si se cumplen estos dos requisitos nos encontraríamos ante el modo que más resolución proporciona siendo también el que menos dificultad entraña tanto desde el punto de vista del hardware necesario (sobre todo el modo análogo), como desde la dificultad en el desarrollo de la programación (protocolo MMCCP en el modo serie).

1.3.5.3 Elección del modo de funcionamiento.

Una vez analizadas las posibilidades que ofrece el circuito MMC parece que la opción más acertada es la configuración en modo R/C. Las principales razones para su elección son las siguientes:

Desde el punto de vista de la programación evitamos la dificultad que supone el protocolo MMCCP que utiliza el modo serie.

A diferencia del modo análogo, no es necesario añadir una tarjeta con salida analógica, lo cual supone un abaratamiento de costes y disminuye el tamaño del prototipo.

En Ada 95 disponemos de herramientas de alta resolución para la gestión del tiempo que nos permitirán obtener mayor precisión que la que ofrecen tanto el modo analógico como el modo serie. La resolución de reloj en una plataforma como la nuestra está en torno a los 3 microsegundos, lo que para un rango de tiempo en el pulso RC de 1ms representa un error menor al 0.3%.

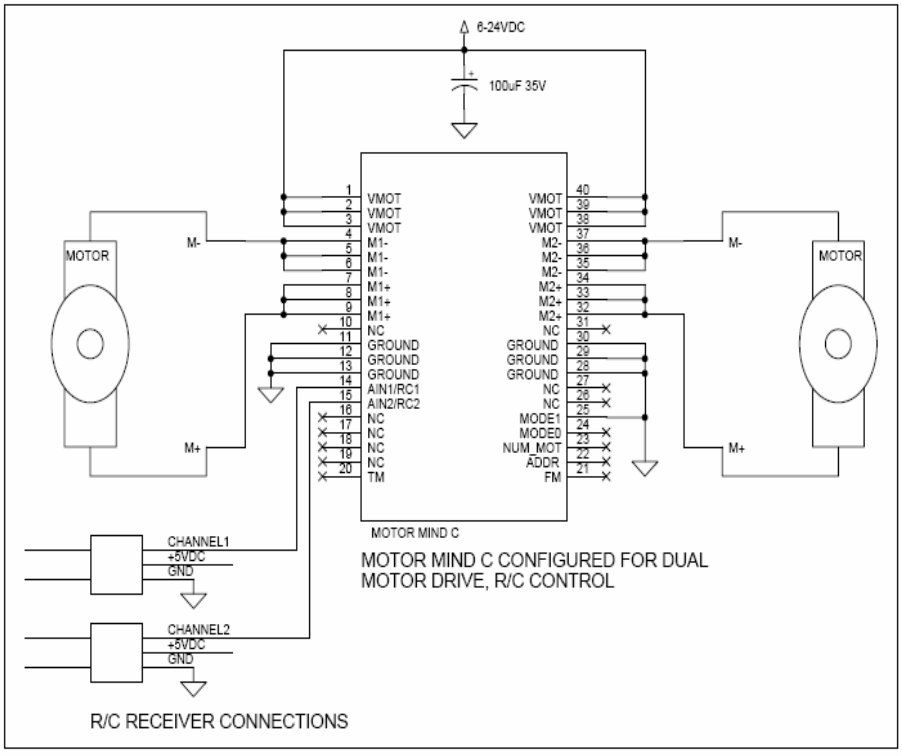


figura 18:Configuración para dos motores en el modo R/C.

1.3.6 Etapa alimentación.

En el diseño del prototipo se pueden diferenciar 2 zonas en cuanto a necesidades de voltaje se refiere.

La primera es la parte de los motores que funcionan a una tensión nominal de 6V DC con un consumo de 1.2 A en el peor de los casos. La segunda zona comprende la alimentación de la placa base y de los encoders que necesitan 5 V DC. Además esta tensión debe ser lo más estable posible sobre todo en el caso de la placa. En el peor de los casos la placa consume 2 A mientras que los encoders 12 mA cada uno.

Para los motores se han utilizado pilas recargables NiMH AA HR6 de 1.2 V cada una y capacidad de 2100 mAh. Para llegar a suministrar los 6 V se han colocado cinco de estas pilas en serie.

Para la alimentación de la electrónica se han combinado las pilas recargables con reguladores de tensión. Se utiliza el regulador 7805 CT. Se trata de un regulador fijo de 5 voltios que es capaz de mantener estable esta tensión suministrando una intensidad máxima de 1 A. A la entrada del regulador se requiere una tensión superior a los 6.3 V aproximadamente. Por este motivo se han empleado 6 pilas en serie que son capaces de establecer 7.2 V. Debido a que el consumo supera la máxima intensidad que es capaz de suministrar un solo regulador se han utilizado dos dispuestos en paralelo.

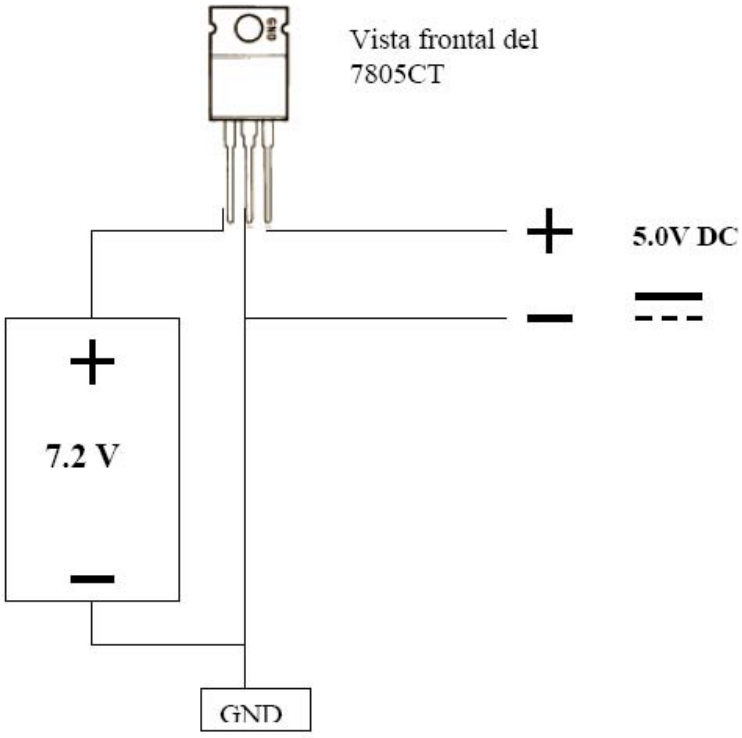


Figura 19: Configuración del regulador de tensión 7805CT.

1.3.7 Interfaz de comunicación con la etapa de control: el puerto paralelo.

Para implementar el control de lazo cerrado es necesario comunicar la etapa de control con el controlador de potencia (MMC) y realimentar la señal de los elementos sensores hacia la etapa de control.

Al controlador de potencia llegan 2 señales digitales mientras que de los encoders salen dos señales también digitales por cada uno de ellos. Se debe por tanto disponer de un dispositivo que ofrezca al menos dos salidas y cuatro entradas TTL.

La tarjeta PCM 3718 de Advantech cuenta con dos canales digitales de 8 bits de entrada/salida. Su utilización, además de la adicción de un nuevo piso a la estructura PC-104, supondría un incremento de los costes del robot.

Si se estudian las alternativas, se observa que en la misma placa del PC existen puertas I/O estándar como el puerto serie o el paralelo.[7,8] Este último cubre los requerimientos planteados anteriormente. Sus principales características son las siguientes:

- Dispone de 8 líneas de datos que pueden transmitir de manera bidireccional. Cuenta también con un conjunto de líneas unidireccionales, algunas de salida y otras de entrada, que se utilizan normalmente para mantener el protocolo de comunicación entre el PC y otros equipos.
- La información se maneja a través de tres registros mapeados en el bus I/O a partir de la dirección $ 378.
- En la siguiente figura se detallan los nombres de cada patilla así como el registro desde el que podemos acceder (D=Datos, C=Control, S=Status). También hemos señalado en color rojo las dos líneas de salida que realizan la trayectoria directa del control y en color azul las cuatro líneas que habilitan la realimentación.

DB25	Señal	Registro	Tipo	Activo	Sentido
1	Control 0	C0-	Salida	Bajo	Invertido
2	**Dato 0**	**D0**	**Salida**	**Alto**	**directo**
3	**Dato 1**	**D1**	**Salida**	**Alto**	**directo**
4	Dato 2	D2	Salida	Alto	directo
5	Dato 3	D3	Salida	Alto	directo
6	Dato 4	D4	Salida	Alto	directo
7	Dato 5	D5	Salida	Alto	directo
8	Dato 6	D6	Salida	Alto	directo
9	Dato 7	D7	Salida	Alto	directo
10	Estado 6	S6+	Entrada	Alto	directo
11	**Estado 7**	**S7-**	**Entrada**	**Bajo**	**Invertido**
12	**Estado 5**	**S5+**	**Entrada**	**Alto**	**directo**
13	**Estado 4**	**S4+**	**Entrada**	**Alto**	**directo**
14	Control 1	C1-	Salida	Bajo	Invertido
15	**Estado 3**	**S3+**	**Entrada**	**Alto**	**directo**
16	Control 2	C2+	Salida	Alto	directo
17	Control 3	C3-	Salida	Bajo	Invertido
18-25	Tierra				

Desde la BIOS se ha configurado el modo de funcionamiento del puerto pasando de SSP al EPP que permite alcanzar ratios desde 500KB hasta 2MB por segundo. En estas condiciones el puerto paralelo puede operar a los mismos niveles de rendimiento que una tarjeta ISA equivalente.[8].

1.3.8 Visión de conjunto.

Hasta ahora se ha detallado individualmente cada uno de los elementos hardware utilizados. En la siguiente figura podemos ver un diagrama explicativo de cómo se interconectan entre ellos.

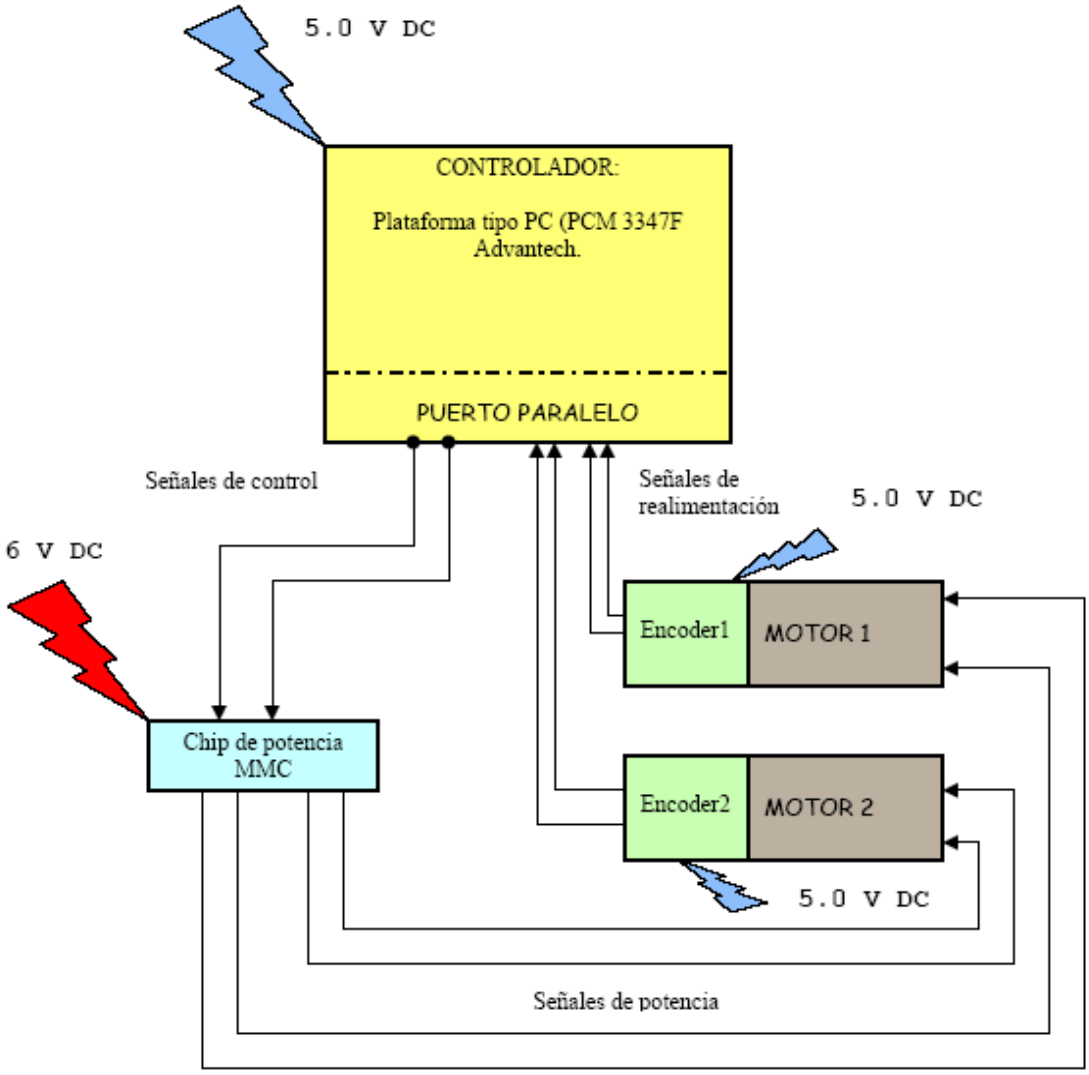


Figura 20: Visión de conjunto de interconexión de los elementos hardware.

El controlador basado en una plataforma PC se encarga de realizar el control del conjunto. Por un lado genera las señales de control necesarias sobre el circuito de potencia MMC usando el modo RC (pulsos de anchura entre 1 y 2ms, cada 20ms) y por otro recibe las señales de realimentación de los codificadores de posición (2 trenes de pulsos desfasados), todo ello a través del puerto paralelo.

El chip MMC se encarga de suministrar la potencia a los motores en función de las señales de control que le sean enviadas. Esto lo realiza mediante señales PWM.

Por otro lado los encoders envían las señales de realimentación al controlador que las utilizará para calcular nuevas señales de control en combinación con los datos de entrada del sistema.

