UNIVERSIDAD
POLITECNICA
DE VALENCIA

Implementación de controladores usando la librería de tiempo real del entorno Orocos

Alumno: Jose Ignacio Cazalilla Morenas

Director: Ángel Valera Fernández

Codirectora: Marina Vallés Miquel

Valencia, 1 de Julio de 2011

1 Introducción

Es conocido por todos que los robots industriales están presentes en la mayoría de los procesos de fabricación actuales. Gran cantidad de robots son programados para que realicen diferentes tareas de forma eficaz y repetitiva. En el mundo en el que vivimos, donde la dura competencia, la calidad del producto, los plazos de entrega, y sobre todo el precio del producto final marcan el éxito o fracaso de cualquier multinacional o pequeña y mediana empresa, es muy importante optimizar los procesos de fabricación. En este punto está muy presente el campo de la robótica, y los robots industriales tienen un importante papel en el logro de un producto de calidad a bajo precio.

La dura competencia, los continuos cambios en elementos hardware y software, y sobre todo la complejidad de las tareas sobre las que los robots deben interactuar hacen necesario un nuevo enfoque de programación. Por ejemplo, en la actualidad muchas aplicaciones de robots industriales, integran visión artificial, reconocimiento de formas, planificación de trayectorias… Todos estos conceptos y más son de un alto nivel, y provocan que la fusión de todos los elementos que los hacen posible en una aplicación, que los gobierne adecuadamente para que junto con el robot se pueda realizar la tarea objetivo de forma adecuada, sea complicado. Pensemos que cada elemento citado anteriormente puede ser de una marca determinada, o bien esté programado en un determinado lenguaje…

La aplicación de técnicas avanzadas de control permiten mejorar sustancialmente las prestaciones desistemas mecánicos complejos sin alterar significativamente su estructura mecánica.Un buen ejemplo de elloes la incorporación del control dinámico a robots industriales que, hasta hace pocos años, tenían un sistema decontrol puramente cinemático. Sin embargo, hay que tener en cuenta que, debido a las característicasdinámicas de este tipo de técnicas, los controladores resultantes derivan en aplicaciones de tiempo real querequieren tener un cierto cuidado en su implementación para que los resultados obtenidos sean satisfactorios.

En el caso de los sistemas mecánicos complejos, su implementación puede llevar a la aparición de múltiplesbucles de control, con distintas variables a medir o

estimar según la magnitud sea medible a través desensores o no, con distintos requerimientos tanto temporales como de recursos de computación y que,además, necesiten una cierta sincronización a la hora de compartir información entre ellos.

Con el presente proyecto se pretende evaluar la mejora en prestaciones que puede suponer la implementaciónen tiempo real de controladores para un robot paralelo, previamente diseñados, usando el entorno dedesarrollo de software Orocos.

Objetivos de este proyecto:

- Estudiar las librerías proporcionadas por el entorno Orocos.
- Estudio de componentes sencillos, así como la prueba de los mismos.
- Implementación de componentesbásicos y utilizarlos para la consecución de tareas más complejas, añadiendo su documentación relacionada.
- Implementación del algoritmo de control, a partir de los modelos de simulación creados en *Matlab-Simulink*.
- Desarrollo del software necesario para implementar un sistema de control dinámico y cinemático del robotparalelo.

La justificación de por qué se ha decidido trabajar en este proyecto final de carrera sobre la implementación de controladores usando el entorno Orocos es, sobre todo, el auge de la robótica y la programación en tiempo real.

Por otro lado, el entorno Orocos tiene como objeto permitir desarrollar controladores para robots y máquinas en un entorno modular, de software libre y de propósito general. La programación se realiza en lenguaje C++ usando las librerías Real-Time Toolkit, la de Kinematics and Dynamics, la Bayesian Filtering y la de Orocos Component. La ventaja de ser software libre es que todo el código es accesible, por lo que, en caso de querer adaptar el software para un propósito específico, no existen grandes restricciones.

Finalmente, cabe destacar la principal característica de la programación usando la librería Orocos, que es su programación modular, en la que cada módulo es totalmente independiente a los demás, de forma que, para dos robots iguales, no

basta más que duplicar ese componente para cada uno de los robots, y lanzarlo en cada una de las máquinas, teniendo una gran ventaja en lo que a la portabilidad se refiere.

2 Desarrollo teórico

2.1 Introducción a la robótica

La robótica es la ciencia encaminada a diseñar y construir aparatos y sistemas capaces de realizar tareas propias de un ser humano. La robótica es la ciencia y la tecnología de los robots. Se ocupa del diseño, manufactura y aplicaciones de los robots y combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control. Otras áreas importantes en robótica son el álgebra, los autómatas programables y las máquinas de estados.

2.1.1 Referencias históricas de la robótica

La historia de la robótica ha estado unida a la construcción de "artefactos", que trataban de materializar el deseo humano de crear seres a su semejanza y que lo descargasen del trabajo. Desde hace cientos de años antes de Cristo, ya se intentaban crear dispositivos, denominados artefactos o máquinas, que tuvieran un movimiento sin fin y que no estuvieran controlados ni supervisados por personas.

La historia nos dice que la fabricación de máquinas que imitan al ser humano o a otros seres vivos es incluso anterior a la era cristiana. De hecho, hace unos 4000 años, en el Antiguo Egipto existían estatuas de dioses o reyes que despedían fuego desde sus ojos, tal como laestatua de Osiris. Otros ingenios estaban dotados de brazos mecánicosque eran operados por los sacerdotes del templo.

Pero quizás el autómata que más llamó la atención de los antiguos fue la estatua de Memon de Etiopía, que era capaz de emitir sonidos cuando los rayos del sol la iluminaban, causando el lógico temor y el respeto entre la población. No había nada

de mágico en el autómata: al amanecer, el cambio de temperatura provocaba la evaporación del agua almacenada en su interior, y era ese vapor el que al escapar por fisuras muy bien calculadas de la estatua producía un sonido semejante al habla.

Sin embargo, los primeros datos descriptivos acerca de la construcción de un autómata aparecen en el siglo I. El matemático, físico e inventor griego Herón de Alejandría describe múltiples ingenios mecánicos en su libro *Los Autómatas*, por ejemplo aves que vuelan, gorjean y beben. Todos ellos fueron diseñados como juguetes, sin mayor interés por encontrarles una aplicación. Sin embargo, se describen algunos ingenios que sí presentaban una función útil, como es el caso de un molino de viento para accionar un órgano o un precursor de la turbina de vapor, conocida como Eolípila. Herón también construye un Odómetro, un instrumento dedicado a medir la distancia recorrida por un vehículo. El sistema utilizado era muy ingenioso y consistía en una transmisión que cada vez que daba una vuelta la rueda final caía una bola en un contenedor. Solo había que contar el número de bolas para conocer la distancia recorrida.

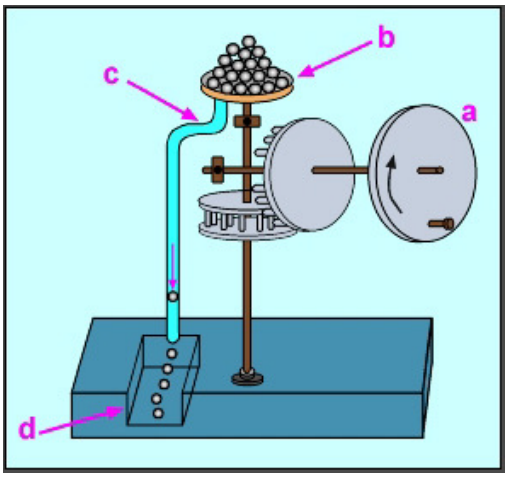

Figura 1. Odómetro de Herón.

El “motor” de estas maquinas antiguas era el movimiento del agua, la fuerza de gravedad o bien algún sistemas de palancas.

El libro antiguo sobre autómatas que más se destaca es sin duda “El libro del conocimiento de los ingeniosos mecanismos”, escrito por el árabe Al-Jazarien1260. En él se recopilaban muchos mecanismos ingeniosos de los siglos anteriores, a la vez que se exponían algunos creados por el autor.

Durante los siglos XV y XVI algunas de las figuras más relevantes del Renacimiento también realizaron sus propias aportaciones a la historia de los autómatas. El más famoso quizá sea el *León Mecánico*, desarrollado por Leonardo Da Vinci para el rey Luis XII de Francia, que abría su pecho con la garra y mostraba el escudo de armas del rey.

Siglo y medio más tarde, Jacques de Vaucanson construía una serie de autómatas que actuaban como flautistas ytamborileros, que resultó un gran éxito. Su último y más famoso mecanismo fue el llamado “Pato con Aparato Digestivo”, un dispositivo dotado de más de cuatrocientas piezas móviles capaz de batir las alas, comer, realizaba la digestión y defecar tal como lo hace un pato vivo.

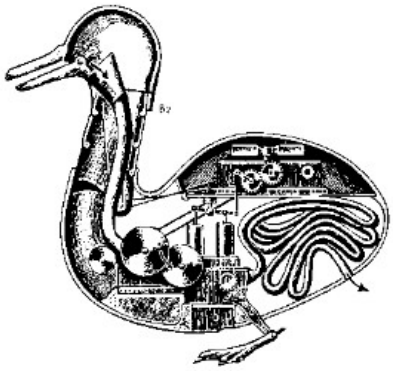

Figura 2: Pato con Aparato Digestivo

Una hazaña como esa, realizada con los escasos recursos técnicos de la época, parece casi imposible de superara. Pero durante el siglo XVIII el suizo PierreJaquet-Drozcreó obras que causaron impresión en cada lugar que fueron exhibidas. Tres de ellas, llamadas “La Pianista”, “El Dibujante” y “El Escritor” pueden verse en el Museo

de Arte e Historia de Neuchâtel, en Suiza. Las tres tienen más de 2000 piezas móviles, y “El Escritor” supera las 6000.

Figura 3: El “escritor” creado por Pierre Jaquet-Droz

A finales del siglo XVIII y principios del XIX, en plena Revolución Industrial, se desarrollaron diversos ingenios mecánicos utilizados fundamentalmente en la industria textil. Entre ellos se puede citar la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779), el telar mecánico de Cartwright (1785), el telar de Jacquard (1801), que constituyó con sus tarjetas perforadas los primeros precedentes históricos de las máquinas de control numérico. Más tarde se incorporaron los automatismos en las industrias mineras y metalúrgicas. Pero sin duda, el automatismo que causó mayor impacto por tener un papel fundamental en la Revolución Industrial lo realiza Potter a principios del siglo XVIII, automatizando el funcionamiento de las válvulas en la máquina de vapor atmosférica, creada por el inventor inglés Thomas Newcomen.

La palabra robot se empleó por primera vez en 1920 en una obra de teatro llamada "R.U.R." o "Los Robots Universales de Rossum" escrita por el dramaturgo checo KarelCapek. La trama era sencilla: el hombre fabrica un robot luego el robot

mata al hombre. Muchas películas han seguido mostrando a los robots como máquinas dañinas y amenazadoras. La palabra checa 'Robota' significa servidumbre o trabajador forzado, y cuando se tradujo al inglés se convirtió en el término robot. Entre los escritores de ciencia ficción, Isaac Asimov contribuyó con varias narraciones relativas a robots, comenzó en 1939, a él se atribuye el término Robótica. La imagen de robot que aparece en su obra es el de una máquina bien diseñada y con una seguridad garantizada que actúa de acuerdo con tres principios. Estos principios fueron denominados por Asimov las Tres Leyes de la Robótica, y son:

1.- Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.

2.- Un robot debe de obedecer las órdenes dadas por los seres humanos, salvo que estén en conflictos con la primera ley.

3.- Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.

Consecuentemente todos los robots de Asimov son fieles sirvientes del ser humano, de ésta forma su actitud contraviene a la de Kapek. Inicialmente, se definía un robot como un manipulador reprogramable y multifuncional diseñado para trasladar materiales, piezas, herramientas o aparatos a través de una serie de movimientos programados para llevar a cabo una variedad de tareas.

El desarrollo en la tecnología, donde se incluyen las poderosas computadoras electrónicas, los actuadores de control retroalimentados, transmisión de potencia a través de engranes, y la tecnología en sensores han contribuido a flexibilizar los mecanismos autómatas para desempeñar tareas dentro de la industria.

Las primeras patentes aparecieron en 1946 con los muy primitivos robots para traslado de maquinaria de Devol. También en ese año aparecen las primeras computadoras: J. PresperEckert y John Maulchy construyeron el ENAC en la Universidad de Pensilvania y la primera máquina digital de propósito general se desarrolla en el MIT. En 1954, Devol diseña el primer robot programable y dándole el

término "autómata universal", que posteriormente recorta a Unimation. Así llamaría Engleberger a la primera compañía de robótica.

En 1964 se abren laboratorios de investigación en inteligencia artificial en el MIT, el SRI (Stanford ResearchInstitute) y en la universidad de Edimburgo. Poco después los japoneses que anteriormente importaban su tecnología robótica, se situaron como pioneros del mercado.

Otros desarrollos Importantes en la historia de la robótica fueron:

En 1960 se introdujo el primer robot "Unimate''. Utilizan los principios de control numérico para el control de manipulador y era un robot de transmisión hidráulica.

En 1961 Un robot Unimate se instaló en la Ford Motors Company para atender una máquina de fundición de troquel.

En 1966 Trallfa, una firma noruega, construyó e instaló un robot de pintura por pulverización.

En 1971 El "StandfordArm'', un pequeño brazo de robot de accionamiento eléctrico, se desarrolló en la StandfordUniversity.

En 1973 Se desarrolló en SRI el primer lenguaje de programación de robots del tipo de computadora para la investigación con la denominación WAVE. Fue seguido por el lenguaje AL en 1974. Los dos lenguajes se desarrollaron posteriormente en el lenguaje VAL comercial para Unimation por Víctor Scheinman y Bruce Simano.

En 1978 Se introdujo el robot PUMA (Programmable Universal Machine forAssambly) para tareas de montaje por Unimation, basándose en diseños obtenidos en un estudio de la General Motors.

Figura 4: Robot PUMA Unimate

En 1980 Un sistema robótico de captación de recipientes fue objeto de demostración en la Universidad de Rhode Island. Con el empleo de visión de máquina el sistema era capaz de captar piezas en orientaciones aleatorias y posiciones fuera de un recipiente.

En los setenta, la NASA inicio un programa de cooperación con el Jet Propulsión Laboratory para desarrollar plataformas capaces de explorar terrenos hostiles. El primer fruto de esta alianza seria el MARS-ROVER, que estaba equipado con un brazo mecánico tipo STANFORD, un dispositivo telemétrico láser, cámaras estéreo y sensores de proximidad.

En los ochenta aparece el CART del SRI que trabaja con procesado de imagen estéreo, más una cámara adicional acoplada en su parte superior. También en la década de los ochenta, el CMU-ROVER de la Universidad Carnegie Mellon incorporaba por primera vez una rueda timón, lo que permite cualquier posición y orientación del plano.

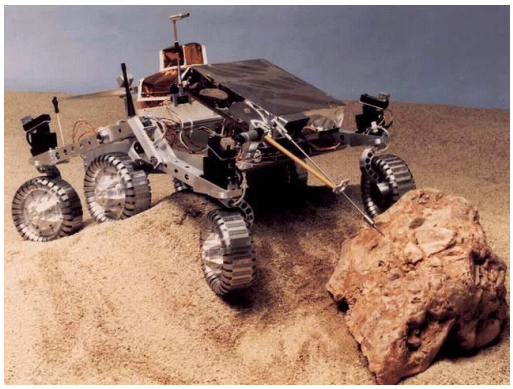

Figura 5: Mars-Rover

En la actualidad, la robótica se debate entre modelos sumamente ambiciosos, como es el caso del IT, diseñado para expresar emociones, el COG, también conocido como el robot de cuatro sentidos, el famoso SOUJOURNER o el LUNAR ROVER, vehículo de turismo con control remotos, y otros mucho más específicos como el CYPHER, un helicóptero robot de uso militar, el guardia de tráfico japonés ANZEN TARO o los robots mascotas de Sony.

En general, la historia de la robótica la podemos clasificar en cinco generaciones (división hecha por Michael Cancel, director del Centro de Aplicaciones Robóticas de ScienceApplication Inc. En 1984). Las dos primeras, ya alcanzadas en los ochenta, incluían la gestión de tareas repetitivas con autonomía muy limitada. La tercera generación incluiría visión artificial, en lo cual se ha avanzado mucho en los ochenta y noventas. La cuarta incluye movilidad avanzada en exteriores e interiores y la quinta entraría en el dominio de la inteligencia artificial en lo cual se está trabajando actualmente.

En los últimos años, los robots han tomado posición en todas las áreas productivas industriales. La incorporación del robot al proceso productivo ha representado uno de los avances más espectaculares de la edad moderna. En poco más de cuarenta años, hemos pasado de aquellos primeros modelos, rudos y limitados, a sofisticadas máquinas capaces de sustituir al hombre en todo tipo de tareas repetitivas o peligrosas, y además, hacerlo de forma más rápida, precisa y económica que el ser humano. Hoy en día, se calcula que el número de robots

industriales instalados en el mundo es de un millón de unidades, unos 20.000 en España, siendo Japón el país más tecnológicamente avanzado, con una media de 322 robots por cada 10.000 trabajadores.

2.1.2 Introducción a la historia de la automatización

La historia de la automatización industrial está caracterizada por sus constantes innovaciones tecnológicas. Esto se debe a que las técnicas de automatización están muy ligadas a los sucesos económicos mundiales.

Aunque el crecimiento del mercado de la industria Robótica actual sigue siendo lento en comparación con los primeros años de la década de los 80´s, de acuerdo a algunas predicciones, la industria de la robótica está todavía en su infancia. El uso de robots industriales junto con los sistemas de diseño asistidos por computador (CAD), y los sistemas de fabricación asistidos por computador (CAM), son la última tendencia en automatización de procesos de fabricación. Estas tecnologías son el inicio de una nueva etapa en la automatización industrial, de alcances aún desconocidos.

En la actualidad el uso de los robots industriales está concentrado en operaciones muy simples, como tareas repetitivas que no requieren tanta precisión. Los análisis de mercado en cuanto a fabricación predicen que en ésta década y en las posteriores los robots industriales incrementarán su campo de aplicación, debido a los avances tecnológicos, como la incorporación de la inteligencia artificial o nuevas extensiones sensoriales, los cuales permitirán tareas más sofisticadas en la automatización industrial.

Como se ha explicado anteriormente, la automatización y la robótica son dos tecnologías estrechamente relacionadas. En un contexto industrial, se puede definir la automatización como una tecnología que está relacionada con el empleo de sistemas electro-mecánicos basados en computadores para la operación y control de la producción.

Existen tres clases de automatización industrial: *la automatización fija, la automatización programable, y la automatización flexible*.

La automatización fija se utiliza cuando se necesita una gran productividad y eficiencia debido al gran volumen de producción. El principal inconveniente es su alto coste económico de equipo especializado capaz de procesar el producto, con un alto rendimiento y tasas de producción elevadas. Otro inconveniente de la automatización fija es su ciclo de vida, ya que está ligado a la vigencia del producto en el mercado.

La automatización programable se emplea cuando el volumen de producción es relativamente bajo y hay una diversidad de producción a obtener. En este caso el equipo de producción es diseñado para adaptarse a las variaciones de configuración del producto. Esta adaptación se realiza generalmente por medio de un programa software.

La automatización flexible es la más adecuada para un rango de producción medio. Estos sistemas flexibles combinan características de la automatización fija y de la automatización programada. Los sistemas flexibles suelen estar constituidos por una serie de estaciones de trabajo conectadas entre sí por sistemas de almacenamiento y manipulación de materiales, controlados por un ordenador.

2.1.3 Definición y clasificación de robots industriales

Existen diversas definiciones formales del término robot industrial, atendiendo principalmente a las diferencias conceptuales surgidas entre el mercado japonés y el euro-americano a finales del siglo XX. La principal discrepancia radica en la diferencia entre un robot y un manipulador.Mientras que para los japoneses un robot industrial es cualquier dispositivo mecánico dotado de articulaciones móviles destinado a la manipulación, el mercado occidental es más restrictivo exigiendo una mayor complejidad, sobre todo en lo relativo al control.

La definición del *RoboticsInstitute of América (RIA),* define al robot industrial como:

"Un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas."

Sin embargo, la *Asociación Japonesa de Robótica Industrial (JIRA)* lo define como:

"Dispositivos capaces de moverse de modo flexible análogo al que poseen los organismos vivos, con o sin funciones intelectuales, permitiendo operaciones en respuesta a las órdenes humanas."

Como se puede observar en las definiciones anteriores, la definición japonesa es más amplia y genérica mientras que la americana es más concreta. Por ejemplo, una máquina automática que no es programable se considera un robot en Japón, sin embargo no sería considerado un robot según la definición americana. No obstante, la definición más aceptada internacionalmente es la americana.

Por último, la *Federación Internacional de Robótica (IFR)* distingue entre robot industrial de manipulación y otros robots:

"Por robot industrial de manipulación se entiende una maquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento."

Un resumen de todas las definiciones anteriores es la aceptación del robot industrial como un brazo mecánico con capacidad de manipulación, que incorpora un control complejo para llevar a cabo la ejecución de movimientos y diversas tareas, además de la posibilidad de interactuar con su entorno. Un sistema robotizado, en cambio, es un concepto más amplio. Engloba todos aquellos dispositivos que realizan tareas de manera automática en sustitución de un ser humano y que pueden incorporar uno o varios robots, siendo esto último lo más frecuente.

Estas definiciones mencionadas anteriormente, no llegan a abarcar todas las posibilidades de aplicaciones presentes y futuras de los robots. La particularidad de su arquitectura y sus objetivos enfocados al sistema productivo convierten al robot industrial en una nueva y revolucionaria concepción del sistema productivo.

La creciente utilización de robots industriales en el proceso productivo, ha dado lugar al desarrollo de controladores industriales rápidos y potentes, basados en microprocesadores, así como un empleo de servos en bucle cerrado que permiten establecer con exactitud la posición real de los elementos del robot y su desviación o error. Esta evolución ha dado origen a una serie de tipos de robots, que se citan a continuación:

- **Manipuladores**

Son sistemas mecánicos multifuncionales, con un sencillo sistema de control, que permite gobernar el movimiento de sus elementos de los siguientes modos:

- Manual: Cuando el operario controla directamente la tarea del manipulador.
- De secuencia fija: cuando se repite, de forma invariable, el proceso de trabajo preparado previamente.
- De secuencia variable: Se pueden alterar algunas características de los ciclos de trabajo

Existen muchas operaciones básicas que pueden ser realizadas de forma óptima mediante manipuladores. Por ello, estos dispositivos son utilizados generalmente cuando las funciones de trabajo son sencillas y repetitivas.

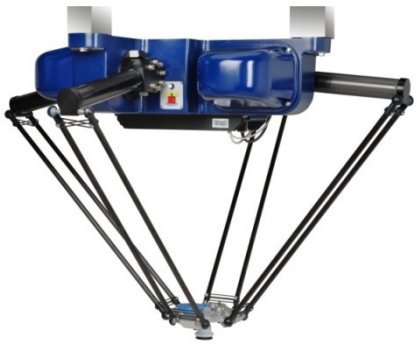

Figura 6. Ejemplo de un robot móvil manipulador

- **Robots de repetición o aprendizaje**

Son manipuladores que se limitan a repetir una secuencia de movimientos, previamente ejecutada por un operador humano, haciendo uso de un controlador manual o un dispositivo auxiliar. En este tipo de robots, el operario durante la fase de enseñanza se vale de una pistola de programación con diversos pulsadores o teclas, o bien de joysticks, o bien utiliza un maniquí, o desplaza directamente la mano del robot. Actualmente, los robots de aprendizaje son los más conocidos en algunos sectores de la industria (como puede ser en un taller de pintura), y el tipo de programación que incorporan recibe el nombre de "*gestual*".

Figura 7. Robot de aprendizaje

- **Robots con control por computador**

Son manipuladores o sistemas mecánicos multifuncionales, controlados por un computador, que habitualmente suele ser un microcomputador. El control por computador dispone de un lenguaje específico de programación, compuesto por varias instrucciones adaptadas al hardware del robot, con las que se puede diseñar un programa de aplicación utilizando sólo el ordenador. A esta programación se le denomina "*textual*" y se crea sin la intervención del manipulador.

Las grandes ventajas que ofrece este tipo de robots, hacen que se vayan imponiendo en el mercado rápidamente, lo que exige la preparación urgente de personal cualificado, capaz de desarrollar programas de control que permitan el manejo del robot.

Figura 7. Robot FANUC

- **Robots inteligentes**

Son similares a los del grupo anterior, pero tienen la capacidad de poder relacionarse con el mundo que les rodea a través de sensores y de tomar decisiones en función de la información obtenida en tiempo real. De momento, son muy poco conocidos en el mercado y se encuentran en fase experimental, donde grupos de investigadores se esfuerzan por hacerlos más efectivos, al mismo tiempo que más económicamente asequibles. El reconocimiento de imágenes y algunas técnicas de inteligencia artificial son los campos que más se están estudiando para su posible aplicación en estos robots.

Figura 8. Robot ASIMO de Honda

2.1.4 Estructuras y componentes de un robot

En esta sección, se presentará la estructura general de un robot, así como sus componentes básicos, de forma que en apartados posteriores se entienda la utilidad de cada uno de ellos.

De forma general, la estructura de un robot se puede dividir en tres unidades funcionales que permiten al usuario influir en el entorno de trabajo del robot:

- **Unidad de programación:**

La unidad de programación se utiliza para programar y operar con el robot. A través de la unidad de programación, los resultados de los programas ejecutados y los comandos, pueden ser analizados. Normalmente suele consistir en un teclado industrial, una pantalla y un control remoto.

- **Sistema de control:**

El sistema de control recibe los comandos y las instrucciones de la unidad de programación. Después de interpretar los comandos e instrucciones, su tarea es generar las acciones de control adecuadas. Las acciones de control deben manejar correctamente el sistema mecánico, dependiendo de su estado, con el objetivo de evitar errores. Además, el sistema de control informará al usuario sobre el resultado de las acciones de control a través de la unidad de programación. El sistema de control

consiste típicamente de un equipo industrial, que incluye placas de circuito impreso para la comunicación con el robot mecánico, herramientas y otros dispositivos.

- **Mecánica del robot:**

La mecánica del robotconvierte las acciones de control en movimientos del robot, con el fin de interaccionar con el medio y realizar una actividad determinada. Además, los dos componentes mecánicosinforman periódicamente al sistema de control sobre el estado del robot. La mecánica del robot se compone de varios elementos individuales que están conectados por articulaciones, produciéndose un movimiento de éstas mediante motores eléctricos, hidráulicos o actuadores neumáticos. Por otro lado, los sensores, tales como potenciómetros o encóders, permiten conocer en cada instante la posición de una articulación.

En la siguiente figura se puede apreciar la estructura general del robot y las diferentes interacciones entre los diversos componentes.

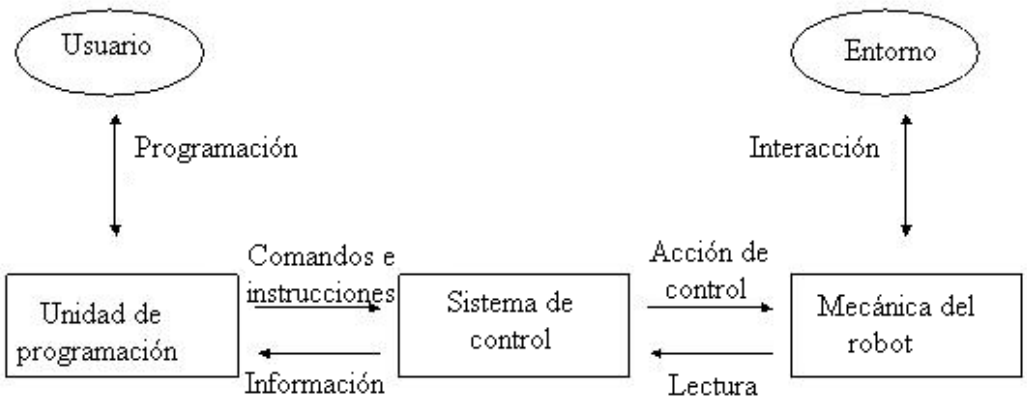

Figura 9: Estructura funcional de un robot

Los cuatro componentes básicos del robot son:

• Sistema Control

• Actuadores

• Articulaciones

• Sensores

Mediante los componentes básicos del robot, el comportamiento del mismo será diferente en un caso u otro, dependiendo del tipo y la colocación (dependiendo también de la tarea que deba desarrollar).

