UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

SIMULACIÓN VIRTUAL, RECONFIGURACIÓN ELECTRÓNICA Y PROGRAMACIÓN REAL DEL MOVIMIENTO DEL ROBOT HEXAPOD II

AUTOR: GUILLERMO OLIVER PEIRÓ

TUTOR: JOSE LUIS OLIVER HERRERO

Curso Académico: 2014-15

Agradecimientos

Este proyecto es fruto de un gran esfuerzo y dedicación, pero no habría sido posible sin el apoyo y el consentimiento de algunas personas. Quiero destacar a mi padre y tutor, José Luis Oliver Herrero, por haberme ayudado a lo largo del desarrollo del proyecto. Por supuesto, reconocer enormemente el apoyo técnico recibido por parte de Alejando Alís Rivas y Alfonso Merchante Camilleri, padres del robot alrededor del cual se basa este trabajo. Agradecer también la ayuda recibida para la parte de automática y control por parte de Juan Manuel Herrero Durá y Ángel Valera Fernández. Por último, quiero dar las gracias a todo el Departamento de Ingeniería Mecánica por ser tan tolerantes, a mis compañeros durante los cuatro años que estuve cursando el grado, y a todas las personas que de algún modo han ayudado y aportado conocimientos para poder alcanzar los objetivos marcados.

Resumen y Objeto del proyecto

Este proyecto tiene como objetivo realizar la puesta a punto del robot hexápodo *Hexapod II*, desarrollado hace poco más de una década, adecuándolo a las nuevas tecnologías y dotándolo de un control telemático. De esta forma, tras realizar las correspondientes modificaciones, se le colocará un ordenador de a bordo, que hará la función de servidor, y al que se podrán conectar clientes desde distintos dispositivos. Utilizando el protocolo de comunicación desarrollado, se establecerá una comunicación bidireccional que permitirá realizar su control y tomar datos acerca de su funcionamiento. Se implementará un controlador tipo PID para el manejo de las patas y se hará un estudio y simulación de su movimiento con el objeto de conseguir realizar movimientos complejos de forma sincronizada sobre una superficie plana.

Palabras clave: robot, hexápodo, controlador, simulación, microcontrolador, movimiento, programación, control, telemático, Hexapod II.

Abstract

The aim of this project is to bring back to life a six legged robot, known as *Hexapod II*, which was developed more than a decade ago. The robot will be updated in order to catch up with new available technologies and endowed with a wireless control. After doing the necessary modifications, an onboard computer will be attached to it, assuming the role of server to which clients from a wide range of devices can connect to. Using the implemented communication protocol, a bidirectional communication will be established, allowing it to be remotely manned and obtain data about its performance. A PID controller will be the one responsible for the movement of the legs, and a study and simulation of its movement will be carried out in order to perform complex movements in a synchronized fashion over a flat surface.

Keywords: six legged, robot, controller, simulation, microcontroller, movement, programming, wireless, control, Hexapod II.

Índice general

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Trabajo Fin de Grado

Simulación Virtual, Reconfiguración Electrónica y Programación Real del Movimiento del Robot *Hexapod II*

Memoria del Proyecto

Autor:
Guillermo Oliver Peiró

Tutor:
Dr. Ing. Ind. José Luís Oliver Herrero

Universidad Politécnica de Valencia
Valencia, ESPAÑA

Curso 2014-2015

Parte I

Memoria del proyecto

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En primer lugar introduciremos la base de la cual se parte para la realización de este proyecto, el robot *Hexapod II*. Este robot fue desarrollado entre diciembre de 2002 y mayo de 2004, por dos jóvenes estudiantes, Alejando Alís Rivas y Alfonso Merchante Camilleri, bajo la supervisión del mismo tutor que dirige este proyecto, José Luís Oliver Herrero. Estos dos ingenieros contaban con una excelente experiencia previa pues habían desarrollado conjuntamente un robot hexápodo de unas dimensiones considerablemente más reducidas, bautizado como *Hexapod*, que fue desarrollado entre los años 1999 y 2001, y que apareció en los informativos y en los periódicos de ámbito regional, así como en otros programas de televisión de la época.[1] El éxito de este primer proyecto llamó la atención del por entonces rector de la Universidad Politécnica de Valencia, Don Justo Nieto Nieto, y se les brindó la posibilidad de llevar a cabo el proyecto a una escala mucho mayor y con el soporte financiero de la Universidad. A pesar de todo el trabajo y la dedicación que se puso en el proyecto, no se pudo finalizar en su totalidad, y ese es el objetivo que persigue la realización de este trabajo de final de grado, hacer un esfuerzo genuino en la continuación de este proyecto, que durante 10 años ha estado en estado latente.

Centrándonos en el robot *Hexapod II*, sus características básicas son un peso de 43,7 kilogramos, una longitud de 117 centímetros, 58 centímetros de ancho, y una altura variable, medida desde el suelo a su punto más alto, que va desde los 23,6 estando apoyado en el suelo a los 55,6 centímetros con las patas totalmente extendidas. Podemos distinguir dos partes claramente diferenciadas en cuanto a su desarrollo: la parte mecánica y la parte informática. A continuación presentaremos las características más importantes del robot, destacando aspectos fundamentales y explicando *grosso modo* el funcionamiento de cada una de las partes.

[1] El robot **Hexapod** hizo aparición en los siguientes medios: el periódico *Diario de Valencia*, a fecha martes 29 de enero de 2002; la revista *Info Industrial* del Colegio Oficial de Ingenieros Industriales de la Comunidad Valenciana, en su publicación del mes de abril de 2002; en el informativo *Notícies 9* de Canal 9, a fecha 29 de enero de 2002; en el programa *Bon Dia Comunitat Valenciana* de Canal 9, del día 30 de enero de 2002; en los informativos de Antena 3 Televisión del día 30 de enero de 2002; en *El programa de Julio Tormo* de Valencia TV de fecha 30 de enero de 2002; y en los informativos de TVE 1 de fecha 27 de marzo de 2001 en su reportaje sobre el *Alcabot 2001*.

Peso (kg)	**Longitud (cm)**	**Anchura (cm)**	**Altura (cm)**
43,7	117	58	23,6 - 55,6

Cuadro 1.1: Características del robot *Hexapod II*

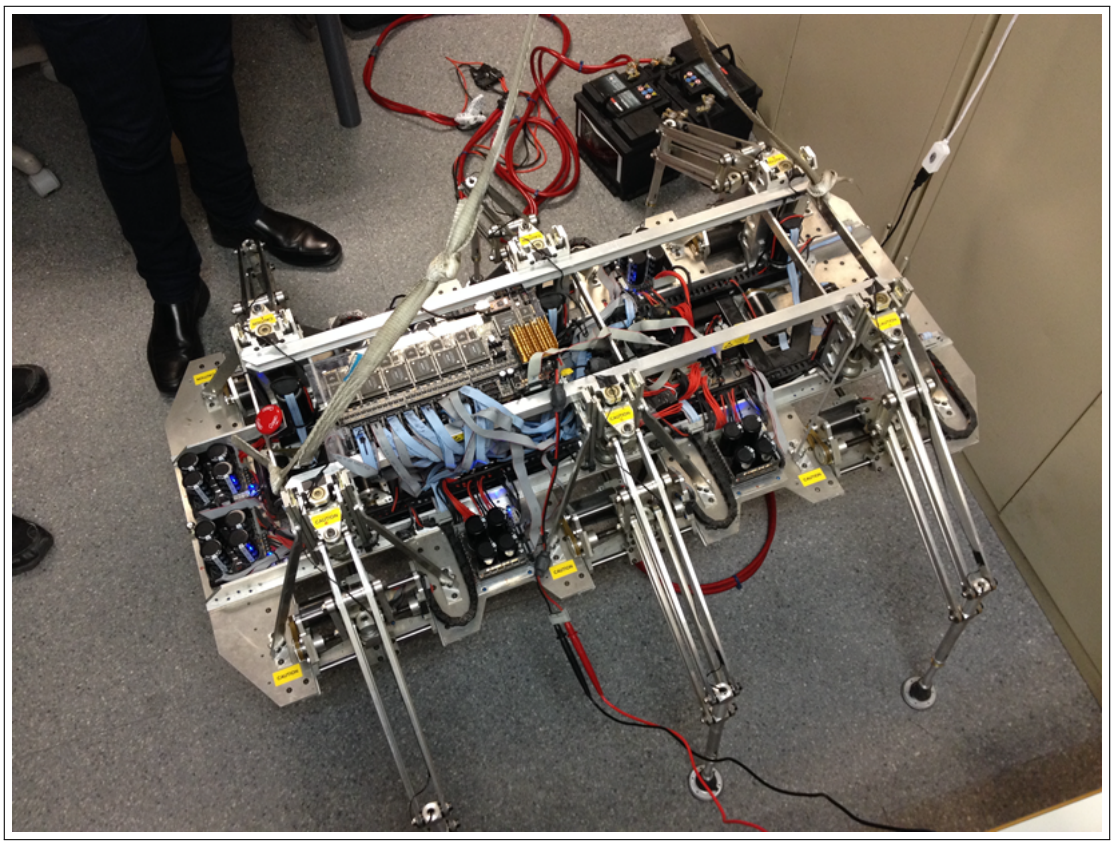


Figura 1.1: El robot *Hexapod II*.

1.1. Aspectos mecánicos

Morfológicamente se trata de un robot hexápodo, inspirado en un insecto, y su configuración de las patas presenta una serie de ventajas determinantes. El robot posee equilibrio estático en todo momento, lo que le permite caminar con seguridad de manera veloz a través de distintos tipos de terreno. La disposición de las patas, que sobresalen transversalmente del cuerpo, posee la ventaja de contar con mucha capacidad para sortear obstáculos de gran tamaño relativo, dotando al robot de equilibrio tanto transversal como longitudinal.

1.1.1. Mecanismo

El mecanismo utilizado en las patas es el denominado *pantógrafo tridimensional*, que posee una serie de ventajas destacables. En primer lugar, desde el punto de vista mecánico, permite que los actuadores, en este caso los motores de corriente continua, se encuentren situados en el chasis, de modo que las patas resultan ligeras, facilitando su movimiento y repercutiendo favorablemente en su rendimiento energético. También resulta un mecanismo interesante en el ámbito de su control, pues el movimiento del punto trazador (el pie del robot) es directamente proporcional al movimiento de la entrada. La transmisión de potencia se realiza a través del camino formado por el motor, la polea, la correa, el husillo, y finalmente la deslizadera de la pata. El motor está conectado a una polea mediante un tornillo pasador, y esta polea está conectada mediante una correa a la polea que está solidaria al husillo. Por el husillo se desplaza la deslizadera y este movimiento se traducirá a un movimiento del pie del robot según un grado de amplificación que depende de las longitudes de las distintas piezas del mecanismo. Tal y como está configurado el mecanismo, permite que los movimientos de las patas, entre los que se encuentran el de avance, el de desplazamiento lateral, y el de alzado, estén desacoplados. De esta manera, se pueden realizar los distintos movimientos usando un único actuador cada vez, lo que permite facilitar su control y además permitir un ahorro energético considerable. Las patas cuentan con tres grados de libertad cada una, y por sus dimensiones le permiten caminar hacia delante avanzando hasta 50 centímetros por paso, caminar lateralmente avanzando un máximo de 40 centímetros, y además superar obstáculos de hasta 40 centímetros de altura.

Es de gran importancia el uso de poleas y correas para la transmisión de potencia entre los motores de corriente continua y el husillo por el cual se mueven las deslizaderas. Si se hubiesen utilizado elementos de engrane continuo, como podrían ser dos engranajes de dientes rectos, si por algún fallo de control llegara la deslizadera al final de sus recorrido e intentara seguir avanzando, esto causaría importantes daños en el mecanismo. Sin embargo, estas correas hacen el papel fundamental de *fusible mecánico* o *eslabón más débil* en el camino de transmisión de potencia, de este modo son lo primero en romper preservando la integridad del resto del mecanismo. Durante la realización de este proyecto varias correas tuvieron que ser reemplazadas, dando buen crédito del fundamental papel que desempeñan como elementos de seguridad pasiva del mecanismo.

La utilización de un husillo para realizar el movimiento final sobre la deslizadera también es un aspecto destacable del mecanismo, puesto que presenta una serie de ventajas. En primer lugar, el control del motor se simplifica en gran medida puesto que el husillo permite que el movimiento rotatorio del motor se convierta en movimiento lineal sobre la deslizadera. Además este componente es unidireccional, puesto que se puede mover el mecanismo actuando sobre la polea solidaria al husillo, pero no desplazando la deslizadera, lo que permite que las inercias de la pata en su conjunto no afecten al motor, puesto que actuando sobre la deslizadera no se conseguirá desplazarla sobre el husillo, de modo que cuando se esté moviendo el motor y se quiera parar su desplazamiento, éste se parará justo donde se le ordene, sin dar ningún giro extra.

1.1.2. Motores y encoders

El robot tiene un motor por cada uno de los ejes de las patas, es decir, tres motores por cada pata, haciendo un total de 18 motores. Estos motores llevan acoplados mecánicamente unos encoders para obtener información acerca de los movimientos que se realizan. La elección de los motores y de los encoders es muy importante puesto que serán los elementos que se encarguen del movimiento del robot, y de su correcto funcionamiento depende el éxito o el fracaso del proyecto.

1.1.2.1. Motores

Los motores se encargan de traducir una señal eléctrica en movimiento, y conectados a los mecanismos son los encargados de realizar el movimiento del robot. Los motores se controlan aplicando una tensión de corriente continua en sus bornes, utilizando una PWM[2] para controlar la velocidad a la que gira éste. Atendiendo a las características del mecanismo y según el funcionamiento deseado, se realizó la selección de los motores. Se utilizaron motores de corriente continua *RE40* de la marca *Maxon*[3] con las siguientes características:

- Potencia nominal: $150W$.
- Tensión de funcionamiento: $24V$.
- Máxima corriente en continuo: $6A$.
- Máxima potencia útil: $440W$.
- Máximo par momentáneo: $2290mNm$.

[2]La **modulación por ancho de pulso**, PWM por sus siglas en inglés, es una técnica utilizada para variar la tensión aplicada a una carga. La tensión aplicada es una señal periódica con un período suficientemente reducido para que no afecte a la carga, y esta tensión se modifica partiendo la señal enviada en un tiempo en el cual la señal está a nivel alto, el llamado ciclo de trabajo, y un tiempo en el cual la señal está a nivel bajo. La carga percibirá una tensión igual al valor medio de la señal aplicada.

[3]*maxon motor*, motores DC de alta calidad. `http://www.maxonmotor.es`

1.1.2.2. Encoders

Los encoders en este caso se encargan de realizar la realimentación del movimiento del motor, traduciendo un movimiento a una señal eléctrica. Los encoders de los motores generan una cantidad de n pulsos eléctricos por cada revolución del motor. Estos pulsos permiten conocer la velocidad real del motor, qué distancia de movimiento ha realizado, etc. según la tensión aplicada, y de esta forma se realiza la realimentación necesaria para el control. Los encoders elegidos son el modelo *MR Digital* de la marca *Maxon* con las siguientes características:

- Alimentación: $5V \pm 5\,\%$.
- Salida: puerto compatible TTL.
- Número de pulsos por vuelta: 256.
- Posibilidad de conocer el sentido de giro del motor.

1.1.3. Diseño y construcción del robot

Para el diseño del robot se emplearon programas informáticos de distinta índole: programas de diseño asistido por ordenador (CAD), en este caso SolidWorks; programas de simulación dinámica como VisualNastran 4D; y programas de análisis por elementos finitos, más concretamente Ansys y Nastran. El montaje y la formación del robot se realizó casi en su totalidad con aleaciones de aluminio, y sus piezas se mecanizaron por medio de las máquinas-herramienta de control numérico de las que dispone la Universidad Politécnica de Valencia en sus instalaciones.

1.2. Aspectos electrónicos

Desde el punto de vista electrónico, se puede decir que el diseño del robot está dividido en dos partes: la electrónica de potencia y la electrónica de control. Ambas se hicieron a medida para el proyecto, de manera que pudieran encajar correctamente y tener las características deseadas.

1.2.1. Electrónica de potencia

La electrónica de potencia se encarga principalmente de alimentar todos los circuitos y manejar los motores mediante las órdenes recibidas desde la electrónica de control. Existen siete placas de potencia: una etapa principal de potencia, la placa *MOTION POWER*, y otras seis placas idénticas que conforman la etapa de potencia para cada una de las patas.

1.2.1.1. Alimentación

La alimentación del robot se hace por medio de baterías puesto que es en su totalidad eléctrico. La electrónica de control se alimenta mediante una batería o una fuente de alimentación a $15V$ y mediante la propia conexión USB con el ordenador, bien de a bordo o PC, a $5V$. Mientras que la electrónica de potencia se alimenta mediante unas baterías de plomo ácido a $24V$. Actualmente se disponen de unas baterías de uso en automoción con las siguientes características:

- Configuración: 2 baterías de 12V en serie.
- Capacidad: 68,3Ah.
- Corriente máxima de pico de descarga: 550A.

Se planteó la posibilidad de utilizar un generador eléctrico basado en un acoplamiento con un motor de explosión, pero se descartó debido a que está fuertemente desaconsejado para su uso en espacios interiores que no tengan una adecuada ventilación.

1.2.1.2. Etapa de potencia general

La etapa de potencia general está diseñada para conducir la potencia a hacia las etapas de potencia de las distintas patas. Está formada por 4 relés que permiten activar y desactivar la alimentación a las etapas de las patas, el encendido de los relés se realiza mediante transistores *MOSFET*[4], aplicando una tensión a la puerta de manera que se alimenta la bobina del relé, cerrando o abriendo el interruptor

[4]El **transistor de efecto de campo metal-óxido-semiconductor**, *MOSFET* por sus siglas en inglés, es un transistor excitado por tensión, a diferencia del transistor de unión bipolar, por lo que permite obtener un consumo muy bajo. Cuenta con tres terminales: el surtidor (S), el drenador (D) y la puerta (G). Se activa aplicando una tensión positiva con respecto al surtidor a la puerta (G), lo que provoca el flujo de electrones desde el drenador (D) hacia el surtidor (S).

de éste. La tensión aplicada a la puerta se controla desde una salida digital de la electrónica de control, y se dará un valor alto o bajo según se quiera activar o desactivar la etapa. También cuenta con un convertidor DC-DC de tipo *bomba de carga*[5] estabilizado con un diodo zéner para conseguir $38V$ para el control del puente H de la etapa de potencia de las patas a partir de los $24V$ de las baterías.

1.2.1.3. Etapa de potencia para cada pata

Para el manejo de los motores se diseñó un puente H realizado con MOSFET que permite el control del sentido de giro y el voltaje aplicado a los bornes del motor, además de permitir el frenado de manera casi instantánea. Las etapas de las patas cuentan también con unos condensadores de filtrado para que la señal aplicada sea lo más estable posible.

Los MOSFET se excitan mediante la aplicación de una señal a la puerta que ha de ser al menos $10V$ superior a la tensión en el surtidor, y por tanto surge la necesidad de utilizar el convertidor DC-DC para conseguir una tensión mínima de $34V$, pero sin elevado requerimiento de potencia en este caso.

Debido a la gran dimensión de las pérdidas energéticas que se podrían ocasionar al manejar grandes corrientes para alimentar los motores, se puso especial interés en el diseño de las etapas de potencia de cada una de las patas, logrando unas etapas de dimensiones reducidas capaces de manejar $20A$ a $24V$ de manera continua, soportando picos de hasta $300A$, y con un rendimiento energético muy alto, por encima del 95 %.

1.2.2. Electrónica de control

La electrónica de control manda las órdenes oportunas en tiempo real a la parte de potencia para conseguir que el robot siga el movimiento deseado. Cada pata está dotada de un microcontrolador que se encarga de controlar los motores de los tres ejes y además comunicarse con un microcontrolador principal del que va recibiendo las órdenes. El microcontrolador principal tiene también la función de recibir todos los eventos ocasionados en las patas y de transmitirlos hacia el exterior. El robot dispone de sensores que le permiten detectar cuando se ha alcanzado el final de carrera de cada uno de los tres ejes o cuando se ha apoyado sobre el suelo, para hacer su control lo más preciso posible.

1.2.2.1. Estructura

De la electrónica de control destacaremos dos características esenciales.

[5]Una **bomba de carga** es un dispositivo electrónico que convierte corriente continua de una tensión a otra utilizando condensadores. En el caso de querer elevar la tensión a la salida, se carga un condensador a una tensión y se conecta en serie con el voltaje de entrada, de modo que $V_{salida} = V_{entrada} + V_{condensador}$.

- Módulos de microcontrolador.
- Optoaislamiento.

1.2.2.2. Módulos de microcontrolador

El robot tiene siete módulos de microcontrolador; contando cada uno con un microcontrolador de 8 bits, una *FPGA*[6], y una memoria RAM externa. Con el micro se pueden programar todos los algoritmos secuenciales y gracias a la FPGA programar los concurrentes[7], por lo que en conjunto se puede solucionar cualquier problema. La RAM externa aumenta la capacidad de variables de los programas ya que la que lleva interna el micro es un poco reducida. El diseño se ha hecho de tal modo que para el microcontrolador la lectura y/o escritura de los datos se hace como si de posiciones de la RAM externa se tratase, de modo que todos y cada uno de los periféricos implementados en la FPGA tienen su propia posición de memoria y empiezan a partir dela 0x8000, de esta manera la programación del micro es más eficiente.

El microcontrolador utilizado fue un ATmega128-16 de la compañía *Atmel*[8]. Se trata de un microcontrolador de 8 bits, con una potencia de cálculo de hasta 16 *MIPS*[9] funcionando a 16MHz. Cuenta con una memoria flash para almacenar el programa de 128KB, una memoria RAM de 4KB, que es insuficiente por lo que se le ha puesto la RAM externa de 32KB, y una EEPROM de $4KB$. También dispone de multitud de conexiones para periféricos y la posibilidad de implementar manejadores de interrupciones. Aquí será dónde se ejecuten los programas desarrollados para el control del robot a lo largo de la realización del proyecto, diferenciándose entre un módulo que será para el programa del micro principal, y los otros seis módulos que encargarán de controlar cada una de las patas. A día de hoy existen microcontroladores mucho más potentes, pero en la época era un microprocesador puntero con un elevado rendimiento, que sigue manteniendo hasta la fecha.

Para la *FPGA* se ha elegido una de la compañía *Altera*[10], más concretamente la EPM7192S de la familia MAX 7000. Este dispositivo funciona enteramente a 5V, y cuenta con 3750 puertas lógicas disponibles y con 128 pines de entrada/salida. En la

[6] Una **FPGA**, *field-programmable gate array* en inglés, es un circuito integrado que permite une elevado nivel de configuración, pudiendo implementar en él circuitos de electrónica digital. Su programación se hace mediante un lenguaje de descripción de *hardware*, en este caso se utilizó *VHDL*, que permite realizar todo tipo de diseños electrónicos digitales utilizando puertas lógicas, multiplexores, biestables, y otros elementos más sofisticados.

[7] Gracias a la utilización de la FPGA y el lenguaje VHDL, las tareas que suponen una mayor carga computacional se podrán ejecutar de manera simultánea y a elevada velocidad, como puede ser la formación del PWM, mientras que el microcontrolador se encargará de dar las órdenes y leer los datos de la FPGA.

[8] *Atmel Corporation*, empresa líder mundial en el desarrollo de microcontroladores. `http://www.atmel.com`

[9] Siglas en inglés de **millones de instrucciones de código máquina por segundo**.

[10] *Altera Corporation*, empresa dedicada al sector de la lógica programable. `https://www.altera.com`

FPGA se implementaron una serie de módulos según se tratase del micro principal o el de las patas. Para los micros de las patas se implementaron los siguientes: un detector del sentido de giro del encoder, un contador de pulsos del encoder, un generador *PWM*, y el multiplexor para los distintos finales de carrera. Para el micro principal se implementaron otros módulos distintos: multiplexor para elegir con cuál de los seis micros de las patas se realizará la conexión. El *PWM* se optó por realizarse en la *FPGA* en lugar de en el micro de modo que fuera más eficiente y veloz su implementación. La tensión del *PWM* está discretizada mediante un valor de seis bits, pudiendo elegir entre 64, que se corresponde con la conexión directa a 24V durante todo el período de la señal resultante (ciclo de trabajo del 100 %), y 0, que se corresponde con una señal plana con valor efectivo de 0V. Por otra parte debido a que el micro principal tiene sólo dos conexiones serie y una se utilizará para la comunicación con el exterior, fue necesario implementar un modo de selección para la comunicación interna con los micros de las patas utilizando un multiplexor que permite seleccionar bajo demanda con cuál de los micros de las patas se va a realizar la conexión.

1.2.2.3. Optoaislamiento

Un motor eléctrico genera muchísimas interferencias que pueden afectar a los circuitos digitales. Para evitar esto la forma mas segura es que la electrónica digital esté separada de los motores y todo lo conectado a ellos: etapas de potencia, baterías, etc. Pero por otro lado las señales que genera la electrónica digital para controlar las etapas de potencia de los motores tienen que poder llegar de algún modo y uno de los mejores métodos para hacer esto es aislarlo mediante el uso de optoacopladores[11]. Los optoacopladores permiten conectar ambos circuitos y la información pasa en forma de luz, de manera que quedan efectivamente aislados y no se pueden causar interferencias entre sí.

[11]Un optoacoplador es un componente electrónico que permite la transmisión de señales de un circuito a otro en forma de luz. Está formado por un LED y un fototransistor, de modo que cuando se le aplica una señal al LED se provoca un haz de luz que excita el fototransistor y activa su conducción. De esta forma los dos circuitos están aislados entre sí, pero puede realizarse la transmisión de señales.

1.3. Motivación

La realización del proyecto responde a motivos académicos, en concreto para la obtención de la titulación de graduado en Ingeniería en Tecnologías Industriales. La elección del objeto es fruto de un interés por la robótica y la programación, además de la motivación que supone el afán por el aprendizaje y la satisfacción de la autorrealización. Durante mucho tiempo el autor de este proyecto pudo observar el robot en el lugar donde estaba almacenado, despertando, a medida que pasaba el tiempo, su interés y fascinación por éste. Poco a poco, a lo largo de los años en los que cursó el grado, tanto en las clases asistidas como por cuenta propia, el autor fue adquiriendo conocimientos que le permitieron ir comprendiendo el funcionamiento de los componentes del robot, hasta que finalmente se decidió por hacer de su estudio el presente trabajo.

A fecha de inicio de este proyecto, se partía de la base mecánica del robot totalmente terminada y en excepcionales condiciones de funcionamiento, así como gran parte de la electrónica de control funcionando correctamente. Se consiguió rescatar la documentación correspondiente a la programación de las *FPGA*, así como los esquemas de los circuitos y las placas de circuito impreso. Sin embargo, la electrónica de potencia tenía importantes carencias y el robot tenía falta de una revisión a fondo de su programación.

Capítulo 2

Objetivos

Los objetivos de la realización de este proyecto son los siguientes:

- Obtener el título de graduado en Ingeniería en Tecnologías Industriales.
- Iniciarse en el diseño y la resolución de funcionamientos erróneos de circuitos de electrónica de control y de potencia.
- Aprender a desarrollar programas de una cierta complejidad para microcontroladores y sistemas embebidos.
- Formarse en el uso de programas de simulación de mecanismos por ordenador.
- Volver a poner en marcha el robot *Hexapod II*.

Capítulo 3

Desarrollo del proyecto

Este proyecto consta de varias partes diferenciadas:

- Puesta a punto de la electrónica de potencia original para conseguir un correcto funcionamiento.
- Desarrollo de código en lenguaje C para los micros de la electrónica de control.
- Desarrollo de un programa cliente para PC con el objetivo de comunicarse mediante puerto serie (USB) directamente con la electrónica de control, o mediante Ethernet con el ordenador de a bordo del robot.
- Realización del control para el movimiento de las patas.
- Desarrollo de un programa servidor para un entorno Linux embebido implementado como ordenador de a bordo del robot y que sea capaz de realizar la comunicación entre los micros del robot y un cliente externo; ya sea PC, tableta inteligente o mando.
- Planificación de los movimientos para que el robot se desplace, primero mediante simulación con programa de *CAE*[1] y posteriormente implementándolos en el código del micro principal.
- Desarrollo de programas cliente para un dispositivo Android y un mando inalámbrico.
- Prueba del movimiento sobre el robot real.

A lo largo de esta memoria se irán detallando todos en lo que ha consistido el proceso de realización de cada uno de los puntos expuestos anteriormente. Se presentarán cada uno de los pasos que se han dado hasta lograr alcanzar las metas establecidas, dando una breve descripción y detalles técnicos al respecto. En este

[1]Los programas de ingeniería asistida por ordenador, **CAE** por sus siglas en inglés *Computer-Aided Engineering*, permiten el uso de los ordenadores para realizar tareas de análisis en ingeniería, abarcando una larga lista de disciplinas, que van desde la simulación de flujos de fluidos hasta simulaciones de sistemas multicuerpo.

documento se tratan los aspectos claves que han dado lugar a la realización de este proyecto, que son fundamentalmente de carácter electrónico, informático, y mecánico, para que el lector comprenda cómo se ha ido realizando cada punto, cuál es el funcionamiento de los distintos bloques, cómo han ido encajando unos con otros, por qué se han adoptado unas u otras soluciones, etc.

Esta memoria no pretende ser un manual para que se pueda reproducir paso por paso todo aquello realizado durante el desarrollo del proyecto, ni tampoco se detalla la información hasta el punto de que cualquiera pueda reproducir todos y cada uno de los pasos teniendo únicamente esta memoria como referencia. Durante el desarrollo de este proyecto se han escrito miles de líneas de código, las cuales no serán incluidas en el presente documento, puesto que provocarían que el documento tuviese una extensión exagerada, pero sí que se presentarán las definiciones de las clases y las funciones desarrolladas, explicándose el funcionamiento interno de cada una de ellas de la manera más clara y concisa posible.

En los anexos que acompañan a este documento, el lector podrá encontrar un listado de los ficheros que componen el código fuente desarrollado con una breve explicación, un manual del usuario avanzado que muestra como es el proceso de utilización del robot una vez terminado el proyecto, y otros documentos como esquemas electrónicos y resultados de las pruebas realizadas durante el desarrollo de los controladores.

3.1. Puesta a punto de la electrónica

Cuando se empezó a trabajar en el proyecto, el primer desafío al que tuvo que hacerse frente fue el de la puesta a punto y adecuación de la electrónica de potencia. La electrónica estaba un poco obsoleta y ya se había demostrado que su funcionamiento y diseño no era del todo óptimo. En primer lugar fue necesario reemplazar el diodo Schottky ubicado a la entrada de la etapa de potencia que sirve como protección ante una conexión inadecuada de las baterías. Se llevó a cabo la reconfiguración de la placa de potencia *MOTION POWER* sustituyendo transistores *MOSFET* quemados por transistores *BJT* con resistencias y por otros *MOSFET*. Se sustituyó uno de los cuatro relés originales que se había quemado durante su funcionamiento por un grupo de 3 relés extra, de tal modo que la corriente suministrada a los motores desde las baterías fuese distribuida por 6 relés en paralelo. También se simplificó el sistema de inicialización y se hicieron cambios menores en el circuito. Esta reparación permitió poder iniciar la etapa de potencia.

El funcionamiento de la electrónica de potencia se realiza desde órdenes gestionadas por la electrónica de control. Mediante dos señales (*POWER ON B*, *POWER ON C*) se encienden los relés que dan paso a la activación de la potencia para las etapas de las distintas patas. En primer lugar, la señal de *POWER ON B* se activa, lo que provoca la conexión de las etapas de las patas a las baterías a través de unas resistencias. Esto se hace para cargar los condensadores de filtrado y estabilización con los que cuentan cada una de las etapas de las patas a 24 voltios. Las resistencias son resistencias de potencia que pueden disipar suficiente potencia, y sirven para limitar los picos de corriente que se producen al cargar los condensadores, ya que si se conectaran las etapas directamente con los condensadores descargados a las baterías se produciría un pico de amperaje inaceptable que provocaría daños en las placas. Una vez transcurrido el transitorio y cargados los condensadores (esto ocurre después de aproximadamente un segundo), se pasa a la siguiente fase de la activación de la potencia, en la cual además de *POWER ON B* también está conectada *POWER ON C*, estado intermedio que está activado durante medio segundo. La señal *POWER ON C* es la que conecta directamente las etapas a las baterías, mediante 6 relés en paralelo por los que se distribuirá la demanda de corriente de las etapas cuando estén los motores en movimiento. Por último, se apaga la señal *POWER ON C* para que cuando se produzca el movimiento no estén conectadas las resistencias de potencia que han servido para cargar los condensadores, ya que debido a la gran demanda de potencia de los motores resulta más conveniente conectarlos directamente a las baterías, con el PWM que se le aplique según el modo de funcionamiento del puente H de las etapas de las patas.

Surge la necesidad de comprobar el estado de los voltajes, de manera que se pueda comprobar si todo ha ido según lo previsto. Los micros están alimentados a 5V, por lo que no podemos conectarles 12, 24, ni 38V a las entradas de los conversores analógico-digital, así que para consultar el estado de las tensiones de alimentación, la etapa de potencia cuenta con unos convertidores de voltaje en frecuencia. De este modo el conversor analógico-digital del micro principal puede leer estos valores y

comprobar si se dispone de las tensiones necesarias. Las tensiones de 12 y 24V están disponibles desde el momento en que se conectan las baterías, y una vez iniciada la etapa de potencia se realiza la conexión de la alimentación a las salidas para las etapas de las patas y se suministrarán 24V al convertidor DC-DC que es el encargado de proporcionar 38V para el control del puente H de los motores. Tras realizar el inicio de la etapa de potencia y comprobar que esta tensión está disponible, se puede dar por satisfactoriamente inicializada la parte de la electrónica de potencia.

Una vez comprobado que funcionaba correctamente la etapa de potencia, se optó por hacer salir de ella unos cables de alimentación para conectarlos al convertidor DC-DC USB que permitirá alimentar el ordenador de a bordo. Intentó hacerse el montaje lo más sencillo posible, adquiriendo un convertidor DC-DC USB de 5V con entrada de 24V ya ensamblado y conectándolo a los cables que provienen de la placa de potencia. De esta forma estará disponible alimentación por USB para el ordenador de a bordo y sus periféricos con, atendiendo a las características técnicas del convertidor, una tensión estable de 5V y un suministro máximo de 3A, más que suficiente para el consumo que se producirá durante cualquier momento de su funcionamiento.

En el anexo de electrónica que acompaña al presente documento se puede encontrar el esquema electrónico del circuito de la electrónica de potencia resultante. Se ha desarrollado utilizando el *OrCAD Capture* consultando para ello un libro de utilización del programa[2].

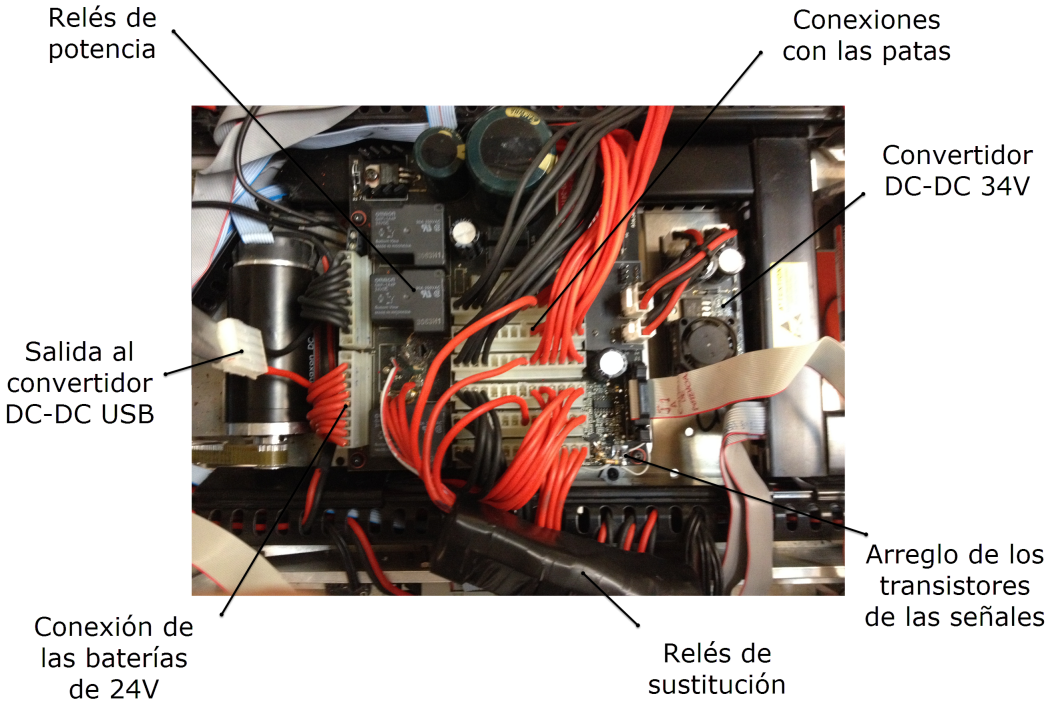


Figura 3.1: Vista general de la etapa de potencia tras las modificaciones.

[2][1] (Mitzner, 2009)

3.2. Desarrollo de los programas de los micros

La programación de la electrónica de control ha consistido en el desarrollo de programas totalmente nuevos para los micros. Existen dos programas distintos que han sido desarrollados *ex profeso* para la ocasión: uno para el micro principal, llamado a partir de este momento UCP; y otro para los micros de las patas, que es el mismo para los micros de las seis patas. Durante el desarrollo de los programas se consultó la página web de *Atmel* y las *datasheets* correspondientes al micro[3].

3.2.1. Características compartidas

En primer lugar se diseñó un protocolo de paquetes de información como forma de comunicación; tanto para los micros entre sí, como para la comunicación del micro principal con el ordenador. El protocolo se basa en el envío de paquetes de información denominados mensajes, definidos en la siguiente estructura:

```
typedef struct Mensaje
{
    uchar address[2]; //a quién va dirigido y de dónde viene?
    uchar type; //orden
    uchar size; //tamaño de los datos en 'data'
    uchar data[TAM_MSG_DATOS]; //buffer para datos
    uchar crc; //checksum
} Mensaje;
```

Figura 3.2: Definición de la estructura de los mensajes.

Lo primero que tenemos en la estructura es el vector de dos bytes de nombre *address*, en el primer byte se especifica a quién va dirigido el mensaje y en el segundo quién es el remitente. Los identificadores han sido definidos previamente y están codificados en código hexadecimal de 8 bits. El siguiente byte que enviaremos será el correspondiente al tipo de comando que se va a enviar, indicado en *type*. La comunicación se hace byte a byte, y cualquier dato adicional se ha de transformar en un vector de bytes que se almacenarán en el vector de datos *data* para poder ser enviados. Por lo que se enviará el tamaño de los datos adicionales en el byte de *size* y posteriormente el vector de bytes de datos. Por último, se envía el checksum, que se corresponde con el byte *crc*. Este no es más que un byte de control, se realiza la suma de todos los datos que conforman el mensaje y una vez recibido por parte del destinatario, éste comprobará que la suma de los bytes que ha recibido se corresponde con el valor del checksum. Es una comprobación muy rudimentaria, ya que se pueden utilizar CRCs mucho más potentes, pero es suficiente para comprobar la validez de los datos enviados y recibidos, para detectar cualquier problema o pérdida de información en el proceso.

La comunicación se realiza primero enviando un *carácter de control*, para hacerle saber al micro que se va a iniciar la transmisión de un mensaje. Una vez

[3] [3] (Atmel, 2011)

llegado el carácter de control, los datos siguientes se colocan en la estructura Mensaje y posteriormente se procesarán, previamente comprobando el checksum. Ambos micros tienen pilas de entrada y salida de mensajes, cada uno con sus peculiaridades.

Los programas de los micros están estructurados haciendo un uso intensivo de funciones e interrupciones. La función principal, main, es la que se ejecuta en bucle y va realizando las distintas tareas que se le van solicitando, cada una definida en distintas funciones. El micro dispone de un vector de interrupciones que permiten atender a los eventos que se produzcan mientras está en ejecución el programa del micro. Han sido implementados dos manejadores de interrupciones primordiales: uno para la llegada de datos por el puerto serie UART, y otro para el desbordamiento del contador del timer0. Cuando se produce un suceso que está siendo monitorizado por un manejador de interrupción, se dispara el código contenido en dicho manejador, parando el proceso principal del micro una vez terminada la instrucción que éste esté ejecutando. Una vez terminado el código del manejador, el proceso principal continúa su ejecución desde donde estaba cuando fue interrumpido.

3.2.2. Programa para el micro principal, UCP

El micro principal es el que coordina los movimientos para los distintos modos de caminar y es el puente para la comunicación entre el usuario y las patas. La etapa de potencia está controlada desde el UCP y se activa o desactiva a través de éste. Los distintos modos de movimiento están implementados en el UCP mediante órdenes que va enviando a cada una de las patas cuando se le ordena que se produzca el movimiento. Hay un sistema de realimentación acerca del movimiento de las patas, de modo que el UCP está en todo momento al tanto de la posición de las patas a las que ha mandado la orden de movimiento, esto permite realizar movimientos más precisos y seguros, ya que el micro puede esperar a que todas las patas lleguen a su destino antes de seguir con la siguiente secuencia en el movimiento.

En el micro principal la interrupción de llegada de datos por el puerto serie recibe datos y los va introduciendo en la estructura Mensaje y colocando en la pila de entrada. Todos los mensajes se procesan desde el bucle principal, excepto el mensaje de parada de emergencia que tiene un efecto inmediato. La interrupción de desbordamiento del *timer0* sirve para realizar las lecturas de los convertidores analógico-digitales cada 4,096mS.