1.4 Realización práctica y programación.

1.4.1 Configuración del entorno de desarrollo.

1.4.1.1 Instalación de Linux, Gnat y MaRTE OS.

Como ya se comentó en capítulos anteriores el entorno de desarrollo de las aplicaciones sobre MaRTE OS requiere un computador principal (Host) donde se escriben y compilan las aplicaciones, y otro computador (Target) donde se descargan y ejecutan.

En el host se ha instalado un sistema operativo de propósito general como es Linux (Fedora Core 3.0). Seguidamente se ha descomprimido e instalado el compilador Gnat para Ada (GAP 1.0) necesario para compilar el código de MaRTE OS durante su instalación[7]:

```
$ tar –zxvf gnat-3.14p-i686-pc-linux-gnu-bin.tar
$ cd gnat-3.14p-i686-pc-linux-gnu-bin
$ ./doinstall
```

Finalmente se procede a descomprimir e instalar el sistema operativo MaRTE OS (versión1.5.1):

```
$ tar –zxvf marte1.5.1.tgz
$ cd marte/
$ ./minstall
```

Durante la instalación se piden varios datos, entre ellos cabe destacar la dirección IP del Host y el directorio donde el programador quiere que se ubique el

archivo *"mprogram"*. Este archivo es el resultado de compilar la aplicación y enlazarla con las librerías de MaRTE OS.

Una vez finalizada la configuración del host es necesario habilitar mecanismos de comunicación para que *"mprogram"* pueda ser ejecutado en el target. Durante el desarrollo de este proyecto se ha descargado la aplicación de 3 maneras distintas que serán brevemente descritas a continuación.

1.4.1.2 Descarga por red con etherboot.

Para implementar este método es necesario que tanto el host como el target dispongan de tarjeta de red.

En el target tendremos conectada una disquetera en la cual se encuentra una imagen ROM (*eb-5.0.8-rtl8139.lzdsk*) que ha sido obtenida en la página *rom-o-matic.net*. .

En el host se ejecuta el script *mexport* con los privilegios de superusuario. Posteriormente se configura el servidor DHCP ('*etc/dhcp.conf*') con las direcciones IP de ambos computadores, la MAC del target, la ruta de acceso a '*mprogram*' y algunos otros parámetros. Además se activa el demonio NFS.

Cuando el target arranca el disquete de etherboot busca un servidor DHCP a través de la red. Cuando lo encuentra, a partir de los datos de configuración ('*dhcp.conf*') descarga '*mprogram*' y este se ejecuta en el target.

1.4.1.3 Descarga por LAN con PXE +etherboot.

PXE (Preboot Execution Environment) de Intel es una especie de arranque inteligente a partir de una memoria de sólo lectura (ROM) que se encuentra en algunas placas bases y tarjetas de red.

Este procedimiento de descarga ha sido desarrollado en la realización de un proyecto paralelo a este. La principal ventaja es que se puede prescindir de la disquetera, siendo además la descarga más rápida.

La idea es que PXE carga primero etherboot (‘mprogram’ es muy grande) y después etherboot carga ‘mprogram’. La descarga de etherboot requiere además del protocolo DHCP, el uso de un protocolo de transferencia de ficheros TFTP. La descarga de ‘mprogram’ se produce mediante los protocolos DHCP y NFS.

- Se debe acceder a la BIOS para configurar el arranque por LAN.
- Al arrancar se pulsa SHIFT+F10 para configurar el arranque y se escoge:
    - PXE.
    - Int/9h.
    - Enabled.
- La imagen etherboot obtenida debe tener habilitada la característica PXE.
- Se configura DHCP y se habilitan NFS y TFTP.

1.4.1.4 Descarga usando GRUB en una Compact Flash (CF).

El uso de la CF permite ejecutar una aplicación tantas veces como se quiera sin la necesidad de estar conectado al host en el momento del arranque.

- Se debe crear una partición FAT16, con la marca ‘bootable’ mediante */sbin/fdisk/dev/sdb.* (Si no se conoce el nombre asignado por el sistema para el dispositivo se puede usar el programa Kwikdisk).

- Se crea un sistema de ficheros en la partición recién creada mediante */sbin/mkdosfs/dev/sdb1*.
- En esta partición se copian los archivos de *boot/grub:*
    - *Stage1*.
    - *Stage2.*
    - *Menu.lst.*
    - *Grub.conf.*

- El siguiente paso es copiar '*mprogram*' del host en la CF. Ya está lista para ser insertada en el target .
- También será necesario modificar en la BIOS el primer dispositivo de arranque, además de desactivar el error por ausencia de keyboard.

1.4.2 Principio de generación de trayectorias.

En el capitulo 1 se estableció como principal objetivo el dotar al robot de un control de trayectorias que utilizase una configuración de lazo cerrado que facilite la corrección de los posibles errores.

Las trayectorias a realizar serán las básicas: rectilíneas hacia adelante, rectilíneas hacia atrás y curvilíneas con distintos radios de curvatura. Todas ellas se podrán realizar a distintas velocidades de avance.

Se ha considerado tomar como datos de entrada al sistema por un lado la velocidad lineal de avance de la plataforma (m/seg) y por otro la velocidad angular de rotación de la plataforma sobre su propio eje de giro (rad/seg).

Se realizará una suma de ambas magnitudes siendo la velocidad angular la que permita realizar trayectorias curvilíneas de distinto radio de giro.

Al tratarse de un sistema formado por dos ruedas independientes la adición de ambas velocidades resultará finalmente en la existencia de dos vectores de velocidad lineal, uno para cada rueda, de igual magnitud en trayectorias rectilíneas y de distinta en trayectorias curvilíneas.

Para realizar esta suma de velocidades debemos tener en cuenta la distancia entre las ruedas de tal manera que:

$$V1 = V + V' = V + \omega * (d/2).$$

$$V2 = V - V' = V - \omega * (d/2).$$

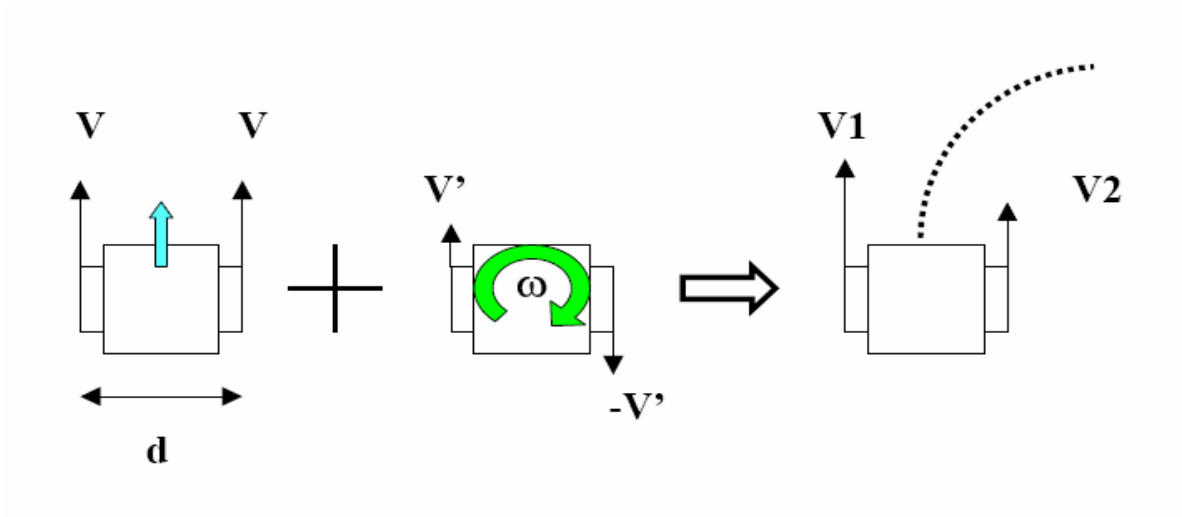


figura 21: Efecto de la suma de velocidad lineal y angular.

Además de tener en cuenta la suma de las velocidades lineal y angular tenemos que introducir un tercer dato: el tiempo. Conociendo el tiempo es posible establecer el final de una trayectoria de tal manera que no pueda el robot realizarla de manera indefinida.

En el siguiente gráfico podemos apreciar las distintas posibilidades ante las que nos podemos encontrar dependiendo de las magnitudes y signos que tomen las variables velocidad lineal y velocidad angular.

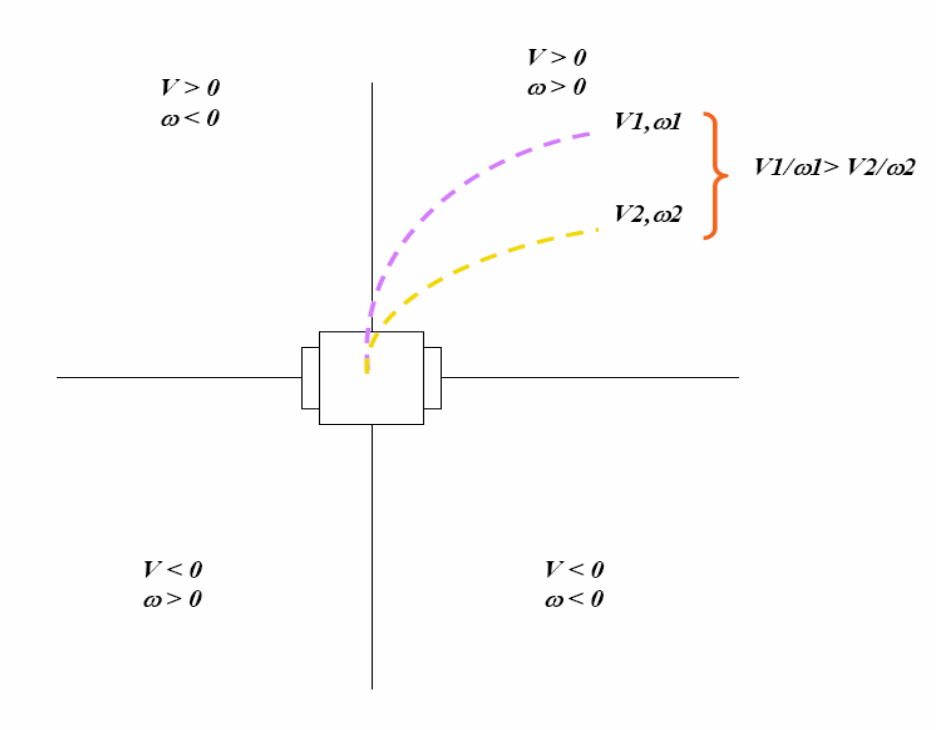


figura 22: Distintas zonas en función del signo de los datos de entrada.

1.4.3 Esquema general de programación.

El lenguaje de programación utilizado es Ada95. Como ya se comentó en anteriores capítulos se trata de un lenguaje especialmente concebido para el uso en sistemas de tiempo real debido a su legibilidad, modularidad y sobre todo a la fiabilidad de las aplicaciones que genera.

Se trata de un lenguaje que soporta concurrencia, permitiendo la ejecución paralela de varias tareas (*tasks*) sobre un único procesador, utilizando una política de

tareas expulsoras por prioridad fija[2]. La concurrencia aumenta el grado de modularidad y permite un diseño más claro ya que representa el modelo más natural para resolver problemas del mundo real que son de naturaleza concurrente.

Las tareas comparten datos utilizando *objetos protegidos*, que proporcionan acceso mutuamente exclusivo mediante funciones, procedimientos o puntos de entrada protegidos. Los procedimientos protegidos no pueden ser interrumpidos por otros procedimientos o funciones protegidas del mismo objeto[3].

Ada95 proporciona varias posibilidades para la gestión del tiempo. El paquete *Ada.Real_Time* permite acceder a un reloj monótono que se incrementa siempre, permitiendo saber la hora desde un instante determinado lo cual aumenta la fiabilidad de las aplicaciones de tiempo real. Dentro de este paquete se definen los tipos *Time* (tiempo absoluto) y *Time_Span* (tiempo relativo).

También es posible suspender una tarea durante un periodo de tiempo (instrucción *delay*) o hasta un instante determinado (instrucción *delay until*), con lo cual otras tareas podrán ejecutarse mientras dure la suspensión.

Para el control de trayectoria del robot en el presente proyecto se utilizan tres tareas y dos objetos protegidos tal y como se aprecia en la siguiente figura.