Como se ha comentado anteriormente, el sistema de control consiste en un equipo industrial y los elementos adicionalessiguientes:

- Convertidor D/Apara proporcionar lasacciones de control.
- Transformadores para las acciones de control (de cara a aplicárselas a robot).
- Tarjetas de adquisición de datospara obtener información de lossensores.
- Tarjetas de comunicación (serie, paralelo, Ethernet).

Las principales funciones del sistema de control son:

- Ser capaz de controlar la interfaz de comunicación con el usuario, así como interpretar comandos enviados por el usuario, tales como activaciones de las salidas digitales o analógicas, o alguna situación excepcional como una parada de emergencia. Los comandos correspondientes son interpretados y ejecutados de inmediato. El usuario, además puede introducir los programas en la unidad de programación, que serán interpretados y ejecutados por el sistema de control.
- Generar y controlar los movimientos del robot mediante la creación de una señal de referencia que dirige los actuadores del robot. Durante todo el proceso, la referencia se compara con la producción obtenida por los sensores. El sistema de control debe intentar de reducir la diferencia entre la referencia y la salida (es decir, el error) por la aplicación de un algoritmo de control.

El movimiento de las articulaciones del robot es producido por los actuadores que actúan de acuerdo a las acciones de control que genera el sistema de control. Se puede distinguir entre tres tipos de actuadores de acuerdo con su fuente de energía:

- Actuadores neumáticos
- Actuadores hidráulicos
- Los motores eléctricos

La fuente de energía de actuadores neumáticos es de aire comprimido, por lo tanto requieren de compresores externos que puedan almacenar el aire a una presión determinada. Debido al hecho de que el aire es muy compresible, es difícil lograr el control completo del tipo todo o nada para las articulaciones prismáticas (totalmente extendida o retraída). Además, la precisión de los actuadores neumáticos sólo es aceptable para cargas pequeñas. Por otro lado, los actuadores neumáticos permiten la aplicación de grandes aceleraciones y son bastante económicos.

Figura 10: Actuador neumático

Los actuadores hidráulicos tienden a tener la misma configuración que los actuadores neumáticos, pero con la diferencia que utilizan líquidos (aceites de baja compresión) como fuentes de energía. Ofrecen más potencia, por lo tanto se utilizan en los robots (como grúas industriales), mejorando su rendimiento y precisión. Los inconvenientes de estos actuadores son su instalación complicada y de alto costo. Sin embargo, hoy en día son los más utilizados en robots que deben manipular cargas pesadas (varias toneladas) o trabajar en entornos difíciles.

Figura 11: Actuador hidráulico

En la actualidad, los robots industriales, tienden a usar motores eléctricos como actuadores debido a su mejor precisión y rendimiento. Podemos distinguir entre cuatro tipos:

• Los motores paso a paso

• Motores de corriente continua (DC)

• Servomotores

• Los motores de corriente alterna (AC)

En resumen, un motor paso a pasoes un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa es que es capaz de avanzar una serie de grados (paso), en cualquiera de los dos sentidos, dependiendo de sus entradas de control.

Las principales características de los motores paso a paso son:

• Precisión buena, aunque problemático a bajas velocidades.

• Suministro de baja potencia

Debido a la baja potencia, los motores paso a paso se utilizan poco en la robótica industrial, aplicándose generalmente en los dispositivos periféricos del robot, en los movimientos del eje de herramientas, etc.

Figura 12: Motor paso a paso

El principio básico de motores de corriente continua se basa en la inducción. El rotor está formado por un núcleo de hierro dulce que está "envuelto" por un campo magnético estático. Generalmente, en este tipo de motores, la conmutación se lleva a cabo por las escobillas. Uno de los principales problemas es que, el uso de escobillas implica su limpieza, reparación, reemplazo, etc., por lo que eso influye en el coste. Los motores*brushless* no implican estos problemas debido a la ausencia de escobillas, de forma que las conmutaciones se llevan a cabomediante la activación o desactivación de ciertos transistores en un determinado instante. La velocidad de rotación de un motor de corriente continua se controla por el nivel de tensión o la intensidad de la emoción, proporcionando una buena precisión y control. Durante muchos años, fue el tipo de motor más utilizado, aunquehoy en día los motores de corriente alterna son más populares.

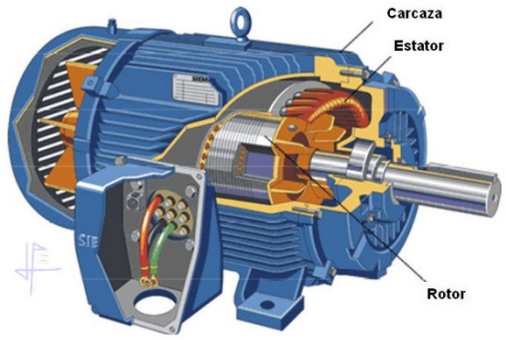

Figura 13: Motor corriente continua

Un servomotor es, realmente, un motor eléctrico "inteligente". Se puede determinar su posición actual y recibir una posición deseada. Una vez que la posición deseada se sabe, el motor intenta mover su eje a la posición deseada girando hacia la izquierda o derecha. Salvo que se modifique, los servomotores no son capaces de girar continuamente, ya que en cada instante está comprobando si está en la posición solicitada, no pudiéndose utilizar, por ejemplo, para ruedas motrices. Si se necesita un posicionamiento de precisión, un servomotor se puede utilizar, por ejemplo, (dependiendo de la función que deba desempeñar) en las patas del robot. Los sistemas con servospueden ser poderosos, ya que, de una forma remota, se podrían mover

cargas pesadas a una posición deseada con gran precisión. Pese a su precisión, los precios son muy económicos.

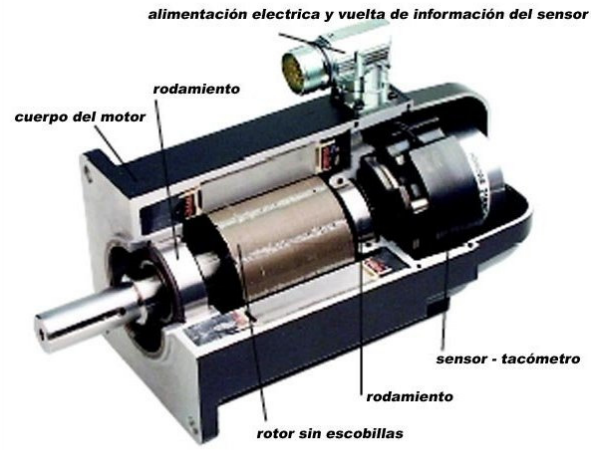

Figura 14: Servomotor

El rotor de un motor de corriente alterna se compone de imanes permanentes. Su estator consta de tres bobinas igualmente distribuidas en 120 ° que se alimenta por un sistema de tres fases de tensión. Puesto que los motores de CA ofrecen una potencia mejor en relación con el peso de los motores DC son, en la actualidad, los motores más comunes en el mundo de la robótica.

Figura 15: Motor corriente alterna

Como se mencionó anteriormente, la mecánica del robot se compone de varios elementos individuales que están conectados por articulaciones que,a través de ellas,

cada elemento se mueve con respecto a los elementos. El movimiento de una articulación puede ser de traslación, rotación o una combinación de los dos. Las articulaciones más comunes son:

- Cilíndricas.
- De revolución.
- Prismáticas.
- Esféricas.

La mayoría de los robots industriales contienen sólo las articulaciones de revolución y prismáticas por el hecho de que son simples, económicas y robustas.

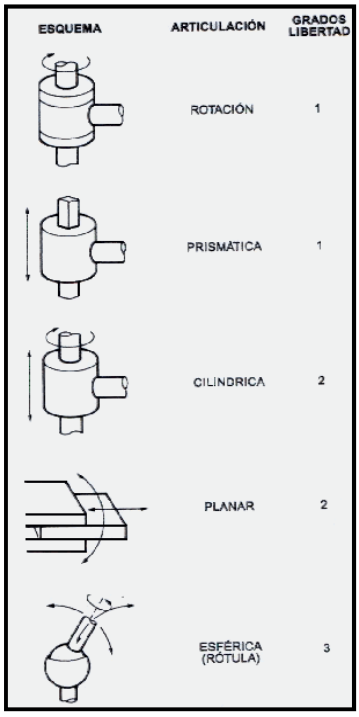

Figura 15: Tipos de articulaciones

La medición de las posiciones de las articulaciones se lleva a cabo por los sensores. A continuación se explicarán algunos de los sensores más utilizados:

• Encóder

Un encóder es un dispositivo mecánico que puede controlar el movimiento o la posición. Un encóder convencional usa sensores ópticos para proporcionar una serie

de pulsos que se puede traducir (en el sistema de control) en movimiento, posición o dirección. Los dos tipos básicos de encóders son los incrementales y los absolutos.

La figura 16muestra el modo de funcionamiento de un codificador incremental rotativo. Un disco delgado, con ranura (verfigura 17) está conectado a un eje del motor, cuya posición se está detectando. La luz de una fuente de luz estacionaria está continuamente enfocada en el disco. En el otro lado del disco está el fotodetector para detectar la luz de la fuente de luz, el cuál emite un pulso cada vez que detecta luz. Losencóders rotativos incrementales con dos sensores que están fuera de fase con el fin de indicar la dirección de rotación (pudiendo obtener si es una cuenta positiva o negativa de pulsos). Dado que el codificador incremental sólo proporciona una serie de pulsos, es necesario indicar un punto base o de referencia (con un reset antes de la ejecución, por ejemplo), en el que empiece a hacer la cuenta de pulsos. Losencóders rotativos incrementales son baratos y simples, dependiendo de la resolución del tamaño de los discos.

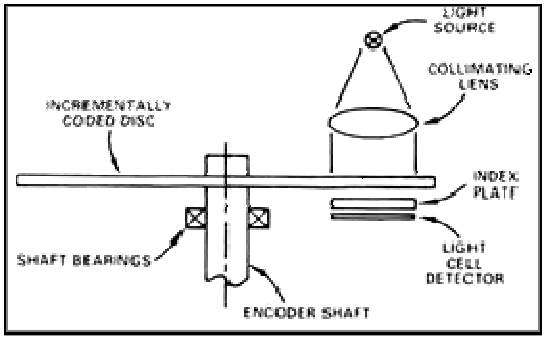

Figura 16: Funcionamiento encóder incremental

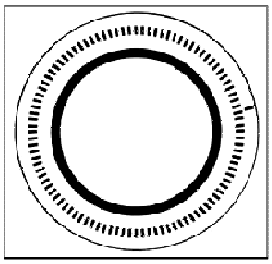

Figura 17: Encóder incremental

Un encóder absoluto está diseñado de tal manera que siempre sabrá su posición actual, generando un código binario completo para cada posición. La figura 18 muestra un ejemplo de una rueda del encóder absoluto. Por cada anillo de la rueda del encóder, existe una fuente de luz y un receptor. En la figura 19, se puede apreciar la configuración del encóder. Puesto que el codificador absoluto produce sólo un código binario distinto (patrón de bits) para cada posición, conoce exactamente dónde está entre los dos extremos de su recorrido. Este tipo de encódersse utilizan en aplicaciones donde la información de posición es más importante que un cambio de posición.

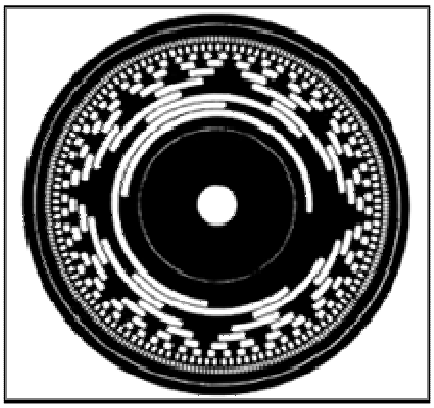

Figura 18: Encóder absoluto

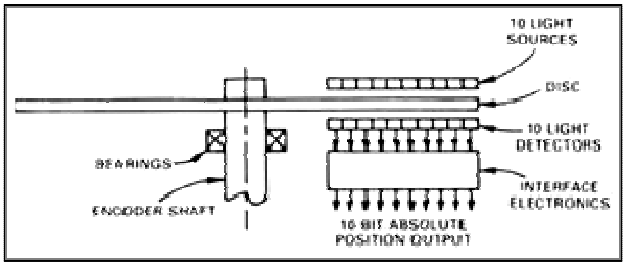

Figura 19: Funcionamiento encóder absoluto

2.1.5 El robot paralelo

Dado que el objetivo de este proyecto está relacionado con la implementación de controladores para un robot paralelo, a continuación se explicarán los conceptos más importantes y generales de un robot de este tipo (basado en [1] y [2]).

Los robots paralelos son robots articulados que utilizan mecanismos similares para el movimiento en cualquiera de sus miembros. Su distinción "paralela" es porque el extremo final del miembro de robot está conectado a su base por una serie de (por lo general tres o seis) miembros independientes, con la particularidad de que trabajan en paralelo. Por ello,“paralelo” se utiliza aquí en el sentido topológico, en lugar de la geometría, puesto que los miembros pueden trabajar juntos, y no tienen por qué estar alineados en forma de líneas paralelas.

Normalmente, consiste en una plataforma móvil que se conecta a una base fija por varios miembros o patas. Por lo general, el número de patas es igual al número de grados de libertad que implica que cada miembro está controlado por un actuador.

En la actualidad, y desde hace 20 años, la investigación sobre robots paralelos está creciendo, ya que este tipo de robots presenta ventajas, frente a los robots en serie, como las siguientes:

- Mayor capacidad de carga debido a que la carga externa se distribuye en las diversas patas del robot y, por lo tanto, es compartida por los actuadores.
- La relación entre la lectura de sensores y el valor de posicionamiento es pequeño, lo que permite una muy buena precisión.
- La cinemática inversa es relativamente sencilla, lo que permite la simulación de modelos, para posteriormente aplicarlo al modelo real.
- Una buena rigidez.

Por otro lado, las desventajas de este tipo de robots son:

- Espacio de trabajo bastante limitado.
- Problemas a la hora de planificar su control.

Finalmente, los robots paralelos se pueden clasificar como planos, esféricos o espaciales de acuerdo a sus características de movimiento, pudiéndolos encontrar en aplicaciones tales como simuladores de aviones (Stewart, 1965) o máquinas andantes (Waldron et al., 1991), entre otras muchas más actividades.

2.1.6Cinemática en robots

La cinemática es la ciencia del movimiento que trata a éste sin importarle las fuerzas que lo causan. Dentro de la cinemática se estudia la posición, la velocidad, aceleración y todas las derivadas de las variables de posición de mayor orden con respecto al tiempo o cualquier otra variable. El estudio de la cinemática de los manipuladores se refiere a todas las propiedades geométricas basadas en el tiempo del movimiento.

Los robots consisten en un conjunto de eslabones conectados mediante articulaciones que permiten el movimiento relativo entre los eslabones vecinos. El número de grados de libertad que un robot posee es el número de variables de posición independientes que deberían ser especificadas para localizar todas las partes del mecanismo. En el caso de los robots industriales el número de grados de libertad suele equivaler al número de articulaciones siempre y cuando cada articulación tenga un solo grado de libertad.

2.1.6.1 Modelo Cinemático Directo

El modelo cinemático directo es el problema geométrico que calcula la posición y orientación del efector final del robot. Dados una serie de ángulos entre las articulaciones, el problema cinemático directo calcula la posición y orientación del marco de referencia del efector final con respecto al marco de la base.

2.1.6.2 Modelo Cinemático Inverso

Dada la posición y orientación del efector final del robot, el problema cinemático inverso consiste en calcular todos los posibles conjuntos de ángulos entre las articulaciones que podrían usarse para obtener la posición y orientación deseada.

El problema cinemático inverso es más complicado que la cinemática directa ya que las ecuaciones no son lineales, sus soluciones no son siempre fáciles o incluso posibles en una forma cerrada. También surge la existencia de una o de diversas soluciones. La existencia o no de la solución lo define el espacio de trabajo de un robot

dado. La ausencia de una solución significa que el robot no puede alcanzar la posición y orientación deseada porque se encuentra fuera del espacio de trabajo del robot o fuera de los rangos permisibles de cada una de sus articulaciones.

2.1.7 Control Dinámico de robots

La dinámica se ocupa de la relación entre las fuerzas que actúan sobre un cuerpo y el movimiento que en él se origina. Por lo tanto, el modelo dinámico de un robot tiene por objeto conocer la relación entre el movimiento del robot y las fuerzas implicadas en el mismo.

Esta relación se obtiene mediante el denominado modelo dinámico, que relaciona matemáticamente:

- La localización del robot definida por sus variables articulares o por las coordenadas de localización de su extremo, y sus derivadas: velocidad y aceleración.
- Las fuerzas pares aplicados en las articulaciones (o en el extremo del robot).
- Los parámetros dimensionales del robot, como longitud, masa e inercias de sus elementos.

La obtención de este modelo para mecanismos de uno o dos grados de libertad no es excesivamente compleja, pero a medida que el número de grados de libertad aumenta, el planteamiento y obtención del modelo se complica enormemente. Por este motivo no siempre es posible obtener un modelo dinámico expresado de una forma cerrada, esto es, mediante una serie de ecuaciones, normalmente del tipo diferencial de segundo orden, cuya integración permita conocer que el movimiento surge al aplicar unas fuerzas o que fuerzas hay que aplicar para obtener un movimiento determinado.

El modelo dinámico debe ser resuelto entonces de manera iterativa mediante la utilización de un procedimiento numérico.

El problema de la obtención del modelo dinámico de un robot es, por lo tanto, uno de los aspectos más complejos de la robótica, lo que ha llevado a ser obviado en numerosas ocasiones. Sin embargo, el modelo dinámico es imprescindible para conseguir los siguientes fines:

- Simulación del movimiento del robot.
- Diseño y evaluación de la estructura mecánica del robot.
- Dimensionamiento de los actuadores.
- Diseño y evaluación del control dinámico del robot.

Este último fin es, evidentemente, de gran importancia, pues de la calidad del control dinámico del robot depende la precisión y velocidad de sus movimientos. La gran complejidad, ya comentada, existente en la obtención del modelo dinámico del robot, ha motivado que se realicen ciertas simplificaciones, de manera que así pueda ser utilizado en el diseño del controlador.

Es importante hacer notar que el modelo dinámico completo de un robot debe incluir no solo la dinámica de sus elementos (barras o eslabones) sino también la propia de sus sistemas de transmisión y de los actuadores. Estos elementos incorporan al modelo dinámico nuevas inercias, rozamientos, saturaciones de los circuitos electrónicos, etc. aumentando aun más su complejidad.

Por último, es preciso señalar que si bien, en la mayor parte de las aplicaciones reales de robótica, las cargas e inercias manejadas no son suficientes como para originar deformaciones en los eslabones del robot, en determinadas ocasiones no ocurre así, siendo preciso considerar al robot como un conjunto de eslabones no rígidos. Aplicaciones de este tipo pueden encontrarse en la robótica espacial o en robots de grandes dimensiones.

2.2 Middleware de tiempo Real

El Middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o

sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos. De esta forma se provee una solución que mejora la calidad de servicio, seguridad, envío de mensajes, directorio de servicio, etc.

Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).El middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema que se tenga que resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware. En la siguiente ilustración se puede observar lo expuesto anteriormente.

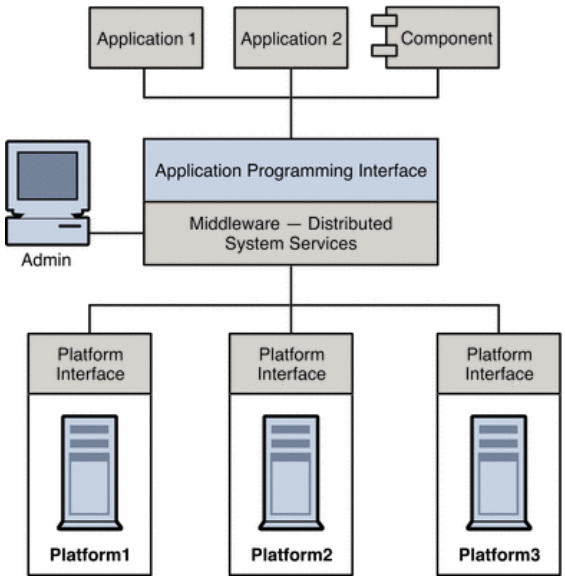

Figura 20: Middleware

Dependiendo de la categoría de integración, existen, por ejemplo, middlewares orientados a componentes (como los usados en Orocos), cuya principal característica es que “este middleware es configurable y reconfigurable. La reconfiguración se puede

realizar en tiempo de ejecución, lo que ofrece una gran flexibilidad para satisfacer las necesidades de un gran número de aplicaciones".

Por otro lado, dependiendo de la aplicación específica que se vaya a realizar nos encontramos con el middleware de tiempo real (como el que nos centramos en este proyecto: Orocos), que soporta las peticiones sensibles al tiempo y políticas de planificación. Las grandes ventajas de este tipo de middleware son que proveen un proceso de decisión que determina el mejor criterio para resolver procesos sensibles al tiempo y poder ayudar a los sistemas operantes en la localización de recursos cuando se tienen tiempos límites de operación.

Si nos detenemos a analizar las características del middleware orientado a componentes y el middleware de tiempo real, nos damos cuenta que es exactamente lo que nos ofrece el proyecto Orocos. De ahí la importancia en todo el campo a la robótica (sobre todo en la rama industrial, en la que un proceso debe estar siendo controlado en todo momento).

2.3 OROCOS

En este apartado se comentará todo lo relacionado con la librería gratuita de tiempo real en la que nos hemos apoyado para realizar el proyecto: OROCOS ("Open RObot COntrol Software"). Puesto que el objetivo, además de llegar a controlar el prototipo de robot paralelo, era aprender otro modo distinto de programación, se han realizado diversas pruebas para consolidar conceptos y comprobar que los pasos que se iban dando, eran seguros y en la dirección correcta.

Ya que el novedoso entorno Orocos está compuesto de diferentes funcionalidades, se comentará el potencial de cada una de ellas, pese a que en nuestro proyecto no se han utilizado todas de las diversas opciones que nos ofrece.

2.3.1 Historia y proyecto

La idea de iniciar un proyecto de Software Libre para el control del robot nació en diciembre de 2000, motivada por más de dos décadas de experiencias más bien decepcionantes, y fracasos al tratar de utilizar software comercial de control de robots para la investigación robótica avanzada. La idea, junto con un borrador de una posible propuesta de proyecto, se puso en marcha en la lista de correo de EURON, la Red Europea de Robótica. Este correo electrónico dio lugar a una gran cantidad de respuestas, a pesar de que fue enviado durante el período de Navidad. Dentro de unas dos semanas, una propuesta fue preparada y enviada a la Unión Europea. Los contactos con el responsable de temas de software en la Unión Europea dejó claro que el tamaño del proyecto tenía que ser muy modesto, por lo que sólo se seleccionaron tres socios o responsables: Universidad Católica de Lovaina en Bélgica, LAAS Toulouse en Francia y la KTH de Estocolmo, en Suecia. Cada uno de estos tres grupos recibió únicamente la ayuda de una sola persona durante todo un año. El proyecto patrocinado por la UE se inició en septiembre de 2001, y tuvo una duración de dos años.

La patrocinación de la Unión Europea también proporcionó algunas ayudas, con el objetivo de invitar a reuniones del proyecto Orocos, a diferentes personas que no estaban implicados en el proyecto. Esto, junto con las herramientas clásicas de una página web y una lista de correo, generaron diversos debates e intercambio de ideas.

Una primera versión de lo que sería el núcleo en tiempo real del proyecto Orocos fue lanzado en el verano de 2002, pero fue muy preliminar y difícil de usar. En noviembre de 2002, la primera versión fue lanzada con la que se podía controlar la posición y velocidad de un robot manipulador con seis grados de libertad.

Después de que el proyecto patrocinado por la UE hubiera terminado, los socios del proyecto continuaron mejorando el software ya entregado. Tal fue el avance, que incluso el marco de desarrollo de tiempo real en Orocos de la KU Leuven, había conseguido lanzar 7 versiones importantes sobre septiembre de 2003. Por otro lado,en la KTH de Estocolmo, se realizaron diversas versiones basadas en los componentes más comunes en el marco de la robótica.

Debido a su aplicabilidad a las aplicaciones industriales, el marco Orocos en tiempo real se ha instaurado más en el campo de control de la máquina, y no en lo que es en sí la robótica, dejado atrás sus inicios en la robótica. La modularidad de los paquetes Orocos refleja una versatilidad y portabilidad aún desconocida. El Centro Tecnológico de mecatrónica de Flandes,patrocinó y financió el desarrollo desde 2005 hasta el año 2009, facilitando la integración de Orocos en máquinas industriales.

PAL Robotics financió el esfuerzo requerido para la creación de la versión 2.0 de la Orocos ToolChain (explicada en siguientes apartados), siendo necesaria una reunión de desarrolladores en Barcelona, con el objetivode preparar el lanzamiento de esa versión.

Actualmente, SourceWorks es el principal patrocinador a la infraestructura de tiempo real Orocos ToolChain, mientras que muchas otras organizaciones contribuyen en la generación de las capas superiores. Por ejemplo, DFKI en Bremen (Alemania), ofrece a Rock Robotics las herramientas para la generación de código y componentes Orocos.

Los usuarios y colaboradores de todo el mundo, usan el software Orocos para el procesamiento de datos obtenido por sensores, la cinemática de una máquina, el control del robot. Como último apunte, los derechos de autor del código Orocos son compartidos por más de 20 colaboradores de diferentes países.

2.3.2 Kinematics y Dynamics Library (KDL)

Un esqueleto de un brazo robot en serie con seis articulaciones de revolución como el que se aprecia en la figura 21 es un ejemplo de una estructura cinemática. Mediante la utilidad que nos proporciona la KDL (librería de cinemática y dinámica) de Orocos, se consiguereducir la modelación y especificación del movimiento a un problema meramente geométrico (aunque con varios sistemas de referencia) con cálculos matemáticos, pudiéndose resolver un movimiento en base a unas especificaciones en cada una de las articulaciones.

La librería de Cinemática y Dinámica (KDL) desarrolla un marco de aplicación independiente para el modelado, y otro para el cálculo de cadenas cinemáticas en robots, modelos biomecánicos humanos, figuras animadas por ordenador, máquinas herramientas, etc. Además, la KDL proporciona librerías predefinidas de clases de objetos geométricos (como el punto o la línea), cadenas cinemáticas de varias familias (en serie, humanoide, paralelas o móviles) así como su especificación de movimiento e interpolación.