Las funciones de inicialización contenidas en el código del micro principal son de ámbito general para la programación de cualquier microcontrolador. Las funciones *timer0_init*, *timer1_init*, *uart0_init*, *uart1_init*, *adc_init*, *port_init*, e *init_devices* son las que se encargan de inicializar todos los periféricos, configurando los registros de tal manera que se activen las distintas interrupciones y los dispositivos de comunicación por puerto serie. La función *ini_variables* se utiliza para inicializar todas las variables y estructuras globales declaradas en el código del micro. Es muy importante inicializar las variables antes de utilizarlas para asegurarse que todas tienen

```
//Funciones básicas de inicialización
void ini_variables(void);
void port_init(void);
void init_devices(void);
void timer0_init(void);
void timer1_init(void);
void uart0_init(void);
void uart1_init(void);
void adc_init(void);
```

Figura 3.3: Definición de las funciones de inicialización del UCP.

los valores adecuados. Estas funciones de inicialización se van llamando consecutivamente nada más arrancar el programa principal del micro.

```
//Funciones sistema
void LeeAd (void);
void Reset_Micro(int update);
void IniciaEtapa(uchar tipo);
uchar Test_RAM(void);
```

Figura 3.4: Definición de las funciones de sistema del UCP.

El UCP tiene una serie de funciones de sistema que son las encargadas de realizar las tareas internas del micro. La función *Reset_Micro* provoca el reinicio del programa del micro, y dispone de un parámetro en el que se le indica si en el siguiente inicio se producirá una actualización. El sistema de actualización no está implementado pero se pretende que en un futuro se pueda actualizar el firmware de los micros de manera remota, sin la necesidad de usar un programador conectado a la placa de cada uno de ellos. La función *LeeAd* se encarga de leer los valores del conversor analógico-digital del que dispone el micro, obteniéndose así el estado de los voltajes y de otros datos de aplicación futura. El micro posee una RAM externa y mediante el uso de la función *Test_RAM* se realiza una sencilla comprobación acerca del estado de la RAM, esta función devuelve un 0 si hay algún error y un 1 si la comprobación es satisfactoria. Por último, la función *IniciaEtapa* se utiliza para activar o desactivar la etapa de potencia del robot, según si el parámetro que reciba sea un 1 o un 0, respectivamente.

```
//Interrupciones
ISR (ADC_vect);
ISR (TIMER0_OVF_vect);
ISR (TIMER1_OVF_vect);
ISR (USART0_RX_vect);
```

Figura 3.5: Manjeadores de vectores de interrupción definidos para el UCP.

El micro principal tiene una serie de manejadores de interrupciones, ISR, implementados para atender a ciertos eventos en tiempo real, en el momento justo en que ocurren. La interrupción definida por *ADC_vect* indica que el proceso de conversión que estaba realizando el conversor analógico-digital ha finalizado, y el manejador correspondiente se encarga de almacenar el valor resultante de la conversión en la variable indicada. Hay dos manejadores definidos para los eventos de desbordamien-

to de los contadores, uno para el *timer0* y otro para el *timer1*. El definido por el *TIMER1_OVF_vect* se encarga de ir comandando y leyendo las conversiones del conversor analógico-digital, mientras que el *TIMER0_OVF_vect* se encargará de otros procesos rutinarios. Quizás el más importante de los manejadores de interrupciones es aquel que vigila la llegada de datos por el puerto serie que está conectado al ordenador, la interrupción definida por *USART0_RX_vect*. El micro realiza las comunicaciones a través de los puertos serie *UART0* y *UART1*, siendo el primero el conectado al ordenador y el segundo conectado al resto de micros. Debido a que sólo se dispone de un puerto serie para comunicarse con los seis micros de cada una de las patas, en la FPGA que está conectada al micro principal hay implementado un multiplexor, que redirige la comunicación del puerto serie *UART1* al micro de la pata que se comande desde la ejecución del programa del UCP. Este sistema de comunicación hace que los micros de las patas no puedan enviar datos de vuelta al UCP siempre que quieran, sino que han de esperar a que se le de paso a través del multiplexor para hacerlo. De modo que la comunicación del UCP con los micros de las patas es bajo demanda, y no hay manejador de la interrupción de llegada de datos por el puerto UART1. El manejador de la interrupción definida por *USART0_RX_vect* se encarga únicamente de almacenar los mensajes que van llegando en la pila de entrada, para ser posteriormente procesados. El único mensaje que se procesa automáticamente es el de parada de emergencia, que provoca un reset del UCP y de los micros de las patas, a través del envío del equivalente para parar también la ejecución de éstos.

```
//Funciones de comunicación
void GeneraMensaje(uchar msg_add, uchar msg_type,
    uchar msg_size, uchar *data, Mensaje *generado);
void PilaMensajesEntrada(Mensaje *msg);
void PilaMensajesSalida(Mensaje *msg);
void ProcesaMensajesEntrantes(void);
void EnviaPilaSalidaPC(void);
uchar GeneraCRC(Mensaje msg);
void PreguntaPata(uchar pata_id);
void Monitor(uchar pata_id, uchar estado);
void PreguntaPataMonitor(uchar pata_id);
uchar EnviaMensajePata(Mensaje msg_pata, uchar pata_id);
void EnviaDatosPata(Mensaje msg_pata, uchar num_pata);
uchar RecibeDatosPata(Mensaje *msg_pata, uchar num_pata);
void EnviaMensajePC(Mensaje *msg_pc);
```

Figura 3.6: Definición de las funciones de comunicación del UCP.

Las funciones de comunicación gestionan la transmisión bidireccional de datos del UCP con los distintos micros y con el ordenador[4]. Teniendo en cuenta que el UCP es un nexo de unión entre los micros de las patas y el PC, habrá un elevado tráfico de datos a través de él, por lo que se requiere el uso de dos pilas distintas, una de salida y otra de entrada, con unas dimensiones adecuadas y gestionadas correctamente para hacer funcionar todo el sistema según lo previsto. En cuanto se recibe un mensaje por puerto serie desde el PC o desde cualquiera de los micros, se almacena

[4]Cabe destacar que con **ordenador** se hace referencia tanto a un ordenador portátil o de sobremesa (PC), como al sistema embebido que se puede conectar para hacer la función de ordenador de a bordo (Raspberry Pi 2). Según sea la conexión realizada físicamente en el robot, puede ser cualquiera de estos dos sistemas indistintamente.

este en la pila de mensajes correspondiente. Los mensajes recibidos del ordenador se almacenan en la pila de entrada mediante la función *PilaMensajesEntrada*, mientras que los mensajes recibidos de los micros de las patas se almacenarán en la pila de entrada mediante la función anterior, o en la pila de salida mediante la función *PilaMensajesSalida*, según sea la naturaleza del proceso que se está realizando y que ha provocado la creación de ese mensaje.

Los mensajes contenidos en la pila de entrada son procesados llamando a la función *ProcesaMensajesEntrantes*, y se ejecutarán las órdenes pertinentes o se redirigirán a su destino. Mientras que los mensajes contenidos en la pila de salida se enviarán al PC a través de la ejecución de la función *EnviaPilaSalidaPC*. Las llamadas a estas dos funciones se hacen de manera periódica en el bucle del programa principal del micro. A su vez, estas funciones hacen uso de las funciones auxiliares de comunicación que contienen las instrucciones necesarias para enviar los mensajes a los distintos destinatarios. El envío de mensajes al ordenador se produce a través de la función *EnviaMensajePC*, que se llama o bien desde la función de envío de la pila de salida, o bien desde otra función que requiera que se envíe el mensaje de manera inmediata al PC, sin pasar por el estado intermedio de permanecer en la pila. Hay dos formas de realizar la comunicación con los micros de las patas, y para eso se utiliza la función *EnviaMensajePata* o el par de funciones *EnviaDatosPata* y *RecibeDatosPata*. *EnviaMensajePata* se utiliza cuando el mensaje que se va a enviar al micro de la pata no va a producir un resultado de más de un mensaje, como pueden ser aquellos mensajes de devolverán una confirmación de recepción o un envío de datos inmediato. El mensaje resultante se coloca en la pila de salida para ser enviado en el próximo ciclo de envío de mensajes. Por otro lado, las funciones *EnviaDatosPata* y *RecibeDatosPata* se utilizan para enviar y recibir datos en forma de mensajes bajo demanda, donde la respuesta va a ser un mensaje con información necesaria para seguir realizando el proceso que se está ejecutando. Más adelante se profundizará en el uso de un modo de comunicación u otro.

Las funciones *GeneraMensaje* y *GeneraCRC* son las utilizadas para crear los mensajes de datos cuando se produce un evento que lo requiere. La función *GeneraMensaje* recibe todos los parámetros necesarios para generar un mensaje con la estructura definida, así como una variable de tipo mensaje vacía previamente definida; el puntero al mensaje que ha recibido lo utiliza para actualizar sus datos y una vez ha finalizado el proceso, se tiene el mensaje generado disponible en la variable de tipo mensaje original. Los tipos de mensajes existentes están definidos previamente, y se producen, en su mayor parte, como respuesta a algún comando que se ha recibido, bien sea de confirmación de recepción, de confirmación de ejecución satisfactoria, o de error en algún momento durante el proceso. Todos los aspectos, así como los errores que pueden surgir en la comunicación entre el UCP y los micros de las patas, producen una serie de mensajes que son enviados al ordenador para que el usuario tenga la mayor cantidad de información posible acerca de lo que está ocurriendo durante la ejecución del programa del micro, pero sin llegar a ser una cantidad exagerada de información que ralentizara el correcto funcionamiento del sistema.

Debido a que los mensajes que se producen por la ejecución de los distintos procesos de los micros de las patas no pueden ser enviados directamente al ordenador, y se almacenan en un pila de salida dentro del propio micro, se necesita una función que permita consultar el estado de esta pila, y que vaya recibiendo y reenviando los mensajes al su destino. Mediante el uso de *PreguntaPata* se hace esta consulta. Primero recibimos un mensaje con la cantidad de mensajes que contiene la pila de salida del micro de la pata preguntada, y posteriormente se procede al envío de esta pila, almacenándose en la pila de salida del UCP. Esta es una función que hace uso del par de funciones *EnviaDatosPata* y *RecibeDatosPata*, puesto que la comunicación es más compleja que la respuesta de un sólo mensaje.

El sistema de monitorización implementado en las patas para la obtención de datos respecto a su movimiento, que se detallará en la próxima sección, provoca un gran volumen de datos que se tienen que enviar desde el micro de la pata al ordenador. La activación de este modo para cada una de las patas se realiza mediante la función *Monitor*, y los parámetros que se le pasan a la función indican la pata objetivo y si ha de activarse o desactivarse. Una vez se ha terminado el proceso de obtención de datos del modo monitor tienen que extraerse esos datos del micro de la pata, y esto se realiza mediante la función *PreguntaPataMonitor* que tras obtener un mensaje con la información del volumen de datos que va a ser procesado, se irán recibiendo los datos obtenidos en forma de mensajes. Esta función hace uso tanto de *EnviaDatosPata* y *RecibeDatosPata*, como de *EnviaMensajePC*, puesto que los datos que van llegando han de ser enviados directamente al ordenador puesto que de ser almacenados desbordarían el tamaño de la pila de salida.

```
//Funciones de movimiento
void MuevePataX (uchar num_pata, uint pos_pie, uint vel_x, uchar final_libre);
void MuevePataY (uchar num_pata, uint pos_pie, uint vel_y, uchar suelo_sigue, uchar final_libre);
void MuevePataZ (uchar num_pata, uint pos_pie, uint vel_z, uchar final_libre);
uchar EsperaPata(uchar num_pata, uchar eje);
uchar EsperaPataTodo(uchar num_pata);
void CambiarControl(uchar tipo_control);
void HacerResetRobot(uchar velocidad_reset);
void LevantarRobotTodasPatas(uint posicion, uchar vel);
void PosicionInicial(uchar velocidad);
void MoverRobot(uchar velocidad_suelo, uchar velocidad_aire, uchar velocidad_y);
void CaminaTripodeAlternanteFrontal(uint altura);
void CaminaTripodeAlternanteLateral(uint altura);
void BailaLado(void);
void BailaArriba(void);
void CaminaOnda(uint altura);
void Gira(uint altura);
```

Figura 3.7: Definición de las funciones de movimiento del UCP.

El movimiento del robot en su conjunto está coordinado desde las funciones de movimiento definidas en el UCP. Se optó por esta solución, predefiniendo unos movimientos en el código del micro, ante la alternativa de que cada una de las secuencias se tuvieran que recibir desde el ordenador por el puerto serie una a una, ya que esto provocaría una saturación desmedida del protocolo de comunicación. El ordenador

envía la orden de realizar un movimiento y una vez recibida es el UCP el encargado de coordinar este movimiento, informando al ordenador de los mensajes que vayan surgiendo durante la ejecución del proceso.

Cuando se enciende por primera vez el robot, es necesario que todas las patas conozcan su posición actual, de modo que lo primero que hay que hacer es el proceso de *reset*, o *puesta a cero*, de todas las patas, llamando a la función *HacerResetRobot*. Se hace el reset de todas las patas, siguiendo una secuencia establecida, y una vez ha terminado el proceso se actualiza el estado del robot a reseteado y listo para iniciar los movimientos. La secuencia de reset del robot está especialmente cuidada para que pueda realizarse estando el robot en el suelo, puesto que los comandos de reset se envían pata a pata, esperando a la confirmación de cada una de ellas de que ha realizado el reset y ha tocado suelo para enviar el comando a la siguiente. De esta forma se evitan situaciones de inestabilidad que podrían provocar si coincidieran varias patas en movimiento realizando el reset al mismo tiempo. Si se intenta realizar algún movimiento sin haber hecho previamente el reset de las patas, se devolverá un mensaje de error informando que es requisito indispensable realizar el proceso de reset antes de iniciar cualquier movimiento. También está implementada una función para modificar la posición con respecto al suelo del robot, *LevantarRobotTodasPatas*, donde se mueven los ejes Y de todas las patas a la vez para hacer ascender o descender al robot con respecto al suelo.

La manera de asegurar que el robot está en una posición compatible con el movimiento es llevándolo a una posición inicial antes de realizar cualquier movimiento. Para cada tipo de movimiento existe una posición inicial y antes de realizar los movimientos se comprueba que efectivamente esté el robot en esta posición, devolviendo en caso contrario un error. Para llevar al robot a las posiciones iniciales de movimiento o para colocarlo en otras posiciones que no suponen desplazamiento del conjunto, como puede ser la posición de plegado, se utiliza la función *PosiciónInicial*. Una vez colocado en la posición inicial adecuada, el robot puede recibir órdenes de movimiento compatibles con esa posición inicial. Cuando se recibe una orden de movimiento, se llama a la función *MoverRobot*, y según cuál sea el modo de movimiento elegido, se llamará a la función de movimiento pertinente. Las funciones de movimiento disponibles son las siguientes: *CaminaTripodeAlternanteFrontal*, *CaminaTripodeAlternanteLateral*, *CaminaOnda*, *BailaLado*, *BailaArriba*, y *Gira*. Los distintos modos de movimiento serán detallados en la sección correspondiente a la planificación de movimientos.

Dentro de las funciones de movimiento, se mandan los comandos de movimiento a los ejes de las patas por separado, haciendo uso de las funciones *MueveePataX*, *MueevePataY*, y *MueevePataZ*. Los movimientos están divididos en secuencias, que son conjuntos de órdenes de movimientos que se envían a las patas de manera prácticamente simultánea. Una vez enviada la secuencia a las patas, mediante la función *EsperaPata* o *EsperaPataTodo*, la ejecución del micro esperará a que termine la secuencia para proceder con los siguientes movimientos. De esta manera nos aseguramos de que todas las patas han llegado al lugar correcto antes de iniciar el siguiente

paso. Si se iniciara la siguiente secuencia de movimiento sin asegurarse de que la anterior se ha realizado correctamente podrían producirse estados incompatibles o hacer perder el equilibrio al robot en su conjunto, con consecuencias desastrosas. Por último, tenemos la función *CambiaControl*, que envía la orden de cambiar el tipo de control de todas las patas a todo/nada o al gestionado por el controlador tipo PID implementado.

3.2.3. Programa para los micros de las patas

Los micros de las patas son los encargados de aplicar el *PWM* para alimentar los motores a distinta tensión según se le solicite a través de la acción de control. Desde aquí se consultan los valores de los encoders de los motores y los finales de carrera para saber la posición exacta de los tres ejes de la pata en todo momento. También se desarrolló un sistema de monitorización (*modo monitor*) para guardar datos relativos al movimiento de las patas y posteriormente ser procesados por computador.

La interrupción de llegada de datos por el puerto serie es ligeramente distinta en el caso de los micros de las patas. Los mensajes que son de efecto inmediato o de envío de datos únicamente se procesan de manera inmediata, mientras que aquellos mensajes que requieren de consulta de datos o de realizar movimientos se colocan en la pila de entrada para ser procesados desde el bucle principal. La interrupción de desbordamiento del *timer0* en este caso se utiliza para leer los datos de los encoders y actualizar la posición actual de la pata, mandar aplicar el *PWM* necesario, y realizar el control de los límites de movimiento para evitar que se salga la pata de sus límites mecánicos.

```
//Funciones basicas de inicialización y de sistema
void ini_variables(void);
void port_init(void);
void init_devices(void);
void timer0_init(void);
void uart0_init(void);
void Reset_Micro(int update);
uchar Test_RAM(void);
```

Figura 3.8: Definición de las funciones de inicialización y de sistema de los micros de las patas.

Igual que sucede con el UCP, los micros de las patas tienen una serie de funciones de sistema que son las encargadas de realizar las tareas internas del micro. La función *Reset_Micro* provoca el reinicio del programa del micro, y dispone de un parámetro en el que se le indica si en el siguiente inicio se producirá una actualización. El sistema de actualización no está implementado pero se pretende que en un futuro se pueda actualizar el firmware de los micros de manera remota, sin la necesidad de usar un programador conectado a la placa de cada uno de ellos. El micro posee una RAM externa y mediante el uso de la función *Test_RAM* se realiza una sencilla comprobación acerca del estado de la RAM, esta función devuelve un 0 si hay algún error y un 1 si la comprobación es satisfactoria. Las funciones de inicialización *timer0_init*, *uart0_init*, *port_init*, e *init_devices* son las que se encargan

de inicializar todos los periféricos, configurando los registros de tal manera que se activen las distintas interrupciones y los dispositivos de comunicación por puerto serie. La función *ini_variables* se utiliza para inicializar todas las variables y estructuras globales declaradas en el código del micro. Es muy importante inicializar las variables antes de utilizarlas para asegurarse que todas tienen los valores adecuados. Estas funciones de inicialización se van llamando consecutivamente nada más arrancar el programa principal del micro.

```
//Interrupciones
ISR (TIMER0_OVF_vect);
ISR (USART0_RX_vect);
```

Figura 3.9: Manjeadores de vectores de interrupción definidos para el micro de las patas.

Los manejadores de eventos de interrupción cobran una gran importancia en el caso de los micros de las patas. Será *USART0_RX_vect* el manejador encargado de recibir y procesar los datos recibidos por el puerto serie. Debido a que la comunicación entre los micros de las patas y el UCP sólo está abierta cuando el micro principal lo solicita, algunos mensajes se procesarán directamente en el manejador de la interrupción, mientras que otros se almacenarán en la pila de entrada para ser procesados posteriormente. Los mensajes que serán procesados de manera inmediata por el micro de las patas son aquellos que son de alta prioridad o de respuesta inmediata de un mensaje datos, mientras que los que se guardarán para ser procesados en la ejecución del bucle principal son los que suponen realizar algún movimiento o procesos de mayor duración temporal. El movimiento se realiza desde el manejador para el desbordamiento del contador *timer0*, *TIMER0_OVF_vect*, ejecutado cada 4,096mS. Esta función realiza el grueso de las tareas de movimiento de la pata; se encarga de leer los datos de los encoders, realiza las comprobaciones necesarias para asegurar que los movimientos se mantienen dentro de los límites físicos del mecanismo de la pata, manda las órdenes de movimiento para la generación del *PWM* a la FPGA, y, cuando está activado el modo monitor, se encarga de recoger los datos acerca del movimiento.

```
//Funciones de comunicación
void GeneraMensaje(uchar msg_add, uchar msg_type,
    uchar msg_size, uchar *data, Mensaje *generado);
void PilaMensajesEntrada(Mensaje *msg);
void PilaMensajesSalida(Mensaje *msg);
void ProcesaMensajes(void);
uchar GeneraCRC(Mensaje msg);
void EnviaMensajeUCP(Mensaje msg_pc);
uchar RecibeDatosUCP(Mensaje *msg_recibido);
```

Figura 3.10: Definición de las funciones de comunicación de los micros de las patas.

El entramado de comunicaciones para los micros de las patas sigue teniendo pilas de entrada y de salida, pero es más sencilla que en caso del UCP. Los mensajes recibidos por el puerto serie que no se procesan inmediatamente se almacenan en la

pila de salida, a través de la función *PilaMensajesEntrada*, para posteriormente ser procesados desde el bucle del programa principal mediante la función *ProcesaMensajes*.

Las funciones *GeneraMensaje* y *GeneraCRC* son de nuevo las utilizadas para crear los mensajes de datos cuando se produce un evento que lo requiere. La función *GeneraMensaje* recibe todos los parámetros necesarios para generar un mensaje con la estructura definida, así como una variable de tipo mensaje vacía previamente definida; el puntero al mensaje que ha recibido lo utiliza para actualizar sus datos y una vez ha finalizado el proceso, se tiene el mensaje generado disponible en la variable de tipo mensaje original. Los tipos de mensajes existentes están definidos previamente, y se producen, en su mayor parte, como respuesta a algún comando que se ha recibido, bien sea de confirmación de recepción, de confirmación de ejecución satisfactoria, o de error en algún momento durante el proceso. Las distintas funciones que tiene el micro, tanto de movimiento como de cualquier otro ámbito, producen mensajes informando acerca de su funcionamiento que serán almacenados en la pila de salida, usando *PilaMensajesSalida*, esperando a que sean solicitados desde el UCP con el comando de pregunta pertinente. Mediante la función *EnviaMensajeUCP* y *RecibeDatosUCP* se realiza la comunicación bidireccional con el micro principal.

```
//Funciones principales
void ResetPata (uchar velocidad_reset);
void ModoMonitor(void);
void EnviaDatosMonitor(void);
void MueveMotor(uchar num, uchar vel, uchar masc);
void ControlLimites(void);
void MueveEjeX(uchar posicion, uchar velocidad);
void MueveEjeY(uchar posicion, uchar velocidad);
void MueveEjeZ(uchar posicion, uchar velocidad);
void IniciaPIDEjeX(uchar posicion);
void IniciaPIDEjeY(uchar posicion);
void IniciaPIDEjeZ(uchar posicion);
void LeeEncoders(void);
void Stop(uchar motor, uchar nivel);
void StopLibre(uchar motor);
void ModificarVelocidadX(uint velocidad);
void ModificarVelocidadY(uint velocidad);
void ModificarVelocidadZ(uint velocidad);
```

Figura 3.11: Definición de las funciones principales de los micros de las patas.

Las funciones principales del micro comprenden lo relacionado con el movimiento: la configuración, el muestreo de datos, el control de los límites, y las órdenes de movimiento y de parada de motores. La primera función de movimiento que ha de ejecutarse cuando se inicia el micro es la encargada de la puesta a cero de la pata, *ResetPata*. Esto es un paso necesario antes de realizar cualquier movimiento, y si se envía una orden de movimiento sin haber realizado el reset, se devolverá un mensaje de error informando acerca de la necesidad de hacer el reset previamente. Las funciones *MueveEjeX*, *MueveEjeY*, y *MueveEjeZ*, son las funciones de movimiento iniciales que se llaman cuando el control está en modo todo/nada, y no mandan directamente la orden de movimiento, sino que preparan todos los parámetros necesarios. En función de la posición objetivo, determinan el sentido de giro necesario del

motor (CW o CCW), se comprueba si el movimiento es mayor de un valor determinado, modifica la velocidad en la FPGA con el dato de velocidad recibido y modifica una bandera para que se produzca el movimiento desde la función *MueveMotor*. La función *MueveMotor* se ejecuta cuando se produce el desbordamiento del contador *timer0*, recibiendo como parámetros el identificador del motor, la velocidad y el sentido de giro, y es la encargada de transmitir estos parámetros a la *FPGA* para que se realice el *PWM*. Esta función se llama cuando el motor está parado y se ha activado la bandera de inicio de movimiento, pero también se puede modificar la velocidad bajo demanda cuando el motor está en movimiento mediante el uso de *ModificarVelocidadX*, *ModificarVelocidadY*, y *ModificarVelocidadZ*. La función *LeeEncoders* se encargará de leer los datos de los encoders periódicamente o cuando se solicite desde el ordenador. Las funciones *Stop* y *StopLibre* mandan la orden a la *FPGA* de parar en seco el motor, o para pararlo pero permitiendo su movimiento bajo inercia, respectivamente.

La función *ControlLimites* es de gran importancia para preservar la integridad de la pata, puesto que se encarga de comprobar que no se salgan las deslizaderas de sus límites de movimiento. Cuando la posición de alguna de las deslizaderas está encima del final de carrera o va a salirse de los límites de movimiento por el otro lado, el control de límites para en seco el motor y termina todas las órdenes de movimiento. También se usa para parar el giro del motor cuando se realiza el control de posición. Esta es una función primordial en el movimiento de las patas, puesto que debido a la fuerza que poseen los motores de las patas, cualquier contacto de las deslizaderas con las guías que las contienen provoca una situación mecánica poco recomendable, dando lugar a una rotura de la correa o a un fallo mecánico más grave.

Por último, tenemos las funciones relacionadas con el modo monitor y las de inicio del control de tipo PID: *ModoMonitor*, *EnviaDatosMonitor*, *IniciaPIDEjeX*, *IniciaPIDEjeY*, e *IniciaPIDEjeZ*. La función *ModoMonitor* se llama mientras se está produciendo el movimiento y se encarga de almacenar los datos de posición, velocidad, encoders, sentido y acción de control, en la memoria RAM externa, de modo que cuando se solicite se puedan enviar al ordenador a través del UCP utilizando la función *EnviaDatosMonitor*. Las funciones *IniciaPIDEjeX*, *IniciaPIDEjeY*, e *IniciaPIDEjeZ* inicializan los valores y hacen las comprobaciones necesarias para poner en funcionamiento el control tipo PID cuando se solicita un movimiento estando el micro configurado para este tipo de control.

```
//Funciones de control
uchar CambiaParametrosPID(Mensaje msg_parametros);
uchar CambiaLimitesPID(Mensaje msg_limites);
void AplicaRampaX(void);
void AplicaRampaY(void);
void AplicaRampaZ(void);
void AplicaControlX(void);
void AplicaControlY(void);
void AplicaControlZ(void);
```

Figura 3.12: Definición de las funciones de control del movimiento de las patas.

Las funciones relacionadas con el control de los motores una vez se encuentran en movimiento varían según se trate del control todo/nada o de controlador tipo PID. *AplicaRampaX*, *AplicaRampaY*, y *AplicaRampaZ* son las funciones del control todo/nada, que una vez se recibe el comando de movimiento por parte del micro, se encargan de realizar este movimiento de manera progresiva, siguiendo una rampa en cuanto a fijar la referencia para el movimiento se refiere. Las funciones *AplicaControlX*, *AplicaControlY*, y *AplicaControlZ*, se encargan del control tipo PID, obteniendo realimentación de los encoders y aplicando la acción de control necesaria en cada momento según requiera el tipo de controlador y los parámetros introducidos, esto se hace en este caso utilizando *ModificarVelocidadX*, *ModificarVelocidadY* y *ModificarVelocidadZ* respectivamente. Precisamente las funciones *CambiaParametrosPID* y *CambiaLimitesPID* son las que reciben los nuevos parámetros para el controlador y límites para las acciones de control, si fuera necesario, desde el ordenador y los almacenan en las estructuras de control, sustituyendo a los ya existentes.

3.3. Desarrollo del programa cliente para el PC

Para realizar el control del robot, primero mediante puerto serie utilizando un cable USB, y posteriormente mediante Ethernet con el ordenador de a bordo, se desarrolló un programa con interfaz gráfica utilizando el entorno de desarrollo C++Builder[5]. La interfaz gráfica cuenta con varias ventanas: elegir modo de comunicación (Serie o Internet), panel principal, datos del robot, datos de las patas, y parámetros PID. Para realizar las comunicaciones se utilizan dos clases distintas creadas especialmente para este caso: la comunicación por puerto serie se controla mediante una instancia de la clase HexapodSerie y la comunicación por Internet mediante una instancia de la clase InternetClient. La recepción de datos y el procesado de mensajes llevados a cabo por las instancias de estas clases se ejecutan dentro de un *Timer*, herramienta de la que dispone el entorno de desarrollo para que el código se ejecute en un hilo de ejecución distinto al del programa principal, permitiendo que realicen las tareas a una elevada velocidad sin afectar al resto del programa. Estas clases se han desarrollado de manera que resulten lo más portable posible, utilizándose las peculiaridades pertinentes para compilar bajo Windows, pero que con unas ligeras modificaciones y ajustes podrían ser utilizadas perfectamente bajo otro sistema operativo.

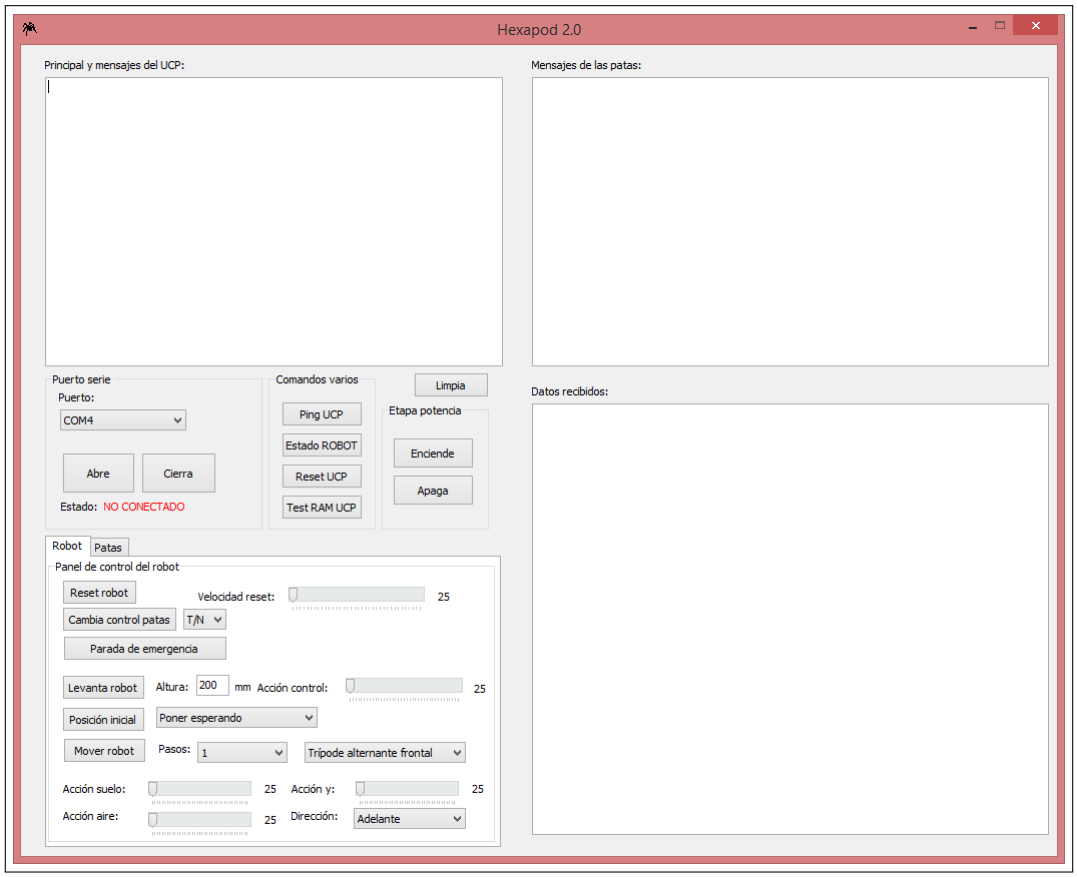


Figura 3.13: Vista general del programa cliente para PC.

[5] `http://www.embarcadero.com/products/cbuilder`

3.3.1. Clase de comunicación por puerto serie, *HexapodSerie*

```
class HexapodSerie {
    private:
        //variables
        bool estado;
        int com_port;
        int indice_monitor;

        //funciones previas
        void Inicializar(void);

        //mensajes
        unsigned char GeneraCRC(Mensaje msg);
        void LeeMensaje(unsigned char *buf, int tam);

        //datos
        void ProcesaDatos(Mensaje msg, int pata_id);
        void ProcesaEncoders(Mensaje msg, int pata_id);
        void ProcesaEstado(Mensaje msg);
        void ProcesaDatosPID(Mensaje msg, int pata_id);
        void ProcesaDatosLimitesPID(Mensaje msg, int pata_id);
        UnicodeString sensores;

        //buffer de entrada
        unsigned char buffer_entrada[BUFFER_DATOS_ENTRADA];
        int posicion_escritura_buffer;
        int posicion_lectura_buffer;

        //pila de entrada
        Mensaje PilaEntrada[N_PILA_ENTRADA];
        unsigned int posicion_escritura_pila_entrada;
        unsigned int posicion_lectura_pila_entrada;

        //monitor
        bool modo_monitor;
        void CrearLog(Mensaje msg, int pata_id);
    public:
        //constructor
        HexapodSerie();
        //destructor
        ~HexapodSerie();

        //estructuras de datos
        PataStr Pata[6];
        EncodersStr Encoders[6];
        PIDStr PID[6];
        RobotStr Robot;

        //variables
        int pata_seleccionada;

        //conexion puerto serie
        bool Conectar(int puerto);
        bool CompruebaEstado(void);
        void Cierra();

        //funciones princpiales
        void GeneraMensaje(unsigned int msg_add, unsigned int msg_type,
            unsigned int msg_size, unsigned char *data, Mensaje *generado);
        void EnviaMensaje(Mensaje msg_enviar);
        int RecibeDatos(unsigned char *buffer, int size);
        void LeeMensajes(void);
        void PilaMensajesEntrada(Mensaje *msg);
        void ProcesaMensajesEntrantes(void);
};
```

Figura 3.14: Definición de la clase *HexapodSerie*, con sus variables y funciones correspondientes.

La comunicación por puerto serie se realiza con el UCP de la electrónica de control del robot y está centralizada en la clase *HexapodSerie*, que tiene la estructura definida en la Figura 3.14. La organización de los paquetes de datos es la misma que la implementada en los micros, la estructura *Mensaje*, de forma que se pueda realizar la comunicación sin problemas.

Cuando la clase es instanciada a través del constructor, se llama a la función de *Inicializar* donde se inicializan las variables y se reserva memoria para el búfer de entrada. Cuando se solicita iniciar la conexión abrimos el puerto COM seleccionado con *Conectar* y una vez que se comprueba el estado usando *CompruebaEstado* la comunicación está preparada para ser usada. Periódicamente se comprobará si han llegado datos por el puerto serie usando la función *RecibeDatos*, la cual recibe un puntero a un búfer y la longitud de éste y devuelve el número de bytes leídos. Los datos leídos se almacenan en el búfer que ha recibido la función como argumento, de modo que puedan ser mostrados en codificación hexadecimal por pantalla, pero también se copian a la posición establecida por *posicion_escritura_buffer* del búfer de entrada. Una vez tenemos datos almacenados en el búfer de entrada, se llama a la función *LeeMensajes* que se encargará de procesar los datos del búfer, introducirlos en un *Mensaje* y colocarlos en la pila de entrada de mensajes, listos para ser procesados. Cuando leemos los mensajes se actualizan las posiciones registradas en *posicion_lectura_buffer* y *posicion_escritura_pila_entrada*. Es la función *ProcesaMensajesEntrantes* la encargada de procesar los mensajes que se encuentran en la pila de entrada, tomando como inicio el lugar al cual apunta *posicion_lectura_pila_entrada*, descifrando los datos que contienen y mostrando la información oportuna. Los datos relacionados con el estado general del robot y de las patas (posiciones, datos de encoders, parámetros de los controladores, etc.) son procesados y almacenados en las estructuras creadas para este propósito que son accesibles y se muestran por pantalla en las respectivas ventanas. La comunicación es obviamente bidireccional, la función *EnviaMensaje* es la encargada de enviar el mensaje que reciba como parámetro. Para generar un mensaje le pasamos a la función *GeneraMensaje* todos los parámetros y datos que queremos que contenga y recibimos un puntero al mensaje creado. Por último, la función *Cierra* finaliza la conexión, dejando abierta la posibilidad de realizar una nueva conexión siguiendo el mismo procedimiento.

3.3.2. Clase de comunicación por Internet, *InternetClient*

La comunicación por Ethernet se realiza a través del ordenador de a bordo del robot y está centralizada en la clase *InternetClient*, definida en la Figura 3.15. El servidor que corre el ordenador de a bordo tiene un sistema de identificación, que no permite la comunicación con el robot sin que se haya verificado la identidad del usuario mediante el envío de una contraseña justo después de iniciarse la conexión. Una vez realizado este trámite, todos los mensajes que reciba el servidor los redirigirá al UCP y los mensajes que reciba del UCP también serán redirigidos al PC. La comunicación en este caso se realiza también por medio de datos organizados en la misma estructura *Mensaje*, de modo que sean intercambiables entre las dos clases.

El funcionamiento de la clase de comunicación por Ethernet es similar a la del puerto serie, teniendo en cuenta las peculiaridades que supone realizar una conexión en lugar de la otra. Internamente, la comunicación se realiza a través de los sockets y las funciones que nos brinda el estándar de C++ utilizado, por lo que podría ser utilizado en Linux, y de hecho se usa un código similar para el servidor de la Rasp-

berry. Sin embargo, hay que inicializar la conexión utilizando una API específica de sockets para poder realizar la comunicación de manera satisfactoria bajo el sistema operativo Windows. Además el *carácter de control* se ha actualizado a una *cadena de control*, puesto que la comunicación a través de red es mucho más rápida y tiene un mayor trasiego de datos, de esta manera nos aseguraremos de que una vez recibida la cadena lo que está llegando es un mensaje de verdad, y no cualquier otra interferencia.

```
class InternetClient {
    private:
        //variables
        bool estado;
        bool autorizado;
        int s_escucha, s_cliente;
        struct sockaddr_in servidor;

        //mensajes
        void PilaMensajesEntradaInternet(Mensaje *msg);
        unsigned char GeneraCRC(Mensaje msg);

        //datos
        void ProcesaDatos(Mensaje msg, int pata_id);
        void ProcesaEncoders(Mensaje msg, int pata_id);
        void ProcesaEstado(Mensaje msg);
        void ProcesaDatosPID(Mensaje msg, int pata_id);
        void ProcesaDatosLimitesPID(Mensaje msg, int pata_id);
        UnicodeString sensores;

        //buffer de entrada
        unsigned char buffer_entrada[BUFFER_DATOS_ENTRADA_INTERNET];
        int posicion_escritura_buffer;
        int posicion_lectura_buffer;

        //pila de entrada
        unsigned int posicion_escritura_pila_entrada_internet;
        unsigned int posicion_lectura_pila_entrada_internet;
        Mensaje PilaEntradaInternet[N_PILA_ENTRADA_INTERNET];

    public:
        //constructor
        InternetClient(void);
        //destructor
        ~InternetClient(void);

        //estructuras de datos
        PataStr Pata[6];
        EncodersStr Encoders[6];
        PIDStr PID[6];
        RobotStr Robot;

        //variables
        int pata_seleccionada;

        //conexion internet
        void Conecta(std::string ip, int puerto);
        void Desconecta(void);
        bool EstadoConexion(void);

        //funciones principales
        void EnviaMensaje(Mensaje msg_enviar);
        int RecibeDatos(char *buffer, int size);
        void LeeMensajes(void);
        void ProcesaMensajesEntrantesInternet(void);
        void GeneraMensaje(unsigned int msg_add, unsigned int msg_type,
            unsigned int msg_size, unsigned char *data, Mensaje *generado);
};
```

Figura 3.15: Definición de la clase *InternetClient*, con sus variables y funciones correspondientes.

3.3.3. Interfaz gráfica

La interfaz gráfica se ha dividido en varias pantallas, para evitar concentrar demasiada información en una sola vista. A continuación se irán presentando las distintas pantallas y sus características. Podemos distinguir entre ventanas principales, donde se hace la mayor parte del control; y ventanas auxiliares, donde se muestran otros datos y parámetros sobre el funcionamiento.

3.3.3.1. Ventanas principales

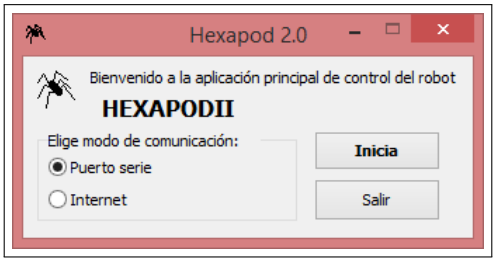


Figura 3.16: Pantalla de inicio del programa de PC.

La pantalla de inicio permite seleccionar la conexión que se va a utilizar para comunicarse con el robot, bien por puerto serie, bien por Internet. Una vez elegido el tipo de comunicación, se muestra la pantalla principal, donde se tiene acceso a todos los controles disponibles. Según el tipo de conexión, aparecerá una sección en la pantalla principal para conectar vía puerto serie o Internet, y aquí se introducirán los parámetros necesarios para cada tipo de conexión. Para el caso de la conexión por puerto serie será necesario únicamente seleccionar el puerto COM por el cual esté conectado el robot. La conexión por Internet requiere la dirección IP del ordenador de a bordo del robot y el puerto por el cual está el servidor a la escucha de nuevas conexiones, además de la contraseña para realizar la correcta identificación en el servidor.

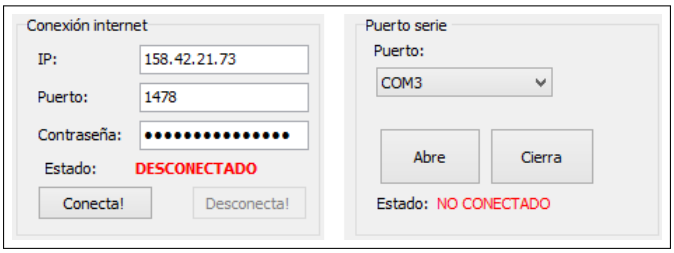


Figura 3.17: Los dos recuadros para las distintas conexiones disponibles.