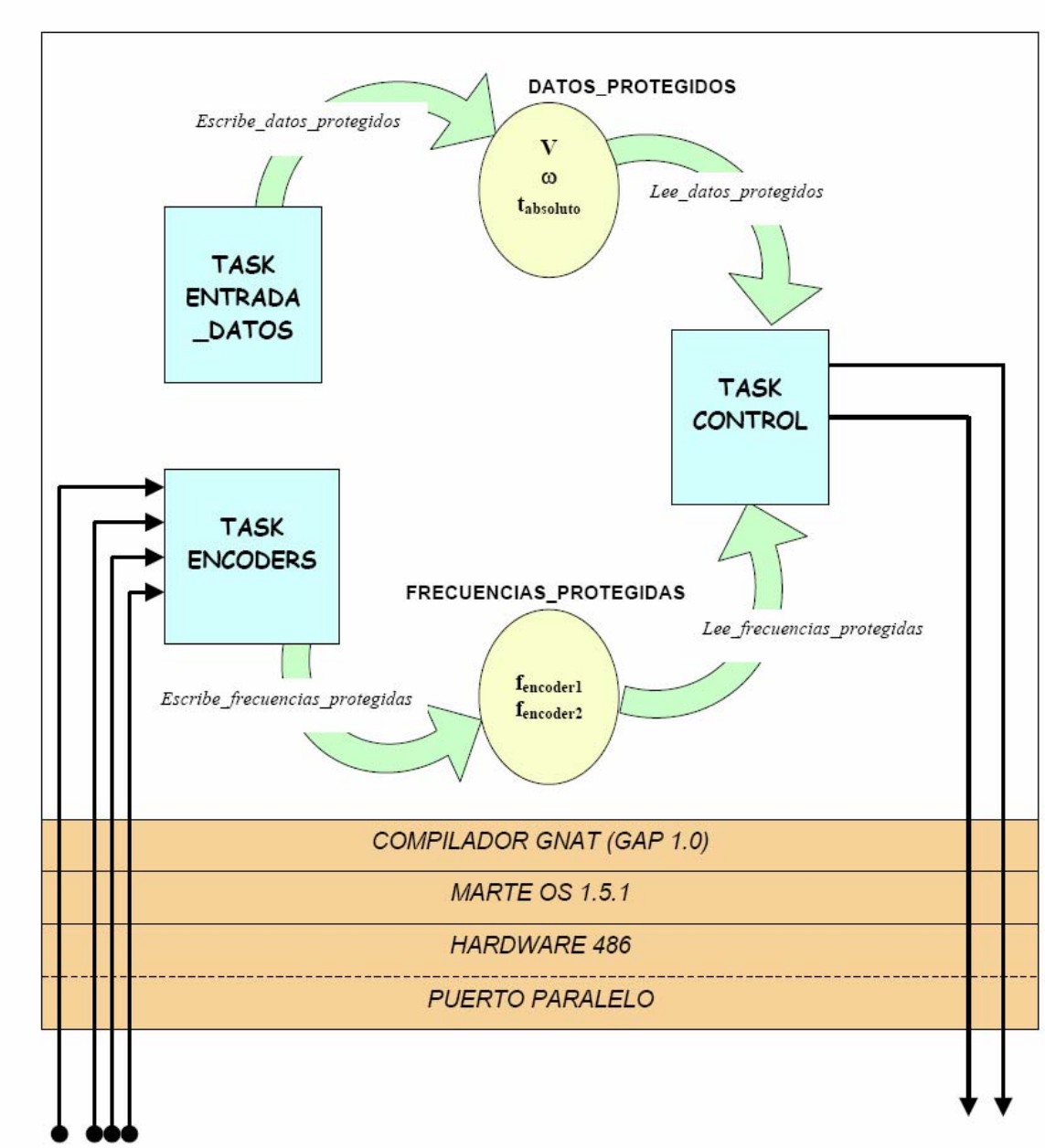


figura 23: Esquema general de la programación.

En la siguiente tabla podemos ver la asignación de prioridades que se ha realizado a cada una de las tareas y de los objetos protegidos en función de sus periodos.

	Tarea Entrada_Datos	*Tarea Control*	*Tarea Encoders*	*Datos_Protegidos*	*Frecuencias_Protegidas*
Periodo	~seg	~mseg	Polling		
Prioridad	9	10	8		
Techo prioridad				10	10

La primera tarea (*Entrada_datos*) nos va a servir para realizar una demostración del funcionamiento del prototipo. Su cometido es el de simular la aportación de datos por parte de un control superior. En un posterior desarrollo del prototipo será otra tarea la cual a partir de la necesidad de trazar una determinada trayectoria genere las entradas (velocidad lineal, velocidad angular y tiempo) que sean capaces de satisfacer los anteriores requerimientos.

Estos datos se escriben en un objeto protegido (*Datos_Protegidos*) mediante una llamada al procedimiento *Escribe_Datos_protegidos*.

La segunda tarea (*Control*) se encarga de acceder por una parte al objeto *Datos_Protegidos* mediante el procedimiento *Lee_Datos_Protegidos*, y por otra al objeto *Frecuencias_Protegidas* usando el procedimiento *Lee_Frecuencias_Protegidas.*

Mediante la conjunción de los datos de entrada y de los de la realimentación esta tarea genera a través del puerto paralelo las señales de control sobre la etapa de potencia.

La tercera tarea (*encoders*) se encarga de leer a través del puerto paralelo las señales de los elementos sensores y escribirlas en el objeto *Frecuencias_Protegidas* mediante el procedimiento *Escribe_Frecuencias_Protegidas.*

A continuación se plantean brevemente las especificaciones de los objetos protegidos:

```
**********************************************************************

with Ada.Real_Time; use Ada.Real_Time;
with Tipos_de_datos;use Tipos_de_datos;

package Paquete_Datos_Protegidos is

   protected dato is
      -- techo de prioridad
      pragma Priority (10);

      procedure Lee_datos_Protegidos(Datos: out
      registro_velocidades);
      procedure Escribe_datos_Protegidos (datos:in
      registro_velocidades);

      --

   private
      -------------------------------------
        datos_protegidos:registro_velocidades:=(0.0,0.0,clock);
      -------------------------------------

   end dato;
end Paquete_Datos_protegidos;
```

```
**********************************************************************

with Ada.Real_Time; use Ada.Real_Time;
with Tipos_de_datos;use Tipos_de_datos;
```

```
package paquete_frecuencias_protegidas is

   protected frecuencia is
      -- techo de prioridad
      pragma Priority (10);

            procedure Lee_frecuencias_Protegidas(frecuencias: out
                      registro_frecuencias);
            procedure Escribe_frecuencias_Protegidas (frecuencias:in
                      registro_frecuencias);

      --

   private
      ------------------------------------
        frecuencias_protegidas:registro_frecuencias:=(0.0,0.0);
      ------------------------------------

   end frecuencia;
end Paquete_Frecuencias_protegidas;

*********************************************************************
```

En el paquete *tipos_de_datos.ads* se han definido algunos de los tipos que se usarán en la programación de este proyecto:

```
package Tipos_De_Datos is


   type registro_velocidades is record
        Velocidad_lineal:float;
        Velocidad_angular:float;
        Tiempo_Max:time;
   end record;
```

```
   type registro_frecuencias is record
        frecuencia1:float;
        frecuencia2:float;
   end record;


   type Dato_Entrada is record
       Velocidad_Lineal:Float;
       Velocidad_Angular:Float;
       Tiempo_Max:Time_Span;
       Espera_Hasta:Time_Span;
   end record;

   type Datos_Entrada is array(1..100) of Dato_Entrada;


end Tipos_De_Datos;
```

1.4.4 Tarea Entrada_Datos.

Los datos se encuentran almacenados en una variable del tipo *Datos_Entrada* cuya especificación se ha definido anteriormente. Cada dato de este array está formado por un registro de cuatro campos. Los tres primeros campos son la velocidad lineal, la velocidad angular y el tiempo límite de ejecución de un segmento de la trayectoria. El cuarto campo se utiliza para indicar cuándo se deberá enviar el siguiente dato. Se ha planteado un bucle que va recorriendo el array al ritmo que marca el campo *espera_hasta.*

```
.....
.....
Now:=Clock;

loop
```

```
         Velocidades.Velocidad_lineal:=Datos(I).Velocidad_lineal;
         Velocidades.Velocidad_angular:=Datos(I).Velocidad_angular;
         velocidades.Tiempo_Max:=Now+Datos(I).Tiempo_max;
         dato.Escribe_Datos_Protegidos(velocidades);
         Now:=Now+Datos(I).Espera_hasta;

            if I=Max_Pruebas then
               I:=0;
            end if;
            I:=I+1;
            delay until Now;

 end loop;
.....................................
.....................................
```

1.4.5 Tarea Control.

Su cometido final es el de calcular y enviar las señales a la etapa de potencia accediendo para ello al puerto paralelo. Como ya se expuso con anterioridad estas son señales TTL con un periodo de 20 ms y con un ciclo útil que puede variar entre 1 ms (máxima velocidad en un sentido) y 2 ms (máxima velocidad en el otro sentido).

Para el cálculo del ancho de los pulsos se utilizan los datos leídos de ambos objetos protegidos:

- De *Datos_Protegidos* se obtiene la velocidad teórica de cada rueda a partir de la velocidad lineal (m/seg) y la velocidad angular (rad/seg) teniendo en cuenta que 0.032 es la mitad de la distancia entre las ruedas del chasis expresada en metros:

  *Velocidad_teórica1=velocidad_lineal+0.032 * velocidad_angular.*
  *Velocidad_teórica2=velocidad_lineal-0.032 * velocidad_angular.*

- De *Frecuencias_Protegidas* obtenemos la frecuencia de la señal de cada encoder. En este dato ya viene incluido el sentido de giro dependiendo del signo, positivo para velocidades positivas (hacia delante) y negativo para velocidades negativas (hacia atrás). A partir de estas frecuencias se obtiene la velocidad real de giro de cada rueda teniendo en cuenta que:
  - $Velocidad_{motor}(rps)=f_{encoder}/512$.
  - Relación de transformación motor a rueda = 7.5 : 1.
  - Avance de 1 vuelta de la rueda = 0.136 m.

  *Velocidad_real1=(frecuencia1/512) * (0.136/7.5) (m/seg).*
  *Velocidad_real2=(frecuencia2/512) * (0.136/7.5) (m/seg).*

- Se calcula el error de velocidad como la diferencia entre el valor teórico y el valor real:

  Error_velocidad1=velocidad_teórica1-velocidad_real1.
  Error_velocidad2=velocidad_teórica2-velocidad_real2.

- Se establece a partir del error de velocidad un factor de corrección o regulador proporcional que servirá para acercar el valor real al valor teórico:

  *Velocidad_corregida1=velocidad_teórica1+Kp1*Error_velocidad1.*
  *Velocidad_corregida2=velocidad_teórica2+Kp2*Error_velocidad2.*

- Para transformar la velocidad en ancho de pulsos tenemos en cuenta que en 0.50 ms (desde 1.5 ms hasta 2 ms) se pasa de velocidad nula a máxima. La máxima velocidad del motor es 8200 rpm, así que aplicando la relación de transformación y conociendo el avance de una vuelta de las ruedas concluimos que la máxima velocidad teórica que se puede

alcanzar es 2.47 m/seg. Por lo tanto asumiendo que la relación entre el ancho de los pulsos y la velocidad de los motores es lineal obtenemos:

Relación transformación velocidad a pulsos = 0.5/2.47 = 0.20243 $ms/_{(m/seg.)}$.

*Pulso1= 1.5 + 0.20243*Velocidad_corregida1. (ms).*

*Pulso2= 1.5 + 0.20243*Velocidad_corregida2. (ms).*

La tarea de control se encarga de generar pulsos de las anchuras calculadas por las dos primeras líneas de datos del puerto paralelo, una para cada motor. Un primer intento consistió en generar el flanco ascendente a la vez para los dos, hacer una comparación para saber cual es el menor, bajar este primero y después el mayor, tal y como se aprecia en la siguiente figura:

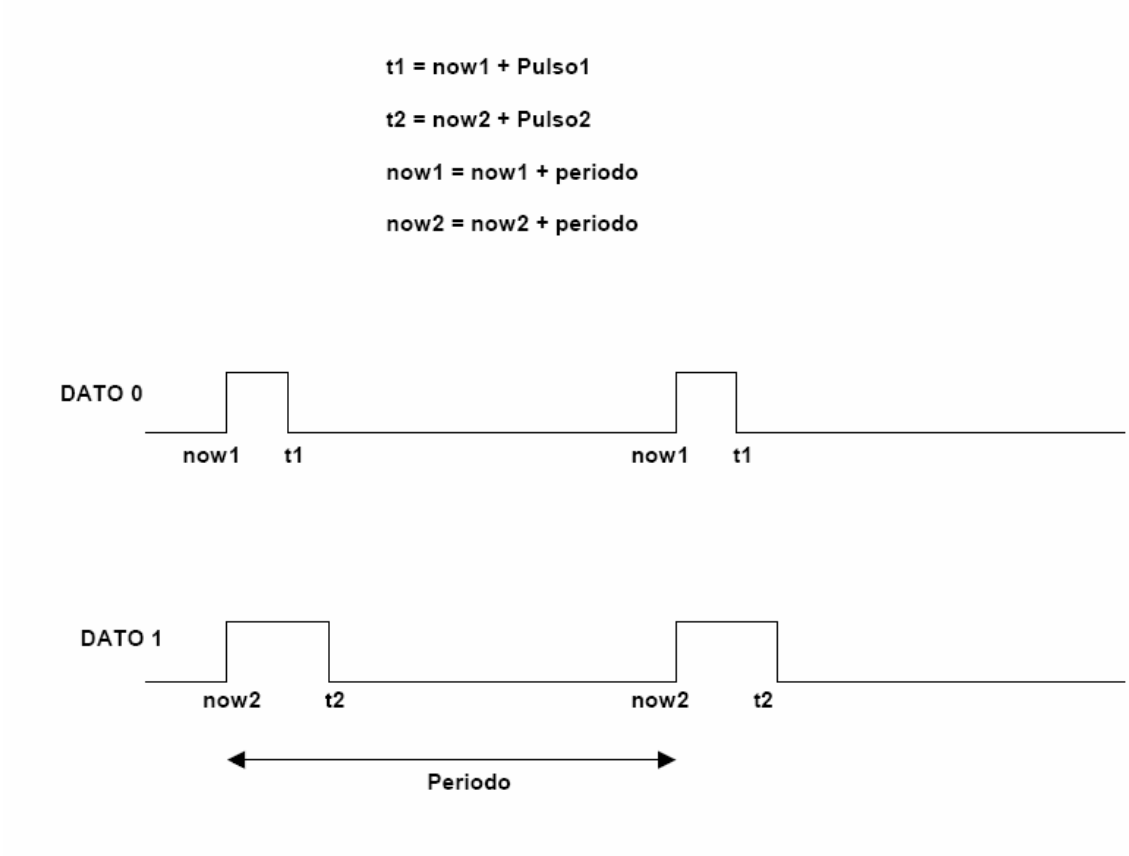


figura 24: Generación de pulsos sin desfasar.