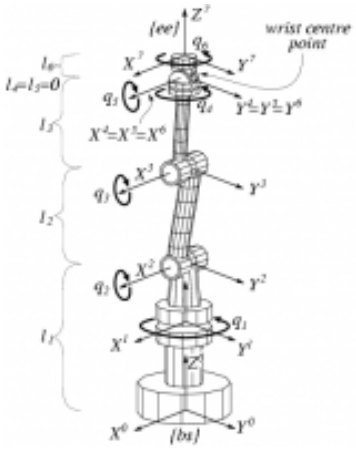

Figura 21: Brazo robot serie

El modo de funcionamiento es tan sencillo como definir los segmentos de las articulaciones y el “solver”:

```
//Definition of a kinematic chain & add segments to the chain
    KDL::Chainchain;
chain.addSegment(Segment(Joint(Joint::RotZ),Frame(Vector(0.0,0.0,1.020))));
chain.addSegment(Segment(Joint(Joint::RotX),Frame(Vector(0.0,0.0,0.480))));
chain.addSegment(Segment(Joint(Joint::RotX),Frame(Vector(0.0,0.0,0.645))));
chain.addSegment(Segment(Joint(Joint::RotZ)));
chain.addSegment(Segment(Joint(Joint::RotX),Frame(Vector(0.0,0.0,0.120))));
chain.addSegment(Segment(Joint(Joint::RotZ)));
// Create solver based on kinematic chain
ChainFkSolverPos_recursivefksolver=ChainFkSolverPos_recursive(chain);
```

Indicarle las posiciones finales de algunas articulaciones y, finalmente, calcular la ingeniería inversa:

```
kinematics_status=fksolver.JntToCart(jointpositions,cartpos);
if(kinematics_status>=0){
std::cout<<cartpos<<std::endl;
printf("%s \n","Succes, thanks KDL!");
}else{
printf("%s \n","Error: could not calculate forward Kinematics”)}
```

2.3.3 Orocos Real Time Toolkit (RTT)

La Orocos RTT aporta un marco o rutina en C++, encaminado o en vista a la aplicación de sistemas de control tanto en tiempo real como no. En el siguiente esquema se puede apreciar la idea principal.

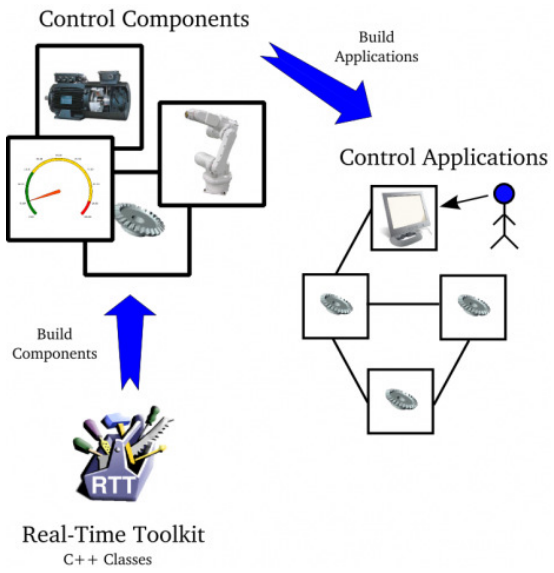

Figura 22:Orocos RTT Services

El Real Time Toolkit permite a los desarrolladores la creación de componentes totalmente configurables (incluso en tiempo de ejecución) e interactivos basados en el control de aplicaciones en tiempo real, pudiéndose usar en aplicaciones con características tales como:

- Capturar y graficar -el flujo de datos entre los componentes.
- Modificarlos algoritmos en tiempo de ejecución.
- Configurar los componentes y la aplicación a partir de archivos XML.
- Interactuar con otros dispositivos directamente desde una interfaz gráfica de usuario omediante el “prompt”.
- Ampliar las aplicacionescon estructuras de datos propias.
- Ejecutar la aplicación tanto en sistemas operativos estándar como sistemas de tiemporeal.

Además, el Real-Time Toolkit, permite que los componentes se ejecuten en cualquier sistema operativo (en tiempo real), ofreciendo todas las funciones y comandos del tiempo real, como lacomunicación de los componentes y la configuración de los mismos mediante un archivo XML.

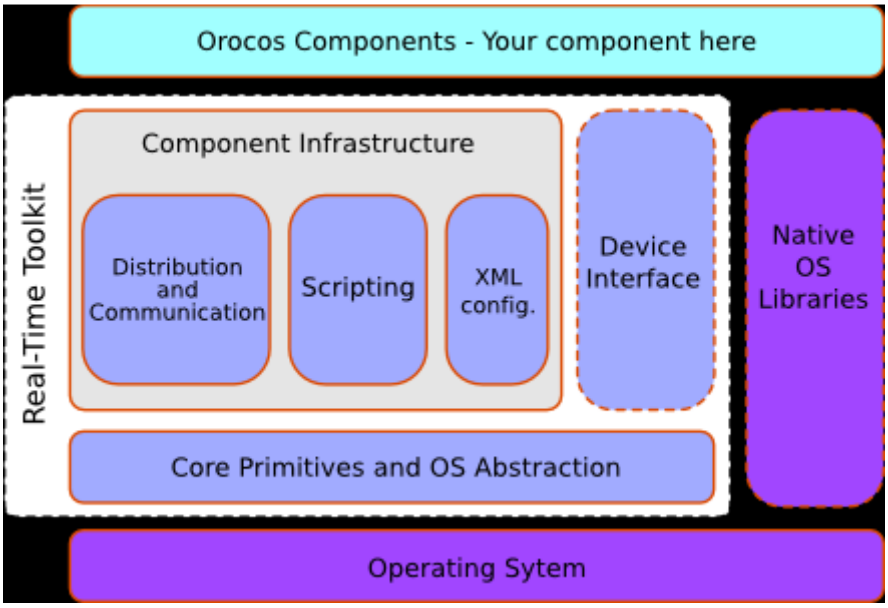

Figura 23:Orocos RTT AplicationStack

En Orocos, cada componente creado está basado en la primitiva “TaskContext” (un objeto que ofrece una transmisión de datos de forma eficiente y segura). En la siguiente figura se puede comprobar la estructura general de un componente.

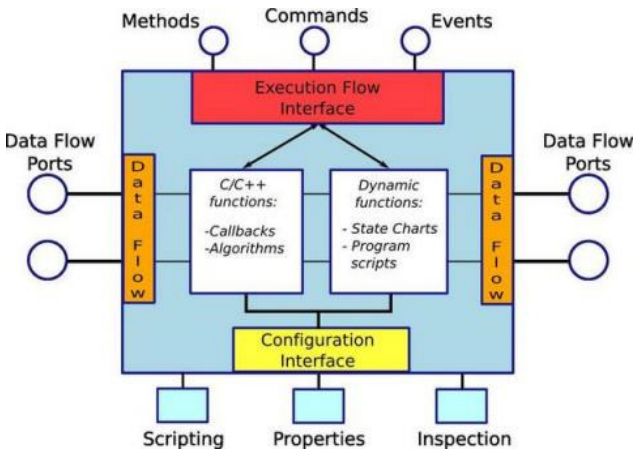

Figura 24: Primitiva TaskContext

Un componente puede estar configurado para reaccionar ante eventos o, simplemente, que se ejecute en un periodo determinado, fijado estrictamente por el sistema operativo de tiempo real. Estas opciones se puede configurar en tiempo de ejecución (dinámicamente mediante una interfaz)o mediante los archivos de configuración XML.

Además, mientras el componente está en ejecución, mediante las numerosas interfaces de distintos productos hardware del mercado, tales como encóders o tarjetas de conversión A/D, la recogida y muestreo de datos en tiempo real no es una tarea extremadamente complicada.

2.3.4 Bayesian Filtering Library (BFL)

La Bayesian Filtering Library (BFL) proporciona un marco de aplicación independiente para, por ejemplo, el procesamiento de la información recursiva y algoritmos de cálculo basado en la regla de Bayes, tales como filtros deKalman o Filtros de Partículas. Estos algoritmos pueden, por ejemplo, ser ejecutadoscomo una aplicación en tiempo real, o ser utilizado para la estimación de las aplicaciones de Cinemática y Dinámica.

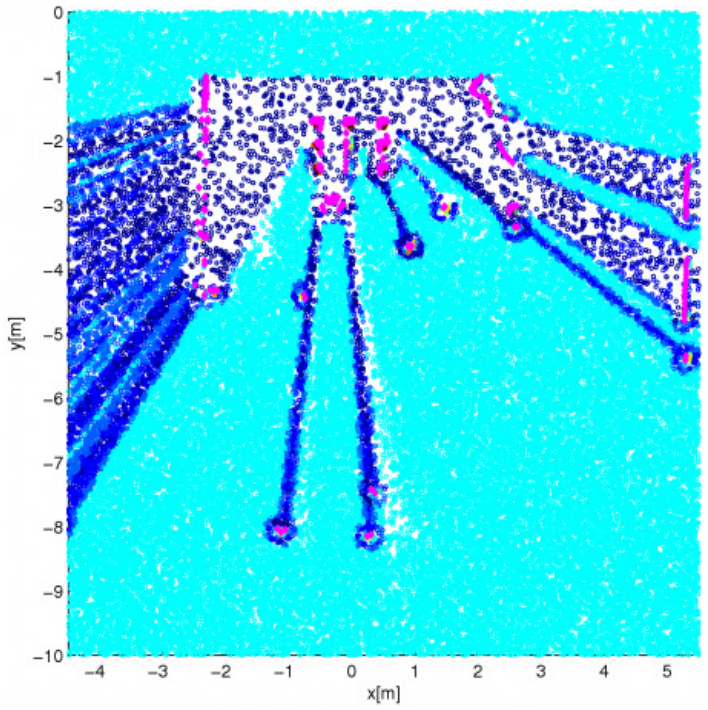

Figura 25: Estimación *pallet* con Orocos BFL

Cabe destacar que en la actualidad existen aplicaciones similares a la que nos propone Orocos, llegando a ser, incluso, mejores y más sencillas en su manejo. Por ello, en desarrollo en los últimos años de la BFL es mínimo, ya que no es tan usado como otras partes del proyecto Orocos.

2.3.5 Orocos Component Library

La Orocos Component Library es una de las partes más importante del proyecto Orocos, ya que mediante ésta, se crean todos los tipos de componentes (apoyándose enla RTT, anteriormente comentada).

Tal y como se ha comentado anteriormente, cada uno de los componentes está definido a partir de una primitiva llamada “TaskContext”, la cual define el entorno (contexto) mínimo o básico que debe tener un componente en Orocos (que posteriormente se modifica y configura según las preferencias del usuario). Este contexto está descrito por cinco primitivas de Orocos: “Event”, “property”, “Command, “Method” y “Data port”.

Un componente en sí es una unidad básica de funcionalidad que puede ejecutar uno o más programas (en tiempo real) en un único hilo o “thread”. Por otro lado, el sistema está libre de prioridades (de cara al usuario) y todas las operaciones son sin bloqueo (también el intercambio de datos y otras formas de comunicación tales como los eventos y los comandos). Otro aspecto importante es que los componentes en tiempo real pueden comunicarse con los componentes que no están en tiempo real, y viceversa, de forma transparente.

En el siguiente ejemplo se puede observar como varios componentes (unos de tiempo real y otro no) se pueden comunicar gracias a que la propia Real Time Toolkit se encarga de gestionar todo el sistema de prioridades de la manera más eficiente posible.

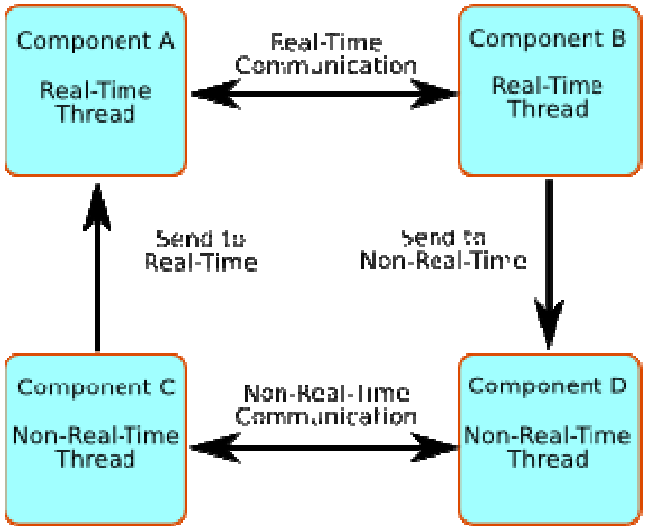

Figura 26: Comunicación entre varios componentes

2.3.6 Orocos ToolChain

La Orocos ToolChain es la creación más reciente del proyecto Orocos. Su primera versión apareció por julio de 2010 y, poco a poco, se está convirtiendo en la herramienta más usada.

La particularidad de la ToolChain es que en esta herramienta vienen incluidas otras herramientas como:

- “AutoProj”, que es una herramienta para descargar y compilar las librerías necesarias de forma automática.
- ”Real Time Toolkit” (anteriormente descrita).
- ”Orocos Component Library” (anteriormente descrita).
- ”OroGen y TypeGen”, que son dos herramientas para generar código Orocos a partir de unas cabeceras descritas, o de un fichero de descripción del componente (con una sintaxis ya predefinida).

De este modo, y en un único paquete, está todo lo necesario para comenzar a realizar componentes que tengan distintas funcionalidades. Además, una de las ventajas de la Orocos ToolChain es su soporte multiplataforma, ya que se puede trabajar tanto en Linux como en Windows o MAC (en algunos casos, sin utilizar la parte del tiempo real).

3 Desarrollo práctico

En este apartado, se comentará todo lo relacionado con el desarrollo práctico de este proyecto final de carrera. Se abordarán temas como el sistema operativo utilizado o la creación de los primeros componentes (para comprobar si funcionaban bien en Orocos), además de cómo se ha conseguido realizar la integración hardware con el entorno Orocos (la parte más tediosa y complicada de cualquier proyecto de investigación).

3.1 Sistema operativo en tiempo real

Uno de los aspectos importantes a determinar para la realización del proyecto era la elección correcta de un Sistema Operativo que nos proporcionara los servicios que se precisaban.

La opción de Windows se declinó, ya que no nos podía garantizar una ejecución en tiempo real (además de no ser de código abierto). Por ello se decidió optar por un Linux con un "parcheado" para que trabajara en tiempo real.

Finalmente se decidió instalar Ubuntu 10.04, un sistema operativo que utiliza un núcleo Linux, estando su origen basado en Debian. Ubuntu está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y mejorar la experiencia de usuario.

Por otro lado, una vez instalado el sistema operativo, hacía falta un parche para dotarlo de la funcionalidad típica de un RTOS ("Real Time Operating System"). Se optó por Xenomai.

Xenomai se basa en un núcleo de RTOS, de forma que se puede utilizar para la construcción de cualquier tipo de aplicación en tiempo real, sobre un núcleo genérico, como es el que tiene la versión 10.04 de Ubuntu.

Finalmente, tras la instalación del sistema operativo y el parche en tiempo real, al teclear la orden $uname –a, obtenemos la versión del núcleo que se está utilizando, tal y como se ve en la siguiente figura.

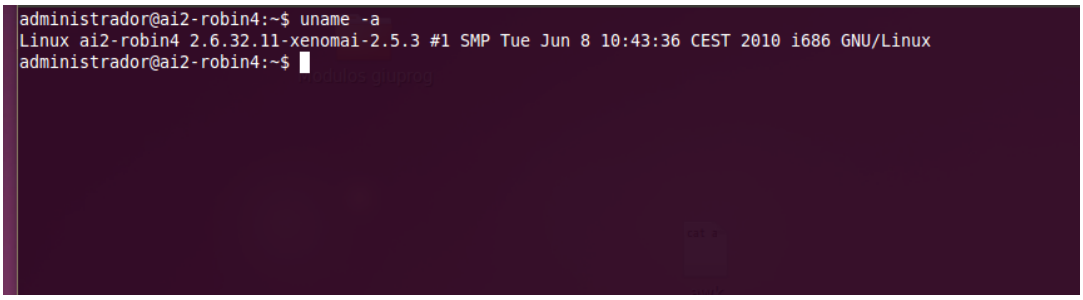

Figura 27: Salida uname -a

De esta forma, comprobamos que la instalación de Xenomai es correcta y estamos preparados para la instalación de la librería Orocos.

3.2 Orocos ToolChain

Como se comentó en apartados anteriores, la herramienta de Orocos que se va a utilizar va a ser la Orocos ToolChain, que es un paquete en el que vienen incluidos la Real Time ToolKit y la Orocos Component Library. Mediante estos dos paquetes se conseguirán crear numerosos componentes, ayudándonos de los ya creados, y del script "OroGen" (que nos ayudará para crear la estructura básica del mismo).

3.2.1 Instalación

Por lo que respecta a la instalación de la Orocos ToolChain, en nuestro proyecto se instaló la versión 2.2.0. En la actualidad (junio de 2011), la última versión estable es la 2.4.0. Pese a que la versión 2.2.0 funciona sin ningún problema, en proyectos futuros se debería actualizar a la última versión, ya que, al ser un software con continuas actualizaciones (la ToolChain lleva menos de un año), se van detectando algunos "bugs" que son resueltos por los desarrolladores.

En un primer paso, hay que ir a la página oficial del Orocos ToolChain (http://www.orocos.org/toolchain). Se tiene la opción de instalarlo mediante la red, aunque es recomendable descargarse los fuentes, para luego compilarlos desde el propio sistema.

Una vez se tienen descargados los fuentes de la Orocos ToolChain, se deben extraer en una carpeta del sistema de archivos local.

En este momento, dentro de la carpeta donde se ha extraído, hay que ejecutar las siguientes órdenes:

```
./bootstrap_toolchain

source env.sh
```

Con la primera orden se ejecuta un script, el cual comprueba que estén instalados todos los paquetes necesarios para la instalación de Orocos (como el paquete Ruby, por ejemplo). En caso de no estar instalado, lo indica y termina la instalación. Si en la instalación nos falta algún paquete, mediante el "Gestor de

paquetes de Synaptic" que nos ofrece Ubuntu, se puede instalar cualquier paquete sin ninguna dificultad. Una vez termina la ejecución del script, nuestro sistema está preparado para instalarle la ToolChain.

Es importante realizar la segunda orden cada vez que arranque el sistema, ya que incluye todos los PATH necesarios para la ejecución de algún componente de Orocos. En nuestro caso, se optó por incluirla dentro del archivo de configuración ".bashrc", de forma que cada vez que se inicia Ubuntu, se lanza esa orden de forma completamente transparente al usuario.

Posteriormente, se tiene que ejecutar la orden siguiente:

```
autoprojbuild-reconfigure
```

Este momento es crítico para el correcto funcionamiento de los componentes, puesto que hay que estar muy atento a la configuración y especificarle que nuestro RTT ("Real Time Target") es Xenomai (y no gnulinux, como aparece por defecto). Esta parte crítica de la instalación se puede apreciar en la siguiente captura.

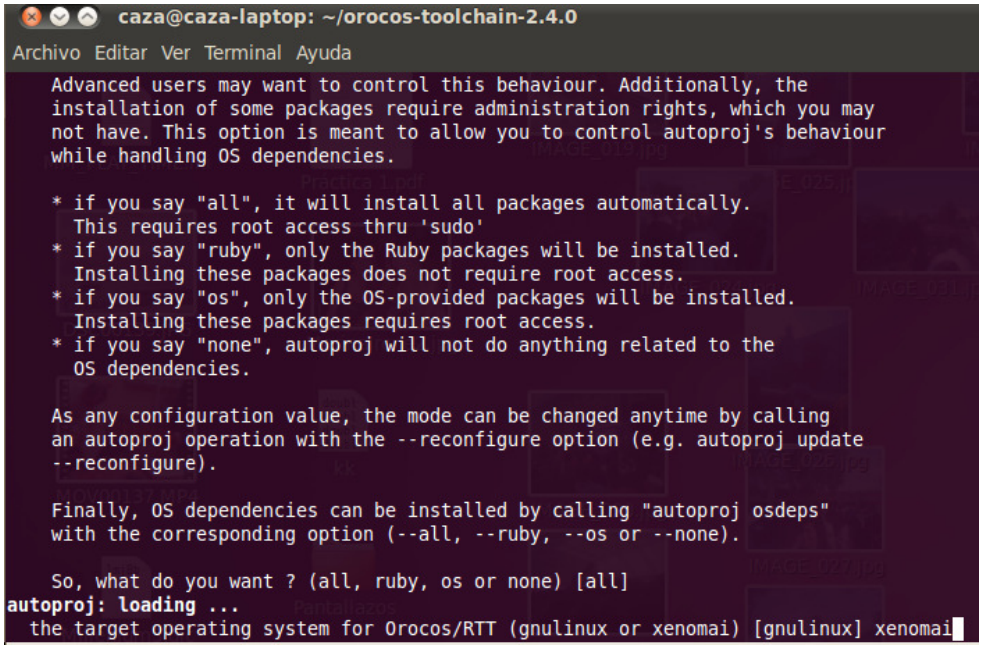

Figura 28: Instalación Orocos ToolChain

Finalmente, y si no se ha encontrado ningún tipo de problema, ya tendremos instalado en el directorio correspondiente la Orocos ToolChain. Al final de la instalación podremos comprobar que se han instalado los módulos “utilm”, “rtt”, “ocl”, “typelib” y “orogen” de forma satisfactoria.

3.2.2 Problemas encontrados

Respecto a los problemas encontrados, la gran mayoría vienen derivados de la falta de algún paquete por instalar. Cabe destacar que todas las pruebas y experimentos descritos en este documento, han sido testados con un PC Industrial con unas determinadas características, así como con una versión del kernel de Linux específica (2.6.32.11), por lo que pueden aparecer problemas distintos dependiendo de muchos factores.

Otro aspecto importante para evitar fallos en la ejecución de algún componente es tener la precaución de darle permisos de lectura universal al archivo “source.env”, comentado en el apartado anterior. Usando el siguiente comando, se soluciona el problema:

```
chmod 777 source.env
```

3.3 Componentes en Orocos

En esta sección se abordará todo lo relacionado con los componentes en Orocos, la parte más importante de todo el proyecto. Además de explicar las partes más importantes de un componente, se comentará cómo podemos crear, compilar y ejecutarlo.

Es muy importante conocer a la perfección cuál es su funcionamiento, puesto que mediante los mismos se deberá controlar el robot paralelo que es el fin de nuestro estudio. Finalmente, como prueba previa a la puesta en marcha del robot real, se ha simulado lo que podría ser el control de cualquier proceso (recibir datos de un sensor - tratar esos datos - escribir la acción de control).

3.3.1 Creación de un componente

Para la creación de un nuevo componente en Orocos, se puede hacer directamente de forma manual (realmente es un fichero en C++) o nos podemos ayudar de la herramienta que viene proporcionada en el OrocosToolChain, llamada OroGen.

OroGen no es nada más que un script, el cual genera toda la estructura de ficheros fuente y directorios, listos para compilar y ser ejecutados. La utilización de Orogen es tan sencilla como teclear lo siguiente:

```
orocreate-pkgnombreDelComponente
```

Una vez se hacreado un componente, tendremos la estructura que muestra la figura siguiente.

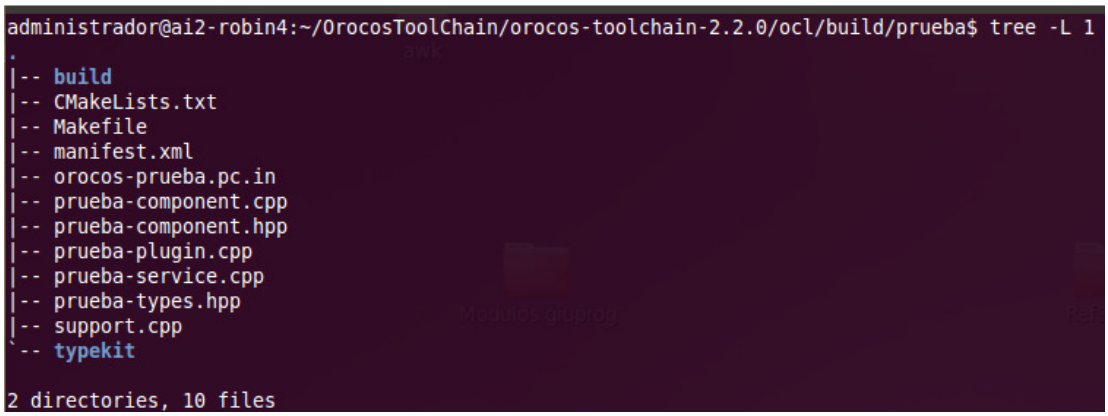

Figura 29: Estructura de ficheros de un componente

Dentro de esta estructura, hay ficheros que son importantes conocer, de cara a poder modificarlos acorde a nuestras necesidades. Estos ficheros son:

- nombreDelComponente-component.hpp: Este fichero es el que contiene el código fuente y estructura mínima que debe tener un componente en Orocos. Será en este fichero en el que se irán

añadiendo las funciones que se vayan necesitando, así como implementar los métodos predefinidos (que se explicarán en el apartado 4.3.2).

- CMakeLists.txt: Este fichero no se debe modificar, puesto que contiene toda la información necesaria para la creación del Makefile (y así poder compilar el componente de forma automática). Para la generación del Makefile se usarán los programas ccmake y cmake (que se comentarán en el apartado 4.3.3)

3.3.2 Estructura de un componente

Otro aspecto a tener en cuenta a la hora de crear o modificar un componente es su estructura. Pese a utilizar el lenguaje de programación C++, es muy distinto a un programa típico en C++. En nuestro caso, un aspecto importante a destacar es que no existe ninguna función "main".

En un componente de Orocos se pueden diferenciar claramente 5 partes:

1. **Zona de declaración de variables globales.**En Orocos es muy importante tener en cuenta todas las variables que se van declarando. Es altamente recomendable cuando se quiere que alguna variable vaya cambiando de valor en cada una de las iteraciones, que la variable se declare como variable global. En caso de que no se haga así (y se declare en una función que se ejecute iterativamente), cuando el componente realice la segunda iteración de forma automática, se producirá un error en tiempo de ejecución de "Violación de Segmento" (y todo lo que ello conlleva) ya que en ese supuesto se intentaría re-declarar la variable.

2. **Función "ConfigureHook()".** Esta función es característica del entorno Orocos, y es obligada en todos los componentes. Ya que se ha usado la herramienta OroGen para la creación de la estructura, todas la funciones obligatorias en un componente ya está definidas (estando listas para introducir el código que el usuario desee). En esta función, que es para la configuración del componente,

se debe dar valor a las distintas variables globales que se hayan declarado previamente. En posteriores apartados, se verá la importancia de definir las variables en esta función**.**

3. **Función "StartHook()".** El código que se añada en esta función, se ejecutará inmediatamente antes de poner en ejecución el componente principal. Es decir, primero se ejecuta el código, y posteriormente se pone en marcha el componente. Generalmente no se suele añadir código en esta función, ya que es más común añadir ese código en el "ConfigureHook()" (que primero define las variables, y luego le da la opción al usuario de poner en ejecución el componente).

4. **Función "UpdateHook()".** Esta es la función central y más importante de un componente en Orocos. Todo el código que se introduzca aquí dentro, se ejecutará estrictamente en un determinado periodo que se deberá definir (de forma interactiva antes de ejecutar el componente, o dentro de la función "ConfigureHook()"). Es muy importante saber gestionar e implementar, por ejemplo, un bucle "While", ya que no hay que dejar de pensar que absolutamente todo el código que esté dentro de esta función se ejecutará cada cierto periodo. Por ejemplo,si se pone un bucle dentro de esta función, se ejecutará todo el bucle en cada iteración del componente (pudiendo comprometer una ejecución en tiempo real).

5. **Función "StopHook()".** Esta función será la última que se ejecute, ya que es la que se lanza cuando se quiere detener el componente. Generalmente, se programa un componente en el que en la función "UpdateHook()", mediante alguna variable booleana que indique el fin un ejecución, se pueda llamar a "StopHook()" para así detener el componente. En esta función, lo primero que hay que hacer es poner el periodo de ejecución a cero (en caso de no hacerlo, genera un error en ejecución). Tras poner el periodo a cero, en esta función es donde hay que, por ejemplo, cerrar las tarjetas de conversión A/D y las tarjetas de encóders.

En la siguiente captura se puede observar la estructura de un componente.

```
class Prueba
    : public RTT::TaskContext
{protected:
/*
ZONA DE DECLARACIÓN DE VARIABLES GLOBALES
*/
 public:
    Prueba(string const& name)
        : TaskContext(name)
    {
        std::cout << "Prueba constructed !" <<std::endl;
    }

    bool configureHook() {
        std::cout << "Prueba configured !" <<std::endl;
        return true;
    }

    bool startHook() {
        std::cout << "Prueba started !" <<std::endl;
        return true;
    }

    void updateHook() {
        std::cout << "Prueba executes updateHook !" <<std::endl;
    }

    void stopHook() {
        std::cout << "Prueba executes stopping !" <<std::endl;
    }
```

Figura 30: Estructura de un componente

3.3.3Compilación de un componente

Tal y como se ha comentado en el apartado 4.3.1, cuando se crea un componente mediante "OroGen", ya se crean todos los ficheros necesarios para la compilación del mismo, así como toda la estructura de directorios.