La pantalla principal, cuya vista general se presenta en la Figura 3.13, está dividida en varias zonas. Hay dos cuadros de texto que muestran los comandos enviados al UCP y a los micros de las patas por separado, mostrando también información acerca de la respuesta recibida. También hay un cuadro de texto donde se muestran los datos recibidos directamente tanto por el puerto serie, como por el socket, en

codificación hexadecimal y sin ningún tipo de formato. Para realizar el control del robot, hay varias zonas delimitadas mediante unos recuadros debidamente identificados.

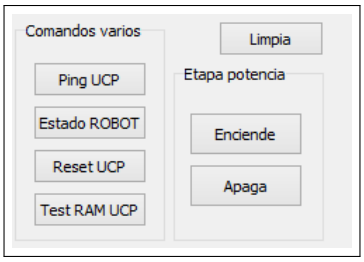


Figura 3.18: Comandos varios.

Por un lado se tiene la zona de *Comandos varios*, que contiene botones para enviar los comandos de ámbito general para el UCP; el ping, la solicitud de información acerca del estado del robot, el reset del micro, y el test de la RAM externa. Cuando se pulsa en el botón para conocer el estado del robot, se abre una nueva pestaña donde se muestran los valores que obtiene el micro de los convertidores analógico-digital que tiene incorporados y los estados internos más representativos. De en esta manera se obtendrán las tensiones de alimentación y otros datos que serán de aplicación en futuros desarrollos. La gestión de la etapa de potencia se decidió separar del resto de comandos debido a su gran relevancia, y desde la zona denominada *Etapa potencia* se produce la corresponderte activación o desactivación.

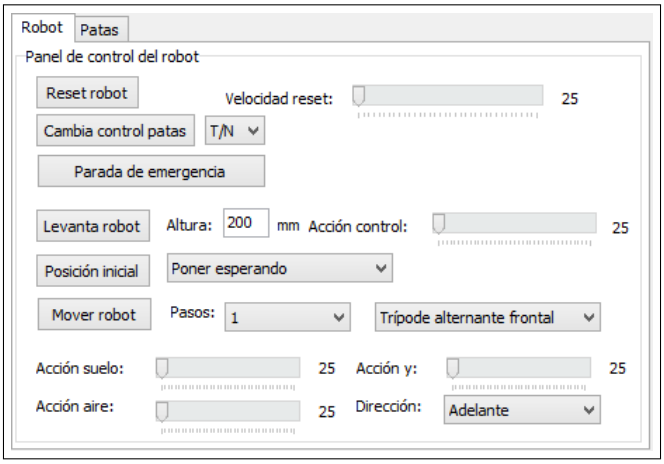


Figura 3.19: Panel de control del robot.

Los comandos de movimiento del robot en conjunto y de comunicación con cada una de las patas por separado se encuentran divididos dos pestañas distintas. La primera pestaña, *Robot*, que es la que está activada por defecto, se corresponde con el panel de control del robot propiamente dicho, donde se pueden mandar órdenes de movimiento al robot como un todo. Para cada orden de movimiento se pueden cambiar parámetros como altura o velocidad, además de seleccionar los distintos

tipos de movimiento que hay implementados. Se tiene la opción de hacer el reset del robot, requisito previo imprescindible para poder realizar cualquier otro movimiento, seleccionando la velocidad a la que se moverán los motores durante el proceso de reset. También se puede cambiar el tipo de control de todas las patas del robot a la vez, pudiendo elegirse el control todo/nada o el de tipo PID mediante el accionamiento de una lista desplegable. El cambio de control a tipo PID actualizará el texto y las barras de acción de control aplicada para los movimientos a otras de velocidad con unos rangos distintos, y viceversa. Junto a estos dos botones de carácter general, también se encuentra el botón de parada de emergencia que para el movimiento de todas las patas y también cualquier proceso que se esté ejecutando en el micro principal de manera instantánea. Es una medida de seguridad imprescindible cuando se trata de un robot de tan grandes dimensiones y que maneja una elevada potencia. Por debajo de estos botones se hallan los botones de los distintos comandos de movimiento. Eligiendo la altura y la velocidad de los ejes Y de las patas, se podrá subir o bajar la altura absoluta del robot respecto del suelo con el botón disponible. La barra de velocidad usada para levantar el robot también controla la velocidad que se utiliza a la hora de enviar el comando de llevar el robot a la posición inicial deseada.

Hay dos listas desplegables para el movimiento, una lista para las posiciones iniciales y otra para los movimientos de avance. Además de las posiciones iniciales de movimiento, la lista de posiciones iniciales también contiene otras posiciones que son movimientos de un solo evento, como la de *Poner esperando* o *Plegar robot*. Antes de enviar la orden de realizar cualquier movimiento de los definidos en la lista desplegable para los movimientos de avance, se ha de llevar el robot a una posición inicial compatible, que se hace eligiendo de la lista de posiciones iniciales aquella que tiene el mismo nombre que el movimiento a realizar. Una vez elegido el movimiento y el robot en posición inicial adecuada para realizarlo, se puede elegir las veces que lo repetirá y las distintas velocidades. El número de pasos permite seleccionar cuantos ciclos enteros se realizarán de ese movimiento. Los movimientos están implementados en dos sentidos que podrán ser seleccionados, bien hacia delante y hacia detrás, o bien hacia la derecha y hacia la izquierda, según la naturaleza del movimiento. Además se pueden elegir las velocidades de avance en suelo, avance en aire, y de movimiento de las patas en el eje Y. Todos los detalles de los movimientos se explicarán a fondo en la sección dedicada al movimiento del robot.

Para la comunicación con cada una de las patas de manera independiente se tiene otra pestaña, donde se pueden enviar los comandos generales y recibir información acerca de la pata que seleccionemos. En este panel de control lo primero que se hará será seleccionar la pata con la que se quiere realizar la comunicación, haciendo uso de la lista desplegable. Una vez elegida la pata, se pueden enviar comandos para comprobar el estado del micro, como es el caso del ping y el test de la RAM externa. Desde este panel se puede preguntar a la pata acerca de los mensajes que tiene almacenados en su pila de salida, de esta forma se comunicará mediante un mensaje cuántos son los que poblan esta lista, y posteriormente serán recibidos. Los mensajes que se van generando y llegando durante la comunicación aparecerán en el cuadro de texto correspondiente. Antes de nada se ha de realizar el reset de

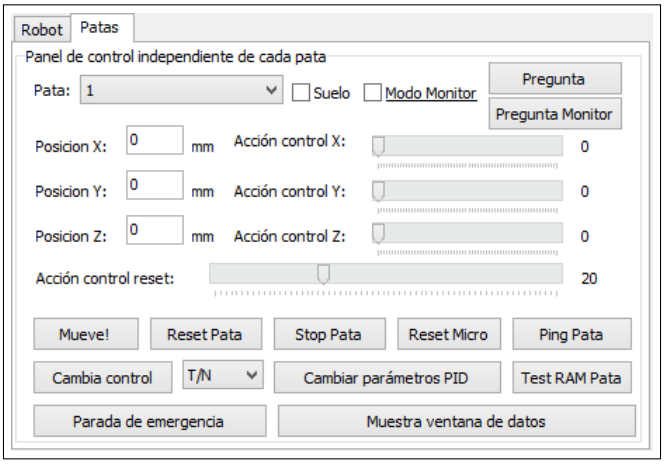


Figura 3.20: Panel de control independiente de las patas.

la pata, pudiendo elegir la velocidad desde la barra deslizable. Desde este panel se puede cambiar el tipo de control y ajustar los controladores de cada una de las patas, mediante el botón correspondiente, que abrirá una nueva ventana donde se podrán modificar cada uno de los parámetros que tienen los controladores de tipo PID de las patas. Realizar el cambio del tipo de control a tipo PID modificará las barras de selección de acción de control a velocidad de movimiento, y viceversa. También se puede realizar el movimiento de manera independiente para cada uno de los ejes de la pata, modificando los valores de posición y velocidad para cada uno de ellos. La opción de *Suelo*, permite indicar si el movimiento del eje Y parará si se alcanza el suelo o no, esta opción se ha de desmarcar cuando se deseé que el realice un movimiento de ascenso. Cabe destacar que si se opta por mover las patas independientemente, se ha de tener en cuenta que hace falta realizar el reset de la pata antes de realizar cualquier movimiento, en caso contrario se producirá un mensaje de error que se almacenará en la pila de salida. Hay un botón que abre una nueva ventana de datos donde se pueden obtener datos bajo demanda de los sensores, de los finales de carrera, de los encoders, y de la posición de la pata en cada momento. Hay dos botones de emergencia implementados en el panel de control, *Stop pata* y *Parada de emergencia*. El primero para cualquier movimiento que esté realizando la pata, y el segundo, además de realizar la parada, detendrá la ejecución del programa del micro, ambos mensajes son de alta prioridad y se procesarán nada más sean recibidos. El modo monitor se inicia marcando la caja correspondiente y se desactiva desmarcándola. Una vez realizado un muestreo de datos usando el modo monitor, mediante el correspondiente botón de preguntar se recibirán los datos del movimiento, guardándose en el fichero *log_monitor.txt* en la ruta donde se encuentre el programa.

Los cambios del tipo de control en ambas pestañas, bien sea controlando el robot en su conjunto o cada para por separado, provocarán la actualización de las barras y las leyendas destinadas a la selección de las velocidades de movimiento. En el caso de estar activado el control todo/nada, las barras permitirán seleccionar la acción de control (en voltios sobre un máximo de 63) aplicada para el movimiento,

limitada entre 0 y 63; mientras que si está seleccionado el control tipo PID, las barras permitirán seleccionar la velocidad de movimiento en pulsos por ciclo, estando esta limitada a valores entre 60 y 130 puslsos/ciclo. Además, puede parecer que las alturas del robot y los movimientos de los distintos ejes de las patas no están limitados, puesto que se puede introducir cualquier valor para el parámetro en milímetros, pero los micros tienen unas limitaciones internas que a pesar del valor recibido este se saturará a su valor máximo o mínimo según corresponda.

3.3.3.2. Ventanas auxiliares

A continuación presentaremos y haremos una explicación de las pantallas auxiliares que se han ido comentando en las explicaciones anteriores. Estas ventanas se accionan desde los distintos paneles de control y nos permiten obtener y modificar datos del funcionamiento del robot. Se optó por realizar ventanas separadas para evitar saturar la ventana principal con demasiados datos. Algunas de estas ventanas nos permiten acceder a datos del funcionamiento interno del robot de carácter más técnico.

Figura 3.21: Ventana de estado del robot.

En la ventana de estado del robot, aparecerán datos acerca del estado interno del robot en su conjunto, que se obtendrán bajo demanda siempre que se solicite a través del botón de actualización. La sección de *Voltajes y baterías* muestra los valores de tensión de las baterías de la etapa de potencia y de la electrónica de control. Desde aquí se tiene disponible el valor de tensión de las baterías prinicpales que suministran 12 y 24 voltios desde el momento en que se conectan, y el valor del convertidor DC-DC de 38 voltios que en valores comprendidos entre 32 y 38 voltios indicará cuándo está activada la etapa de potencia, así como el estado de la batería de la electrónica de control, siempre que las baterías utilizadas sean compatibles con esta opción. La sección *Robot* muestra el estado actual del robot, en cuanto a movimiento se refiere. Aquí se podrá consultar si el robot ha sido reseteado o puesto a cero, de modo que esté listo para realizar el movimiento, y si está en su totalidad en modo control todo/nada o tipo PID. En la sección de *Otros* se mostrarán otros datos de interés en desarrollos futuros. En futuras implementaciones se tendrá acceso a la inclinación del robot cuando se coloque un inclinómetro biaxial o una unidad de medición inercial, y también se tendrá una lectura de la temperatura exterior

usando un termistor.

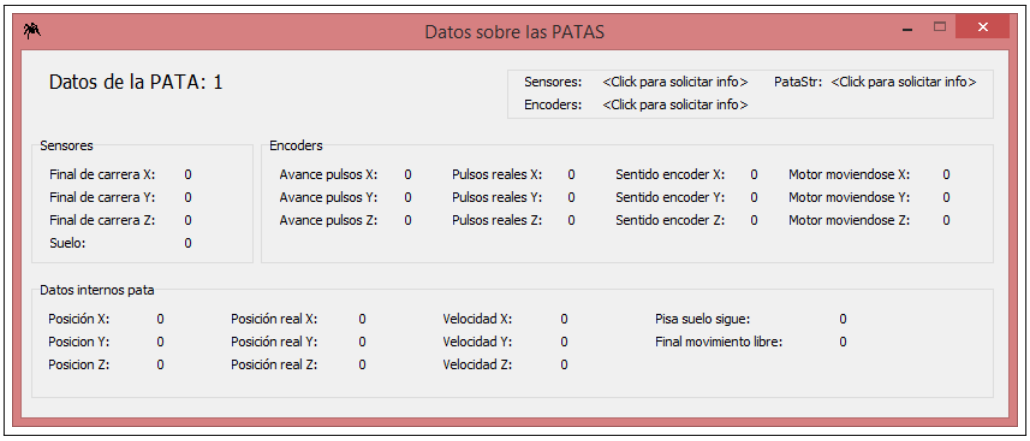


Figura 3.22: Ventana de datos de la pata.

Desde el panel de control de las patas se tiene acceso a dos ventanas de datos auxiliares, la ventana de datos de posición de pata y la ventana de parámetros del controlador.

En la ventana de datos de la pata se solicitan bajo demanda los datos de los distintos sistemas de realimentación que se tienen implementados, divididos en tres secciones. En primer lugar, se pueden solicitar los datos de los sensores, que devolverán un 1 o un 0, según esté la deslizadera correspondiente sobre el final de carrera o no. En el mismo lugar se mostrará si la pata está en contacto con el suelo o no. En la sección de datos internos se obtendrán los valores de posición de la pata en milímetros. Los datos obtenidos se dividen en varios grupos: la posición a la que se le ha ordenado que se mueva la pata, la posición real en la que se encuentra, la última velocidad a la que se ha realizado el movimiento, si ha de seguir si llega al suelo, y si el giro del motor se deja libre una vez finalizado el movimiento. Por último, se pueden solicitar datos técnicos acerca de los motores de las patas, obtenidos directamente de los encoders que se encuentran acoplados a cada uno de los motores. Todos estos datos se encuentran expresados en la unidad básica de lectura del encoder, vueltas del eje del motor. El avance del motor representa la velocidad a la que se está moviendo el motor, expresado en vueltas por ciclo. Los pulsos reales muestran la cuenta de vueltas absoluta en la que se encuentra el motor, dividiendo este valor por el número de vueltas que necesita cada motor para avanzar un milímetro de la deslizadera se obtienen los datos de posición representados en la sección descrita anteriormente. También se obtienen los datos acerca del sentido en que estaba girando el motor en el último movimiento realizado, mediante el último sentido registrado por el encoder, y si el motor está en movimiento en ese momento.

La configuración del controlador tipo PID para cada una de las patas tiene una ventana propia, en la que se pueden consultar y cambiar los valores del controlador que está implementado en cada pata. Desde aquí se pueden solicitar los datos de cada

controlador implementado en cada uno de los ejes, bien realizando la solicitud para obtener todo los parámetros a la vez, o uno a uno. Los parámetros están divididos en dos partes: los parámetros de la expresión del PID en sí para cada una de los ejes, es decir, Kc, Ti, y Td; y los parámetros de limitaciones de velocidades y acciones de control. Modificar los parámetros de esta ventana puede ser muy útil, pero a la vez peligroso, por lo que ha de hacerse con cuidado. A pesar de que la implementación del código en el micro de la pata, tanto el controlador como el control de los límites, tiene una serie de mecanismos de control, hacer modificaciones sobre los parámetros de la expresión del PID pueden provocar que la respuesta del sistema sea inestable y producir oscilaciones de cierta violencia. Las limitaciones de velocidades y de acciones de control permite modificar el valor máximo que tomará la acción de control para que no supere un límite que se establezca, así como introducir un factor de escala que multiplique todos los valores de acción de control obtenidos de aplicar el controlador. Estas limitaciones son de gran utilidad a la hora de hacer pruebas y ajustar los distintos valores de los parámetros del controlador, y se utilizaron para su ajuste de manera experimental durante el desarrollo del controlador PID para la posición y la velocidad.

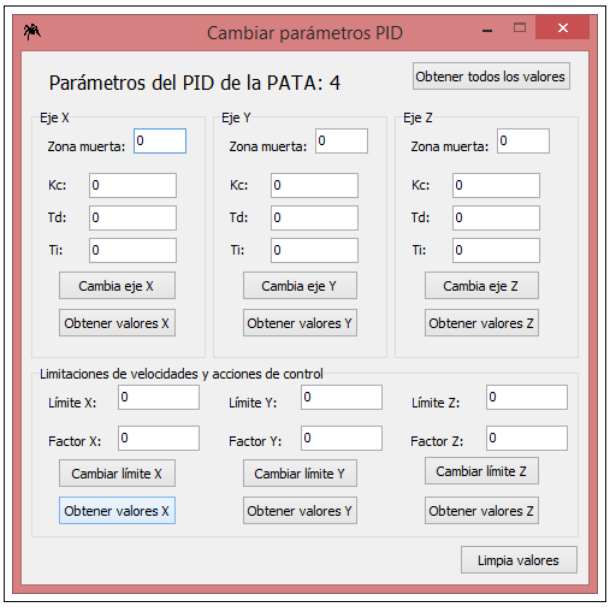


Figura 3.23: Ventana de configuración de los parámetros del controlador de la pata.

3.4. Control del movimiento de los motores

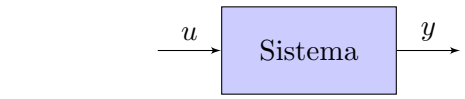
Se han implementado dos tipos de control para el movimiento de los motores de las patas. El primero es un control sencillo todo/nada, que corta la alimentación al motor cuando la realimentación del encoder indica que se ha alcanzado la posición deseada, aplicándole directamente la tensión equivalente a la acción de control elegida. Posteriormente, el desarrollo del *modo monitor* haciendo uso de la RAM externa disponible en el módulo del micro permitió la implementación de un controlador tipo PID para el movimiento de los motores de las patas. Para realizar el control se implementó un controlador de tipo PID siguiendo el estándar ISA[6], en el que se pueden modificar los valores de los distintos parámetros desde PC usando el programa cliente desarrollado. En una primera aproximación se optó por utilizar un controlador para la posición, pero tras el desarrollo de las pruebas se llegó a la conclusión de que no era lo más adecuado. Al tratarse de un proceso al que se le aplica una tensión y el motor gira indefinidamente hasta que se corte la tensión suministrada, la posición final se alcanzará siempre, lo único que necesita es que el sistema de realimentación vigile que esa posición no sea superada, y por tanto lo que se necesita realmente es un control de velocidad para que se desplace a una velocidad concreta. Finalmente se optó por un control mixto, teniendo un control todo/nada para la posición y un control tipo PI para la velocidad, lo que dio lugar a buenos resultados.

3.4.1. Teoría del control

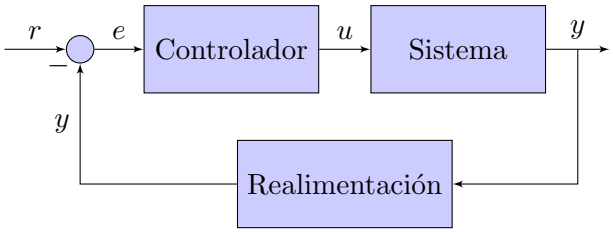
El funcionamiento de un proceso se basa en la modificación de una variable de entrada, llamada *variable manipulada* o *acción de control*, que aplicada a la entrada del sistema da como resultado la modificación de sus salidas. Un sistema con un controlador altera este esquema de funcionamiento, el controlador se coloca entre la entrada y el proceso y recibe un valor deseado que será la salida que se quiera conseguir para el sistema. En este caso concreto se implementará un controlador de tipo PID. Este funcionamiento se representa en la Figura 3.24, siendo: u la variable manipulada o acción de control, y salida del sistema, r valor a seguir o referencia, e error o diferencia entre la variable manipulada y la salida.

Un controlador proporcional-integral-derivativo, *PID*, es un mecanismo de control en bucle cerrado que utiliza el valor deseado, o referencia a seguir, y el valor real recibido mediante realimentación, para calcular el error y mediante el ajuste automático de la acción de control intenta minimizarlo. La acción de control se ajusta según tres parámetros distintos: la acción proporcional, que depende del error actual; la acción integral, que tiene en cuenta la acumulación de errores anteriores; y la acción derivativa, que es una predicción de errores futuros según la línea de tendencia del error actual. En la Figura 3.25 se muestra la ecuación que describe el PID estandarizado que transforma el error $e(t)$ en una acción de control $u(t)$, siendo K_p la ganancia proporcional, T_i el tiempo integral, y T_d el tiempo derivativo.

[6]La sociedad internacional de automatización, **ISA**, por sus siglas en inglés: the *International Society of Automation*. `https://www.isa.org`



(a) Sistema sin controlador



(b) Sistema con controlador

Figura 3.24: Comparación de sistemas.

$$\mathrm{u(t)} = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau)\, d\tau + T_d \frac{de(t)}{dt} \right)$$

Figura 3.25: Acción de control ideal del control tipo PID.

Aplicando la transformada de Laplace a esta expresión obtendremos la función de transferencia del controlador que tendrá la forma indicada en la Figura 3.26. Esta expresión es de utilidad cuando se realiza la identificación del modelo, obteniendo la función de transferencia del sistema y se quiere obtener la expresión de la función de transferencia del sistema en su totalidad en bucle cerrado.

$$\mathrm{Gr(s)} = \frac{U(s)}{E(s)} = K_p + \frac{K_p}{T_i}\frac{1}{s} + T_d K_p s = K_p \left(\frac{T_d s^2 + s + \frac{1}{T_i}}{s} \right)$$

Figura 3.26: Función de transferencia para el controlador de tipo PID.

Para poder aplicar este tipo de controlador en un sistema electrónico hace falta discretizarlo, puesto que este control no se hará de una forma continua en el tiempo, sino que se hará en intervalos finitos, determinados por el tipo de microprocesador utilizado y su frecuencia de funcionamiento. La discretización se hace utilizando el método de las diferencias finitas para aproximar el valor del término derivativo y la integral se aproxima mediante una suma, siendo Δt el período de muestreo, que es el intervalo de tiempo transcurrido entre cada una de las veces en que se aplicará el control al movimiento, esto se resume en la Figura 3.27.

La expresión final del controlador que se implementará en el micro para el control de las patas es la de la Figura 3.28.

$$\int_0^{t_k} e(\tau)\, d\tau = \sum_{i=1}^{k} e(t_i)\Delta t$$
$$\frac{de(t_k)}{dt} = \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$

Figura 3.27: Aproximación de los términos integral y derivativo.

$$\mathrm{u(t_k)} = K_p \left(e(t_k) + \frac{T_s}{T_i} \sum_{i=1}^{k} e(t_i) + \frac{T_d}{T_s}(e(t_k) - e(t_{k-1})) \right)$$

Figura 3.28: Acción de control discretizada implementada en el micro para el control tipo PID.

3.4.2. Desarrollo del control

El *modo monitor* va recopilando datos acerca de los parámetros más significativos del movimiento y los va guardando en la RAM externa de la que dispone el micro. Una vez terminado el movimiento se desactiva este modo y se solicitan estos datos desde el PC. Se realizaron pruebas para los tres ejes de una misma pata, y los ficheros de datos obtenidos de las pruebas de movimiento fueron analizados utilizando MATLAB[7] y se identificó la respuesta de posición como un modelo de primer orden con integrador, por lo que la función de transferencia tendrá la forma mostrada en la Figura 3.29. Todas las gráficas obtenidas durante el proceso del desarrollo del control están incluidas en la parte del anexo correspondiente.

$$\mathrm{G(s)} = \frac{K}{s\,(1 + \tau s)}$$

Figura 3.29: Función de transferencia de un sistema de primer orden con integrador.

Durante las pruebas se tomaron muestras para los movimientos para distintas acciones de control, pudiendo observarse que el modelo cambia sustancialmente según el valor de la tensión aplicada al motor. A partir de estos datos, y de manera empírica, se ajustó un controlador de tipo PD para afinar el control de posición de los motores de las patas, utilizando para ello la identificación el sistema realizada que fue validada usando la herramienta *Simulink*. Las pruebas se hicieron en una única pata, y los parámetros obtenidos del controlador se extrapolaron al resto.

A pesar de conseguir buenos resultados para el movimiento de los ejes cuando se trataba de una única pata, a lo largo de las distintas pruebas realizadas aplicadas al robot en su conjunto, se llegó a la conclusión de que con un único control de tipo

[7]Software de cálculo técnico desarrollado por la empresa *MathWorks*. `http://es.mathworks.com/products/matlab/`

PID para la posición no era suficiente para conseguir una respuesta suave y de esta manera poder sincronizar el movimiento de todas las patas a la perfección. Un controlador que sólo tiene control sobre la posición no es capaz de hacer un movimiento fino del robot en su conjunto, ya que cada eje de cada pata tiene sus peculiaridades, y para conseguir un movimiento coordinado habría que conseguir que los movimientos similares en las distintas patas se realicen todos al unísono. Para la posición, al tratarse de un proceso con integrador, la posición final se alcanzará siempre, lo único que regula el controlador es la acción de control que se va aplicando, cuando lo que se necesita realmente es un control de velocidad para que todas las patas se muevan a una misma velocidad establecida.

Aprendida la lección, se procedió a realizar un control para la velocidad, descartando el uso del controlador de posición. El modelo de la respuesta del sistema en este caso es de primer orden, perdiéndose el integrador al pasar de posición a velocidad. Se optó por utilizar el control todo/nada para la posición, mientras se desarrollaba el controlador de velocidad.

Una vez implementado el código del controlador en el programa del micro y actualizado el esquema de control, se realizaron las pruebas de movimiento y se procedió al ajuste del controlador de forma experimental. Se hicieron pruebas en los tres ejes de una pata, obteniendo para cada uno un controlador distinto. Para el diseño de los controladores, primero se modificó la ganancia, utilizando un controlador proporcional manteniendo la parte integral y la parte derivativa desactivadas, de modo que la respuesta en la velocidad en los momentos iniciales no fuera demasiado brusca. El tiempo integral se introdujo con un valor alto, y se fue disminuyendo hasta que la respuesta en régimen permanente tuviera un error prácticamente nulo con respecto a la referencia, intentando que no se produjeran oscilaciones, y modificando la constante proporcional ligeramente en algún caso. Se intentó afinar el controlador utilizando la acción derivativa, pero debido a que es muy susceptible al ruido de alta frecuencia, y que el método de adquisición de la velocidad se hace discretizando el tiempo en duraciones de los períodos de muestreo y aproximando la velocidad utilizando el cambio en los pulsos desde la última medición mediante $\frac{\Delta x}{\Delta t}$ con una cierta resolución que recibe de la FPGA, las pequeñas variaciones de la velocidad provocaban oscilaciones en la respuesta, sin llegar a mejorar el controlador, por lo que se decidió mantenerse con un controlador PI.

Los parámetros de los controladores obtenidos fueron diferentes dependiendo del eje, pero obteniéndose un buen resultado en ambos sentidos de desplazamiento para cada uno de ellos. El análisis del control de velocidad permitió observar un fenómeno que no era evidente cuando se realizó el control de posición. Se pudo observar que una vez aumentada la velocidad con un salto inicial, la acción de control podía disminuirse manteniéndose la velocidad, esto se puede explicar físicamente debido a la diferencia entre los factores de rozamiento estático y dinámico que provocan la necesidad de aumentar el par en el arranque aumentando la acción de control, pudiendo luego disminuirse ligeramente manteniéndose la velocidad en el valor actual. La limitación más evidente fue que al tratarse de un sistema no

lineal, con zonas muertas y un cierto grado de histéresis, la respuesta empeora si se hacen cambios significativos en la referencia de la velocidad, por lo que tendría que realizar el controlador para un rango de velocidades determinada. Durante la realización del controlador se decidió ajustar su funcionamiento para velocidades medias-altas, consiguiéndose una buena respuesta para velocidades en los rangos de 60 a 120 pulsos/ciclo.

Eje	Kp	T_d **(s)**	T_i **(s)**
X	0,3	0	0,01
Y	0,5	0	0,01
Z	0,5	0	0,01

Cuadro 3.1: Parámetros de los controladores para la velocidad obtenidos para cada eje.

Como resultado final se obtuvo un *control híbrido*: utilizando un sistema todo/nada para la posición que cortara la aplicación del voltaje una vez alcanzado el objetivo, y un controlador PI para la velocidad que se encargará de seguir la referencia de velocidad establecida.

3.4.3. Implementación del control

Partiendo de la base del controlador de posición, se volvió a escribir el código, adecuándolo para la velocidad, y poniendo especial cuidado en hacerlo lo más eficiente posible. El nuevo controlador resultó ser de más rápida ejecucción que el anterior, con menos instrucciones de máquina, puesto que los rangos del valor de la velocidad son más reducidos que los de la posición en pulsos (varios órdenes de magnitud inferiores) permitiendo manejar menos bits de resolución, y sólo se utilizaron operaciones con números de coma flotante cuando era estrictamente necesario.

El sistema implementado utiliza una estructura interna, detallada en la Figura 3.30, para almacenar todos los datos correspondientes al controlador. Todos estos valores se inicializan a cero o a sus valores predeterminados al arrancar el programa del micro, mediante la función *ini_variables*. En la estructura están almacenados los valores de los parámetros de la ganancia proporcional K_c, el tiempo derivativo T_d, y el tiempo integral T_i para cada uno de los ejes.

El valor del tiempo de muestreo, Ts, que se corresponde con cada cuanto tiempo se ejecuta el código del controlador, toma el valor de 4,096mS. Este valor toma el valor del período de ejecución de la interrupción de desbordamiento del *timer0*, ya que está calculado para que cada ciclo del bucle principal, que es donde se realiza el control, se corresponda prácticamente con el disparo del evento de interrupción del contador. Esto se hace mediante el uso de una bandera que se escribe a nivel alto al ejecutarse la interrupción y se comprueba su valor en el bucle principal, volviendo a ponerla a nivel bajo al realizar el control.

```
typedef struct PIDStr
{
//tiempo de muestreo
float Ts;

//errores en distintos tiempos
signed int error_x[3], error_y[3], error_z[3];

 //acciones de control anteriores
unsigned int accion_anterior_x, accion_anterior_y, accion_anterior_z;

//accion integral anterior
unsigned int u_integral_anterior_x, u_integral_anterior_y, u_integral_anterior_z;

//estado del PID (1: activado, 0: desactivado)
uchar x_activado, y_activado, z_activado;

//limites de acciones de control
uchar limite_x, limite_y, limite_z;

//factores de escala
float factor_x, factor_y, factor_z;

//zonas muertas de los motores
uchar zona_muerta_x, zona_muerta_y, zona_muerta_z;

//parametros de los controladores para cada eje
//X
float Kc_x;
float Td_x;
float Ti_x;
//Y
float Kc_y;
float Td_y;
float Ti_y;
//Z
float Kc_z;
float Td_z;
float Ti_z;

} PIDStr;
```

Figura 3.30: Definición de la estructura para el control tipo PID.

Cuando llega una orden de movimiento y el control está en modo PID, se pone a 1 el valor de *x,y,z_activado* usando la función de inicio del controlador del eje correspondiente utilizando la función *IniciaPID* de los micros, esta es la variable que se comprobará a la hora de ver si ha de realizarse el control cada vez que se ejecuta el bucle principal o no. El error de los dos tiempos inmediatamente anteriores al tiempo actual se almacenan en los vectores correspondientes para poder realizar la parte derivativa del control, en el caso de estar definido el tiempo derivativo. Los valores de la acción de control anterior y la acción integral anterior también se almacenan en la estructura, para poder realizar la parte integral del control en caso de ser distinto de cero el tiempo integral.

Los motores tienen una cierta resistencia al giro debido a las inercias de las deslizaderas y los elementos del sistema de distribución de la potencia, por lo que cada eje tiene un valor de acción de control por debajo del cual no se produce el

movimiento. Estos valores son los denominados *zonas muertas* y están almacenados en la estructura de control. Por último, se tienen los límites de la acción de control para limitar el resultado de la acción de control calculada por el controlador a un valor máximo, y el factor de escala que multiplica toda acción de control resultante. Tanto los valores de zonas muertas como los de los límites de la acción de control se utilizaron para afinar los controladores y durante el desarrollo del controlador para la posición, no siendo finalmente necesarios en el control híbrido resultante.

Los valores de los parámetros, las zonas muertas, los límites, y los factores de escala se pueden cambiar desde la ventana destinada para ello en el programa cliente. Se han limitado las operaciones de coma flotante al mínimo indispensable, únicamente a las operaciones de los cálculos de las aportaciones de cada una de las partes del control (parte proporcional, integral, o derivativa) a la acción de control a realizar, trabajando y almacenando valores de tipo *char* o *int* en lugar de números de coma flotante *float*.

Como complemento a la parte de control de velocidad, se realiza una parte de control de posición que se ejecuta en la interrupción del *timer0*, dentro de la función de *ControlLimites*. Además de comprobar los finales de carrera y los límites de movimiento, esta función velará por que se alcance la posición destino, cortando la alimentación del motor una vez esto ocurra. Como se ha comentado anteriormente, al tratarse de un sistema con integrador el de la posición, al aplicar la tensión al motor éste comenzará a moverse alcanzando con total seguridad la posición destino, siendo lo único necesario parar el motor una vez haya sido alcanzada, y de esto es lo que se encarga *ControlLimites*. Cada vez que se realiza la ejecución del manejador de la interrupción, se comprueba la posición recibida de los encoders y si se ha alcanzado o superado la posición destino se realiza el fin del movimiento con prioridad alta, desactivando la ejecución del controlador PID y parando el movimiento de los motores de manera inmediata. Ya que el manejador de la interrupción se puede ejecutar en cualquier momento y a mitad del código del controlador, para evitar que se pueda terminar algún ciclo de control una vez alcanzada la posición, se hacen comprobaciones acerca del estado de activación del controlador durante la ejecución de su código, y además se activa una bandera que no permitirá la aplicación de una nueva acción de control sobre el motor utilizando las funciones para ello destinadas sin volver a iniciarse el PID con una nueva orden de movimiento y desactivando la bandera.

3.5. Desarrollo del servidor Linux embebido

Uno de los objetivos primordiales del proyecto era que se pudiera realizar el control de forma remota. Después de evaluar las distintas alternativas, se decidió dotarlo de un ordenador de a bordo utilizando una *Raspberry Pi 2*[8], un pequeño ordenador de bolsillo que corre el sistema operativo Linux y de gran popularidad en el mundo de los sistemas embebidos. La Raspberry Pi 2 está basada en el circuito integrado de tipo *sistema completo en un solo chip* Broadcom BCM2836, y cuenta con un procesador quad-core ARM Cortex-A7 a 900MHz, un procesador gráfico VideoCore IV dual-core, 1GB de memoria RAM, y almacenamiento en tarjeta de memoria SD. De todas las distintas distribuciones de Linux existentes para la Raspberry, se eligió *Arch Linux*[9], distribución de Linux ligera, potente, y con muchas opciones de personalización. Una vez que estuvo todo configurado, se desarrolló de una aplicación para la Raspberry Pi 2 que se usará como servidor para el ordenador de a bordo.

3.5.1. Configuración del sistema operativo

Una vez descargada la imagen del sistema operativo ArchLinux de la página web oficial en su versión para ARMv7 e instalada en la tarjeta de memoria, se procedió a la instalación del adaptador Wi-Fi y del adaptador Bluetooth. Una vez comprobado que todo funcionaba correctamente y estaba actualizado, fue necesario instalar un ordenador de sobremesa para poder realizar el desarrollo de la aplicación para el sistema embebido. Se optó por utilizar el entorno de desarrollo Eclipse[10], en su versión para desarrollo en lenguaje C++ y bajo el sistema operativo Ubuntu. Para poder desarrollar aplicaciones para la Raspberry se tuvo que configurar la compilación cruzada (*cross-compiling*) en Eclipse, de manera que permitiese escribir código en Ubuntu y mediante las herramientas adecuadas compilar este código para la arquitectura ARMv7, de modo que fuese ejecutable en el sistema embebido.

Por último, y debido a que la idea era ejecutar el servidor en la Raspberry sin ningún tipo de pantalla ni periférico innecesario conectado, se configuró un servicio en Linux para que siempre que se iniciara el sistema operativo se conectara a la red Wi-Fi de la universidad y a través de una cuenta de correo previamente configurada enviara un correo con su dirección IP. Esto nos permitirá establecer una conexión por SSH[11] para poder poner en marcha el servidor siempre que sea necesario, al mismo tiempo que nos permitirá ver todos los avisos y la información que va mostrando por pantalla el servidor durante su funcionamiento.

[8] Sistema embebido creación de la **Raspberry Pi** Foundation. `https://www.raspberrypi.org`

[9] **Arch Linux**, una liviana y flexible distribución de Linux. `https://www.archlinux.org`

[10] **Eclipse**, entorno de desarrollo integrado de código abierto. `https://eclipse.org/ide/`

[11] **Secure SHell**, protocolo de comunicación por Ethernet que nos permite conectarnos remotamente a un sistema operativo compatible y nos brinda la opción de ejecutar programas desde línea de comandos.

3.5.2. Servidor para el control del robot

El servidor para el control del robot está desarrollado utilizando el lenguaje de programación C++ en su revisión del año 2011, *C++11*, utilizando todo el potencial que pone al servicio del programador. Haciendo un uso intensivo de las características disponibles de multithreading[12] y manejo de mutex[13]. Puesto que se trataba de un desafío importante durante el cual surgieron muchas preguntas sobre el desarrollo, se buscaron respuestas en Internet siendo la página web de *Stack Overflow*[14], lugar donde los desarrolladores de software exponen sus dudas sobre cualquier lenguaje de programación y otros usuarios exponen sus respuestas, la que más soluciones aportó.

3.5.2.1. Características

El programa consta de una función principal desde la cual se van realizando todas la acciones y procesando las órdenes que va recibiendo. Todo gira en torno a tres grandes clases: comunicación por puerto serie con la placa de la electrónica de control del robot, servidor para conexión por Ethernet, y comunicación por Bluetooth con mando de PlayStation 3. Al proceso del servidor se le han de pasar tres parámetros para su correcta ejecución, que son el puerto por el cual se producirá la escucha a través de Internet, el número de ciclos que se ejecutarán, y si se va a utilizar el mando para realizar el control. Una vez el programa comprueba los parámetros recibidos, los muestra por pantalla, y procede a la ejecución del bucle principal. En el bucle se crean las hebras que ejecutarán los procesos de comunicación y se encarga de procesar los mensajes que se van recibiendo por Ethernet. El bucle principal se ejecutará tantas veces como se le haya indicado en el parámetro de ciclos, siendo definido un ciclo como un proceso de conexión y desconexión del cliente que se conecta a través de Internet. Todos las comunicaciones y los eventos que se van produciendo se quedan almacenadas en un archivo de registro con el nombre *hexapod_log.txt* en la carpeta donde se esté ejecutando el programa del servidor.

Cada una de las clases de comunicación se instancia en una hebra, o hilo de ejecución, que permitirá que cada uno de los bloques en los que fundamenta el programa se pueda ir ejecutando en paralelo, todos ellos siendo comandados desde el proceso principal. Se tiene un subproceso para la comunicación por puerto serie que se encargará de ir recibiendo, procesando, y enviando datos al UCP del robot. Por otro lado también se produce la comunicación por Ethernet en un subproceso distinto, que se encargará, como en el caso anterior, de asegurar la correcta comunicación bidireccional entre el servidor y el cliente PC. También hay implementada una he-

[12] Con la llegada de **C++11** se estandarizó el uso de hilos de ejecución en las propias librerías del núcleo del lenguaje, para poder distribuir las distintas tareas en subprocesos que se ejecutaran de manera simultánea, conformando así la implementación definitiva de lo denominado como multitasking al estándar del lenguaje C++.

[13] **MUTual EXclusion**, una manera sencilla de asegurar que no se produzcan interferencias en el programa cuando se quiera acceder simultáneamente a una información compartida desde dos hilos de ejecución distintos.

[14] [4] `http://www.stackoverflow.com`.

```
//definiciones de las instancias de las clases y los threads
InternetServer *Internet;
SerialComm *Serie;
Controller *Mando;
std::thread *internet_thread;
std::thread *serial_thread;
std::thread *controller_thread;

//variables globales
mutex m;

//funciones
void PararControl(int sign);
void ServidorInternet(int puerto, int tiempo, bool *cierra_internet, mutex& m);
void ClienteSerie(std::string ruta, int tiempo, bool *cierra_serie, mutex& m);
void MandoPS3(std::string ruta, int tipo, bool *cierra_mando, mutex& m);
```

Figura 3.31: Definiciones del programa principal del servidor de la Raspberry Pi 2.

bra que se encargará de conectarse a un mando de PlayStation 3 vía Bluetooth y de ir registrando los botones que se van pulsando y actuando en consecuencia. Todo el sistema de comunicación es bastante sofisticado, ya que además de asegurarse la correcta comunicación bidireccional por puerto serie y por Ethernet, también es necesario que haya un puente entre ambos canales, puesto que, cuando se produce la correcta identificación en el sistema, los mensajes se intercambiarán entre el PC y el UCP del robot, siendo necesario que se reciban desde Ethernet y se envíen por puerto serie los comandos originados en el PC y se reciban desde el puerto serie y se envíen por Ethernet de vuelta al PC las respuestas del robot a las órdenes que recibe. Debido a que los mensajes han de ser lo más estándar posibles para poder ser enviados directamente de un protocolo a otro sin perder tiempo en la reestructuración de éstos, la organización interna de los mensajes es la misma que la utilizada para la comunicación entre el PC y los micros, la estructura *Mensaje*.