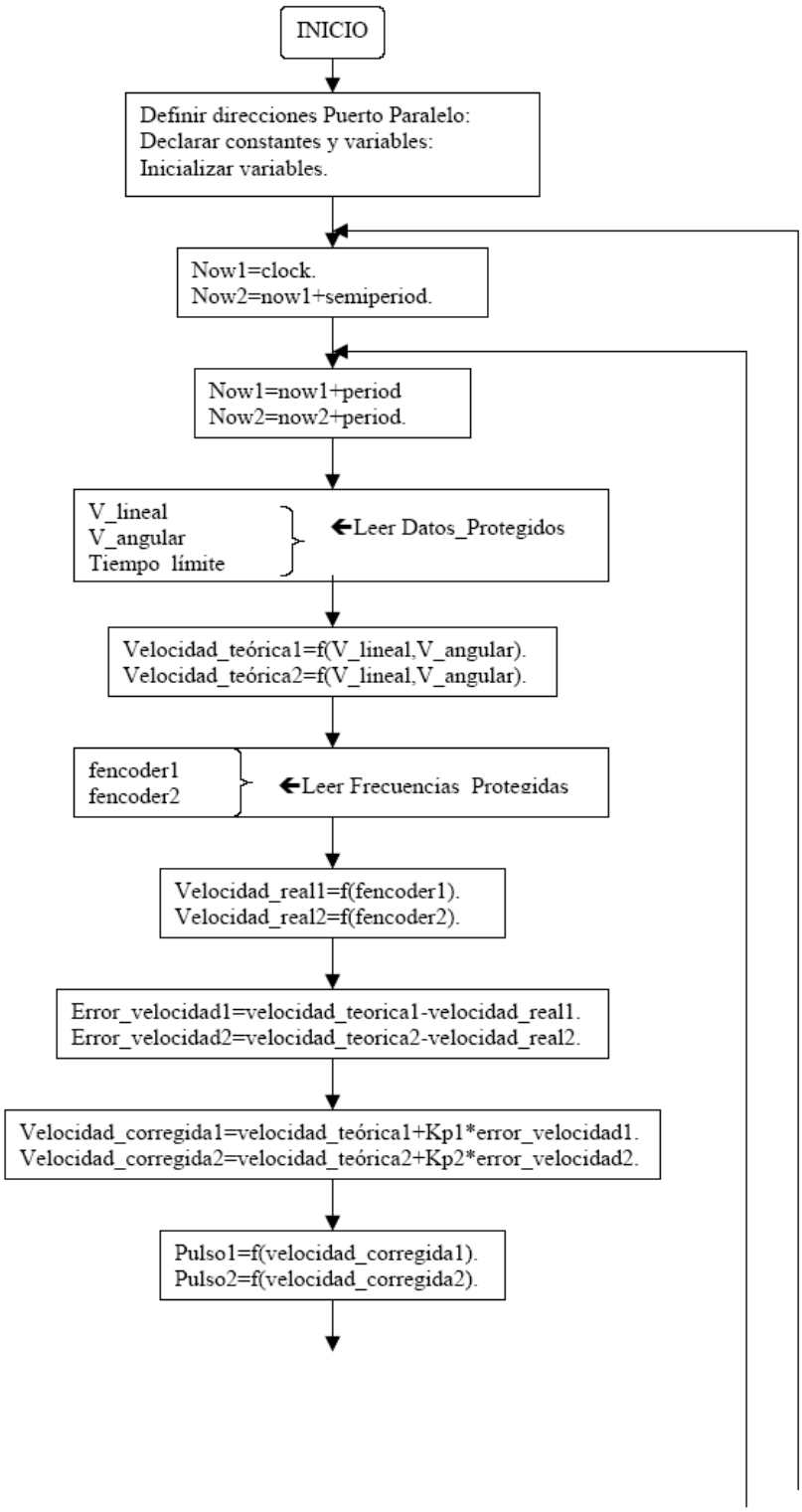
Pero esta posibilidad no resultó nada precisa ya que la bajada del segundo pulso presentaba errores aleatorios cuando la diferencia entre ambos pulsos era inferior a 0.38 ms. Estos errores eran impredecibles ya que en ocasiones la bajada se producía más tarde de lo deseado mientras que en otras lo hacía con antelación.

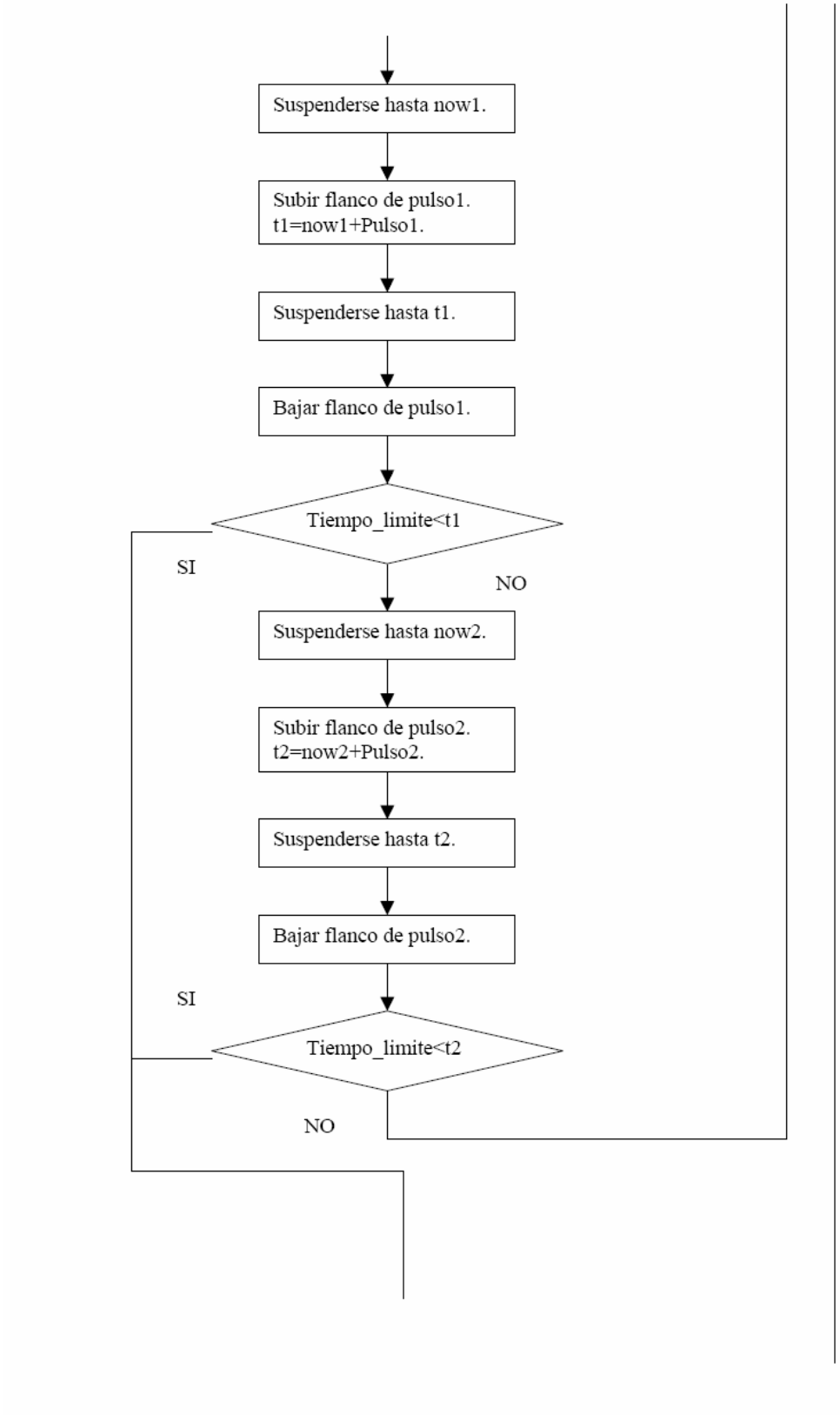
Para solucionar esto se desfasan las señales un semiperiodo (10 ms). De esta forma se comprueba que no se producen errores apreciables y además se evita hacer la comparación entre pulsos que era necesaria con anterioridad. Se miden las señales y se detecta un error máximo de +/- 5 us., lo cual supone un error de velocidad despreciable ($2.47 * 10^{-5}$ m/s).

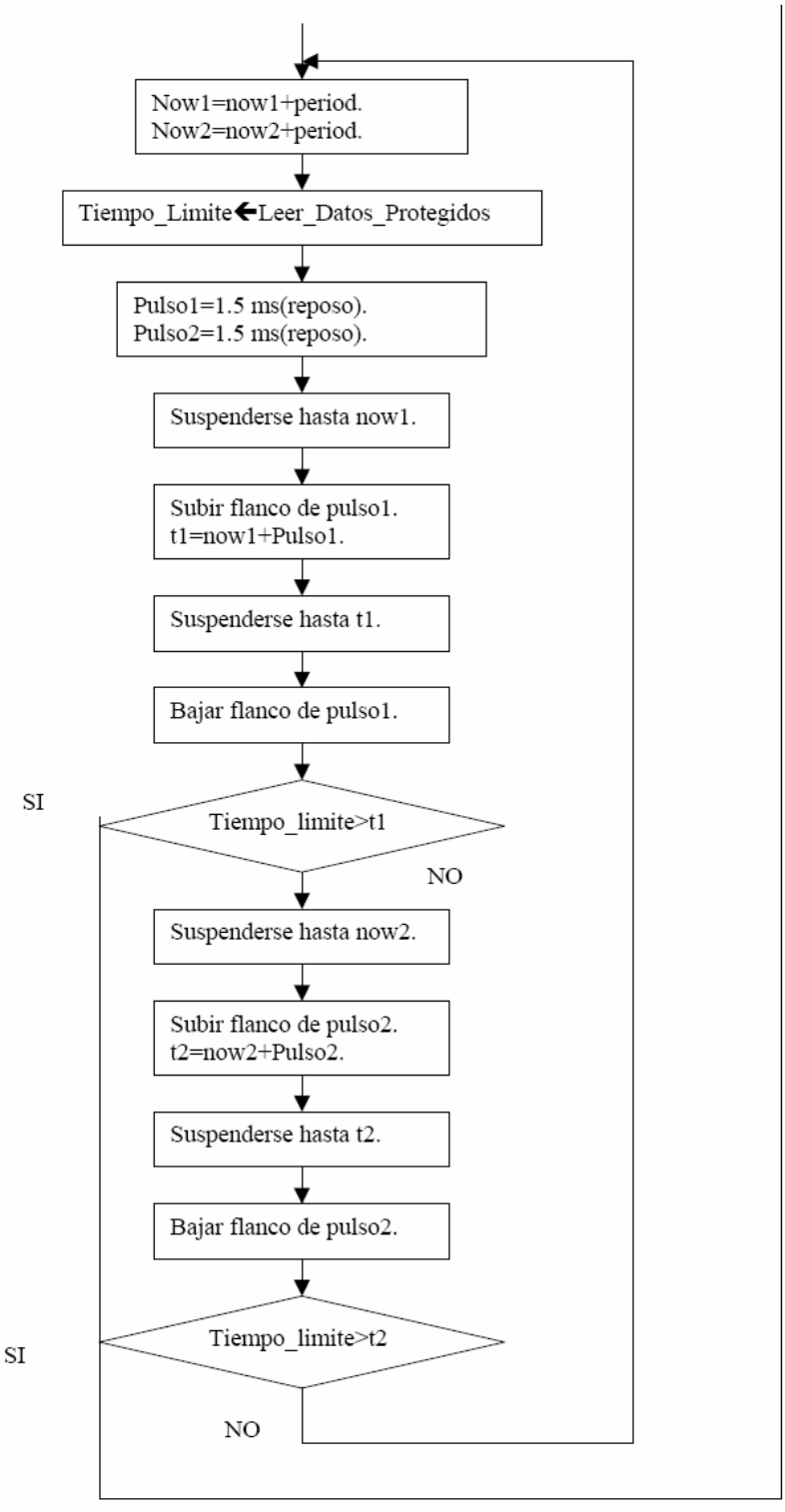
La lectura de los objetos protegidos, el cálculo del ancho de los pulsos y su posterior envío al puerto paralelo se realizan cíclicamente cada 20 ms.

Cuando se realiza la lectura de los datos protegidos además se dispone de un valor de tiempo absoluto que determina la duración de un segmento de trayectoria. Dentro del bucle principal de la tarea se comprueba si se ha llegado a cumplir ese plazo temporal comparándolo con los tiempos de bajada de los flancos, t1 y t2. Cuando ha vencido dicho plazo se sale del bucle para entrar en otro donde se establece el reposo del robot, enviando pulsos de 1.5 ms. En este segundo bucle se efectúa la lectura de *Datos_Protegidos* con la intención de conocer si se ha establecido un límite temporal superior a los tiempos t1 y t2 actuales, lo cual señalaría la llegada de nuevos datos para comenzar a ejecutar un nuevo segmento de trayectoria, volviéndose al primer bucle.

A continuación se muestra un flujograma de la implementación.


INICIO
Definir direcciones Puerto Paralelo:
Declarar constantes y variables:
Inicializar variables.
Now1=clock.
Now2=now1+semiperiod.
Now1=now1+period
Now2=now2+period.
V_lineal
V_angular
Tiempo límite
←Leer Datos_Protegidos
Velocidad_teórica1=f(V_lineal,V_angular).
Velocidad_teórica2=f(V_lineal,V_angular).
fencoder1
fencoder2
←Leer Frecuencias Protegidas
Velocidad_real1=f(fencoder1).
Velocidad_real2=f(fencoder2).
Error_velocidad1=velocidad_teorica1-velocidad_real1.
Error_velocidad2=velocidad_teorica2-velocidad_real2.
Velocidad_corregida1=velocidad_teórica1+Kp1*error_velocidad1.
Velocidad_corregida2=velocidad_teórica2+Kp2*error_velocidad2.
Pulso1=f(velocidad_corregida1).
Pulso2=f(velocidad_corregida2).


Suspenderse hasta now1.
Subir flanco de pulso1.
t1=now1+Pulso1.
Suspenderse hasta t1.
Bajar flanco de pulso1.
Tiempo_limite<t1
SI
NO
Suspenderse hasta now2.
Subir flanco de pulso2.
t2=now2+Pulso2.
Suspenderse hasta t2.
Bajar flanco de pulso2.
SI
Tiempo_limite<t2
NO


Now1=now1+period.
Now2=now2+period.
Tiempo_Limite←Leer_Datos_Protegidos
Pulso1=1.5 ms(reposo).
Pulso2=1.5 ms(reposo).
Suspenderse hasta now1.
Subir flanco de pulso1.
t1=now1+Pulso1.
Suspenderse hasta t1.
Bajar flanco de pulso1.
SI
Tiempo_limite>t1
NO
Suspenderse hasta now2.
Subir flanco de pulso2.
t2=now2+Pulso2.
Suspenderse hasta t2.
Bajar flanco de pulso2.
SI
Tiempo_limite>t2
NO

1.4.6 Tarea Encoders.

La principal función de esta tarea es la de contar los flancos de la señal que generan los encoders. Con el número de flancos que ha generado el encoder en un determinado periodo de tiempo se conoce la frecuencia de la señal recibida y por tanto la velocidad de giro de los motores.