Los pasos a seguir para la compilación de un componente son los siguientes:

- Dentro de la carpeta que nos ha generado "OroGen" (que es donde está el CMakeFileLists.txt, etc.) creamos una carpeta, que se puede llamar "build" y entramos en ella.

```
mkdirbuild
cdbuild
```

- Dentro de ese directorio, mediante la herramienta "ccmake" (que permite configurar el fichero "CMakeLists.txt" del directorio padre

mediante una interfaz muchísimo más sencilla) y “cmake” (que teniendo un fichero de configuración, generado por “ccmake”, es capaz de generar un Makefile) se consigue crear un Makefile con el que se puede compilar el componente. Tras teclear la siguiente orden, nos encontraremos con la siguiente interfaz de configuración.

```
ccmake ..
```

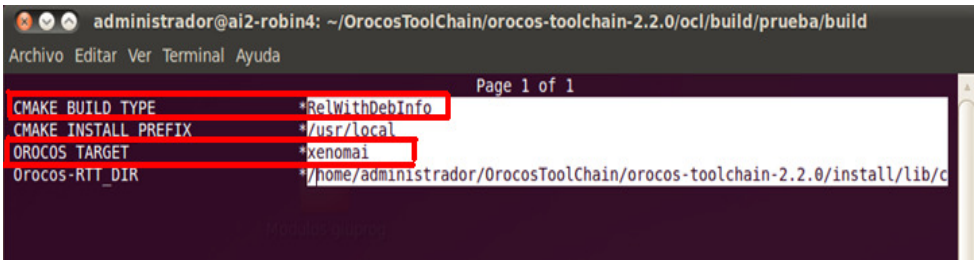

Figura 31: Configuración *ccmake*

Es muy importante en la configuración mediante el “ccmake” introducir los parámetros correctos. En nuestro caso, como nuestro “Real Time Target” es Xenomai, habrá que indicarlo de forma oportuna en la configuración, así como el modo de compilación, en vez de ser “Release” (que viene por defecto), en nuestro caso pondremos “RelWithDebInfo”. Tras ir configurando las diversas opciones, dentro de la interfaz aparece la opción de generar esa configuración, por lo que genera el archivo definitivo.

- Finalmente, cuando se ha creado el fichero final, ayudándonos de la herramienta “cmake” conseguiremos crear el definitivo fichero “Makefile” (para poder compilar). Se deberán de teclear las siguientes órdenes, obteniendo un resultado similar al de la siguiente imagen.

```
cmake ..
```

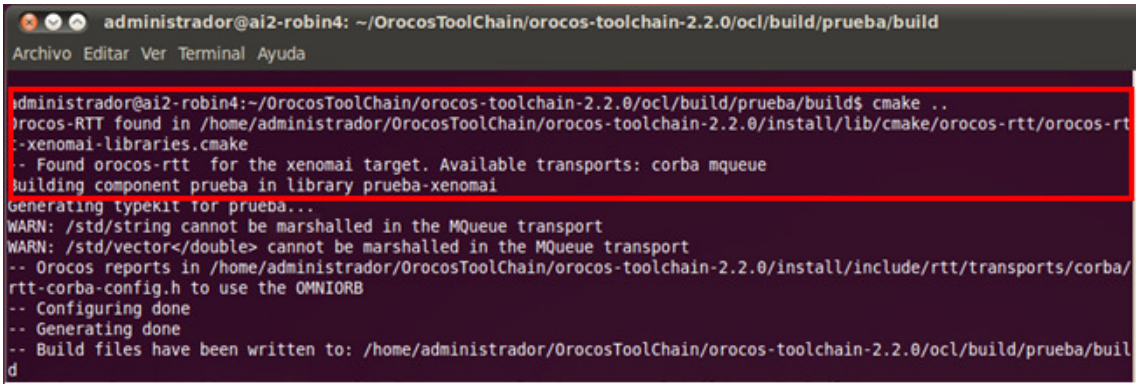

Figura 32: Configuración *cmake*

- Como paso definitivo, sólo queda lanzar el comando "make" y comprobar cómo se generan los archivos con extensión .so. Uno de los cuales será el componente que nosotros hemos creado (libNombreDelComponente-xenomai.so).

make

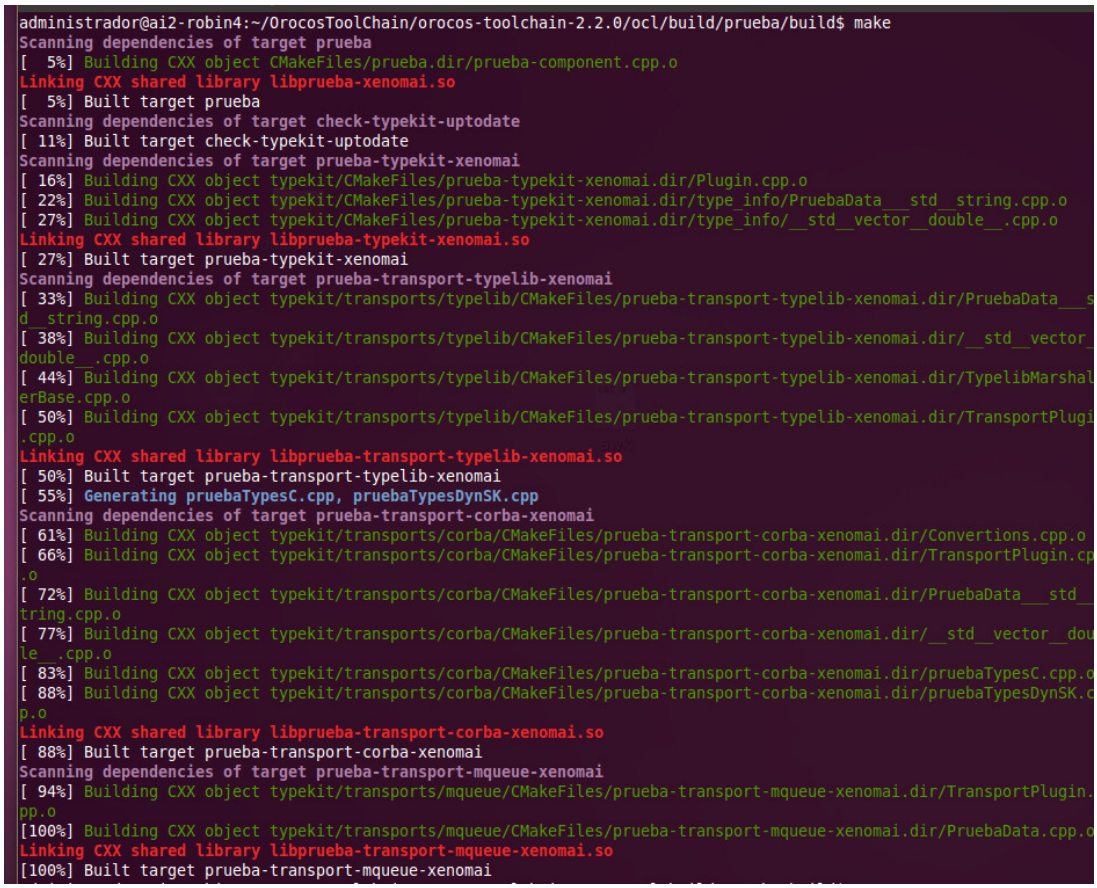

Figura 33: Compilación componente

3.3.4 Ejecución de un componente

Una vez se ha compilado el componente y se tiene en un formato .so, ya se está en disposición de lanzarlo aejecución.

A diferencia de un programa típico en C++, un módulo no puede ser ejecutado sin más, ya que no haría absolutamente nada.

La forma de poner en ejecución un módulo es mediante el módulo “TaskBrowser”, que ya viene con la instalación de Orocos. El componente “Taskbrowser” tiene la característica de poder cargar otros módulos en un marco de ejecución. De esta forma, cuando se carga algún componente mediante este módulo, de forma dinámica crea una conexión con él, la cual le permite enviarle órdenes (del tipo *configure*, *start* o *stop*, comentadas anteriormente). Puesto que cada módulo que se crea ya tiene implícita una serie de órdenes predefinidas, (como “setPeriod”, “start”, “stop”, etc.) desde la “TaskBrowser” se pueden llamar a esas funciones de los componentes de una forma dinámica y sencilla, cambiando, por ejemplo, el periodo de un componente de una forma interactiva. En la siguiente ilustración se muestra la función del módulo “TaskBrowser”.

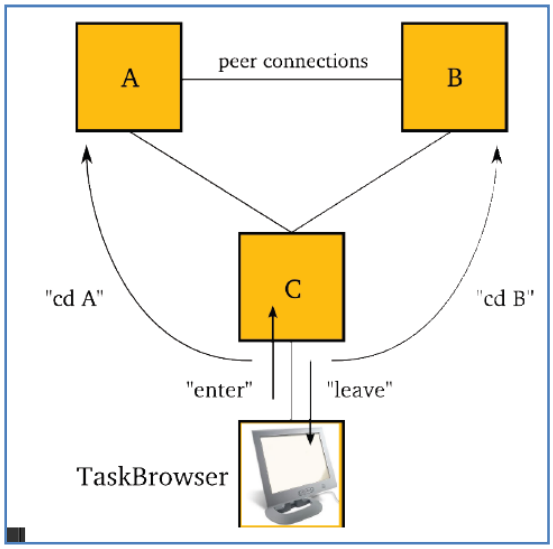

Figura 34: Módulo TaskBrowser

Inicialmente, la “TaskBrowser” está conectada a un módulo, pero a medida que va cargando otros módulos, puede trasladarse a cualquiera de ellos, y así poder enviarle cualquier orden que se desee. Para cambiar a un componente se usa la orden “cd nombreDelComponente”.

3.3.5 Simulación Proceso

Como paso previo a la integración de los componentes de Orocos a una situación real (con un robot, por ejemplo), se decidió crear tres componentes con el fin de simular esa situación.

Realmente, nuestro objetivo final será leer datos de un sensor, tratar esos datos para determinar cuál será la acción de control a aplicar, y finalmente sacar ese dato por un puerto de salida (en dirección al dispositivo que se quiera controlar).

Por ello, se decidió crear un componente que fuera emitiendo valores (simulando un sensor), otra componente que recibiera esos datos e hiciera una operación sobre ellos y los enviara a otro componente, y un último componente que recogiera esos datos y se los enviara al dispositivo final.

El esquema general es el que se puede apreciar en la siguiente ilustración, explicando en los siguientes apartados cómo se han programado estos componentes, haciendo especial hincapié en la transmisión de datos entre componentes mediante los buffers de entrada y salida.

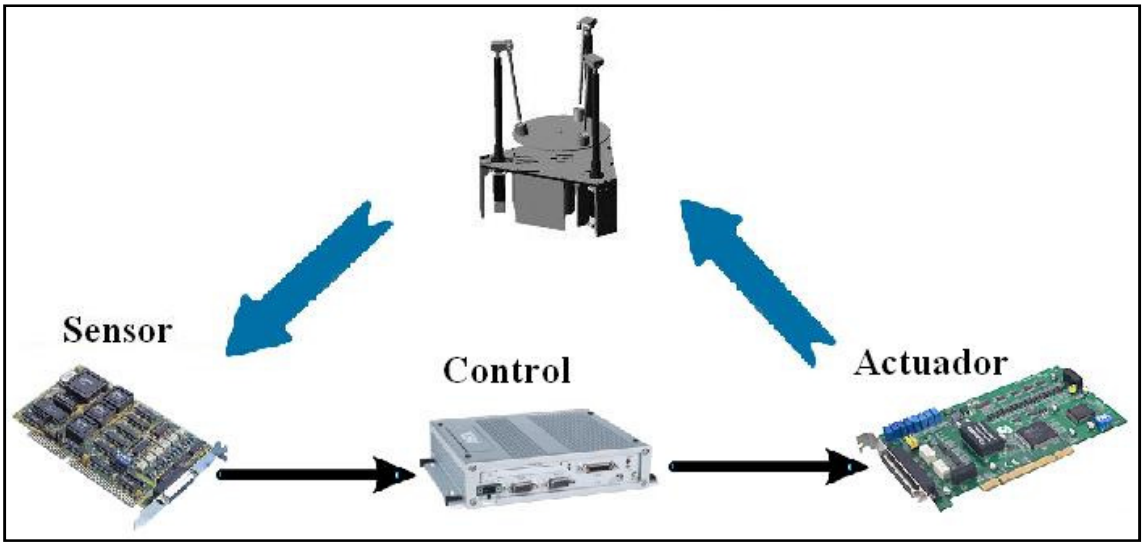

Figura 35: Esquema sensor

3.3.5.1 Simulación sensor

Para simular un sensor, lo que se ha hecho ha sido crear un componente con el nombre "sensor" (mediante el OroGen, explicado en el apartado 4.3.1) y se ha compilado (de la forma descrita en el apartado 4.3.3) para comprobar que se han hecho los pasos correspondientes antes de empezar a escribir código.

La función general del sensor será, en cada ciclo, incrementar una variable contador y enviarla por el puerto de salida. En nuestro caso, se ha definido como variable global un OutPort de la siguiente forma:

```
RTT::OutputPort<int> out_port;
```

No queda más que, en cada iteración (de ello se encarga la función UpdateHook()) ir incrementando y enviando ese valor. La función UpdateHook() quedaría de la siguiente manera:

```
void updateHook() {
            counter+=1;
                        out_port.write(counter);}
```

De esta forma, en cada iteración (dependiendo del periodo que se asigne a este componente mediante el TaskBrowser) se ejecutará el código anterior, incrementando y enviando ese dato por el puerto de salida. Cabe destacar la importancia de que no

haya ninguna declaración de variables en la función anteriormente mencionada, ya que, evidentemente, en cada iteración se intentará volver a definir, y el programa terminaría con una "violación de segmento" en tiempo de ejecución.

Otras funciones que se han implementado han sido las siguientes:

- `void resetCounter()`. Con esta función el contador se pondrá a cero.
- `int getCounter()`. Con esta función se obtiene el valor actual del contador.
- `void setCounter(int c)`.A través de esta función se consigue modificar el valor actual del contador con el valor que se desee.

Pese a que se han definido estas funciones, si cargamos el componente con el TaskBrowser, no podremos utilizar de forma interactiva estas funciones ya que no se han añadido a la interfaz. Para conseguir usarlas en la interfaz de usuario, se deben añadir de la siguiente forma.

```
this->addOperation(   "resetCounter",   &Sensor::resetCounter,   this,
ClientThread ).doc("Reset the counter");
      this->addOperation(   "getCounter",   &Sensor::getCounter,   this,
ClientThread ).doc("Get the counter value");
      this->addOperation(   "setCounter",   &Sensor::setCounter,   this,
ClientThread ).doc("Set the counter value");
```

En esta ocasión, si se carga el componente con el TaskBrowser (loadComponent("sensor","s1")), y vemos cuáles son los métodos que se pueden usar, ya aparecerán los que se han incluido en la interfaz.

Figura 36: Métodos componente sensor

3.3.5.2 Simulación control

Para la simulación del sistema de control, por lo que se refiere a la creación y compilación del componente, es exactamente igual que el componente anteriormente creado. Realmente, aun sistema de control le llegan un cierto número de datos, los procesa, y los envía. En nuestro caso, se ha hecho un módulo, el cual tiene un puerto de entrada y otro de salida (cada uno para comunicarse con cada uno de los componentes).

```
RTT::OutputPort<int> out_port;
RTT::InputPort<int> in_port;
```

Cuando le llega un dato por el puerto de entrada, lo procesa (en nuestro ejemplo, lo multiplica por un factor de ganancia), y lo envía por el puerto de salida. Puesto que el componente “sensor”, que es el que le envía los datos (al puerto de entrada) tiene un determinado periodo, la forma más óptima para recoger el dato nuevo que llegue a esta segunda tarea sería por medio de un evento (el momento en el que el dato que le llega es un nuevo dato). Para realizarlo mediante Orocos es de la siguiente forma:

```
if ( in_port.read(val) == RTT::NewData )
            {val=val*factor;
             out_port.write(val);}
```

Como se puede observar en el anterior ejemplo, sólo tratará y enviará el dato que recibe cuando se tenga la certeza de que es un nuevo dato. Por ejemplo, si el sensor dejara de leer por algún problema de hardware, se estaría detectando que no es un nuevo dato, por lo que no pasaría a tratar y enviar ese valor.

Finalmente, algunas de las funciones que se han implementado en este componente han sido las siguientes:

- `int getFactor()`. Devuelve de factor de ganancia.
- `void setFactor(int c)`. Permite especificar el factor de ganancia.

De forma análoga al anterior componente, si se desea que las funciones implementadas aparezcan en la interfaz del propio componente (y así poder lanzarlas mediante el TaskBrowser), se debe hacer de la misma manera que se ha hecho en el apartado anterior.

3.3.5.3 Simulación actuador

El último componente creado fue el que haría de “actuador”. La función de este componente sería recoger el dato proveniente del “control” y enviarlo al robot.

Se creó un único puerto de entrada para recoger el dato. Por otro lado, no se creó ningún puerto de salida, puesto que la acción de control no se deberá enviar por un puerto de salida de Orocos, sino por una tarjeta física.

De la misma forma que el componente “control”, éste se ha programado mediante eventos, por lo que sólo leerá el dato de su puerto de entrada cuando sea un dato nuevo.

3.3.6 Conexión de componentes

En los apartados anteriores (3.3.5.1, 3.3.5.2 y 3.3.5.3) se han creado tres componentes con el objetivo de simular el comportamiento de un robot de forma muy básica. Pese a haberlos implementado, creando puertos de entrada y salida en cada uno de los componentes por separado, falta realizar el último paso, que es interconectarlos entre sí.

Evidentemente, esta conexión debe ser de forma dinámica e interactiva mediante las opciones que nos da la "TaskBrowser" (lanzándola con el programa binario "deployer-xenomai"), ya que en tiempo de compilación ningún componente sabe si existe algún otro componente con puertos de entrada o salida para conectarse con él.

Para conectar un componente con otro, se deben de cumplir dos condiciones:

- Un puerto de salida sólo se puede conectar con un puerto de entrada, y viceversa.
- Los datos de cada puerto deben ser del mismo tipo (por ejemplo, no puede conectarse un puerto que acepte cadenas de caracteres con otro que acepte número enteros).

En nuestro caso, los tres componentes tenían puertos del tipo entero, y estaban diseñados para conectarse según la figura del apartado 3.3.5. Una vez se tienen cargados los componentes, mediante la siguiente función que nos ofrece la TaskBrowser, se puede conseguir:

```
ConnectTwoPorts (NomComp1, PortIn, NomComp2, PortOut);
```

Finalmente, mediante una conexión correcta de puertos y una puesta en ejecución de los tres componentes, se consigue el resultado esperado: simular de forma satisfactoria y exitosa el comportamiento básico de un robot, así como afianzar conceptos de la librería Orocos.

3.4 Hardware entrada/salida

En este apartado se comentará todo lo relacionado al hardware de entrada/salida que se ha utilizado en la realización del proyecto (en nuestro caso, la tarjeta PCI 1720 y la PCL 833 de Advantech). Además, en apartados posteriores se justificará esta elección, así como la tarea de integrar este hardware con el proyecto Orocos.

3.4.1 PCI 1720

La PCI 1720 es una tarjeta de conversión digital/analógica creada por Advantech. Esta tarjeta contiene cuatro canales de 12 bits para la salida analógica, además de la posibilidad de elegir entre cuatro rangos de tensión. Estos rangos son los siguientes:

- 0~5V
- 0~10V
- ±5V
- ±10V

La posición de los motores del robot va cambiando dependiendo de la tensión que se le esté aplicando, por lo que, mediante la tarjeta PCI 1720, se irá aplicando distintos valores de tensión (que previamente serán calculados mediante el algoritmo de control).

Figura 37: Tarjeta A/D PCI 1720

En nuestro caso, puesto que hay que controlar los tres ejes del robot, como mínimo se necesitarán tres de las cuatro salidas que tiene la tarjeta, escogiendo un rango de tensión de ±10V, pese a que en el algoritmo final se saturará a ±5V para evitar movimientos demasiado rápidos del robot.

En la siguiente tabla se pueden observar los pines más importantes, así como las conexiones que se han realizado usando la placa de conexiones PCLD-880 para, posteriormente, poder realizar la conexión con el robot.

Terminal	**Descripcion**
A19	GND
A9	Channel 0
A15	Channel 1
B1	Channel 2
B7	Channel 3

Tabla 1: Conexiones PCI 1720 – PCLD-880

Se ha usado esta tarjeta, ya que es la más adecuada para nuestro propósito, teniendo un error del ±0.024%, y permitiendo una frecuencia de trabajo acorde a las necesidades para el control del robot.

3.4.2 PCL833

La PCL 833 es una tarjeta de encóders de 3 ejes de cuadratura. A diferencia de la tarjeta 1720, ésta se conecta al bus ISA (y no al PCI). La PCL 833 de Advantech tiene tres contadores independientes de 24 bits.

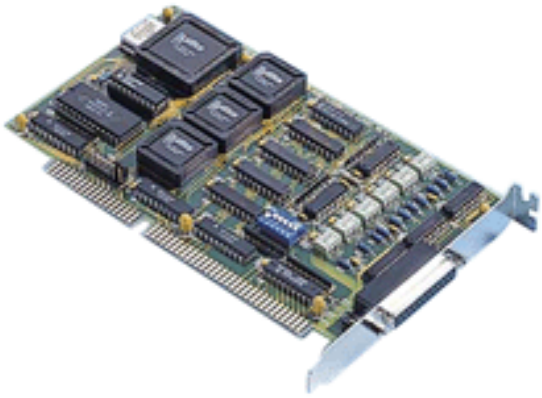

Figura 38: Tarjeta encóders PCL 833

Puesto que los encóders de los motores del robot generan señales que indican información acerca de la posición de los ejes, la principal función de la tarjeta de encóders es recibir esa información e interpretarla, haciendo la conversión de esa secuencia de pulsos a cuentas del motor. Una vez se tengan los pulsos del motor, haciendo una conversión de pulsos a posición (de modo experimental o basándose en la especificación de los encóders) se puede controlar la cinemática y dinámica del robot.

De la misma forma que se ha hecho con la PCI 1720, para una mejor maniobrabilidad a la hora de cablear, se ha conectado a una placa las entradas de los

tres encóders, siendo los pines más importantes lo que se pueden apreciar en la siguiente tabla:

Terminal	Descripción	Tipo
1	GND	
2	Counter 1	A+
3	Counter 1	B+
5	Counter 2	A+
6	Counter 2	B+
8	Counter 3	A+
9	Counter 3	B+
14	Counter 1	A-
15	Counter 1	B-
17	Counter 2	A-
18	Counter 2	B-
20	Counter 3	A-
21	Counter 3	B-

Tabla 2: Conexiones PCL 833

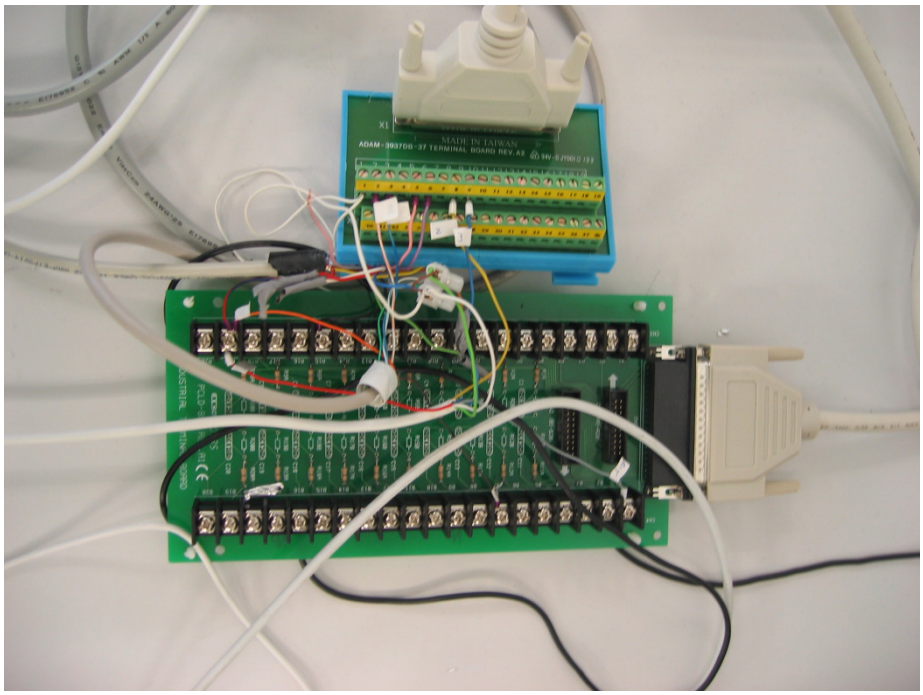

Figura 39: Conexiones PCI1720 y PCL833

3.5 Integración del hardware entrada/salida en Orocos

En esta parte del proyecto, se abordará todo lo relacionado con la integración del hardware escogido para la comunicación entre el robot y el PC.

En la primera parte se explicará las herramientas que se utilizaron para la integración, mientras que en la segunda parte se explicarán los diferentes componentes de prueba que se crearon para la comprobación del correcto funcionamiento, así como los problemas que se encontraron a medida que avanzaba el proyecto.

3.5.1 Comedi

Uno de los factores que más complicaciones supone en un proyecto, en el que el hardware (en nuestro caso una tarjeta PCI y otra ISA) ocupa una papel muy importante, es la instalación, configuración y manejo de los mismos.

Si a esta complicación le añadimos que no se está usando ningún sistema operativo de pago (como Windows), por lo que se está usando un Linux en tiempo real, el hecho de obtener los controladores del hardware es mucho más difícil.

En un primer momento se intentaron instalar los drivers de la tarjeta PCI 1720 para Linux, descargándolos de la página oficial del fabricante (www.advantech.com), aunque el resultado no fue satisfactorio. No se consiguió instalarlos, ya que esos drivers eran para versiones de núcleo muy específicas y demasiado antiguas. Por lo tanto, se estuvieron barajando diversas opciones, siendo la elegida el proyecto "Comedi".

El proyecto Comedi ("Control and Measurement Interface") desarrolla drivers, herramientas y librerías en código abierto para la adquisición de datos. Comedi es una colección de drivers para una gran variedad de tarjetas relacionadas con la adquisición de datos. Además, los drivers están implementados como un "módulo del núcleo de Linux", teniendo la funcionalidad típica de un módulo.

Por otro lado, la "Comedilib" es una librería implementada en C, en la que se incluyen todas las funciones y métodos para poder manejar todo tipo de dispositivos. Aparte de proporcionar al usuario todas las funciones y programas de configuración de las tarjetas, se incluyen diversos programas de testeo y prueba, sirviendo de ejemplo en algunos casos.

Finalmente, mediante Comedi se pudo instalar (insertar el módulo) la tarjeta PCI 1720 y comprobar su funcionamiento. En el apartado 3.5.3 se explicará cómo se realizó esa inserción de módulos y qué programas se usaron para el testeo de la tarjeta.

3.5.2 Orocos Comedi Interface

Además de la "Comedilib", explicada en el apartado anterior, otra forma de poder llamar a las funciones que manejan y testean las tarjetas es mediante unas interfaces (incluidas dentro de la Orocos Component Library) que nos ofrece Orocos: la Orocos Comedi Interface.

