Universidad Carlos III de Madrid

Escuela Politécnica Superior
Departamento de Informática

Trabajo de Fin de Grado
Marzo 2014

DISEÑO, DESARROLLO E IMPLANTACIÓN DE UNA PLATAFORMA EMPOTRADA PARA EL CONTROL DE SISTEMAS ROBÓTICOS

Autor: Kamal El Maataoui
Tutor: Javier Fernández Muñoz
Cotutor: Alberto Jardón Huete

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres su constante apoyo y esfuerzo brindándome la oportunidad de estudiar este grado. Sin su ayuda no hubiera podido alcanzar esta meta.

En segundo lugar quiero agradecer a mis hermanos y a mi tío que han sido buenos consejeros y que me han acompañado en este recorrido con momentos llenos de alegrías y dificultades.

A mis compañeros, con los que he pasado una fase importante de mi vida disfrutando de momentos de trabajo y de ocio.
Por último quiero dar especialmente las gracias a mi profesor y tutor del trabajo, Javier Muñoz por su paciencia, comprensión y tiempo dedicado, a todos los profesores que me han impartido clase durante la carrera y al personal de servicio de la universidad.

A todo vosotros, un fuerte abrazo.

Resumen

La robótica es un indicador de desarrollo tecnológico y progreso. Las empresas que invierten en robots no solamente consiguen altos niveles de productividad y competitividad, sino que también logran crear una imagen de modernidad.

Este proyecto trate de construir una plataforma para sistemas robóticos que ofrezca herramientas para desarrollar fácilmente programas de control.

La plataforma especifica modelos de hardware y un modelado general del sistema robótico para hacer un diseño flexible y escalable.

Abstract

Robotics is an indicator of technological development and progress. Enterprises investing in robots not only attain high rates of productivity and competitiveness, but also create an image of modernity.

This project aims to build a platform for robotic systems offering tools to easily develop software of robotic control.

The platform specifies the hardware models and a general robotic system modeling to make a flexible and scalable design.

Índice general

Índice de Ilustraciones

Índice de Tablas

1. Introducción y objetivos

Este documento recoge las fases de Diseño, Desarrollo e Implantación de una Plataforma Empotrada para el Control de Sistemas Robóticos (en adelante, "la plataforma de control", o "el sistema").

"La robótica es sinónimo de progreso y desarrollo tecnológico. Los países y las empresas que cuentan con una fuerte presencia de robots no solamente consiguen altos niveles de competitividad y productividad, sino también transmiten una imagen de modernidad. En los países más desarrollados, las inversiones en tecnologías relacionadas con la robótica han crecido de forma significativa y muy por encima de otros sectores." [1 El Libro Blanco de la Robótica en España].

1.1.Motivación

La motivación de este trabajo surge, en primer lugar, para cubrir la carencia de firmware basado en código abierto para el Laboratorio de Robótica de la Universidad Carlos III de Madrid. El laboratorio participa en competiciones de robots entre las universidades Españolas.

Los robots con los que cuenta el laboratorio, Bioloid y Robonova, están dotados con capacidades de cómputo y almacenamiento muy limitados, y con entornos de desarrollo propietarios, incompatibles con otros lenguajes de programación muy extendidos como C y C++. A estos efectos, el laboratorio precisa disponer de una alternativa al controlador de esos robots, suministrado de fábrica.

Se necesita de hardware con mayores prestaciones para llevar a cabo tareas con altas necesidades de cómputo y almacenamiento, como por ejemplo: el procesamiento de imágenes, bases de datos para algoritmos de inteligencia artificial,...etc. En cuanto al software, se necesita de lenguajes de programación, estructurados y orientados a objetos, sistemas de tiempo-real, ya que los sistemas robóticos, en función de la complejidad de sus objetivos, suelen ser sistemas críticos.

1.2.Objetivos

El propósito principal de este trabajo es construir una plataforma software, basada en Linux, para agilizar el desarrollo de software de control de sistemas robóticos, o **automáticos**. Desde un enfoque global, se identifican tres objetivos para lograr tal propósito:

Sistema Linux funcional y pulido

Con este objetivo se pretende construir un sistema funcional, con los módulos necesarios y con menos programas redundantes, para reducir el tamaño del Firmware. Otro aspecto importante es la integración de un Framework de tiempo-real.

Framework de apoyo

Con este segundo objetivo, se pretende añade una nueva capa de utilidad, con componentes software específicos a sistemas robóticos. Servirá de herramienta para reducir el tiempo de inmersión en la plataforma de control, y por tanto, el tiempo de producción de Firmware de control se verá aminorado.

Componentes hardware de modelo

En este objetivo, se definen componentes hardware que servirán de modelo para la elección de componentes compatibles con el Firmware de control. Se van a establecer las funcionalidades y restricciones de los componentes del modelo.

1.3.Fases del proyecto

El desarrollo de este proyecto se ha dividido en las siguientes fases:

- Análisis de las necesidades del laboratorio de robótica de la UC3M.
- Definición de los objetivos a conseguir.
- Especificación de los requisitos de usuario y del sistema.
- Planificación y presupuesto del proyecto.
- Diseño de la plataforma de control.
- Implementación de una plataforma ejemplar y pruebas.

1.4.Estructura de la memoria

Este documento se estructura de la siguiente forma:

- Capítulo 1: Introducción y objetivos

  En este apartado se realiza una breve introducción, se explican los objetivos que se pretende alcanzar con este proyecto, y la estructura de la memoria.

- Capítulo 2: Estado del arte

  En este capítulo ponemos en contexto el trabajo realizado, explicando las tecnologías y componentes utilizados.

- Capítulo 3: La Plataforma de Control

  En este capítulo se define con más detalle la plataforma de control, se estudia sus finalidades y alcance, y por último, se procede a la división de la plataforma en subsistemas.

- Capítulo 4: Análisis de la Plataforma de Control

  En este capítulo se realiza el análisis de lo que debe realizar el sistema. El análisis consta de los requisitos de usuario y los casos de uso. Y finalmente los requisitos de software y las correspondientes matrices de trazabilidad.

- Capítulo 5: Diseño de la Plataforma de Control

  En el capítulo se detalla el diseño del sistema, desglosando el diseño del hardware, software del controlador y el Framework de apoyo.

- Capítulo 6: Implantación y Pruebas

  En esta capitulo se seleccionan los componentes para implementar una plataforma para realizar las pruebas. Y finalmente se realizan las pruebas en base a los requisitos de software

- Capítulo 7: Planificación y Presupuesto

En el capítulo se presenta la planificación llevada a cabo en el trabajo y el presupuesto elaborado.

- Capítulo 8: Conclusiones y Líneas Futuras

En este capítulo se expone las conclusiones alcanzadas a la finalización del proyecto, y las posibles mejoras y ampliaciones que se pueden realizar.

- Capítulo 9: Anexos

Este apartado contiene algunos anexos a este documento.

- Capítulo 10: Glosario de Términos

En el capítulo se incluyen los términos poco conocidos acompañados de su correspondiente definición.

- Capítulo 11: REFERENCIAS Y BIBLIOGRAFIA

En este capítulo se incluyen las referencias utilizadas para la realización del trabajo: textos, libros, páginas webs.

2. Estado del arte

En este capítulo describiremos los componentes y tecnologías en robótica relacionados con este trabajo. Realizaremos un vistazo por las principales características para introducir al lector la tecnología con la que vamos a trabajar para después, en los siguientes capítulos, analizar en profundidad cada elemento y su funcionamiento dentro del proyecto.

2.1.Robótica[2]

La robótica no es un concepto del futuro. Los robots están en todas partes y será mayor esta entrada en el futuro. Desde las lavadoras hasta los cohetes espaciales, utilizan tecnologías relacionadas con la robótica.

La revolución tecnológica nos ha puesto al alcance de conseguir el sueño de muchas generaciones. Pero esto comenzó antaño. El deseo de crear un ser parecido a nosotros viene desde los primeros tiempos. En el antiguo Egipto nos encontramos estatuas de dioses que tenían brazos mecánicos operados por los sacerdotes. En el siglo XIX, se crearon robots que jugaban al ajedrez, aunque la verdad es que eran dirigidos por una persona de pequeña estatura en su interior.

La palabra robot se remonta a comienzos del siglo XIX. El checo Karel Capek la utilizó por primera vez en su obra dramática *Rossum's Universal Robots* para referirse a un conjunto de máquinas que realizaban tareas mecánicas y repetitivas. Isaac Asimov hizo conocido el término ya que lo utilizó en algunos de sus libros de ciencia ficción. A mediados del siglo XX el crecimiento de la robótica ha sido exponencial, debido al la reducción de tamaño y de costos y el aumento de la capacidad de cálculo de los procesadores. Aún queda mucho camino para construir un robot que se parezca a un humano.

A continuación presentamos una clasificación de los robots según su utilidad práctica:

- Industriales: se usan dentro de un proceso de trabajo industrial. Este tipo de robot es el que ha sido más desarrollado a lo largo de la historia.
- Espaciales: se manejan en zonas inexploradas y a larga distancia de su centro de control.
- Médicos: son empleados en intervenciones médicas y como complemento para personas con alguna discapacidad.
- Doméstico: se emplean en labores del hogar. Por ejemplo, aspiradoras inteligentes, lavadoras, frigoríficos,…etc. que modifican su comportamiento de forma autónoma según el ambiente en el que trabajan.
- Sociales: robots empleados en ambientes sociales (como películas y supermercados) con funciones de comunicación intensiva con los humanos. En estos casos se pone énfasis en el aspecto del robot, y se estudia la interfaz con el humano para realizar una comunicación completa (se buscan gestos, tonos parecidos a los humanos).
- Agrícolas: la robótica comenzó enfocándose en la industria pero en los últimos años ha crecido exponencialmente su uso en el sector agrícola y ganadero. Por

ejemplo, cosechadoras autónomas, sembradoras controladas por mapas satelitales, fumigadoras robotizados y otros dispositivos hicieron su aparición dentro de lo que actualmente se conoce como agricultura de precisión.

Según la ubicación de la inteligencia del robot, se clasifican los robots en:

- Autónomos: la inteligencia está ubicada en el mismo robot. Puede comunicarse con otros o con un sistema central, pero los aspectos esenciales de funcionamiento se resuelven en forma independiente.
- Control automatizado: la mayor parte de la inteligencia se encuentra en un sistema central. Los sensores pueden ser locales, es decir que envían la información obtenida a ese sistema central, o globales. El sistema central comunica a los robots las acciones que deben realizar.
- Híbridos: son robots autónomos que en ciertos momentos del proceso pueden ser controlados por humanos o por un sistema central. Un ejemplo son los robots que se utilizan en misiones espaciales, que operan autónomamente pero ante algún percance pueden ser dirigidos desde nuestro planeta.

2.2.Actuadores

Si un robots observara el mundo sin interactuar con el sería un robots con limitaciones. La intención es que pueda modificar su estado y el del ambiente según la información que obtiene del proceso. Para ello disponemos de motores, displays, leds, zumbadores… Al conjunto de estos dispositivos se les denomina actuadores.

Los actuadores más sencillos de usar son las bombillas o leds. Simplemente con conectarlas a alguna salida del procesador y proveer de alimentación necesaria podremos encenderlas y apagarlas con nuestro programa. Los motores son actuadores más complejos y que proporcional movilidad y libertad de movimiento a los robots.

Por definición, el motor electico es un dispositivo electromotriz, es decir, que convierte energía eléctrica en energía motriz. Todos los motores tienen un eje de salida donde acoplan un engranaje, una rueda, una polea o cualquier mecanismo capaz de transmitir el movimiento creado por el motor. Las características del motor son el tamaño, el peso, la velocidad (que se mide en revoluciones por minuto, RPM), torque, tensión y el precio. Existen diferentes tipos de motores:

Motores de corriente continua (CC): Son los motores más empleados en robótica. El funcionamiento se basa en la acción de campos magnéticos que hacen girar el rotor (eje interno) en dirección opuesta al estator (imán interno o bobina). Para cambiar la dirección de giro en estos motores sólo debemos invertir la polaridad de su

alimentación eléctrica. Se utilizan acompañados de un sistema importante que reducen la velocidad y proporcionan mayor fuerza.

Motores paso a paso (PAP): Se diferencia de uno convencional en que pueden ubicar su eje en posiciones fijas o pasos, con lo cual es capaz de mantener la posición. Esta peculiaridad se debe a la construcción del motor: por un lado, tiene el rotor constituido por un imán permanente, y por el otro, es estator construido por bobinas. Al alimentar alguna de esas bobinas, se atrae el polo magnético del rotor opuesto al polo generado por la bobina, y este permanece en esa posición hasta que la bobina deje de generar el campo magnético y se active otra bobina, la cual hace avanzar o retroceder al rotor. De esta manera, al variar los campos magnéticos en torno al eje del motor se logra que gire.

Servomotores: El servo es un pequeño pero potente dispositivo que disponen en su interior de un pequeño motor con un reductor de velocidad y un multiplicador de fuerza. También cuenta con un pequeño circuito eléctrico encargado de gobernar el sistema. El recorrido del eje de salida es de 180º en muchos casos, pero se puede modificar fácilmente hasta alcanzar los 360º, actuando entonces como un motor normal.

El control de posición se logra gracias a un potenciómetro conectado al eje de salida. Este controla un PWM (Pulse Width Moduler, modulador de anchura de pulso) interno para compararlo con la entrada PWM externa del servo, mediante un sistema diferencial y así modificar la posición del eje de salida hasta que los valores se igualen y el servo se detenga en la posición indicada. En esta situación, el motor deja de consumir corriente y sólo circula una pequeña cantidad hasta el circuito interno. Si forzamos aquí el servo, el control diferencial interno lo detecta y manda la corriente necesaria para corregir la posición.

Ilustración 1: Ejemplo de servomotores

Para controlar un servo tenemos que aplicar un pulso y una frecuencia determinada. Los servos disponen de tres cables, dos para la alimentación y uno para aplicar los pulsos de control que hará que el circuito interno ponga al servo en la posición indicada.

Estos servos se utilizan mucho en automóviles y aviones radiocontrolados, y como limitación trae que son en general lentos.

2.3.Comunicación serial

La transmisión en serie es el proceso de envió de datos bit a bit, secuencialmente sobre un canal de comunicación o bus. Se diferencia de la comunicación en paralelo, en la que varios bits se envían a la vez.

Existen tres tipos de comunicación en serie:

Simplex: La comunicación es unidireccional entre el emisor y receptor. Se usa en radiodifusión donde los receptores no necesitan enviar ningún tipo de dato al transmisor.

Duplex, half duplex o semi-duplex: La comunicación es bidireccional pero no se establece de manera simultánea.

Full Duplex: Similar al dúplex, pero los datos se desplazan en los dos sentidos. Para que sea posible ambos emisores deben utilizar frecuencias distintas de transmisión o dos caminos de comunicación separados.

Los estándares que más se utilizan para representar la información son el RS-232 y el TTL.

2.4.Kernel de Linux

El kernel de Linux es un kernel de sistema operativo de tipo Unix usados por los sistemas operativos que se basados en él. El kernel de Linux es un ejemplo de software libre y abierto.

Fue lanzado el 25 de agosto de 1991 por Linux Torvalds, que lo desarrollo cuando estaba terminando sus estudios en informática. Linux rápidamente acumuló desarrolladores y usuarios quienes adaptaron este código para otros proyectos de software libre.

Características técnicas:

Actualmente es un kernel monolítico hibrido. Los drivers de dispositivos y las extensiones del kernel se ejecutan en un espacio privilegiado, con total acceso al hardware, aunque algunas excepciones corren en el espacio de usuario. El sistema gráfico que la mayoría de usuarios usa no corre en el kernel, en contraste con Microsoft Windows. Los drivers de dispositivo pueden ser fácilmente configurables como módulos, e iniciarlos y matarlos mientras el sistema está en funcionamiento.

El harware es también incorporado en la jerarquía de ficheros. Los interfaces de drivers de dispositivos a las aplicaciones de usuario entrando en los directorios/dev y /sys. La

información de procesos también es mapeada al sistema de ficheros a través del directorio /proc.

A pesar de que no fue diseñado para ser portable, Linux es uno de los kernel de sistemas operativos más portados, corriendo en diverso rango de sistemas desde la arquitectura ARM hasta los ordenadores mainframes con arquitectura Z de IBM.

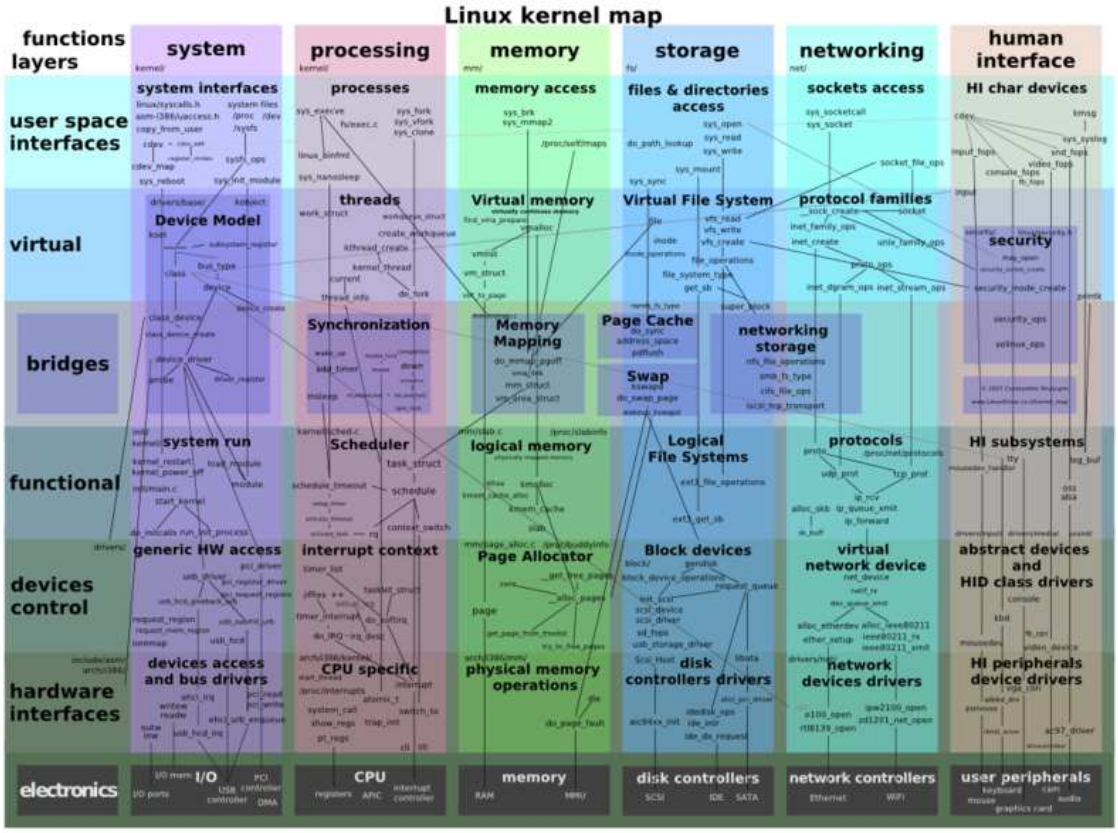


Ilustración 2: Mapa del kernel de Linux

Uno de los errores más típicos es el error “panic. En Linux, “panic” es un error irrecuperable detectado por el kernel, como opuesto a errores detectados por código en el espacio de usuario. Es posible desde el código del kernel indicar esta condición llamando a la función “panic” que está en el archivo de cabeceras “sys/system.h”. A pesar de esta posibilidad, la mayoría de los errores “panic” son el resultado de excepciones del procesador no controladas en el código, como referencias a direcciones de memoria inválidas. También pueden indicar errores en el hardware, como celdas de memoria RAM dañadas o errores en funciones aritméticas en el procesador causadas por un bug en el procesador, sobrecalentamiento o daño o por un error software.

Linux fue escrito en lenguaje C, soportado por GCC, el cual introdujo un gran número de extensiones y cambios al estándar de C, junto con un número de secciones cortas escritas en lenguaje ensamblador.

La seguridad es un tema muy publicitado en relación con el núcleo de Linux ya que un gran número de errores en el kernel pueden dar lugar a posibles fallos de seguridad, ya que permiten una escalada de privilegios o crear vectores de ataque de denegación de servicio. Los críticos han acusado a los desarrolladores del kernel de encubrir estos fallos de seguridad o por lo menos de no publicarlos.

El kernel de Linux puede ejecutarse sobre muchas arquitecturas de máquina virtual, tanto como host del sistema operativo como cliente. Esta máquina virtual emula usualmente a la familia de procesadores x86, aunque también son emuladores procesadores de PowerPC o ARM.

Se siguen lanzando nuevas versiones del núcleo de Linux. Estos son los núcleos denominados “vanilla”, lo que significa que no han sido modificados por nadie.

2.5. Sistemas embebidos

Un sistema embebido es aquel que se diseña para realizar pequeñas tareas, a diferencia de un computador que cubre un mayor rango de necesidades. Podemos encontrarlos en multitud de aparatos (televisores, lavadoras, periféricos…).

Las características destacadas de un sistema embebido son:

- Concurrencia: Los componentes del sistema controlado funcionan simultáneamente.
- Fiabilidad y seguridad: Hay que tener en cuenta los posibles fallos y excepciones en el diseño del sistema.
- Eficiencia: Gran parte de los sistemas de control deben responder con gran rapidez a los cambios en el sistema controlado.
- Interacción con dispositivos físicos: Se integran dentro del dispositivo que controlan y pueden interactuar con distintos tipos de dispositivos (impresoras, teclados…).
- Bajo peso y consumo.