Los encoders producen dos señales TTL iguales pero desfasadas 90 grados. Depediendo del estado de una de ellas cuando en la otra ocurre un flanco activo se puede saber el sentido de giro.

La primera opción que se barajó fue la de utilizar interrupciones para el conteo de los flancos. Una de las señales provocaría la interrupción hardware, incrementando un contador, mientras que la otra sería leída desde la rutina de servicio de interrupción para conocer el sentido. Esta solución se adapta perfectamente a nuestras necesidades ya que nos encontramos ante un evento asíncrono externo.

Para implementar esta opción es preciso una línea de interrupción para cada encoder. En el puerto paralelo la línea ACK provoca la interrupción IRQ7 cada vez que detecta un flanco descendente. Se necesitarían dos entradas capaces de producir interrupciones , pero esta es la única línea de que disponemos en el puerto paralelo con lo cual se abandona este camino.

Otra forma de abordar el problema es la técnica polling (encuesta). Consiste en realizar periódicamente lecturas de las señales. Cuando se detecten diferencias en dos lecturas consecutivas se habrá producido un flanco. En ese instante se procederá a leer la señal desfasada para conocer el sentido. Podemos realizar la encuesta de 2 formas distintas:

- Un bucle que lee las señales y se suspende hasta un instante prefijado en el que volverá a realizar una nueva lectura (utilizando una instrucción delay until).

- Un bucle que haga lecturas de las señales continuamente sin suspenderse.

La primera opción es más eficiente que la segunda ya que supone una sobrecarga de la CPU mucho menor. Para hacer pruebas se realizó una tarea que leía una señal enviada por un generador de funciones.

Para una velocidad de 30 cm/seg. los encoders devuelven una señal de 8.3 KHz. que tiene un periodo de 120 us. Para detectar los cambios es necesario hacer encuesta cada semiperiodo (60 us.). Se comprueba que a partir de una frecuencia de 6.3 KHz aparece error en la medición, creciendo este a medida que crece la frecuencia a medir. Este error es debido a las limitaciones de la instrucción de suspensión temporizada de tareas que se expondrán a continuación.

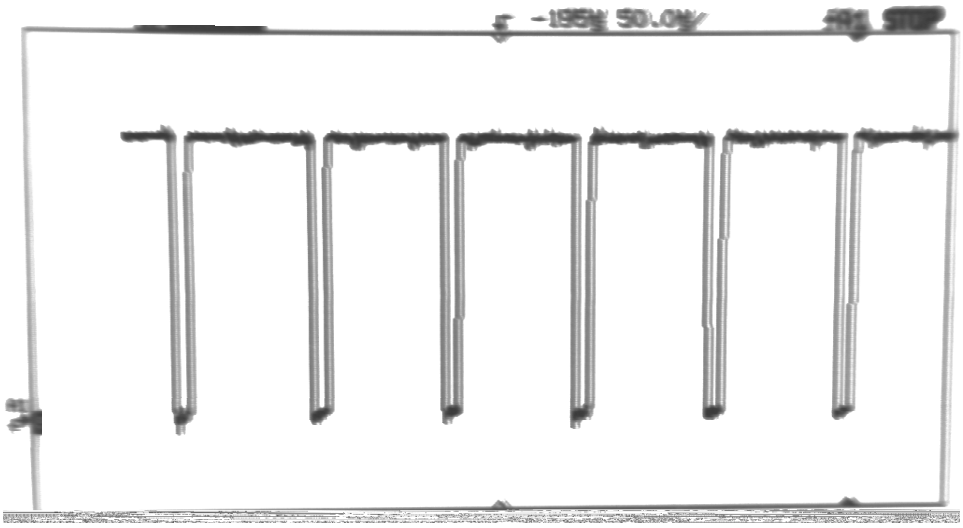
Para calcular cuál es el mínimo tiempo que puede suspenderse una tarea se escribe una que pone a 0 un bit del puerto paralelo, seguidamente le pone a 1 y se suspende un tiempo determinado. Tenemos en cuenta que el retraso de la puerta es del orden de los 2 us.

```
Now:=clock;

      Loop
            IO_Interface.Outb(PP_BASE_REG + PP_DATA_REG, 0);
            Now:=now+microseconds(X);
            IO_Interface.Outb(PP_BASE_REG + PP_DATA_REG, 1);
            Delay until now;
      End loop;
```

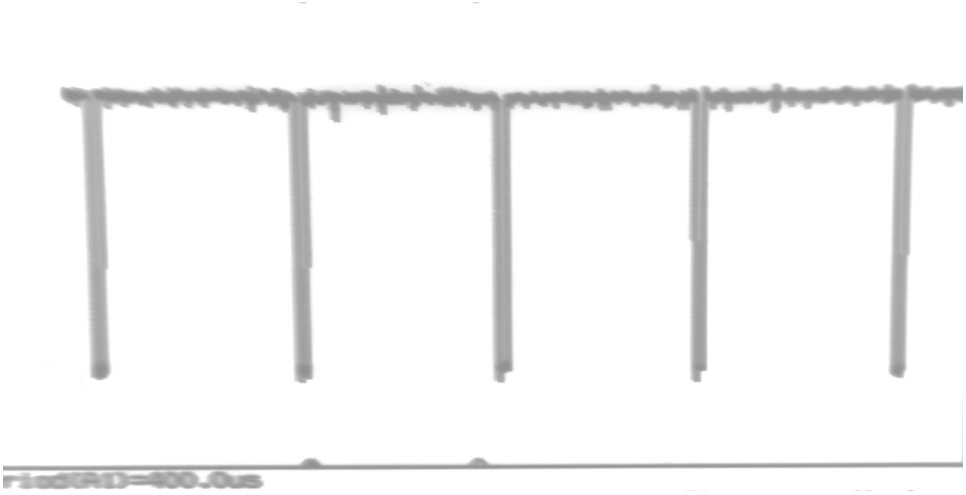
Se obtienen las siguientes conclusiones:

- a) *X<70 us:* se obtiene una señal periódica pero el periodo nunca es menor de 70 us.



- b) *70<X<390 us:* se obtiene una señal no periódica con importantes derivas tanto por exceso como por defecto respecto del valor deseado.

- c) *X>390 us*: se obtiene una señal periódica sin derivas.



Cabe recordar que en la tarea de control sucedía algo parecido cuando la instrucción *delay until* hacía referencia a dos instantes separados menos de 380 us. aproximadamente.

Hay que tener en cuenta que nos encontramos en una plataforma 486 [14] y que la temporización se basa en el temporizador 8254 de Motorola más conocido como PIT (*Programable Interrupt Timer*). La estrategia de programación de este dispositivo es similar a la de RT Linux, utilizando el contador 0 y el contador 2, ambos de 16 bits. La gran desventaja de este temporizador es que se accede a través del bus I/O que es bastante lento pudiendo tardar una lectura incluso varios microsegundos.

En arquitecturas Pentium la labor del contador 2 del PIT la realiza el TSC (Time-stamp counter) que tiene un registro de 64 bits al que se accede a través del bus interno, siendo los tiempos de acceso del orden de los nanosegundos.

A partir de la familia Pentium II se prescinde totalmente del PIT combinándose el uso del TSC con el del Local APIC para mejorar aún más las prestaciones del sistema en cuanto a temporización se refiere. Aquí se puede ver una tabla comparativa entre las tres posibilidades, expresadas en ns:

Operation	PIT	TSC+PIT	TSC+APIC
Get time	3100	105	105
Program timer	12900	3000	324

figura 25: Comparativa entre los distintos temporizadores hardware.

Ante las dificultades encontradas en el uso de la suspensión temporizada se plantea la utilización del polling continuo pese a la sobrecarga que conlleva.

Hay que tener en cuenta la asignación de prioridades que se debe realizar a cada una de las tareas. La tarea que realiza la encuesta continua debe tener la prioridad más baja ya que si no fuese así acapararía la CPU no permitiendo ejecutar a las demás tareas del sistema. De entre las otras dos tareas es prioritaria la de control ya que pequeñas variaciones en el ancho de los pulsos de control provocan efectos no deseados mientras que un pequeño retraso en mandar nuevos datos no provoca mayores problemas. Por lo tanto:

Prioridad(*Encoders*)<Prioridad(*Escribe_Datos*)<Prioridad(*Control*).

Para evitar el fenómeno de inversión de prioridad se asigna a cada uno de los objetos protegidos como mínimo la prioridad más alta de las tareas que acceden a él, tratándose en ambos objetos de la tarea de *contro*l:

Prioridad(*Datos_Protegidos*)=Prioridad(*Frecuencias_Protegidas*)=Prioridad(*Contro l*).

Con esta asignación de prioridades la tarea *Encoders* perderá flancos mientras que se encuentren activas las tareas *Control y Escribe_Datos,* que tienen la más alta prioridad. Para medir estos efectos y ante la poca precisión que ofrecen los servicios de temporización que proporcionan tanto Ada como POSIX (debido a la utilización del PIT), en la tarea que realiza polling se introducen dos nuevas líneas de código, una que sube una línea de datos del puerto paralelo y otra que baja dicha línea:

```
Loop

     IO_Interface.Outb(PP_BASE_REG + PP_DATA_REG, 8);

     .   .   .   .
     .   .   .   .

     IO_Interface.Outb(PP_BASE_REG + PP_DATA_REG, 0);

     .   .   .   .
     .   .   .   .
End loop;
```

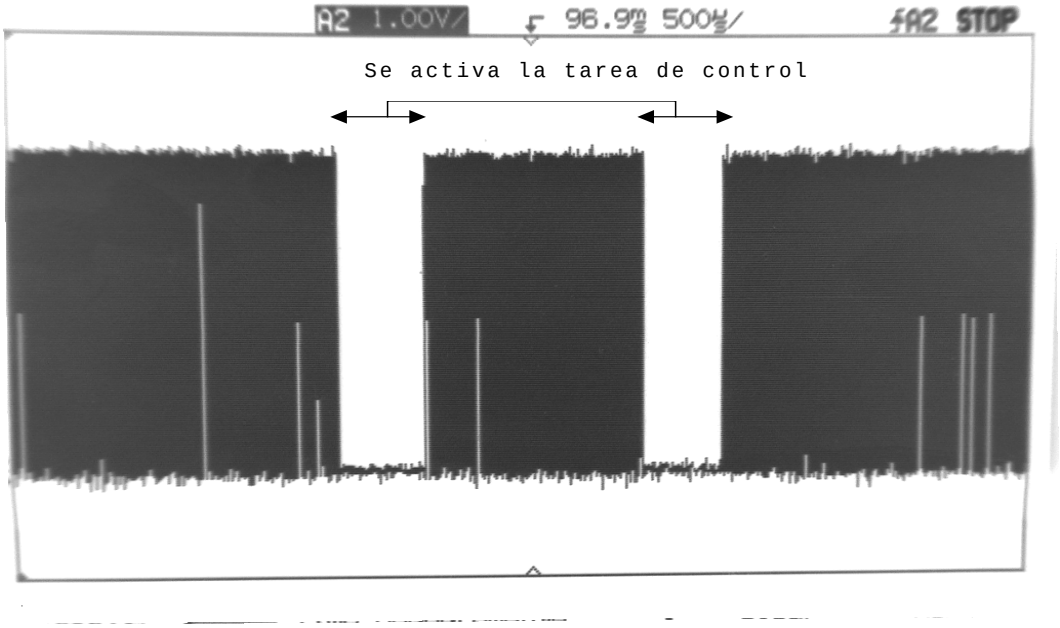


figura 26: Perdida precisión en el polling cuando se activa la tarea de control.

En la anterior figura se puede ver cómo la tarea que realiza polling permanece inactiva durante 400 us cada vez que se activa la tarea de control. Este tiempo es el tiempo que tarda en ejecutarse la tarea de control y provoca la pérdida de flancos en la lectura de los encoders.

Pero la pérdida de datos (flancos no detectados) no es la misma en todas las frecuencias que se miden:

- Si la frecuencia es menor de 1.2 KHz no se aprecia pérdida de datos.
- Para frecuencias entre 1.2 KHz y 2.5 KHz se produce un error creciente que aunque no es perfectamente lineal se puede asumir como tal.
- A partir de 2.5 KHz el error más o menos constante estando las medidas realizadas entre un 8 % y un 9% por debajo del valor esperado.

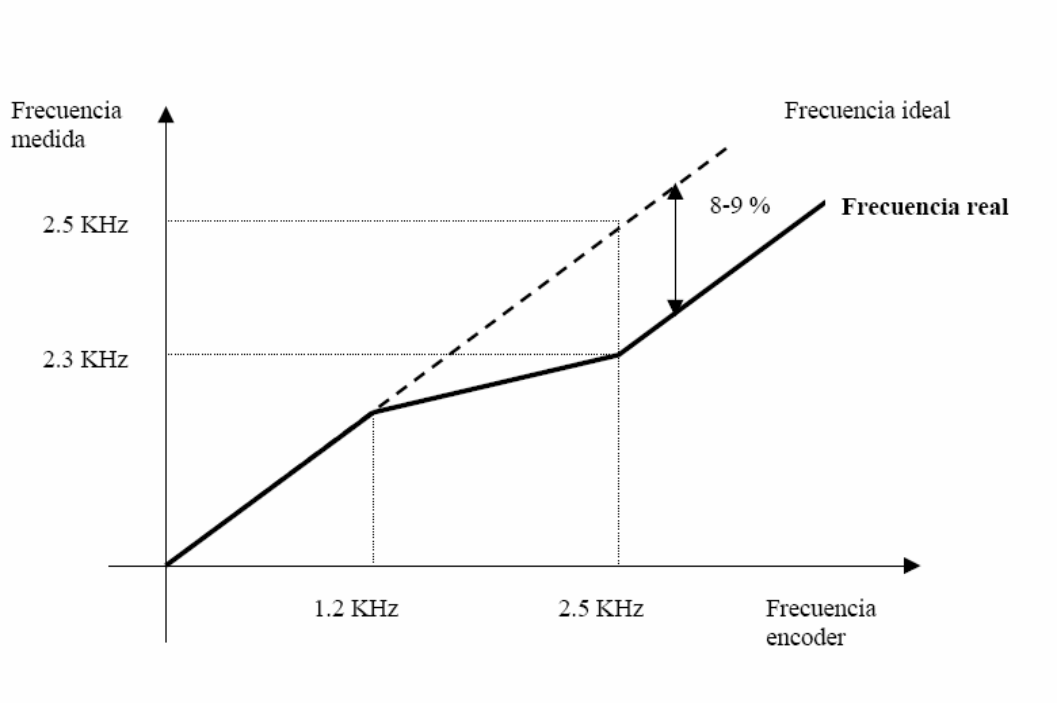


figura 27: Gráfica del error para las distintas frecuencias.