Las clases se instancian de manera dinámica y los punteros a las instancias de las distintas clases están definidas de forma global, de forma que sean accesibles desde todas las funciones del programa. En la Figura 3.31 se pueden observar las definiciones de los punteros que apuntarán a las clases de comunicación y las definiciones de las hebras que serán utilizadas para realizar las distintas tareas en paralelo. La hebra *internet_thread* ejecutará la función *ServidorInternet* en un nuevo subproceso, la hebra *serial_thread* ejecutará la función *ClienteSerie* en un nuevo subproceso, y la hebra *controller_thread* ejecutará la función *MandoPS3* en un nuevo subproceso. La función *ServidorInternet* escuchará a través del puerto indicado y una vez establecida la conexión irá recibiendo los datos y colocándolos en mensajes, también se mostrarán por pantalla todos los datos recibidos con codificación hexadecimal. Estos mensajes que reciba los colocará en una pila y será el bucle principal el encargado de procesarlos. Si no se produce una conexión en un tiempo previamente definido se terminará la función, y con ello la hebra, finalizando el ciclo. Mediante la función *ClienteSerie* se crea una nueva instancia de la clase de comunicación por el puerto serie, si la comunicación con el robot se realiza de manera satisfactoria, se encargará de mostrar por pantalla y procesar los datos que vayan llegando del micro principal del robot a través del puerto serie. La función *MandoPS3* será la que se

encargue de comunicarse a través de Bluetooth con el mando de PlayStation 3. Una vez conectado con el mando, se realizará un muestreo de los eventos y cada vez que se pulse un botón o se mueva un eje la función lo recibirá y actuará en consecuencia. Cuando se realice la desconexión del cliente, el bucle principal iniciará el proceso de finalización de las hebras de comunicación y una vez terminadas se dará por finalizado el ciclo. Las variables booleanas *cierra_serie*, *cierra_mando* y *cierra_internet* se utilizan para la comunicación cruzada entre las distintas hebras, los valores de estas variables se están comprobando en cada iteración del proceso de modo que si ocurre algún suceso que requiera la finalización de la hebra se actualiza el valor de esa bandera y al realizar la comprobación procederá a su finalización. Tras la realización de todos los ciclos solicitados, se dará por terminada la ejecución del servidor. Si se pulsa la señal de parada por excelencia de cualquier línea de comandos, *Control+C*, será la función *PararControl* la encargada de terminar el funcionamiento del servidor de manera adecuada, llevando a cabo todo los procesos de liberación de memoria pertinentes.

```
class SerialComm {
    private:
        //variables
        int handle;
        int posicion_escritura_pila_entrada;
        int posicion_lectura_pila_entrada;
        Mensaje PilaEntrada[N_PILA_ENTRADA];
        struct timeval timeout;
        unsigned char buf[TAM_BUF_ENTRADA];
        int posicion_lectura_buffer;
        int posicion_escritura_buffer;
        //funciones
        void AbrirRuta(std::string ruta, int rate);
        int CambiarBaud(std::string ruta, int rate);
        int EnviaByte(unsigned char byte);
        int EnviaBuffer(unsigned char *buf, int size);
        void LlenaBuffer(unsigned char *buf, int tam);
        unsigned char GeneraCRC(Mensaje msg);
        void PilaMensajesEntrada(Mensaje *msg);
    public:
        //funciones de clase
        SerialComm(std::string ruta, int tiempo); //constructor
        //funciones principales
        bool EstaConectado();
        int RecibeDatos(unsigned char *buf, int size);
        void GeneraMensaje(unsigned int msg_add, unsigned int msg_type,
            unsigned int msg_size, unsigned char *data, Mensaje *generado);
        void EnviaMensaje(Mensaje msg_enviar);
        bool HayMensajes(void);
        void ProcesaDatos(void);
        void ProcesaMensajesEntrantes(Mensaje *msg_internet);
        void Cierra(void);
};
```

Figura 3.32: Definición de la clase *SerialComm*, con sus variables y funciones correspondientes.

3.5.2.2. Clase de comunicación por puerto serie

La comunicación por puerto serie está comandada desde una instancia de la clase *SerialComm* y la estructura, que es muy similar a la que fue implementada en

el programa desarrollado para PC, se muestra en la Figura 3.32.

Cuando se cree la instancia de la clase ha de pasarse al constructor como parámetro la ruta al puerto USB, que tendrá la forma */dev/ttyUSBX*, siendo valor de X en este caso el número 0, al ser el único periférico USB conectado a la Raspberry, y el tiempo de *timeout* para la comunicación. En Linux todo es tratado como un archivo; ya sean ficheros, tarjetas de sonido, particiones de disco, o cualquier otro tipo de periférico. Los dispositivos con los que cuenta el ordenador se encuentran indexados en el directorio */dev*. También es de gran importancia la configuración de un tiempo de *timeout*, ya que si se le da la orden de ponerse a la espera de recibir datos por el puerto y éstos nunca llegan, la ejecución del proceso quedaría congelada en esa línea. Estableciendo este tiempo de espera máximo hacemos que la función retorne cada vez y el ciclo de ejecución no quede interrumpido. El constructor se encarga de realizar esta conexión con los parámetros recibidos. Para ello se realiza la llamada a la función *AbrirRuta* para abrir la dirección del dispositivo y la función *CambiarBaud* para cambiar la *tasa de baudios*[15] utilizada en la comunicación.

Una vez se ha creado la instancia de la clase, mediante el uso de la función *EstaConectado* se comprueba si la conexión se ha realizado correctamente para proceder con la ejecución del resto del proceso. No tiene sentido seguir con la ejecucción del proceso si no se ha producido correctamente la conexión vía USB con el robot por lo que se daría por finalizado el ciclo. Mientras esté abierta la comunicación se procede a la recepción de datos mediante el uso de *RecibeDatos* y se procesan estos datos utilizando la función *ProcesaDatos*. Los datos recibidos por *RecibeDatos* se muestran por pantalla y se almacenan en el búfer de entrada mediante el uso de la función *LlenaBuffer*, actualizando el valor de la posición de escritura, *posicion_escritura_buffer*. La función *ProcesaDatos* va leyendo los datos almacenados en el búfer, almacenándolos en mensajes, y colocándolos en la pila de entrada usando *PilaMensajesEntrada*, todo esto mientras se va actualizando la posición de lectura, *posicion_lectura_buffer*.

Usando *HayMensajes* se comprueba el estado de la pila de entrada, y si hay mensajes sin procesar, se procesan utilizando la función *ProcesaMensajesEntrantes*, que comprobará la validez de cada uno de los mensajes que hay en la pila de entrada usando *GeneraCRC* y si son válidos los irá procesando. *ProcesaMensajesEntrantes* recibe un puntero a una posición de memoria de una variable de tipo *Mensaje*, ya que los mensajes que llegan por el puerto serie han de ser redirigidos al PC, de modo que una vez retorne la función se tendrá el mensaje que ha de ser redireccionado guardado en la variable correspondiente para ser posteriormente enviado al PC a través de la clase de comunicación por Ethernet. La función *GeneraMensaje* no tiene aplicación por este momento, puesto que la Raspberry no tiene que recibir ni enviar mensajes por puerto serie por iniciativa propia. Para realizar el envío de un mensaje se utiliza la función *EnviaMensaje*, que usará internamente la funciones de *EnviaByte*

[15]La **tasa de baudios** expresa la cantidad de información que se transmite por un canal de comunicación cada segundo. La unidad fundamental es el *baudio*, que se corresponde a un símbolo por segundo, siendo un símbolo la unidad fundamental de transmisión del sistema utilizado; ya sean tonos, pulsos, bits, etc.

y *EnviaBuffer* para realizar la comunicación. Esta función se usará para redireccionar los mensajes que llegan del PC por Internet. Por último, la función *Cierra* termina la comunicación una vez se desconecta el cliente que estaba conectado a través de Internet.

```
class InternetServer {
    private:
        //variables
        int port;
        int s_escucha, s_cliente;
        struct sockaddr_in cliente;
        bool estado;
        int contador_conexiones;
        bool password_aceptada;
        unsigned int posicion_escritura_pila_entrada_internet;
        unsigned int posicion_lectura_pila_entrada_internet;
        Mensaje PilaEntradaInternet[N_PILA_ENTRADA_INTERNET];
        FILE *log;
        time_t t;
        struct tm tm;
        //funciones
        void PilaMensajesEntradaInternet(Mensaje *msg);
        unsigned char GeneraCRC(Mensaje msg);
        bool CompruebaContrasenya(Mensaje msg_inicial);
        bool Seguridad(void);

    public:
        //funciones
        InternetServer(void);
        void Inicializa(void);
        void IniciaEscucha (int port);
        void TerminaEscucha(void);
        int ConexionEstablecida(int tiempo);
        bool EstadoConexion(void);
        void TerminaConexion(void);
        void RecibeDatos(char *buf_in, int tam_buf);
        void EnviaDatos(char *buf_out, int tam_buf);
        int RecibeMensajes(char *buf_msg, int tam_msg);
        void EnviaMensaje(Mensaje msg_salida);
        void ProcesaMensajesEntrantesInternet(Mensaje *msg_serie);
        void GeneraMensaje(unsigned int msg_add, unsigned int msg_type,
            unsigned int msg_size, unsigned char *data, Mensaje *generado);
};
```

Figura 3.33: Definición de la clase *InternetServer*, con sus variables y funciones correspondientes.

3.5.2.3. Clase de comunicación por Internet

Todo el proceso de comunicación por Internet se controla desde una instancia de la clase *InternetServer* con una organización interna muy parecida a la que cuenta la clase utilizada en el programa desarrollado para PC, su estructura se muestra en la Figura 3.33.

Una vez creada la instancia de la clase mediante el constructor, se utiliza la función *Inicializa* para inicializar todas las variables de la clase y, puesto que esta es la primera clase que se instancia, es la encargada empezar el archivo de registro de actividad. Una vez inicializado, se pone a la escucha por el puerto seleccionado mediante la función *IniciaEscucha*. Después de la llamada a esa función, el programa intenta establecer conexión utilizando la función *ConexiónEstablecida*. Para evitar

que el proceso se quede colgado, hay un *timeout* de 20 segundos en la escucha y si después de este tiempo no se ha producido ningún intento de conexión, el ciclo se da por finalizado. Con la función *EstadoConexion* se comprueba el estado de la comunicación, y mientras esté ésta abierta se intentarán recibir datos mediante *RecibeMensajes*, teniendo establecido un *timeout* de 10 segundos. Si se desconecta el cliente bruscamente la función comunicará este evento y se finalizará la conexión por parte del servidor también, finalizando así un ciclo. En este caso los datos recibidos no se colocarán en un búfer de entrada, sino que se introducirán en la estructura del mensaje y se colocarán directamente en la pila de entrada usando *PilaMensajesEntradaInternet*. El procesado de los mensajes de la pila de entrada se realiza con llamadas a la función *ProcesaMensajesEntrantesInternet* que, igual como sucedía en la función de procesado del puerto serie, se le pasa un puntero a una variable de tipo *Mensaje* de modo que, cuando la función termine su ejecución, los mensajes que tengan que ser reenviados por el puerto serie estén disponibles y se envíen con la función correspondiente. El envío de mensajes se hace a través de la función *EnviaMensaje*, que se llama o bien cuando se recibe por el puerto serie un mensaje que ha de ser reenviado por Internet o cuando se produce la comunicación inicial entre el PC y el ordenador de a bordo. Las funciones *GeneraMensaje* y *GeneraCRC* se usan para crear estos mensajes para la comunicación entre el PC y la Raspberry, mientras que los mensajes recibidos desde el puerto serie son enviados directamente a *EnviaMensaje*. Cuando se recibe la orden de cerrar la comunicación por parte del cliente o cuando se pierde repentinamente la comunicación, se llama a la función *TerminaConexion* y posteriormente a *TerminaEscucha* para así finalizar la conexión y dar por concluido el ciclo.

La comunicación por Internet usando la clase *InternetServer*, como todo servidor que se precie, tiene un sistema básico de identificación por contraseña. Toda solicitud de comunicación que se establezca entre un cliente y el servidor ha de venir acompañada por un mensaje de identificación con una contraseña, si el mensaje viene con una contraseña incorrecta o no se envía la contraseña tras la solicitud de conexión, se termina la conexión de manera inmediata. Una vez realizada la identificación con éxito, se puede realizar la comunicación entre el robot y el cliente de manera bidireccional. La función *CompruebaContrasenya* se encarga de comprobar si la contraseña recibida en el mensaje es la correcta y la función *Seguridad* comprueba si la contraseña ha sido aceptada, finalizando la comunicación en caso contrario. Este es un sistema de seguridad básico, pero necesario para conseguir evitar comunicaciones no autorizadas.

3.5.2.4. Clase de comunicación por Bluetooth

La clase de comunicación por Bluetooth, denominada *Controller* y presentada en el figura Figura 3.34, es una novedad en cuanto a la comunicación se refiere respecto de lo desarrollado para el programa de PC.

Igual que en el caso de la comunicación por puerto serie, para abrir la comunicación por el mando hay que indicar el archivo apropiado dentro del sistema de ficheros de Linux, que para los controladores de tipo *joystick* tiene la forma */dev/input/jsX*,

```
class Controller {
    private:
        //variables
        int handle;
        bool estado;
        //funciones
        void AbrirRuta(std::string devicePath);
    public:
        //funciones de clase
        ~Controller(); //destructor
        Controller(std::string ruta); //constructor
        //funciones principales
        bool EstaConectado();
        bool MuestreoEvento(EventoMando* evento);
        int TipoEvento(EventoMando evento);
        void CerrarComunicacion(void);

};
```

Figura 3.34: Definición de la clase *Controller*, con sus variables y funciones correspondientes.

siendo en este caso el 0 el índice del dispositivo, puesto que es el único periférico de este tipo conectado al sistema. Al constructor se le pasa como parámetro la ruta del mando, y mediante el uso de la función *AbrirRuta*, de manera similar a lo que ocurría con el puerto serie, se abre el fichero del mando. Usando el *EstaConectado*, se comprueba si está la conexión abierta y se procede a recibir los datos que envíe el mando. Para recibir datos del mando se utiliza la estructura *EventoMando*, definida en la Figura 3.35. Se realiza lo que se denomina un *muestreo* de un evento del mando utilizando la función *MuestreoEvento* que recibe un puntero a una variable de tipo *EventoMando* vacía y la devuelve con los datos que ha recibido del mando. Mediante el uso de *TipoEvento* se comprueba la naturaleza del evento muestreado y se actúa en consecuencia. La función *CerrarComunicacion* se encarga de cerrar la comunicación con el mando cuando se pulse el botón adecuado.

```
typedef struct EventoMando {
        //tiempo del evento (milisegundos)
        unsigned int time;
        //valor del evento. botones: 1 (pulsado), 0 (arriba). ejes: -32768, 0, 32767.
        short value;
        //tipo de evento
        unsigned char type;
        //identificador del boton/eje
        unsigned char number;
} EventoMando;
```

Figura 3.35: Definición de la estructura *EventoMando*.

Cuando se pulsa algún botón o se mueve algún eje del mando, mediante *MuestreoEvento* se rellena la estructura *EventoMando* tiene toda la información acerca del evento. En *time* se almacena el tiempo en que se ha producido el evento en milisegundos, tomando como origen el inicio de la comunicación. Según sea un botón lo que se ha pulsado o un eje que se ha modificado su posición, se trata de un evento u otro, y por tanto el tipo de evento ocurrido se almacenará en *type*. El valor que toma el evento se almacena en *value*. Si se trata de un botón, tomará el valor 1 ó 0

si el botón ha sido pulsado o soltado, respectivamente; y si se trata de un eje que ha sido desplazado, el valor tomará un valor comprendido entre -32768 y 32767, según la posición que ha alcanzado.

3.5.3. Ubicación sobre el robot

Una vez configurado el ordenador de a bordo se colocó sobre el chasis del robot, más concretamente en la zona central, justo debajo de la electrónica de control. Las conexiones realizadas sobre él se redujeron al mínimo, adquiriendo adaptadores USB para la conexión Bluetooth y WiFi de reducidas dimensiones. También es el encargado de proporcionar la alimentación USB necesaria para el funcionamiento de la electrónica de control, realizándose a través del mismo cable USB utilizado para la comunicación serie.

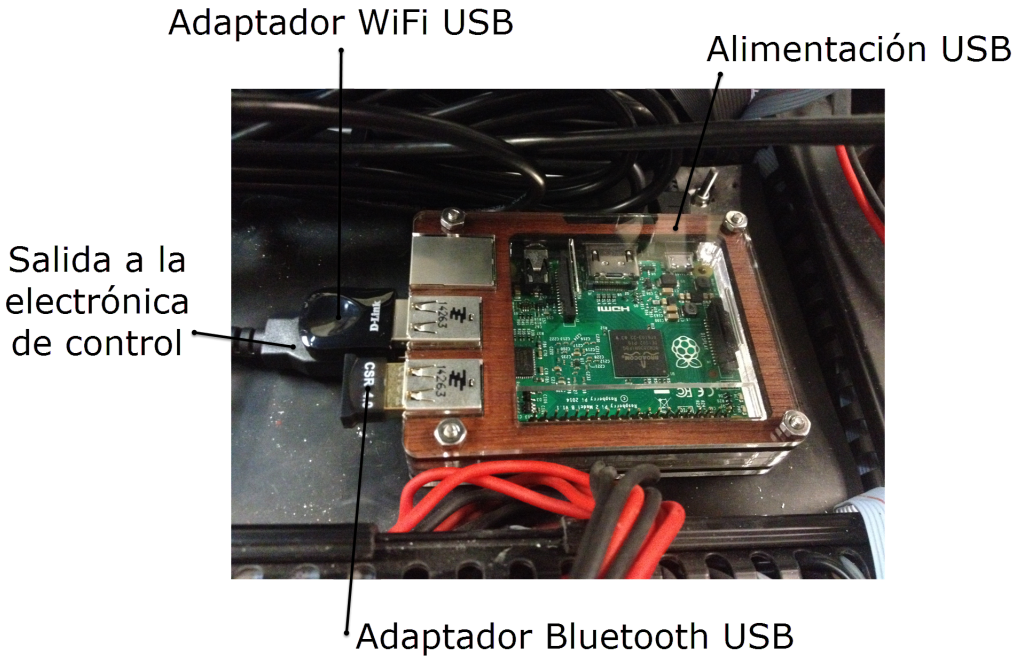


Figura 3.36: Vista general del ordenador de a bordo una vez instalado.

3.6. Planificación de movimientos

Utilizando las piezas que fueron proporcionadas por el tutor se realizó el ensamblaje del modelo virtual del robot en el programa *SolidWorks*®[16], para posteriormente ser simulado su movimiento en *COSMOSMotion*™[17]. Una vez simulado este movimiento se obtuvieron las órdenes necesarias y se pasó a probarlo sobre el modelo real. Para realizar una buena definición las uniones del movimiento se precisa hacer uso de la versión *COSMOSMotion*™ del año 2007 que tiene las herramientas adecuadas. Las versiones posteriores ya no disponen de dichas herramientas, pero se utilizará la versión más actual, la versión 2014-2015, para tener un mejor acabado visual durante la toma de instantáneas y la grabación de vídeos del modelo una vez completamente definido.

3.6.1. Modelo virtual

El primer paso para poder realizar la planificación de los movimientos fue ensamblar el modelo en *SolidWorks*®. El tutor preparó las piezas que conservaba de cuando fue realizado el proyecto una década atrás para que se pudieran utilizar en las nuevas versiones del programa de CAE. Las piezas del robot han sido reducidas a sólo aquellas que tienen movimiento relativo entre ellas, simplificando todo lo demás, pero intentado que el resultado sea lo más cercano posible al robot real. Mediante la utilización de relaciones de posición, llamadas *Mates*, se consiguió ensamblar todas las piezas de las patas para conseguir disponer de un modelo virtual del robot a escala con la misma movilidad que la que posee en la realidad. Este modelo virtual sirve como base para la parte de simulación del movimiento y se colocará por defecto con todas las patas con las deslizaderas X y Z en las posición media de su movimiento y el eje Y de tal forma que el robot se encuentre a 200mm de altura sobre el suelo, esta posición se denominará *Poner esperando*, y de ella partirán los distintos movimientos.

3.6.2. Simulación de movimiento

La simulación del movimiento se realiza utilizando el complemento *COSMOSMotion*™. Lo primero que hay que hacer cuando se abre el modelo en este complemento es definir las piezas del modelo como piezas móviles, de lo contrario y al no haber definido nada aún en este entorno se importan por defecto como piezas fijas.

En el modelo en *SolidWorks*® se colocan todas las distintas piezas en sus posiciones de las cuales se parte para la simulación, pero al abrir el modelo en *COSMOSMotion*™ las relaciones de posición no sirven y han de volver a definirse dentro de este entorno. Para definir los movimientos de las piezas de una forma que se puedan utilizan, han de usarse las llamadas restricciones, *Constraints*; pudiendo ser de tipo

[16] **SolidWorks** es un programa de *CAE* desarrollado por la empresa francesa *Dassault Systèmes*. `http://www.solidworks.es`

[17] **COSMOSMotion** es una extensión del programa *SolidWorks* que permite realizar simulaciones de movimiento de los modelos ensamblados en éste.

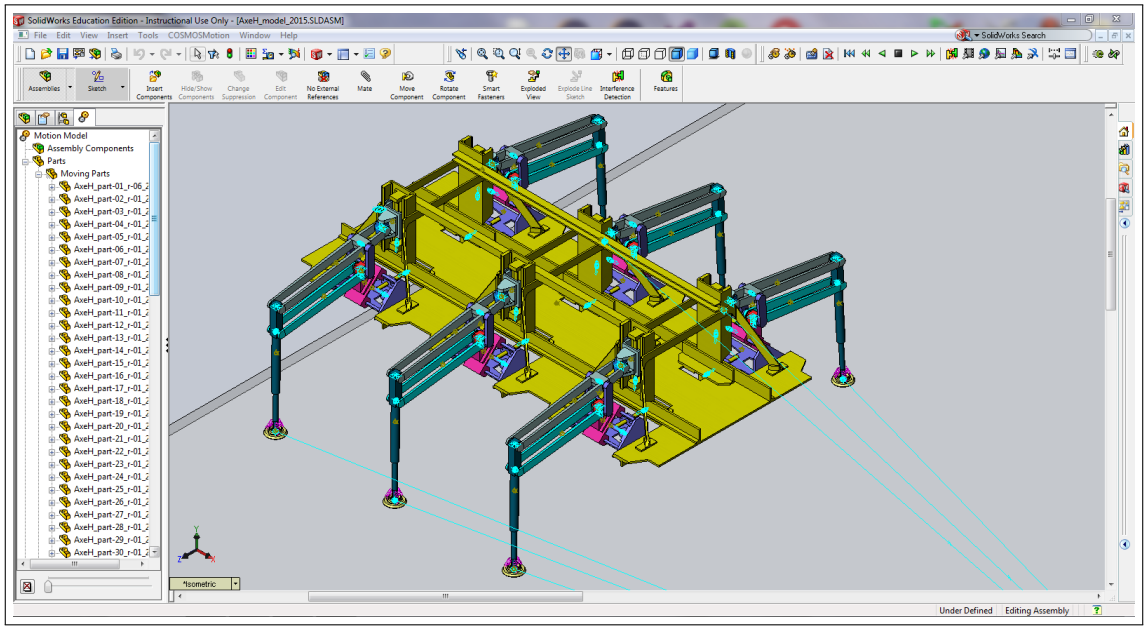


Figura 3.37: Representación del modelo virtual en la posición de esperando ensamblado en el entorno de *SolidWorks*®.

unión, *Joint*, o de contacto, *Contact*. Se utilizaron distintos tipos de uniones para juntar todas las distintas piezas unas con otras: uniones de revolución para conectar las distintas partes que forman la pata, uniones traslacionales y cilíndricas para unir las deslizaderas a los husillos por donde se desplazan, y uniones esféricas para unir el pie de la pata al resto del mecanismo. También se definieron los contactos entre los pies de las patas y el suelo para limitar el movimiento vertical de éstas.

Una vez preparado el modelo en *COSMOSMotion*™, esto es, correctamente definido y sin restricciones en exceso, se puede proceder a la simulación de su movimiento. Se realizaron simulaciones de cuatro movimientos principales y de sus correspondientes posiciones iniciales: *trípode alternante frontal*, *trípode alternante lateral*, *onda*, *plegar robot*, y *levantar robot*. En todas las simulaciones realizadas se parte de la posición de espera definida en el modelo en *SolidWorks*®, y de ahí se lleva a la posición inicial del movimiento y realiza el movimiento en ambos sentidos.

La definición del movimiento en el entorno de *COSMOSMotion*™ se realiza definiendo acciones sobre las uniones, indicando el movimiento a realizar en milímetros y el tiempo empleado para ello. Con el fin de identificar las uniones de las deslizaderas se les asignan nombres propios representativos y sobre ella se irán definiendo las secuencias de movimiento, utilizando comandos de *Step*, que permiten ir concatenando los distintos movimientos de cada una de las deslizaderas tiempos estipulados para conseguir el resultado final deseado.

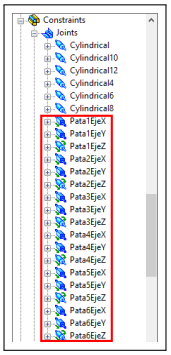


Figura 3.38: Nombres de las uniones de las deslizaderas en el modelo virtual.

3.6.3. Implementación en el micro

Una vez comprobado que el resultado de la simulación era el adecuado, se procedió a implementar las secuencias de movimiento en el micro principal para probar su rendimiento en el robot real. Hay dos opciones distintas para realizar los movimientos: usando el control todo/nada o utilizando el controlador de velocidad tipo PID. El control todo/nada permite modificar directamente la acción de control según sea necesario dar más potencia o menos al movimiento de los motores, mientras que el control de velocidad permite fijar una referencia para la velocidad de movimiento de las patas en pulsos por ciclo que el controlador PI implementado seguirá. En ambos casos cuando se llega a posición destino se termina el movimiento del motor con un comando de alta prioridad que se ejecuta en una interrupción.

Los modos de movimiento, bien sean funciones propias de movimiento o las denominadas posiciones iniciales, se estructuran en secuencias en las cuales se mandan los comandos de movimiento a las patas y se espera a que todas finalicen la secuencia ordenada. Todos los movimientos se realizan mediante llamadas a las funciones *MuevePataX*, *MuevePataY*, y *MuevePataZ*; mientras que las esperas se realizan dentro de las funciones *EsperaPata* y *EsperaPataTodo*. En las funciones para mover los ejes de las patas, se indica la pata a mover y los distintos parámetros: de posición, velocidad, si se ha dejar libre el movimiento del motor una vez alcanzada la posición final, y para el eje Y si ha de seguir al tocar el suelo. Una vez enviados los comandos de movimiento, se ha de esperar a que cada una de las patas llegue a la posición solicitada, y esto se hace mediante el uso de la función *EsperaPata* en el caso de que sea un sólo eje de una pata concreta al que se está esperando o usando *EsperaPataTodo* si es el movimiento de los tres ejes de una pata el que ha de llegar a su destino.

Las órdenes de movimiento suelen corresponderse con secuencias de movimientos simultáneos de tres patas en el aire, mientras las otras tres se mantienen en el suelo. El robot mantiene el equilibrio mientras tenga tres patas en el suelo, bien las patas con numeración impar (1, 3, y 5), o las patas con numeración par (2, 4, y 6). Después de cada secuencia se manda la orden de espera para asegurarse de que todas las patas alcanzan la posición a la que han sido enviadas, de lo contrario no se producirá el inicio de la siguiente secuencia. Estas secuencias están clasificadas

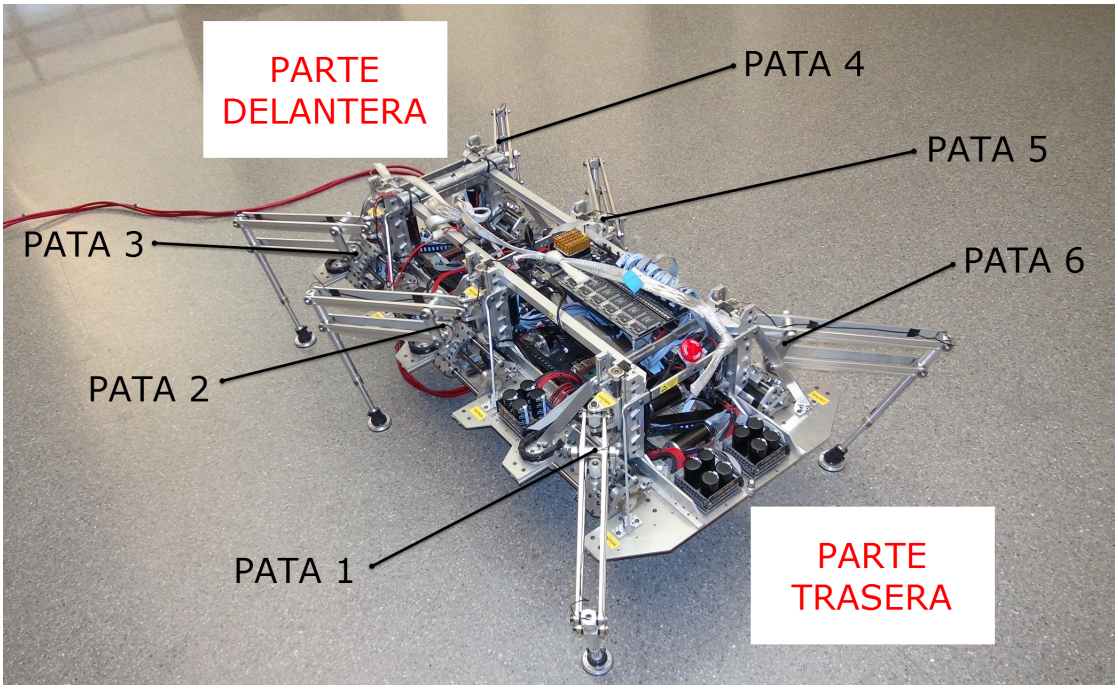


Figura 3.39: Disposición y numeración de las patas del robot.

en dos grupos: el de los *movimientos de posición inicial* y levantar robot, y el de los *movimientos de avance*.

3.6.3.1. Consideraciones acerca del movimiento

Cada uno de los modos de movimiento del robot tiene estipulados unos límites, tanto para la altura a la que ha de realizarse como para las velocidades o acciones de control que aplicar al movimiento. Si se recibe un valor fuera del rango establecido, desde la propia función del micro principal se saturará a su valor máximo o mínimo correspondiente. Además existe también una comprobación de los límites de movimientos en el código de los micros de las patas, de esta forma nunca se podrá solicitar al robot hacer un movimiento que exceda sus posibilidades.

Se ha establecido una distancia mínima sobre el suelo de 80mm puesto que la conexión de las baterías de 24V a la etapa de potencia general se realiza por la parte inferior del chasis y hay que dejar una distancia al suelo para no interferir con el conector, además de que para poder levantar las patas y realizar el movimiento en el aire es necesario dejar un margen para el movimiento de la deslizadera Y. Si no fuera por esta limitación, el robot puede descender hasta prácticamente tocar el suelo con la parte inferior.

Para realizar cualquier movimiento es necesario que el robot esté en la posición inicial adecuada o que el último movimiento efectuado sea de ese tipo, sin importar el

sentido de movimiento. Con el fin de que el movimiento sea fácilmente reproducible, todo movimiento empieza y acaba en la posición de su posición inicial correspondiente. De esta forma, una vez iniciado un modo de movimiento se puede realizar el mismo en ambas direcciones sin volver a llevarlo a la posición inicial. También se han hecho algunas implementaciones de modos de movimiento extra, que se han probado en el aire a la espera de realizar las correspondiente simulaciones y verificar su viabilidad para su realización sobre el suelo.

```
//Funciones de movimiento
void MuevePataX (uchar num_pata, uint pos_pie, uint vel_x, uchar final_libre);
void MuevePataY (uchar num_pata, uint pos_pie, uint vel_y, uchar suelo_sigue, uchar final_libre);
void MuevePataZ (uchar num_pata, uint pos_pie, uint vel_z, uchar final_libre);
uchar EsperaPata(uchar num_pata, uchar eje);
uchar EsperaPataTodo(uchar num_pata);
void CambiarControl(uchar tipo_control);
void HacerResetRobot(uchar velocidad_reset);
void LevantarRobotTodasPatas(uint posicion, uchar vel);
void PosicionInicial(uchar velocidad);
void MoverRobot(uchar velocidad_suelo, uchar velocidad_aire, uchar velocidad_y);
void CaminaTripodeAlternanteFrontal(uint altura);
void CaminaTripodeAlternanteLateral(uint altura);
void BailaLado(void);
void BailaArriba(void);
void CaminaOnda(uint altura);
void Gira(uint altura);
```

Figura 3.40: Definición de las funciones de movimiento del UCP.

Las funciones iniciales están definidas dentro de la función *PosicionInicial*, la modificación de la distancia al suelo se hace desde la función *LevantarRobotTodasPatas*, y el movimiento de avance se centraliza en la función *MoverRobot*. Desde la función *MoverRobot* se llama a las distintas funciones de movimiento *CaminaTripodeAlternanteFrontal*, *CaminaTripodeAlternanteLateral* y *CaminaOnda*. Es desde estas funciones de movimiento desde las que se realizan las llamadas a *MuevePataX*, *MuevePataY*, y *MuevePataZ* para mover los distintos ejes de las patas y a las funciones *EsperaPata* y *EsperaPataTodo* para realizar las esperas. Las funciones de movimiento *BailaLado*, *BailaArriba*, y *Gira* están en proceso de desarrollo y validación para una futura implementación.

Para ir teniendo constancia del estado actual del robot y de los movimientos que se hacen, se definió la estructura *MovimientoRobot* que contiene los datos principales del movimiento. En *tipo_inicial* y en *tipo_movimiento* se tienen el tipo de movimiento inicial y de movimiento de avance a realizar respectivamente, y una vez realizados se pondrán a cero. En *modo_inicial* se almacena el tipo de movimiento inicial realizado, para que el sistema sepa para qué tipo de movimiento está correctamente configurado y listo para moverse cuando llegue la orden. Este valor se actualiza cuando se ejecuta la función *PosicionInicial* y se mantiene mientras se realice el tipo de movimiento de avance indicado con el movimiento inicial correspondiente. También se tienen los distintos valores para la acción de control (en modo todo/nada) o para la referencia de velocidad (en modo PID) de los movimientos en las siguientes variables: *velocidad_levantar*, valor para el comando de levantar; *velocidad_reset*, valor para el reset de todas las patas (en este caso únicamente acción

```
typedef struct MovimientoRobot
{
    uchar tipo_inicial;
    uchar modo_inicial;
    uchar tipo_movimiento;

    uint altura_robot;
    uchar sentido;
    uchar num_pasos;

    uchar velocidad_levantar;
    uchar velocidad_reset;
    uchar velocidad_inicial;
    uchar velocidad_movimiento_y;
    uchar velocidad_movimiento_aire;
    uchar velocidad_movimiento_suelo;

} MovimientoRobot;
```

Figura 3.41: Estructura con los detalles del movimiento del UCP.

de control); *velocidad_inicial*, valor para realizar el movimiento a la posición inicial; *velocidad_movimiento_y*, valor para levantar y bajar el eje Y de la pata durante el movimiento de avance; *velocidad_movimiento_aire*, valor para mover los ejes X y Z en el aire; y *velocidad_movimiento_suelo*, valor para mover los ejes X y Z en el suelo. Por último, se almacenan también los parámetros de la altura a la que realizar el movimiento, en *altura_robot*, el sentido del movimiento (hacia adelante o hacia atrás y hacia la derecha o hacia la izquierda), en *sentido*, y el número de pasos a realizar, en *num_pasos*.

3.6.3.2. Posiciones iniciales

Los movimientos de posición inicial y de levantar robot tienen la característica de que no se produce un movimiento de avance del robot con respecto al suelo, únicamente se modifica el valor de la distancia al suelo del robot o se colocan las patas en una configuración que posteriormente permita un movimiento de avance. Después de terminar la ejecución de las secuencias de los movimientos iniciales se le da un valor determinado a una variable que será la que indique cuando se le envíe el comando de movimiento si es compatible con la posición actual. Los movimientos de posición inicial tienen menos consumo de potencia que los de movimiento de avance, mientras que el movimiento para modificar la altura del robot produce una demanda de potencia significativa al tener que mover todo el peso del robot en dirección vertical. El valor de la acción de control o la velocidad de referencia se ha indicar al enviar el comando de movimiento, y para todas las secuencias se trata del mismo valor.

La posición *poner esperando* está dividida en ocho secuencias de movimiento, cuatro secuencias para las patas pares y cuatro para las patas impares. Las patas con numeración par son las primeras que se colocan en posición: primero se levantan del suelo, se colocan después en la posición central del recorrido de la deslizadera X, posteriormente en la posición media de la deslizadera Z, y por último descienden

hasta tocar el suelo. Una vez realizado este proceso, se repiten las mismas secuencias para las patas de numeración impar. De esta forma se quedan todas las patas a la altura previamente definida y en una posición intermedia de las deslizaderas, de modo que sea compatible con cualquier otra posición que se quiera realizar posteriormente.

El modo de movimiento que modifica la altura del robot respecto al suelo, denominado *levantar robot*, envía la orden de movimiento del eje Y de las patas a todas al mismo tiempo. Se han establecido unos límites para la altura a la que ha de situarse el robot, que está comprendida entre 80 y 320mm. En este caso no está definido el movimiento en secuencias, sino que se envía directamente la orden a todas las patas de la manera más simultánea posible y espera a que todas envíen la confirmación de que han llegado a la posición destino.

Para poder realizar el transporte del robot, se implementó la posición de *plegar robot* que acerca las patas lo máximo posible al chasis el robot. Este modo se diseño de manera que el robot ocupe el menor volumen posible, pero al mismo tiempo que sea capaz de mantener el equilibrio estático y que pueda realizarse sobre el suelo. Debido a las incompatibilidades de movimiento existentes en el robot, se decidió que este movimiento de posición inicial se realizara a una altura determinada que además fuera adecuada para su transporte. Si el robot no se encuentra a la altura de 200mm sobre el suelo, lo primero que se hará será llevarlo a esa altura utilizando el comando de *levantar robot*, y posteriormente se realizarán las seis secuencias que componen el movimiento. De nuevo se comienza por las patas con numeración par: se levantan del suelo, las patas 2 y 4 se llevan a uno de sus extremos de movimiento de la deslizadera X y la pata 6 al otro mientras que se recoge la deslizadera Z de las tres patas a la posición más retraída de la deslizadera Z, y finalmente se bajan las patas hasta tocar el suelo. Lo mismo se repite con las patas con numeración impar.

El movimiento *trípode alternante frontal* tiene una posición inicial sin límites especiales de comprobación y se realiza en ocho secuencias de movimiento. El movimiento frontal es el más problemático de todos, puesto que produce un cierto desequilibrio dinámico del robot. Se va a presentar la solución final empleada para el movimiento, puesto que tras realizar distintas pruebas se optó por poner las patas de los extremos más retraídas y las patas centrales más alejadas del chasis con el fin de mejorar la estabilidad. Las secuencias comienzan por las patas con numeración impar: levantando patas, llevando la posición de la deslizadera en el eje X 25mm por debajo de la posición media, realizando la configuración anteriormente descrita para el eje Z de donde no se moverá durante el movimiento frontal, y bajando finalmente la pata hasta llegar al suelo. Lo mismo se repetirá con las patas con numeración par, llevando la posición de sus deslizaderas del eje X a 25mm por encima de la posición media.

La posición inicial para el movimiento de *trípode alternante lateral* tiene una comprobación de la altura máxima a la que ha de realizarse, siendo el valor de ésta 300mm, para evitar alcanzar una posición de incompatibilidad conocida relacionada con las posiciones de las deslizaderas Y y Z, en caso de estar en una altura superior

lo primero será enviar el comando adecuado para llevar el robot a la altura compatible. El movimiento se compone de seis secuencias, y comienza esta vez por las patas con numeración impar: levanta las patas, mueve la deslizadera X a su posición central de donde no se moverá durante el movimiento de avance lateral, mueve las deslizaderas del eje Z patas 1 y 3 a una posición 20mm por encima del valor medio y la pata 5 a una posición 20mm por debajo del valor medio, y descienden las patas hasta tocar el suelo. La misma configuración se realiza con las patas con numeración par, llevando las patas 4 y 6 a una posición 20mm por encima del valor medio y la pata 2 a una posición 20mm por debajo del valor medio en el eje Z.

La posición inicial del movimiento de avance *onda* es idéntico a la posición de *poner esperando*.

3.6.3.3. Movimientos de avance

Los movimientos de avance se producen una vez el robot esté en configuración de posición inicial para los distintos modos implementados: trípode alternante frontal, trípode alternante lateral, y onda. Los movimientos se han implementado en ambos sentidos, avance hacia adelante y hacia atrás, y avance hacia la derecha y hacia la izquierda. Se pueden elegir tres valores de acción de control o velocidad en pulsos por ciclo: el valor correspondiente al desplazamiento en el suelo, el valor correspondiente al desplazamiento en el aire, y el valor para realizar el desplazamiento vertical de las patas.

El movimiento de *trípode alternante frontal* realiza el movimiento del eje X del robot, permitiendo avanzar hacia adelante y hacia atrás, manteniendo la altura y la posición de eje Z. Se produce el movimiento con tres patas en el aire, mientras que la otras tres se encuentran sujetando al robot sobre el suelo. El movimiento está dividido en ocho secuencias, y se produce el movimiento de la siguiente manera: las patas con numeración par se levantan y avanzan horizontalmente, cuando éstas están en el aire las patas con numeración impar situadas en contacto con el suelo realizan el desplazamiento a lo largo de sus ejes X para avanzar, y las patas con numeración par descienden hasta tocar el suelo. Estas secuencias se repiten de nuevo, moviendo las de numeración impar en el aire y las de numeración par en el suelo.

El desplazamiento lateral se realiza con el *trípode alternante lateral*, permitiendo al robot dar pasos a la derecha y a la izquierda. Se sigue la misma dinámica que en el caso anterior: las patas con numeración par se levantan del suelo y avanzan transversalmente en el aire mientras que las patas con numeración impar que están en contacto con el suelo se desplazan transversalmente permitiendo el avance el robot con respecto al suelo, finalmente las patas con numeración par bajan hasta alcanzar el nivel del suelo. Esto se repite con el orden de numeración de patas contrario.