	PROPÓSITO GENERAL	TIEMPO REAL
NO EMPOTRADOS	Supercomputadores Servidores Estaciones de trabajo PC's Calculadoras *Cálculo científico* *Gestión (bancos, empresas)* *Bases de datos*	Tarjetas microprocesadores + tarjetas E/S + bus Autómatas Programables Reguladores digitales *Control industrial* *Simuladores de vuelo* *Robótica*
EMPOTRADOS		Tarjetas microprocesadores + tarjetas E/S + bus Microcontroladores, DSPs *Electrodomésticos* *Aeronáutica* *Teléfonos móviles*

Ilustración 3: Clasificación de computadores

Los tipos de arquitectura hardware para dispositivos empotrados son los siguientes:

FPGA

Una FPGA (Field Programmable Gate Array) se definen como dispositivos con bloques electrónicos lógicos variados (puertas lógicas, biestables, bloques de memoria…). La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip.

Microcontrolador

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Incluye CPU, memoria, unidades de entrada y salida. Permite conseguir una computadora en funcionamiento con muy pocos o ningún chips externos de apoyo. Hay un gran rango de posibilidades:
Se elige el más ajustado al sistema particular tratando de reducir costes y consumo de energía.
Representan la inmensa mayoría de los chips de computadoras vendidos.

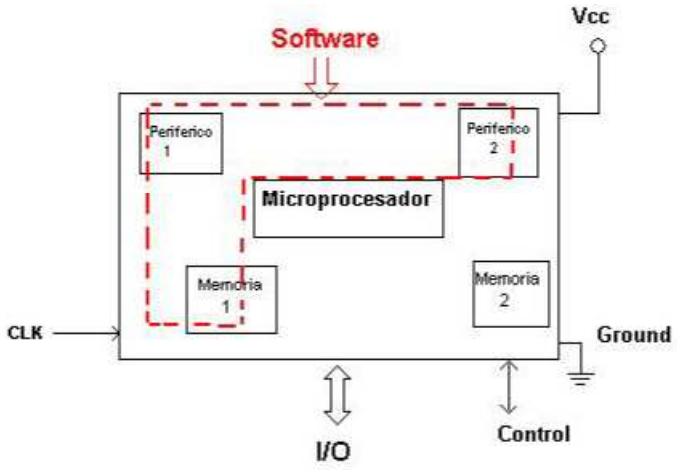


Ilustración 4: Esquema de un microcontrolador

La configuración típica de un microcontrolador es:

- CPU: Existe un gran variedad
  - 8 bits, 16 bits, 32 bits.
  - Arquitectura Von Neumann o Harvard
  - RISC, CISC, VLIW, etc
- Memoria volátil (RAM)
- Memoria no volátil (ROM, PROM, EEPROM, FLASH)
- Sistemas de E/S
  - Puertos Serie (UART)
  - Comunicaciones (CAN-BUS, etc).
- Otros periféricos (reloj, watchdogs conversores A/D y D/A).

Sistemas completos en un chip (SoC)

Los sistemas completos en un chip utilizan la misma filosofía que los microcontroladores, la única diferencia es la escala. Son mucho más potentes y suelen necesitar almacenamiento externo y dispositivos de entrada y salida externos más complejos.

Placas madres

Las placas madre son definiciones estándar de placas eléctricas para sistemas embebidos. Definen la forma de interconexión de elementos y las medidas, facilitando futuras ampliaciones consiguiendo abaratar costes. Ejemplos: Estándar PC104, micro ATX, mini ITX, etc.

3. Estudio Previo

En este capítulo, se pretende identificar subsistemas con funcionalidades autocontenidas. Paso siguiente, establecer una jerarquía entre los subsistemas, y definir las fronteras de estos. El objetivo de esta división es transformar el análisis integral en análisis modular, mas manteniendo el propósito global de la plataforma de control. La visión jerárquica de los subsistemas, definida en este capítulo, en el siguiente capítulo nos permitirá organizar los casos de uso en subsistemas para facilitar la comprensión de la sección de requisitos.

3.1. Objetivos de la Plataforma de Control

Antes de proceder a la definición de la plataforma, conviene describir sus objetivos prioritarios:

Sistema Linux funcional

Para ofrecer soporte a las familias de hardware, para las cuales esta plataforma fue diseñada, hay que generar una imagen de Kernel con los módulos necesarios para lograr un mayor soporte de hardware. Por defecto, en una imagen de Kernel, vienen habilitados una gran variedad de módulos. Con ello, en una máquina, el Kernel es capaz de reconocer e instalar un mayor número de dispositivos. Por otra parte, una distribución corriente, por defecto viene con muchos programas de utilidad para los usuarios de escritorio.

En un sistema empotrado, el tamaño sí importa, por lo que, a cuanto más software redundante se elimina, menos tamaño tendría el sistema final. Y lo más importante es la reducción del uso de recursos (por ejemplo: el acceso por la consola de comandos, consume muy pocos recursos comparado con el acceso con las aplicaciones de escritorio).

Con este objetivo se pretende construir un sistema funcional, con los módulos necesarios y con menos programas redundantes.

Comunicación a través de puertos serie (UART)

La comunicación en serie es uno de los pilares de la robótica. Ojeando en internet, uno puede encontrar un sinfín de variedades de conversores de señales y protocolos de comunicación para conectar dos, o más, dispositivos y lograr que intercambien datos. Para no perderse en esas variedades, nos concentraremos en la base, en el estándar. Los estándares nos ofrecen documentación, soporte por parte de la comunidad y compatibilidad entre versiones.

Para este proyecto, se va a adoptar el estándar RS-232[3], tanto para los conectores DB-9 como los USB. Es un estándar extensamente usado, viene integrado en los equipos que empleamos en nuestras vidas cotidianas, requiere un hardware simple y es fácil de adaptar a los requisitos del proyecto.

El sistema operativo ofrece un conjunto de parámetros para configurar los puertos seriales. Con este objetivo, brindamos al usuario una interfaz práctica, directa y flexible para configurar los puertos.

Sistemas de tiempo-real (STR)

Los sistemas robóticos actuales incorporan en muchos casos sistemas empotrados de tiempo-real. Lo que tienen en especial estos sistemas, es que para que el funcionamiento del sistema sea correcto deben cumplirse los tiempos de respuesta (*deadlines*) marcados para cada tarea. Los sistemas operativos, generalmente usados en los ordenadores personales, como Linux y Windows, no ofrecen ningún soporte de tiempo-real. Eso es porque, las tareas que el usuario realiza son interactivas y el tiempo no es un factor decisivo para llevarlas a cabo.

Para cubrir los requisitos de tiempo-real, se va emplear el framework Xenomai. Es un pequeño núcleo que se integra con el kernel de Linux, y requiere la configuración y compilación del kernel. Al igual que el Kernel, ofrece una interfaz, nueva, de llamadas del sistema, pero, con la peculiaridad de que su latencia es mínima; una requisito esencial de un sistema de tiempo-real.

En un sistema de tiempo-real, las tareas tienen un tiempo de respuesta, conocido a priori, por lo que la latencia del sistema en sí, es un factor que se tiene muy en cuenta. De hecho, las llamadas al sistema están diseñadas de tal forma que su tiempo de cómputo sea despreciable, para no influir en el tiempo de respuesta de las tareas planificadas. De entre los factores que influyen en la latencia son los componentes hardware de la computadora, como por ejemplo el módulo de ahorro de energía, ya que el rendimiento del procesador depende el consumo de energía.

Con este objetivo, se logra configurar el Kernel para integrar el Xenomai, y ejecutar las pruebas ofrecidas por este, para verificar su correcta integración y funcionamiento.

Gestión de paquetes de datos de la comunicación en serie

La gestión de paquetes de datos de un sistema robotizado es esencial. Entre otros aspectos, puede influir enormemente en la complejidad del sistema. Las principales tarea del sistema de control son: enviar instrucciones de control, leer datos sensoriales y procesar esos datos. Es de vital importancia tener una política para producir los paquetes de datos de control, y para consumir los paquetes de datos sensoriales.

El envío de paquetes a los actuadores no es siempre trivial. En muchos casos hay que realizar esperas entre los envíos. Crear esperas en un sistema crítico, no es conveniente, porque esas esperas se pueden aprovechar para hacer otras tareas.

Lo que se pretende con este objetivo es ofrecer un método de gestión, con cierta autonomía, e intuitivo al usuario. ¡Porqué complicar el código con bucles de esperas para enviar paquetes, si el gestor de paquetes puede encargar de eso!

Otra de las facilidades es orientar la gestión de paquetes a eventos, de forma que, cuando haya datos, se lance un fragmento de código para tratarlo. Ofrecer la

posibilidad de ordinar los paquetes en el tiempo, y encadenarlas, de modo que, cuando un paquete haya sido tratado, envíe otros dependientes de él.

Control remoto

El control remoto es una práctica frecuente en los sistemas automáticos. En domótica, y otros campos, es deseable poder controlar los aparatos de la casa vía internet, por ejemplo, saber la temperatura de la casa, poder subir o bajar persianas y controlar otros aparatos conectados mediante el controlador.

Con este objetivo se pretende proveer al sistema de control de una capa de interconexión con dispositivos de control remotos. De modo que, éstos pueden estar conectados a la misma máquina que al controlador, o en máquinas aisladas e interconectadas vía internet. Esta topología puede requerir un intermediario, enrutador, entre el controlador y los dispositivos.

Simuladores

La simulación es una utilidad de bajo coste para poner el sistema desarrollado al límite, y probar su correcto funcionamiento. No se pretende crear un simulador, sino, ofrecer las interfaces y estructuras necesarias para conectar el sistema de control con los entornos simulados.

Generación de Log

Los ficheros de log son muy útiles para la identificación de problemas, y la localización de *bugs*. La naturaleza de sistemas embebidos, a los que está orientado este *framework*, no suelen contar con pantallas, por lo que, la información relacionada con determinados sucesos del sistema es inservible.

Con este objetivo se desea cubrir los detalles técnicos de la generación de ficheros log, y la escritura de trazas y sucesos en estos.

Bases de datos embebidas

Las bases de datos son un mecanismo eficiente para el guardado y la recuperación de la información. Para sistemas embebidos, los motores de bases de datos como Oracle, o SQL, son inviables por la cantidad de recursos que consumen, comparada con la cantidad de información que se desea gestionar.

Los sistemas embebidos, por lo general, manejan una pequeña cantidad de datos, por lo que una base de datos ligera, como SQLite, sería ideal. Control de servos Dynamixel[4]

La implementación de software para el control de actuadores no es una tarea sencilla. Generalmente, los fabricantes de servomotores ofrecen los detalles de la interfaz de comandos de sus servos, sin código software. Esto podría ser debido a la inmensa

variedad de lenguajes de programación, y sería el usuario final, quien desarrolle su programa de control en el lenguaje que le convenga.

Se ha elegido la marca Dynamixel de servos, porque los robots del laboratorio de la UC3M están compuestos por este tipo de servos. Además, éstos tienes una interfaz de comandos bastante sencilla, y su uso está bastante extendido. El hecho de que incorporen un microcontrolador, que se encarga de los aspectos técnicos para el control de velocidad y la posición del giro, hacen que las líneas de código y complejidad del programa de control sean menores.

3.2.Definición de la Plataforma de Control

Cuando se habla de plataformas de control de sistemas robóticos, existen dos componentes interrelacionados que se deben combinar cuidadosamente para lograr resultados aceptables del sistema global. Tales componentes, software y hardware, se diseñan en conjunto, o al menos, el desarrollo del primero es influenciado por el diseño del segundo.

El software de la plataforma que se pretende desarrollar en este trabajo, estará acotado por las restricciones del hardware. Porque se parte de la premisa de que el hardware ya está construido. Elaborar nuevos componentes hardware de cero es una tarea laboriosa, y que, ante todo, sale del ámbito de este trabajo.

El sistema se va a dividir en dos componentes independientes; el controlador y el sistema robótico. El primero encapsulará toda la funcionalidad de control, y el segundo, el sistema robótico en sí, sobre el que se ejerce el control. Si existen dos componentes independientes, es necesario un componente intermedio que establece la conexión entre las partes, y aquí el componente de la comunicación serial. Este último, se ocupa de convertir las señales de comunicación para adaptarlas a cada componente, ya que el controlador y el sistema robótico trabajan con señales distintas, en la mayoría de los casos.

La arquitectura global de la plataforma de control se muestra en la siguiente ilustración, en ella se basará la descomposición de esta plataforma en subsistemas:

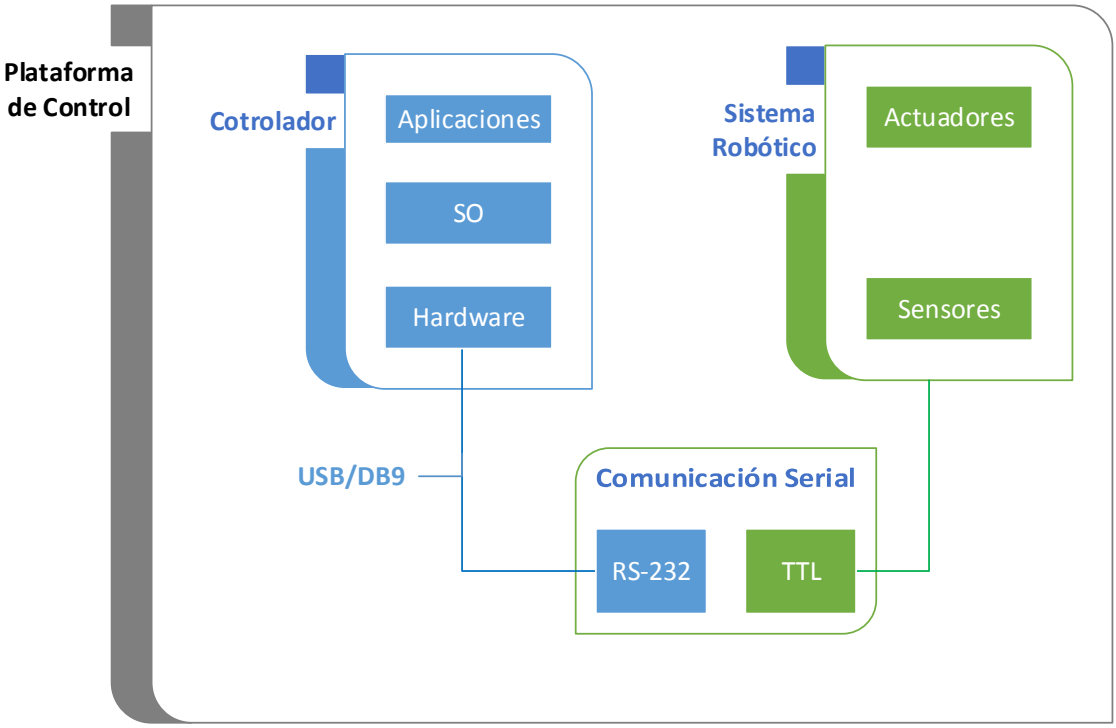


Ilustración 5: arquitectura global; controlador y sistema robótico unidos.

Para cumplir con los objetivos establecidos en el primer capítulo, surge una segunda arquitectura, con una adición: un sistema de control remoto conectado al controlador vía una red TCP/IP[5]:

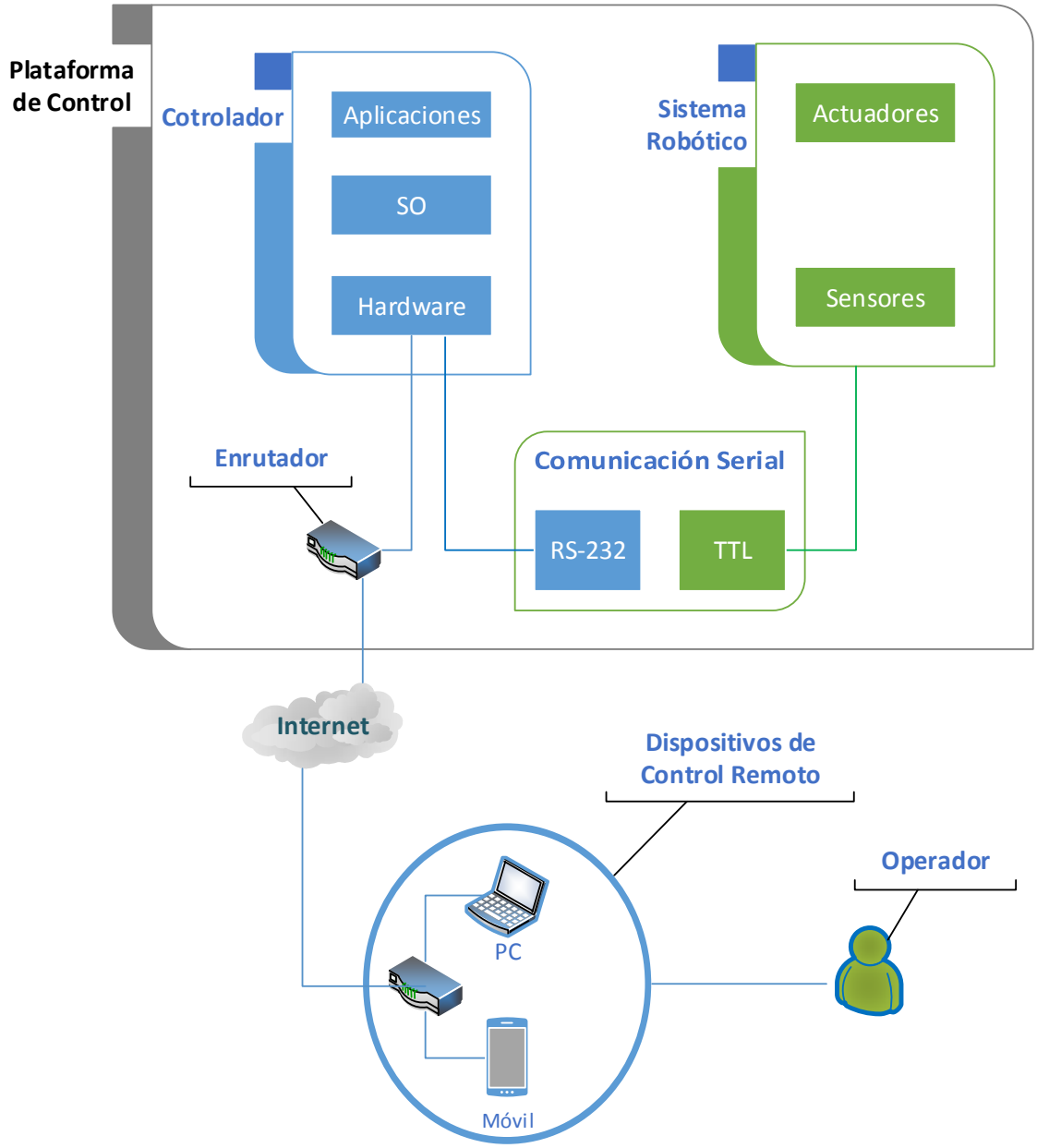


Ilustración 6: arquitectura global; controlador, sistema robótico y control remoto.

En esta segunda arquitectura se permite el envío de datos de control vía internet, o intranet. Desde un ordenador, o móvil inteligente, se puede comunicar con el controlador. Éste recibe comandos de un operador externo al sistema de control, y los traduce en paquetes de control específicos al sistema robótico en cuestión. La plataforma de control especifica que el controlador soporta las comunicaciones TCP/IP, la topología de interconexión del sistema de control con el sistema remoto queda a elección del usuario final de la plataforma.

El sistema robótico, en la definición de la plataforma de control, no tiene una estructura definida, es el usuario final quien da forma al sistema controlado. Ya que es inabordable tener en cuenta todas las estructuras robóticas.

3.3.Alcance del Sistema

Como se ha mencionado en el capítulo 1, la plataforma de control no es el fin, sino, el medio. Al tratarse de una arquitectura de referencia, la vista conceptual de la arquitectura indicará las características de los componentes y conectores, pero no todos los detalles, como qué instancias de componentes y conectores existen, como se da en el caso del diseño de arquitecturas de un sistema concreto.

Teniendo en cuenta lo anteriormente expuesto, la plataforma de control tendrá las finalidades:

- Entorno de desarrollo de firmware de sistemas empotrados.
- Entorno de desarrollo de firmware de sistemas empotrados de tiempo-real.
- Entorno de desarrollo de firmware de sistemas empotrados para el control de sistemas robóticos, tanto con, o sin restricciones de tiempo-real. En esta tercera opción, lo que se aporta al sistema es un framework de apoyo al desarrollo de aplicaciones de control de sistemas robóticos.
- Como combinación de las anteriores, pudiendo ser como la suma de todas ellas.

Cuando se habla de firmware, se refiere al sistema operativo y las aplicaciones de usuario, eso es, el sistema software que se instala en el hardware del controlador.

En cuanto al hardware, se va a especificar los componentes físicos soportados por la plataforma, en los que el firmware desarrollado podrá ser instalado y ejecutado sin problemas de compatibilidad. También, manifestar que el hardware servirá como referencia y no tiene por qué ser idéntico al indicado.

El sistema robótico planteado en la arquitectura de referencia, es un modelo abstracto. En ningún momento se definen las características funcionales o de estructura. Tan solo se plantea un requisito de comunicación; los componentes del sistema robótico deben estar conectados a un bus de control, a través del cual se envían los paquetes de control o sensoriales. Y la comunicación predefinida es en serie.