Por lo tanto es necesario establecer unas correcciones respecto de los valores medidos:

- Hasta 1.2 KHz no se modifica la medición.
- Entre 1.2 KHz y 2.5 KHz tenemos en cuenta la pendiente del error para realizar la corrección:

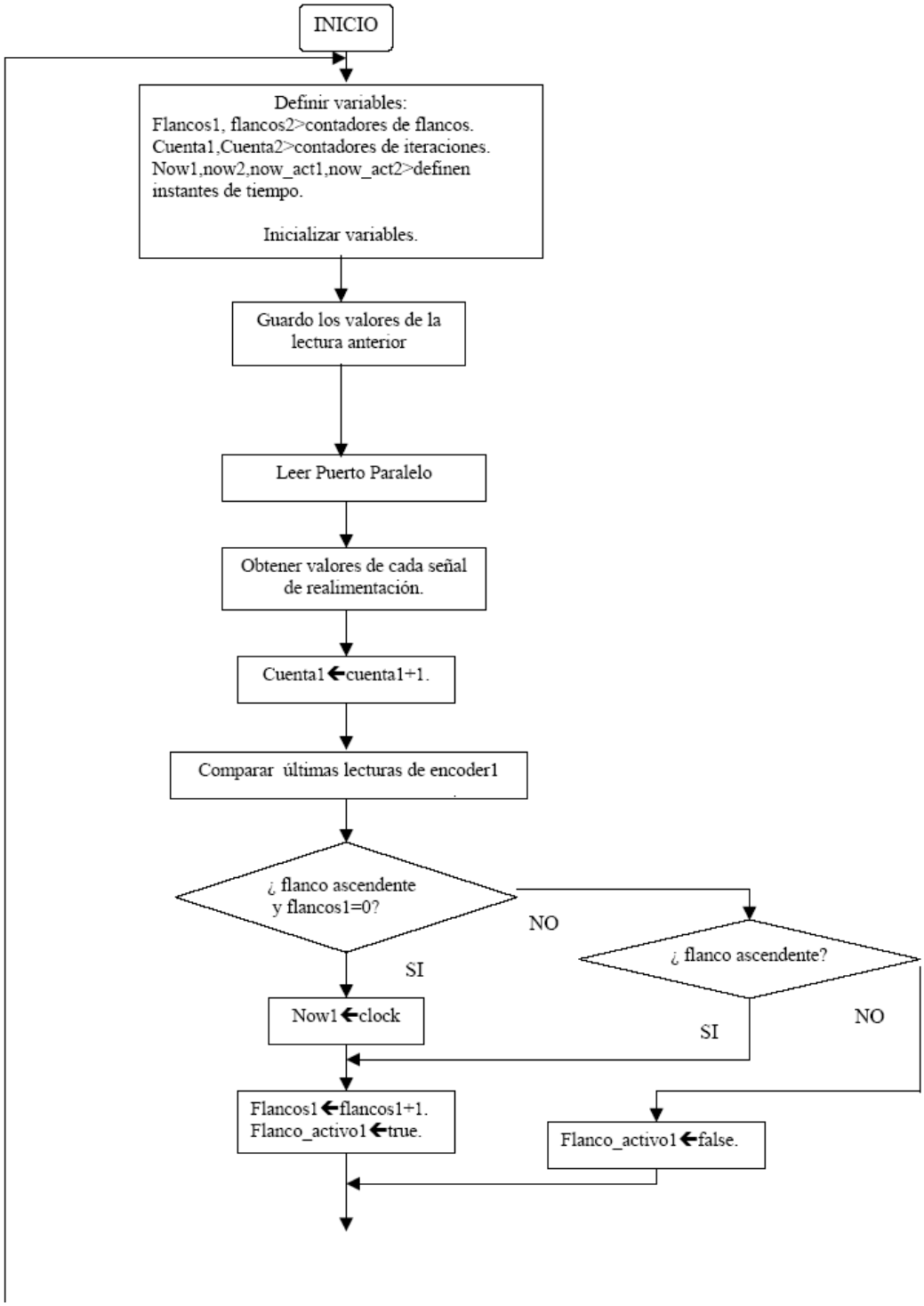
  f = f * (1+error) = f * (1 + (f-1200)*(0.085/1300)).

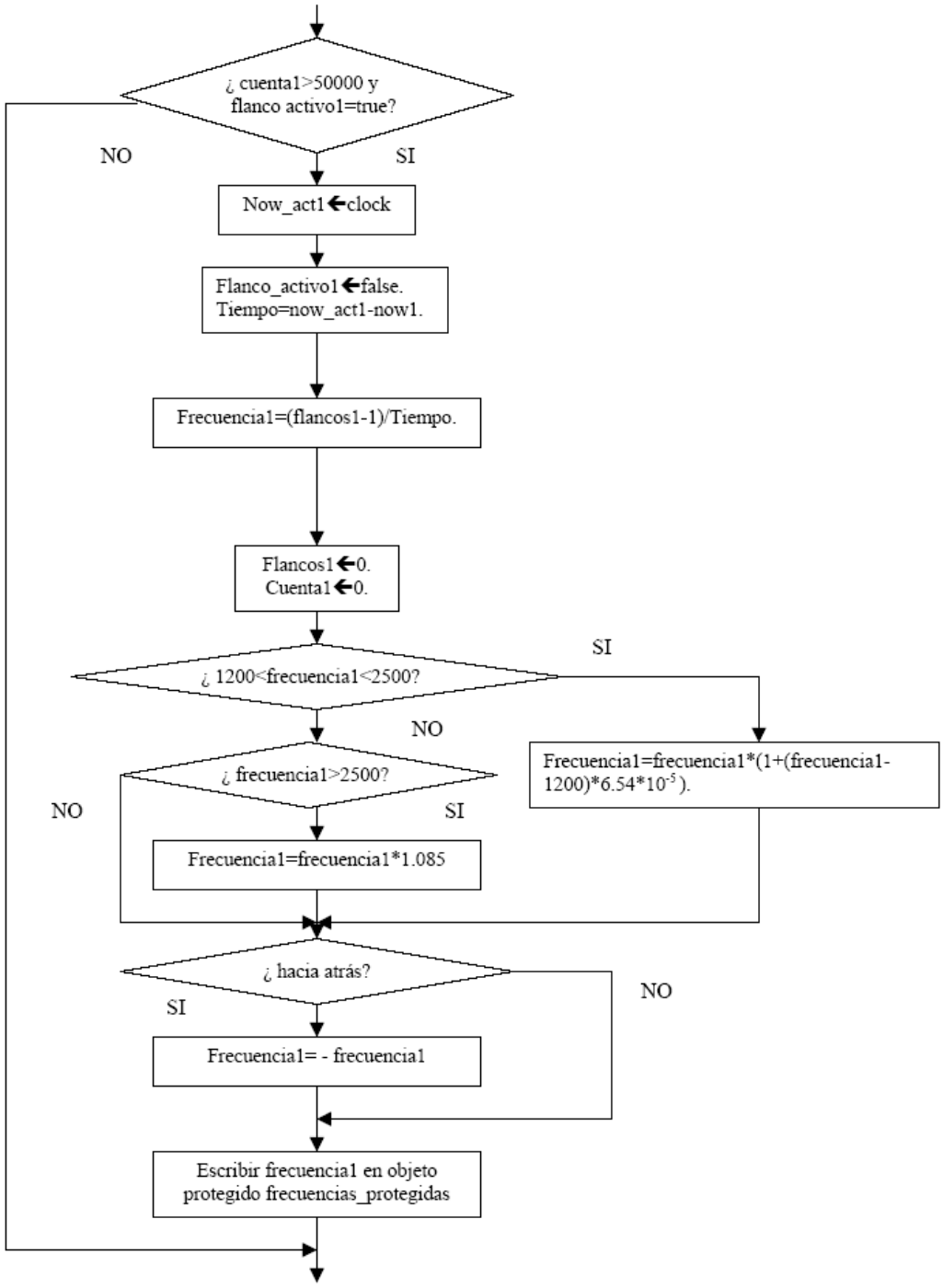
- A partir de 2.5 KHz tomamos un error del 8.5 %:

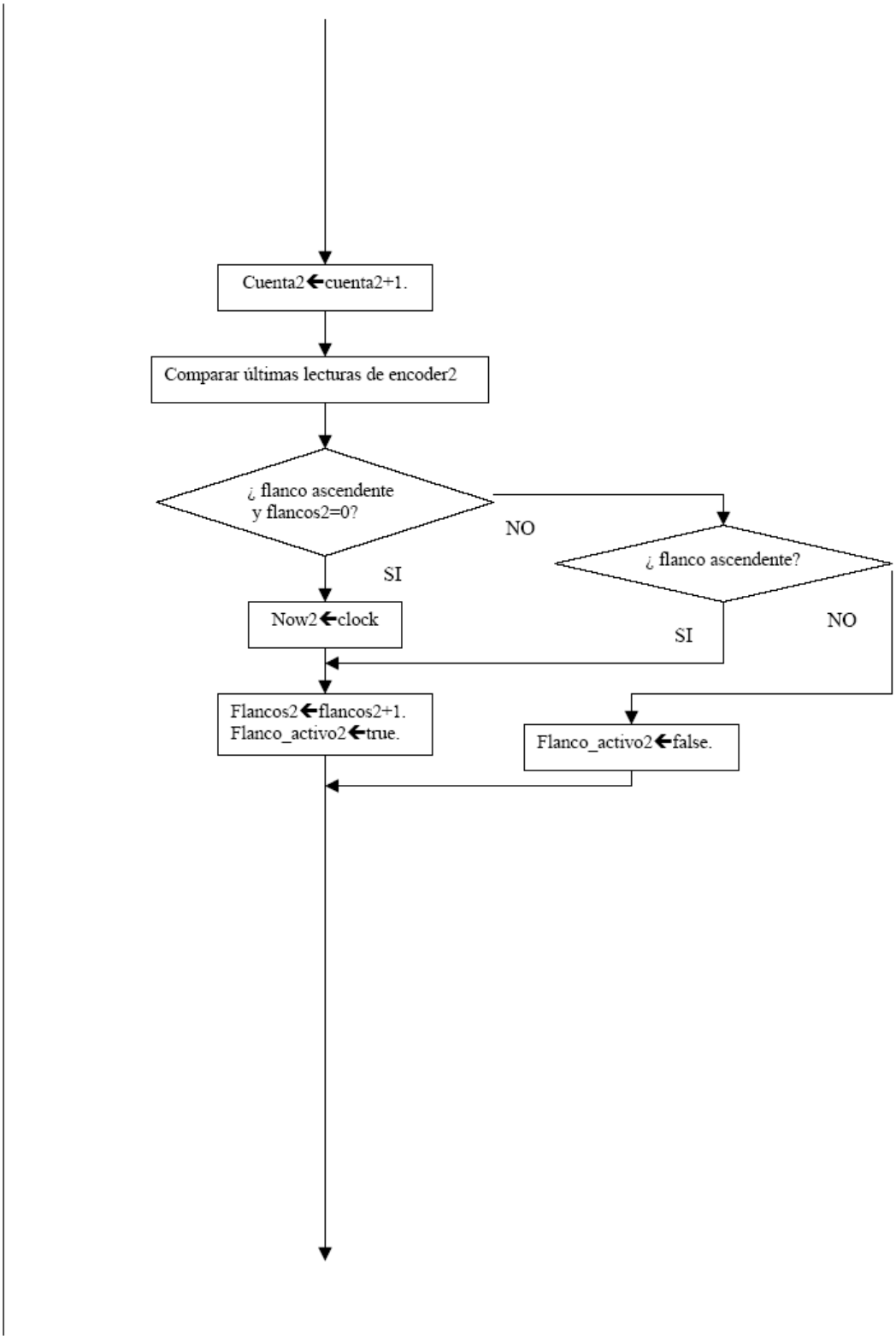
  f = f * 1.085.