El movimiento de *onda* es el más sencillo de todos, simplemente se van levantando y desplazando las patas una a una siguiendo la numeración establecida y

finalmente cuando todas se han desplazado se produce el movimiento de avance del robot desplazando la deslizadera del eje X al mismo tiempo en todas las patas.

3.7. Desarrollo del programa cliente para Android

Durante el curso académico, aprovechando la necesidad de implementar el control telemático del robot y ya que el desarrollo del servidor estaba bastante adelantado, se aprovechó una asignatura de programación de dispositivos móviles para crear un cliente para un tableta con el sistema operativo Android utilizando el lenguaje de programación Java, y el entorno de desarrollo Android Studio[18]. Debido a que la comunicación entre el robot y el ordenador de a bordo no estaba terminada, lo que se realizó fue un prototipo, probando la conexión entre el servidor y el dispositivo móvil. A la hora de probar el movimiento, esta vez con el servidor terminado y en funcionamiento, tuvo que adecuarse el código y la interfaz gráfica de la aplicación Android a los nuevos requerimientos. De esta forma pudo conseguirse una aplicación capaz de comunicarse con el robot para enviar órdenes de movimiento y recibir las pertinentes respuestas.

3.7.1. Programa cliente prototipo

El programa desarrollado inicialmente no era más que una *prueba de concepto*, para comprobar que se podía desarrollar un cliente con las características generales deseadas. El programa tiene unas clases correspondientes a la gestión de las distintas pantallas que se muestran al usuario, además de unas clases propias para realizar la comunicación: ClienteInternet, Mensaje, y Globales.

La clase *Mensaje* tiene una adaptación a Java de la definición de la estructura de los mensajes utilizados para la comunicación entre los distintos programas. Mediante la clase *ClienteInternet* se realizan todas las comunicaciones a través de Internet entre el cliente y el servidor utilizando Sockets. La clase *Globales* contiene el estado actual de la instancia de la clase *ClienteInternet* que se está utilizando en la comunicación, de esta forma se puede compartir la misma instancia de la clase entre las distintas actividades y mantener la comunicación abierta desde que se pulsa el botón de conectar hasta que se desconecta, pudiendo enviar mensajes con los distintos comandos desde las distintas actividades sólo siendo necesaria la identificación en primera instancia. Una explicación más detallada puede encontrarse en el *manual del usuario avanzado* adjunto o consultando la memoria redactada acerca de dicho trabajo[19].

3.7.2. Adecuación a los nuevos requerimientos

Las pantallas se modificaron para adaptarse a los modos de funcionamiento y las órdenes que ha de recibir el robot, puesto que cuando se realizó el prototipo no estaba todo completamente planificado. Además de esto se implementó una modificación importante del código original, que supuso la introducción de una nueva clase

[18] Entorno de desarrollo gratuito de aplicaciones para dispositivos Android. `https://developer.android.com/sdk/index.html`

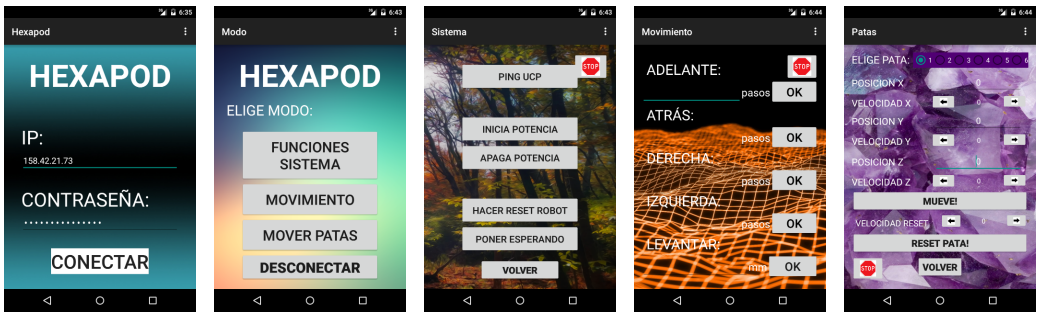
[19] [2] (Oliver G. & Uriel A., 2015)

y permitió un funcionamiento más fluido del programa.

Una de las principales limitaciones del prototipo era cómo se realizaba el proceso de envío y recepción de mensajes. Los mensajes se enviaban y se recibían desde el hilo de ejecucción de la actividad con interfaz gráfica que se estaba ejecutando en ese momento, provocando ciertos retardos en la respuesta de la aplicación, además de ser inviable en cuanto aumentara el tráfico de datos. Este problema se pudo solucionar empleando una nueva clase especial para la recepción de datos y el procesado de mensajes.

Para poder realizar la recepción y procesado de mensajes en otro hilo de ejecución, se hizo uso de la clase *AsyncTask*, que permite realizar una tarea asíncrona, que se ejecute de manera independiente del resto de la interfaz gráfica. Esta clase, denominada *HiloEjecucion*, se encargará de recibir datos a través de Internet, de colocarlos en mensajes de la pila de entrada, y de ir procesando estos mensajes. De esta manera se puede conseguir la misma dinámica que la empleada en el programa cliente para Windows, donde se separan los procesos de interfaz gráfica y procesado interno.

También se incluyó en las clase de variables globales una serie de atributos para poder compartir entre el proceso principal y el hilo de procesado de mensajes los distintos parámetros de movimiento, como el número de pasos, el tipo y la dirección de movimiento, así como si se estaba realizando el control de todo el robot o de las patas de manera independiente.



(a) Pantalla de inicio. (b) Selección de modo. (c) Funciones de sistema del robot. (d) Control de movimiento del robot. (e) Movimiento independiente de las patas.

Figura 3.42: Vista de las pantallas del cliente para dispostivos Android.

3.8. Desarrollo del programa cliente para el mando

Estando la clase de comunicación por Bluetooth implementada en el servidor del ordenador de a bordo, fue necesario desarrollar el código que procesara las órdenes que se fueran recibiendo del mando. A diferencia de los clientes para Windows o para Android, el mando no dispone de almacenamiento ni visor de información, por lo que el programa cliente tuvo que ser incluido en el programa servidor del Linux, dentro de la función del código principal correspondiente al control del mando.

Se implementó una serie de comandos predefinidos para cada uno de los botones, eligiéndose de modo que el movimiento se pudiera hacer de forma intuitiva y que todo los comandos más importantes estuviesen contemplados para logar un buen control del robot con el mando.

Figura 3.43: Controles implementados en el mando.

3.9. Prueba del movimiento

Se realizaron dos pruebas principales de movimiento, llevando el robot a un lugar amplio donde se pudiesen probar los distintos modos de caminar y hacer los montajes adecuados. La primera prueba se realizó usando el cliente para Windows directamente conectado por puerto serie, puesto que el ordenador de a bordo no estaba finalmente desarrollado. Para cuando se realizó la segunda prueba, el servidor ya estaba funcionando, por lo que se pudo probar todo el montaje completo, utilizando el ordenador de a bordo, el cliente para Windows, el cliente para Android, y el mando a distancia.

3.9.1. Primera prueba

La primera prueba de campo se realizó en el taller de la planta baja del edificio 5E. Como medida de seguridad se usó un polipasto y unas cadenas a modo de arnés de seguridad para tener el robot anclado al techo durante las pruebas. Para realizar las pruebas se utilizaron unas baterías para automoción, de una capacidad de 68,3Ah, para suministrar la potencia necesaria a los motores, y una fuente de alimentación de uso en electrónica de 3000W para suministrar la potencia a la electrónica de control. Además fue necesario el uso de cables alargadores USB para conectar el micro principal del robot al ordenador, y para realizar las correcciones en el código del micro *in situ* utilizando el programador.

En el momento de realizar esta prueba, aún no estaba el programa servidor para el ordenador de a bordo totalmente implementado y tampoco estaban hechas las modificaciones en la placa de potencia para poder acoplar la alimentación directamente a las baterías, por lo que la prueba de movimiento se realizó utilizando la comunicación directa por puerto serie. Para este tipo de comunicación únicamente es necesario un cable de conexión USB lo suficientemente largo como para conectar el robot con el ordenador. El controlador tipo PID no estaba implementado para la velocidad, por lo que se usó únicamente el todo/nada para la posición.

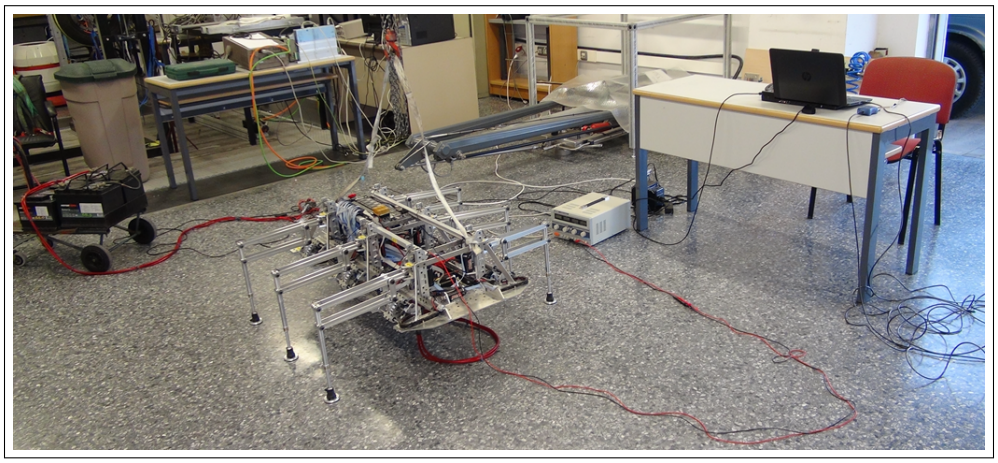


Figura 3.44: Lugar de la primera prueba del movimiento del robot.

3.9.2. Segunda prueba

La segunda prueba de campo se realizó en el hall de la tercera planta del edificio 5E. El control de velocidad de tipo PID ya estaba implementado, por lo que se decidió realizar las pruebas sin ningún tipo de arnés de seguridad para el movimiento. Estando terminada la configuración del ordenador de a bordo, se comprobó el funcionamiento de la comunicación por Internet para mandar órdenes de movimiento, tanto usando el ordenador como una tableta. Por último, también se utilizó la conexión Bluetooth para mover el robot utilizando el mando.

La prueba se realizó siendo el robot casi autónomo, necesitando únicamente la conexión con las baterías de 24V para suministrar potencia a los motores, prescindiendo esta vez de los cables USB, puesto que la comunicación se realizó a través del ordenador de a bordo, para conectar el ordenador con el robot y sin estar colgado del techo. Durante las pruebas se disponía de una grúa para poder colgar el robot en el aire y realizar algunas tareas de mantenimiento que fueron necesarias.

Para la alimentación de la electrónica de control se utilizó una batería de Litio-ion para portátiles, que tuvo que ser modificada colocando un conector compatible. La alimentación del ordenador de a bordo se realizó utilizando el convertidor DC-DC con salida USB que se adaptó para su uso con la etapa de potencia. Durante el desarrollo de las pruebas se agotó la carga de las baterías para automoción, con una capacidad de $68{,}3Ah$, que se estuvieron utilizando durante todo el tiempo que duró el proyecto, y se utilizaron otras baterías que estaban disponibles, con una capacidad de $44Ah$.

Figura 3.45: Lugar de la segunda prueba del movimiento del robot.

Capítulo 4

Resultados

- Control híbrido para el movimiento de las patas.
- Robot *Hexapod II* en condiciones de funcionamiento y un modelo virtual fidedigno al robot real.
- Protocolo de comunicación bidireccional.
- Tablas de consumos.
- Especificaciones técnicas finales del robot.

4.1. Control híbrido para el movimiento de las patas

El control híbrido implementado en las patas que consta de un controlador de velocidad tipo PID y un control todo/nada para la posición tiene un rendimiento excepcional, y gracias a él se han podido realizar movimientos complejos de avance y un movimiento vertical del robot bastante coordinado.

Eje	Kp	T_d **(s)**	T_i **(s)**
X	0,3	0	0,01
Y	0,5	0	0,01
Z	0,5	0	0,01

Cuadro 4.1: Parámetros de los controladores PID para la velocidad definitivos de cada eje.

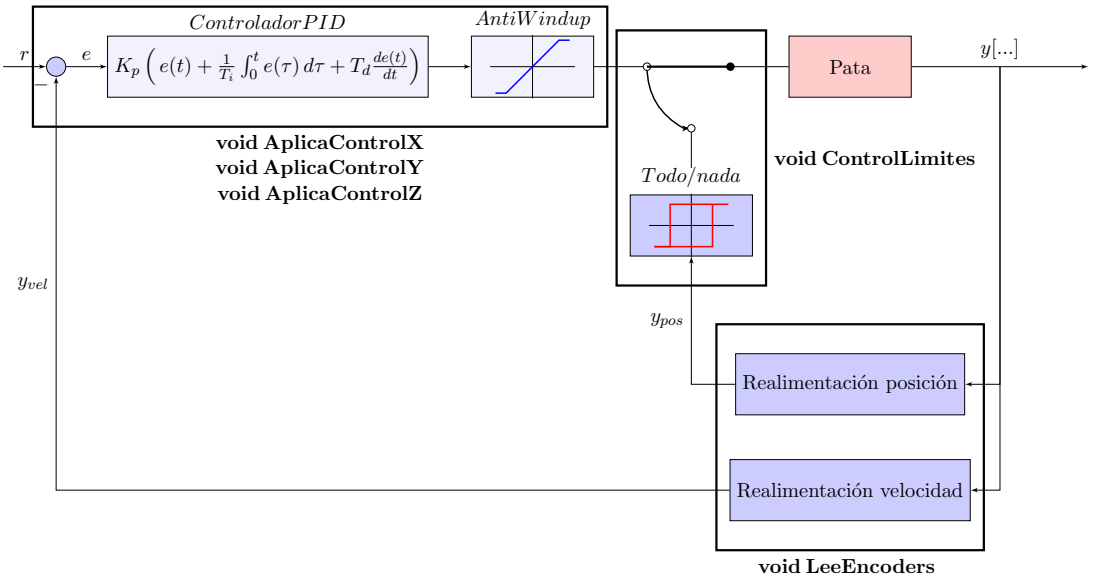
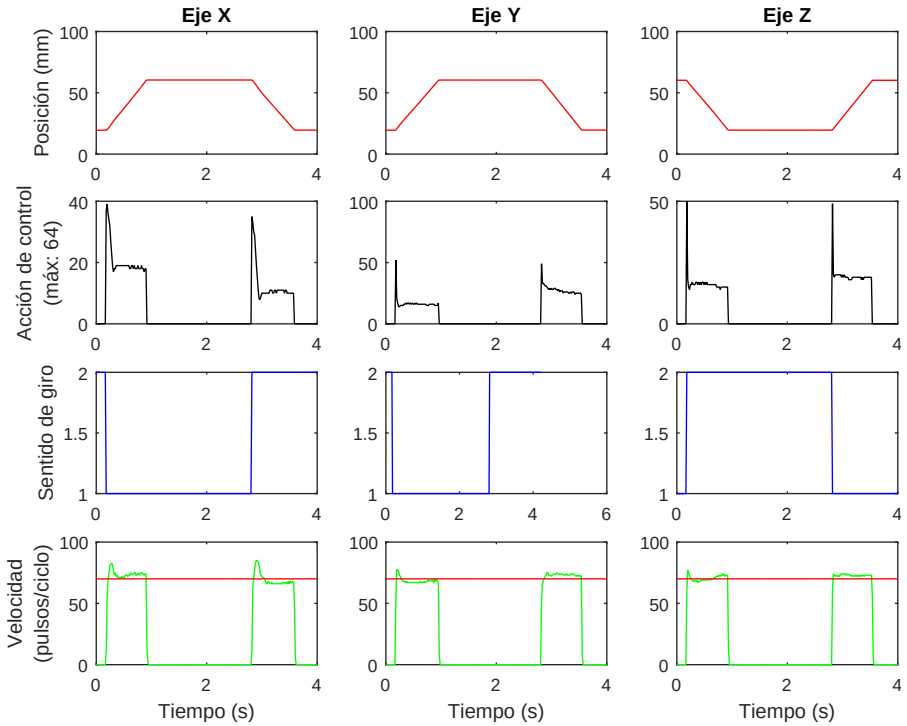


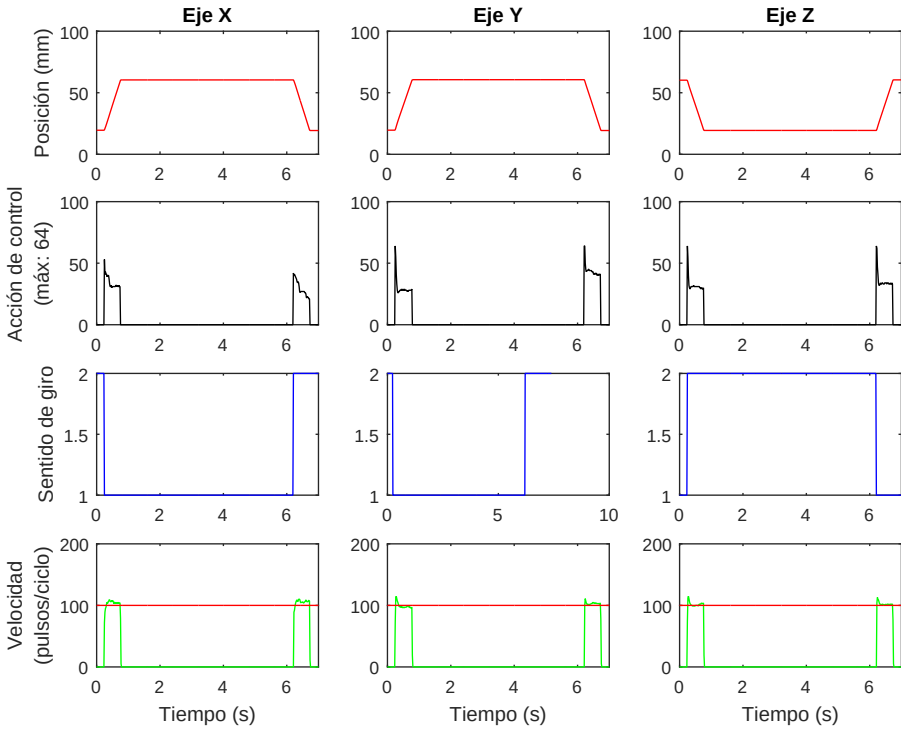
Figura 4.1: Esquema del control implementado para las patas

El sistema funciona bajo el control del regulador PID para la velocidad, implementado en las funciones *AplicaControlX*, *AplicaControlY*, y *AplicaControlZ*, que recibe la realimentación de los encoders cuya lectura se hace a través de la función *LeeEncoders*. Al mismo tiempo se ejecuta dentro de un manejador de interrupción la función *ControlLimites* que vela porque se alcance la posición establecida. Una vez que se detecta que la deslizadera ha alcanzado la posición, toma el control del proceso el regulador todo/nada implementado para la posición, que corta la alimentación de los motores terminando el movimiento.

Al realizar un movimiento de la misma longitud para los tres ejes de la pata a la vez, se puede observar que la referencia de velocidad se sigue con bastante precisión, logrando que las tres deslizaderas lleguen a su destino prácticamente de manera simultánea. Empeora ligeramente la respuesta respecto a si se hace el movimiento de cada eje por separado si se compara con las gráficas presentadas en el capítulo dedicado al control en el anexo, pero mejora significativamente con respecto a cualquier otro tipo de control o controlador utilizado hasta el momento.



(a) Referencia de velocidad de 70 pulsos/ciclo.

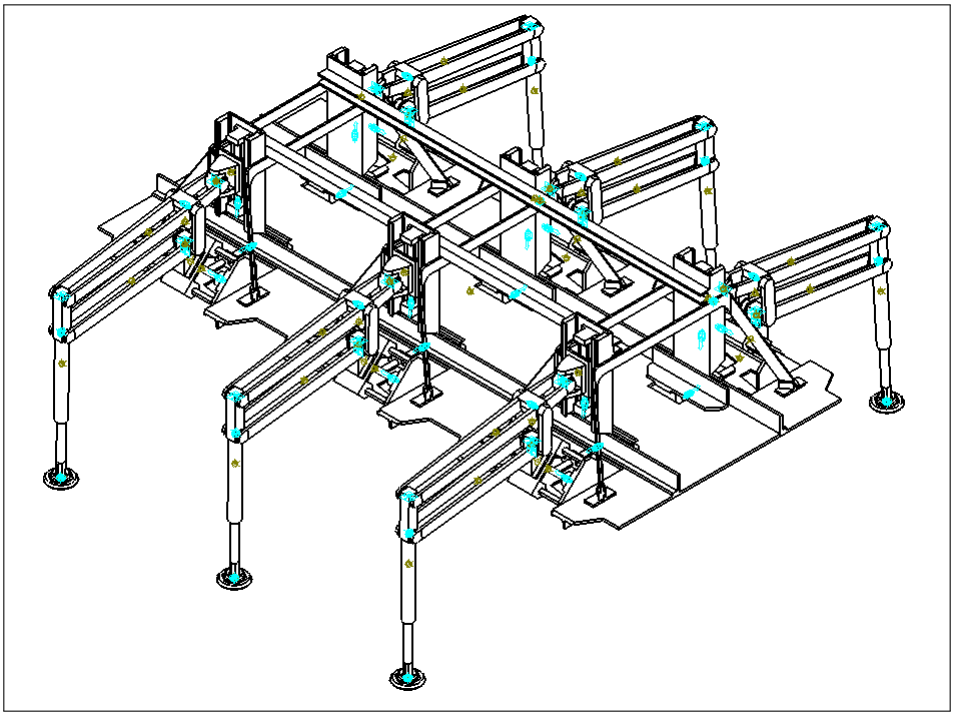


(b) Referencia de velocidad de 100 pulsos/ciclo.

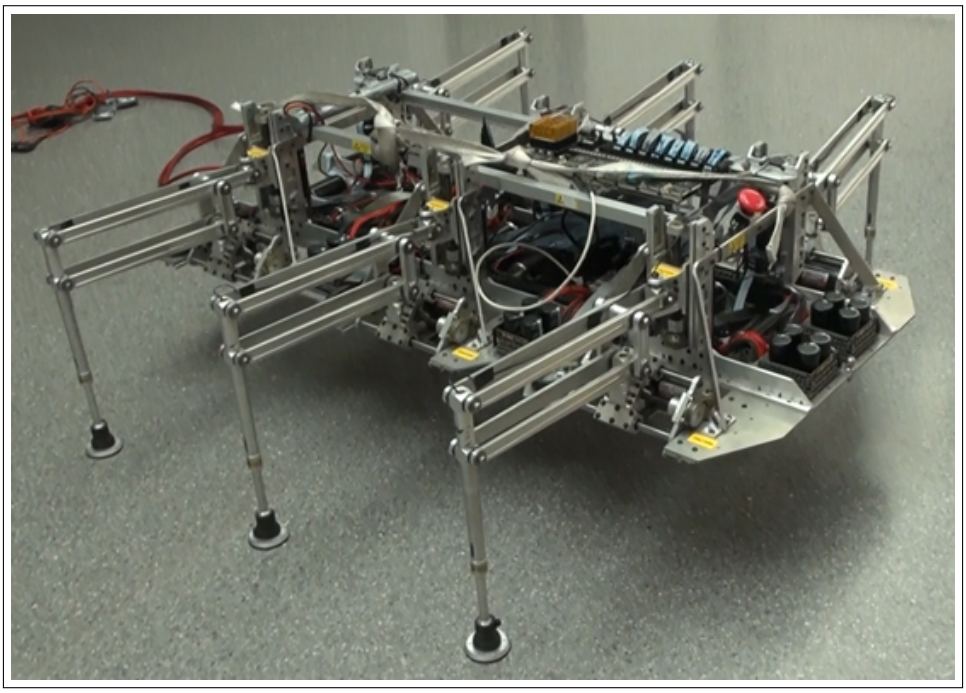
Figura 4.2: Respuesta del sistema con el controlador PI de velocidad para el movimiento de los tres ejes de una misma pata simultáneamente.

4.2. Movimientos simulados y reales

El mayor logro de la realización de este proyecto es haber conseguido realizar movimientos complejos con el robot, previamente simulando el comportamiento en el modelo virtual. A continuación se presentan instantáneas del robot en las distintas formas de movimiento junto a instantáneas de su correspondiente simulación.

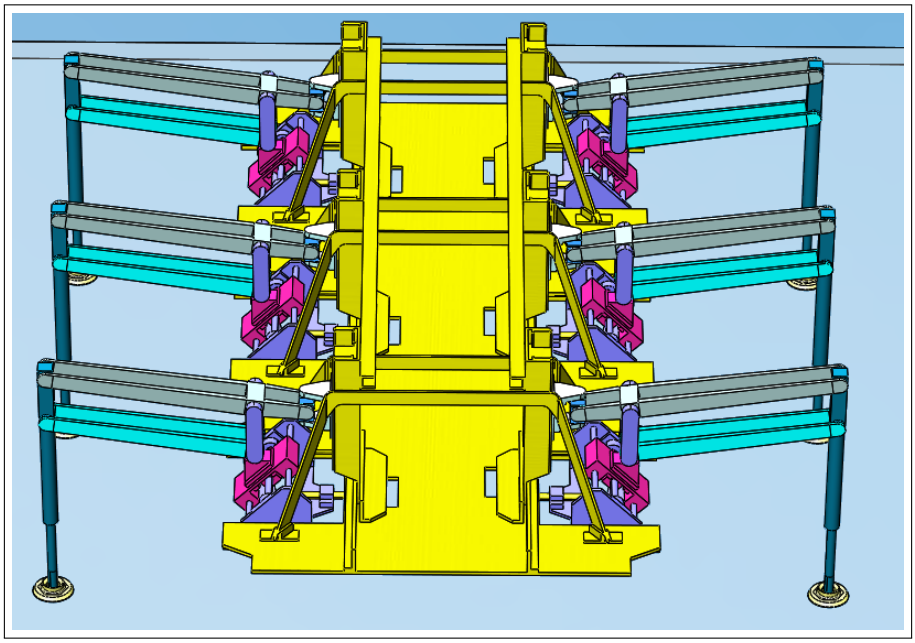


(a) Representación del modelo virtual.

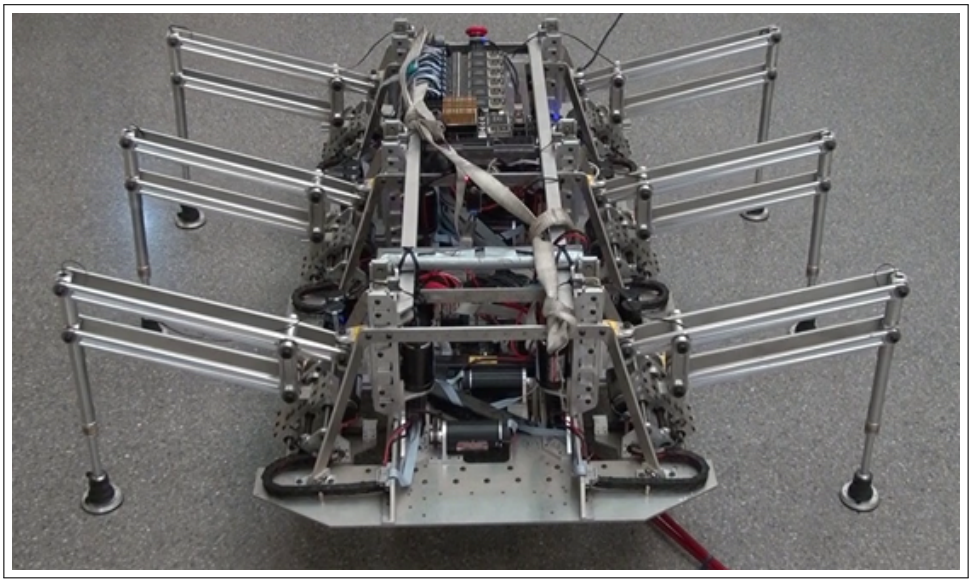


(b) Robot *Hexapod II* real.

Figura 4.3: Modelos real y simulado en posición de espera.

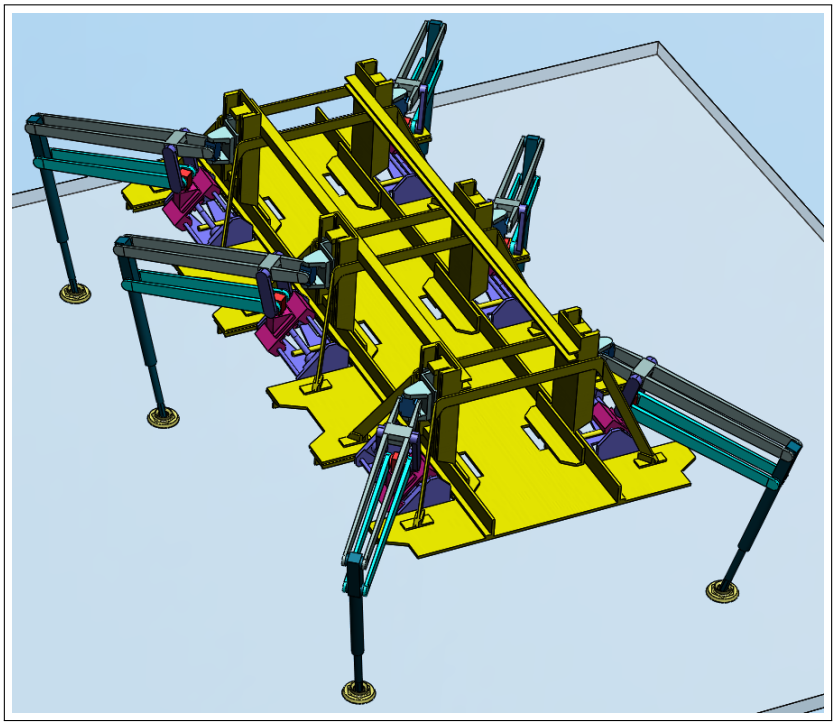


(a) Representación del modelo virtual.

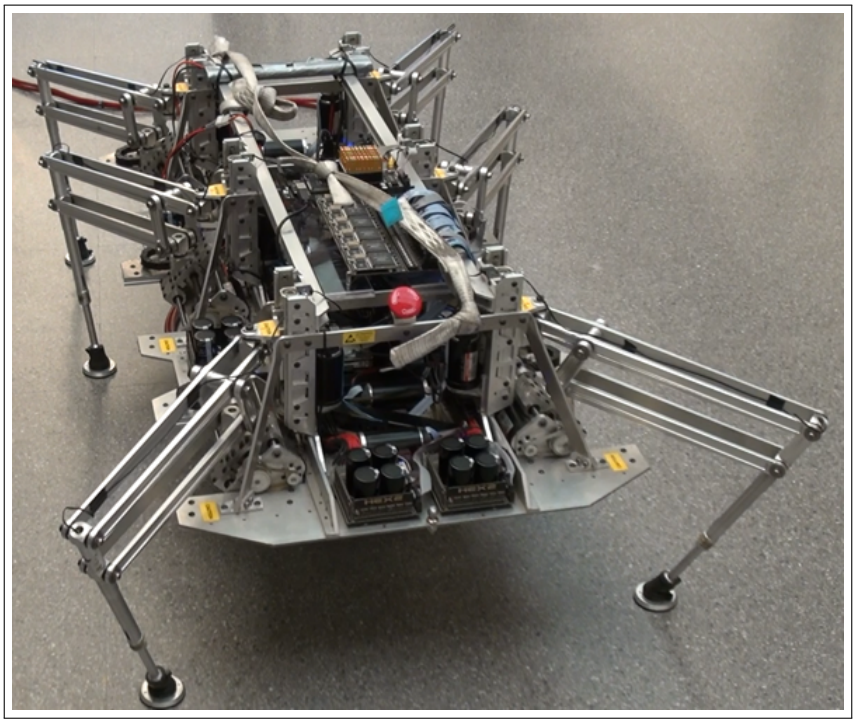


(b) Robot *Hexapod II* real.

Figura 4.4: Modelos real y simulado durante el proceso de levantar robot.

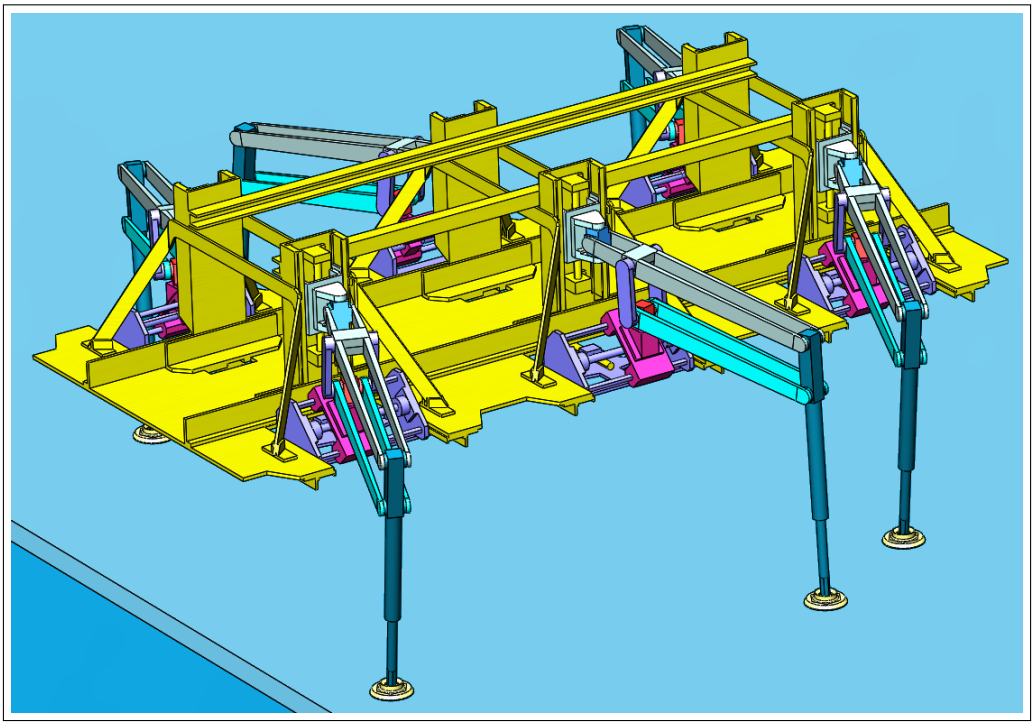


(a) Representación del modelo virtual.

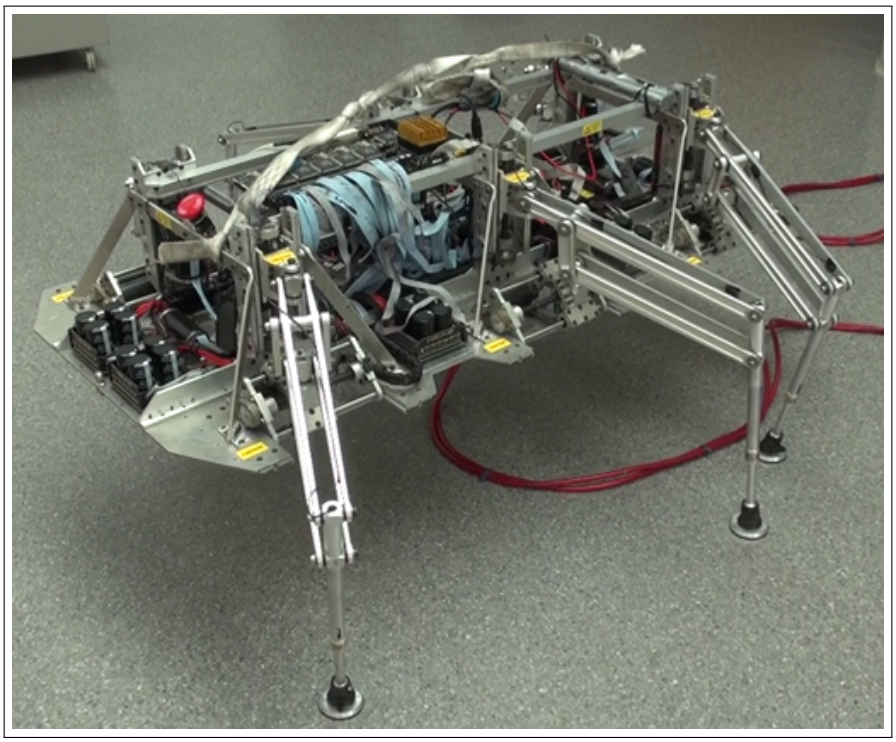


(b) Robot *Hexapod II* real.

Figura 4.5: Modelos real y simulado durante el proceso de plegado.

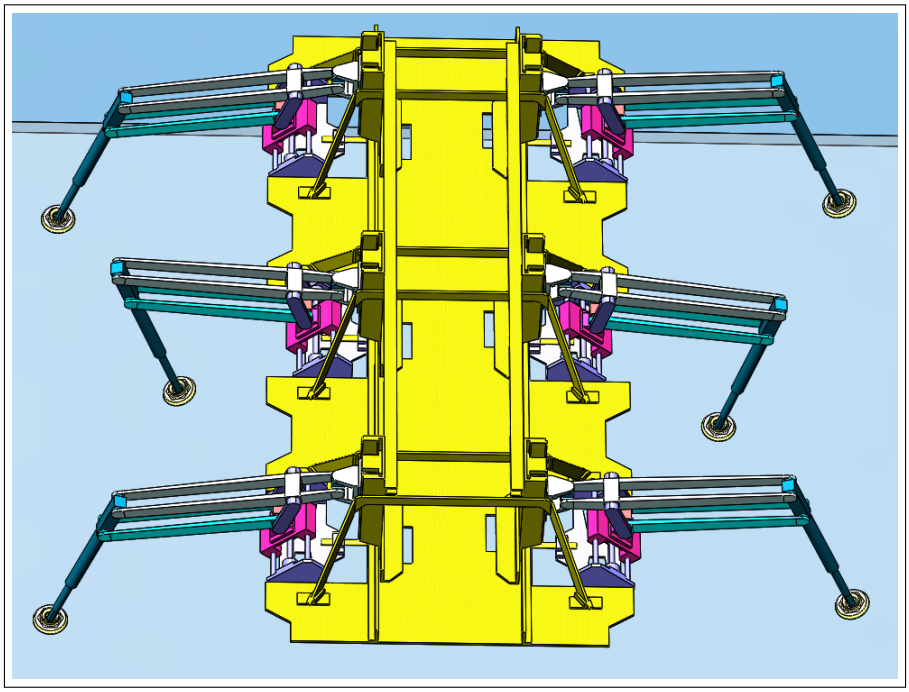


(a) Representación del modelo virtual.

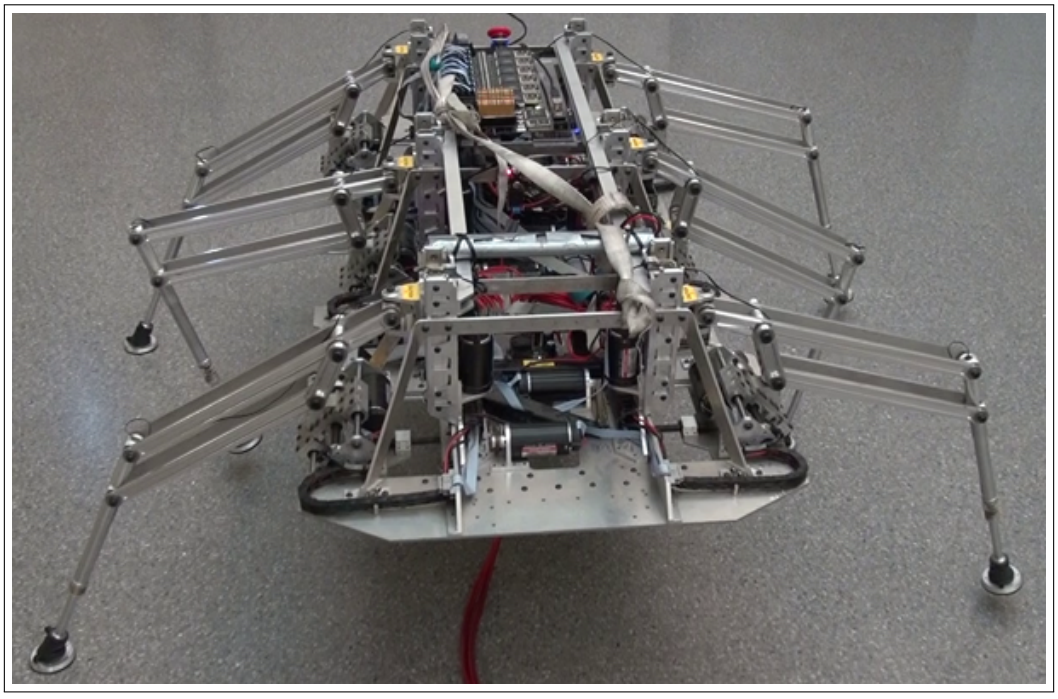


(b) Robot *Hexapod II* real.

Figura 4.6: Modelos real y simulado durante el modo de movimiento frontal.

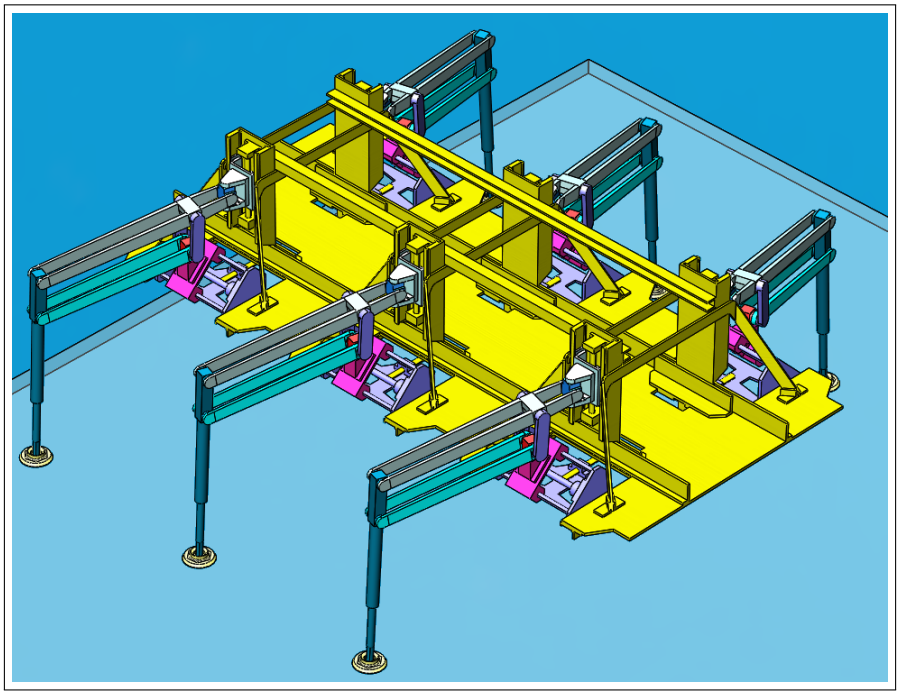


(a) Representación del modelo virtual.