En la Orocos Comedi Interface se implementan diversas interfaces para todos los dispositivos que incluye Comedi. Realmente, es un "mapeo" de la "Comedilib", pero adaptada con las propias clases de Orocos. En la siguiente tabla, aparecen las interfaces disponibles en Orocos:

Interface	Comedi Implementation	Description
RTT::AnalogInInterface	OCL::ComediSubDeviceAIn	Reading analog input channels
RTT::AnalogOutInterface	OCL::ComediSubDeviceAOut	Writing analog output channels
RTT::DigitalInInterface	OCL::ComediSubDeviceDIn	Reading digital bits
RTT::DigitalOutInterface	OCL::ComediSubDeviceDOut	Writing digital bits
RTT::PulseTrainGeneratorInterface	OCL::ComediPulseTrainGenerator	Sends out block waves
RTT::EncoderInterface	OCL::ComediEncoder	A position/turn incremental encoder

Tabla 3: Orocos Comedi Interface

3.5.3 Modulo pci1720Out

Como se ha expuesto anteriormente, existen dos posibilidades para poder manejar las tarjetas: mediante la Orocos Comedi Interface, y a partir de la "Comedilib". Para comprobar ventajas e inconvenientes, se ha creado un módulo, cuya funcionalidad es la misma, aunque la forma de programarlo ha sido distinta.

Independientemente de cómo se van a manejar las tarjetas, la forma de insertar el módulo (para que reconozca la tarjeta) es la misma para ambos casos.

Primeramente hay que instalar la librería "Comedi" (ya sea desde los repositorios o compilando los fuentes), de forma que los módulos de todas las tarjetas que soporta, se quedan guardados dentro de un directorio del sistema. Seguidamente hay que insertar el módulo correcto (véase el listado en la página web de "Comedi" http://www.comedi.org/hardware.html) con el siguiente comando:

```
modprobe adv_pci1710
```

De esta forma, si observamos los mensajes de diagnóstico del sistema (mediante el comando “dmesg”) nos damos cuenta de que la tarjeta se ha insertado de forma correcta.

Finalmente, y tras haber instalado la tarjeta, se detallará las dos formas de poder manejar una tarjeta A/D indicando todo el proceso, así como las dificultades encontradas (y como se han conseguido resolver).

3.5.3.1 Orocos Comedi Interface

La primera opción que se tiene es usando las interfaces (mapeadas mediante Comedi) que nos proporciona Orocos. La principal ventaja de realizarlo mediante esta opción es que su uso es bastante sencillo, puesto que no hay que preocuparse de hacer la conversión (según el rango de la tarjeta) para sacar un valor determinado de tensión, ya que está todo implementado en capas inferiores. Por el contrario, al estar encapsulado en más bajo nivel, no se tiene la opción de realizar algunos cambios (tales como la elección del rango de tensiones que se quiere utilizar).

Para poder usar estas interfaces, los pasos son los siguientes:

- Declarar en la zona de variables globales, las variables que se van a necesitar. En nuestro caso se necesitarán un “ComediDevice”, “ComediSubDeviceAOut” y “AnalogOutput”.

```
ComediDevice* comediDevAOut;
ComediSubDeviceAOut* comediSubdevAOut;
unsignedint minor;
AnalogOutput* signaal0;
```

- Tras tenerlas declaradas, en la parte de configuración deben configurarse de la siguiente forma.

```
comediDevAOut =new ComediDevice(minor);
comediSubdevAOut=new ComediSubDeviceAOut (comediDevAOut,
"pci1720",0);
signaal0 = new AnalogOutput(comediSubdevAOut,0);
```

Como parámetros más significativos, caben destacar que el valor de la variable “minor” debe ser el que se le asigne al dispositivo de “Comedi” en el

momento de la instalación (en nuestro caso es 0, ya que en la instalación se inserta el módulo como “comedi0”), así como a la hora de seleccionar el canal por el que se sacará la tensión (segundo argumento de la función “AnalogOutput”).

- Finalmente, tras haber declarado y definido las variables, si, por ejemplo, se quieren enviar 2.34 V por el canal 0, no habrá más que añadir la siguiente línea:

```
signaal0->value(2.34)
```

Por tanto, de la forma anteriormente descrita, se consigue sacar una determinada tensión (sin preocuparse de la conversión respecto al rango) por un canal específico usando la Orocos Comedi Interface. Pese a ello, se han encontrado inconvenientes como los siguientes:

- No está implementada la opción de poder seleccionar el rango de tensión.
- La Orocos Comedi Interface no está incluida dentro de la Orocos ToolChain 2.x, por lo que para poder utilizarla se ha tenido que compilar (de versiones anteriores), y modificar el “Makefile” para que incluyera la librería compilada de interfaces.

3.5.3.2Comedilib

Otra de las opciones que se tienen para manejar la PCI 1720 es mediante la “Comedilib”. En esta librería (implementada en C) vienen implementadas todas las funciones necesarias para un control y configuración completo de la tarjeta.

A diferencia de la Orocos Comedi Interface, su uso es algo más complicado pero, por lo contrario, se tienen muchas más posibilidades en lo que a funcionalidades ser refiere. Para usarlo en el módulo creado en Orocos, se han seguido los siguientes pasos:

- Declarar en la zona de variables globales las variables necesarias para la inicialización del dispositivo.

```
comedi_t *cf;
```

```
lsampl_t data;
int maxdata;
comedi_range *cr;
```

- En la zona de configuración, inicializar las variables previamente definidas de la siguiente forma.

```
cf = comedi_open("/dev/comedi0");
maxdata = comedi_get_maxdata(cf, subdev, chan);
cr = comedi_get_range(cf, subdev, chan, range);
```

Realmente, únicamente con llamar a la función “comedi_open” sería suficiente, aunque, ya que en nuestro proyecto es vital que el sistema funcione correctamente, por lo que se obtiene el campo “max_data” y el rango para comprobar que todo está correcto.

- Como último paso, cuando se quiere sacar un valor de tensión por un canal determinado, se tiene que realizar lo siguiente, donde “val” es el número de voltios que se desean sacar, “canal” es el canal que por el que se quiere sacar la tensión, y “range” es el rango de tensiones que se quiere escoger.

```
data=comedi_from_phys(val,cr,maxdata);
result=comedi_data_write(cf,subdev,canal,range,AREF_GROUND,data)
```

Como se puede observar, antes de dar la orden de sacar una tensión, es necesario realizar la conversión de los voltios dependiendo del rango de tensiones que se haya seleccionado.

Finalmente, para el proyecto final nos hemos decantado por esta segunda opción, puesto que nos proporciona un mayor control de la tarjeta, pese a que su manejo no es tan trivial respecto a la Orocos Comedi Interface.

Para poder usarse esta librería, se ha tenido que modificar el fichero “Makefile” para indicarle las librerías dinámicas que debe incluir el proyecto (en nuestro caso, en la parte de librerías dinámicas, se han añadido las siguientes: -lcomedi -lm).

3.5.4 Modulo pcl833In

En el módulo “pcl833In” se pretendía comprobar el funcionamiento de la tarjeta de encódersAdvantech PCL 833, aunque uno de los principales problemas que se tuvieron fue la ausencia de drivers para Linux.

El objetivo de este módulo era capturar por los tres canales de entrada el valor de los encóders del robot (mediante un movimiento manual del mismo), y así comprobar el número de cuentas que se obtienen para, finalmente, hacer una conversión de cuentas de encóder a metros.

A continuación se explicará cómo se ha realizado la implementación de los drivers para la tarjeta y, posteriormente, cómo se ha realizado su integración con Orocos.

3.5.4.1 Drivers Linux PCL833

A diferencia de la tarjeta PCI 1720, para esta tarjeta de encóders no estaban implementados los drivers de Comedi. Por tanto, la única opción de poder manejar la tarjeta en Linux, era implementar los drivers para la misma.

Puesto que a esta tarjeta ISA se le puede especificar la dirección base mediante una serie de interruptores (véase la figura 40), su manejo no es otro que leer y escribir una serie de registros en función a esa dirección base. Si se conocen las técnicas de lectura y escritura de registros en Linux, la implementación del controlador no es demasiado complicada.

Range (hex)	Switch Position					
	1	2	3	4	5	6
200 - 20F(*)	X	O	O	O	O	O
210 - 21F	X	O	O	O	O	X
220 - 22F	X	O	O	O	X	O
230 - 23F	X	O	O	O	X	X
240 - 24F	X	O	O	X	O	O
...	...	...	...	...	...	...
3F0 - 3FF	X	X	X	X	X	X

X = On, O = Off, * = Default

Figura 40: Dirección base PCL 833

Por ello, se decidió implementar y documentar una librería en la que incluyera todas las funciones necesarias para un total manejo de la tarjeta de encóders, primeramente para usarla en nuestro proyecto y, posteriormente, distribuirla a diversos grupos que trabajen con esta tarjeta en Linux. Para la implementación de la misma, se ha usado el manual de la propia tarjeta para conocer la dirección de los registros.

Algunas de las funciones más importantes que se han implementado han sido las siguientes:

- **intvInitialize833().** En esta función se inicializan todas las variables, además de abrir los canales, aplicarles un reset, etc. Antes de usar la tarjeta es necesario llamar a esta función para un correcto funcionamiento.

- **intvCh_Read(intChannelNo).** Mediante esta función se consigue obtener el valor del encóder (en cuentas). Además, dependiendo del valor del parámetro "option", se consigue leer cualquiera de los tres canales.

- **intvCh_DefineResetValue(intChannelNo, int option).** Con este método, se puede hacer un reset del contador pero, dependiendo del parámetro "option",

el contador se puede resetear a la mitad de su valor máximo. De esta forma se consigue que si el valor del encóder se decrementa, no se produzca un "underflow" (que se produciría si se inicializara a 0).

Finalmente, cabe destacar que en la librería implementada existen muchas más funciones implementadas (seleccionar el modo cascada, el reloj, etc.), aunque en esta memoria sólo se han comentado las más significativas.

3.5.4.2 Integración con Orocos

En esta ocasión, puesto que la librería se ha implementado en C como parte de este proyecto, lo único que se ha tenido que realizar es incluir los fuentes de la librería dentro del código fuente del componente, de la siguiente manera.

```
#include"833DRIVE_linux.C"
```

De esta forma, en un componente de Orocos, se consigue llamar a todas las funciones que se han creado para manejar y configurar la tarjeta de encóders PCL 833.

Este componente de Orocos ha servido para comprobar el correcto funcionamiento de la tarjeta, así como para determinar la equivalencia entre cuentas de encóder y posición en metros.

Inicialmente, en la zona de configuración del componente, se debe inicializar la tarjeta y especificarle que haga un reset, considerando la mitad del valor máximo que admita la tarjeta (por la razón anteriormente expuesta).

```
iopl(3);
  Base= 0x0220;
  pcl833(Initialize833,NA);
  pcl833(Ch1_SetInputMode,x1);
  pcl833(Ch2_SetInputMode,x1);
  pcl833(Ch3_SetInputMode,x1);
  pcl833(Ch1_SetLatchSource,SwReadLatch);
  pcl833(Ch2_SetLatchSource,SwReadLatch);
  pcl833(Ch3_SetLatchSource,SwReadLatch);
  pcl833(Ch3_DefineResetValue,middle); //Reset a la mitad
  pcl833(Ch1_DefineResetValue,middle);
  pcl833(Ch2_DefineResetValue,middle);
  pcl833(LatchWhenOverflow,FreeAll);
```

```
  pcl833(Ch1_IfResetOnLatch,ResetNo);
pcl833(SetCascadeMode,c24bits);  //no cascade
```

Como se puede observar, antes de inicializar la tarjeta, se llama a la función específica de Linux “iopl(3)”. Mediante esta función, se consigue unos privilegios de superusuario para poder acceder a nivel de registro (de no llamar a esta función, se tendría un permiso denegado). Posteriormente se especifica la dirección base de la tarjeta (según se haya colocado los interruptores, anteriormente comentado) y se inicializa la misma. Por otro lado, tras inicializarla, se configuran los diferentes canales, siendo lo más significativo el reset que se hace a los tres canales (especificándole que se hace en el punto medio, mediante la variable “middle”).

Tras haber configurado la tarjeta, en cada iteración del componente (función “UpdateHook()”), se realiza una lectura de los canales (en el ejemplo siguiente sólo se hace la lectura del canal 1, aunque el resto de canales son similares) de la siguiente forma.

```
pcl833(Ch_Read,ch1);
ch1Dec=InReg[2]*65536 + InReg[1]*256 + InReg[0];
ch1Dec=ch1Dec-8388608;
posRealRobot[0]=ch1Dec*0.02/1000;
```

De esta manera, en la variable InReg[] se obtienen los tres bytes de la lectura que, tras multiplicarlos por potencias de 2 (dependiendo si es el bit de mayor o menor peso) se obtiene el valor de cuentas.

Posteriormente, a ese valor obtenido se le resta la mitad del valor máximo que puede contar la tarjeta. Esta resta se realiza porque al haber hecho previamente un reset al valor medio, un cero absoluto debería ser ese valor (8388608). Por tanto, al restar el valor obtenido por ese valor, se obtiene un valor absoluto de forma que si es negativo, el motor ha ido hacia abajo, y si es positivo, el motor ha ido hacia arriba (solucionando, por otro lado, el problema del “underflow”).

Finalmente, para realizar la conversión a metros (y así realizar todos los algoritmos usando las unidades del Sistema Internacional), de forma experimental y mediante la especificación de los encóders y motores usados se ha determinado lo siguiente.

$$1mm \cong 200\ cuentas\ del\ encóder$$

$$1\ cuenta\ de\ encóder = 0.02\ mm$$

$$N\ cuentas\ de\ encóder * \frac{0.02}{1000} = posición\ en\ metros$$

3.6 Modelos cinemáticos del robot 3-PRS

La elección de la arquitectura y movimiento del robot paralelo han venido determinadas por la necesidad de desarrollar un robot de bajo coste capaz de generar rotación angular en dos ejes (roll y pitch) y elevación como movimiento lineal. Se consideraron dos arquitecturas alternativas: 3-RPS y 3-PRS. Se seleccionó la arquitectura 3-PRS tras comparar las ventajas e inconvenientes de cada una de las alternativas. Por ejemplo, una de las ventajas de la arquitectura PRS es que los actuadores se localizan en la base fija mientras que en la arquitectura 3-RPS, los actuadores se mueven junto con las articulaciones de revolución.

La figura 41 muestra el modelo CAD del robot diseñado. El manipulador se puede ver como tres patas que conectan la plataforma móvil con la base. Cada pierna consiste en: 1) un motor, que mueve un husillo a bola, 2) un deslizador y 3) un vástago de conexión.

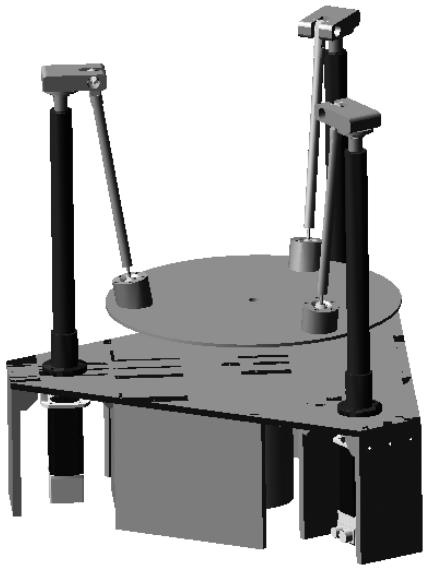

Figura 41: Modelo CAD robot 3-PRS

La cinemática directa del PM consiste en, dados los movimientos lineales de los actuadores, encontrar los ángulos de roll (γ) y pitch (β) y la elevación (z). Se puede utilizar la notación de Denavit-Hartenbert para establecer las coordenadas generalizadas del modelo cinemático. La tabla 3 muestra los parámetros D-H del robot considerado. A partir de la tabla se puede ver cómo, a partir de 9 coordenadas generalizadas, se puede definir la cinemática del robot. La localización de los sistemas de coordenadas para modelar la cinemática se muestra en la figura 42.

i	1	2	3	4	5	6	7	8	9
d_i	q_1	0	0	0	0	q_6	0	q_8	0
a_i	0	0	l_a	0	0	0	0	0	0
θ_i	π/6	q_2	q_3	q_4	q_5	5π/2	q_7	-π/2	q_9
α_i	0	π/2	0	π/2	π/2	0	π/2	0	π/2

Tabla 3: Parámetros D-H del robot

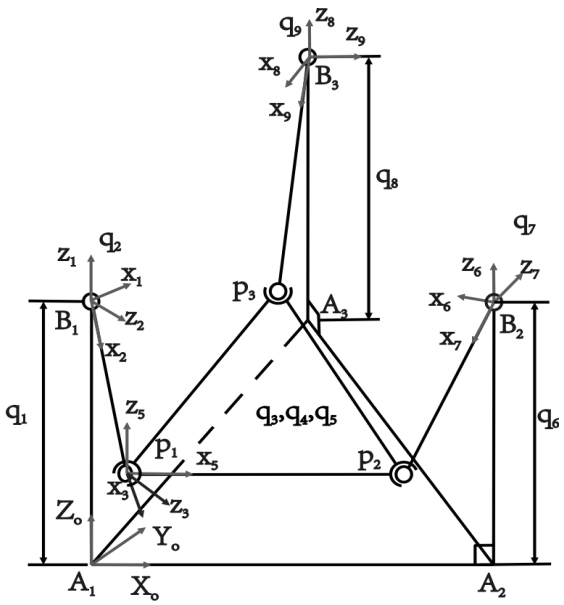

Figura 42: Coordenadas Robot

El robot que tiene 3-DOF, aplicando la aproximación geométrica, se puede ver que la longitud entre p_i y p_j es constante e igual a l_m. Así,

$$f_1(q_1,q_2,q_6,q_7)= \quad (1)$$
$$\left\|\left(\vec{r}_{A_1B_1}+\vec{r}_{B_1p_1}\right)-\left(\vec{r}_{A_1A_2}+\vec{r}_{A_2B_2}+\vec{r}_{B_2p_2}\right)\right\|-l_m=0$$

$$f_2(q_1,q_2,q_8,q_9)= \quad (2)$$
$$\left\|\left(\vec{r}_{A_1B_1}+\vec{r}_{B_1p_1}\right)-\left(\vec{r}_{A_1A_3}+\vec{r}_{A_3B_3}+\vec{r}_{B_3p_3}\right)\right\|-l_m=0$$

$$f_3(q_6,q_7,q_8,q_9)= \quad (3)$$
$$\left\|\left(\vec{r}_{A_1A_3}+\vec{r}_{A_3B_3}+\vec{r}_{B_3p_3}\right)-\left(\vec{r}_{A_1A_2}+\vec{r}_{A_2B_2}+\vec{r}_{B_2p_2}\right)\right\|-l_m=0$$

3.6.1 Cinemática directa

En la cinemática directa, la posición de los actuadores es conocida, así, el sistema de ecuaciones (1)-(3) se puede ver como un sistema no-lineal con q_2, q_7 y q_9 como desconocidas. Se escoge el método numérico de Newton para resolver el sistema no-lineal. El método es iterativo y se puede escribir como,

$$\begin{bmatrix} q_2 \\ q_7 \\ q_9 \end{bmatrix}^{i+1} = \begin{bmatrix} q_2 \\ q_7 \\ q_9 \end{bmatrix}^{i} - \mathrm{J}_i^{-1} \begin{bmatrix} f_1(q_2, q_7) \\ f_2(q_2, q_9) \\ f_3(q_7, q_9) \end{bmatrix}^{i} \quad (4)$$

En la ecuación *i* significa que las variables y funciones se evalúan en la iteración *i*. La matriz J es la matriz Jacobiana de f_i con respecto a las variables [q_2, q_7, q_9]. El proceso iterativo finaliza cuando,

$$\sqrt{\left(f_1(q_2,q_7)^i\right)^2 + \left(f_1(q_2,q_9)^i\right)^2 + \left(f_1(q_7,q_9)^i\right)^2} < \varepsilon \quad (5)$$

El parámetro ε es una cantidad positiva pequeña establecida por el usuario (en nuestro caso particular se ha especificado como 1e-8).

El método de Newton requiere una aproximación inicial tan cercana como sea posible al valor solución. En este caso, esto no supone ningún problema ya que la posición inicial de la articulación que conecta la plataforma con el actuador es aproximadamente 2π/5. La subsecuente aproximación inicial considera los valores de la posición previa del robot.

La localización de la plataforma móvil se define usando el sistema de coordenadas unido a él. Una vez encontradas las coordenadas generalizadas para las patas del robot, se puede encontrar la posición de los puntos p_i. Estos tres puntos comparten el plano de la plataforma. Basado en estos puntos, se puede construir la matriz rotacional de la plataforma con respecto a la base. Se define un eje local X_p como un vector unitario $\vec{u}$ con la dirección dada por p_1p_2. El eje Z_p se define mediante un vector $\vec{v}$ y es un eje perpendicular al plano definido por los puntos p_1, p_2 y p_3. Finalmente, el eje Y_p se define mediante la dirección de los ejes $\vec{w}$, el cual está determinado mediante el producto vectorial entre los ejes $\vec{u}$ y $\vec{v}$. La matriz de rotación de la plataforma móvil viene dada por,

$${}^{O}R_p = \begin{bmatrix} \vec{u}^T & \vec{v}^T & \vec{z}^T \end{bmatrix} \quad (6)$$

A partir de la matriz de rotación se pueden encontrar el resto de coordenadas generalizadas q_3, q_4, q_5.

3.6.2 Cinemática inversa

La cinemática inversa consiste en dado el giro (γ), el ángulo balanceo (β) y la elevación (z) encontrar el movimiento lineal de los actuadores. Usando el sistema de ángulos fijo X-Y-Z; la matriz de rotación se puede definir como:

$$ {}^{O}R_{p} = \begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta c_\gamma - s_\alpha s_\gamma \\ s_\alpha c_\beta & s_\alpha s_\beta s_\gamma - c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma \end{bmatrix} \quad (7) $$

Donde c_* y s_* hacen referencia al cos(*) y sen(*) respectivamente. Dados γ y β, el ángulo de cabeceo (α) se puede encontrar como sigue.

$$ \alpha = \operatorname{atan2}\left(s_\beta s_\gamma, \left(c_\gamma + c_\beta\right)\right) \quad (8) $$

Tras encontrar el ángulo α, se pueden encontrar el resto de términos de la matriz de rotación. Las posiciones de los actuadores se pueden encontrar mediante las siguientes expresiones [6],

$$ \begin{aligned} q_1 = {} & p_x{}^2 + p_y{}^2 + p_z{}^2 + 2h\left(p_x u_x + p_y u_y + p_z u_z\right) \\ & - 2g\,p_x - 2g\,h u_x + g^2 + h^2 \end{aligned} \quad (9) $$

$$ \begin{aligned} q_2 = {} & p_x{}^2 + p_y{}^2 + p_z{}^2 - h\left(p_x u_x + p_y u_y + p_z u_z\right) \\ & + \sqrt{3}h\left(p_x v_x + p_y v_y + p_z v_z\right) + g\left(p_x - \sqrt{3}p_y\right) \\ & + g\,h\left(u_x - \sqrt{3}u_y\right)/2 + g\,h\left(v_x - \sqrt{3}v_y\right)/2 + g^2 + h^2 \end{aligned} \quad (10) $$

$$q_3 = p_x{}^2 + p_y{}^2 + p_z{}^2 - h\left(p_x u_x + p_y u_y + p_z u_z\right) \quad (11)$$
$$-\sqrt{3}h\left(p_x v_x + p_y v_y + p_z v_z\right) + g\left(p_x - \sqrt{3}p_y\right)$$
$$-gh\left(u_x - \sqrt{3}u_y\right)/2 + gh\left(v_x - \sqrt{3}v_y\right)/2 + g^2 + h^2$$

En la ecuación, $h = l_m/\sqrt{3}$, $g = l_b/\sqrt{3}$, $p_x = -hu_y$, $p_y = -h\left(u_x - v_y\right)$, $p_z = z$ y l_b es la longitud entre $A_i A_j$.

Por tanto, mediante el roll, el pitch y la altura de la plataforma, se consiguen obtener las referencias q_1, q_2 y q_3 (a partir de las ecuaciones (9)-(11)).

3.7 Arquitectura de Hardware propuesto

En los posteriores apartados se comentará el hardware que se ha empleado en la realización de los experimentos con el robot 3-PRS. Además de las tarjetas que se han comentado en el apartado 3.4, se explicarán los diversos dispositivos hardware, así como su estructura y la conexión entre los mismos.

3.7.1 Diseño del Hardware

La configuración de la arquitectura de control del experimento puede observarse en la figura 43. En la imagen 44 se puede ver un esquema de la arquitectura empleada. El sistema de control consiste en tres amplificadores AEROTECH BA10, una tarjeta PCI (PCI 1720), una tarjeta ISA (PCL 833) y un equipo industrial (IEI RACK-360GW-R20).

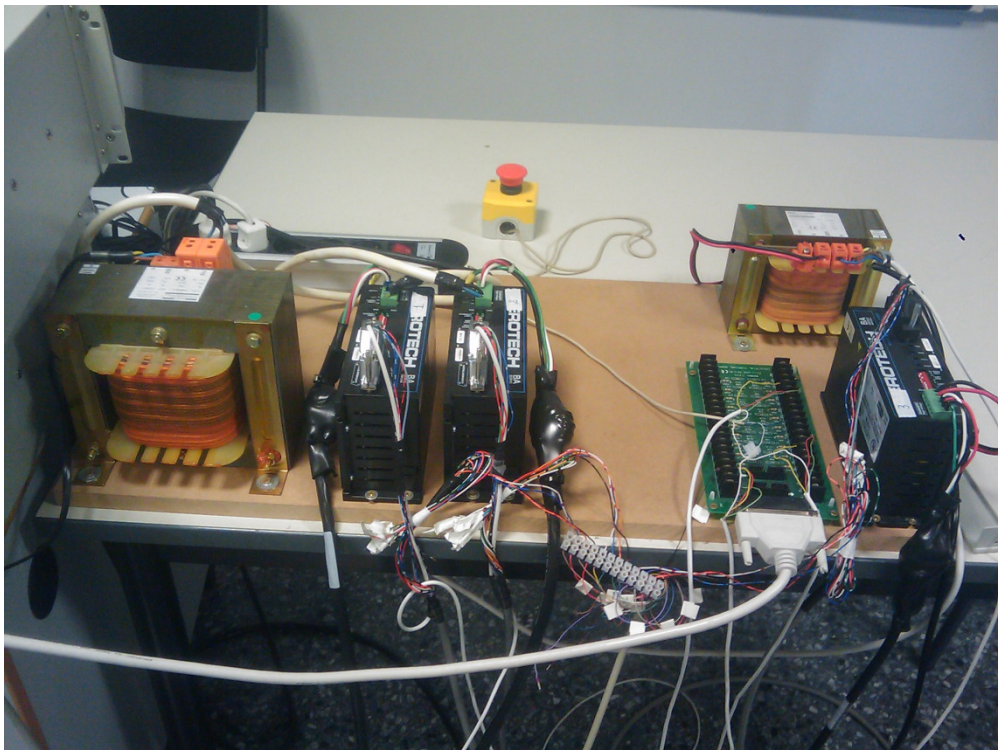

Figura 43: Foto Hardware

Los tres servomotores Aerotech BMS 100 UFA tienen la función de actuadores en el robot paralelo, siendo los sensores de posición los encóders. Cada servomotor está conectado a un amplificador y a las dos tarjetas. Por otro lado, la acción de control a aplicar se calcula a través de un algoritmo en el PC (y se saca el valor por la PCI 1720), mientras que mediante la tarjeta de encóders PCL 833 se lee el valor de los encóders y se envían al PC. Finalmente, los dos transformadores adaptan la tensión de la red eléctrica a una tensión adecuada para la alimentación de los amplificadores (el transformador pequeño alimenta a uno, mientras que el grande alimenta a los dos restantes).