La plataforma de control puede ser el punto de partida de proyectos en las áreas de robótica, domótica y todos los proyectos con una arquitectura similar la ya definida en este capítulo (ver *Definición de la Plataforma de Control*). La siguiente ilustración expone el alcance de la plataforma de control:

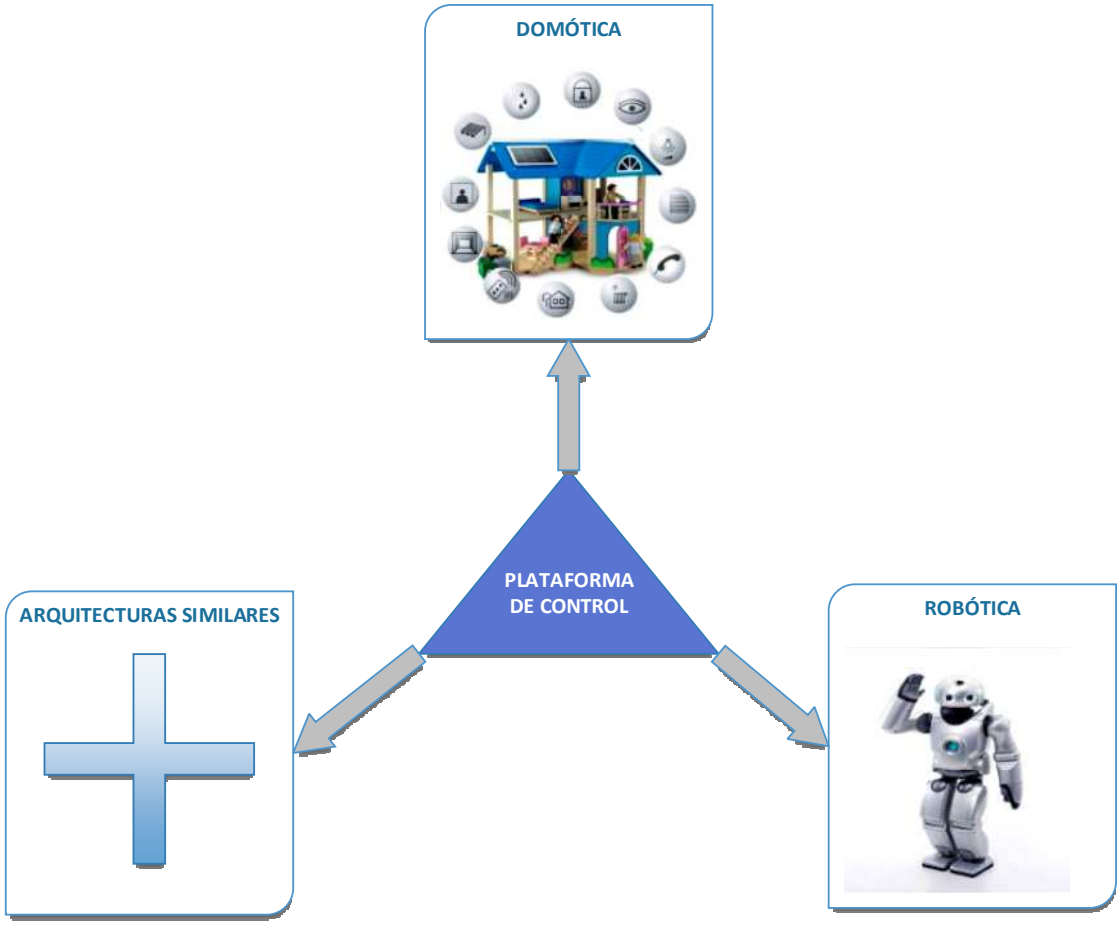


Ilustración 7: Alcance de la plataforma de control

El triángulo del centro, la plataforma de control, representa un sistema base abstracto. Los otros tres círculos de la ilustración, serían sistemas concretos en cada una de las áreas: robótica, domótica y arquitecturas similares a éstas. Las flechas indican una especificación de la plataforma de control.

3.4.Identificación de Subsistemas

Hasta este punto hemos definido la plataforma de control y su alcance. Ahora bien, la identificación de subsistemas se basará en la arquitectura planteada en este capítulo (ver *Ilustración* ***5***).

Dado que el sistema robótico es un modelo abstracto, es absurdo descomponerlo en subsistemas, ya que carece de funcionalidad.

En cuando al sistema de control, en donde reside, en mayor parte, la funcionalidad de la plataforma de control, se identifican dos subsistemas principales:

- Subsistema 1: Entorno de desarrollo y ejecución.

- Ofrece las librerías y módulos necesarios para la generación de software de control y su ejecución. Consistirá en un sistema Linux con las modificaciones requeridas para cubrir las necesidades de ese entorno.
  Por ser un entorno de desarrollo, debe de tener la capacidad de crecer. Eso para un sistema Linux se traduce en permitir la instalación de nuevo software, o paquetes. Y como entorno de ejecución, requiere que el Kernel de Linux esté configurado y compilado para soportar determinados componentes hardware, por ejemplo: las memorias USB, SD y micro SD. Para integrar el framework de tiempo-real Xenomai en Kernel, éste debe ser reconfigurado tal y como específica el desarrollador del framework, y recompilado.

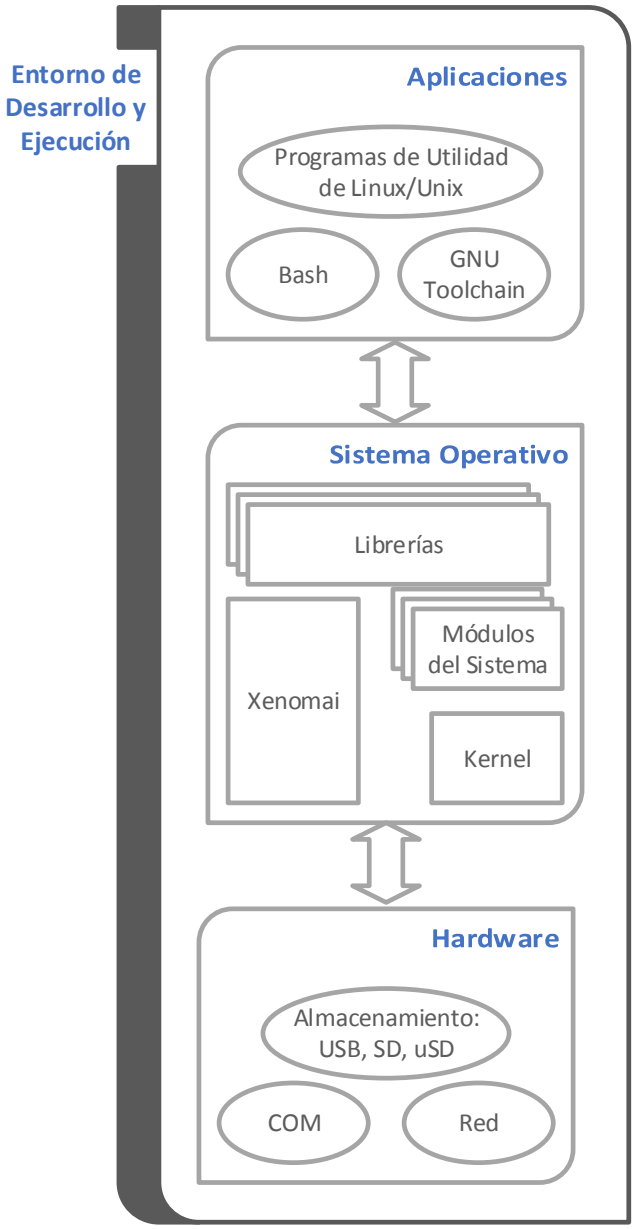


Ilustración 8: Entorno de desarrollo y ejecución.

- Subsistema 2: Framework de apoyo.
- Una vez desarrollada la funcionalidad del subsistema anterior, la funcionalidad de este Framework es, por un lado, gestionar la comunicación serial, y por otro lado, ofrecer entidades, lógicas, abstractas para guiar al desarrollador al buen uso de la plataforma de control.
- Esas entidades abstractas, o interfaces en el lenguaje de programación C++ y otros lenguajes, tendrían dos principales finalidades: un mecanismo de extensión de la plataforma de control, y mantener la consistencia, de las entidades ofrecidas, entre la plataforma de control y el nuevo software creado por desarrollador, es decir, que las nuevas entidades del desarrollador puedan interactuar con las entidades de la plataforma y viceversa, sin importar las implementaciones específicas de cada entidad.

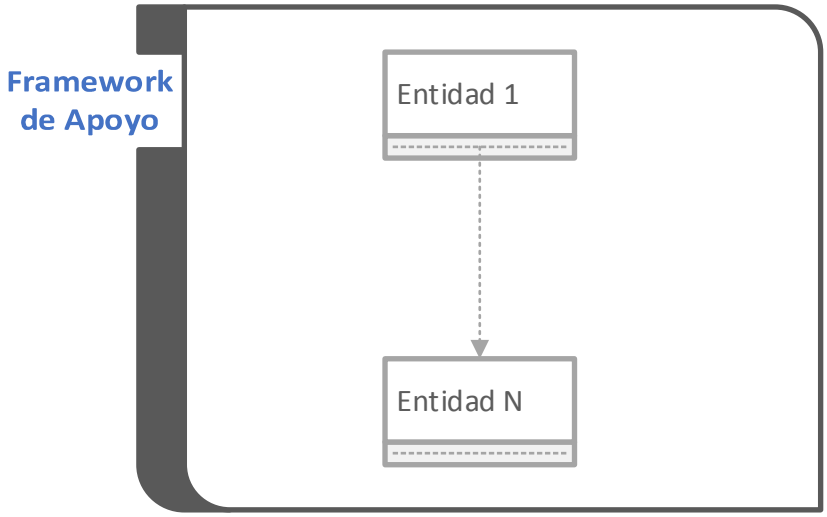


Ilustración 9: El Framework de apoyo.

Tal como se han definido los subsistemas, se puede intuir que el Framework es una capa por encima del primer subsistema. La siguiente ilustración muestra la dependencia entre los dos subsistemas:

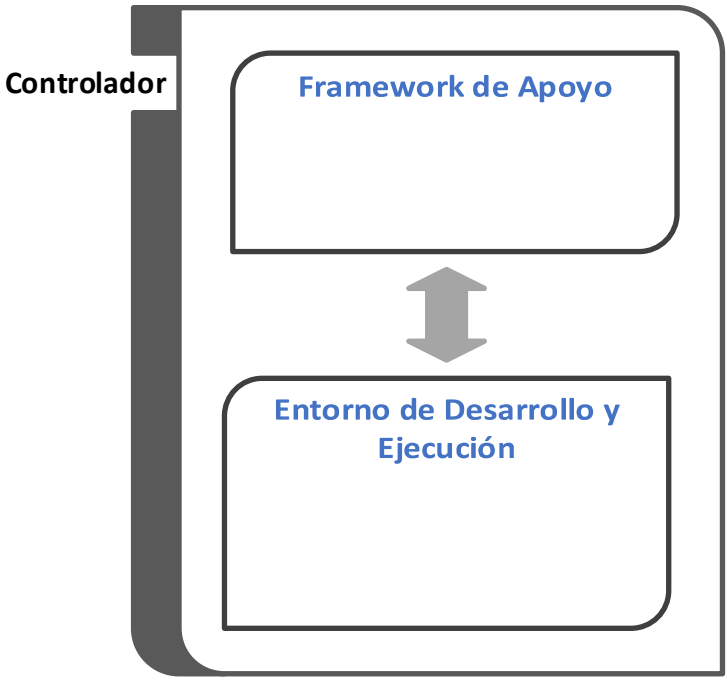


Ilustración 10*: Controlador; relación entre los subsistemas.*

4. Análisis de la Plataforma de Control

En este capítulo se realiza un análisis detallado del sistema que incluye formular los requisitos de usuario, la representar los respectivos casos de uso, y el refinamiento de los requisitos de usuario para generar los de software.

En el capítulo 3, se ha estructurado el sistema en dos subsistemas, los casos de uso y los requisitos de usuario se van a organizar conforme a esa estructuración; por subsistemas.

4.1.Requisitos de usuario

En esta sección se recogen los requisitos de usuario que se han considerado para la realización del trabajo. Estos requisitos constan de los siguientes atributos:

Identificador	Nombre
Descripción	
Necesidad	Opcional/ Deseable /Esencial
Prioridad	Baja/Media/Alta
Estabilidad	Baja/Media/Alta

Tabla 4.1.00: Plantilla requisitos de usuario

- **Identificador**: Se trata de un identificador único para cada requisito.
- **Nombre**: Titulo breve y descriptivo.
- **Descripción**: Comentario conciso y preciso sobre la funcionalidad requerida.
- **Necesidad**: Grado de necesidad de ese requisito. Se clásica en esencial, deseable u opcional.
- **Prioridad**: Establece la prioridad para concretar el orden de realización del conjunto de requisitos. Se valora entre alta, media o baja.
- **Estabilidad**: A cada requisito se le asigna una probabilidad de cambio (alta, media o baja).

Los requisitos de usuario se dividen en dos grupos:

- **Requisitos de capacidad**: Especifican las funcionalidades deseadas a alcanzar. Su identificador tendrá el prefijo *RU.C.Sx.* La "Sx" indicará el subsistema al que corresponde el requisito, por ejemplo, el requisito 3 del subsistema 1 tendría el siguiente identificador: *RU.C.S01-03.*
- **Requisitos de restricción**: Establecen restricciones de cómo estos requisitos funcionales son implementados. Su identificador tendrá el prefijo *RU.R.Sx.* La "Sx" indicará el subsistema al que corresponde el requisito, por ejemplo, el requisito 2 del subsistema 1 tendría el siguiente identificador: *RU.R.S01-02.*

Los requisitos extraídos del análisis del sistema, en dos grupos, son los siguientes:

Requisitos de usuario: subsistema 1

- Requisitos de capacidad:

RU.C.S01-01	Sistema controlador
Descripción	Se tiene que construir un sistema operativo adaptado donde ejecutar los programas del controlador.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.01: RU.C.S01-01, Sistema controlador

RU.C.S01-02	Hardware del controlador
Descripción	El sistema podrá funcionar sobre las arquitecturas ARM y legacy-x86.
Necesidad	Esencial.
Prioridad	Media.
Estabilidad	Alta.

Tabla 4.1.02: RU.C.S01-02, Hardware del controlador

RU.C.S01-03	Soporte de varios tipos de puertos serie
Descripción	El sistema deberá soportar más de un tipo de puertos serie. El sistema debe ofrecer soporte a puertos serie de interfaz hardware USB y DB9. El usuario podrá conectar los actuadores al controlador mediante el puerto serie de interfaz hardware USB y DB9.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.03: RU.C.S01-03, Soporte de varios tipos de puertos serie

RU.C.S01-04	Soporte de interfaces de red
Descripción	El sistema debe soportar comunicaciones Intranet. Debe poder establecer conexiones TCP/IP.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.04: RU.C.S01-04, Soporte de interfaces de red

RU.C.S01-05	Soporte de almacenamiento
Descripción	El sistema debe soportar memorias Flash, por ejemplo: USB, SD, micro-SD.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.05: RU.C.S01-05, Soporte de almacenamiento

RU.C.S01-06	Habilitar un entorno de desarrollo de tiempo-real.
Descripción	Parchear el kernel de Linux para dotarle de características de tiempo-real.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.06: RU.C.S01-06, Habilitar un entorno de desarrollo para tiempo-real

RU.C.S01-07	Crecimiento del sistema
Descripción	El sistema debe proporcionar un gestor de paquetes de software, para instalar, o eliminar, aplicaciones del sistema.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.07: RU.C.S01-07, Crecimiento del sistema

- Requisitos de restricción:

RU.R.S01-01	Sistema basado en Linux
Descripción	El sistema estará basado en Linux.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.08: RU.R.S01-01, Sistema basado en Linux

RU.R.S01-02	Framework de tiempo-real Xenomai
Descripción	El sistema debe integrar el *framework* Xenomai, para generar el entorno de desarrollo de sistemas con restricciones de tiempo-real.
Necesidad	Esencial.
Prioridad	Media.
Estabilidad	Alta.

Tabla 4.1.09: RU.R.S01-02, Framework de tiempo-real Xenomai

Requisitos de usuario: subsistema 2

- Requisitos de capacidad:

RU.C.S02-01	Comunicación simultánea con más de un puerto serie
Descripción	El sistema debe permitir la existencia de múltiples puertos serie abiertos en un momento dado. Se debe permitir la posibilidad de que un puerto serie de interfaz hardware USB y otro DB9 estuvieran abiertos simultáneamente.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.10: RU.C.S02-01, Comunicación simultánea con más de un puerto serie

RU.C.S02-02	Configuración variable de los puertos serie
Descripción	El sistema debe permitir la configuración de los puertos serie.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.11: RU.C.S02-02, Configuración variable de los puertos serie

RU.C.S02-03	Gestión de paquetes
Descripción	El sistema debe de ofrecer un gestor de envío y recepción de paquetes de datos, para temas de esperas y sincronización.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.12: RU.C.S02-03, Gestión de paquetes

RU.C.S02-04	Control remoto
Descripción	Se debe permitir al sistema que sea controlado por remoto vía internet.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.13: RU.C.S02-04, Control remoto

RU.C.S02-05	Simulación
Descripción	Se debe ofrecer las interfaces y estructuras necesarias para conectar el sistema de control a entornos simulados.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.14: RU.C.S02-05, Simulación

RU.C.S02-06	Ficheros de log
Descripción	Se debe generar ficheros log.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.15: RU.C.S02-06, Ficheros de log

RU.C.S02-07	Bases de datos embebidas
Descripción	Se debe integrar la librería de SQLite para crear y consultar bases de datos embebidas.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.16: RU.C.S02-07, Bases de datos embebidas

RU.C.S02-08	Servos Dynamixel
Descripción	Se debe implementar la interfaz de comandos de los servos Dynamixel “Ax-12+”.
Necesidad	Esencial.
Prioridad	Alta.
Estabilidad	Alta.

Tabla 4.1.17: RU.C.S02-08, Servos Dynamixel

- Requisitos de restricción:

RU.R.S02-01	Codificación del framework en lenguaje C++
Descripción	El framework se desarrollara en lenguaje C++.
Necesidad	Esencial.
Prioridad	Media.
Estabilidad	Alta.

Tabla 4.1.18: RU.R.S02-01, Codificación del framework en lenguaje C++

4.2.Casos de Uso

En esta sección se detallan los casos de uso considerados en el análisis del sistema. Estos casos han sido elaborados mediante el análisis de los requisitos de usuario. El objetivo es servir de base de conocimiento que permita refinar los requisitos de usuario en requisitos software.

Diagrama de casos de uso

En el diagrama de casos de uso se encontraran los siguientes elementos (la notación utilizada para la implementación ha sido UML [6])

- Actor: entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad. El actor puede ser humano o también puede referirse a cualquier sistema externo así como a entidades abstractas como el tiempo. En el caso de actores humanos, éstos suelen corresponderse con los roles del sistema, de forma que un mismo individuo puede corresponderse con más de un actor.
- Caso de Uso: Representa la acción o evento que realiza el usuario dentro de un escenario
- Escenario: Determina los límites del sistema, es una secuencia de pasos que realiza el actor principal o el sistema, en donde cada paso se escribe como una oración sobre una meta que se cumple.
- Relación de asociación: Relación entre un actor y un caso de uso que denota la participación del actor en dicho caso de uso.
- Relación de inclusión: <<incluir>> Relación de dependencia entre dos casos de uso que denota la inclusión del comportamiento de un escenario en otro.
- Relación de especialización: <<extender>> Relación de dependencia entre dos casos de uso que denota que un caso de uso es una especialización de otro.

Como hemos mencionado anteriormente, los casos de usos también se van a organizar por los subsistemas definidos en el capítulo 3.