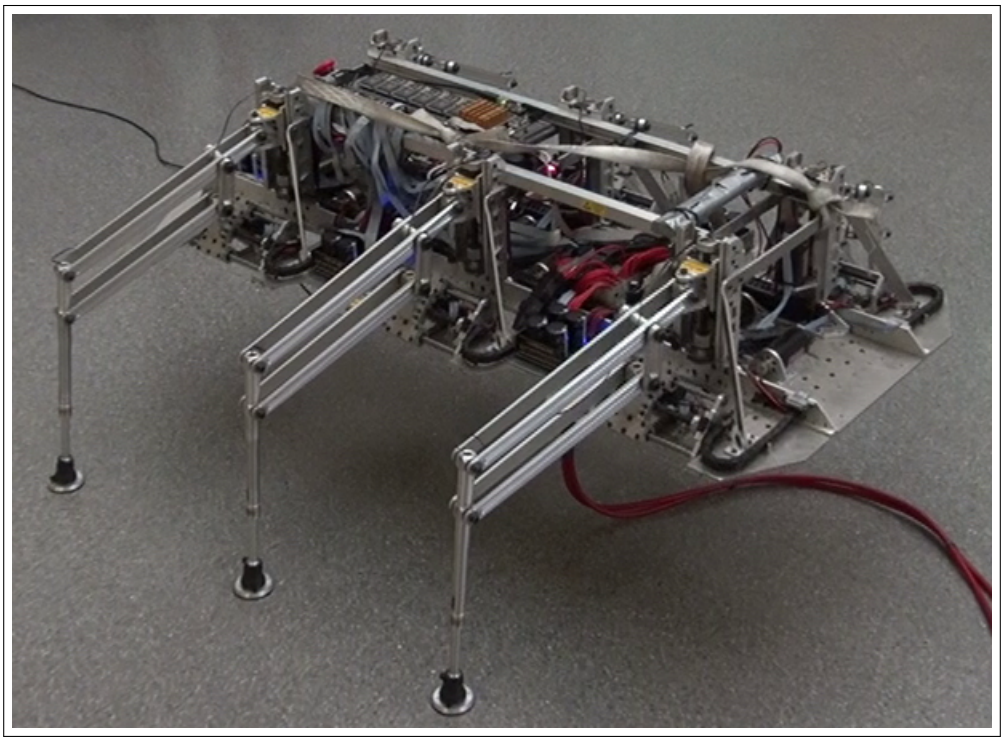


(b) Robot *Hexapod II* real.

Figura 4.7: Modelos real y simulado durante el modo de movimiento lateral.



(a) Representación del modelo virtual.



(b) Robot *Hexapod II* real.

Figura 4.8: Modelos real y simulado durante el modo de movimiento onda.

4.3. Limitaciones de movimiento

Durante la realización de la primera prueba de movimiento se pusieron en evidencia las inestabilidades dinámicas que se producen cuando se realiza el movimiento frontal, modificando ligeramente la configuración de las patas para aumentar la estabilidad del robot. Las cuatro patas de los extremos se acercaron al chasis, modificando la posición de la deslizadera del eje Z, mientras que las dos del centro se mantuvieron en su posición, más extendidas que las anteriores.

En cuanto a los modos de movimiento con avance frontal y lateral, cada uno tiene sus ventajas e inconvenientes. El único modo de movimiento con avance lateral, *trípode alternante lateral*, presenta un gran equilibrio dinámico y puede hacerse a alta velocidad. Entre el modo *trípode alternante frontal* y *onda*, el primero tiene mayor velocidad de movimiento a expensas de tener menor equilibrio y cierto balanceo, mientras que el segundo es más estable pero más lento.

Existen una serie de incompatibilidades en el plegado y los movimientos de las patas. Las incompatibilidades están relacionadas con la altura del robot. Si la deslizadera del eje Z se posiciona al final de su recorrido, estando lo más alejada del chasis posible, no puede estar el robot en su posición más erguida, con las deslizaderas del eje Y en su posición más elevada, puesto que se produce una interferencia entre las barras longitudinales de las patas. Para que no suceda esta situación cuando se realiza el movimiento de avance lateral se ha limitado la altura a la que puede realizarse éste. Vuelve a ocurrir una situación de interferencia mecánica si se intenta realizar el plegado con el robot en su posición más alta. Si la deslizadera del eje Z se retrae al máximo y se lleva la deslizadera del eje X a su origen estando el robot a su altura máxima, cuando desciende la deslizadera del eje Y se produce un contacto entre el chasis y la parte inferior de la pata.

La utilización de esperas para el movimiento con la comprobación de la posición de las patas se ha demostrado una importantísima medida de seguridad, puesto que si hay algún problema en la realización del movimiento y no alcanza alguna pata la posición a la que ha sido enviada, el proceso queda bloqueado y no se produce la siguiente secuencia. Además de ser necesario para la correcta realización del movimiento, esto es en cierta forma una parada de emergencia limitada, parando el movimiento hasta que se arregle la situación, se revise el funcionamiento, y se vuelva a poner a cero el robot, reiniciando el movimiento.

Analizando el movimiento del robot sobre el suelo, se han realizado una serie de observaciones. Cuando se realiza el movimiento de las patas que están en contacto con el suelo, se produce un deslizamiento entre los pies del robot y el suelo. Para futuros desarrollos sería interesante modificar los "zapatos" del robot, para que aumentando el coeficiente de rozamiento con el suelo se obtenga un mejor agarre y que no se produzca el deslizamiento. También se ha observado que al tratarse de un mecanismo bastante robusto al utilizar husillos para los desplazamientos, las

inercias de las patas en movimiento del robot son absorbidas por las holguras de los rodamientos de unión. Los movimientos bruscos de parada y arranque producen el balanceo del robot, por lo que sería necesario afinar el control para hacer estos movimientos más graduales.

Distintos fallos producidos durante la realización de la programación y el control permitieron observar qué sucede cuando las deslizaderas intentan salirse de sus posiciones límite. Se produjeron roturas de las correas de transmisión de potencia y fue necesario reemplazarlas en más de una ocasión. Sin embargo, la integridad del mecanismo permaneció inalterada, evidenciando el buen diseño de éste y el sistema de seguridad que supone la utilización de las correas como *fusible mecánico* en el camino de la transmisión de la potencia desde los motores hasta las patas.

4.4. Protocolo de comunicación

Se ha implementado un protocolo de comunicación basado en paquetes de información, denominados *mensajes*, totalmente bidireccional y con un funcionamiento excepcional. El protocolo de comunicación se basa en unos axiomas básicos en cuanto a la estructura, los tipos de mensajes disponibles, y la forma de realizar la comunicación se refiere. Teniendo esto en cuenta, si se conoce el protocolo se puede hacer una conexión con el servidor implementado en el robot desde cualquier dispositivo que disponga de una conexión a Internet.

A continuación se muestran unas capturas de pantalla realizadas durante el proceso de comunicación con el robot, donde se puede observar el trasiego de datos entre los distintos clientes y el servidor del ordenador de a bordo.

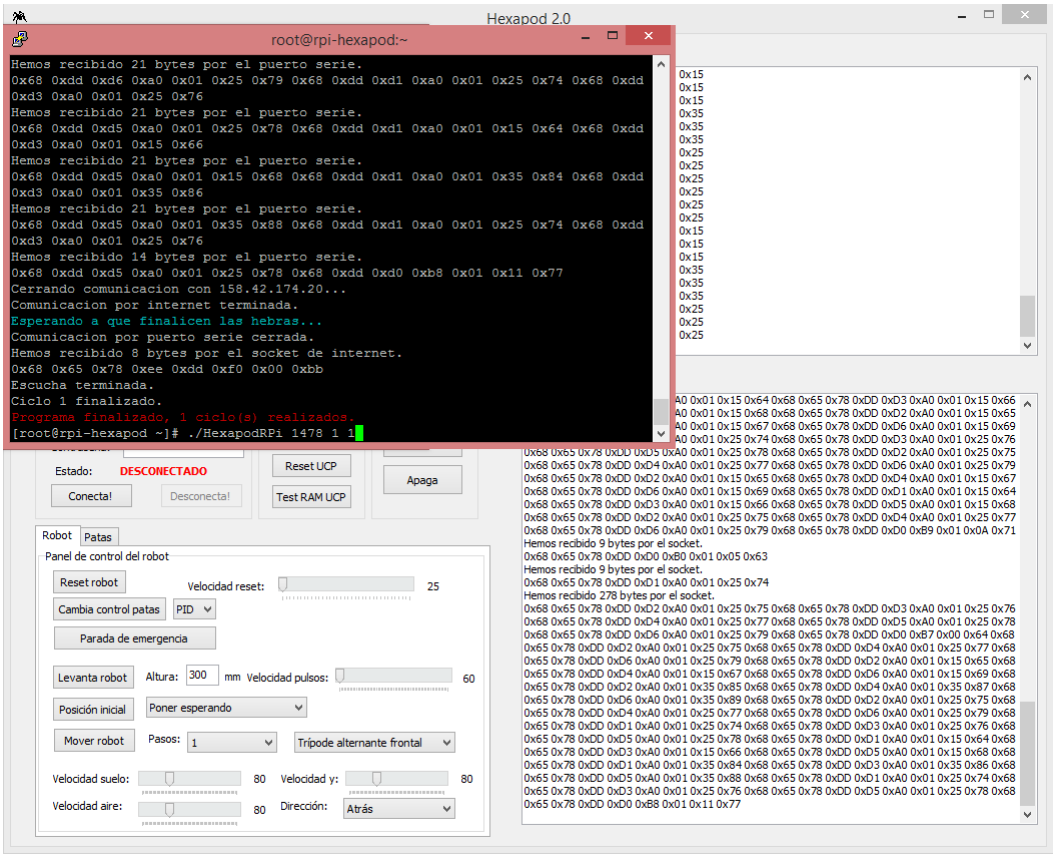


Figura 4.9: Captura de pantalla del control utilizando el cliente para Windows.

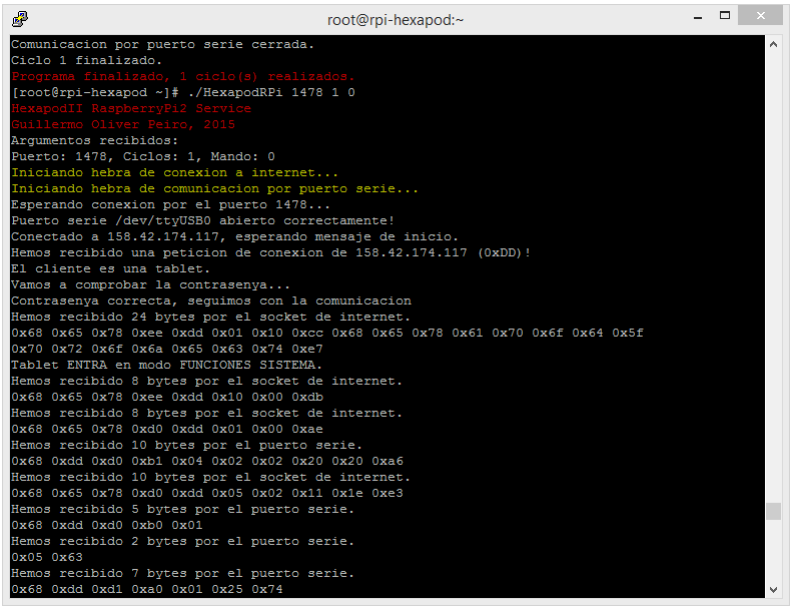


Figura 4.10: Captura de pantalla del control utilizando el cliente para Android.

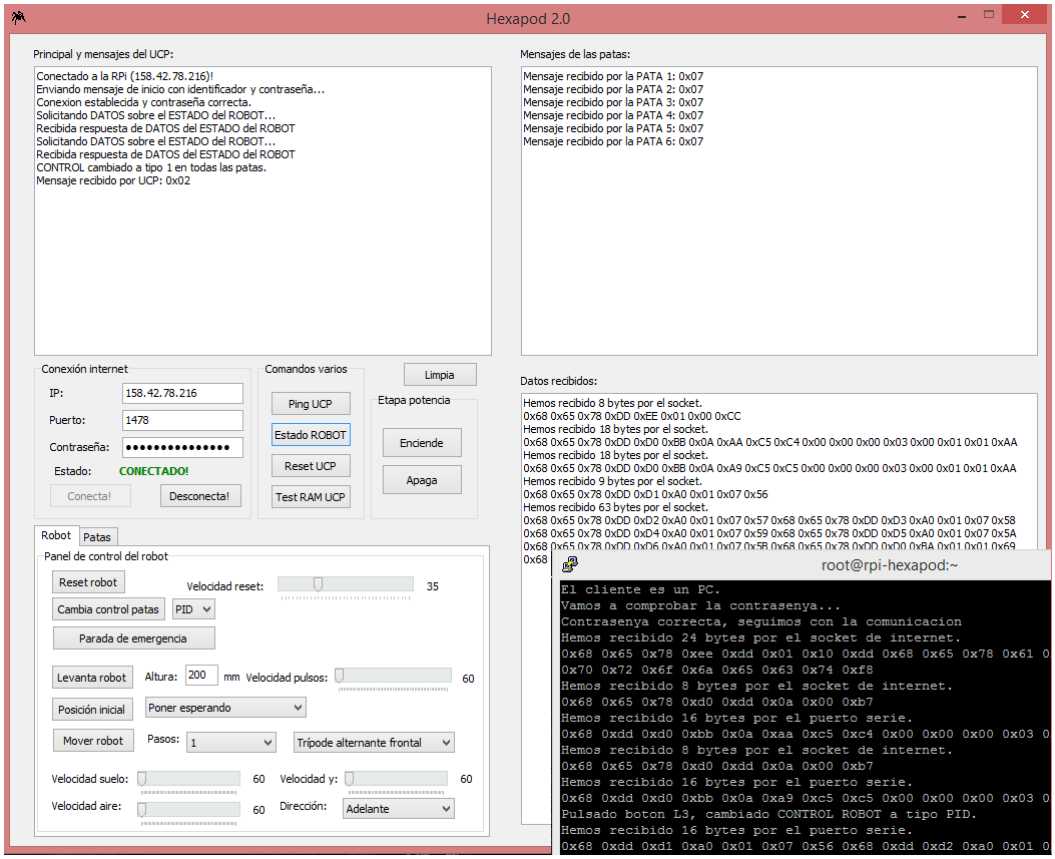


Figura 4.11: Captura de pantalla del control utilizando el mando y supervisando desde el cliente de Windows.

4.5. Consumo de potencia

A falta de un estudio más exhaustivo, se realizaron mediciones del consumo de robot con los medios disponibles. Utilizando una pinza amperimétrica digital se realizaron mediciones de consumo de amperios para la etapa de potencia, y alimentando el ordenador de a bordo y la electrónica con una fuente de alimentación se anotaron sus valores de consumo de potencia. Los valores de cada uno de los adaptadores del ordenador de a bordo se obtuvieron de la información que pone a disposición el sistema operativo Linux acerca de los periféricos conectados.

Los valores que se presentan en el caso de la electrónica de control y de algunas partes estáticas de la electrónica de potencia son de consumo continuo, mientras que en la electrónica de potencia se corresponden con picos de consumo durante la realización de los movimientos. También cabe decir que en el caso del movimiento son corrientes obtenidas una vez están los motores en movimiento, puesto que las demandas de corriente instantáneas durante el arranque de estos serán significativamente mayores.

Descripción	**Tensión de suministro (V)**	**Consumo (mA)**	**Potencia (W)**
Electrónica de control del robot			
Placa principal	14,4	890	12,816
Ordenador de a bordo			
Raspberry Pi 2	5	800	4
Conexión al robot	5	44	0,22
Adaptador WiFi USB D-Link DWA-131	5	500	2,5
Adaptador Bluetooth USB 4.0	5	200	1

Cuadro 4.2: Consumo de la electrónica de control y el ordenador de a bordo.

Descripción	Tipo de control	Tensión de suministro (V)	Consumo (A)	Potencia (W)
Potencia desactivada	-	24	0,2	4,8
Potencia activada	-	24	0,6	14,4
Reset robot	-	24	2.7	64,8
Consumo en vacío				
Levantar robot	Todo/nada (30)	24	12	288
Levantar robot	PID (60)	24	18	432
Movimiento general	Todo/nada (30)	24	6,8	163,2
Movimiento general	PID (60)	24	9	216
Consumo en el suelo				
Levantar robot	Todo/nada (30)	24	17	408
Levantar robot	PID (60)	24	38	912
Movimiento general	Todo/nada (30)	24	8,5	204
Movimiento general	PID (60)	24	12	288

Cuadro 4.3: Consumo de la electrónica de potencia

4.6. Especificaciones técnicas

Se presentan a continuación las especificaciones técnicas finales del robot obtenido.

Características mecánicas	
Peso (kg)	43.7
Longitud (cm)	117
Anchura (cm)	58
Altura (cm)	23.6-55.6
Características electrónicas	
Ordenador de a bordo	Raspberry Pi 2
Conexión	USB, WiFi, o mando Bluetooth
Alimentación electrónica	Batería Li-ion 14.4V 5Ah
Alimentación motores	2 baterías de 12V en serie, 24V, 68.3Ah

Cuadro 4.4: Características técnicas finales del robot.

Capítulo 5

Futuro

Debido a las dimensiones del proyecto, quedaron algunas mejoras pendientes, algunas de las cuales se conocían desde antes de iniciar el proyecto y otras surgieron durante la realización del mismo.

- Rediseño de la electrónica de potencia.
- Estudio cinemático exhaustivo del modelo, desarrollo de la cinemática inversa, y planificación de trayectorias.
- Acoplamiento de una unidad de medición inercial.
- Protocolo de actualización del programa de los micros.
- Estudio energético para reemplazar baterías de plomo ácido.
- Sistema de desplazamiento de masas para mejorar el equilibrio del robot en movimiento.
- Documentación del código fuente desarrollado.

5.1. Electrónica de potencia

La electrónica de potencia tiene una serie de carencias que fueron arregladas *ex profeso* para poder realizar el movimiento. El arreglo puede considerarse como temporal, y para su versión final debería ser rediseñada. En primer lugar, algunas de las vías no tienen las dimensiones adecuadas para manejar con seguridad toda la potencia que se necesitaría para efectuar el movimiento del robot a máxima potencia en todo tipo de situaciones. Habría que diseñar de nuevo la placa de circuito impreso para corregir este asunto. También sería buena idea cambiar los interruptores que se utilizan, sustituyendo los relés actuales que tienen una intensidad nominal de 20A cada uno por transistores MOSFET de potencia, capaces de soportar intensidades mucho más elevadas sin problema. La modificación de estas características supondría tener que diseñar de nuevo la etapa de potencia, volviendo a realizar todos los cálculos y las pruebas pertinentes.

5.2. Estudio cinemático

El control mixto implementado actualmente funciona de manera satisfactoria, permitiendo mover los motores a una posición determinada a la velocidad que se indique. Por la imposibilidad de abarcar todo esto en el plazo de tiempo disponible, se deja el desarrollo de un control avanzado para el futuro, un control basado en la aplicación de la cinemática inversa. Una vez se tenga hecho el estudio cinemático del robot en profundidad, el objetivo sería definir trayectorias para seguirlas paso a paso y ajustar un controlador para este tipo de movimiento, con el fin de poder realizar movimientos de manera suave y exacta. Se espera que una vez realizado este estudio y obtenidas las ecuaciones correspondientes al movimiento del robot por la cinemática inversa, pueda realizarse un control que sea capaz de seguir las referencias de posición, de velocidad, e incluso de aceleración, para conseguir un movimiento suave y perfectamente ajustado.

5.3. Unidad de medición inercial

Para mejorar el control del robot podría incluirse una unidad de medición inercial, de modo que se tuviesen datos en tiempo real de la inclinación y la aceleración del robot en todo momento. De esta forma podría realizarse un control más preciso del movimiento, con la posibilidad de corregir situaciones de equilibrio inestable e implementar un modo de giro sabiendo exactamente cómo responde el robot. De esta manera se podría controlar cuántos grados ha girado el robot, a qué velocidad, etc.

Es importante saber la inclinación del robot para el control del mismo ya que si el robot esta subiendo una rampa o bajando una pendiente el comportamiento de las patas y del cuerpo es distinto. Con una unidad de medición inercial se puede saber la inclinación en los tres ejes de coordenadas, de modo que podría detectarse si se inclina del morro o de un lado y controlar el giro. También permitiría realizar un

control avanzado, como por ejemplo hacer caminar al robot perpendicular respecto a la superficie por donde se está desplazando aunque ésta sea inclinada. Si el robot va a perder el equilibrio porque la superficie en la que está situado es móvil, podrían utilizarse estos datos para recuperarlo, permitiendo que el robot además de equilibrio estático posea equilibrio dinámico.

5.4. Actualización de firmware

Los micros han de ser actualizados usando el programador AVRISP mkII manualmente, usando la conexión integrada en la placa del micro, pero cabe la posibilidad de actualizar los micros desde el ordenador. Se tendría que desarrollar un protocolo basado en la comunicación actual para que los micros recibieran paquetes de actualización y se reprogramaran usando un *bootloader*.

Los micros de *Atmel*, como muchos otros, tienen la posibilidad de incluir un *bootloader* en su programación. Un *bootloader* es un programa que se ejecutará nada más iniciarse el micro, antes de iniciarse el programa principal. Desde el código del bootloader se podrían recibir los datos de actualización y actualizar el firmware, o programa principal, de los micros en tiempo de ejecucción. Sería necesario desarrollar un sistema de actualización robusto, que sea capaz de manejar las grandes cantidades de datos que supone actualizar todo el programa del micro, tanto para el micro principal como el de las patas. Se hicieron avances al respecto al inicio del proyecto, pero se paró la implementación dando preferencia al desarrollo del movimiento del robot, dejando este tipo de ajustes finos para más adelante.

5.5. Estudio energético

En la actualidad el robot está alimentado con dos baterías de 12V de plomo ácido para uso en automoción. Cuando se empezó este proyecto por primera vez, hace diez años, todo el mercado de las baterías no estaba tan desarrollado como en la actualidad. Las baterías de plomo ácido tienen la ventaja de que su capacidad es muy elevada, pero a consta de su elevado peso. La sustitución de estas baterías por otras basadas en el litio, bien sea de Litio-ión o de Litio-polímero, supondría un aumento extraordinario en la movilidad del robot. La utilización de baterías con mayor densidad energética permitiría reducir el tamaño y el peso de estas, logrando que el robot sea más autónomo, pudiendo llevarlas colocadas en su chasis.

5.6. Equilibrio en movimiento

Cuando se desplaza frontalmente, las inercias de las masas del robot provocan un cierto desequilibrio, que se traduce en un balanceo del robot hacia delante y hacia detrás. Una solución sería dotarlo en los raíles que posee en su parte superior con unas masas en movimiento, bien podrían ser las propias baterías, de modo que mediante un sistema de poleas y correas podría desplazarse el peso en una o varias

direcciones cuando se produzca el movimiento para equilibrar el robot y reducir su balanceo, y de esta manera lograr la estabilidad dinámica del robot.

5.7. Documentación del código fuente

El proceso de documentación exhaustiva del código fuente está en desarrollo mediante la utilización de la herramienta *Doxygen*[1]. La documentación del código permitirá un mejor mantenimiento y que el código sea de más fácil comprensión si en algún momento una tercera persona distinta del autor tenga que continuar con su desarrollo.

[1] **Doxygen**, herramienta para la generación de documentación de código fuente. `http://www.stack.nl/~dimitri/doxygen/`

Capítulo 6

Conclusiones

Tras la finalización de un proyecto de estas dimensiones, ha de dedicarse un momento para la reflexión acerca de las conclusiones que se pueden extraer de lo realizado. Es por tanto pertinente hacer balance acerca del esfuerzo aportado, los nuevos conocimientos obtenidos, y las competencias adquiridas.

En primer lugar, se han podido ampliar los conocimientos de programación que ya se poseían a nuevos lenguajes y plataformas. Este proyecto ha ofrecido la posibilidad de trabajar con distintos tipos de dispositivos programables, cada uno con sus peculiaridades. Se han adquirido nuevos conocimientos acerca de la programación de microcontroladores, mediante el uso de manejadores de interrupciones y comunicaciones por puertos serie, muy útiles para el futuro desarrollo de aplicaciones para sistemas embebidos de uso en el ámbito industrial. Por otro lado, se ha hecho una introducción a la programación y a la configuración de características avanzadas bajo el sistema operativo Linux, un mundo totalmente desconocido antes de empezar este proyecto.

En segundo lugar, se ha hecho evidente la importancia que tiene el desarrollo de un control lo suficientemente avanzado y debidamente meditado cuando se trata de manejar un robot de estas características. Más allá de lo estudiado en la teoría, cuando uno se enfrenta a un desafío como este ha de aplicar los conocimientos adquiridos de las clases teóricas a una situación concreta, y la mayoría de veces resulta más laborioso de lo que se pensaba en un primer momento. A pesar de que queda bastante trabajo por hacer en este aspecto, se han adquirido competencias en cuanto al manejo de sistemas complejos, su identificación y el sintonizado de controladores para ellos, que serán sin duda de gran importancia en los proyectos que se desarrollen en un futuro cada vez más automatizado.

En cuanto a la dimensión mecánica del proyecto, se logró conocer mejor el funcionamiento del robot y las limitaciones del diseño de éste. A pesar de las bondades que posea un mecanismo, siempre hay una serie de limitaciones acerca de su funcionamiento que a primera vista pueden no resultar evidentes. Se pudo comprobar qué sucede cuando se fuerza un mecanismo a sus posiciones límites, aprendiendo de

esta forma la importancia de realizar un buen diseño del mecanismo y en la medida de lo posible intentar que cuando esté sometido a un esfuerzo grande éste pueda ser evacuado a través de un elemento de fácil sustitución.

Por otra parte, se aprendió a realizar simulaciones de movimientos utilizando programas de ingeniería asistida por ordenador, pudiendo conocer sus aplicaciones y limitaciones. En un robot complejo como el que se trata, resulta importante realizar una simulación previa para poder comprobar el comportamiento antes de aplicar las secuencias de movimiento al modelo real. Durante la simulación de un mecanismo se pueden apreciar algunos detalles acerca de la respuesta obtenida que a priori podrían no estar contemplados y que han de ser corregidos antes de convertir las secuencias en realidad. Se consiguieron nuevos conocimientos de manejo de programas de simulación de uso en el mundo industrial, en este caso *SolidWorks*®, que sin duda serán de utilidad si se quiere aplicar el mismo proceso a cualquier otra máquina o mecanismo.

En último lugar, cabe destacar las habilidades adquiridas a lo que la electrónica se refiere. Se realizó una introducción a la electrónica de control y de potencia, pudiendo identificar funcionamientos erróneos y realizar las correcciones oportunas para volver a poner en marcha los circuitos afectados. Mucho se pudo aprender acerca de cómo se realiza un proceso de diseño de una electrónica a medida como esta, descubriendo errores comunes y los escollos del diseño, intentando hacer todo lo posible por corregirlos. Además, fue necesario llevar a cabo soldaduras de conectores y componentes discretos de manera precisa, aumentando así la experiencia y mejorando las capacidades con las que se contaba antes de la realización del trabajo. Las habilidades técnicas y de solución de problemas obtenidas serán de aplicación si se quiere continuar el proyecto actual o desarrollar módulos electrónicos para otros proyectos.

El resultado de la ejecución de este proyecto se puede considerar como un éxito, puesto que se consiguieron alcanzar los objetivos marcados de manera satisfactoria. Se ha conseguido volver a poner en marcha el robot y desarrollar un control telemático a tiempo real que ha resultado ser fiable y robusto. Sin embargo, la realización de este proyecto no buscaba ser un punto y final al viaje emprendido por aquellos dos jóvenes ingenieros cuando decidieron convertir en realidad el robot, sino un punto y seguido en la historia de este ambicioso proyecto, esperando que se escriban muchas más líneas acerca de *Hexapod II* y sus andanzas.

Bibliografía

Libros y artículos:

[1] Mitzner K. (2009). *Complete PCB Design Using OrCAD® Capture and PCB Editor*. Editorial Elsevier.

[2] Oliver G. & Uriel A. (2015). *Cliente Android para el robot hexápodo: HEXAPOD II*. UPV.

Contenido en línea:

[3] Atmel (2011). *Atmega 128 datasheet*. `http://www.atmel.com/devices/atmega128.aspx`.

[4] *Stack Overflow*. Consulta de dudas acerca de programación. `http://www.stackoverflow.com`.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Trabajo Fin de Grado

Simulación Virtual, Reconfiguración Electrónica y Programación Real del Movimiento del Robot *Hexapod II*

Presupuesto

Autor:
Guillermo Oliver Peiró

Tutor:
Dr. Ing. Ind. José Luís Oliver Herrero

Universidad Politécnica de Valencia
Valencia, ESPAÑA

Curso 2014-2015

Parte II

Presupuesto

Índice general

Índice de tablas

Capítulo 1

Introducción

En primer lugar, cabe decir que el presupuesto se redactará como si de un proyecto a medida se tratase, es decir, como si se estuviera realizando bajo encargo. Se parte de la base de que el cliente posee el robot en las mismas condiciones que al inicio del proyecto y precisa de alguien se encargue de realizar la puesta a punto tal y como se ha hecho durante el desarrollo del este proyecto. Se detallará el presupuesto correspondiente a volver a poner en funcionamiento el robot, implementando todo lo necesario para lograr el funcionamiento deseado, por lo que el presupuesto se corresponde con el proceso de reparación y adecuación del robot, sin entrar a valorar lo que ha costado el desarrollo y la construcción del mismo desde cero. Cualquier gasto de material tangible que sea necesario para la posterior utilización correrá a cargo del cliente.

Durante el desarrollo del proyecto se han realizado labores de ingeniería correspondientes a distintas ramas, por lo que podemos diferenciar tres grandes ámbitos involucrados en el desarrollo: la informática, la electrónica, y la mecánica. Teniendo en cuenta lo anterior, el presupuesto se ordenará por secciones, que se enumeran a continuación: mano de obra, informática, electrónica, software, y mecánica.

Por último, se presentará un resumen del presupuesto, donde se engloben todos las secciones anteriores y se calcule el coste total aplicando los correspondientes porcentajes de gastos generales, beneficio industrial, y de impuestos sobre el valor añadido.

Capítulo 2

Presupuesto

2.1. Mano de obra

En la mano de obra se tendrá en cuenta las horas invertidas por parte del ingeniero jefe, que se encargará de realizar las simulaciones, la programación, y el montaje de la informática. También se ha tenido en cuenta las horas de las distintas soldaduras de conexiones y componentes, que pudieran ser realizadas por un técnico capacitado, pero que han sido realizadas por el ingeniero.

Según la orden ministerial *ESS/86/2015, de 30 de enero, por la que se desarrollan las normas legales de cotización a la Seguridad Social, desempleo, protección por cese de actividad, Fondo de Garantía Salarial y formación profesional, contenidas en la Ley 36/2014, de 26 de diciembre, de Presupuestos Generales del Estado para el año 2015* publicada en el B.O.E. de 31 de enero de 2015, se establece un tipo de cotización para contingencias comunes del 28,3 %. Se parte de un salario base de 35000 euros al año y para elaborar el presupuesto se consideran 46 semanas de trabajo al año, que equivalen a sus correspondientes 230 jornadas laborales de 8 horas, y 10 horas extra al mes con precio adicional del 20 %.

Tomando como referencia lo dispuesto en el *XVII Convenio colectivo nacional de empresas de ingeniería y oficinas de estudios técnicos* publicado en el B.O.E. de 25 de octubre de 2013, se considerarán los siguientes valores: 3 euros al día en concepto de dieta de comida, y 2,28 euros al día de pluses de transporte (12 kilómetros de media al día con una tarifa de 0,19 euros por kilómetro recorrido).

Descripción	Coste anual (€)	Coste (€/hora trabajada)
Salario base (230 días)	35000,00	19,02
Seguridad social (28,3 %)	9905,00	5,38
Dietas	690,00	0,38
Pluses	524,40	0,29
Horas extra (10h/mes)	2625,00	1,43
Total	48744,40 €	26,49 €

Cuadro 2.1: Desglose del precio de la mano de obra del Ingeniero Industrial

A continuación se detallan las horas de trabajo dedicadas a cada una de las partes de las que se compone este proyecto.

Descripción	Cantidad (horas)	Precio unitario (€/h)	Subtotal (€)
Puesta a punto de la electrónica	50	26,49	1324,58
Desarrollo de los programas de los micros	100	26,49	2649,15
Desarrollo del programa cliente para el PC	80	26,49	2119,32
Control del movimiento de los motores	45	26,49	1192,12
Desarrollo del servidor Linux embebido	80	26,49	2119,32
Planificación de movimientos	50	26,49	1324,58
Desarrollo del programa cliente para Android	30	26,49	794,75
Desarrollo del programa cliente para el mando	5	26,49	132,46
Prueba del movimiento	12	26,49	317,90
Redacción de los documentos del proyecto	45	26,49	1192,12
Total	**497 horas**		13166,29 €

Cuadro 2.2: Presupuesto mano de obra

2.2. Informática

La informática se dividirá en la informática a medida, diseñada *ad hoc* para este caso; y la informática general, que abarca los equipos informáticos y de grabación de vídeo.

Los productos relacionados con la informática se compraron a través de la página web de *Amazon*, en *Mouser Electronics* o en tiendas de informática de venta al público general localizadas en la ciudad de Valencia.

Se tendrá en cuenta un período de amortización para los equipos informáticos de 3 años a un ritmo de 1840 horas al año (230 días laborables con jornadas de 8 horas), calculando así el coste de éstos por hora trabajada y utilizando este valor para calcular la parte correspondiente para el presupuesto.

$$\text{Costo de amortización} = \frac{\text{Precio del equipo (€)}}{\text{Período de amortización (h)}}$$

Descripción	Cantidad	Precio unitario (€)	Subtotal (€)
Raspberry Pi 2 1GB	1	39,99	39,99
Tarjeta de memoria microSD HC 16 GB	1	9,68	9,68
Adaptador WiFi USB D-Link DWA-131	1	13,99	13,99
Adaptador Bluetooth USB 4.0	1	7,99	7,99
Carcasa para Raspberry Pi	1	14,50	14,50
Alargador activo de cable USB	2	14,95	29,90
Batería de portátil Li-ion 14.4V 5Ah	1	24,70	24,70
Controlador inalámbrico de PlayStation3	1	35,00	35,00
Total informática a medida			175,75 €

Cuadro 2.3: Presupuesto de informática a medida

Descripción	Precio unitario (€)	Costo de amortización (€/h)	Cantidad (h)	Subtotal (€)
Ordenador portátil HP Zbook	849,99	0,15	400,00	61,59
Videocámara HD Sony HDR-700V	1298,00	0,24	12,00	2,82
Ordenador de sobremesa HP	749,99	0,14	85,00	11,55
Total informática general				75,96 €

Cuadro 2.4: Presupuesto de informática general

Total informática a medida	175,75 €
Total informática general	75,96 €
Total informática	251,71 **€**

Cuadro 2.5: Presupuesto total informática

2.3. Electrónica

Los productos relacionados con la informática se compraron a través de Internet en la página de *Mouser Electronics* o en tiendas de electrónica venta al público general localizadas en la ciudad de Valencia.

Hay que tener en cuenta que estos precios son por unidad, considerando la compra de cada elemento por separado, pero que la mayoría de tiendas de informática ofrecen descuentos en los precios si se realizan pedidos de un número mayor de unidades.

Descripción	**Cantidad**	**Precio unitario (€)**	**Subtotal (€)**
Programador AVRISP mkII	1	39,99	39,99
Convertidor DC 7V-24V a DC 5V 3A USB	1	6,99	6,99
Conectores banana 4mm	6	1,94	11,64
Conectores alimentación	5	1,24	6,20
Transistor MOSFET SI2308BDS	1	0.48	0,48
Transistor BJT 2N2222	1	1,74	1,74
Resistencia 47K	2	0,13	0,25
Resistencia 2K2	1	0,13	0,13
Relé universal 24V 20A	3	3,12	9,36
Diodo Schottky	1	0,47	0,47
Pinza amperimétrica 400 AC DC	1	32,81	32,81
Fuente de alimentación 30V 5A	1	199,99	199,99
Batería de plomo ácido 12V $44Ah$	2	60,95	121,90
Batería de plomo ácido 12V $68{,}3Ah$	2	83,95	167,90
Total componentes discretos			599,85 €

Cuadro 2.6: Presupuesto componentes discretos de electrónica

Descripción	**Cantidad (metros)**	**Precio lineal (€/m)**	**Subtotal (€)**
Cable de conexión 1mm - Color rojo	5	1,13	5,65
Cable de conexión 1mm - Color negro	5	1,05	5,25
Total cables de conexión			10,90 €

Cuadro 2.7: Presupuesto cables de conexión electrónica

Total componentes discretos	599,85 €
Total cables de conexión	10,90 €
Total electrónica	620,75 **€**

Cuadro 2.8: Presupuesto total electrónica

2.4. Software

En esta sección se contemplarán las licencias de uso de todos los programas utilizados para el desarrollo del proyecto. Cabe destacar la apuesta en la medida de los posible por programas que permiten su utilización libre de costes.

Se considerará un período de amortización de las licencias de los programas de 3 años a un ritmo de 1840 horas al año, calculando así el coste de amortización del software por hora trabajada y utilizando este valor para calcular la parte correspondiente para el presupuesto.

$$\text{Costo de amortización} = \frac{\text{Precio de la licencia (€)}}{\text{Período de amortización (h)}}$$

Descripción	Precio licencia (€)	Costo de amortización (€/h)	Cantidad (h)	Subtotal (€)
OrCAD PCB Designer STANDARD	1795,00	0,33	25,00	8,13
MAX+PlusII BASELINE Version 10.2	0,00	0,00	25,00	0,00
Atmel Studio 6.2	0,00	0,00	100,00	0,00
Embarcadero C++Builder XE8 Professional	1353,63	0,25	83,67	20,52
Eclipse Luna IDE for C/C++ Developers	0,00	0,00	85,00	0,00
Android Studio 1.2.2	0,00	0,00	30,00	0,00
SolidWorks + COSMOSMotion 2007	5300,00	0,96	25,00	24,00
SolidWorks 2014-2015	4500,00	0,82	25,00	20,38
MATLAB R2015a	2000,00	0,36	26,67	9,66
Simulink	3000,00	0,54	26,67	14,49
TeXnicCenter + MiKTeX	0,00	0,00	45,00	0,00
Windows 8.1 Pro	199,99	0,04	412,00	14,93
Ubuntu 14	0,00	0,00	85,00	0,00
Total programas informáticos				112,11 €

Cuadro 2.9: Presupuesto de programas informáticos

2.5. Mecánica

La parte mecánica estaba en perfectas condiciones a la hora del comienzo del proyecto, los únicos gastos acaecidos fueron los de sustitución de correas de distribución. Durante el desarrollo del proyecto, al realizarse las distintas pruebas de movimiento, fue preciso reemplazar correas de distribución que fallaron debido a errores en la programación y la electrónica. También se adquirió un dinamómetro digital para registrar el peso total del robot, algo que no se había realizado hasta el momento.

Descripción	**Cantidad**	**Precio unitario (€)**	**Subtotal (€)**
Dinamómetro digital 300 kg	1	53,50	53,50
Correa de distribución Synchroflex 6 / T2.5 / 160 SS	10	4,56	45,60
Total mecánica			99,10 €

Cuadro 2.10: Presupuesto de mecánica

Capítulo 3

Resumen

A continuación se detalla el presupuesto final, que engloba todos los puntos anteriores, añadiendo el 6 % de beneficio industrial, el 13 % en concepto de gastos generales y el correspondiente 21 % de IVA.

Descripción	**Coste (€)**
Total mano de obra	13166,29
Total programas informáticos	112,11
Total informática	251,71
Total electrónica	610,75
Total mecánica	99,10
Presupuesto de Ejecución Material	14239,96 €

Cuadro 3.1: Presupuesto de Ejecución Material

Descripción	**Coste (€)**
Presupuesto de Ejecución Material	14239,96
Gastos generales (13 %)	1851,20
Beneficio industrial (6 %)	854,40
Presupuesto de Ejecución por Contrata	16945,56 €

Cuadro 3.2: Presupuesto de Ejecución por Contrata

Descripción	**Coste (€)**
Presupuesto de Ejecución por Contrata	16945,56
IVA (21 %)	3558,57
Presupuesto Total	20504,12 €

Cuadro 3.3: Presupuesto Total

El presupuesto total del proyecto asciende a VEINTE MIL QUINIENTOS CUATRO EUROS CON DOCE CÉNTIMOS.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Trabajo Fin de Grado

Simulación Virtual, Reconfiguración Electrónica y Programación Real del Movimiento del Robot *Hexapod II*

Anexos

Autor:
Guillermo Oliver Peiró

Tutor:
Dr. Ing. Ind. José Luís Oliver Herrero

Universidad Politécnica de Valencia
Valencia, ESPAÑA

Curso 2014-2015

Parte III

Anexos

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los anexos se incluyen ciertos detalles que profundizan más en algunos aspectos del desarrollo del proyecto. Por un lado se presenta un listado de los materiales utilizados con su correspondiente comentario. También se adjunta aquí un esquema de la electrónica de potencia resultante tras las modificaciones. Puesto que incluir todo el código fuente de la programación hubiese supuesto cientos de páginas de anexo, se incluye únicamente un listado de los ficheros que lo componen, debidamente clasificados y con una pequeña descripción. Por último, se pueden encontrar aquí los detalles relacionados con los aspectos de control y automática del proyecto, estando incluidas las gráficas de las tareas de identificación del modelo y del desarrollo de los controladores.