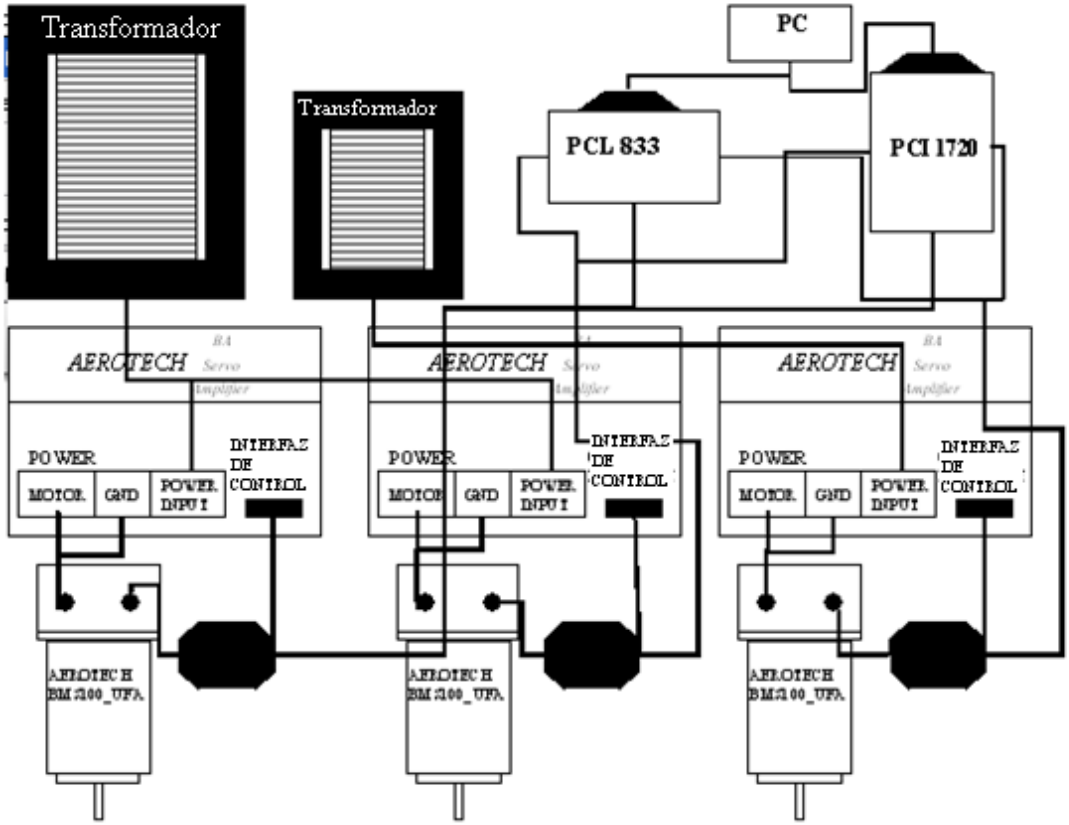

Figura 44: Arquitectura del Hardware

3.7.2 Componentes Hardware

Los componentes hardware se pueden clasificar en tres grupos:

- Sistema de control
- Actuadores y sensores
- Transformadores

En primer lugar, se comentará el hardware usado en el sistema de control. Posteriormente se abordará el tema de los actuadores y sensores, para finalizar con los transformadores.

Sistema de control

El hardware del sistema de control consiste en un PC Industrial, amplificadores y las dos tarjetas anteriormente mencionadas (la PCI y la ISA).

El PC Industrial es un Intel(R) Core(TM)2 Quad CPU Q8300 a 2.5 GHZ, con 4GB de memoria RAM. Es preciso un ordenador de estas características, ya que el algoritmo de control se implementa en él (requiriendo un gran número de operaciones).

Los amplificadores que se han usado han sido los Aerotech BA-10. La información sobre los mismos se han obtenido de los *data sheets* proporcionados por el fabricante ([7] y [8]).En términos generales, un amplificador es un dispositivo para amplificar y transmitir una señal eléctrica. La tarea de los amplificadores, con respecto al sistema de control, es alimentara los actuadores (es decir, los servomotores del robot). Las aplicaciones típicas son, por ejemplo, máquinas-herramienta, embalaje o equipamiento médico. En la tabla 4 se pueden observar las configuraciones para elAerotech BA-10.

Voltage estádar	Corriente de salidamáxima	Salida de corriente continua	Rango de voltaje DC
160V	10A	5A	80-100V, 80-160V, 80-320V

Tabla 4: Configuración Aerotech BA-10

La figura 45 ([7]) da una visión general del amplificadorAerotech BA-10. Consta de dos conexiones de potencia (potencia del motor y la potencia de entrada), cuatro potenciómetros, un interruptor DIP de 10 posiciones, un indicador de*LEDenable*, y un 25-pin conector.

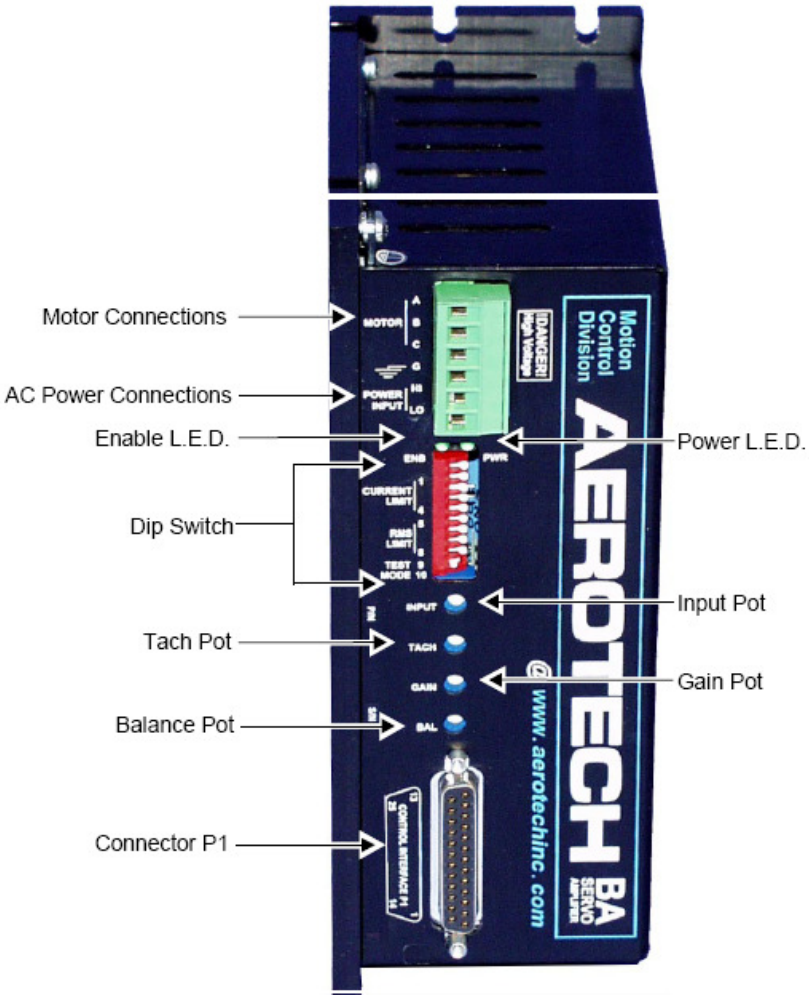

Figura 45: Aerotech BA 10

La figura 46 ([7]) da una vista más cercana del motor y de sus conexiones de alimentación. La potencia de entrada del amplificador se conecta a los terminales HI (línea) y LO (neutro), con la toma de tierra conectado a la G (masa). Los terminales del motor están conectados al amplificador en las conexiones A, B y C.

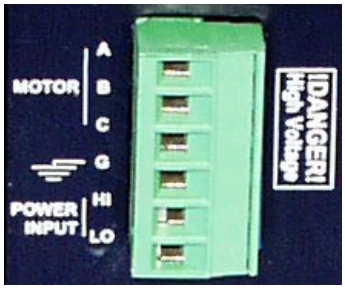

Figura 46: Conexiones de alimentación

Los10 interruptores de configuración del amplificador se muestran en la figura 47 ([7]). Ofrece cuatro funciones diferentes. Utilizando el interruptor, el usuario puede controlar el máximo de corriente que va al motor, la potencia máxima admisible de corriente continua, y si el amplificador está trabajando en el modo de velocidad o modo de prueba. De hecho, el usuario puede adaptar las configuraciones generales del amplificador a sus necesidades. Véase la tabla 1-2 de [7] para una descripción detallada de las funciones $\hat{I}_{\max} = 87\% \cdot I_{\max} = 8.7A$ de los interruptores. En nuestro proyecto, los interruptores 1, 3, 4, 5, 8 y 10 están cerrados, que implica:

- Corriente máxima:
- Corriente continua máxima: $I_{RMS,\max} = 30\% \cdot I_{\max} = 3A$

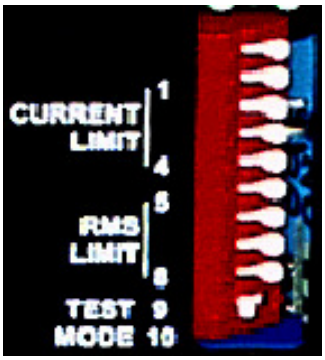

Figura 47: Posición de los interruptores

La figura 48 ([7]) muestra las entradas del potenciómetro del amplificador. El amplificadorAerotech BA10 contiene cuatro potenciómetros: INPUT, TACH (tacómetro), GAIN y BAL (balance). Los cuatro potenciómetros pertenecen al circuito pre-amplificador. Se utilizan cuando el amplificador trabaja en el modo de velocidad. Su tarea es, entonces, ajustar la ganancia del preamplificador. Dado que, en este proyecto, el amplificador no está en modo velocidad, los potenciómetros no son considerados. Para obtener más detalles acerca de las funciones potenciómetro véase la sección 1.4.3 del *data sheet* [7].

La figura 48 también incluye el conector P1. Actúa como interfaz de control, es decir, proporciona la interfaz para las acciones de control de entrada y salida.

Figura 48: Interfaz de control

Actuadores y sensores

Como actuadores, los elegidos han sido los Aerotech BMS100, servomotores sin escobillas (*brushless*)[8]. Realizan un movimiento de las articulaciones prismáticas. Para la sensorización de la posición de los mismos, se realiza mediante un encóder incremental (cada actuador lleva su propio encóder). La figura 49 muestra el motor utilizado.

Figura 49: Motor utilizado

Transformadores

En este experimento,los dos transformadores se utilizan para transformar la tensión de la red eléctrica a una tensión adecuada para los amplificadores. Como se ha vistoen la tabla 4, los amplificadores aceptan una tensión de entrada de 160V, pero la red eléctrica suministra 220V.

Como se ha comentado anteriormente, el transformador grande suministra la tensión a dos amplificadores. LaTabla 5 revela la tensión y la corriente.

	Bobinado primario	**Bobinado secundario**
Voltaje	$V_P = 220V$	$V_S = 110V$
Corriente	$I_P = 9.23A$	$I_S = 18.18A$

Tabla 5: Transformador grande

La frecuencia de trabajo es de 50 Hz y la potencia suministrada es de 2000W.

Por otro lado, el transformador pequeño suministra la tensión al tercer amplificador. En este caso, la potencia suministrada es de 800W. En la tabla 6 se puede observa la tensión y la corriente.

	Bobinado primario	**Bobinado secundario**
Voltaje	$V_P = 220V$	$V_S = 110V$
Corriente	$I_P = 3.73A$	$I_S = 7.27A$

Tabla 6: Transformador pequeño

En la siguiente figura se pueden observar los dos transformadores utilizados en el experimento.

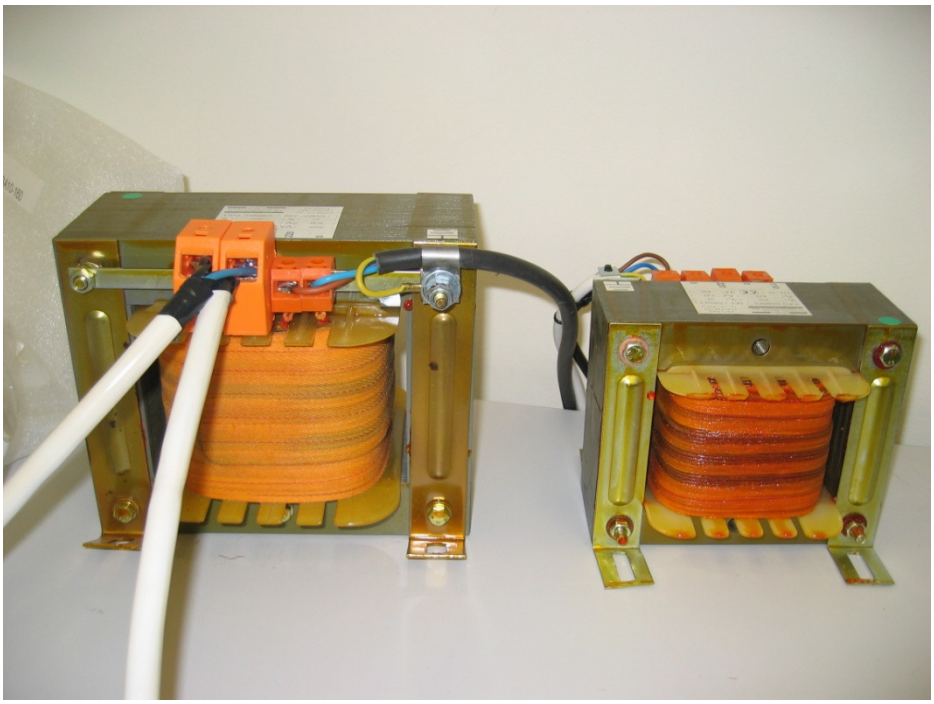

Figura 50: Transformadores utilizados

3.8 Algoritmos de control

Tras haber realizado toda la interconexión del hardware, y haberlo integrado con el proyecto Orocos, apoyándonos de con módulos de Orocos se van a implementar tres tipos de controladores (con el objetivo de observar el comportamiento del robot).

Los controladores implementados han sido controladores basados en pasividad. Esta aproximación resuelve el problema de controlar un robot mediante la explotación de la estructura física del sistema robotizado, y en especial de su propiedad de pasividad. La filosofía de diseño de estos controladores es reconfigurar la energía natural del sistema de tal forma que se consiga el objetivo de seguimiento del control [9].

En el problema de punto-a-punto (regulación), los controladores basados en pasividad se pueden ver como casos particulares de la siguiente ley de control general:

$$\tau_e = -K_p e - K_d v - u \quad (12)$$

Donde $e = q - q_d$ y u y v varían dependiendo de la clase de controlador:

Controlador(problema punto-a-punto)	***U***	***V***
PD+G	$-G(q)$	$\dot{q}$
PID	$K_i \int_0^t e \, \mathrm{dt}$	$\dot{q}$

Tabla 7: Controladores punto a punto

Finalmente, se ha implementado el controlador de Paden, el cual es una modificación del PD+G, en el que se contempla la velocidad y la aceleración en el algoritmo de control. Sigue la siguiente ley de control general:

$$\tau_e = M(q)a + C(q,\dot{q})v_1 + G(q) - K_p e - K_d v_2 \quad (13)$$

Donde v_1, v_2 y $e$varían dependiendo de la clase de controlador:

Controlador	***A***	v_1	***A***	***v***
Paden, Panja	$\ddot{q}_d$	$\dot{q}_d$	$q - q_d$	$\dot{e}$

Tabla 8: Controlador Paden

3.8.1 Control Pasivo PD+G

El primer controlador implementado ha sido un PD con compensación de la gravedad. Este controlador está formado por dos partes: la primera es una realimentación lineal del estado y la segunda es la compensación de fuerzas de la gravedad. En la figura 51 (en general) y 52 (específicamente el control), se puede apreciar su esquema en *Matlab-SImulink*.

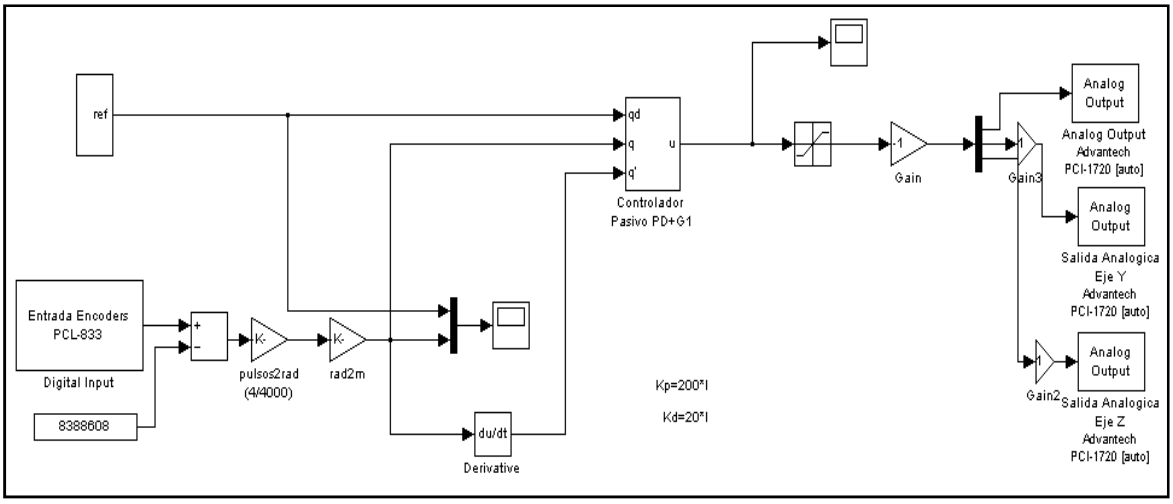

Figura 51: Modelo control PD+G general

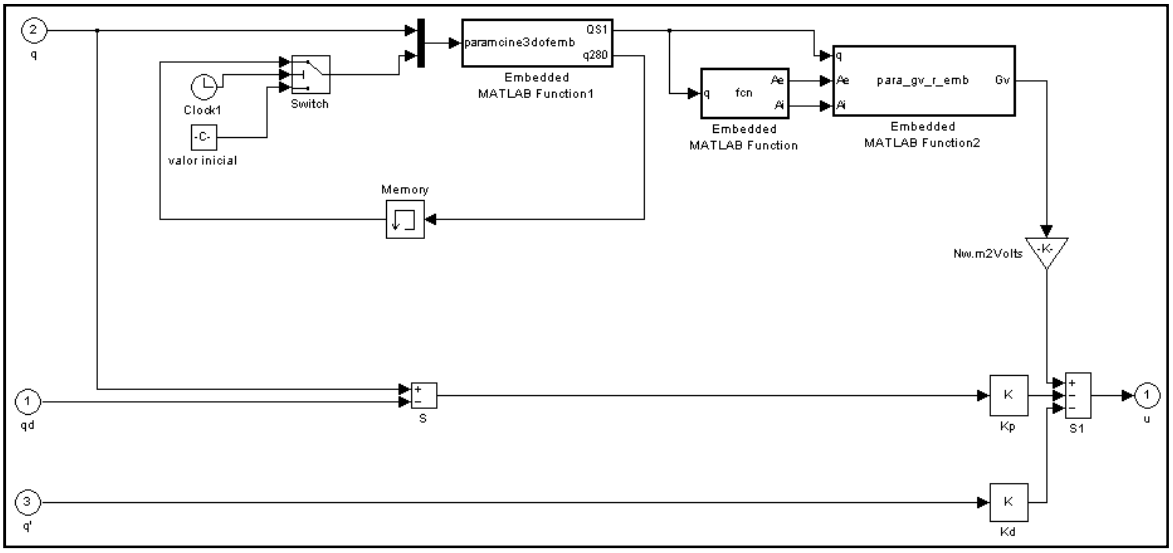

Figura 52: Modelo control PD+G detallado

Este tipo controlador es muy simple pero tiene el inconveniente de gran la complejidad computacional del término gravitatorio. Dependiendo del robot y/o su modelo dinámico, ésta puede ser tan alta que sea imposible calcularlo en tiempo real.

Para comprobar su correcto funcionamiento, se han considerado tres ficheros de referencia, usando el *roll, pitch y la zeta* (véase apartado 3.6.2).

1. En la referencia 1, la *gamma y beta* son iguales a cero, mientras que la *zeta* va variando.

2. La referencia 2 mantiene la *beta* a cero, mientras que las otras dos componentes van variando su valor.
3. En la referencia 3, la *gamma* vale cero y el resto de valores van cambiando.

La realización del movimiento se compone de tres partes diferenciadas:

- Un *spline cúbica natural* desde el punto de origen del robot hasta la primera referencia generada. Este movimiento se realiza en 8 segundos, más 1 segundo en el que el robot se mantiene estable en esa posición (a modo de tiempo de seguridad).
- El movimiento central, en el que en cada iteración se va cogiendo una referencia (que se va generando en cada ciclo).
- Cuando se llega a la última referencia generada, se realiza otra *spline cúbica natural* desde esa referencia hasta el origen del robot. De la misma forma que la primera *spline*, este movimiento se realiza en 8 segundos, con 1 segundo en el que la referencia se mantiene constante.

A continuación se puede observar enlas siguientes gráficas, para cada articulación y para cada uno de los 3 movimientos realizados, la relación de la referencia introducida al robot y la respuesta obtenida. Por otro lado, para cada movimiento, se puede analizar la acción de control aplicada.

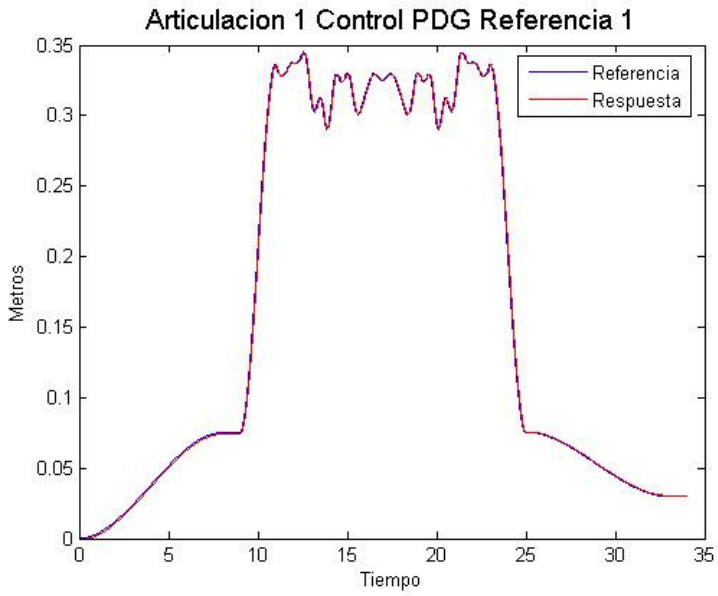

Figura 53: Articulación 1. Control PDG. Referencia 1.

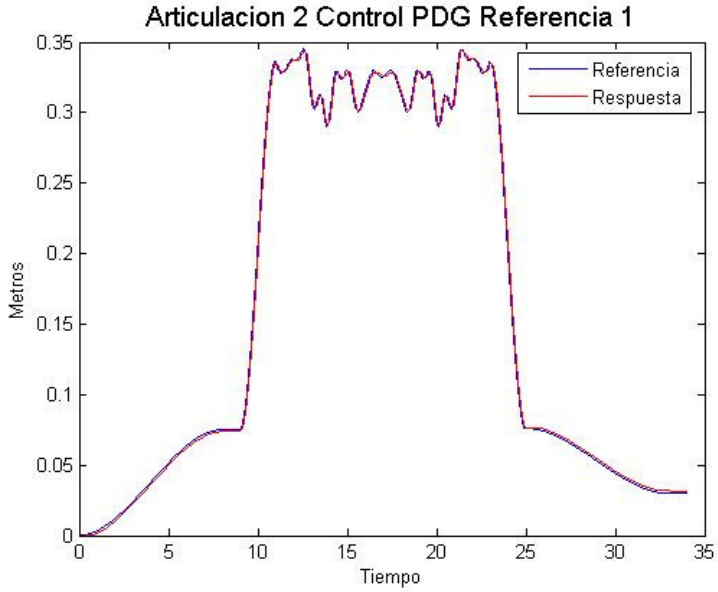

Figura 54: Articulación 2. Control PDG. Referencia 1.

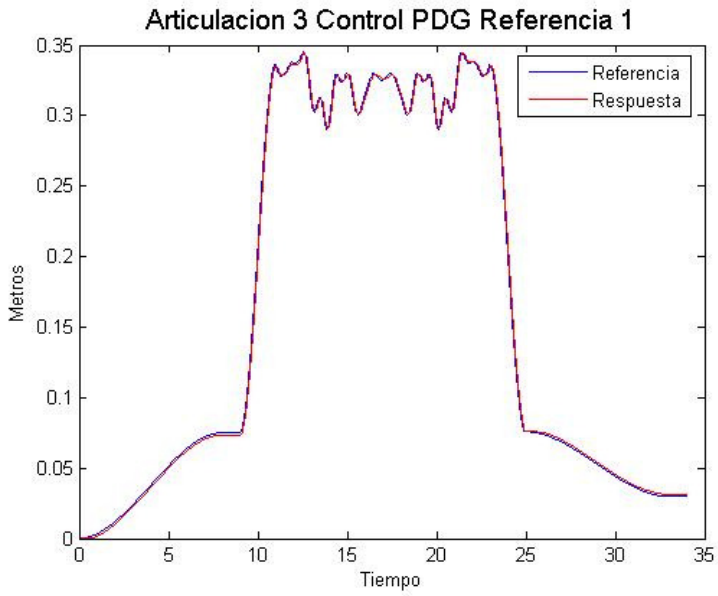

Figura 55: Articulación 3. Control PDG. Referencia 1.

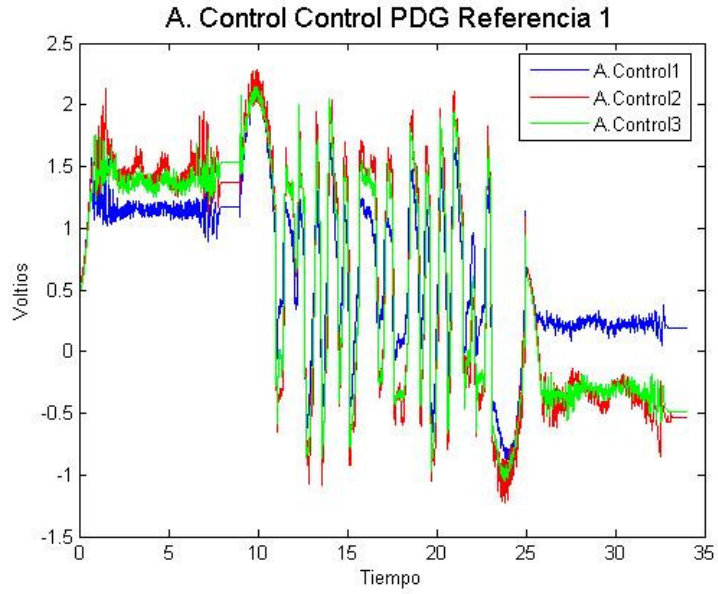

Figura 56: Acciones de control. Control PDG. Referencia 1.

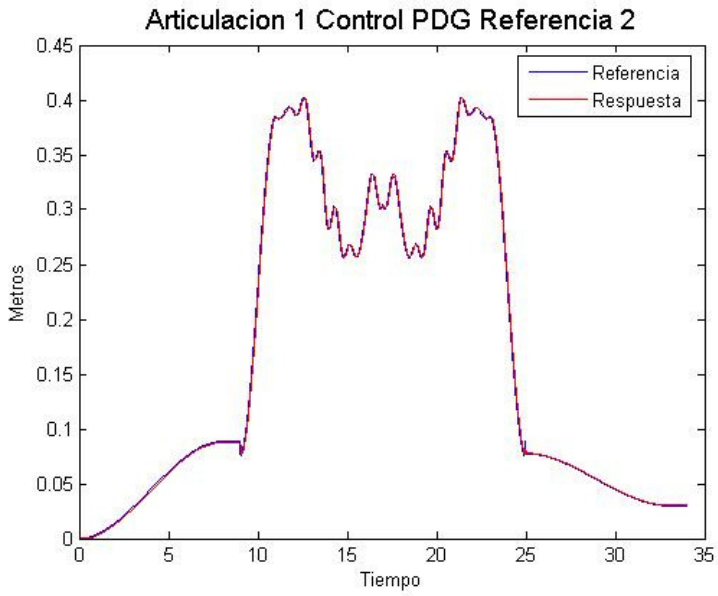

Figura 57: Articulación 1. Control PDG. Referencia 2.