- El diagrama de casos de uso: subsistema 1

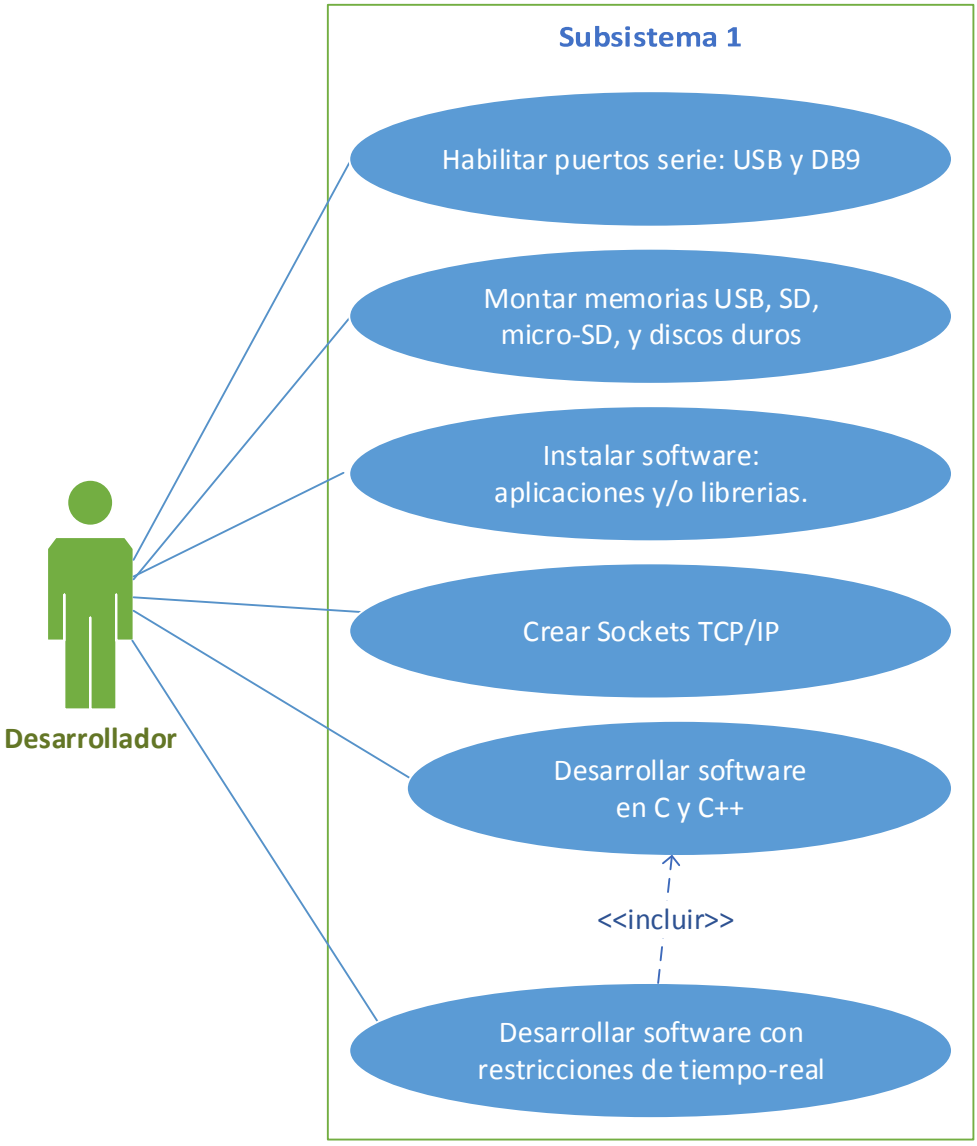


Ilustración 11: diagrama de casos de uso (subsistema 1)

- El diagrama de casos de uso: subsistema 2

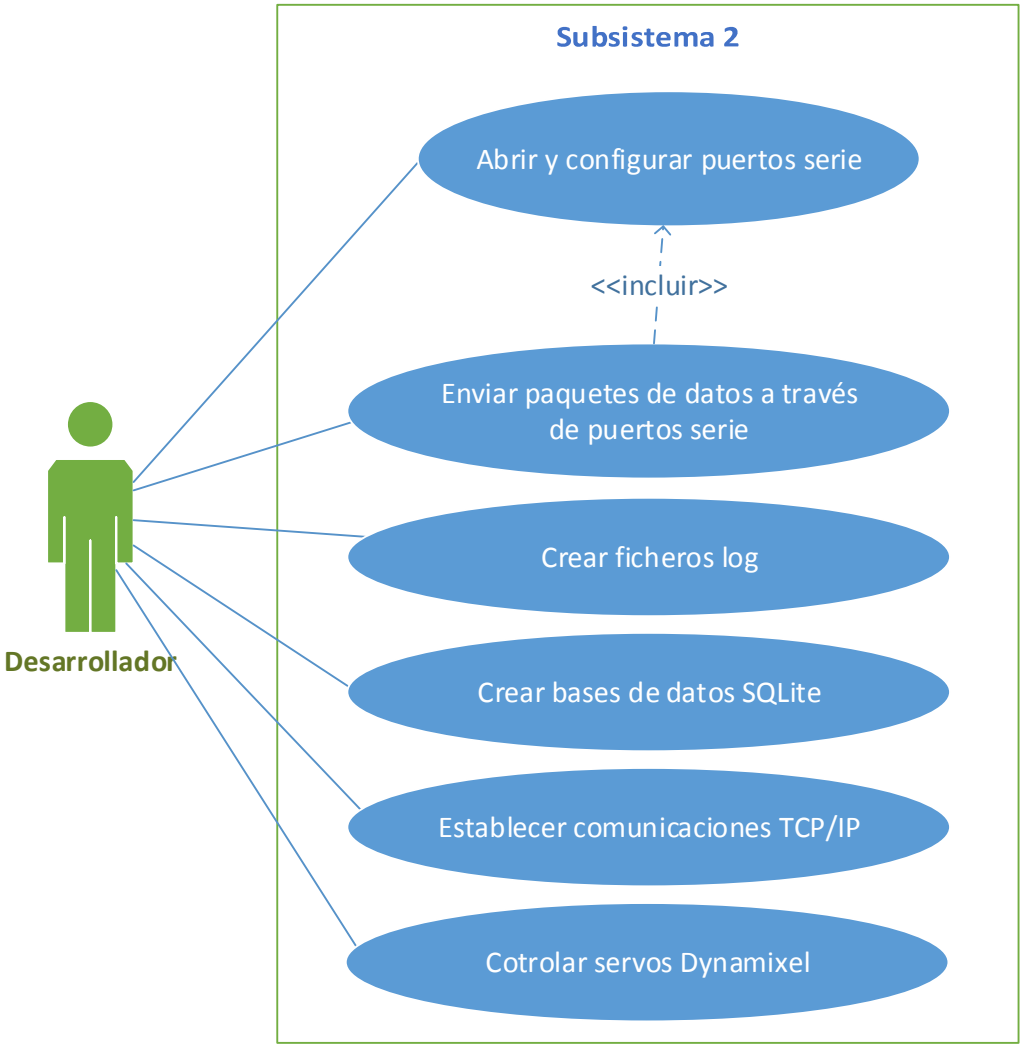


Ilustración 12: diagrama de casos de uso (subsistema 2)

Descripción textual de los casos de uso

A continuación se describen de forma detallada, los casos de uso representados en los diagramas de los dos subsistemas.

No es objetivo de este trabajo plantear un estudio del medio para ver qué componentes hardware, o qué arquitectura del sistema, podrían ser la mejor solución, puesto que este trabajo no aborda una solución concreta, sino que pretende dar una arquitectura de referencia que se pueda utilizar para muchos sistemas. Pero sí que es un objetivo el dotarla de las características que garanticen suficientemente su reutilización en distintas configuraciones. Por todo ello, los casos de uso formulados son abstractos. Se consideran abstractos porque no describen la realización de un objetivo concreto, sino un objetivo ejemplar. Como consecuencia, no todos los atributos de los casos de uso estarán descritos.

Los casos de uso van a constar de los siguientes atributos:

Identificador	Nombre
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando <evento de activación>.
Comentarios	

Tabla 4.2.00: Plantilla casos de uso

- **Identificador**: Se trata de un identificador único para cada caso de uso. Tendrá el prefijo *CU.Sx.* La "Sx" indicará el subsistema al que corresponde el caso de uso, por ejemplo, el caso de uso 3 del subsistema 1 tendría el siguiente identificador: *CU.S01-03.*
- **Nombre**: Titulo breve y descriptivo.
- **Descripción**: Especifica, en texto libre, el servicio que debe proporcionar el sistema a los usuarios.
- **Comentarios**: Algunos aspectos que se deben tomar en cuenta.

En esta plataforma de control, el único Actor que va existir es el *Desarrollador*, es por eso que, la plantilla de casos de uso, no incluye tal atributo.

Los casos de uso son los siguientes:

- Descripción textual de los casos de uso: subsistema 1

CU.S01-01	Habilitar puertos serie: USB y DB9
Descripción	El sistema deberá permitir al desarrollador cargar los módulos de los puertos serie. Una vez cargados, el sistema crea los dispositivos de entrada y salida.
Comentarios	Ninguno.

Tabla 4.2.01: CU.S01-01, Habilitar puertos serie: USB y DB9

CU.S01-02	Montar memorias USB, SD, micro-SD, y discos duros
Descripción	El sistema deberá permitir al desarrollador: 1- Cargar los módulos de estos dispositivos de almacenamiento. 2- Montarlos en el sistema de ficheros del sistema.
Comentarios	El sistema soportará todos los sistemas de ficheros (SF) que un sistema Linux incluye por defecto.

Tabla 4.2.02: CU.S01-02, Montar memorias USB, SD, micro-SD, y discos duros

CU.S01-03	Instalar software: aplicaciones y/o librerías
Descripción	El sistema deberá permitir al desarrollador la instalación de paquetes de software, desde la línea de comandos, invocando el gestor de paquetes del sistema. Una vez instalado, en un futuro se puede actualizar o eliminar.
Comentarios	Se empleará un gestor de paquetes de Linux.

Tabla 4.2.03: CU.S01-03, Instalar software: aplicaciones y/o librerías

CU.S01-04	Crear Sockets TCP/IP
Descripción	El sistema deberá permitir al desarrollador crear sockets TCP/IP. Una vez creados, se pueden emplear para establecer comunicaciones con otras máquinas.
Comentarios	Ninguno.

Tabla 4.2.04: CU.S01-04, Crear Sockets TCP/IP

CU.S01-05	Desarrollar software en C y C++
Descripción	El sistema deberá permitir al desarrollador crear aplicaciones empleando el lenguaje de programación C y C++. Una vez creadas las aplicaciones, el sistema podrá ejecutarlas.
Comentarios	Ninguno.

Tabla 4.2.05: CU.S01-05, Desarrollar software en C y C++

CU.S01-06	Desarrollar software con restricciones de tiempo-real
Descripción	El sistema deberá permitir al desarrollador crear aplicaciones con restricciones de tiempo-real empleando el lenguaje de programación C y C++. Una vez creadas las aplicaciones, el sistema podrá ejecutarlas.
Comentarios	Ninguno.

Tabla 4.2.06: CU.S01-06, Desarrollar software con restricciones de tiempo-real

- Descripción textual de los casos de uso: subsistema 2

CU.S02-01	Abrir y configurar puertos serie
Descripción	El sistema deberá permitir al desarrollador: 1- Abrir puertos serie. 2- Una vez abiertos, configurarlos en función de un conjunto de parámetros. Una vez finalizada la configuración de un puerto, podrá ser empleado para establecer comunicaciones con otros dispositivos.
Comentarios	El sistema debe ofrecer una entidad lógica que represente un puerto serie.

Tabla 4.2.07: CU.S02-01, Abrir y configurar puertos serie

CU.S02-02	Enviar paquetes de datos a través de puertos serie
Descripción	El sistema deberá permitir al desarrollador enviar y recibir paquetes de datos a través de puertos series abiertos y configurados. Una vez tratados los datos, el sistema devuelve el estado de la operación.
Comentarios	El sistema debe ofrecer una entidad lógica que represente un paquete de datos.

Tabla 4.2.08: CU.S02-02, Enviar paquetes de datos a través de puertos serie

CU.S02-03	Crear ficheros log
Descripción	El sistema deberá permitir al desarrollador crear ficheros log. Una vez creados, en tiempo de ejecución se podrá escribir en ellos todas las trazas oportunas.
Comentarios	El sistema debe ofrecer una entidad lógica que represente un fichero log.

Tabla 4.2.09: CU.S02-03, Crear ficheros log

CU.S02-04	Crear bases de datos SQLite
Descripción	El sistema deberá permitir al desarrollador crear bases de datos SQLite. Una vez creadas, se podrá ejecutar las sentencias SQL de consulta e inserción.
Comentarios	El sistema debe ofrecer una entidad lógica que represente una base de datos.

Tabla 4.2.10: CU.S02-04, Crear bases de datos SQLite

CU.S02-05	Establecer comunicaciones TCP/IP
Descripción	El sistema deberá permitir al desarrollador abrir puertos para establecer conexión TCP/IP. Una establecida la conexión, el sistema estará dispuesto comunicar con el otro extremo de la conexión.
Comentarios	El sistema debe ofrecer una entidad lógica que represente conexión TCP/IP.

Tabla 4.2.11: CU.S02-05, Establecer comunicaciones TCP/IP

CU.S02-06	Controlar servos Dynamixel
Descripción	El sistema deberá permitir al desarrollador enviar datos de control y consultar información de estado de los servos Dynamixel.
Comentarios	El sistema debe ofrecer una entidad lógica que represente un servo Dynamixel.

Tabla 4.2.12: CU.S02-06, Controlar servos Dynamixel

4.3.Requisitos Software

En esta sección se recogen los requisitos software que configuran la plataforma de control. Estos requisitos son el resultado del refinamiento de los requisitos de usuario y constan de los siguientes atributos:

Identificador	Nombre
Descripción	
Verificabilidad	Baja/Media/Alta
Dependencias RU	RU.C.Sx -xx / RU.R.Sx -xx

Tabla 4.3.00: Plantilla requisitos software

- **Identificador**: Se trata de un identificador único para cada requisito software. Tendrá el prefijo RS.Sx. La "Sx" indicará el subsistema al que corresponde el requisito, por ejemplo, el requisito 3 del subsistema 1 tendría el siguiente identificador: *RS.S01-03.*
- **Nombre**: Titulo breve y descriptivo.
- **Descripción**: Comentario conciso y preciso sobre la funcionalidad requerida.
- **Verificabilidad**: Establece la facilidad con la que se puede comprobar ese requisito (alta, media o baja).
- **Dependencias RU:** Indica que requisitos de usuario se corresponden con el requisito software.

Los requisitos de software son los siguientes:

Subsistema 1

RS.S01-01	Sistema controlador
Descripción	El sistema se basará en un sistema Linux, con un kernel 3.2
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-01, RU.R.S01-01

Tabla 4.3.01: RS.S01-01, Sistema controlador

RS.S01-02	Hardware de controlador x86
Descripción	El sistema soportará arquitecturas basadas en x86.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-02

Tabla 4.3.02: RS.S01-02, Hardware de controlador x86

RS.S01-03	Hardware de controlador ARM
Descripción	El sistema soportará arquitecturas basadas en ARM.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-02

Tabla 4.3.03: RS.S01-03, Hardware de controlador ARM

RS.S01-04	Soporte de varios tipos de puertos serie
Descripción	El sistema permitirá habilitar los puertos serie de conectores USB y DB9.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-03

Tabla 4.3.04: RS.S01-04, Soporte de varios tipos de puertos serie

RS.S01-05	Soporte de interfaces de red
Descripción	El sistema podrá realizar comunicaciones dentro de una red Intranet. Eso es poder establecer conexiones TCP/IP.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-04

Tabla 4.3.05: RS.S01-05, Soporte de interfaces de red

RS.S01-06	Soporte de almacenamiento
Descripción	El sistema soportará memorias Flash, por ejemplo: USB, SD, micro-SD.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-05

Tabla 4.3.06: RS.S01-06, Soporte de almacenamiento

RS.S01-07	Habilitar un entorno de desarrollo de tiempo-real
Descripción	El sistema tendrá integrado el framework Xenomai, para disponer de un entorno de desarrollo para tiempo-real.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-06, RU.R.S01-02

Tabla 4.3.07: RS.S01-07, Habilitar un entorno de desarrollo para tiempo-real

RS.S01-08	Crecimiento del sistema
Descripción	El sistema proporcionará un gestor de paquetes de software, para instalar, o eliminar, aplicaciones del sistema Linux.
Verificabilidad	Alta.
Dependencias RU	RU.C.S01-07

Tabla 4.3.08: RS.S01-08, Crecimiento del sistema

Subsistema 2

RS.S02-01	Comunicación simultánea con más de un puerto serie
Descripción	El framework podrá enviar y recibir paquetes por puertos serie abiertos. En un momento dado, pueden existir varios puertos abiertos. Cada puerto abierto tendrá un índice único. Esos puertos tienen que estar asociados con varios dispositivos de entrada y salida.
Verificabilidad	Alta.
Dependencias RU	RU.C.S02-01

Tabla 4.3.05: RS.S02-01, Comunicación simultánea con más de un puerto serie

RS.S02-02	Configuración variable de los puertos serie
Descripción	El sistema permitirá la configuración de los puertos serie en función de los siguientes parámetros: ✓ Nombre puerto. ✓ Baudios. ✓ Bits de parada. ✓ Tamaño de byte. ✓ Paridad. ✓ Control de flujo. Cuando alguno, o varios, de estos parámetros no sean proporcionados, se tomarán sus valores por defecto, que deberán ser establecidos y documentados.
Verificabilidad	Alta.
Dependencias RU	RU.C.S02-02

Tabla 4.3.06: RS.S02-02, Configuración variable de los puertos serie

RS.S02-03	Gestión de paquetes de datos
Descripción	La gestión de paquetes se va a realizar por los siguientes criterios: ✓ **Prioridad**: los paquetes de alta prioridad se tratan en primer lugar. ✓ **Tiempo**: Los actuadores en actividad, no deben recibir paquetes de movimiento, hasta pararse. ✓ **Destino**: Paquetes con destinos distintos se pueden enviar sin esperas, es decir, el tiempo de espera de un actuador no debe influir en otro actuador.
Verificabilidad	Media.
Dependencias RU	RU.C.S02-03

Tabla 4.3.08: RS.S02-03, Gestión de paquetes

RS.S02-04	Control remoto
Descripción	Se permitirá la comunicación entre el controlador y dispositivos remotos, a través de internet utilizando el protocolo TCP/IP.
Verificabilidad	Baja.
Dependencias RU	RU.C.S02-04

Tabla 4.3.09: RS.S02-04, Control remoto

RS.S02-05	Simulador
Descripción	Se va a ofrecer una interfaz para conectar el controlador con simuladores, que deben adaptarse a la interfaz ofrecida.
Verificabilidad	Baja.
Dependencias RU	RU.C.S02-05

Tabla 4.3.10: RS.S02-05, Simulador

RS.S02-06	Ficheros log
Descripción	Se permitirá la creación y la escritura en ficheros log.
Verificabilidad	Alta.
Dependencias RU	RU.C.S02-06

Tabla 4.3.11: RS.S02-06, Ficheros log

RS.S02-07	Bases de datos embebidas
Descripción	Con la librería SQLite se permitirá crear tablas de datos y consultarlas.
Verificabilidad	Alta.
Dependencias RU	RU.C.S02-07

Tabla 4.3.12: RS.S02-07, Bases de datos embebidas

RS.S02-08	Servos Dynamixel
Descripción	El sistema ofrecerá una implementación de la interfaz de comandos de los servos Dynamixel “Ax-12+”.
Verificabilidad	Alta.
Dependencias RU	RU.C.S02-08

Tabla 4.3.13: RS.S02-08, Servos Dynamixel

RS.S02-09	Codificación del framework en lenguaje C++
Descripción	El framework se desarrollará en el lenguaje C++.
Verificabilidad	Alta.
Dependencias RU	RU.R.S02-01

Tabla 4.3.14: RS.S02-09, Codificación del framework en lenguaje C++

4.4.Matriz de trazabilidad

A continuación se muestra la matriz de trazabilidad que relaciona los requisitos de usuario con los requisitos software:

Subsistema 1

	RS.S01-01	RS.S01-02	RS.S01-03	RS.S01-04	RS.S01-05	RS.S01-06	RS.S01-07	RS.S01-08
RU.C.S01-01	X							
RU.C.S01-02		X	X					
RU.C.S01-03				X				
RU.C.S01-04					X			
RU.C.S01-05						X		
RU.C.S01-06							X	
RU.C.S01-07								X
RU.R.S01-01	X							
RU.R.S01-02							X	

Tabla 4.4.01: Matriz de trazabilidad (subsistema 1)

Examinada la tabla de trazabilidad, se comprueba que cada requisito de usuario produce al menos un requisito de software.

Subsistema 2

	RS.S02-01	RS.S02-02	RS.S02-03	RS.S02-04	RS.S02-05	RS.S02-06	RS.S02-07	RS.S02-08	RS.S02-09
RU.C.S02-01	X								
RU.C.S02-02		X							
RU.C.S02-03			X						
RU.C.S02-04				X					
RU.C.S02-05					X				
RU.C.S02-06						X			
RU.C.S02-07							X		
RU.C.S02-08								X	
RU.R.S02-01									X

Tabla 4.4.02: Matriz de trazabilidad (subsistema 2)

Examinada la tabla de trazabilidad, se comprueba que cada requisito de usuario produce al menos un requisito de software.

5. Diseño de la Plataforma de Control

En este capítulo se especifica la arquitectura del sistema y el entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes a utilizar y la interacción entre ellos. Este diseño servirá de guía para la fase de implementación.

Todos los componentes especificados en la fase del diseño servirán como modelo, y en ningún momento restringen la utilización de otros componentes. La adecuada elección de componentes queda exclusivamente bajo los criterios del usuario de la plataforma de control.

5.1.Diseño arquitectónico

En el capítulo 3, se ha definido, desde una perspectiva superficial, la arquitectura de la plataforma de control. En esta sección vamos a describir la arquitectura general de la plataforma de control, definir sus componentes y su jerarquía.

En la siguiente ilustración, en el diagrama de despliegue, vemos la naturaleza de los componentes de la arquitectura y su interconexión.