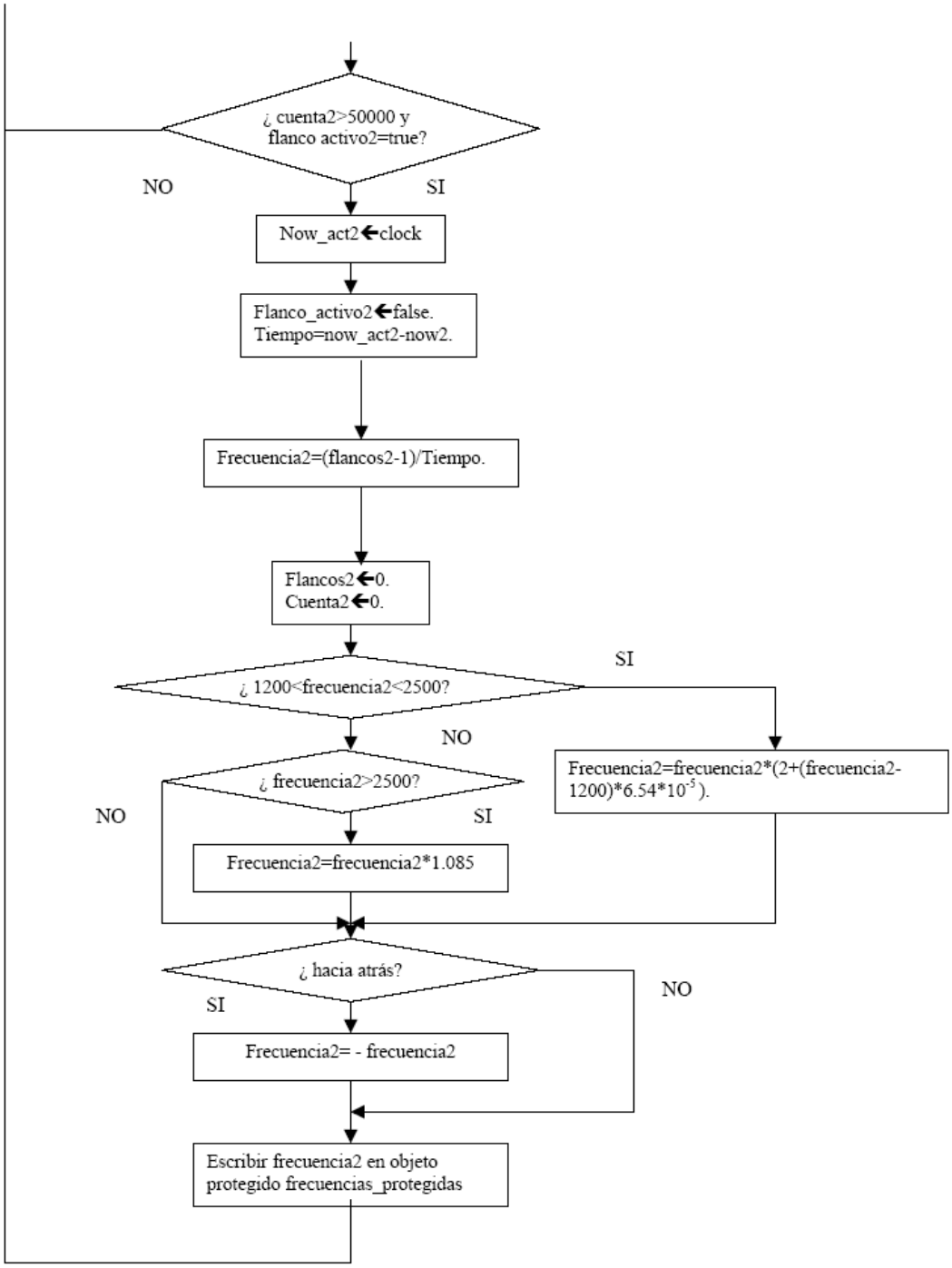
Una vez obtenido este valor de frecuencia sólo falta comprobar el sentido de giro y escribir los resultados en el objeto protegido.

A continuación se muestra un flujograma que intenta reflejar la implementación de la tarea Encoders.


INICIO
Definir variables:
Flancos1, flancos2>contadores de flancos.
Cuenta1,Cuenta2>contadores de iteraciones.
Now1,now2,now_act1,now_act2>definen instantes de tiempo.
Inicializar variables.
Guardo los valores de la lectura anterior
Leer Puerto Paralelo
Obtener valores de cada señal de realimentación.
Cuenta1←cuenta1+1.
Comparar últimas lecturas de encoder1
¿ flanco ascendente y flancos1=0?
NO
SI
¿ flanco ascendente?
SI
NO
Now1←clock
Flancos1←flancos1+1.
Flanco_activo1←true.
Flanco_activo1←false.


¿ cuenta1>50000 y
flanco activo1=true?
NO
SI
Now_act1←clock
Flanco_activo1←false.
Tiempo=now_act1-now1.
Frecuencia1=(flancos1-1)/Tiempo.
Flancos1←0.
Cuenta1←0.
¿ 1200<frecuencia1<2500?
SI
NO
Frecuencia1=frecuencia1*(1+(frecuencia1-1200)*6.54*10-5).
¿ frecuencia1>2500?
NO
SI
Frecuencia1=frecuencia1*1.085
¿ hacia atrás?
NO
SI
Frecuencia1= - frecuencia1
Escribir frecuencia1 en objeto protegido frecuencias_protegidas


Cuenta2←cuenta2+1.
Comparar últimas lecturas de encoder2
¿ flanco ascendente y flancos2=0?
NO
SI
¿ flanco ascendente?
SI
NO
Now2←clock
Flancos2←flancos2+1.
Flanco_activo2←true.
Flanco_activo2←false.


¿ cuenta2>50000 y
flanco activo2=true?
NO
SI
Now_act2←clock
Flanco_activo2←false.
Tiempo=now_act2-now2.
Frecuencia2=(flancos2-1)/Tiempo.
Flancos2←0.
Cuenta2←0.
¿ 1200<frecuencia2<2500?
SI
NO
¿ frecuencia2>2500?
NO
SI
Frecuencia2=frecuencia2*(2+(frecuencia2-1200)*6.54*10-5).
Frecuencia2=frecuencia2*1.085
¿ hacia atrás?
NO
SI
Frecuencia2= - frecuencia2
Escribir frecuencia2 en objeto
protegido frecuencias_protegidas

1.4.7 Montaje y pruebas

Para comprobar el funcionamiento del prototipo se ha realizado el montaje del mismo utilizando una placa de baquelita para el ensamblado de la circuitería. Se trata de un montaje provisional cuya única finalidad es la de realizar comprobaciones de funcionamiento del control que se ha realizado. Debido al rozamiento que existe entre la superficie del chasis y el suelo se ha decidido montar un patín en la parte posterior que permite equilibrar la base, mejorando notablemente el desplazamiento y la ejecución de las trayectorias.

Como se expuso con anterioridad en la tarea de control se han implementado dos reguladores proporcionales, uno para cada rueda. Es necesario calcular el valor apropiado para las constantes proporcionales de ambos reguladores, por lo que se han hecho pruebas hasta conseguir que en línea recta alcance la velocidad deseada ejecutando las trayectorias lo más rectilíneas posibles. Los valores obtenidos son Kp1=0.6 para el motor izquierdo y Kp2=0.9 para el motor derecho.

En la realización de trayectorias curvas se ha observado que a partir de aproximadamente 1 rad/seg no se llega a obtener todo el giro deseado lo cuál podría deberse al rozamiento que provoca el patín trasero al desplazarse lateralmente cuando gira el chasis. Otra solución estaría en el montaje de una pequeña rueda loca sustituyendo al patín.

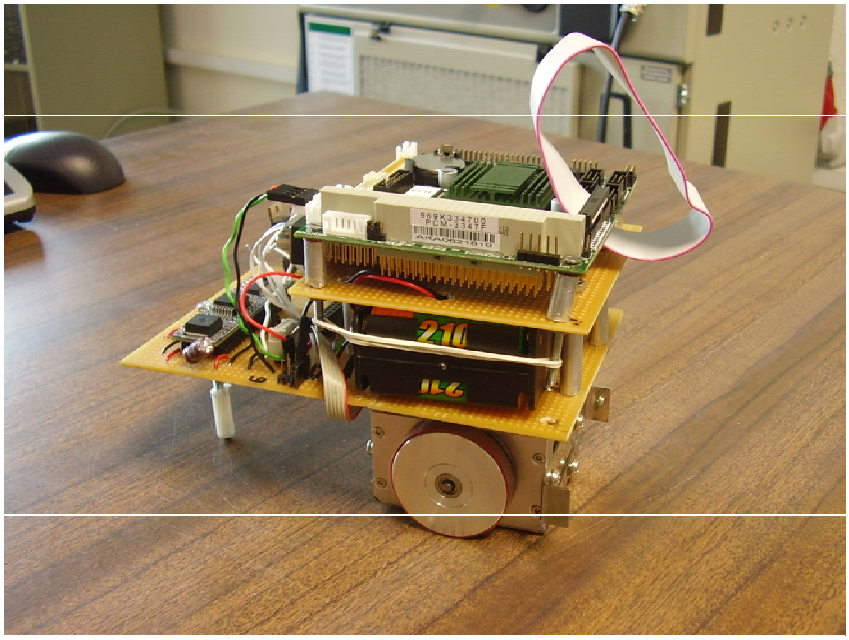


figura 28: Aspecto del prototipo.

1.5 Conclusiones y líneas futuras de trabajo.

1.5.1 Conclusiones.

Se ha conseguido implementar un control de trayectoria para este prototipo dejándolo preparado para que sea un control superior el que calcule y ordene las trayectorias.

Las mayores dificultades encontradas han sido las relativas a la temporización ya que se ha visto que en la plataforma utilizada no es posible que en un mismo programa haya eventos que puedan activarse en un intervalo inferior a 400 us lo que limita bastante el uso de la instrucción *delay until.* Además el acceso a los servicios de temporización para establecer mediciones es lento, no ofreciendo la precisión deseada.

Se ha comprobado la idoneidad del resto de elementos hardware así como del puerto paralelo para realizar el intercambio de información. No obstante la ejecución de las trayectorias se ve limitada por el rozamiento especialmente en giros con elevada velocidad angular.

Debido a que la tarea *Encoders* está basada en la técnica polling sin realizar suspensión ninguna, se produce una cierta sobrecarga de la CPU. Además, la inclusión de nuevas tareas que limitarían el tiempo disponible de la CPU, incrementaría las perdidas de datos por parte de la tarea *Encoders*, haciendo necesario el cálculo de nuevas correcciones.

1.5.2 Líneas Futuras de trabajo.

Como primera línea de trabajo futuro se plantea la revisión de la programación del PIT en MaRTE OS con la finalidad de mejorar la precisión y evitar las derivas temporales de carácter aparentemente aleatorio que se producen.

Si con la anterior iniciativa no se consigue la precisión deseada se plantea la necesidad de incorporar al sistema algún tipo de temporizador hardware externo que sea capaz de generar interrupciones.

También es posible la mejora del desplazamiento del prototipo mediante la instalación de una tercera rueda que permita giros en cualquier dirección con una apreciable disminución del rozamiento.

Debido a que en el futuro además del control de trayectoria el robot deberá soportar nuevos modos de control y ante el previsible aumento del número de tareas se plantea la posibilidad de realizar las tareas de Control y Encoders en una sóla, utilizando un contador hardware externo para los encoders. Esta situación permitiría reducir la carga de trabajo de la CPU además de eliminar el error que actualmente se produce en la medida del número de flancos de los encoders. Con esta intención en el próximo apartado se describe el contador Agilent HCTL-2032.

Además de mejorar el control de trayectorias realizado en el presente trabajo en el futuro se tratará de implementar nuevas funcionalidades al robot. Actualmente otro proyecto se está encargando de la creación del driver de la tarjeta PCM 3718 que permitirá añadir al prototipo nuevos elementos hardware.

Asimismo será necesario habilitar un sistema de visión para la cámara cenital. También se planteará el envío de las señales a los robots de manera inalámbrica. Una vez que se configuren los equipos se tratará el problema de elaborar estrategias

(inteligencia artificial) buscando la “cooperación” y el “entendimiento” entre todos los robots del equipo.

1.5.2.1 Utilización del contador Agilent HCTL-2032 .

Se trata de un circuito concebido para facilitar el control de lazo cerrado en sistemas digitales, sirviendo de interfaz entre los encoders y el PC. Sus principales características son[12]:

- Permite controlar simultáneamente dos ejes.
- I/O compatible CMOS/TTL.
- Posee contadores ascendentes/descendentes de 32 bits (>4294 millones de pulsos).
- Ofrece la posibilidad de programar 3 modos de funcionamiento ofreciendo así tres grados distintos de precisión.
- Elimina el ruido de las señales.
- Necesita un reloj externo para su funcionamiento que puede ser de hasta 32 MHz.
- Los datos son leídos de un buffer a través de 8 líneas de datos.

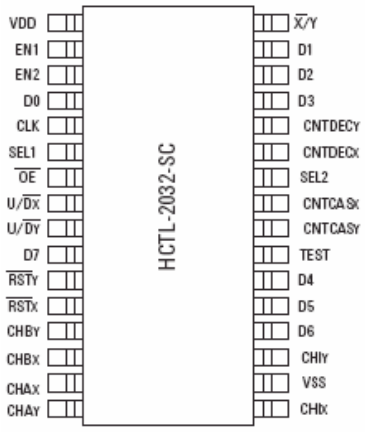


figura 29:Patillaje.

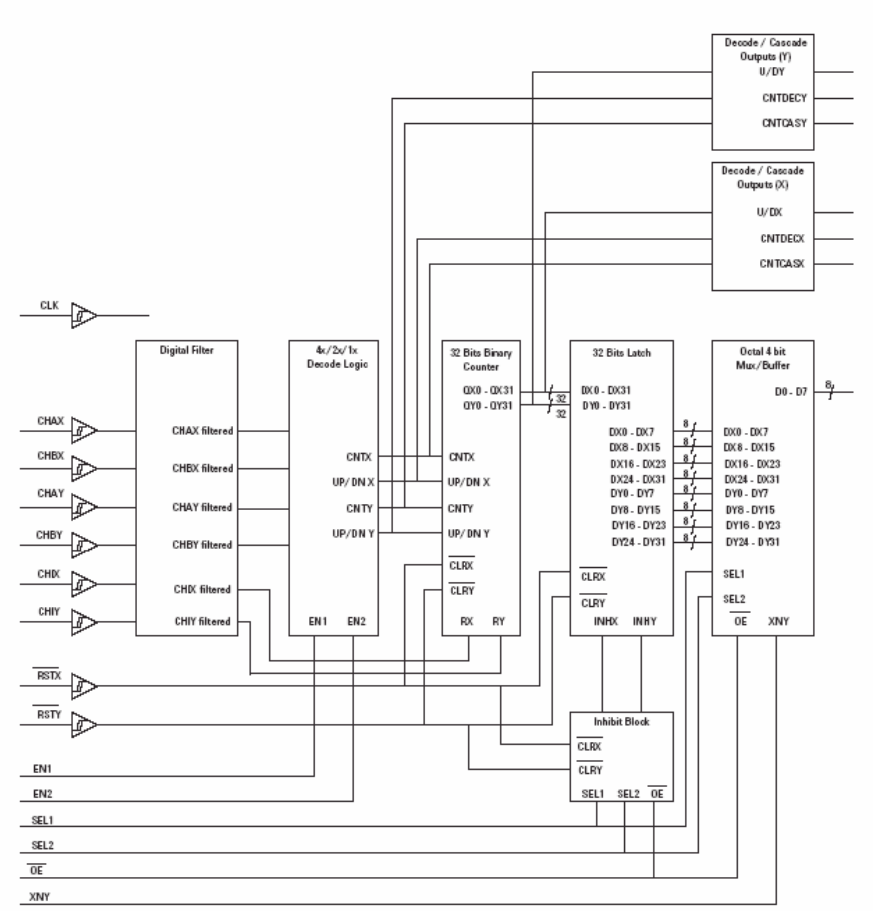


figura 30: Esquema interno del circuito.