Capítulo 2

Materiales

Aquí presentaremos todos los materiales utilizados durante la realización del proyecto, clasificados por categorías y con un pequeño comentario acerca del papel que desempeñan. El material más importante del proyecto es el robot *Hexapod II*, y todo el desarrollo girará alrededor de él. Partiendo de esta base, se pueden clasificar los materiales en tres grandes grupos: informática, electrónica, y mecánica.

2.1. Informática

2.1.1. Hardware

- *Raspberry Pi 2 1GB*: Ordenador de a bordo del robot, sistema embebido donde se desarrollará el servidor para el control telemático del robot.
- *Tarjeta de memoria microSD HC 16 GB*: Sistema de almacenamiento usado por el sistema embebido del ordenador de a bordo.
- *Adaptador WiFi USB*: Medio de conexión del ordenador de a bordo con el WiFi de la universidad.
- *Adaptador Bluetooth USB*: Medio de conexión del ordenador de a bordo con el mando inalámbrico.
- *Carcasa para Raspberry Pi*: Carcasa de protección para el ordenador de a bordo.
- *Alargador activo de cable USB*: Cables USB para conectar el robot al PC cuando se realiza el control por puerto serie, además de ser utilizados para actualizar el programa de los micros utilizando el programador de Atmel.
- *Batería de portátil Li-ion*: Fuente de suministro energético para la electrónica de control
- *Controlador inalámbrico de PlayStation3 Sixaxis*: Mando inalámbrico para realizar el control del robot.
- *Ordenador portátil HP Zbook*: Ordenador portátil con Windows 8.1 Pro donde realizar la programación de los micros, el desarrollo del controlador, el programa cliente para Windows, el programa cliente para Android, la planificación de movimientos, y la redacción de los documentos del proyecto.
- *Videocámara HD Sony HDR-700V*: Medio de grabación de vídeo digital para dar testimonio del desarrollo y el movimiento del robot.
- *Ordenador de sobremesa HP*: Ordenador de sobremesa con el sistema operativo Ubuntu Linux 14 donde realizar la programación del código del servidor.

2.1.2. Software

- *PCB Editor*: programa para visualizar la configuración de las placas de circuito impreso del del robot.
- *OrCAD Capture*: programa para visualizar y diseñar los esquemas electrónicos.
- *MAX+PlusII*: programas para visualizar el código VHDL implementado en la FPGA.
- *Atmel Studio 6.2*: entorno de desarrollo para escribir los programas en C para los micros.

- *C++Builder*: entorno de desarrollo para realizar el programa cliente en C++ para el PC.
- *Eclipse Luna IDE for C/C++ Developers*: entorno de desarrollo para realizar el programa servidor en C++ para la Raspberry Pi 2.
- *Android Studio 1.2.2*: entorno de desarrollo para programar en Java para dispositivos Android.
- *SolidWorks + COSMOSMotion 2007 y SolidWorks 2014-2015*: programas para realizar la simulación de los movimientos del robot.
- *MATLAB R2015a*: programa de procesado de datos para obtener gráficas de respuesta y las funciones de transferencia del sistema.
- *Simulink*: programa de simulación integrado en MATLAB para validar la funciones de transferencia del modelo obtenidas.
- *TeXnicCenter + MiKTeX*: programa para redactar los documentos de la memoria en LaTeX.
- *Windows 8.1 Pro*: sistema operativo donde se ha realizado la mayor parte del desarrollo del proyecto.
- *Ubuntu 15*: sistema operativo donde se ha realizado la parte de la compilación cruzada para desarrollar el programa servidor del sistema embebido Linux.

2.2. Electrónica

- *Programador AVRISP mkII*: instrumento para realizar la programación de los micros del robot.
- *Convertidor DC 7V-24V a DC 5V 3A USB*: circuito de suministro de alimentación para el sistema embebido Raspberry Pi 2.
- *Conectores banana 4mm, conectores de alimentación, y cables de conexión de 1mm*: medios empleados para realizar las conexiones electrónicas entre los distintos componentes (etapa de potencia con convertidor DC-DC USB, batería de li-ion con electrónica de control, etc).
- *Transistor MOSFET SI2308BDS, transistor BJT 2N2222, resistencia 47K, resistencia 2K2, diodo Schottky y relé universal 24V 20A*: componentes electrónicos discretos utilizados para poner en funcionamiento la etapa de potencia del robot.
- *Pinza amperimétrica 400 AC DC*: medio de lectura del consumo de potencia de los motores de las patas durante su movimiento.
- *Fuente de alimentación 30V 5A*: fuente de alimentación para la electrónica de control durante las pruebas del desarrollo y antes de disponer de las baterías de Li-ion.
- *Dos baterías de 12V en serie principales de* 68,3 *Ah y dos baterías de 12V en serie de respaldo de* 44,0 *Ah*: fuente de suministro de potencia para los motores de las patas.

2.3. Mecánica

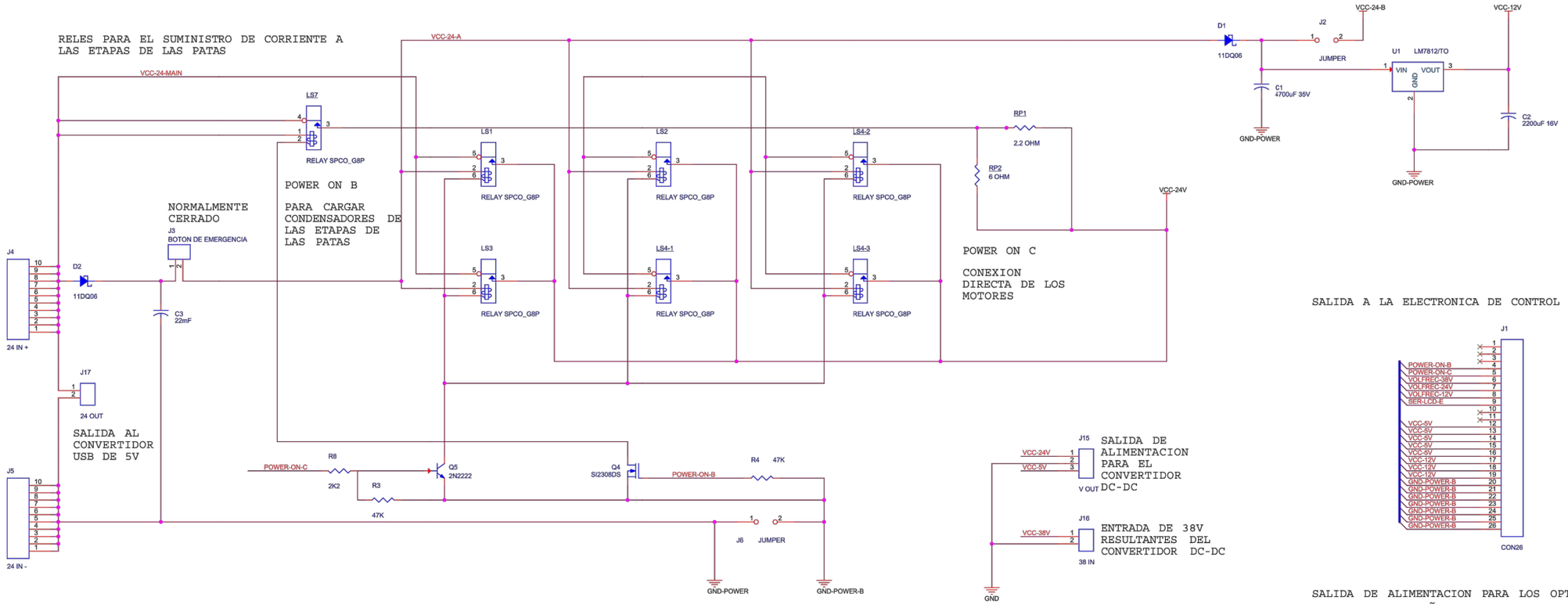
- *Dinamómetro digital 300 kg*: instrumento digital para realizar el registro del peso del robot.
- *Correa de distribución Synchroflex 6 / T2.5 / 160 SS*: correas de distribución que permiten la transferencia de potencia entre los motores de las patas y el husillo por donde se mueve la deslizadera.

Capítulo 3

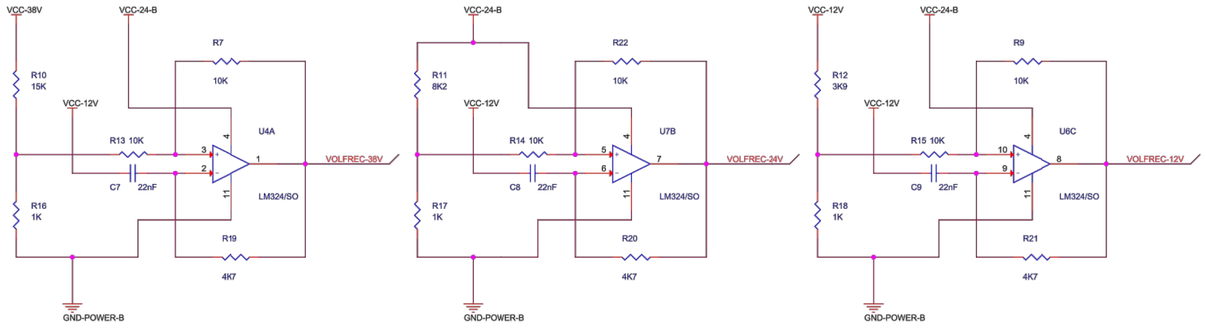
Electrónica

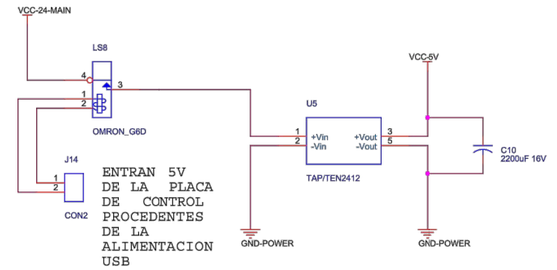
La electrónica de control permaneció inalterada, mientras que se realizaron arreglos y se reconfiguró la etapa de potencia. A continuación se muestra el esquema final del circuito de la electrónica de potencia, una vez revisada y hechas las sustituciones necesarias.

RELES PARA EL SUMINISTRO DE CORRIENTE A LAS ETAPAS DE LAS PATAS

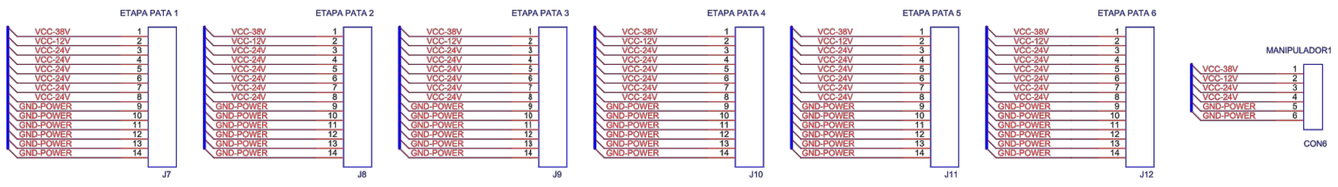


CONVERTIDORES DE TENSION EN FRECUENCIA





SALIDA DE ALIMENTACION PARA LAS ETAPAS DE LAS PATAS



INICIAR ETAPA DE POTENCIA:

1. POWER ON B (1 SEGUNDO)
2. POWER ON B + C (0.5 SEGUNDOS)
3. POWER ON C

Title	MOTION POWER
Size A3	Document Number HEX2-MOTION_POWER Rev A1
Date: Friday, July 03, 2015	Sheet 1 of 1

TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto: SIMULACIÓN VIRTUAL, RECONFIGURACIÓN ELECTRÓNICA Y PROGRAMACIÓN REAL DEL ROBOT HEXAPOD II

Plano: Electrónica de potencia definitiva.

Autor: Guillermo Oliver Peiró

Fecha: Julio 2014

Escala: 1:1

Nº Plano: 1

Capítulo 4

Programación

A continuación se presenta una breve documentación del código, a modo de la relación de los ficheros que comprenden el código fuente y una breve explicación de su contenido. Están organizados según el programa al que pertenecen, pudiendo distinguir entre programas de los micros, programas para Windows, programas para el sistema embebido Linux, y programas para Android. Cabe decir que se presentan únicamente los ficheros de código fuente, excluyendo de esta explicación los ficheros correspondientes a los proyectos y configuraciones de los entornos de desarrollo utilizados.

4.1. Programas de los micros

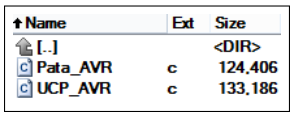


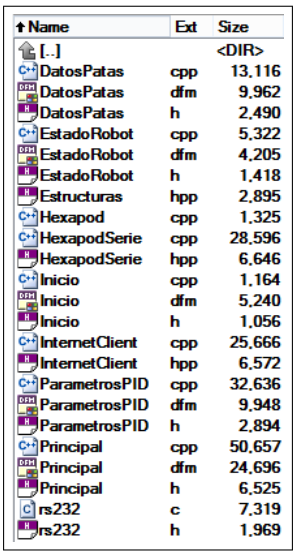
Figura 4.1: Ficheros del código fuente del programa de los micros

Los programas de los micros están escritos en lenguaje C utilizando el entorno de desarrollo Atmel Studio 6.2, proporcionado por la empresa que comercializa los micros. Se han desarrollado dos programas distintos para los micros, uno para el micro principal y otro común para los micros de las seis patas.

Nombre del fichero	**Tamaño (bytes)**	**Longitud (líneas)**	**Descripción**
UCP_AVR.c	133186	3814	Programa para el micro principal.
Pata_AVR.c	124406	3272	Programa para el micro de las patas.

Cuadro 4.1: Descripción ficheros de código fuente de los programas de los micros

4.2. Programa cliente para Windows



Name	Ext	Size
[..]		<DIR>
DatosPatas	cpp	13,116
DatosPatas	dfm	9,962
DatosPatas	h	2,490
EstadoRobot	cpp	5,322
EstadoRobot	dfm	4,205
EstadoRobot	h	1,418
Estructuras	hpp	2,895
Hexapod	cpp	1,325
HexapodSerie	cpp	28,596
HexapodSerie	hpp	6,646
Inicio	cpp	1,164
Inicio	dfm	5,240
Inicio	h	1,056
InternetClient	cpp	25,666
InternetClient	hpp	6,572
ParametrosPID	cpp	32,636
ParametrosPID	dfm	9,948
ParametrosPID	h	2,894
Principal	cpp	50,657
Principal	dfm	24,696
Principal	h	6,525
rs232	c	7,319
rs232	h	1,969

Figura 4.2: Ficheros del código fuente del programa cliente para Windows

El programa cliente para Windows está escrito en lenguaje C++ utilizando el entorno de desarrollo C++Builder para poder realizar la interfaz gráfica rápidamente, además de que el autor tiene mucha experiencia utilizando este entorno facilitando así el desarrollo. Se ha desarrollado varias clases con interfaz gráfica (pantallas) y otras clases internas adicionales.

Nombre del fichero	**Tamaño (bytes)**	**Longitud (líneas)**	**Descripción**
DatosPatas.cpp	13116	288	Programa para la clase *DatosPatas*.
DatosPatas.dfm	9962	493	Interfaz gráfica de la clase *DatosPatas*.
DatosPatas.h	2490	93	Cabecera de la clase *DatosPatas*.
EstadoRobot.cpp	5322	154	Programa para la clase *EstadoRobot*.
EstadoRobot.dfm	4205	212	Interfaz gráfica de la clase *EstadoRobot*.
EstadoRobot.h	1418	51	Cabecera de la clase *EstadoRobot*.
Estructuras.hpp	2895	105	Cabecera de definición de las estructuras de datos.
Hexapod.cpp	1325	44	Programa principal donde se encuentra el procedimiento de inicio del ejecutable.
HexapodSerie.cpp	28596	774	Clase de comunicación por puerto serie.
HexapodSerie.hpp	6646	261	Cabecera de la clase de comunicación por puerto serie.
Inicio.cpp	1164	44	Programa para la clase *Inicio*.
Inicio.dfm	5240	134	Interfaz gráfica de la clase *Inicio*.
Inicio.h	1056	32	Cabecera de la clase *Inicio*.
InternetClient.cpp	25666	675	Clase de comunicación por Internet.
InternetClient.hpp	6572	256	Cabecera de la clase de comunicación por Internet.
ParametrosPID.cpp	32636	862	Programa para la clase *ParametrosPID*.
ParametrosPID.dfm	9948	487	Interfaz gráfica de la clase *ParametrosPID*.
ParametrosPID.h	2894	96	Cabecera de la clase *ParametrosPID*.
Principal.cpp	50657	1511	Programa para la clase *Principal*.
Principal.dfm	24696	1051	Interfaz gráfica de la clase *Principal*.
Principal.h	6525	195	Cabecera de la clase *Principal*.
rs232.c	7319	259	Modulo para la comunicación por puerto serie.
rs232.h	1969	72	Cabecera del módulo para la comunicación por puerto serie.

Cuadro 4.2: Descripción ficheros de código fuente del programa cliente para Windows

4.3. Programa servidor para el sistema embebido Linux

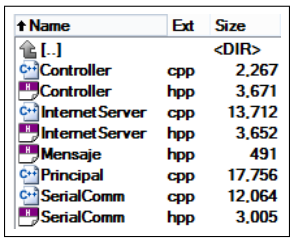


Figura 4.3: Ficheros del código fuente del programa para Linux.

El código fuente desarrollado para el entorno embebido Linux está escrito en C++ utilizando el entorno de desarrollo Eclipse Luna. Se ha desarrollado un programa principal con las correspondientes clases de comunicación.

Nombre del fichero	**Tamaño (bytes)**	**Longitud (líneas)**	**Descripción**
Controller.cpp	2267	105	Clase de comunicación por Bluetooth con el mando.
Controller.hpp	3671	115	Cabecera de la clase de comunicación por Bluetooth con el mando.
InternetServer.cpp	13712	434	Clase de comunicación por Internet.
InternetServer.hpp	3652	132	Cabecera de la clase de comunicación por Internet.
Mensaje.hpp	491	25	Cabecera de definición de la estructura de los mensajes.
Principal.cpp	17756	513	Programa principal del servidor.
SerialComm.cpp	12064	426	Clase de comunicación por puerto serie.
SerialComm.hpp	3005	106	Cabecera de la clase de comunicación por puerto serie.

Cuadro 4.3: Descripción ficheros de código fuente del programa servidor para el sistema embebido Linux

4.4. Programa cliente para Android

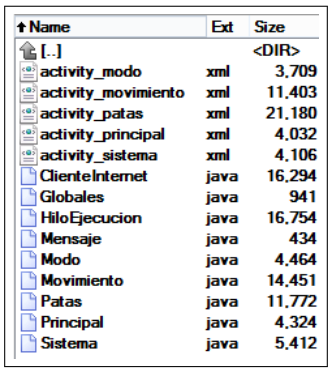


Figura 4.4: Ficheros del código fuente del programa de para Android.

El programa cliente para el dispositivo Android se desarrolló en el lenguaje Java utilizando el entorno de desarrollo Android Studio 1.2.2. Consta de varias pantallas o actividades con sus clases y otras clases internas para el correcto funcionamiento.

Nombre del fichero	**Tamaño (bytes)**	**Longitud (líneas)**	**Descripción**
activity_modo.xml	3709	99	Interfaz gráfica de la clase *Modo*.
activity_movimiento.xml	11403	309	Interfaz gráfica de la clase *Movimiento*.
activity_patas.xml	21180	529	Interfaz gráfica de la clase *Patas*.
activity_principal.xml	4032	103	Interfaz gráfica de la clase *Principal*.
activity_sistema.xml	4106	103	Interfaz gráfica de la clase *Sistema*.
ClienteInternet.java	16294	515	Clase para la comunicación por Internet.
Globales.java	941	25	Clase para alojar las instancias de los atributos y las clases globales.
HiloEjecucion.java	16754	339	Clase para ejecutar procesos en segundo plano.
Mensaje.java	434	18	Clase con la estructura de los mensajes.
Modo.java	4464	118	Código de la clase *Modo*.
Movimiento.java	14451	297	Código de la clase *Movimiento*.
Patas.java	11772	288	Código de la clase *Patas*.
Principal.java	4324	130	Código de la clase *Principal*.
Sistema.java	5412	136	Código de la clase *Sistema*.

Cuadro 4.4: Descripción ficheros de código fuente del programa servidor para el sistema embebido Linux

Capítulo 5

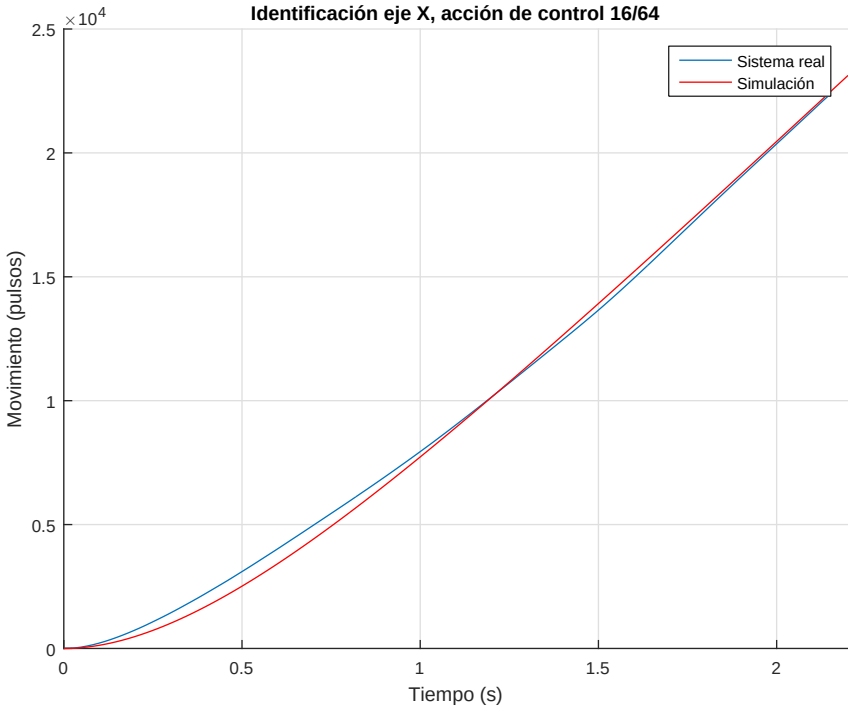
Control

5.1. Identificación del proceso

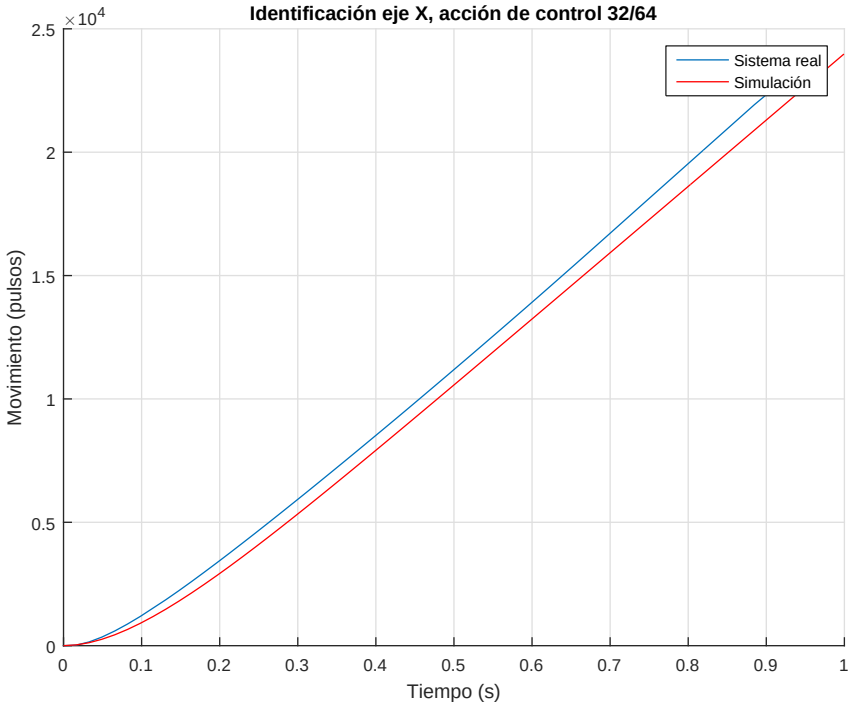
Para la identificación del proceso se tomaron datos de movimiento de los distintos ejes de una pata y se utilizó *MATLAB* y la herramienta *Simulink* para realizar el estudio. Usando *MATLAB* se llevó a cabo el análisis de datos, llegando a la conclusión de que la respuesta del sistema para la posición ante cambios en al referencia se trataba de un modelo de primer orden con integrador y obteniendo los parámetros que lo caracterizan. Una vez obtenido el modelo, se discretizaron sus parámetros y utilizando *Simulink* se realizó la simulación del sistema ante una entrada escalón para comparar el comportamiento de la expresión del modelo obtenida con los datos tomados del sistema real. Superponiendo ambos datos se puede observar cómo de bien se ajusta el modelo identificado y simulado a la respuesta real del sistema.

A continuación se presentan los datos obtenidos para la posición ante un cambio en la acción de control, junto con las respectivas simulaciones, y terminando con una tabla con los modelos obtenidos ordenados por patas y acción de control aplicada. Se muestra la respuesta del sistema real en azul y el simulado en rojo, y en la mayoría de ocasiones están superpuestas ambas líneas, indicando que el ajuste es prácticamente perfecto.

5.1.1. Eje X

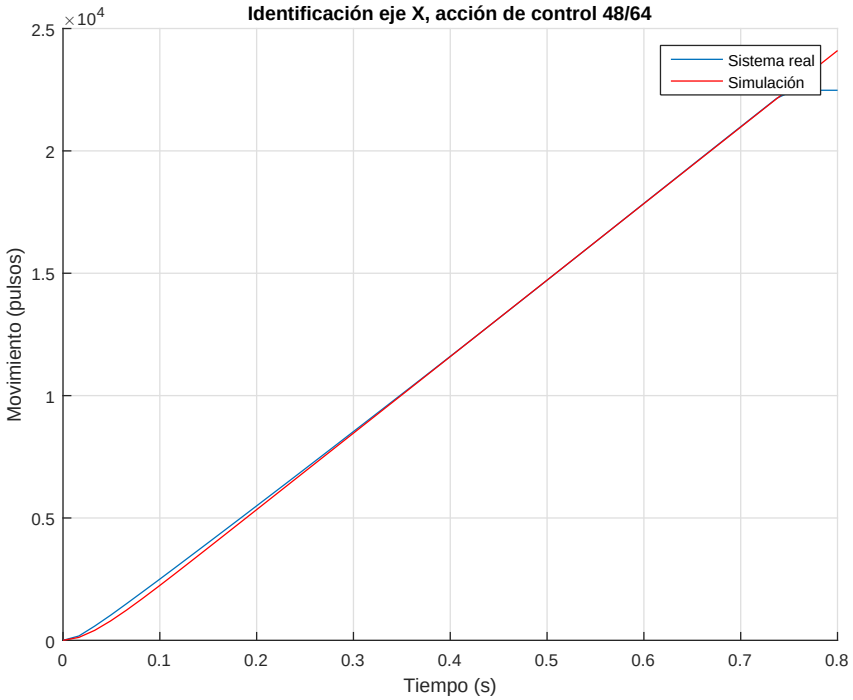


(a) Acción de control de 16/64.

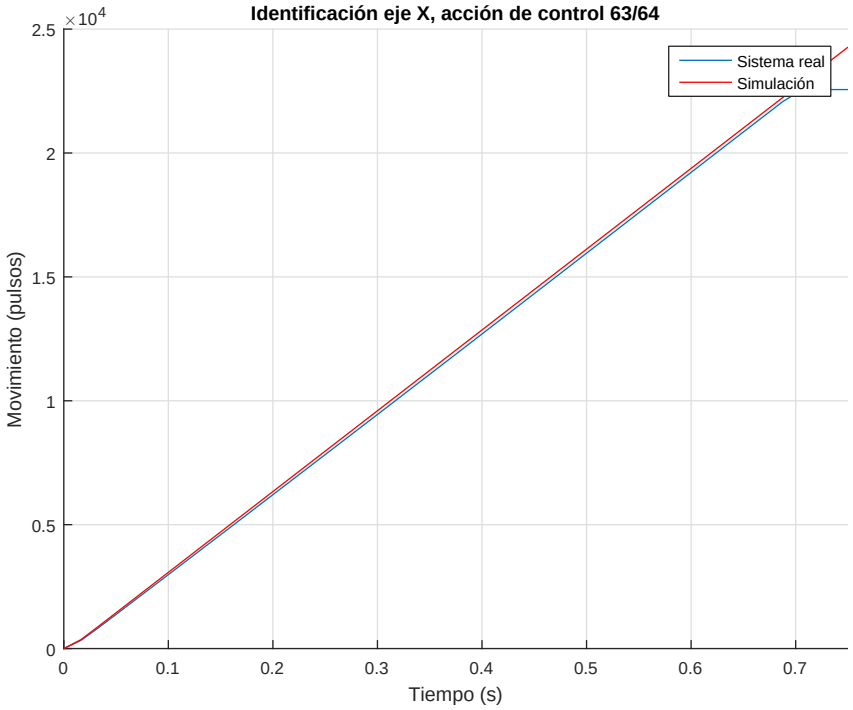


(b) Acción de control de 32/64.

Figura 5.1: Identificación del modelo del sistema para la posición para el eje X según la acción de control aplicada (parte 1).



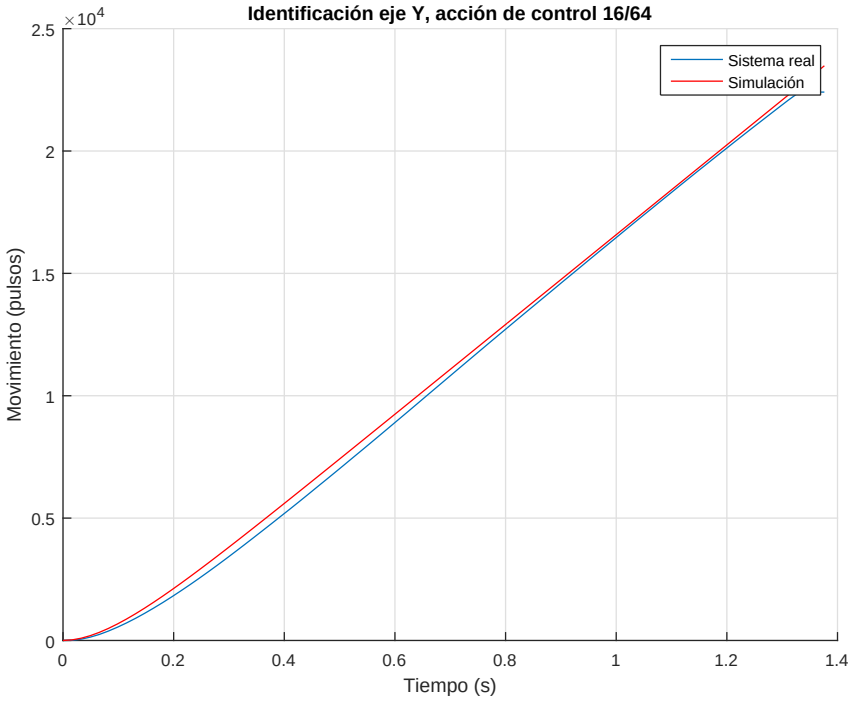
(a) Acción de control de 48/64.



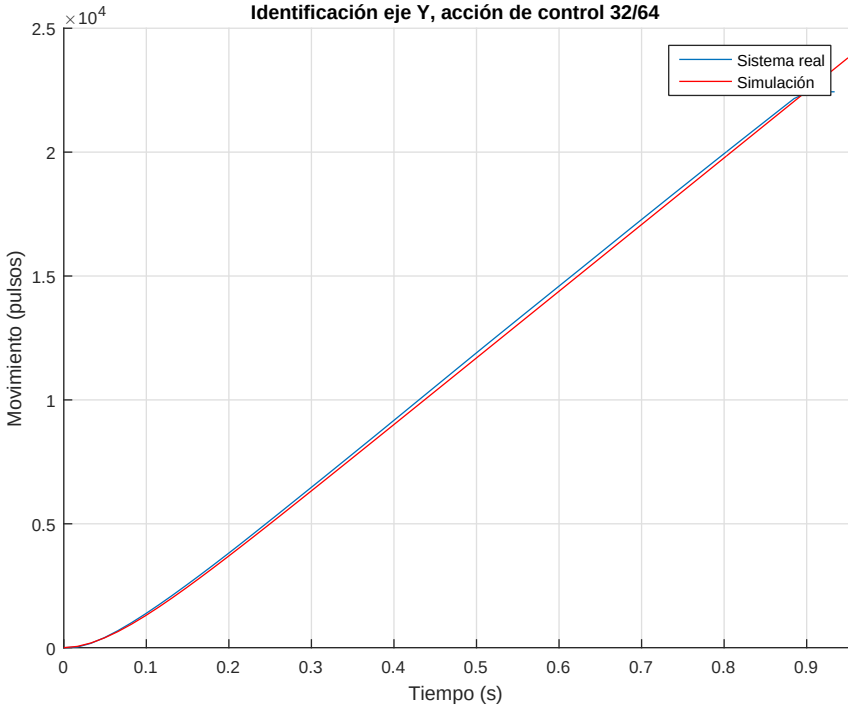
(b) Acción de control de 63/64.

Figura 5.2: Identificación del modelo del sistema para la posición para el eje X según la acción de control aplicada (parte 2).

5.1.2. Eje Y

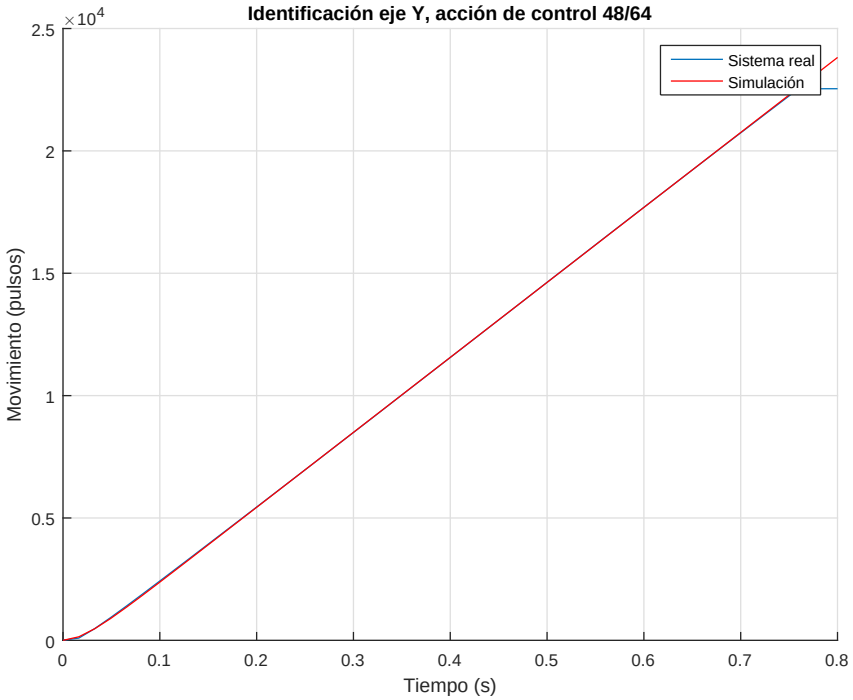


(a) Acción de control de 16/64.

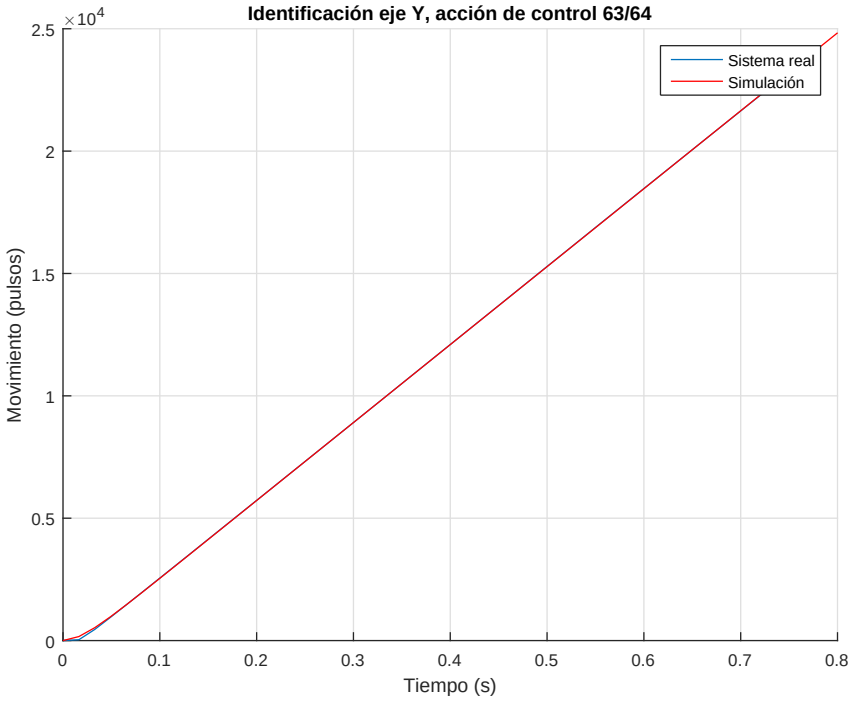


(b) Acción de control de 32/64.

Figura 5.3: Identificación del modelo del sistema para la posición para el eje Y según la acción de control aplicada (parte 1).



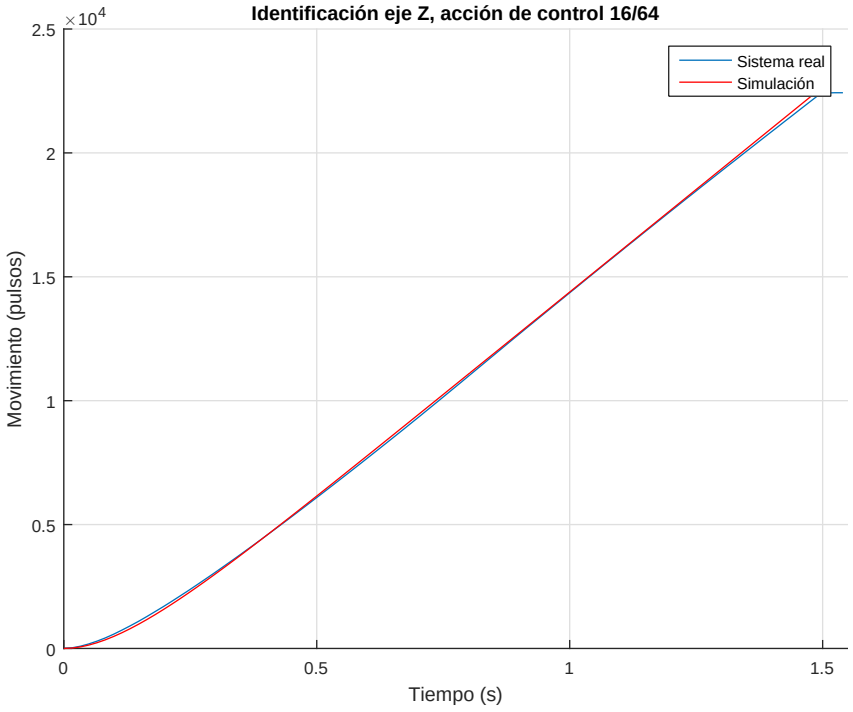
(a) Acción de control de 48/64.



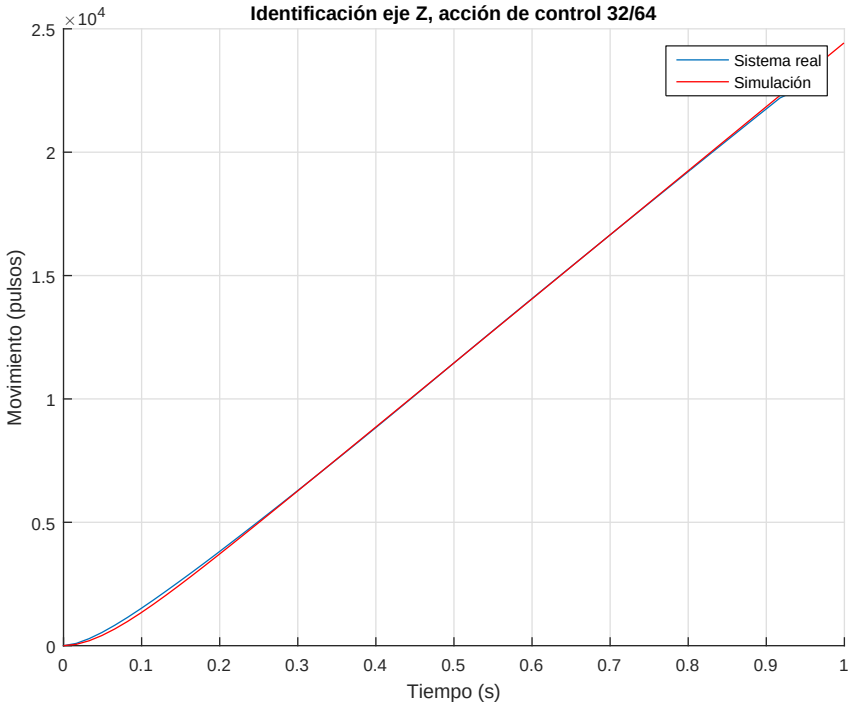
(b) Acción de control de 63/64.

Figura 5.4: Identificación del modelo del sistema para la posición para el eje Y según la acción de control aplicada (parte 2).