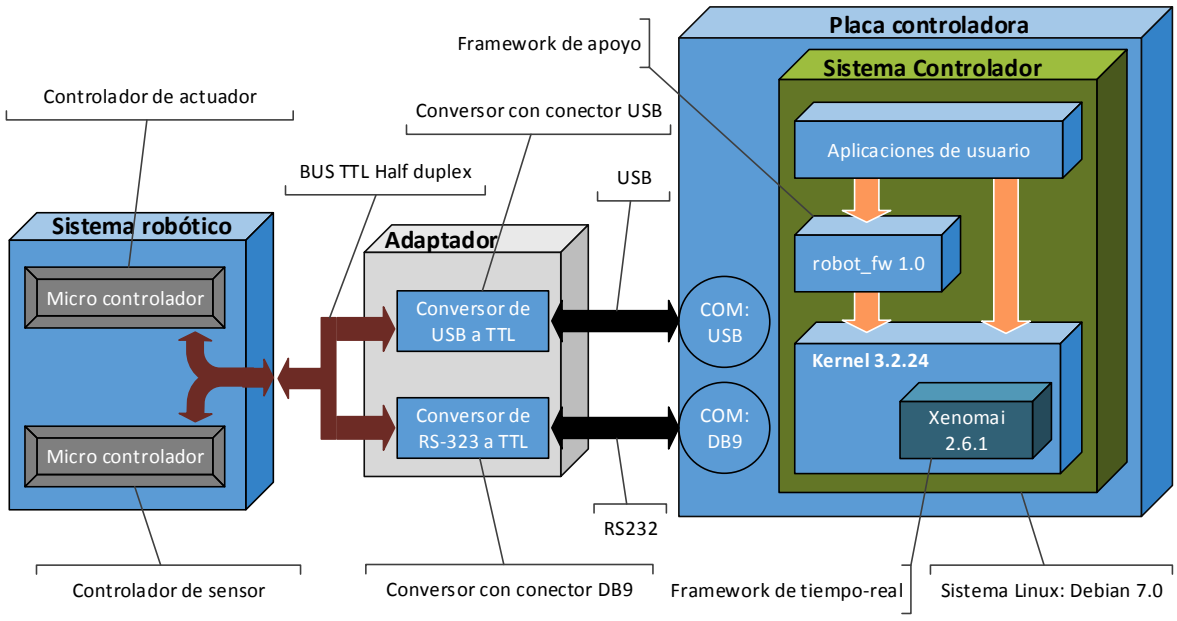


Ilustración 13: diagrama de despliegue

En las siguientes secciones se van a ver en detalle las características de cada componente.

5.2.Diseño Hardware

En este subcapítulo vamos a describir las diferentes alternativas de componentes hardware y el diseño de las conexiones. Otra vez más, los componentes aquí expuestos, servirán para realizar la implementación de la plataforma, y también, como modelo de referencia de componentes compatibles.

Placa controladora

Este componente es esencial, ya que constituye el entorno hardware de ejecución del firmware del controlador. También proporciona los puertos físicos (cableados o inalámbricos) para comunicar con otros sistemas: el sistema robótico y el del control remoto.

Para elegir una placa conforme a las capacidades y restricción de la plataforma, se ha acotado la búsqueda por las siguientes características:

- **Prestaciones:** Capacidad de procesamiento del microprocesador, puertos de entrada y salida, consumo de energía y organización arquitectónica.
- **Coste**: El precio es importante, ya que se pretende abaratar los costes de la plataforma.
- **Tamaño**: Dada la naturaleza de los sistemas robóticos, el tamaño y el peso son decisivos.
- **Posesión del laboratorio**: Si el laboratorio ya dispone de ello, sería un coste de menos.

Teniendo en cuenta todo aquello, se identifican dos placas:

1. **EBOX-3310MX-AP**[7] basada en un **Vortex86MX**, de 32bit **x86** CPU, 933MHz con 256MB DRAM, fabricante DMP Electronics[8].

Ilustración 14: Placa EBOX-3310MX-AP

2. **RoBoard RB-110**[9] basada en un **Vortex86DX**, de 32bit **x86** CPU, 1000MHz con 256MB DRAM, fabricante DMP Electronics.

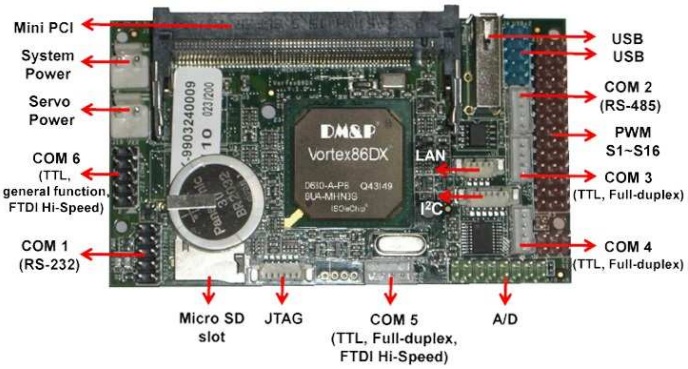


Ilustración 15: Placa RoBoard RB-110

Las dos placas están basadas en un sistema en chip Vortex86 DX/MX[10], SoC (System on Chip). En la siguiente ilustración, se muestra la arquitectura del SoC (La organización de sus componentes hardware: Buses, memoria, CPU, Puertos de entrada y salida, …etc.):

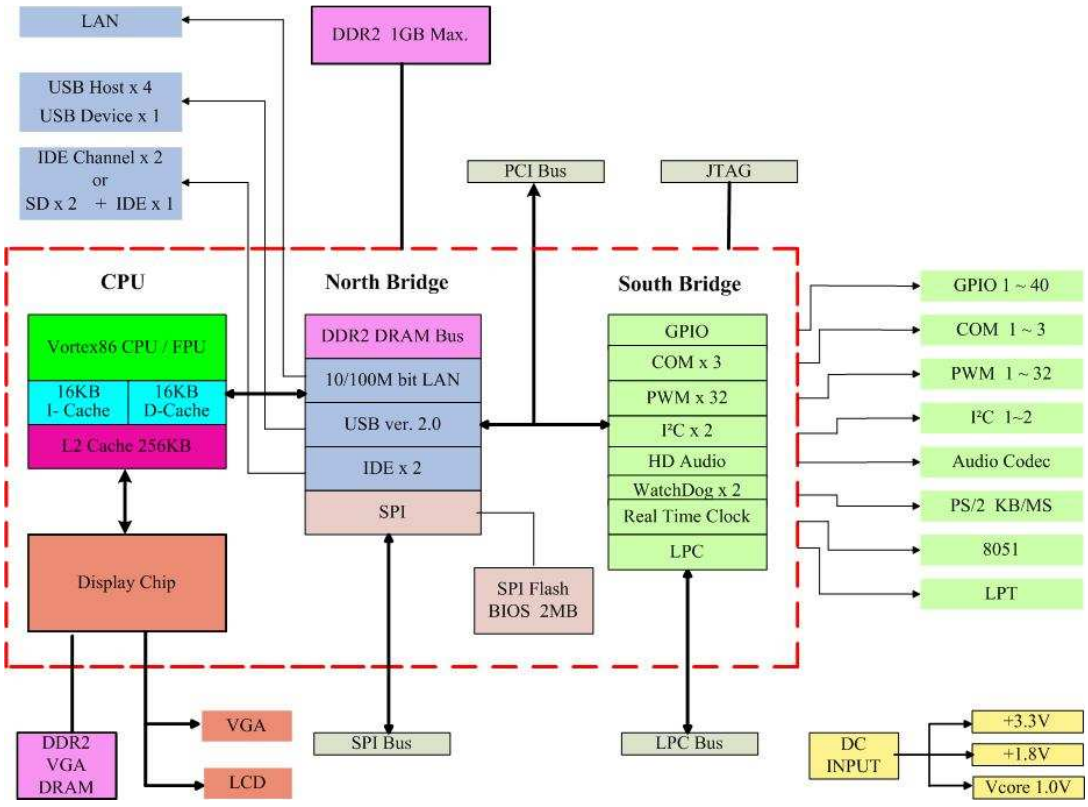


Ilustración 16: arquitectura de vortex86MX-BD

Para el desarrollo de la plataforma decantamos por la Ebox, por comodidad y por su resistencia, porque, a diferencia de la Roboard, lleva una carcasa. Pero a la hora de integrar el controlador en el sistema robótico, la Roboard lleva la ventaja por su diminuto tamaño.

Actuador

Es el componente que ejecutara las órdenes recibidas a través de su circuito de control, ya sea realizar un movimiento o enviar a la placa datos de su estado (tensión, voltaje, id, posición actual...).

El laboratorio de robótica de la UC3M cuenta con dos tipos de robots:

1- **Bioloid**: Está constituido por actuadores (servos) Dynamixel AX-12.

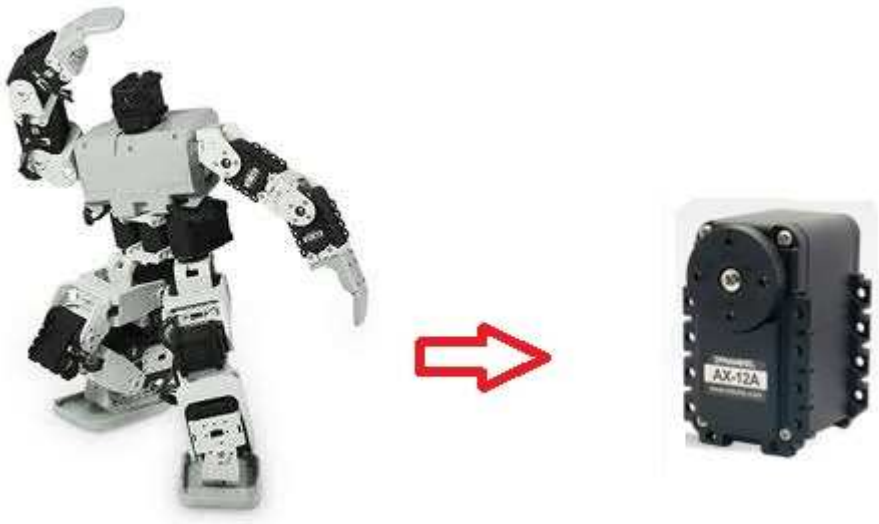


Ilustración 17: Bioloid

2- **Robonova**: Constituido por actuadores Hitec HSR-8498 HR.

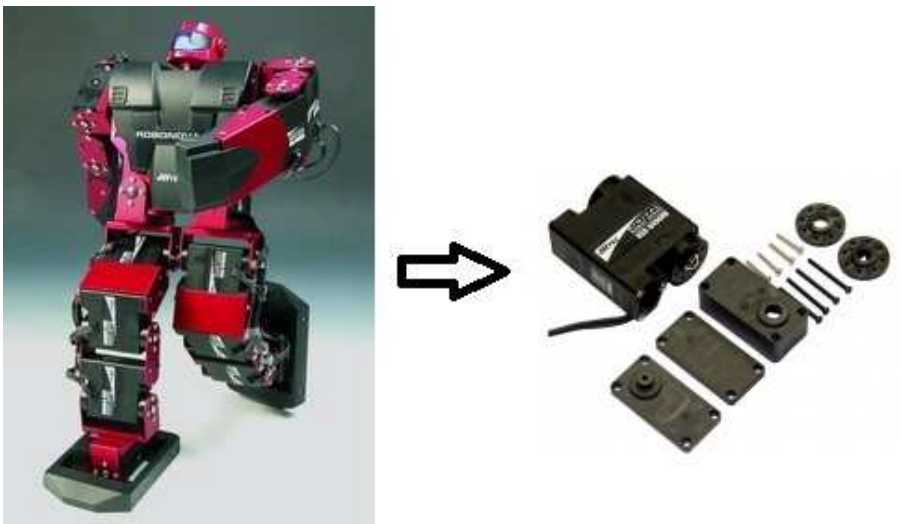


Ilustración 18: Robonova

Actualmente el Laboratorio trabaja con el robot Bioloid, mientras que el Robonova está obsoleto, dedicado para el uso académico.

Las diferencias más importantes entre los dos tipos de actuadores son:

	Dynamixel	HSR-8498
Bus de datos	TTL half-dúplex	HMI half-dúplex
Precio	52€	43€
Tipo conexión	En serie	En paralelo
Interfaz de comandos	Flexible	Estática

Tabla 5.2.1: Diferencias Dynamixel y HSR-8498

En cuanto a la estructura de instrucciones de control y a la conexión al bus de datos, el servo Dynamixel es relativamente más práctico de manejar que al HSR-8498. Una de las ventajas que tiene es la posibilidad de conectar varios servos en cadena. Eso es de vital importancia, ya que supone una disminución del número de cables del sistema robótico, y permite el envío de una única instrucción, a la vez, a un colectivo de actuadores.

En la siguiente ilustración, se muestra la manera conectar los servos Dynamixel al bus de datos, en efecto, al controlador:

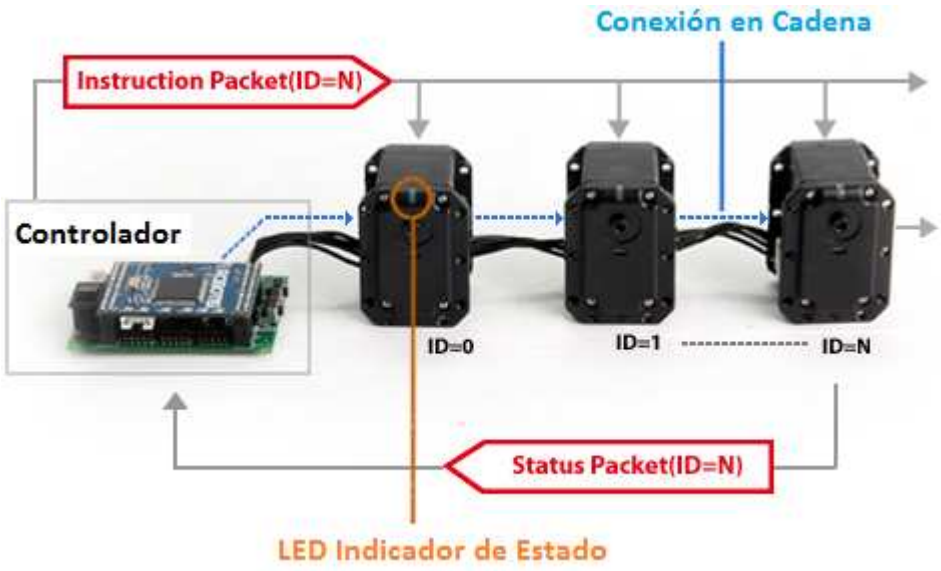


Ilustración 19: BUS Dynamixel

Dentro del la cadena de servos, cada tiene un ID único, y cada instrucción tiene asociado un ID del servo, o servos, de destino.

Con más detalle, en siguiente ilustración, se muestra el esquema de conexión de los pines de un servo Dynamixel. Además, en el circuito se la conexión adecuada de la batería y la salida del adaptador.

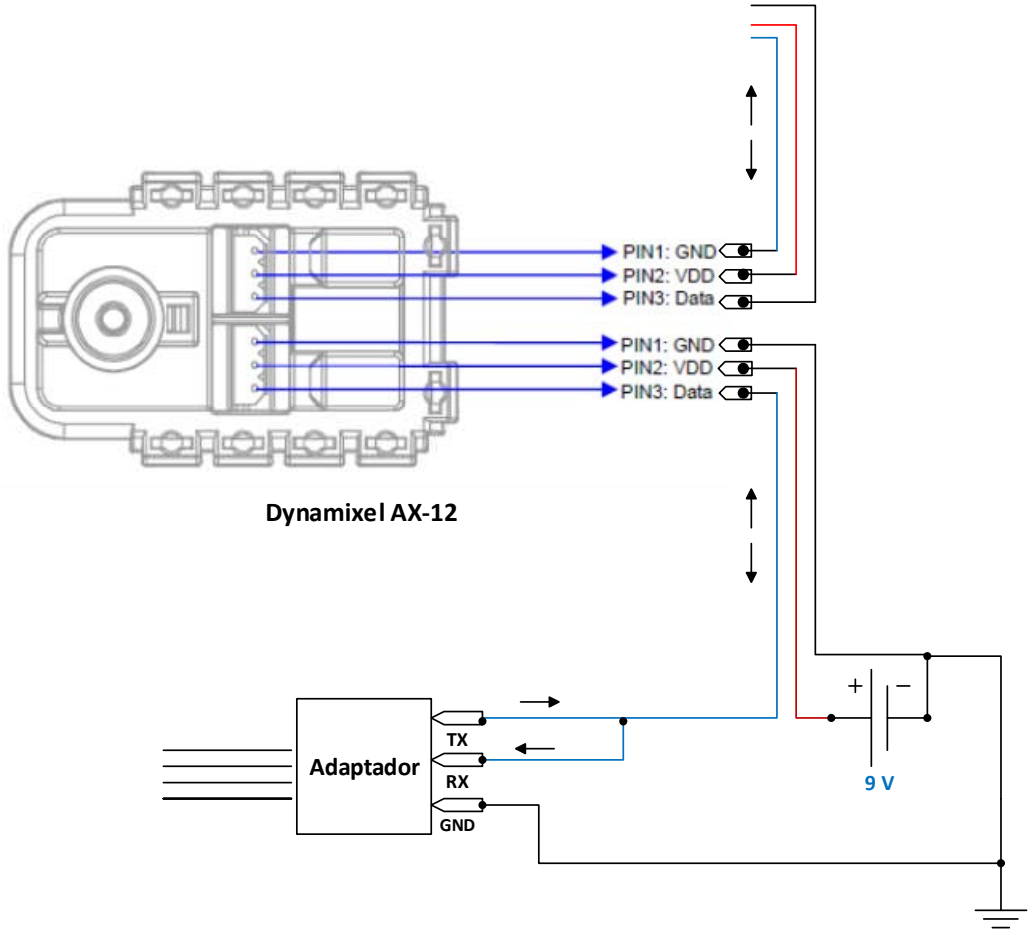


Ilustración 20: Circuito de conexión del servo Dynamixel

Sensor

Es el componente que proporciona datos sensoriales del entorno, a través de su circuito de control (temperatura, proximidad, movimientos, velocidad, centro de gravedad,…)

El fabricante de los servos Dynamixel (ROBOTIS), proporciona otro tipo de servo (AX-S1) que integra sensores IR (Infrarrojos) y de sonido (juegan un papel de ojos y oídos):

Ilustración 21: Dynamixel AX-S1

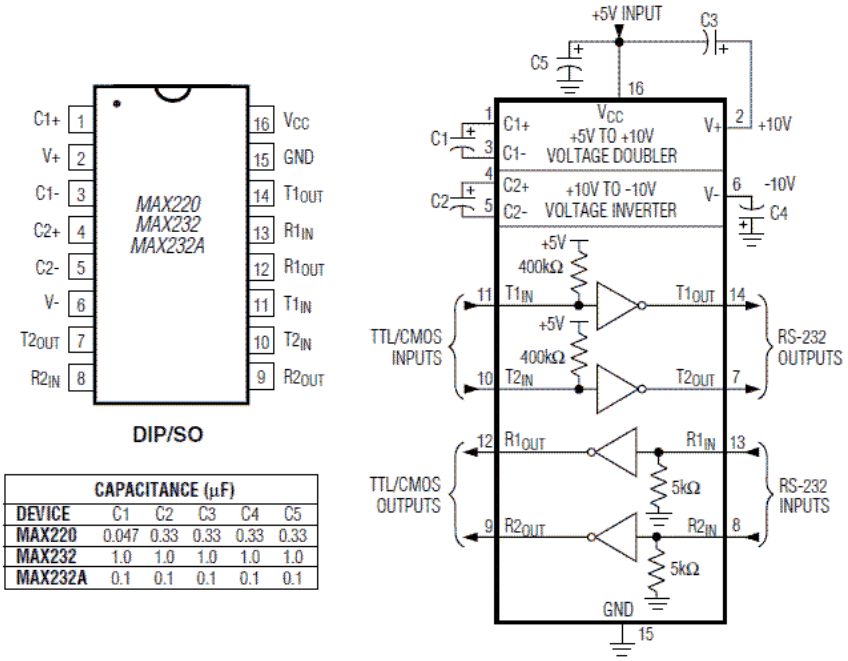
El servo, AX-S1 (Integrated Sensor), tiene la misma interfaz de conexión que al servo AX-12, y el mismo mecanismo de instrucciones.

Para que un sensor esté suportado por la plataforma de control, debe estar conectado a un microcontrolador, y la conexión con el bus de datos se realiza a través de éste último. Los servos Dynamixel AX-12 y AX-S1 llevan un microcontrolador (Atmel de 8bits de palabra) integrado por defecto.

Adaptador

El adaptador RS-232 a TTL está situado entre el controlador y el sistema robótico, y es el encargado de adaptar y convertir las señales eléctricas de tipo RS-232 del controlador a señales TTL que gestiona el microcontrolador del servo, o el sensor.

Existe una gran variedad se convertidores RS-232 a TTL, habitualmente basados en el circuito integrado MAX-232[11], o sus derivados.



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Ilustración 22: MAX-232, interface de conexión

En la siguiente ilustración, se muestra el circuito final del adaptador (de RS-232 a TTL y viceversa):

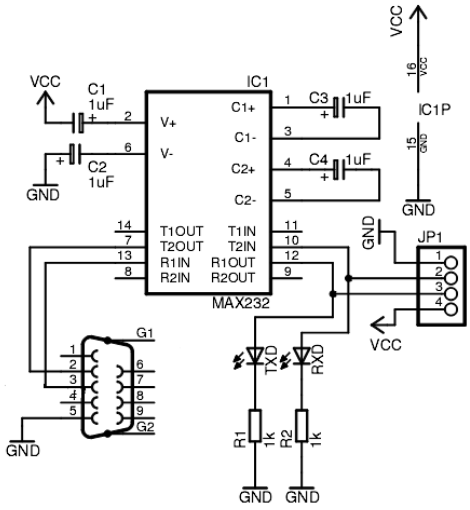


Ilustración 23: Circuito adaptador

Es muy importante tener en cuenta que, el adaptador va a depender del sistema robótico y la placa controladora, en concreto, dependerá de las señales manejadas por esos dos sistemas.