Las señales de los encoders son recogidas por un lado por las entradas CHAX y CHBX para el canal X y por otro por las entradas CHAY y CHBY para el canal Y. Como se aprecia en la figura anterior en una primera etapa se realiza un filtrado de las señales digitales para eliminar pulsos parásitos.

Las señales filtradas acceden al decodificador el cual puede configurarse para operar en tres modos de cuenta diferentes (1X, 2X, 4X). Estos modos permiten establecer la precisión del conteo. La configuración se realiza a través de las entradas EN1 y EN2.

		Count Modes		
EN1	EN2	4x	2x	1x
0	0	Illegal Mode		
1	0	On		
0	1		On	
1	1			On

figura 31: Valores de EN1 y EN2 para seleccionar los modos de funcionamiento.

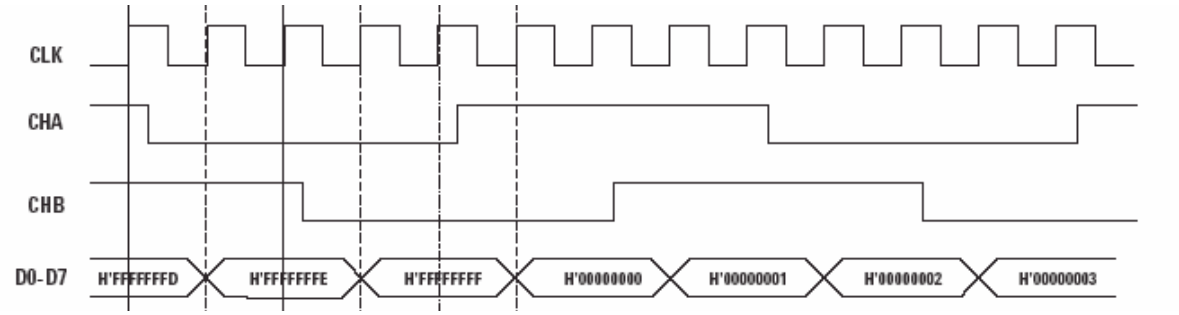


figura 32: Descripción del conteo en el modo 4x (máxima precisión).

Posteriormente se encuentran los contadores de 32 bits tanto ascendentes como descendentes. Con el fin de realizar una lectura segura de los contadores, el circuito HCTL-2032 dispone de un latch de 32 bits que mantiene un valor estable del contador durante todo el tiempo que dura una operación de lectura.

Para facilitar la lectura de los datos se cuenta con un buffer tri-state que posee 8 líneas de salida de datos. Por lo tanto para leer los 32 bits de un contador será necesario realizar cuatro lecturas sirviéndose para ello de las entradas SEL1 y SEL2.

		BYTE SELECTED			
SEL1	SEL2	MSB	2ND	3RD	LSB
0	1	D4			
1	1		D3		
0	0			D2	
1	0				D1

figura 33: Valores de SEL1 y SEL2 asociados a la lectura de cada byte.

La elección del eje que queremos leer se realiza mediante la entrada X(N)/Y.

Además el circuito cuenta con dos entradas asíncronas que permiten resetear el valor de cuenta en cualquier momento: RSTX(N) y RSTY(N).

En cuanto al reloj externo la única restricción que se impone es que su frecuencia debe ser como mínimo 6 veces superior a la de la señal de entrada. Esta última se estima que no sobrepasará los 10-15 KHz, por lo tanto un cristal de cuarzo de 4 MHz es suficiente, aunque valores superiores redundarán en una exactitud mayor en el funcionamiento del circuito.

A la hora de realizar la lectura de los valores hay que tener en cuenta que dependiendo del sentido de giro, los contadores irán hacia arriba o hacia abajo. Para realizar la lectura de los datos se puede emplear uno de los dos canales digitales de 8 bits que posee la tarjeta PCM-3718 H de Advantech.

Para enviar las señales de configuración al circuito (SEL1,SEL2, OE, RSTX, RSTY, EN1, EN2, X/Y) se utilizarán las líneas de datos del puerto paralelo.

Como ya se comentó anteriormente es necesario realizar 4 sucesivas lecturas para conocer el valor de los contadores. Pasamos a detallar los pasos necesarios para realizar un ciclo de lectura completo:

- Se selecciona el eje que se quiere leer mediante la entrada X(N)/Y.
- Se escribe un 0 en la entrada OE(N) para permitir la lectura segura del buffer.

- SEL1=0,SEL2=1.
- Se lee el MSB.

- SEL1=1,SEL2=1.
- Se lee el 2º byte..

- SEL1=0,SEL2=0.
- Se lee el 3º byte..

- SEL1=1,SEL2=0.
- Se lee el LSB.

- Se escribe un 1 en la entrada OE(N).

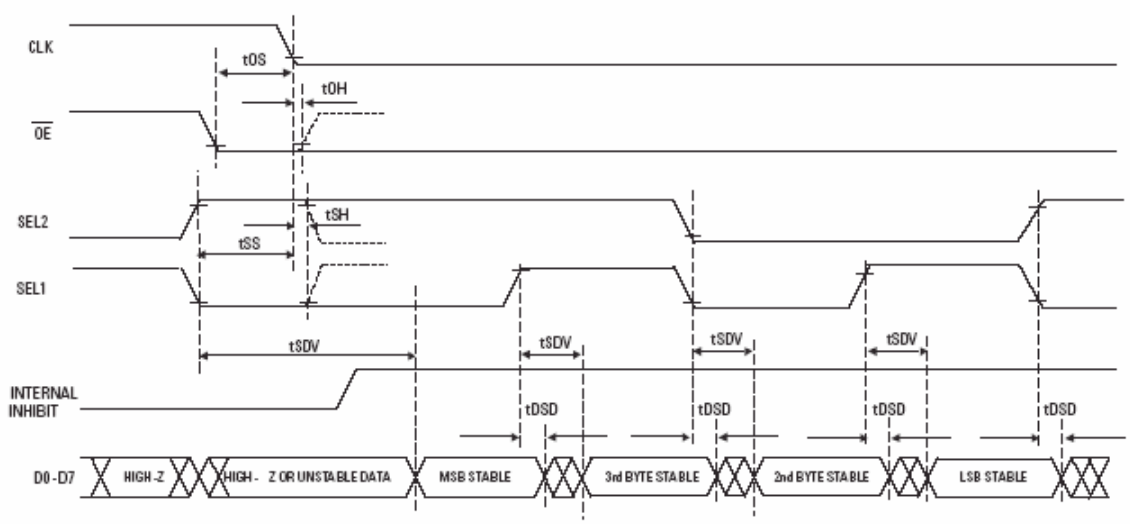


figura 34: Descripción de las señales durante un ciclo de lectura.

2 Pliego de condiciones.

2.1 Comentarios.

Dado el carácter informático del proyecto, no existen condiciones técnicas o ambientales que puedan afectarle o referirse al mismo de forma alguna. No obstante las condiciones generales vinieron impuestas del exterior como consecuencia de las exigencias que se pedían en su realización.

2.1.1 Reseña.

Las exigencias que limitaban la construcción del proyecto de aplicación informática, eran las siguientes :

- Utilización de una plataforma de desarrollo PC compatible.
- Utilización del sistema operativo MaRTE OS.
- Utilización del lenguaje de programación Ada 95.

2.2 Condiciones recomendadas de utilización.

En la utilización del proyecto de aplicación informática se derivan una serie de condiciones que limitan de alguna forma la utilización del mismo. Estas condiciones resultaron como consecuencia de llevar a la práctica la realización del proyecto. Son las siguientes :

2.2.1 Condiciones software:

- Necesidad del sistema operativo Linux (recomendable Fedora).
- Necesidad de un compilador GNAT para las aplicaciones Ada (GAP 1.0).
- Necesidad de instalar un sistema operativo de tiempo real (MaRTE OS).
- Necesidad de facilitar la descarga de los programas mediante la configuración adecuada de servidores (DHCP, NFS, TFTP...).

2.2.2 Condiciones hardware:

- Necesidad de un entorno de desarrollo formado por dos computadores uno denominado Host sobre el que se realizan y compilan las aplicaciones, y otro target sobre el que se ejecutan dichas aplicaciones.

- La tensión de alimentación de los distintos componentes se indica a continuación:

  - Placa PCM 3347 F: 5 voltios de corriente continua con un rizado inferior al 5%.
  - Controlador de potencia MMC: 6 voltios de corriente continua.
  - Motores de corriente continua: son alimentados por el controlador de potencia.
  - Codificadores de posición: 5 voltios de corriente continua con un rizado inferior al 5%.

2.2.3 Condiciones ambientales:

A priori no hay necesidad de implementar ningún sistema de disipación de calor para el correcto funcionamiento del prototipo. No obstante si las circunstancias lo aconsejasen es viable la adopción de alguno de estos sistemas.

En la placa PC se recomienda la utilización de un ventilador para temperaturas de funcionamiento superiores a 75 ºC mientras que para el chip MMC se aconseja la ventilación cuando la temperatura de los puentes H de potencia supere los 170 ºC.

3 Presupuesto.

3.1 Introducción.

En el ejercicio profesional de la ingeniería se hace necesario la planificación y el control de los elementos que intervienen en el desarrollo de dicha actividad.

La elaboración de un presupuesto obliga a tener en cuenta en muchas ocasiones un largo proceso de trabajo, la amortización de inversiones anteriores al comienzo del propio trabajo, materiales utilizados, software y otra serie de costes que en ocasiones no se ven reflejados en el producto final pero que han sido necesarios de una u otra manera para su consecución.

Distinguiremos en la confección del presente presupuesto entre costes directos y costes indirectos.

3.2 Costes directos.

- Coste del trabajador:
  - Creación del software: 250 h * 18 euros/h=4500 euros.
  - Montaje : 20 h * 18 euros/h=360 euros.

COSTE TOTAL TRABAJADOR= 400∈+360∈= **4860 ∈**.

- Coste Equipo:
  - Software: el software utilizado es gratuito (Linux. Gnat, MaRTE...).
  - Hardware:

Descripción	**Precio unitario**	**Cantidad**	**Coste total**
PCM-3347 F	335.98 ∈	1	335.98 ∈
Chasis Soccer Robo Body	232 ∈	1	232 ∈
Motor Mind C-REV 4	45 ∈	1	45 ∈
Placa baquelita	2.80∈	1	2.80∈
Portapilas de 2	1.66∈	2	3.32∈
Portapilas de 4	2.25∈	2	4.50∈
Regulador 7805 CT	1.08∈	2	2.16∈
Tira de conectores	0.45∈	1	0.45∈
Condensador 100 uF	0.07∈	1	0.07∈
Cable trenzado	0.21∈	1	0.21∈
Pilas recargables	3.75∈	12*2	95∈
Cargadores	25∈	4	100∈
COSTE TOTAL HARDWARE			**821.49∈**

COSTE TOTAL DIRECTO = 4860 ∈ + 821.49∈ = **5681.49 ∈**.

3.3 Costes indirectos:

- electricidad: 90 ∈.
- Teléfono e internet: 50 ∈.

COSTE TOTAL INDIRECTO = 90 ∈ + 50 ∈ = **140 ∈.**

3.4 Coste total.

COSTE TOTAL = COSTE TOTAL DIRECTO+COSTE TOTAL INDIRECTO=
= 5681.49 ∈+140 ∈ = **5821.49 ∈**.

EL COSTE TOTAL DE LA REALIZACIÓN DEL PROYECTO ASCIENDE A:

5821.49 ∈

4 BIBLIOGRAFÍA Y REFERENCIAS:

[1] Mario Aldea, "*Planificación de Tareas en Sistemas Operativos de Tiempo Real Estricto para Aplicaciones Empotradas*", Tesis doctoral, Dpto. Electrónica y Computadores, Grupo de Tiempo Real, Universidad de Cantabria, Mar.2003

[2] Michael González. "Apuntes de Ada". U. Cantabria, 2001.

[3] J. Barnes. "Programming In Ada 95", first edition. Addison-Wesley, 1995.

[4] M. Feldman, E. Koffman "Ada 95: Problem Solving and Program Design". Addison-Wesley,1996.

[5] Burns A. and Wellings A. : "Concurrency in Ada". Cambridge, 1995.

[6] José María Drake y Elena Mediavilla. "*Técnicas de entrada/salida de tiempo real*",
Apuntes de la asignatura "Instrumentación de Tiempo Real", Santander, Feb.1998.

[7] "*MaRTE OS home page, (Minimal Real-Time Operating System for Embedded Applications)*", Mario Aldea y Michael González, Dic.2001.
http://marte.unican.es/

[8] "*Cómo programar el puerto paralelo*"
http://www.insflug.org/COMOs/Programacion-Paralelo-Como/Programacion-Paralelo-Como-3.html.

[9] Descarga de información sobre los motores y encoders en formato PDF.
http://www.Faulhaber.com/

[10] Descarga de información sobre controlador MMC en formato PDF. http://www.solutions-cubed.com.

[11] Descarga de información sobre chasis.
http://www.microbotna.com/soccer_set.htm

[12] Información sobre contador HCTL-20032.
http://www.agilent.com.

[12] Información sobre contador HCTL-20032.
http://www.agilent.com.

[13] Información sobre PCM-3347 F..
http://www.advantech.com.

[14] mar-mgh-2001.pdf. “MaRTE OS, an Ada Kernel for Real Time Embebedded Aplications”

[15] Información sobre la Robocup..http://robocup.org.

[16] Información sobre la Robocup http://www.itea.uq.edu.au

[17] Página del Departamento de Computadores y Tiempo Real. http://www.ctr.unican.es

[18] Información sobre la Universidad de Cornell. http://robocup.mae.cornell.edu/.

[19] “Migración de un sistema operativo de tiempo real, MaRTE OS, a un microcontrolador”. Alberto Gutiérrez Castro.2003.