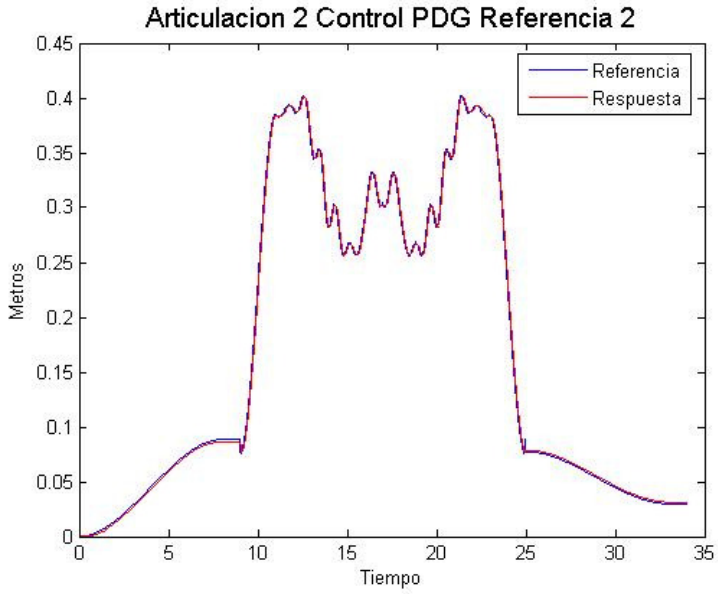

Figura 58: Articulación 2. Control PDG. Referencia 2.

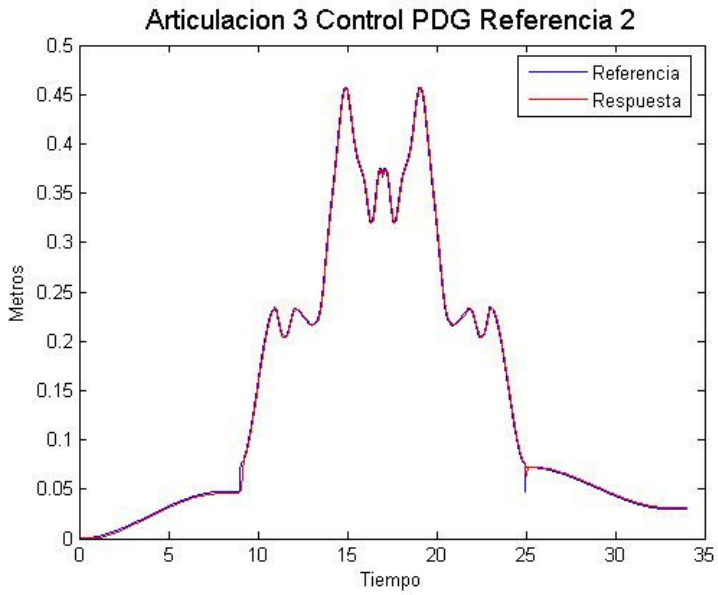

Figura 59: Articulación 3. Control PDG. Referencia 2.

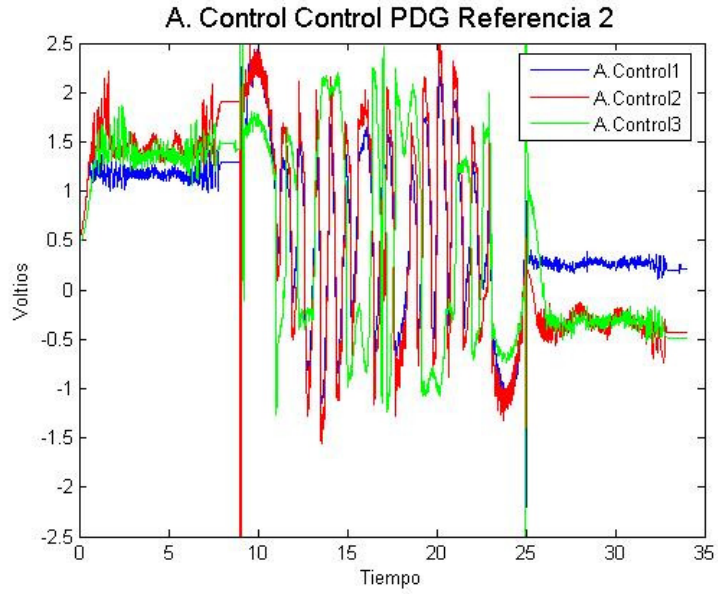

Figura 60: Acciones de control. Control PDG. Referencia 2.

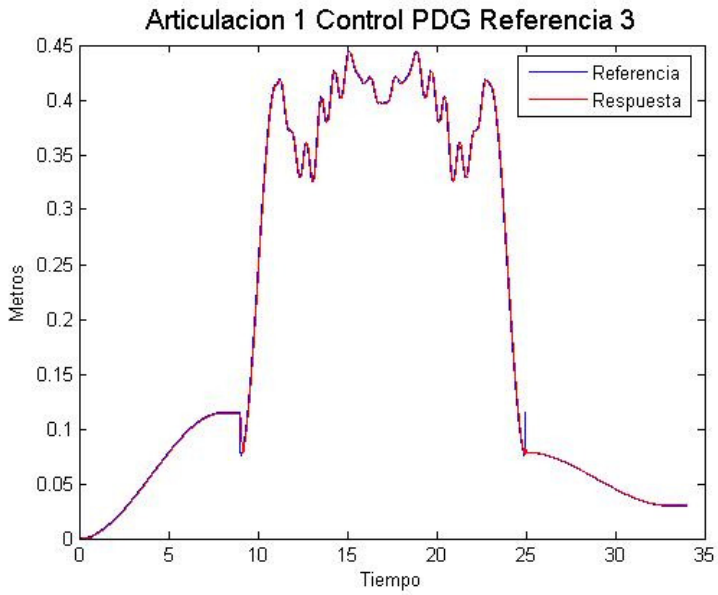

Figura 61: Articulación 1. Control PDG. Referencia 3.

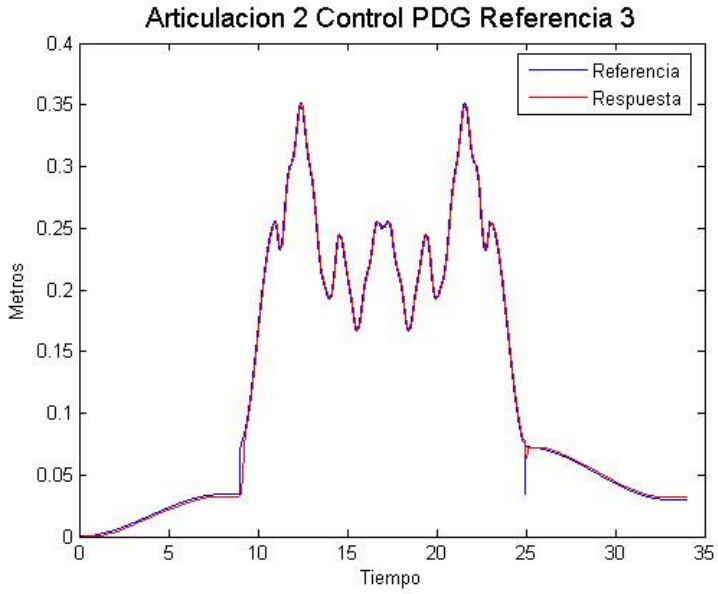

Figura 62: Articulación 2. Control PDG. Referencia 3.

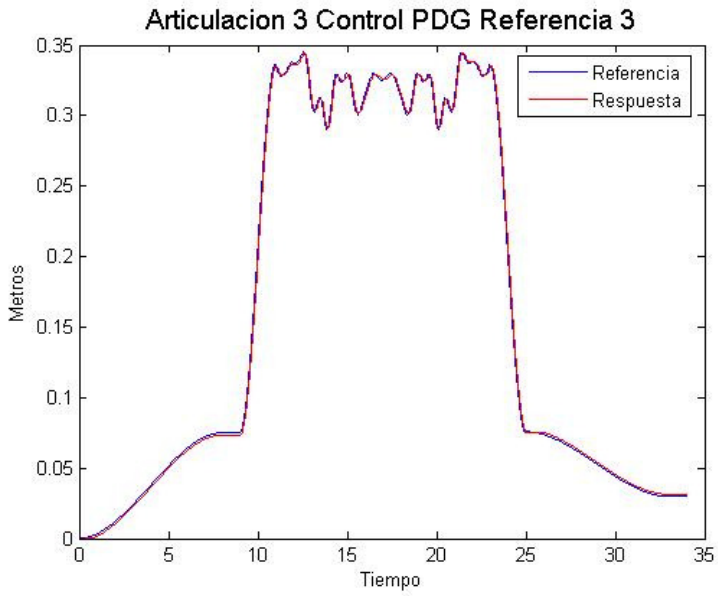

Figura 63: Articulación 3. Control PDG. Referencia 3.

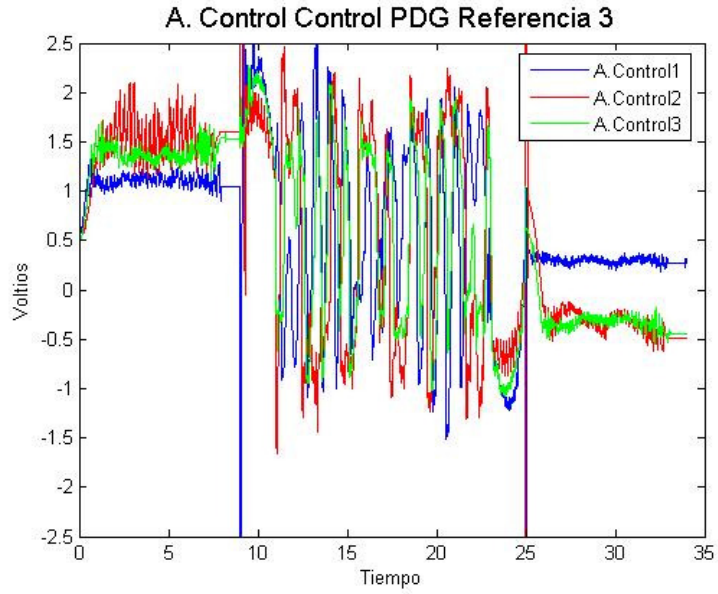

Figura 64: Acciones de control. Control PDG. Referencia 3.

Como se aprecia en las anteriores gráficas, la respuesta del sistema (posición real del robot) es prácticamente igual a la referencia. Por otro lado, por ejemplo en el movimiento 1, pese a que la referencia es la misma para los tres ejes, las acciones de control varían de un eje a otro. Esto es debido a que la fricción no es igual en todos ellos (como es normal).

3.8.2 Control Pasivo PID

Una solución práctica para intentar solucionar el problema de la complejidad en el controlador PD+G, es insertar una acción integral en la ley de control. Estas leyes son básicamente las mismas que en el PD pero la compensación de la gravedad se ha sustituido por la integral del error. Se puede ver un esquema en la figura 65 y 66.

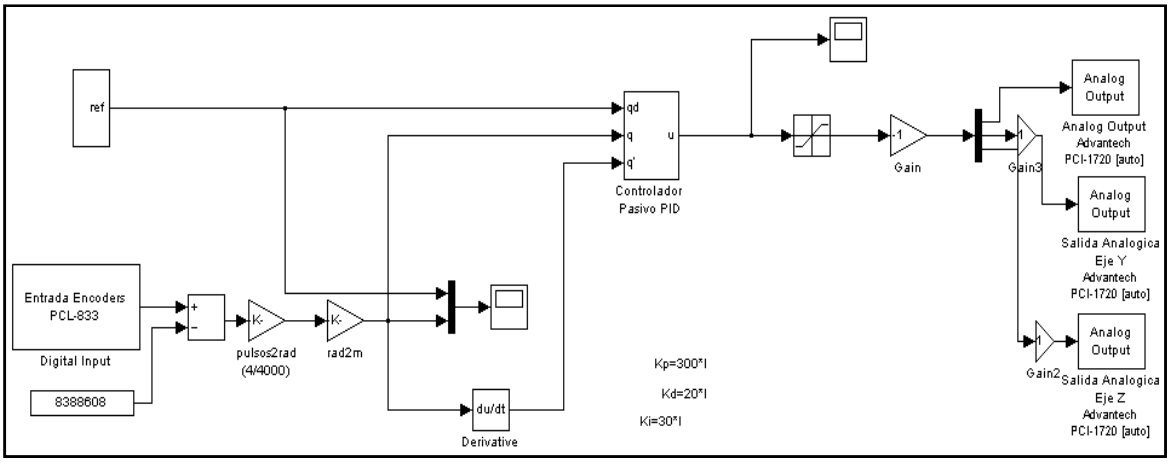

Figura 65: Modelo control PIDgeneral

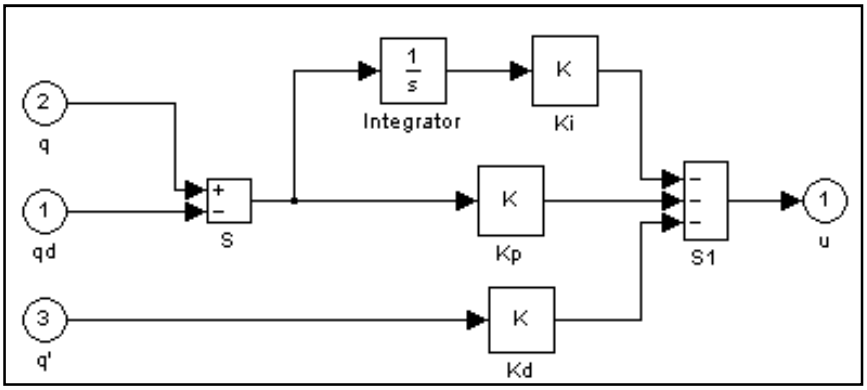

Figura 66: Modelo control PIDdetallado

De igual forma que en el apartado anterior, se han realizado los mismos movimientos de prueba, cuyos resultados pueden observarse en las siguientes gráficas.

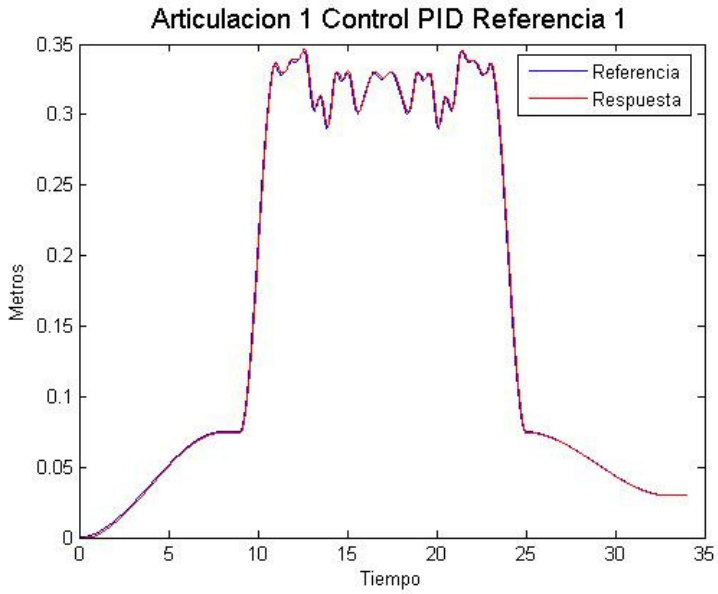

Figura 67: Articulación 1. Control PID. Referencia 1.

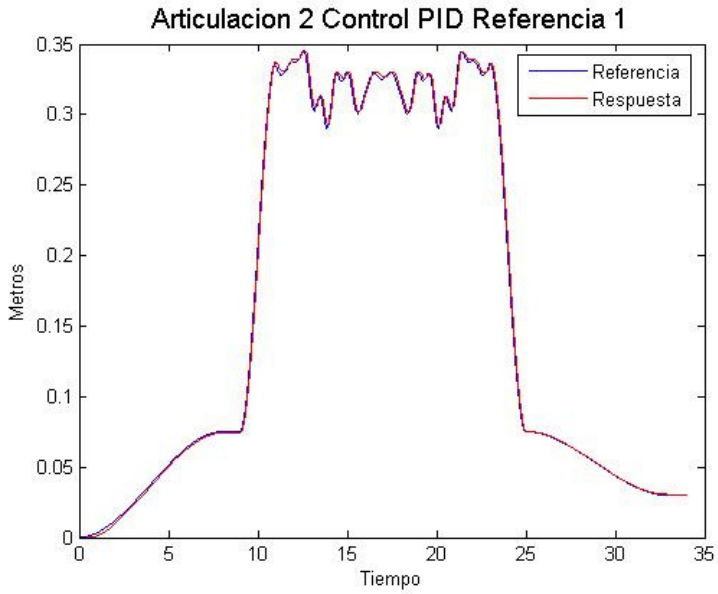

Figura 68: Articulación 2. Control PID. Referencia 1.

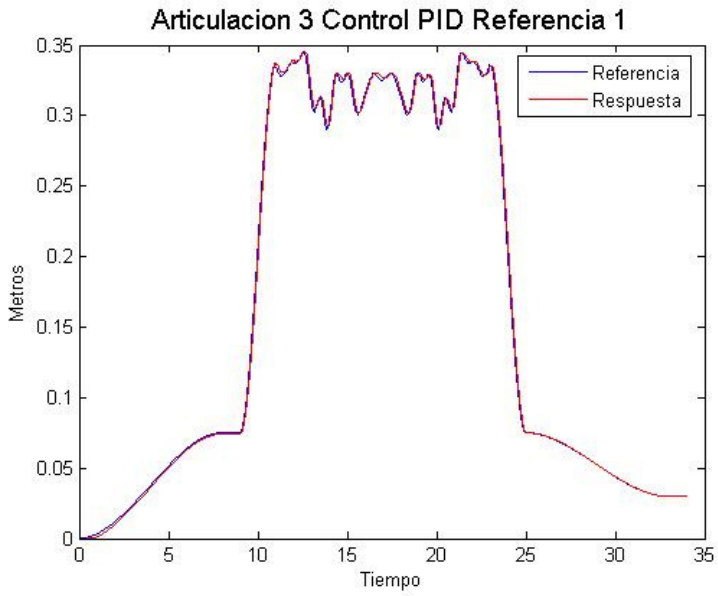

Figura 69: Articulación 3. Control PID. Referencia 1.

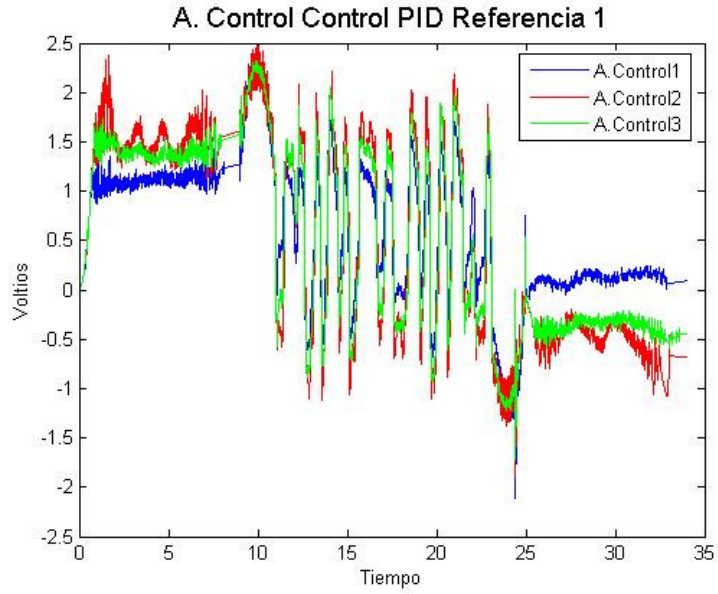

Figura 70: Acciones de control. Control PID. Referencia 1

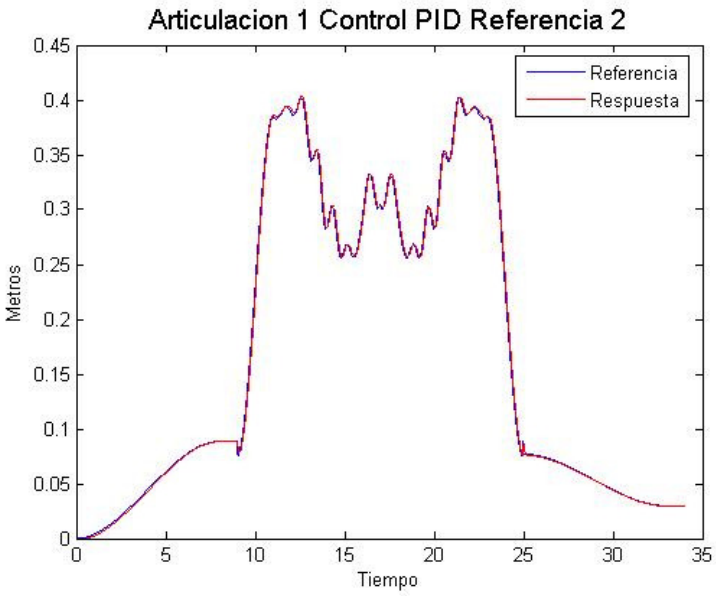

Figura 71: Articulación 1. Control PID. Referencia 2.

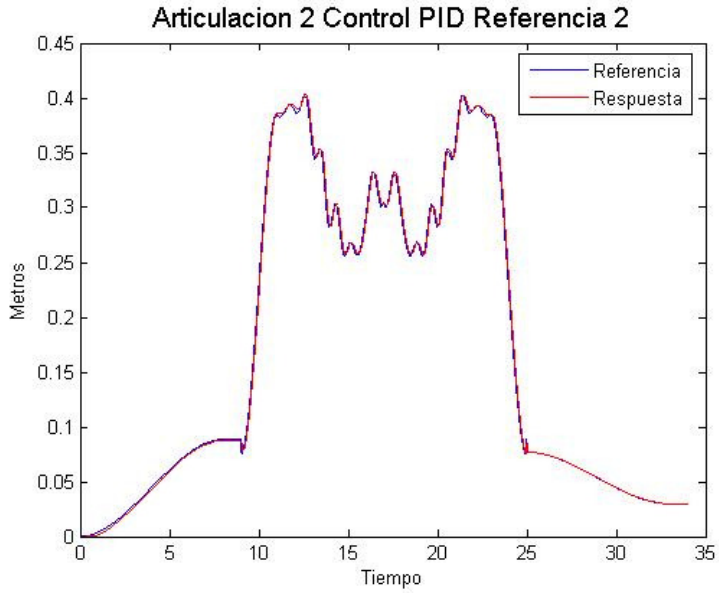

Figura 72: Articulación 2. Control PID. Referencia 2.

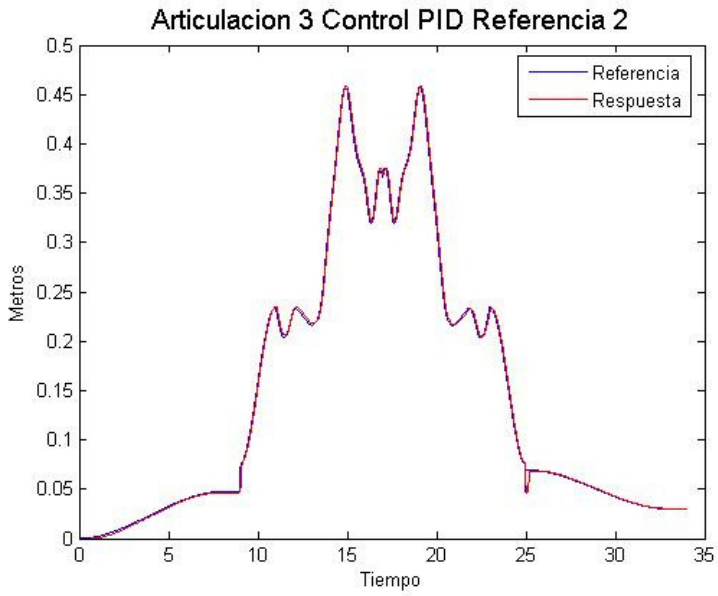

Figura 73: Articulación 3. Control PID. Referencia 2.

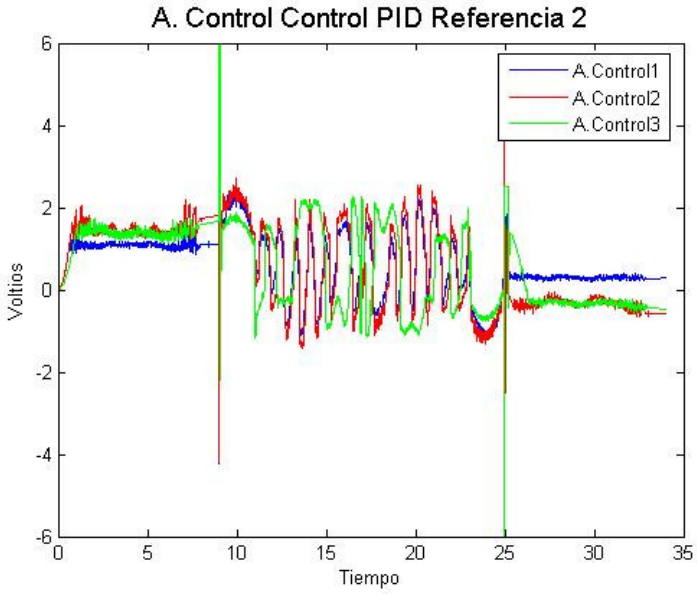

Figura 74: Acciones de control. Control PID. Referencia 2.

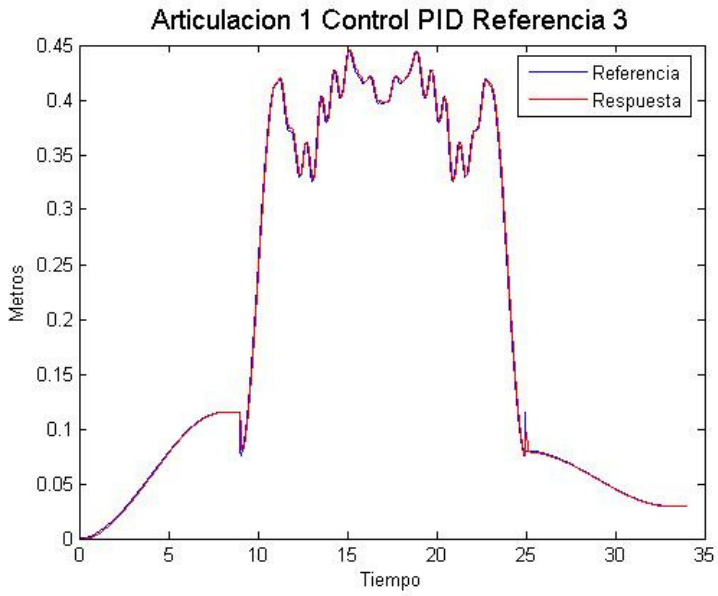

Figura 75: Articulación 1. Control PID. Referencia 3.

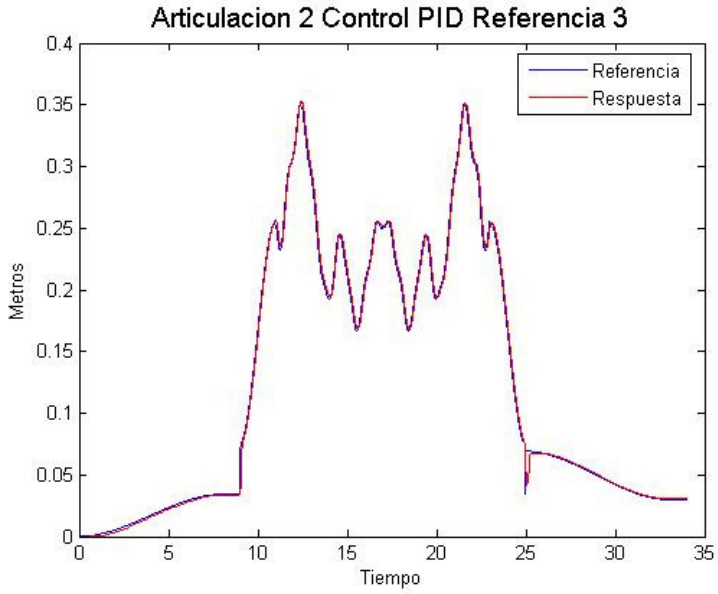

Figura 76: Articulación 2. Control PID. Referencia 3.