5.3.Diseño Software

En este apartado se va a detallar la elección del sistema operativo para instalar en el hardware del controlador, que constituiría el entorno software de producción y

ejecución. También se verán características relevantes del Framework de tiempo-real que se va a integrar en núcleo de Linux.

Sistema Linux

Es un requisito fundamental de que, el sistema sea de software libre. Para seleccionar una distribución de Linux, se tuvieron en cuenta dos aspectos decisivos: las características técnicas de la placa controladora, y en segundo lugar, el coste de adaptar la distribución de Linux a las necesidades de la plataforma de control.

Han sido seleccionadas tres distribuciones como alternativas: LFS[12], Ubuntu[13] y Debian[14]. Los paquetes de las dos distribuciones son compatibles entre sí, ya que, en su comienzo, Ubuntu fue una bifurcación del proyecto Debian.

LFS: Como primera elección, se ha optado por crear un distribución personalizad desde cero. Eso es, seleccionar el software básico y necesario para la distribución, descargarlo y compilarlo. Se puede descargar el código fuente de todos los componentes del sistema, de sitios oficiales como Debian, Ubuntu o sitios personales de desarrolladores, sujeto a la licencia GNU[15]. Por último, se compila la versión del Kernel elegida. El Kernel va independiente de las distribuciones, de hecho, todas ellas trabajan sobre las mismas fuentes del Kernel, tan sólo pueden personalizar el proceso de compilación del Kernel. Se ha logra crear la distribución, pero, no resultó compatible con la placa controlador especificada. Por lo que, hay que centrarse en la otras alternativas.

Ubuntu: El instalador de ésta, instala por defecto paquetes importantes, recomendados y opcionales, además del entorno gráfico. Para poder personalizar la instalación, Ubuntu ofrece un CD de instalación alternativo[16]. La instalación por defecto de Ubuntu no es viable, por el uso excesivo de recursos (el entrono gráfico requiere al menos 300 MB de RAM). Se ha probado la instalación alternativa 3 veces, pero ninguna ha tenido éxito.

Debian: Suporta más arquitecturas que Ubuntu (ARM, MIPS), mantenido por desarrolladores voluntarios, cuenta con una comunidad importante de desarrolladores y dispone de mucha documentación. El instalador es bastante práctico, permite una instalación personalizada.

Se ha seleccionado Debian como sistema base de la plataforma de control, por comodidad. Porque permite una instalación básica (instalación mínima de software, sin entornos gráficos), idónea para este sistema; la plataforma de control.

Para que el sistema Linux cumpla con todos los requisitos definidos, se necesita crear un nuevo Kernel personalizado. Se ha seleccionado la versión 3.2.24 del Kernel por dos

razones: porque es la más reciente, y compatible con Xenomai (por cuestiones de *patches* de *adeos)*.

Tiempo-real

Para cubrir las necesidades de tiempo-real, se ha seleccionado el Framework de tiempo-real Xenomai.

Xenomai sigue un modelo de código fuente divido, la disociación del espacio de soporte del núcleo de las bibliotecas de espacio de usuario, utilizadas para acceder al primero. Proporciona un subsistema de tiempo real perfectamente integrado en Linux, por lo que, se construye como parte del Kernel seleccionado. Soporta las arquitecturas X86_32/64, ARM, PowerPC, NIOS II, Blackfin.

Para efectuar la instalación de Xenomai, se necesita tanto el código fuente del Kernel seleccionado (en nuestro caso la versión 3.2.24) como el de Xenomai (versión 2.6.1). Ambos códigos fuente son disponible en las respectivas páginas web: Kernel-3.2.24[17] y Xenomai-2.6.1[18]. Los componentes Xenomai núcleo y espacio de usuario son, respectivamente, disponibles en las dos subcarpetas *ksrc/* y *src/* en árbol de carpetas de Xenomai.

Bases de datos embebidas

Para las bases de datos ligeras, se va a integrar la librería SQLite3[19] en el sistema. Es el motor de bases de datos con más despliegue en las bases de datos embebidas, por ejemplo, en Android y iOS, SQLite es el motor de bases de datos por defecto (Como desarrollados en esas plataformas móviles, lo he integrado en varios proyectos).

Tiene dos grandes ventajas: es ligero, y emplea la sintaxis SQL (muy extendida).

5.4.Diseño del Framework de apoyo: robot_fw

En este apartado se detalla el diseño software del Framework de apoyo. El robot_fw define un conjunto de entidades abstractas (Interfaces), con los que trabaja internamente, para ofrecer una interfaz común y estable para el usuario de la plataforma. En la siguiente ilustración se muestran las Interfaces proporcionadas:

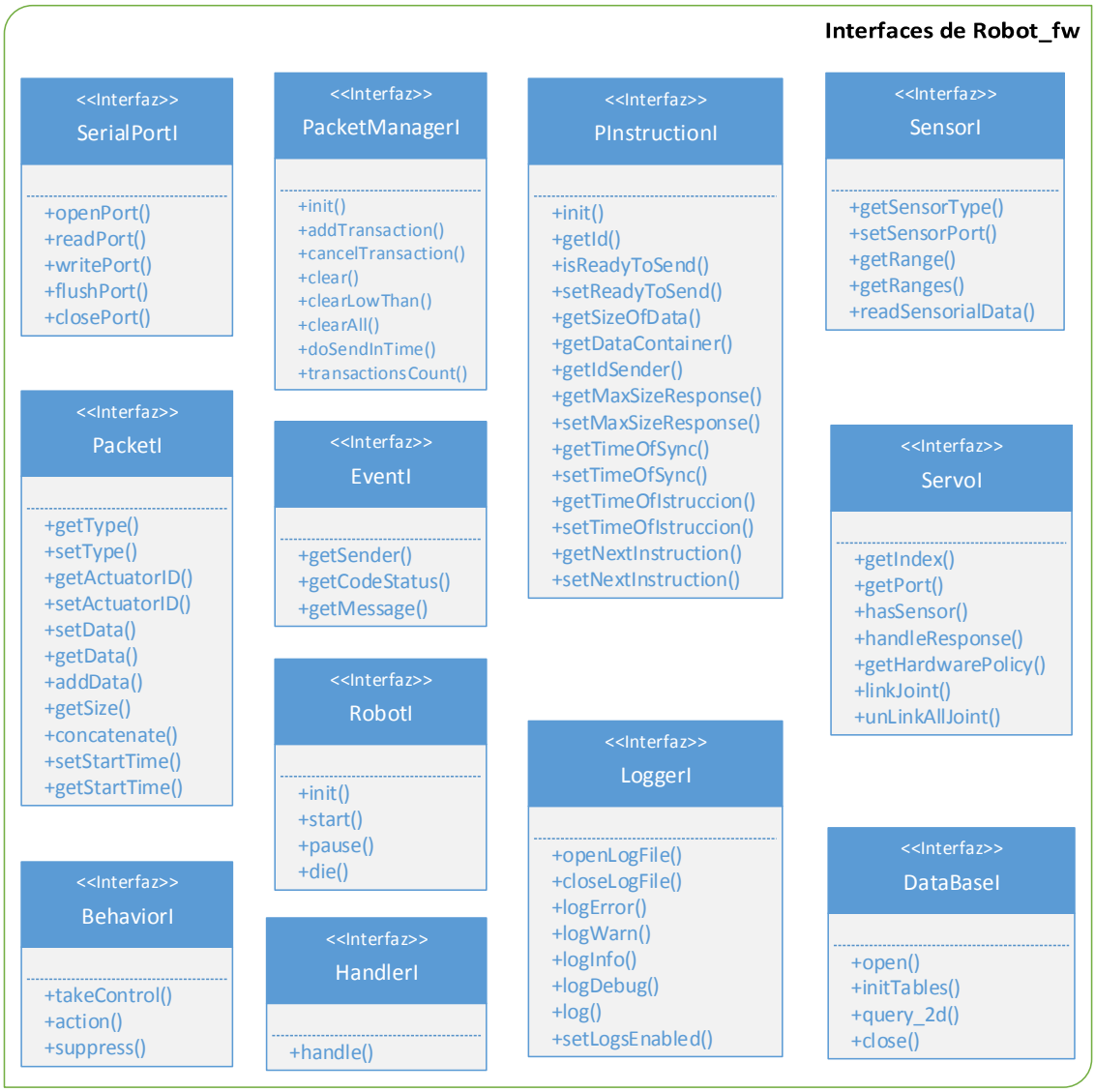


Ilustración 24: Interfaces de robot_fw

El robot_fw también ofrecerá las siguientes clases (las clases que implementan las interfaces, ofrecidas por el robot_fw, están excluidas, ya que tendrán la misma interfaz que la que implementan):

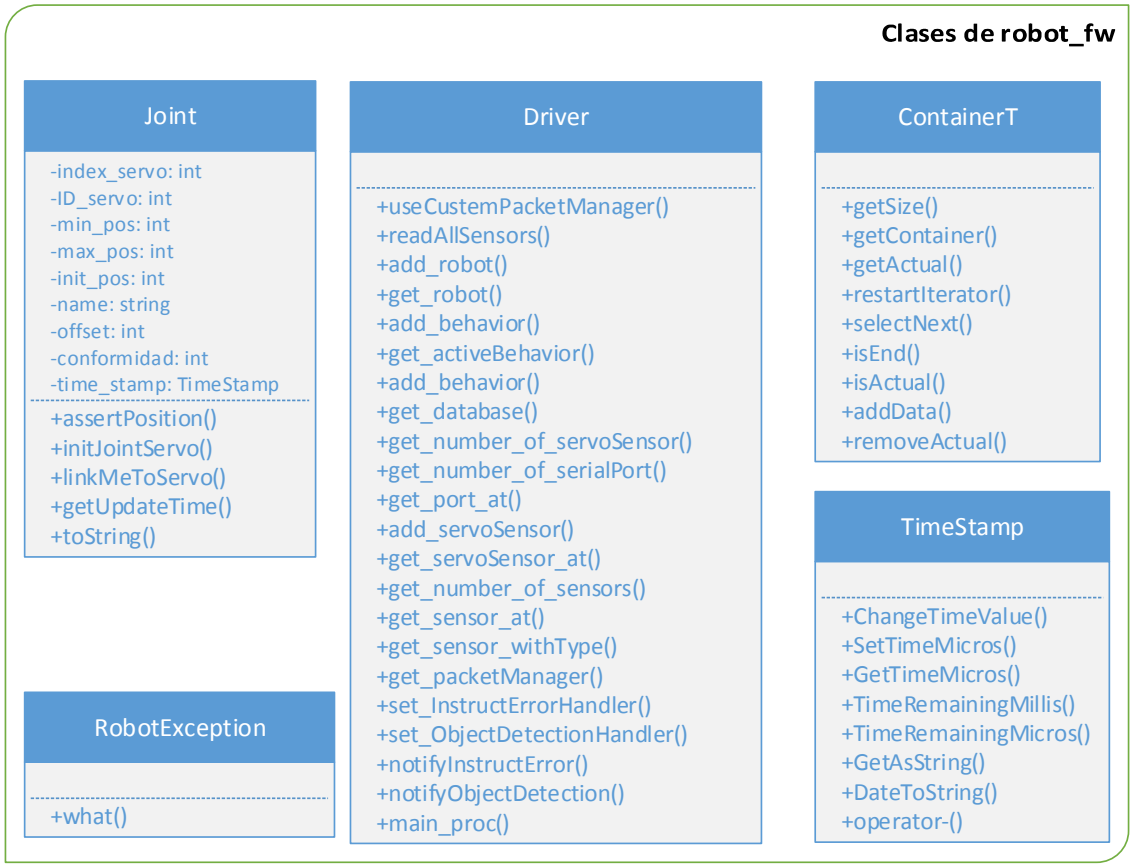


Ilustración 25: Clases de robot_fw

Expuestas las entidades de robot_fw, en la siguiente ilustración, se muestra el diagrama de clases, con las relaciones entre las distintas entidades del Framework:

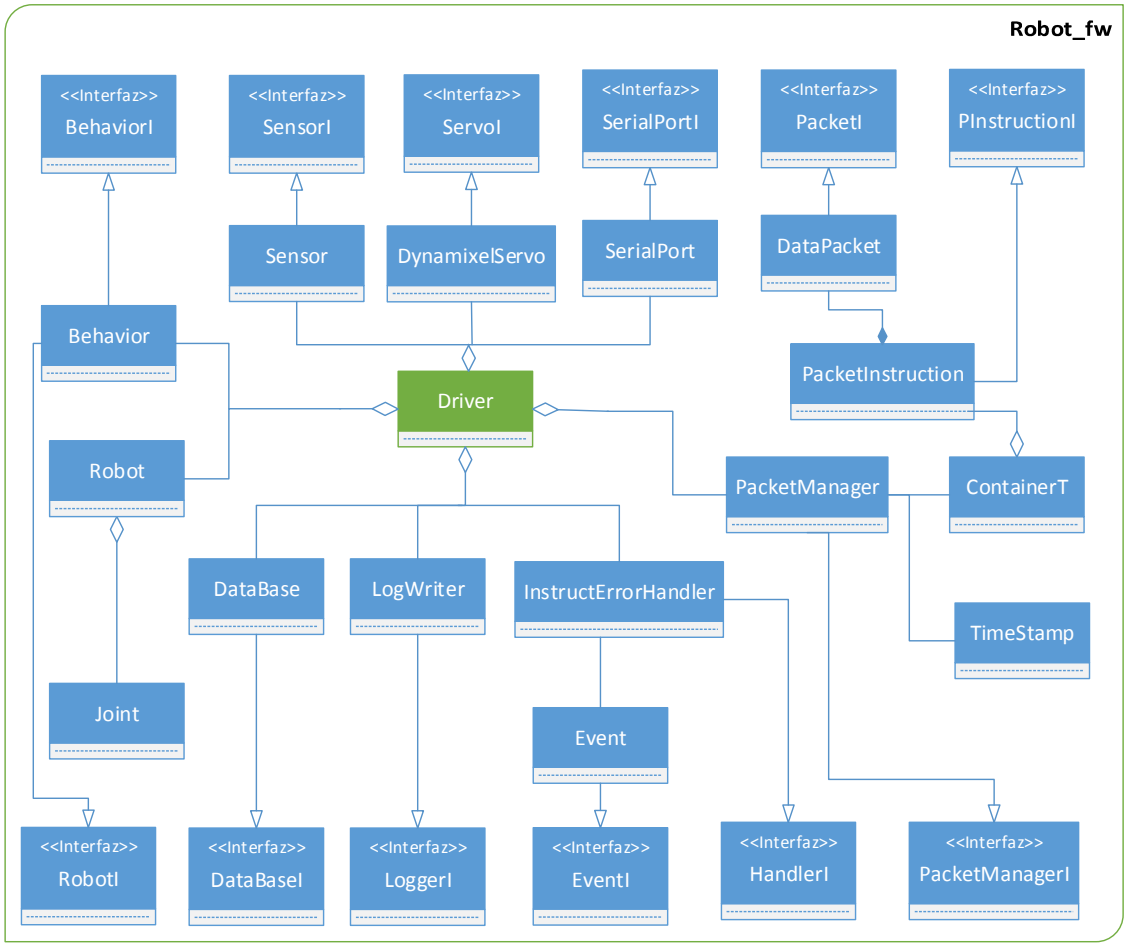


Ilustración 26: Diagrama de clases

Entidades de robot_fw

En esta sección, se describen con más detalle las clases más relevantes del robot_fw:

- Clase **SerialPort**: ofrece un soporte para abrir puertos serie con una variedad de configuraciones (baudios, paridad, bits de parada,...).

- Clase **DynamixelServo**: en este framework se ofrece una implementación para los servos Dynamixel AX-12+. Aunque, el usuario puede implementar la interfaz para otros tipos de servos. La interfaz no define las operaciones de las que, un servo debe ofrecer, sino que, define aspectos necesarios para que el servo pueda ser gestionado por el Framework.

- Clase **PacketInstruction**: un contenedor de un comando para un servo, además de metadatos para la identificación y gestión del paquete.

- Clase **PacketManager**: es la puerta de comunicación con los actuadores y sensores. La unidad de trabajo del gestor de paquetes es la transacción, es una asociación de objetos **PacketInstruction**, con los que se podría producir un

gesto o varios del robot (la transacción es una instancia de la plantilla **ContainerT**). El **PacketManager** envía las instrucciones (teniendo en cuenta los metadatos) y recibe las respuestas. Puede tratar las transacciones en modo secuencial, de forma que, hasta que se agoten las instrucciones de la transacción actual, no tratar la siguiente. Y el otro modo es, enviar todas las instrucciones posible en el momento actual de la transacción actual, y luego tratar la siguiente. En esta clase se definen 4 colas de prioridad, ya que hay instrucciones más prioritarias que otras.

- Clase **Behavior**: una secuencia de gestos que instruyen al robot a actuar de una forma u otra. Ejemplos de comportamientos pueden ser caminar, girar... etc. El usuario final es el que define la acción (el verbo) de un comportamiento.

- Clase **Joint**: un Joint (articulación) tiene una asociación univoca con un servo con un ID físico concreto. Un Joint puede ser rodilla, codo,... etc. Otorgan al robot la capacidad de mover. Desde el punto de vista del programador, son objetos que guardan el estado de un servo con un ID físico concreto. Simplemente, los Joints son una forma para facilitar el control sobre las distintas partes del robot.

- Clase **Robot**: puede ser cualquier estructura formada por un servo o varios. El usuario final es el que da forma al robot, y por eso, tiene que definir la acción de de los comportamientos.

- Clase **Driver**: Inicializa el entorno antes de pasar el control al robot. La interacción del robot con el Framework se realiza mediante el **Driver**. Constituye el punto de entrada del Framework. Incluye la función main_proc, que se encarga de comenzar la ejecución del sistema de control, de un modo consistente.

- Clase **LogWriter**: genera un fichero para escribir los logs (diferentes sucesos de ejecución errores, warnings,...), ya que durante la ejecución no se dispone de un monitor para verlos en vivo. Y también servirá como método de depuración.

6. Implantación y Pruebas

En este capítulo vamos a presentar los resultados de las pruebas efectuadas. Paso siguiente, se presenta la implantación final del sistema.

6.1.Pruebas

En esta sección se recogen las pruebas realizadas para comprobar el comportamiento del sistema, y verificar que se cumplan los requisitos de capacidad (funcionales) extraídos en la fase de análisis. Las constan de los siguientes atributos:

Identificador	
Descripción	
Procedimiento	
Resultado	
Estado	

Tabla 6.1.00: Plantilla de pruebas

Identificador: Se trata de un identificador único para cada prueba realizada. Tendrá la siguiente estructura “PR.xx”, la “xx” corresponde al número de prueba.
Objetivo: el objetivo de la prueba.
Procedimiento: Pasos a seguir para realizar la prueba.
Resultado: Muestra el resultado obtenido al realizar la prueba.
Aceptada: Muestra si la prueba ha sido realizada con éxito.

Para realizar las pruebas se empleado los siguientes componentes:

- Placa controladora: EBOX-3310MX-AP (ver el apartado ?5.2).
- Servos Dynamixel AX-12.
- Sistema Linux: Distribución Debian 7.0 con Kernel 3.2.24.
- Placa Arduino UNO (con microcotrolador ATMEL).

Las pruebas realizadas para son las siguientes:

Identificador	PR.01
Objetivo	Verificar que el sistema soporta el Hardware con las características especificadas.
Procedimiento	Con el disco creado con la máquina virtual, vamos a proceder emplearlo en la placa controladora. Para ello, se inserta el disco (memoria USB) del sistema en la placa EBox. Se conectan también un teclado y una pantalla a la placa. Y finalmente se Inicia la Ebox.
Resultado	El sistema inicia sin fallos, lo que indica que el Hardware está soportado.
Aceptado	Sí

Tabla 6.1.01: Prueba 1

Identificador	PR.02
Objetivo	Verificar que el sistema habilita los puertos serie.
Procedimiento	Una vez iniciado el sistema, desde una terminal, se carga el driver de los puertos serie (con el comando *modprobe 8250_pnp*). Con el comando ls si el sistema habilitó los dispositivos ttyS<n> para los puertos serie: ls /dev/ttyS*
Resultado	Se ve en pantalla: `/dev/ttyS0`, `/dev/ttyS1`, `/dev/ttyS2` y `/dev/ttyS3`
Aceptado	Sí

Tabla 6.1.02: Prueba 2

Identificador	PR.03
Objetivo	Verificar que el sistema puede conectarse a internet, y por lo tanto establecer conexiones TCP/IP.
Procedimiento	Primero hay que modificar el fichero `/etc/networks/interfaces` para asignar a la tarjeta de red la ip, puerta de enlace y DNS adecuados. Reiniciar la interfaz de red para que los nuevos parámetros tengan efecto (`ifconfig ethRoboard down` y luego `ifconfig ethRoboard up`). Final mente se prueba actualizar los paquetes de sistema, conectándose al servidor de Debian.
Resultado	Se ve en pantalla que el software se descarga se instala correctamente.
Aceptado	Sí

Tabla 6.1.03: Prueba 3

Identificador	PR.04
Objetivo	Verificar que Xenomai se integra bien con el Kernel.
Procedimiento	Una vez iniciado el sistema, desde una terminal, examinamos los logs del Kernel (`dmesg | grep Xeno`).
Resultado	Se ve en pantalla se ven los logs de Xenomai, que indican que ha sido cargado fallos.
Aceptado	Sí

Tabla 6.1.04: Prueba 4

Identificador	PR.05
Objetivo	Verificar que Framework de apoyo (robot_fw) aplica correctamente los parámetros de configuración de los puertos serie, y que el manejo de varios puertos serie.
Procedimiento	Para esta prueba se emplea un adaptador RS-232/TTL y una placa Arduino UNO. El adaptador se conecta al puerto DB9, y el Arduino al puerto USB. Mediante una Protoboard, se interconectan el adaptador y el Arduino, de modo que, lo que se envía por un puerto se redirige al otro, eso es, se establece un circuito cerrado de comunicación. Se crea un programa, en el cual se crean dos objetos de la clase SerialPort, un objeto por puerto. En varias iteraciones, se emplea un objetos para enviar una cadena de texto y el otro para recibir, y viceversa.
Resultado	Se ve en pantalla se ve que, los que se envía por un puerto se le por el segundo.
Aceptado	Sí

Tabla 6.1.05: Prueba 5

Identificador	PR.06
Objetivo	Verificar que robot_fw genera ficheros logs y crea una base de datos SQLite.
Procedimiento	Para esta prueba, se crea un programa en el que se incluye el fichero de cabecera “Driver.h”. Se compila y se enlaza con la librería del Framework robot_fw (librobot_fw.so), y finalmente se ejecuta.
Resultado	Cuando finaliza la ejecución, en la carpeta del programa, se ven dos nuevos ficheros: “robot_log_2014-02-21_01-19-34-916.log” que es el fichero de logs y “db_robot.sqlite” que la base de datos SQLite.
Aceptado	Sí

Tabla 6.1.06: Prueba 6

Identificador	PR.07
Objetivo	Verificar que robot_fw permite controlar los servos Dynamixel AX-12.
Procedimiento	Se conecta un servo Dynamixel a la placa controlador mediante un adaptador RS-232/TTL (la conexión entre el adaptador y el servo se puede ver en la **¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.**). Se crea un programa en que se crea un objeto de la clase DynamixelServo y otro de la clase SerialPort. Se emplea el primer objeto para encapsular las instrucciones (según especifica el manual de los servos Dynamixel) y el segundo se emplea para enviarlos por el puerto serie, al que se encuentra conectado el servo. Las instrucciones del servo eran activar el LED de estado, girar a la izquierda y luego a la derecha.
Resultado	El servo actuó acorde a las instrucciones de prueba.
Aceptado	Sí

Tabla 6.1.07: Prueba 7

Identificador	PR.08
Objetivo	Verificar que robot_fw permite gestionar un envío masivo de instrucciones a los servos.
Procedimiento	Se emplea el mismo cirquito de conexión, con la diferencia de que, esta vez se conectan dos servos (A y B) al BUS Dynamixel (ver **¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.**). Se crea un programa en el que se crea un objeto de la clase DynamixelServo, un objeto de la clase PacketManager y otro de la clase SerialPort. Para probar que la gestión del PacketManager por prioridad, tiempo y destino se realiza correctamente, se plantea la prueba de la siguiente forma: enviar dos instrucciones de movimiento por cada servo para que giren una vuelta a la derecha y luego otra a la izquierda. Las instrucciones del servo A se envían a la cola con mayor prioridad, y las del B a una de menor prioridad.
Resultado	El servo A gira a la derecha, acto seguido le siguió el servo B. después de llegar a la posición final, el A empieza a volver a su posición inicial, seguido del servo B.
Aceptado	Sí

Tabla 6.1.08: Prueba 8

Identificador	PR.09
Objetivo	Cómo puede el sistema comunicar con simuladores de Hardware.
Procedimiento	El Framework ofrece la interfaz SerialPortI que define las operaciones sobre un puerto de entrada y salida. Para el sistema de control pueda comunicar con un simulador, basta con implementar la interfaz SerialPortI, de modo que el puerto (COM o TCP) que gestiona la implementación sirva de puente.
Resultado	
Aceptado	Sí

Tabla 6.1.09: Prueba 9

6.2.Implantación

En esta sección se va a detallar la implantación final del sistema. Este proceso se divide en las siguientes etapas:

Instalación de la distribución Debian

La distribución Debian 7.0 dispone de un instalador flexible, que permite una instalación básica, con los componentes mínimos necesarios. Para la instalación del sistema Linux, existen tres fases importantes:

1. **Crear una máquina virtual**: Para acelerar el proceso de instalación, se emplea una máquina virtual (VirtualBox). En el asistente de creación de la maquina, se elige el tipo de sistema de Debian, y los recursos en función de la maquina real (Host). Un paso importante para crear la maquina, a la hora de crear el disco, hay que seleccionar la opción de mapear el disco a un disco físico (en nuestro caso, sería la tarjeta micro-SD, o la que se considera conveniente).

2. **Instalar Debian**: Necesitamos el CD de instalación de Debian (se puede descargar de la página web de Debian). Se enlaza el CD con la maquina creada en el paso 1. Desde aquí, el proceso es el mismo que cuando se instala un sistema Linux en una maquina física (real), con una peculiaridad, hay que instruir al asistente de instalación para que efectúe una instalación básica (con software mínimo y sin entorno gráfico).

3. **Inserta disco en la Ebox**: Con los pasos anteriores, lograríamos construir un disco (micro-SD) con Debian instalado y configurado en su sistema de ficheros. El disco se inserta en la placa Ebox, se inicia la placa, y ¡voilà! El sistema Linux, recién instalado, se inicia con normalidad.

Para ver más detalles del proceso de instalación de una distribución Linux, ver el siguiente manual[20].

Integración de Xenomai con el Kernel

Para llevar a cabo la instalación de Xenomai (integración con el Kernel), se necesita tanto el código fuente del Kernel (versión 3.2.24) como el de Xenomai (versión 2.6.1). Antes de lanzar la compilación, se aprovecha para realizar las configuraciones oportunas, para adaptar el Kernel al Hardware de la placa controladora.

Para que el Kernel sea compatible con la placa, hay que importar el fichero de configuración del Kernel (`.config`) ofrecido por el fabricante, que se encuentra la carpeta /boot. Éste ya proporciona una imagen del Kernel (linux-image-2.6.34.9-vortex86-sg_1.1_i386[21]). Se desempaqueta la imagen, y se importa la configuración del Kernel a nuestro nuevo Kernel. Con el siguiente comando se importa la configuración del fabricante:

1. Se copia el fichero `.config` a la carpeta de fuentes del nuevo Kernel.
2. Se ejecuta `make oldconfig`.
3. Desde aquí, se sigue el procedimiento habitual de configuración y compilación del Kernel.

Con aquello, conseguimos importar las configuraciones específicas del Hardware de la placa, y las nuevas características del nuevo Kernel se mantienen por defecto.

El proceso de instalación de Xenomai se divide en tres pasos:

1. **Preparación del Kernel seleccionado**: Se integra el código fuente con el código fuente del Kernel. Para ello, Xenomai, proporciona un script para preparar el Kernel:
```
$> cd $xenomai_root
$> scripts/prepare-kernel.sh --linux=<linux-srctree> [--
adeos=<adeos-patch>] [--arch=<target-arch>]
```

2. **Configuración y compilación del Kernel**: Una vez preparado el Kernel, se configura y se compila con los procedimientos habituales (make xconfig, gconfig o menuconfig):
```
$> cd $linux_tree
$> make ARCH=i386 menuconfig (para una arquitectura x86)
```
Y para compilar, y generar los paquetes del Kernel:

```
$> make ARCH=i386 bzImage modules
```

3. **Compilación de librerías del espacio de usuario**: Para compilar las librerías, Xenomai sigue el procedimiento regular, mediante script de autoconfiguración:

```
$>mkdir $build_root && cd $build_root
$  $xenomai_root/configure  --enable-x86-sep  [--host=i686-linux
CFLAGS="-m32 -O2" LDFLAGS="-m32"]
$ make install
```

Para la integración de Xenomai en otras versiones del Kernel, o estudiar cómo funciona cada paso, el sitio web oficial de Xenomai[22] ofrece más detalles y ejemplos de cómo proceder.

Habilitar puertos COM

Hay que cargar el modulo (driver) de los puertos serie, ya sea el nativo, o el de Xenomai. Para usar la interfaz de Xenomai, primero hay que cargar el driver de de tiempo-real, de la siguiente forma:

```
$> modprobe xeno_16550A io=<io1>[,<io2>...] irq=<irq1>[,<irq2>...]
                    [baud_base=<base1>[,<base2>...]]
                    [tx_fifo=<len1>[,<len2>...]] [start_index=<index>]
```

Para identificar los argumentos (io, irq, baud_base,…) del puerto serie que se desea gestionar con el driver de Xenomai, empleamos el comando `setserial /dev/ttyS<N>` (La N es un número que identifica el puerto serie).

Si el sistema que se desea construir, no es un sistema crítico, simplemente empleamos el driver nativo del Kernel. Para cargar el driver, empleamos el siguiente: modprobe <modulo del kernel>.

6.3.Matriz de trazabilidad de la funcionalidad

	PR.01	PR.02	PR.03	PR.04	PR.05	PR.06	PR.07	PR.08	PR.09
RS.S01-01	X								
RS.S01-02	X								
RS.S01-03									
RS.S01-04		X			X				
RS.S01-05			X						
RS.S01-06	X								
RS.S01-07				X					
RS.S01-08			X						
RS.S02-01					X				
RS.S02-02					X		X		
RS.S02-03								X	
RS.S02-04			X						
RS.S02-05									X
RS.S02-06						X			
RS.S02-07						X			
RS.S02-08							X		

Tabla 6.4.01: Matriz de trazabilidad de funcionalidades

7. Planificación y presupuesto

En este capítulo se detalla la planificación del proyecto y el cálculo del presupuesto necesario para el desarrollo.

7.1.Planificación

En este apartado se detallará el tiempo dedicado al desarrollo del trabajo. Se muestra el listado de tareas y el tiempo dedicado a cada una, acompañando el diagrama de Gantt.

Listado de tareas

En la siguiente tabla se exponen las tareas realizadas correspondientes a las fases del proyecto, la fecha de inicio y de fin y la duración en días. Se añade una tarea más, que es la documentación, que al estar incluida en las otras fases del proyecto, no se ha puesto como una fase independiente. A tal efecto, las fecha inicio y fin recoge la suma de días dispersos que se han dedicado a esta última tarea. Un aspecto a tener en cuenta, los fines de semana y los festivos están excluidos de la duración de la tareas.

Tarea (fase)	Fecha inicio	Fecha Fin	Duración en días
Análisis de necesidades	10/09/12	10/09/12	1
Definición de objetivos	11/09/12	05/10/12	19
Análisis del sistema.	08/10/12	08/11/12	23
Planificación y presupuesto.	09/09/13	20/09/13	10
Diseño del sistema	23/09/13	23/10/13	21
Implantación y pruebas.	28/10/13	17/01/14	53
Documentación.	20/01/14	14/02/14	20

Tabla 7.1.1: Planificación de tareas

En la siguiente ilustración se muestra el diagrama de Gannt con las tareas desarrolladas.

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Análisis de necesidades	10/09/2012	10/09/2012	1d
2	Definición de objetivos	11/09/2012	05/10/2012	19d
3	Análisis del sistema	08/10/2012	08/11/2012	24d
4	Planificación y presupuesto	09/09/2013	20/09/2013	10d
5	Diseño del sistema	23/09/2013	23/10/2013	23d
6	Implementación y pruebas	28/10/2013	17/01/2014	60d
7	Documentación	20/01/2014	14/02/2014	20d

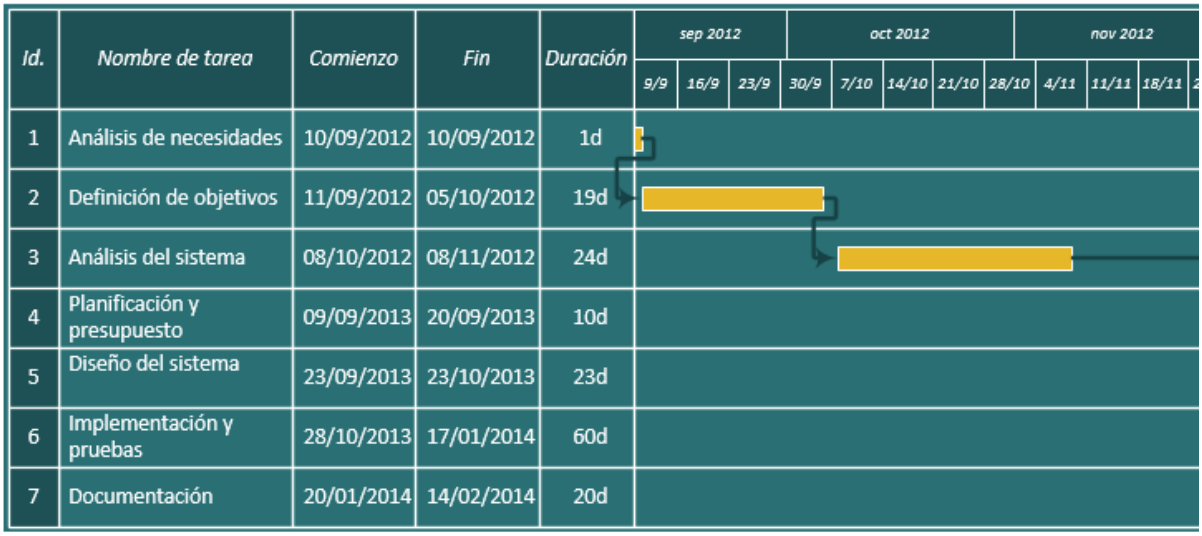


Ilustración 27: Diagrama de Gantt (parte 1)

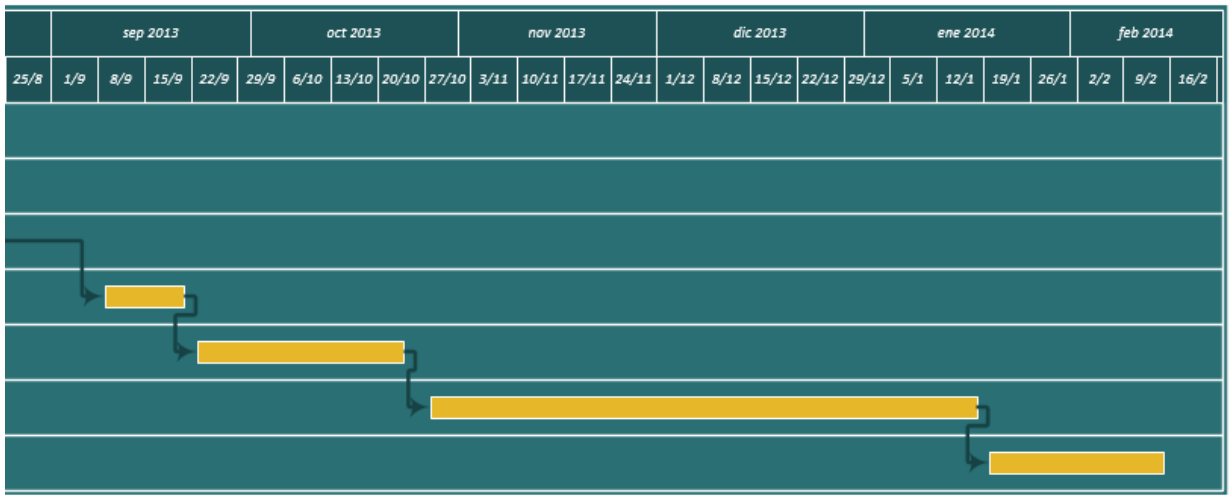


Ilustración 28: Diagrama de Gantt (parte 2)

7.2.Presupuesto

Tiempo dedicado

En este apartado se va a detallar el presupuesto del proyecto. Para los cálculos de los costes se han tenido en cuenta las siguientes consideraciones:

- La fecha de **inicio** de proyecto es el **10 de septiembre de 2012**, y la fecha de **fin** el **14 de febrero de 2014**.
- Se toman como referencia la duración en días de las distintas tareas expuestas en el apartado de planificación. En dicha duración ya se tiene en cuenta que los fines de semana y los festivos nacionales no se trabajan.
- El **total de días** dedicados al proyecto asciende a **147 días**.

- La jornada diaria de trabajo ha sido variable, pero se estima una **media de 6 horas diarias**.
- En total, **el número de horas trabajadas** en el proyecto es de **882**.

Coste de personal

En esta sección se va a especificar los costes asignados al coste de. Debido a la naturaleza del proyecto, Trabajo Fin de Grado, la única persona involucrada en la realización de este proyecto ha sido el autor de este documento, Kamal El Maataoui, que será el que asuma todos los roles especificados a continuación:

Rol	Coste (€/hora)	Número de horas	Coste total (€)
Analista	41	258	10.578,00
Diseñador	31	186	5.766,00
Programador	27	438	11.826,00
	TOTAL	**882**	**28.170,00**

Tabla 7.2.1: Coste de personal

Coste de Hardware

En esta sección se mostrarán los gastos del proyecto imputables a elementos hardware imprescindible para el desarrollo de la plataforma de control.

En la siguientes dos tablas se recogen los elementos hardware adquiridos para el proyecto:

Concepto	Coste (€)	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación (meces)	Coste imputable (€)
Ordenador HP	435	55	7	60	50,75
Monitor Samsung	299	100	7	60	34,88
Teclado PS2 Deluxe	14,99	100	7	60	1,75
Ratón USB Logitech	10,50	100	7	60	1,23
				TOTAL	**88,61**

Tabla 7.2.2: Amortización de componentes

(*) Fórmula de amortización:

$$\frac{A}{B} xCxD$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto

Ilustración 29: Formula de amortización

Concepto	Unidades	Coste unidad sin IVA(€)	Coste total sin IVA (€)
Placa Ebox 3310MX-AP	1	179,9	179,90
Servo Dynamixel AX-12	1	45,00	45,00
Adaptador RS-232/TTL	2	6,95	13,90
Fuente de alimentación	1	13,5	13,5
Placa Breadboard	1	6,95	6,95
Cables	5	0,10	0,50
		TOTAL	**259,75**

Tabla 7.2.2: Coste Hardware

Coste de Software

En esta sección se mostrarán los gastos del proyecto imputables a licencias de Software imprescindible para el desarrollo de la plataforma de control. La plataforma está basada en código abierto, por lo que el coste Software es nulo.

Concepto	Coste (€)
Ubuntu 12.04	0,00
Debian 7.0	0,00
Xenomai 2.6.1	0,00
SQLite 3.0	0,00
Eclipse (IDE)	0,00
Total	**0,00**

Tabla 7.2.3: Coste de software

Resumen de costes

La siguiente tabla recoge la suma de todos los costes anteriormente expuestos:

Tipo de coste	Coste total sin IVA (€)
Personal	28.170,00
Amortización de equipos	88,61
Componentes Hardware	259,75
Licencias de Software	0,00
Costes indirectos (20 %)	5.703,67
Total sin IVA	34.222,03
Total con IVA (21 %)	**41.408,66**

Tabla 7.2.3: Resumen de costes y coste total

El presupuesto total es de CUARENTA Y UN MIL CUATROCIENTOS OCHO CON SESENTA Y SEIS CÉNTIMOS DE EURO.

8. Conclusiones y líneas futuras

Expuestos los objetivos del trabajo, la tecnología a utilizar y validado las pruebas efectuadas sobre la plataforma de control, formularemos las conclusiones sobre este trabajo y las posibilidades de mejorarlo y ampliarlo en el futuro.

8.1.Conclusiones

Objetivos logrados

Para la consecución del trabajo se han utilizado los componentes más adecuados cumpliendo los siguientes requerimientos:

- Utilización de software libre.
- Sistema empotrado de bajo consumo
- Elevadas prestaciones.
- Reducido coste y tamaño.

Se ha logrado controlar satisfactoriamente los servos Dynamixel AX-12.
Se ha desarrollado el framework robot_fw que ofrece las siguientes utilidades:

- Envío desatendido de paquetes, teniendo en cuenta el solapamiento de comandos, prioridades de las instrucciones, y el tiempo de espera en paquetes para tareas de sincronización.
- Encapsular la configuración de los puertos series en una interfaz sencilla.
- Control remoto del sistema robótico a través de internet.
- Gestión de recursos: memoria y puertos de entrada y salida.

Con todo lo anterior podemos afirmar que el objetivo del trabajo se cumple con éxito.

Conclusiones personales

A lo largo del trabajo me he encontrado con dificultades que me han supuesto retrasos importantes en el avance del proyecto. Uno de ellos ha sido la compilación de la distribución de Linux desde cero, que funciono en la máquina sobre la que se desarrolló, pero resultó incompatible con la placa utilizada. Como alternativa se cambió la distribución por una Debian. Otra dificultad encontrada fue dar con la configuración del puerto serie para controlar los servos Dynamixel y extenderla para otros puertos.
Desde el punto de vista personal, en primer lugar la realización del trabajo me ha supuesto poner en práctica conocimientos adquiridos durante la carrera. En segundo lugar, adquirir nuevos conocimientos de robótica, sistema Linux y la configuración y compilación del kernel.

8.2.Líneas futuras

Hoy en día, conseguir placas con altas prestaciones y bajo consumo es fácilmente alcanzable y factible.

El trabajo se puede orientar hacia dos principales usos: la robótica y domótica. Se puede adquirir el hardware necesario e implementar programas que lo controlen, ya sea de forma local o remota.