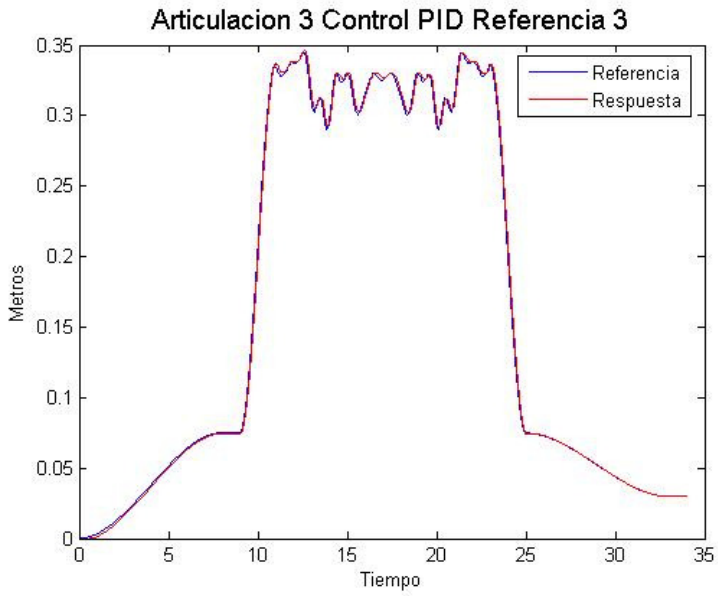

Figura 77: Articulación 3. Control PID. Referencia 3.

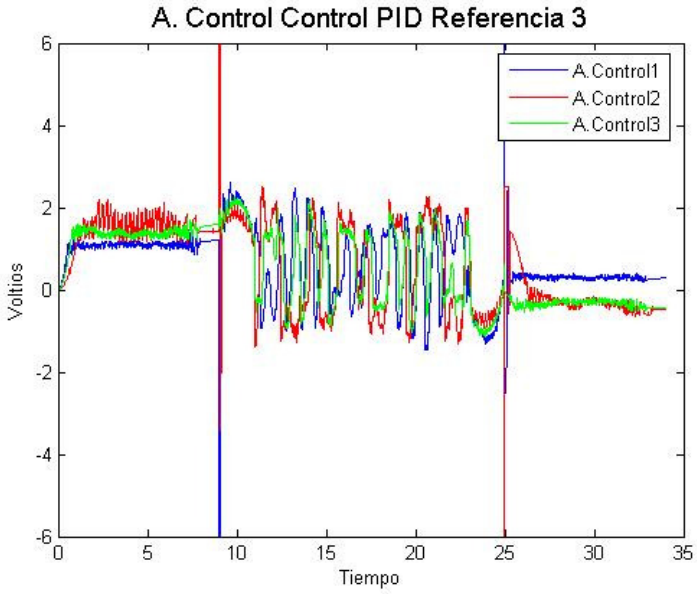

Figura 78: Acciones de control. Control PID. Referencia 3.

3.8.3 Control Pasivo Paden

Finalmente, se implementó un control pasivo de *Paden.* El controlador es muy similar al PD+G, aunque se consideran, además, los parámetros de la velocidad real, la velocidad de referencia, y la aceleración de la referencia.

A continuación se pueden ver los esquemas de *Matlab-SImulink*(figura 79 y 80)*,* así como las diferentes respuestas del sistema al aplicarle los tres movimientos anteriormente descritos.

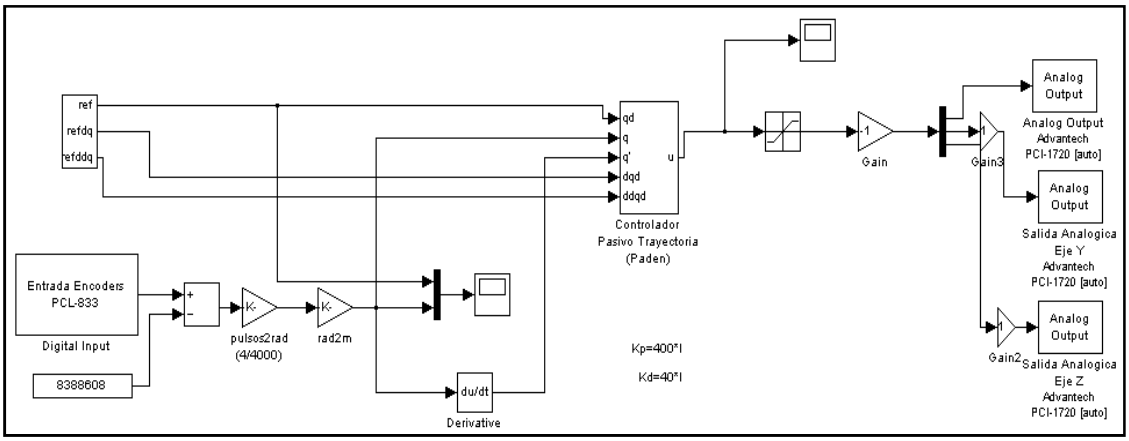

Figura 79: Modelo control Paden general

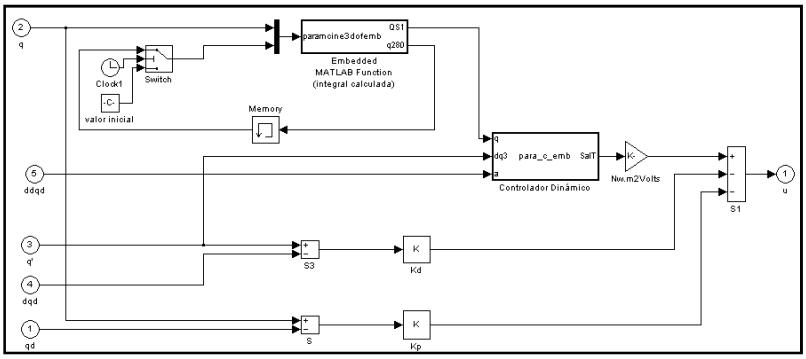

Figura 80: Modelo control Paden detallado

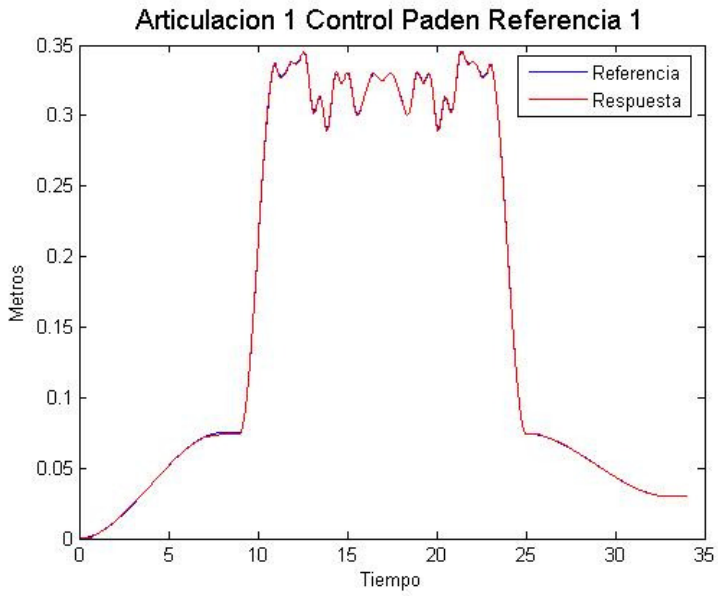

Figura 81: Articulación 1. Control Paden Referencia 1.

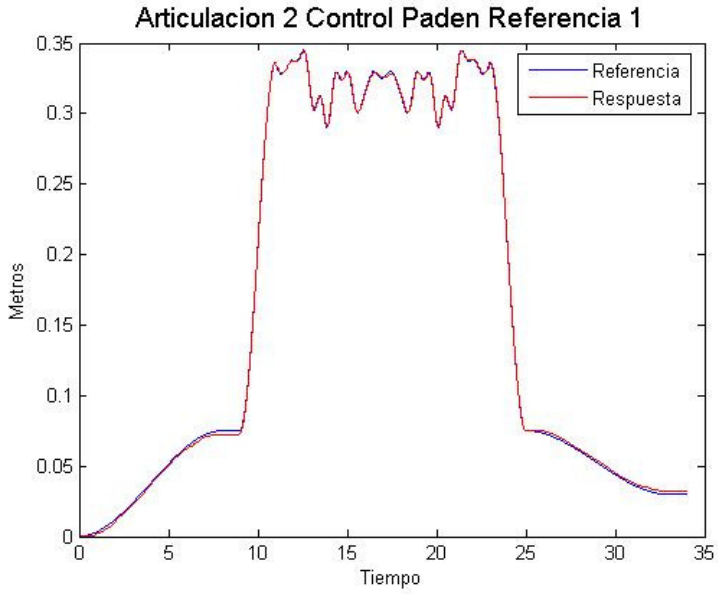

Figura 82: Articulación 2. Control Paden Referencia 1.

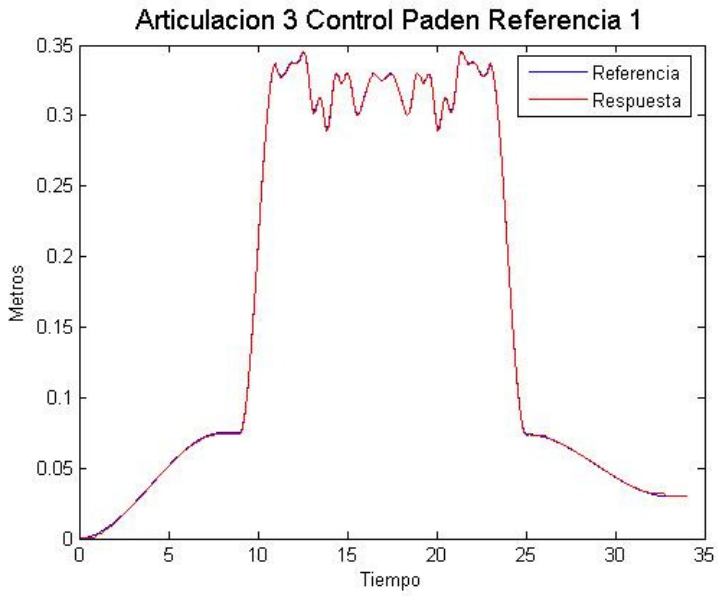

Figura 83: Articulación 3. Control Paden Referencia 1.

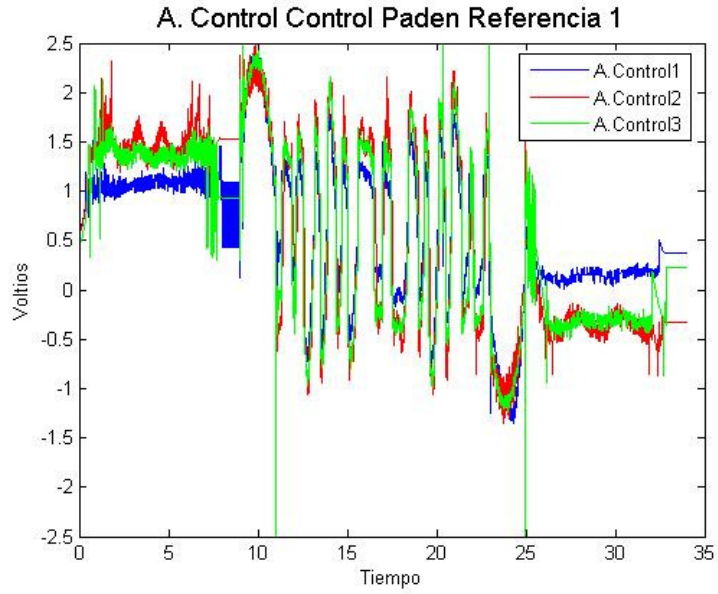

Figura 84: Acciones de control. Control Paden Referencia 1.

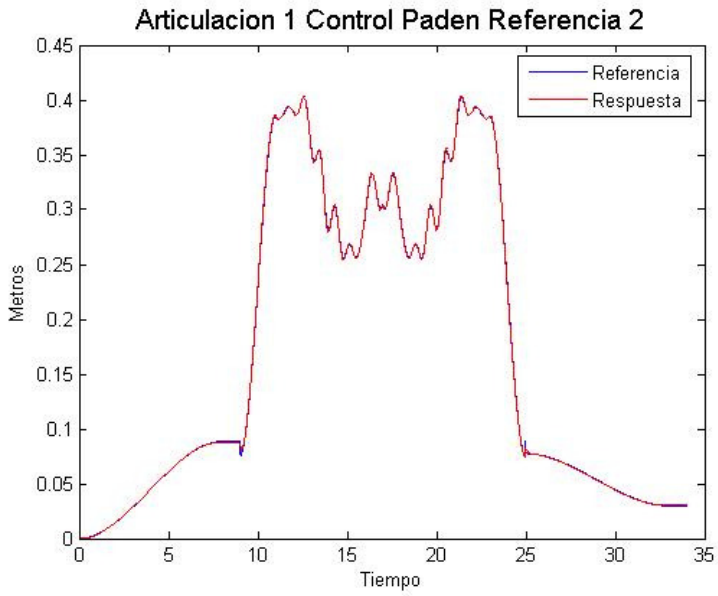

Figura 85: Articulación 1. Control Paden Referencia 2.

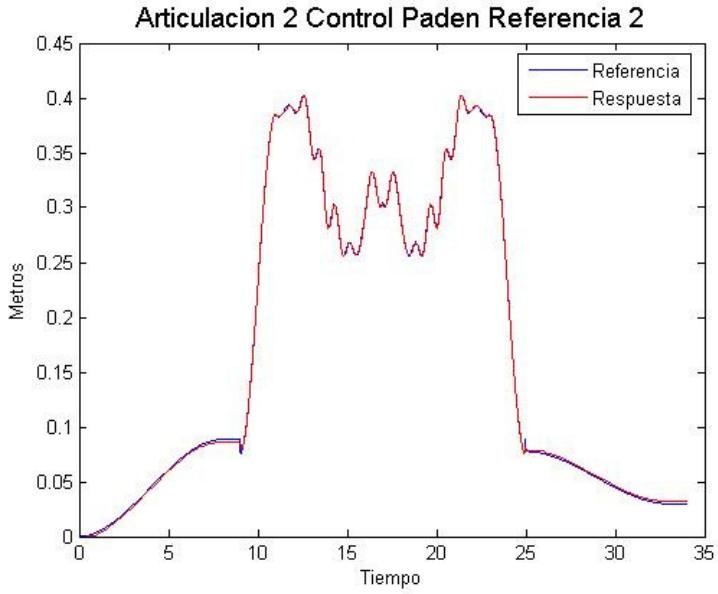

Figura 86: Articulación 2. Control Paden Referencia 2.

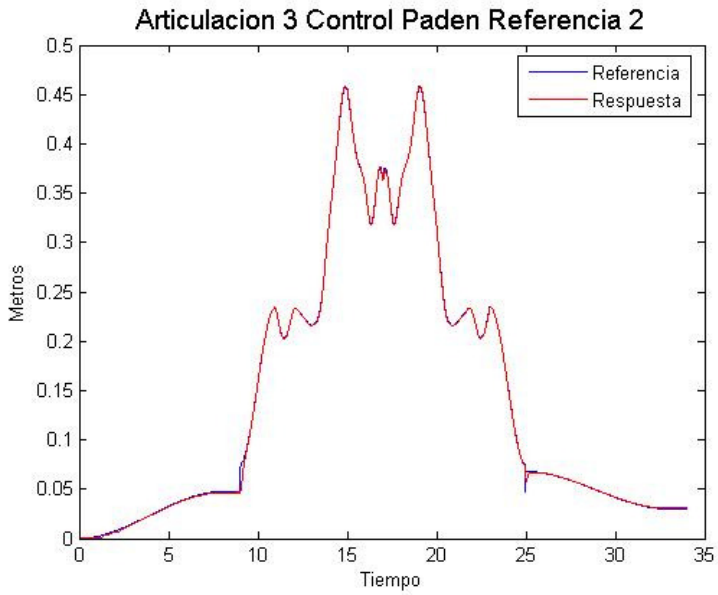

Figura 87: Articulación 3. Control Paden Referencia 2.

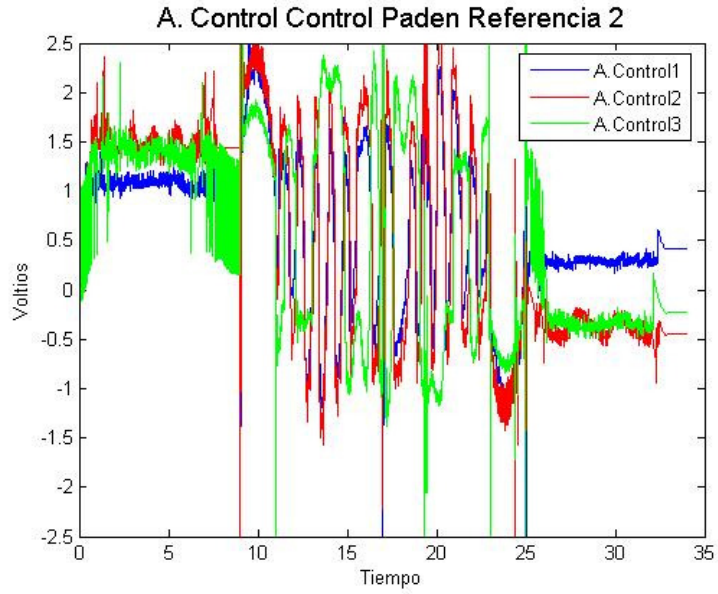

Figura 88: Acciones de control. Control Paden Referencia 2.

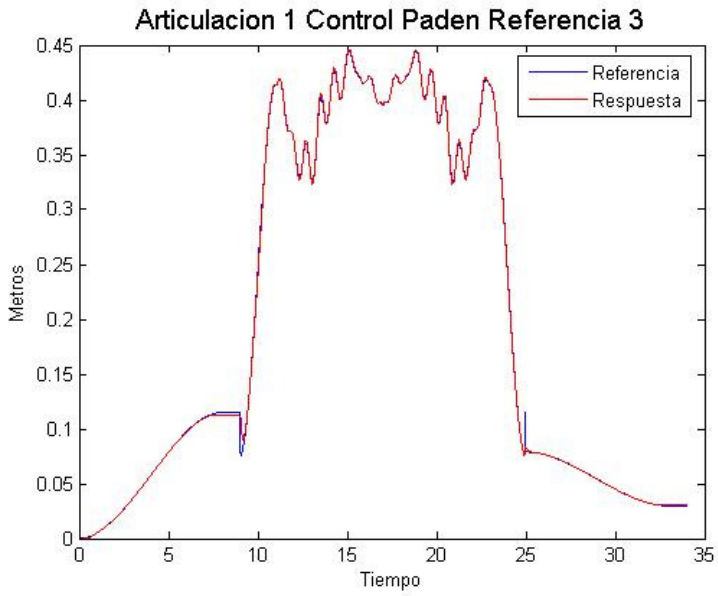

Figura 89: Articulación 1. Control Paden Referencia 3.

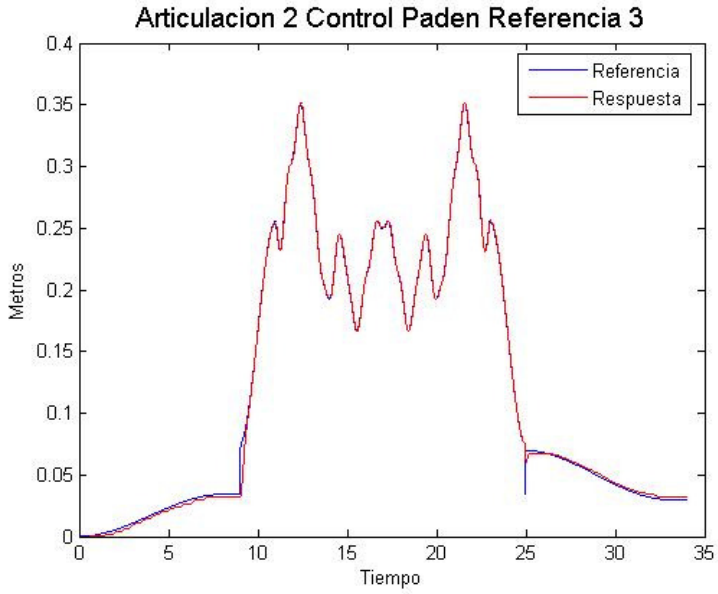

Figura 90: Articulación 2. Control Paden Referencia 3.

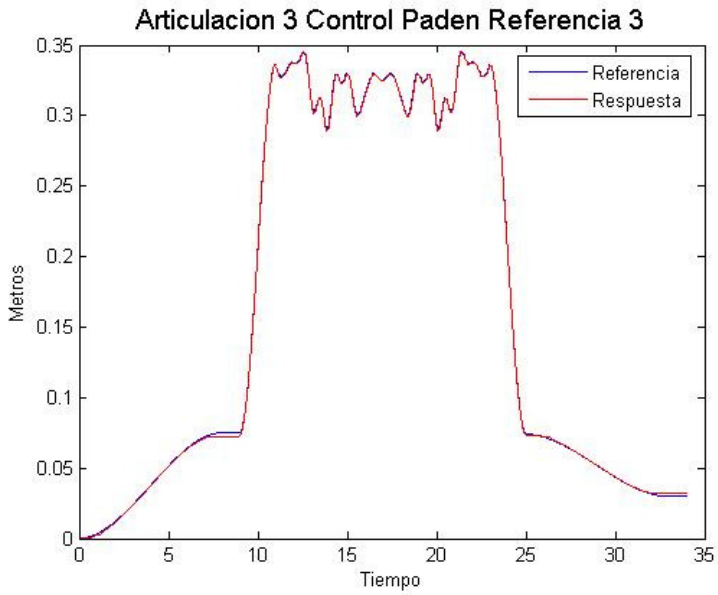

Figura 91: Articulación 3. Control Paden Referencia 3.

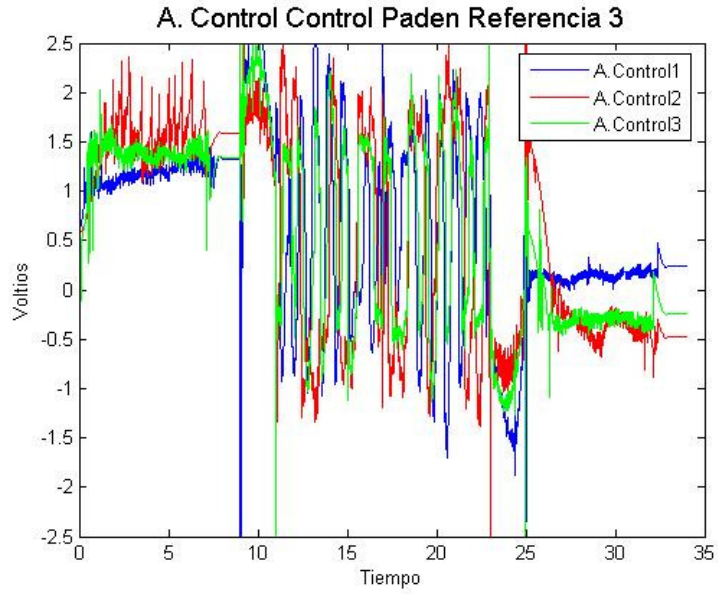

Figura 92: Acciones de control. Control Paden Referencia 3.

4 Conclusiones y trabajos futuros

Con la realización de este proyecto se ha conseguido entender y profundizar conceptos tanto relacionados con la robótica en sí, como relacionados con la programación de aplicaciones en tiempo real.

Inicialmente, se han creado diferentes módulos usando la librería Orocos, de forma que hemos aprendido a utilizar esta nueva forma de programación. Además, mediante la implementación de un buen número de componentes de diferentes tipos, se ha conseguido simular el comportamiento más básico de un robot.

Por otro lado, para la integración del hardware con nuestro proyecto, al tratarse de un Linux en tiempo real, hemos aprendido a ayudarnos de librerías como Comedi, y realizar drivers para una tarjeta de encóders ISA, alojando los mismos en un repositorio público con el objetivo de que otro grupo de investigación los pueda utilizar.

Una vez se dominaron suficientemente los módulos de Orocos, mediante unos esquemas en *Matlab-Simulink,* conseguimos implementar en C++ (y con Orocos) un control PID. Una de las problemáticas fue la realización del algoritmo con la estructura de un componente de Orocos, que se consiguió cambiando el estilo de programación. Tras el control PID, implementamos un control con compensación de la gravedad (PDG) y el control pasivo de Paden.

Finalmente, los controladores anteriormente implementados mediante Orocos, se verificaron con el robot real 3-PRS, aplicándole 3 referencias distintas, obteniendo unos resultados exitosos.

Como proyectos futuros, se podrían implementar más controladores para el robot como, por ejemplo, controladores basados en la dinámica inversa del mismo. Además, generando referencias y controles apropiados, el robot podría ser utilizado en tareas de fisioterapia y en simulaciones de vuelo.

Para añadir más funcionalidad al prototipo, se podría introducir un sensor de fuerza (como el JR3) y diseñar un control en el que se tenga en cuenta el valor del sensor en el transcurso de la ejecución.

Como último apunte, cabe destacar que próximamente se seguirá trabajando en este prototipo a partir de lo realizado en este proyecto (y con la plataforma Orocos), de forma que en un futuro no muy lejano, el robot adoptará una mayor funcionalidad y control del que posee en la actualidad.

5 Referencias

[1] Tsai, Lung-Wen: Robot Analysis – The Mechanics of Serial and Parallel Manipulators; John Wiley & Sons; 1999

[2] Lee, K.-M. and Shah, D. K.: Kinematic Analysis of a Three-Degree-of-Freedom In-Parallel Actuated Manipulator; IEEE Journal of Robotics and Automation, 4(3): 354-360; 1988

[3] Díaz Rodríguez, M. Á. (Profesor: Mata Amela, V.): Identificación de Parámetros Dinámicos de Robots Paralelos Basada en un Conjunto de Parámetros Significativos; Universidad Politécnica de Valencia; 2006/2007.

[4] Advantech: PCI-1720 data sheet http://www.ucs.co.uk/pdf/PCI-1720.pdf

[5] Advantech: PCI-1720, 4-channel isolated D/A output card – User's manual; 1999

[6] J.-P Merlet.Parallel Robots. Kluwer, London, U.K., 2000.

[6] BA Series – Low-Cost, Compact Modular Amplifier; Aerotech Products; 2006 http://www.aerotech.com/products/pdf/baamp.pdf

[7] BA10/20/30 Series, User's Manual, P/N: EDA121 (V1.6); Aerotech Products; 2002. http://www.aerotech.com

[9] R. Ortega, M. Spong. Adaptive Motion Control of Rigid Robots: a Tutorial, Automatica, vol. 25, pp. 877-888, 1989.

[8] Advantech: PCLD-880 data sheet http://www.ucs.co.uk/pdf/PCLD-880.pdf

[9] Advantech: PCL-833, 4-axis quadrature encoder & counter card – User's manual; 2006

[12] B. Paden, R. Panja, Globally Asymptotically Stable 'PD+' Controller for Robot Manipulators. Int. J. on Control, vol. 47, 1697-1712, 1988.

[13] The Orocos Project. http://www.orocos.org/

[14] The Orocos Project. Aplications.http://www.orocos.org/orocos/applications

[15] Rock. The Robot Construction Kit.http://www.rock-robotics.org/orogen/

[16] The TaskBrowser Component http://www.orocos.org/stable/documentation/ocl/v2.x/doc- xml/orocos-taskbrowser.html

[17] Xenomai. http://www.xenomai.org/

[18] Kennerkecht, Judith. Proyecto Final de Carrera: Development and Control of a 3-dof, 3RPS Parallel Robot. Universidad Politécnica de Valencia – Facultad de Informática; 2007