En el framework de apoyo no se ofrecido una implementación de los sensores. Como mejora se puede ofrecer más implementaciones de otros tipos de sensores (IR, sonar, luz, temperatura,…). Para integrarlo en un sistema de domótica sería más ventajoso.

En cuanto a la robótica, la mejora sustancial sería añadir otra capa de abstracción de facilidades y mejoras, como por ejemplo, las localizaciones, reconocimiento de obstáculos, procesamiento de imágenes, ya que el hardware especificado permite realizar este tipo de mejoras.

9. ANEXOS

En este capítulo se recopilan algunos documentos o información que no tenía lugar en el resto del documento ya sea por espacio o por no estar relacionado con ninguno de los puntos a desarrollar, pero que sí es importante añadir a este documento.

9.1.ANEXO B: Manual de uso del Framework de apoyo

Antes de empezar, se parte de la premisa de que el entorno está instalado funcionando (ver 6.2).

Configuración previa:

En primer lugar se copian la cabeceras y las librerías ofrecida por el Framework robot_fw. Para ello crean dos carpetas:

/demo/libs: en esta carpeta se copian las librarías (librobot_fw.so y libsqlite3.so).

- /demo/headers: se copian todas las carpetas de cabeceras del robot_fw.

Modo de uso:

Para usar el framework adecuadamente se debe seguir los siguientes pasos:

1. Implementar la función `void initRobotComponent(void)`, en la que se crean todos los componentes de nuestro sistema robótico (puertos serie, Servos, Robot, crear tablas de bases de datos,…). Por cada elemento creado se debe usar la funciones correspondientes del Driver para mantenerlos accesibles desde otro puntos del programa (`add_Port(), add_servoSensor()`,…. En el fichero de cabecera Driver.h, se puedan emplear y las clase que se añaden al Driver).
2. En la función main, se debe invocar la función `int main_proc(int n, char *argv[])`, que constituye el punto de de inicio de ejecución del robot_fw, y del sistema robótico.

Es importante tener en cuenta que, la clase que implementa la interfaz **`RobotI`**, encapsula todo el funcionamiento del sistema robótico. Todas las tareas de nuestro sistema robótico, se inician desde esta clase (es recomendable).

A lo largo de la ejecución del robot (sistema robótico), se selecciona un comportamiento activo (un objeto de la clase que implementa la interfaz `BehaviorI`), de entre todos los comportamientos creados por el usuario. Mediante un comportamiento, se instruye al robot a realizar tareas específicas (como caminar un paso,…).

Nota: También se pueden emplear los componentes del robot_fw de forma independiente, sin seguir la estructura lógica, descrita anteriormente. Por ejemplo: usar la clase `SerialPort, DynamixelServo, TimeStamp`,…

Ejemplos de uso:

El este ejemplo, se va a ver cómo construir un programa para controlar un robot Bioloid. El fichero inicial del programa, puede tener la siguiente estructura:

robot01.cpp | BioloidRobot.cpp | Accelerometer.cpp | Accelerometer.h | BioloidRobot.h | WalkBehavior.cpp | WalkBehavior.h

```
/***
 * Ejemplo de uso del framework.
 */
#include <include/types.h>
#include <include/defs.h>
#include <servomotor/DynamixelServo.h>
#include <serial/SerialPort.h>
#include <robot/Driver.h>
#include <robot/BehaviorI.h>
#include <exceptions/RobotException.h>
#include "Accelerometer.h"
#include "BioloidRobot.h"
#include "WalkBehavior.h"

using namespace std;
using namespace robot_fw;

/**
 * El usuario debe implementar esta funcion, y realizar las siguientes tareas:
 * 1- crear los puertos serie deseados, y por cada puerto llamar a add_Port(puerto).
 * 2- crear todos los servos/sensores, y por cada uno llamar a add_servoSensor(servo_sensor).
 * 3- si es necesario, crear las tablas de la BD. Usar get_database() para obtener la BD.
 *
 */
void initRobotComponent(void) {
    int port_i, servo_i;

    /*
     * abrir un fichero especifico para los logs, por defecto es la salida estandar (cout)
     */
    //OPEN_LOGFILE("robot01");

    robot_fw::SerialPortI *port = new robot_fw::SerialPort(std::string("/dev/ttyS0"), 115200, 1, 8
    , false,
            false);

    /*
     * Por cada puerto que se crea, incrementa este variable, porque es la que asigna un indice a
     cada puerto
     * en la tabla de puertos.
     */
    port_i = 0;
    if(port->openPort() == false){
        // salir
        throw new RobotException("No se ha podido abrir el puerto  /dev/ttyS0!");
    }
    add_Port(port);

```

C++ source file | length : 2513 lines : 93 | Ln : 34 Col : 20 Sel : 0 | UNIX | ANSI | INS

Ilustración 30: main.cpp (parte 1)

robot01.cpp | BioloidRobot.cpp | Accelerometer.cpp | Accelerometer.h | BioloidRobot.h | WalkBehavior.cpp | WalkBehavior.h

```
46
47      /*
48       * Por cada servo o sensor que se crea, incrementa esta variable, porque es la que asigna un
         indice a cada
49       * servo o sensor en la tabla de servo_sensores.
50       */
51      servo_i = 0;
52      ServoI *servo_dxl = new DynamixelServo(servo_i++, port_i);
53      add_servoSensor(servo_dxl);
54
55
56      ServoI *accelor = new Accelerometer(servo_i++, port_i);
57      add_servoSensor(accelor);
58
59      /*
60       *
61       */
62      RobotI *my_robot = new BioloidRobot();
63      add_robot(my_robot, true);
64
65      BehaviorI* behav = new WalkBehavior();
66      add_behavior(behav);
67  }
68
69  /**
70   *
71   * @param n
72   * @param argv
73   * @return
74   */
75  int main(int n, char *argv[]){
76      int ret = EXIT_FAILURE;
77      try {
78          ret = main_proc(n,argv);
79      } catch (std::out_of_range &e) {
80          LOG_ERROR("out_of_range exception: "+std::string(e.what()));
81      } catch (RobotException *e) {
82          LOG_ERROR("Caught Robot exception: "+std::string(e->what()));
83      }/* catch (exception &e) {
84          LOG_ERROR("Caught an exception of an unexpected type: "+std::string(e.what()));
85      } catch (std::runtime_error &e) {
86          LOG_ERROR("Caught a runtime_error exception: "+std::string(e.what()));;
87      }*/ catch (...) {
88          LOG_ERROR("Exception: Caught an unknown exception");
89      }
90      CLOSE_LOGFILE();
91      return ret;
92  }
```

C++ source file | length : 2513 lines : 93 | Ln : 73 Col : 11 Sel : 0 | UNIX | ANSI | INS

Ilustración 31:main.cpp (parte 2)

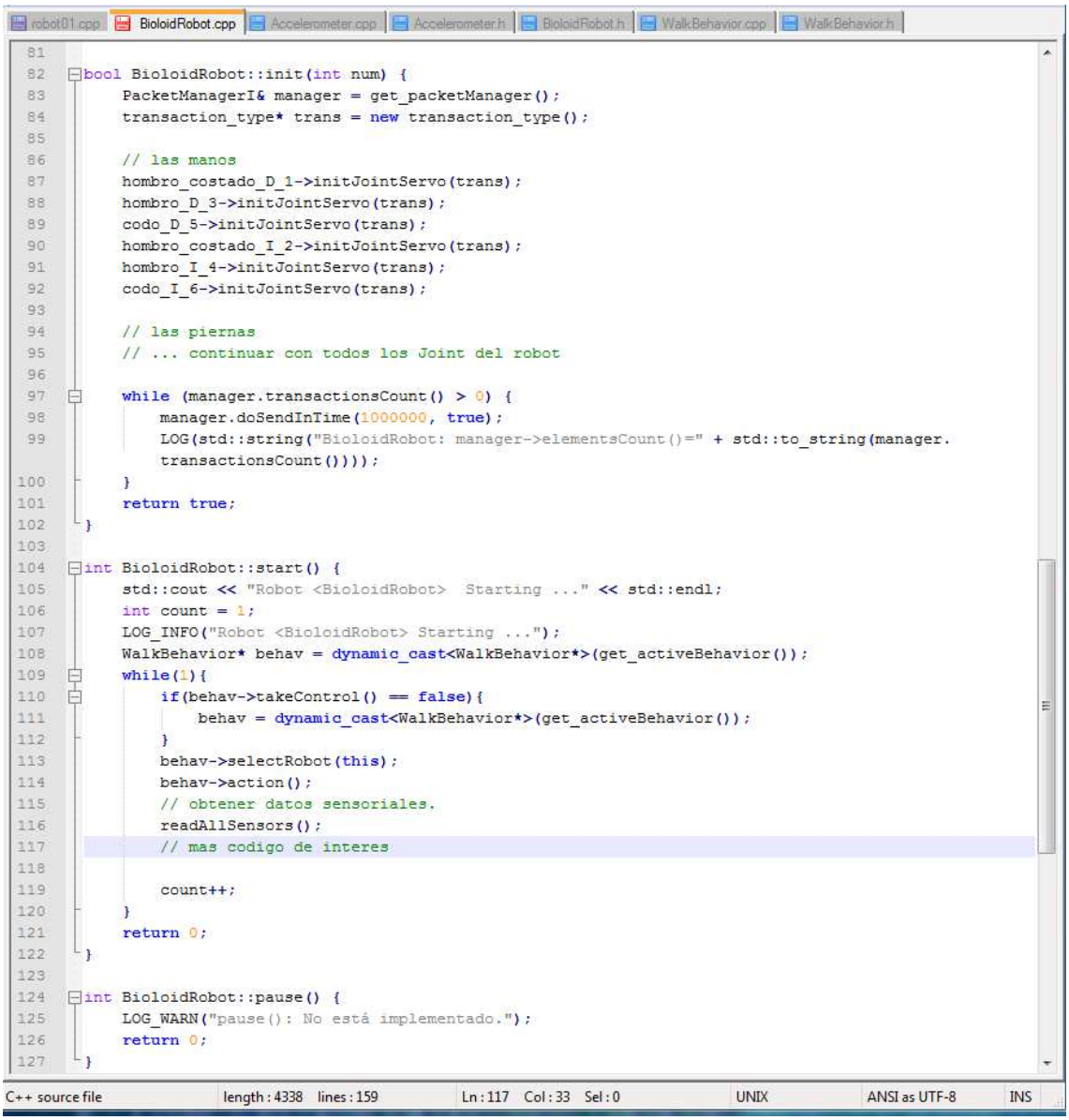
La implementación de la interfaz RobotI:

robot01.cpp | BioloidRobot.cpp | Accelerometer.cpp | Accelerometer.h | BioloidRobot.h | WalkBehavior.cpp | WalkBehavior.h

```
10
11  #include <include/types.h>
12  #include <robot/RobotI.h>
13  #include <robot/Driver.h>
14  #include "WalkBehavior.h"
15
16  using namespace robot_fw;
17
18  class WalkBehavior;
19
20  class BioloidRobot: public virtual robot_fw::RobotI {
21  public:
22      BioloidRobot();
23      virtual ~BioloidRobot();
24
25      bool init(int num);
26      int start();
27      int pause();
28      void die(std::string msg);
29
30      /**
31       * Clases amigas, que puede acceder a los atributos privados del objeto robot.
32       */
33      friend class WalkBehavior;
34
35      /**
36       * Funciones propias para esta implementacion
37       */
38
39  private :
40      /*
41       * el  numero al final de cada variable corresponde al ID fisico del servo
42       * asociado al Joint.
43       */
44      // las manos
45      joint_t hombro_costado_D_1;
46      joint_t hombro_D_3;
47      joint_t codo_D_5;
48      joint_t hombro_costado_I_2;
49      joint_t hombro_I_4;
50      joint_t codo_I_6;
51
52      // las piernas
53      // ... añadir el resto de Joints
54  };
55
56  #endif /* MYROBOT_H_ */
57
```

++ source file | length : 992 lines : 57 | Ln : 53 Col : 37 Sel : 0 | UNIX | ANSI | INS

Ilustración 32: Robot.h




```
81
82  bool BioloidRobot::init(int num) {
83      PacketManagerI& manager = get_packetManager();
84      transaction_type* trans = new transaction_type();
85
86      // las manos
87      hombro_costado_D_1->initJointServo(trans);
88      hombro_D_3->initJointServo(trans);
89      codo_D_5->initJointServo(trans);
90      hombro_costado_I_2->initJointServo(trans);
91      hombro_I_4->initJointServo(trans);
92      codo_I_6->initJointServo(trans);
93
94      // las piernas
95      // ... continuar con todos los Joint del robot
96
97      while (manager.transactionsCount() > 0) {
98          manager.doSendInTime(1000000, true);
99          LOG(std::string("BioloidRobot: manager->elementsCount()=" + std::to_string(manager.
            transactionsCount())));
100     }
101     return true;
102 }
103
104 int BioloidRobot::start() {
105     std::cout << "Robot <BioloidRobot>  Starting ..." << std::endl;
106     int count = 1;
107     LOG_INFO("Robot <BioloidRobot> Starting ...");
108     WalkBehavior* behav = dynamic_cast<WalkBehavior*>(get_activeBehavior());
109     while(1){
110         if(behav->takeControl() == false){
111             behav = dynamic_cast<WalkBehavior*>(get_activeBehavior());
112         }
113         behav->selectRobot(this);
114         behav->action();
115         // obtener datos sensoriales.
116         readAllSensors();
117         // mas codigo de interes
118
119         count++;
120     }
121     return 0;
122 }
123
124 int BioloidRobot::pause() {
125     LOG_WARN("pause(): No está implementado.");
126     return 0;
127 }
```

Ilustración 33: Robot.cpp, implementación de la interfaz RobotI

Implementación de la interfaz BehaviorI:

robot01.cpp | BioloidRobot.cpp | Accelerometer.cpp | Accelerometer.h | BioloidRobot.h | WalkBehavior.cpp | WalkBehavior.h

```
33  void WalkBehavior::action(void) {
34      int count = 0;
35      PInstructionI* instruct;
36      transaction_type * trans;
37
38      TimeStamp t0, t2;
39      //char tmp[10] = { 0 };
40
41      PacketManagerI& manager = get_packetManager();
42      DynamixelServo& servo = dynamic_cast<DynamixelServo&>(*get_servoSensor_at(0));
43      std::cout << "transaction: " << 1 << "\n:";
44      trans = new transaction_type();
45      /*
46      name=R blocking
47      compliance=5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
48      play_param=14 15 1 1.2 4
49      ID      1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  pause  time
50      step= 284 832 385 711 484 522 353 671 405 521 262 543  30 572 774 458 534 647  0.400  0.600
51      */
52      int speed = 300;
53      Position* pos = new Position(284, speed);
54      selected_robot->hombro_costado_D_1->assertPosition(*pos);
55      instruct = servo.moveSpeed(pos->ID_servo, pos->gol_pos,pos->speed);
56      instruct->setTimeOfIstruccion(pos->time_operation);
57      //instruct->setTimeOfSync(0);
58      while (instruct != NULL) {
59          std::cout << "WalkBehavior: instruct: " << count++ << "\n:";
60          trans->addData(instruct);
61          instruct = instruct->getNextInstruction();
62      }
63
64      pos->gol_pos=832; pos->speed=speed;
65      selected_robot->hombro_costado_I_2->assertPosition(*pos);
66      instruct = servo.moveSpeed(pos->ID_servo, pos->gol_pos,pos->speed);
67      instruct->setTimeOfIstruccion(pos->time_operation);
68      //instruct->setTimeOfSync(0);
69      while (instruct != NULL) {
70          std::cout << "WalkBehavior: instruct: " << count++ << "\n:";
71          trans->addData(instruct);
72          instruct = instruct->getNextInstruction();
73      }
74
75      pos->gol_pos=385; pos->speed=speed;
76      selected_robot->hombro_D_3->assertPosition(*pos);
77      instruct = servo.moveSpeed(pos->ID_servo, pos->gol_pos,pos->speed);
78      instruct->setTimeOfIstruccion(pos->time_operation);
79      //instruct->setTimeOfSync(0);
80      while (instruct != NULL) {
```

C++ source file | length : 9054 lines : 268 | Ln : 1 Col : 1 Sel : 0 | UNIX | ANSI | INS

Ilustración 34: Behavior.ccp, implementación de la interfaz BehaviorI

Para compilar el código, hay que pasarle al compilador la opción (-I “/demo/headers”). Para el linkador, se le pasa la opción (-L “/demo/libs”).

10. Glosario de Términos

BIOS: (Basic Input-Output System) es un programa informático inscrito en componentes electrónicos de memoria Flash existentes en la placa base

Código fuente: es un conjunto de líneas de texto que forman un programa a modo de instrucciones que debe seguir la computadora para ejecutar dicho programa.

Compilador: un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.

Corriente continua: se refiere al flujo continuo de carga eléctrica a través de un conductor entre dos puntos de distinto potencial, que no cambia de sentido con el tiempo

Datasheet: es un documento que resume el funcionamiento y características técnicas de un componente, con detalle para que el ingeniero pueda utilizarlo en el diseño de un sistema.

EEPROM: son las siglas de Electrically-Erasable Programable Read-Only Memory) Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente

Firmware: bloque de instrucciones de máquina para propósitos específicos, grabado en una memoria, normalmente de lectura / escritura (ROM, EEPROM, ash, etc), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.

Memoria Fash: permite la lectura y escritura de múltiples posiciones de memoria en la misma operación. Puede ser reescrita miles de veces, lo que representa una gran ventaja para reutilizar el controlador y para programar sin temor al error.

PIC: son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instrument.

Protoboard o Breadboard: tablero con orificios conectados eléctricamente entre sí, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares.

Potenciómetro: resistor cuyo valor de resistencia es variable. De esta manera, indirectamente, se puede controlar la intensidad de corriente que fluye por un circuito si se conecta en paralelo, o la diferencia de potencial al conectarlo en serie.

PWM: Son las siglas en ingles de pulse-duration modulation, es una técnica de modulación que permite generar ondas cuadradas con una determinada frecuencia y ciclo de actividad. La frecuencia la representa el reloj y el ciclo de actividad la representa la anchura del pulso, es decir, cuanto tiempo de cada onda cuadrada hay pulso (5V) y cuando no lo hay (0V).

RAM: (en inglés: random-access memory) se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayoría del software.

Torque: proviene del termino ingles torque, derivado del latín torquere (retorcer). Este término intenta introducirse en la terminología española bajo las formas de torque o torca. El torque hace que se produzca un giro sobre el cuerpo que lo recibe.

USART: (Universal Synchronous Asynchronous Receiver Transmitter) módulo controla los puertos y dispositivos serie.

BIOLOID[23]: (Bio + All + Droid) el nombre del robot humanoide del fabricante ROBOTIS.

11. BIBLIOGRAFÍA

Bjarne Stroustrup, Trends and future of C++. Stroustrup - Madrid 2011.

Bjarne Stroustrup, the C++ Programming Language, Third Edition 1997.

[1] LIBRO BLANCO DE LA ROBOTICA : http://www.ceautomatica.es/sites/default/files/upload/10/files/LIBRO BLANCO DE LA ROBOTICA 2_v2.pdf

[2] Gonzalo Zabala: Robótica, guía teórica y práctica.

[3] RS-232 (Recommended Standard 232): http://www.ti.com/lit/an/slla037a/slla037a.pdf

[4] Servo (Actuador) Dynamixel: http://support.robotis.com/en/techsupport_eng.htm#product/dynamixel/dxl_ax_main.htm

[5] TCP/IP: http://www.ietf.org/rfc/rfc793.txt , http://technet.microsoft.com/en-us/library/cc958821.aspx

[6] UML (The Unified Modeling Language): http://www.uml.org/

[7] EBOX-3310MX: http://www.compactpc.com.tw/en/product/EBOX-3310MX-AP/ebox-3310mx-ap.html

[8] DMP Electronics: http://www.compactpc.com.tw/es/index.html

[9] RoBoard RB-110: http://www.roboard.com/RB-110.htm

[10] Vortex86MX: http://www.dmp.com.tw/tech/vortex86mx/

[11] MAX-232 http://www.ti.com/lit/ds/symlink/max232.pdf

[12] Linux desde cero (Linux From Scratch): http://www.linuxfromscratch.org/

[13] Ubuntu: http://www.ubuntu.com/

[14] Debian: http://www.debian.org/

[15] GNU (Licencia Pública General de GNU, o GNU General Public License en inglés) https://www.gnu.org/licenses/licenses.es.html

[16] Descargas alternativas de Ubuntu: http://es.archive.ubuntu.com/cdimage/releases/

[17] Kernel-3.2.24 código fuente: https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.2.24.tar.gz

[18] Xenomai-2.6.1 código fuente: http://download.gna.org/xenomai/stable/xenomai-2.6.1.tar.bz2

[19] SQLite: http://www.sqlite.org/

[20] Instalación de una distribución Linux (Ubuntu, Debian) mediante VirtualBox: http://www.roboard.com/Files/RB-100/Install_Debian_on_RoBoard.zip

[21] Imagen del Kernel para Vortex86: http://www.roboard.com/Files/RB-100/linux-image-2.6.34.10-vortex86-sg_1.2_i386.zip

[22] Instalación de Xenomai: http://www.xenomai.org/documentation/xenomai-2.6/html/README.INSTALL/

[23] BIOLOID: http://www.robotis.com/xe/BIOLOID_main_en

[] Foro de robótica: http://robosavvy.com/web/