5.1.3. Eje Z

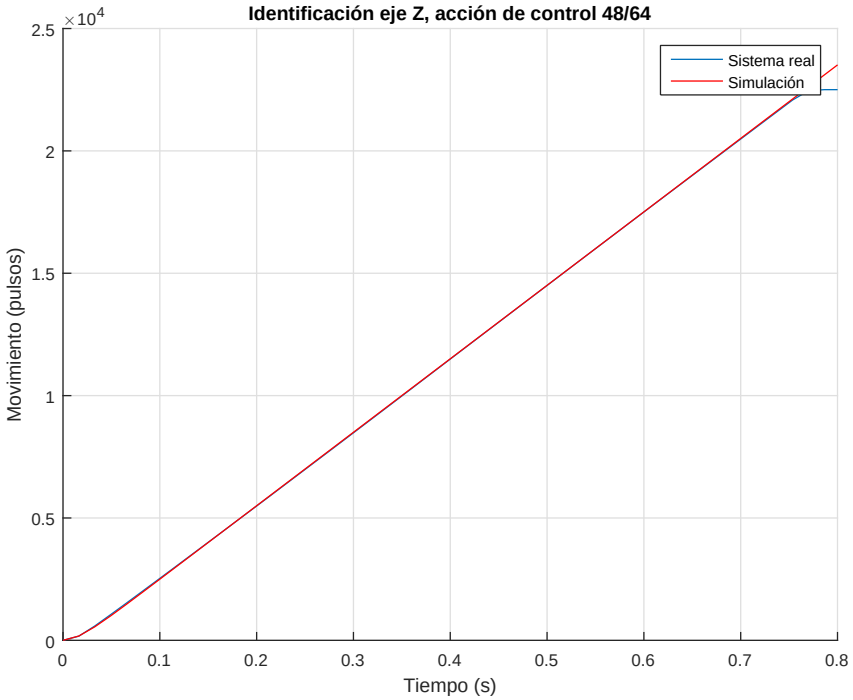


(a) Acción de control de 16/64.

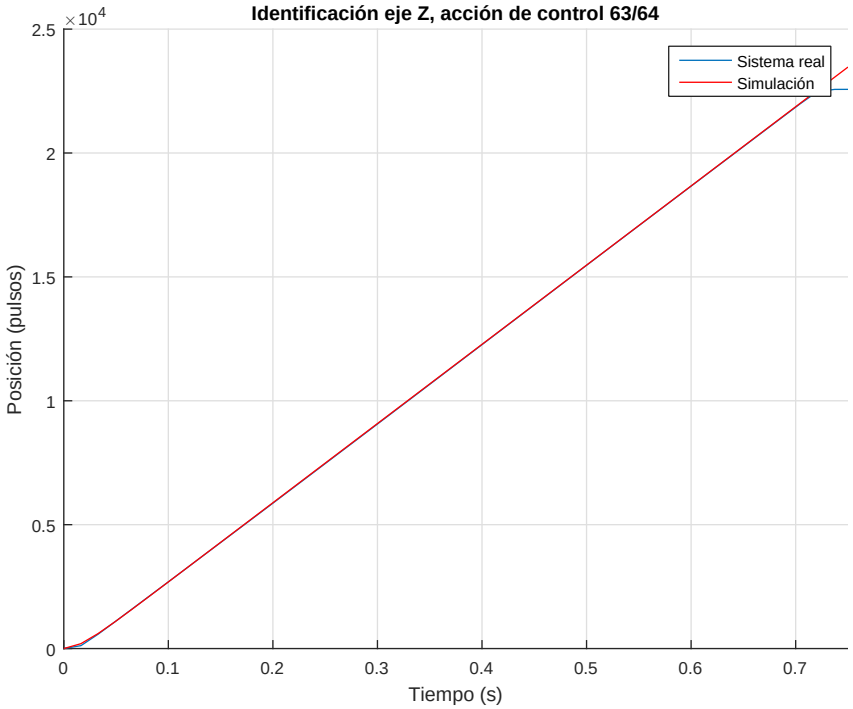


(b) Acción de control de 32/64.

Figura 5.5: Identificación del modelo del sistema para la posición para el eje Z según la acción de control aplicada (parte 1).



(a) Acción de control de 48/64.



(b) Acción de control de 63/64.

Figura 5.6: Identificación del modelo del sistema para la posición para el eje Z según la acción de control aplicada (parte 2).

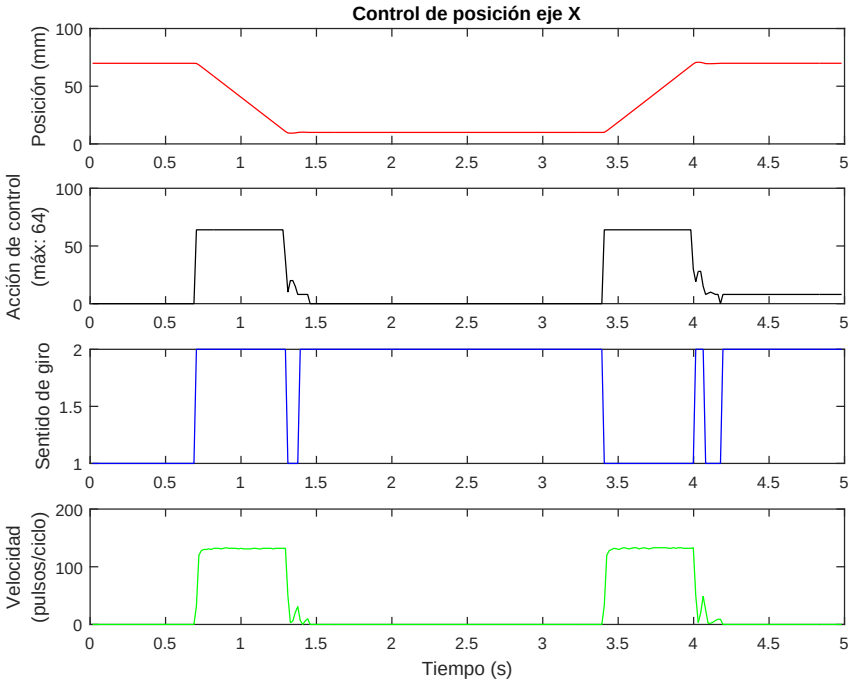
Acción de control	K	τ **(s)**	**Modelo obtenido**
	Eje X		
16	842,9209	0,491	$G(s) = \frac{842,9209}{s(1+0,491s)}$
32	841,0454	0,1085	$G(s) = \frac{841,0454}{s(1+0,1085s)}$
48	651,2960	0,0292	$G(s) = \frac{651,2960}{s(1+0,0292s)}$
63	517,6040	0,0057	$G(s) = \frac{517,6040}{s(1+0,0057s)}$
	Eje Y		
16	1133,5	0,0963	$G(s) = \frac{1133,5}{s(1+0,0963s)}$
32	841,0136	0,0655	$G(s) = \frac{841,0136}{s(1+0,0655s)}$
48	638,2412	0,0226	$G(s) = \frac{638,2412}{s(1+0,0226s)}$
63	505,5261	0,0203	$G(s) = \frac{507,3991}{s(1+0,0203s)}$
	Eje Z		
16	1035,1	0,1315	$G(s) = \frac{1035,1}{s(1+0,1315s)}$
32	811,5133	0,0588	$G(s) = \frac{811,5133}{s(1+0,0588s)}$
48	625,4408	0,0168	$G(s) = \frac{625,4408}{s(1+0,0168s)}$
63	507,3991	0,0159	$G(s) = \frac{507,3991}{s(1+0,0159s)}$

Cuadro 5.1: Funciones de transferencia obtenidas de la respuesta del sistema para la posición.

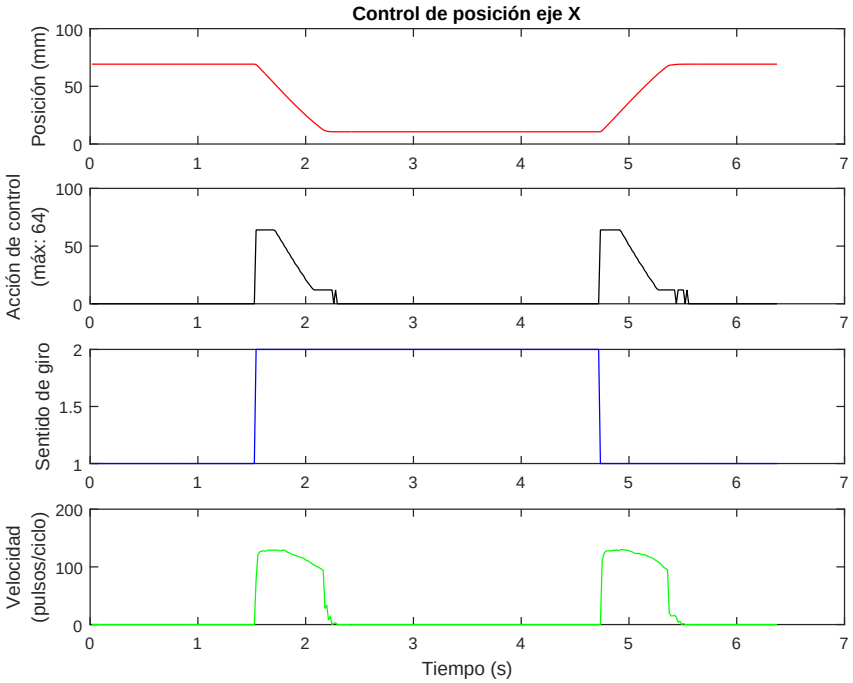
De los modelos obtenidos del sistema para la posición se aprecia una clara tendencia de disminución de la K y la τ, a medida que va aumentando la acción de control, es decir, la tensión aplicada a bornes del motor. El modelo se aprecia que es no lineal, ya que doblando la tensión aplicada no supone doblar la velocidad a la que responde el sistema, y la velocidad no aumenta indefinidamente, sino que para acciones de control elevadas no se aprecia demasiada diferencia en la respuesta. Además existe una zona muerta, puesto que el motor necesita una tensión mínima para poder aplicar suficiente par para vencer el par resistente que ofrece el montaje y ponerlo en movimiento. Por último tiene la peculiaridad de que el comportamiento es distinto según sea el movimiento hacia un lado o hacia otro.

5.2. Desarrollo del controlador para la posición

De manera experimental se realizó un ajuste para un controlador tipo PD para la posición.



(a) Controlador proporcional.



(b) Controlador proporcional-derivativo obtenido.

Figura 5.7: Respuesta del sistema con distintos controladores para la posición.

5.3. Desarrollo del controlador para la velocidad

Lo primero fue implementar un controlador para regular la velocidad en el código del programa del micro de las patas. Una vez implementado, se realizaron una serie de pruebas para ajustar empíricamente el controlador siguiendo unas pautas establecidas.

En primer lugar se modificó la ganancia, utilizando un controlador proporcional manteniendo la parte integral y la parte derivativa desactivadas, de modo que la respuesta en la velocidad en los momentos iniciales no fuera demasiado brusca. El tiempo integral se introdujo con un valor alto, y se fue disminuyendo hasta que la respuesta en régimen permanente tuviera un error prácticamente nulo con respecto a la referencia, intentando que no se produjeran oscilaciones, y modificando la constante proporcional ligeramente en algún caso. Se intentó afinar el controlador utilizando la acción derivativa, pero debido a que es muy susceptible al ruido de alta frecuencia, las pequeñas variaciones de la lectura de la velocidad provocaban oscilaciones en la respuesta, sin llegar a mejorar el controlador, por lo que se decidió mantenerse con un controlador PI. En control de posición se realiza mediante un todo/nada que corta la alimentación a los motores una vez se alcanza la posición deseada.

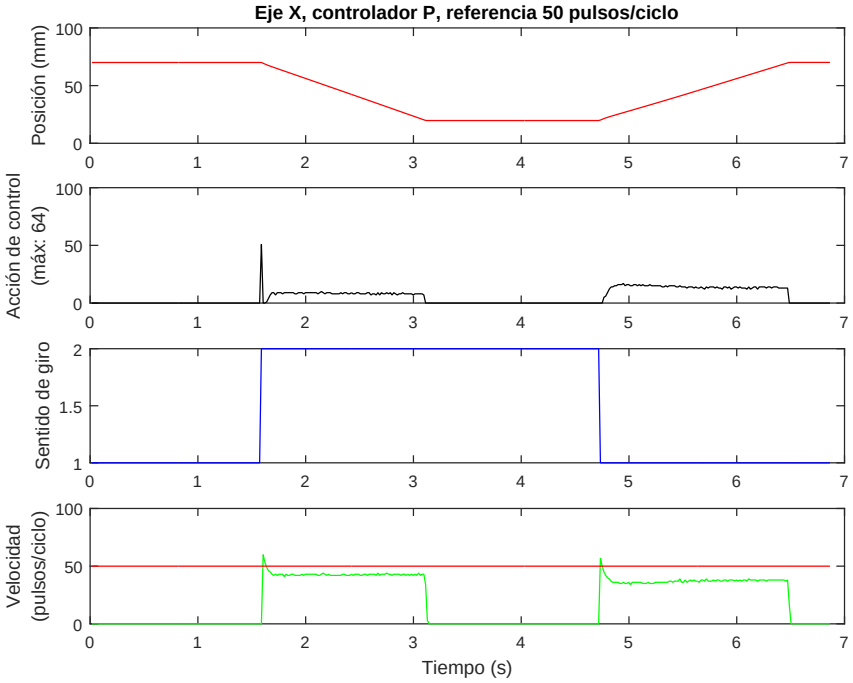
A continuación se muestran las respuestas del sistema obtenidas con la aplicación de algunos controladores intermedios seleccionados y con el controlador final para cada uno de los ejes. En las figuras podemos observar la posición en milímetros, la acción de control aplicada (sobre un máximo de 64), el sentido de giro de los motores, y la velocidad de giro del motor en pulsos por ciclo junto al valor de la referencia a seguir.

Se puede observar como se empieza con un error de velocidad grande al aplicar únicamente la acción proporcional. Una vez incluida la acción integral, a medida que va disminuyendo el valor del tiempo integral va disminuyendo el error de velocidad hasta realizar un buen seguimiento de la referencia de velocidad, incluso para distintos valores de ésta.

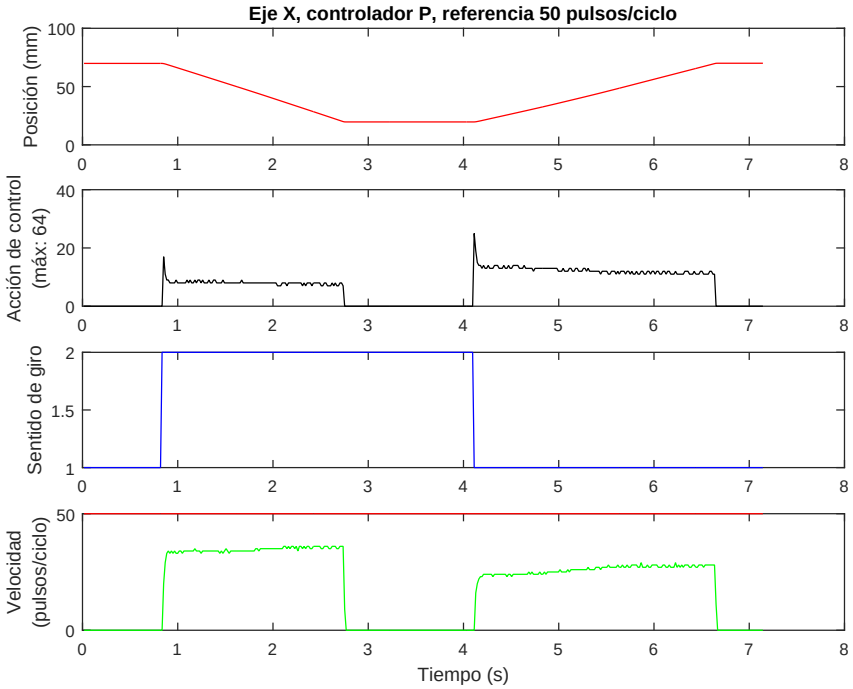
Eje	Kp	T_d **(s)**	T_i **(s)**
X	0,3	0	0,01
Y	0,5	0	0,01
Z	0,5	0	0,01

Cuadro 5.2: Parámetros de los controladores para la velocidad definitivos para cada eje.

5.3.1. Eje X

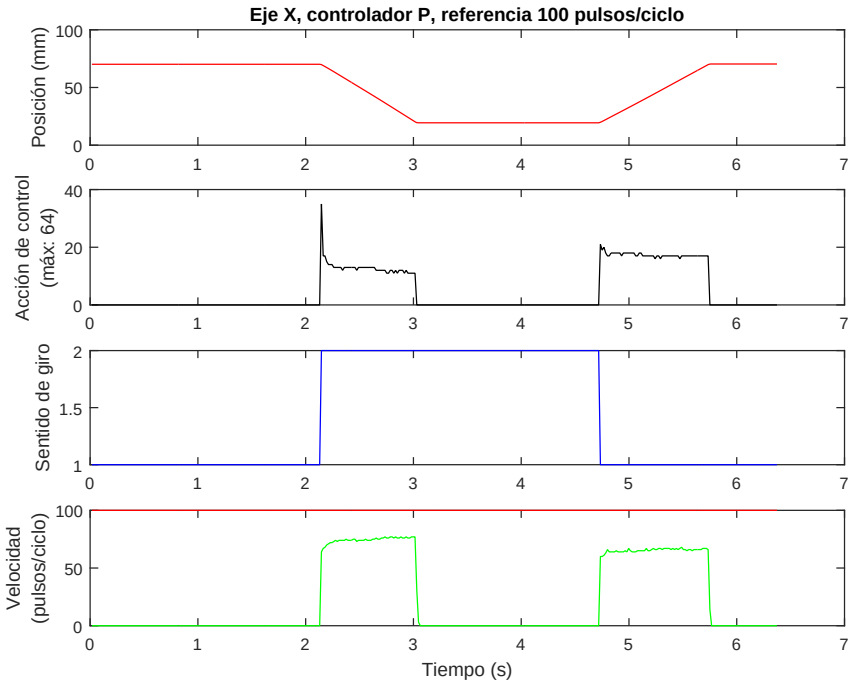


(a) Controlador proporcional con $K_c = 1$ para una referencia de velocidad de 50 pulsos/ciclo.

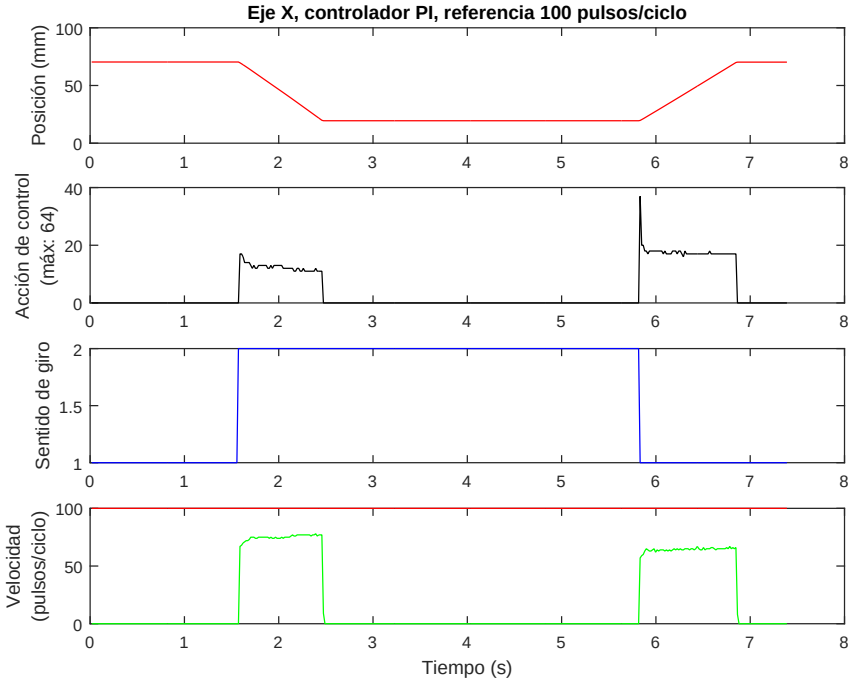


(b) Controlador proporcional con $K_c = 0{,}5$ para una referencia de velocidad de 50 pulsos/ciclo.

Figura 5.8: Ajuste del controlador de velocidad para el eje X (parte 1).

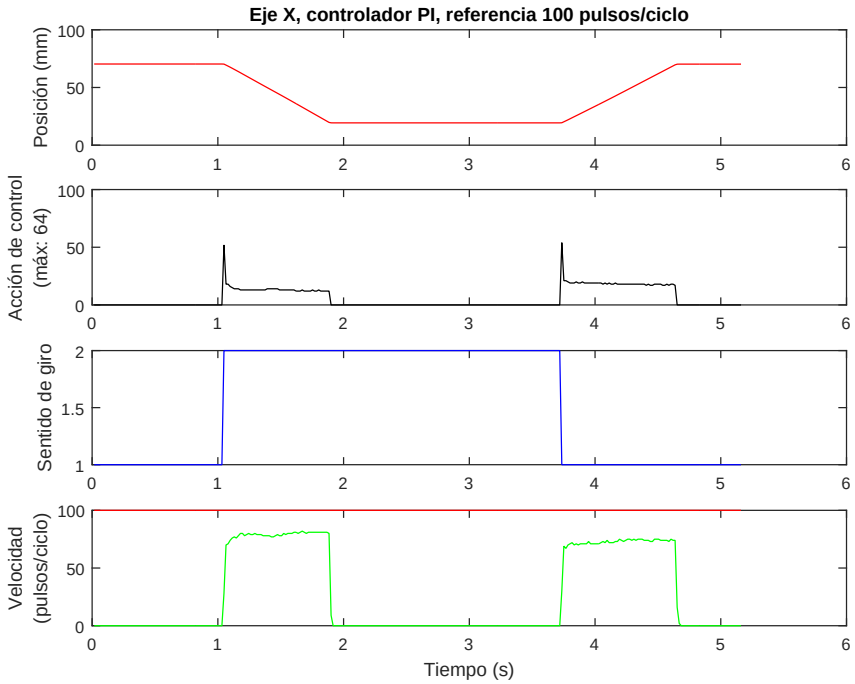


(a) Controlador proporcional con $K_c = 0{,}5$ para una referencia de velocidad de 100 pulsos/ciclo.

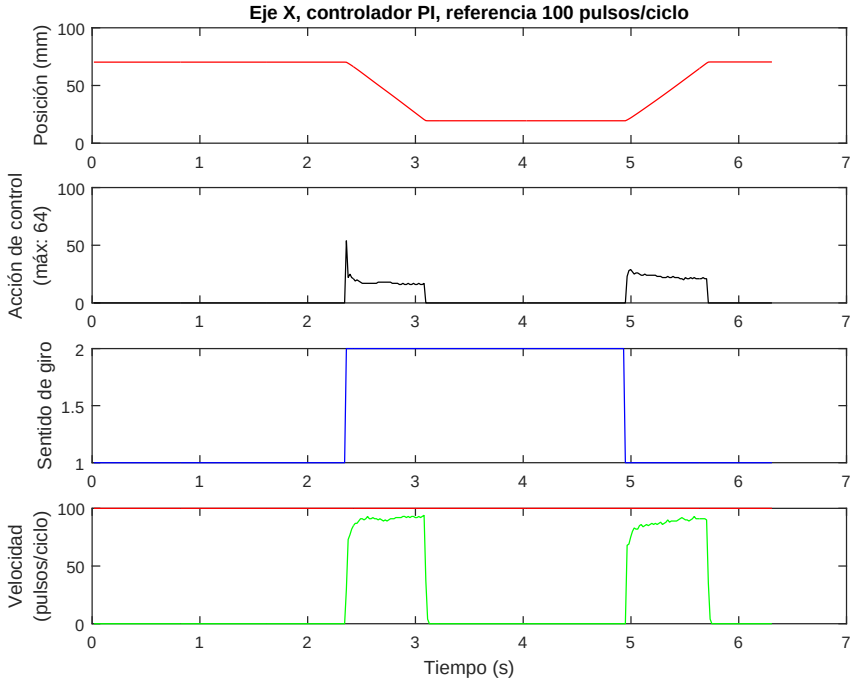


(b) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 1$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.9: Ajuste del controlador de velocidad para el eje X (parte 2).

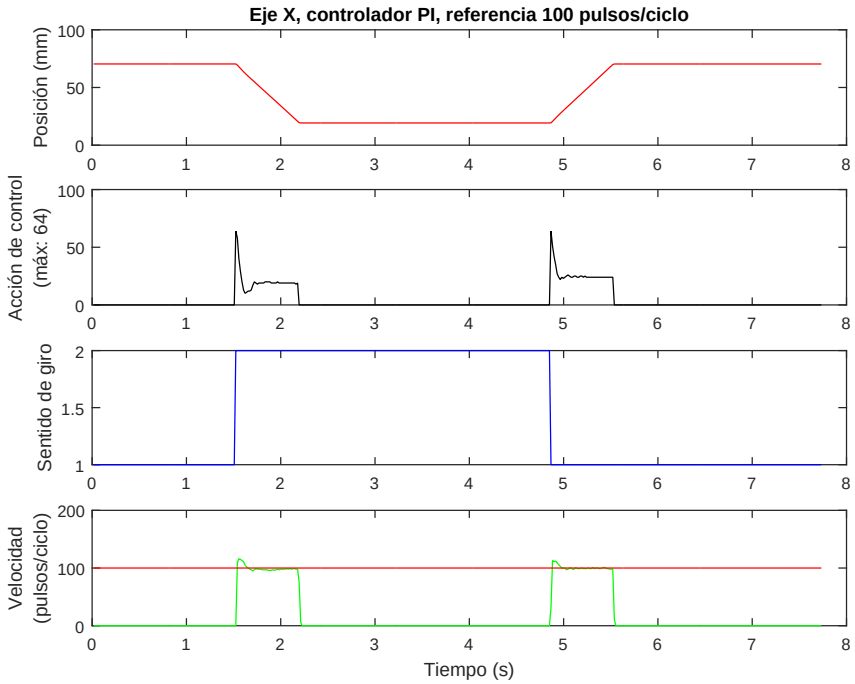


(a) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}1$ para una referencia de velocidad de 100 pulsos/ciclo.

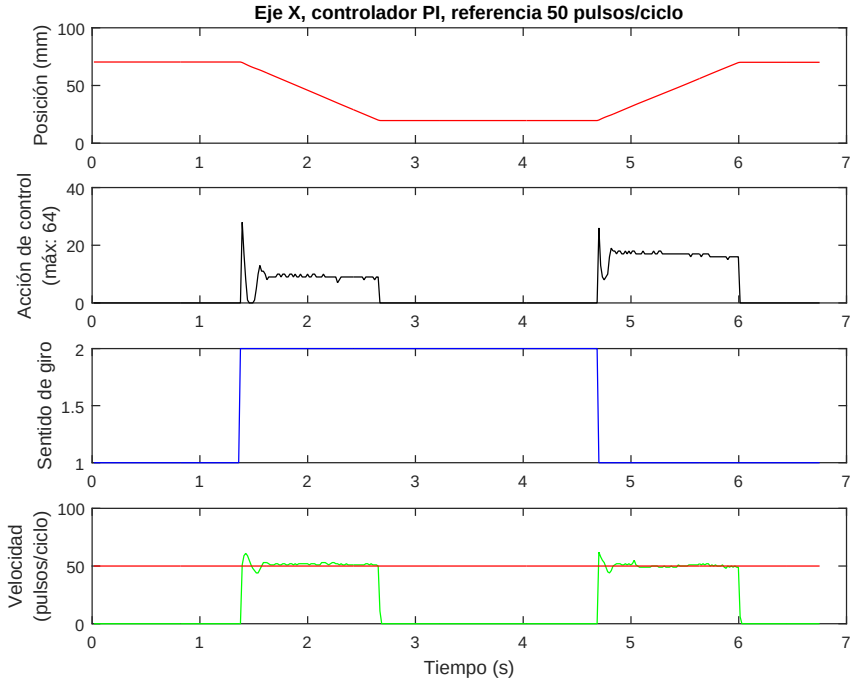


(b) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}05$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.10: Ajuste del controlador de velocidad para el eje X (parte 3).

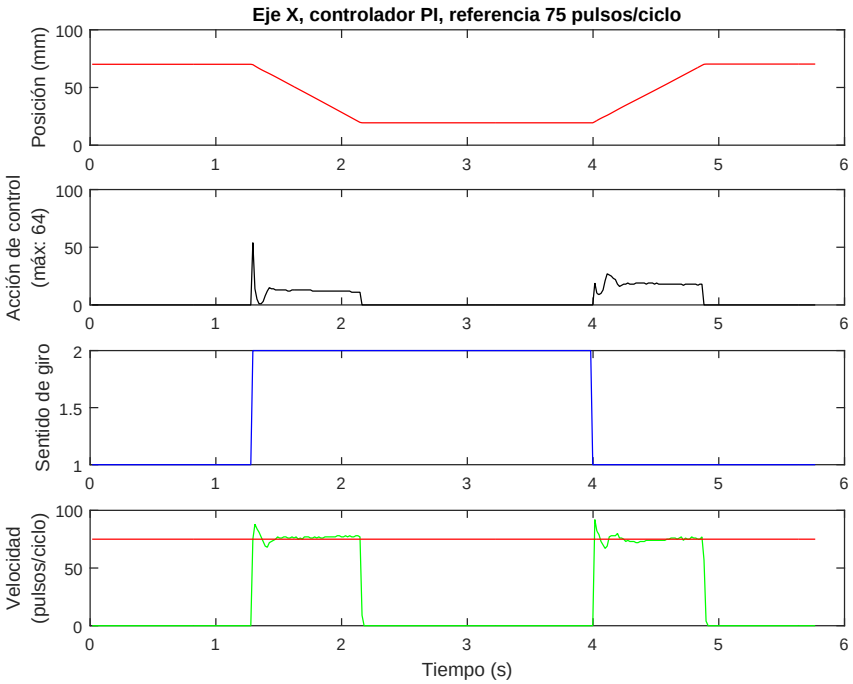


(a) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 100 pulsos/ciclo.

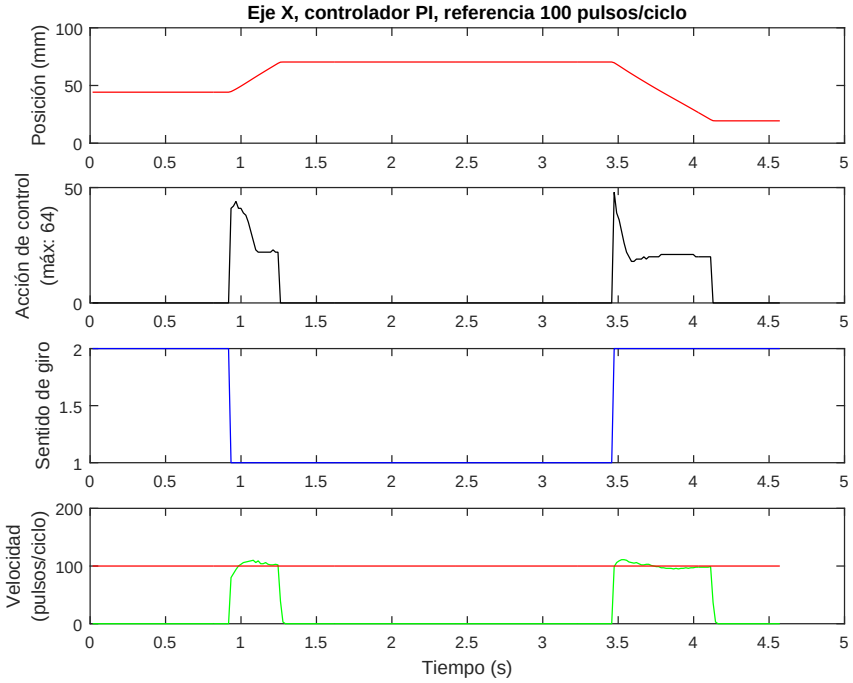


(b) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 50 pulsos/ciclo.

Figura 5.11: Ajuste del controlador de velocidad para el eje X (parte 4).



(a) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 75 pulsos/ciclo.



(b) Controlador proporcional-integral con $K_c = 0{,}35$ y $T_i = 0{,}01$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.12: Ajuste del controlador de velocidad para el eje X (parte 5).

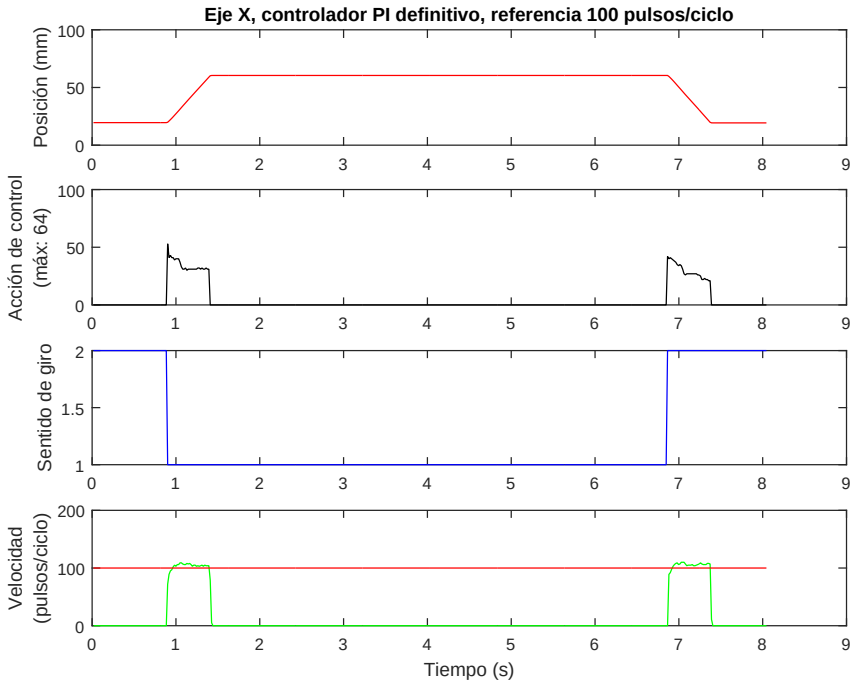
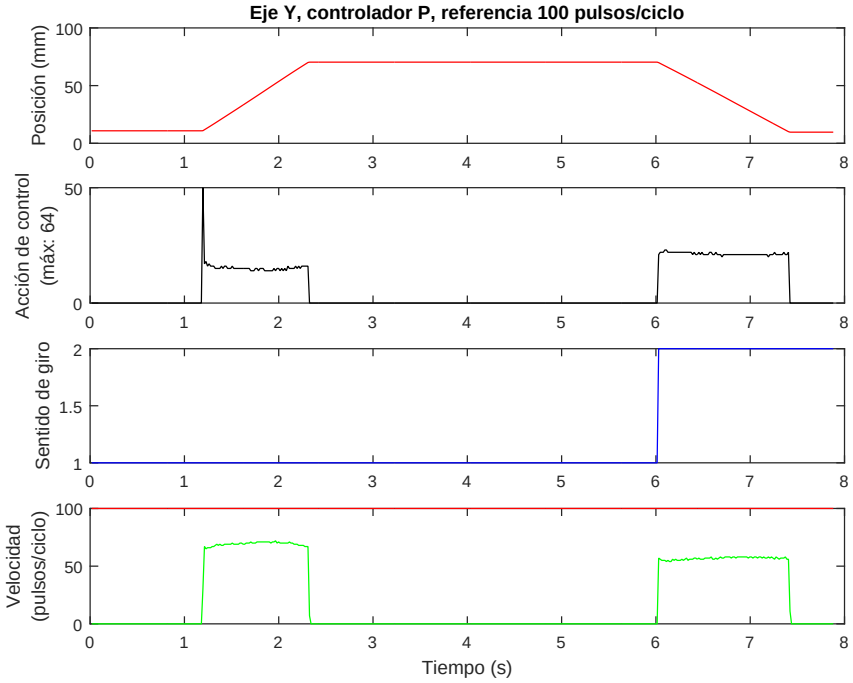
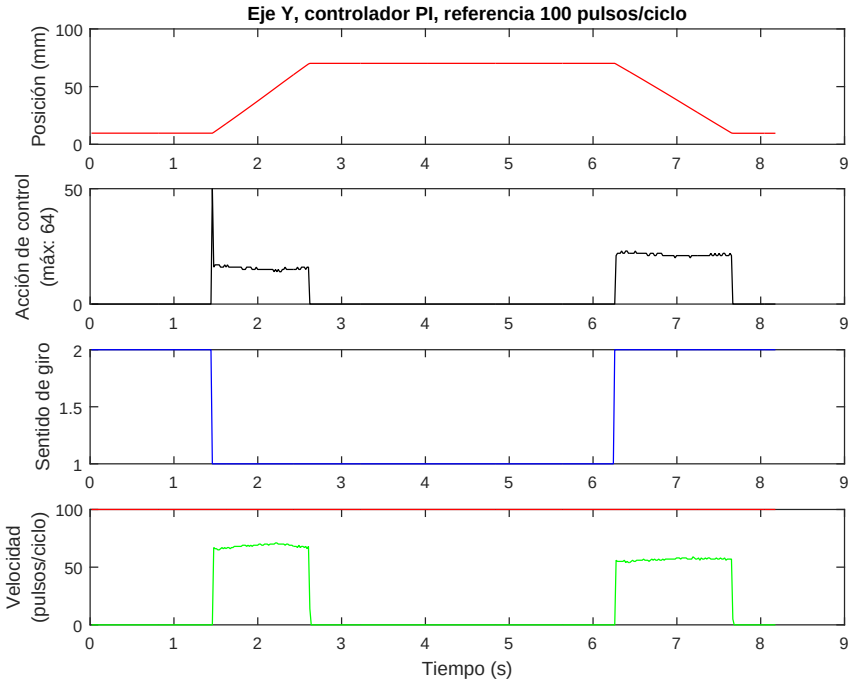


Figura 5.13: Controlador proporcional-integral definitivo para el eje X con $K_c = 0{,}3$ y $T_i = 0{,}01$ para una referencia de velocidad de 100 pulsos/ciclo.

5.3.2. Eje Y

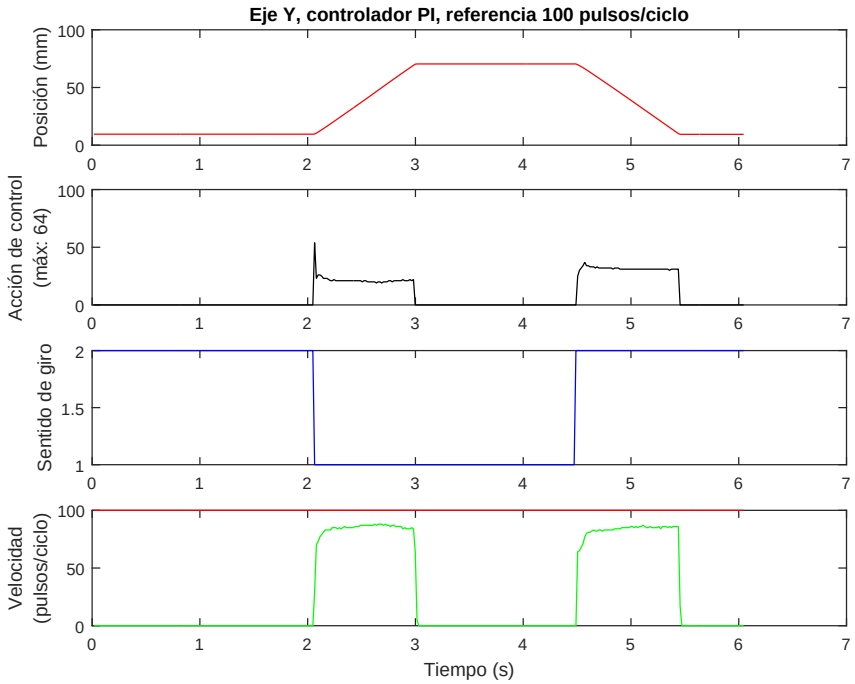


(a) Controlador proporcional con $K_c = 0{,}5$ para una referencia de velocidad de 100 pulsos/ciclo.



(b) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}5$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.14: Ajuste del controlador de velocidad para el eje Y (parte 1).



(a) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}05$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.15: Ajuste del controlador de velocidad para el eje Y (parte 2).

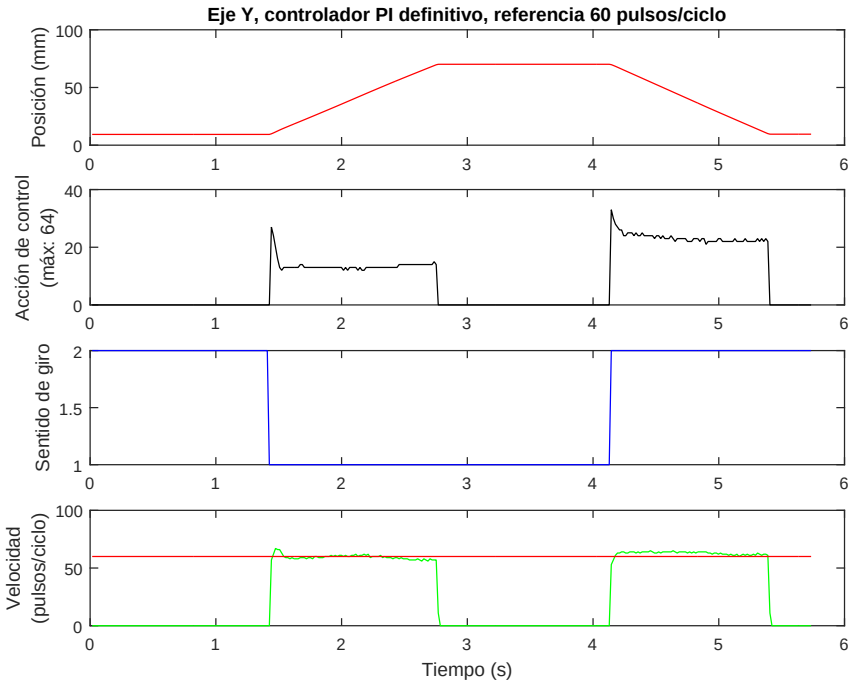


Figura 5.16: Controlador proporcional-integral definitivo para el eje Y con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 60 pulsos/ciclo.

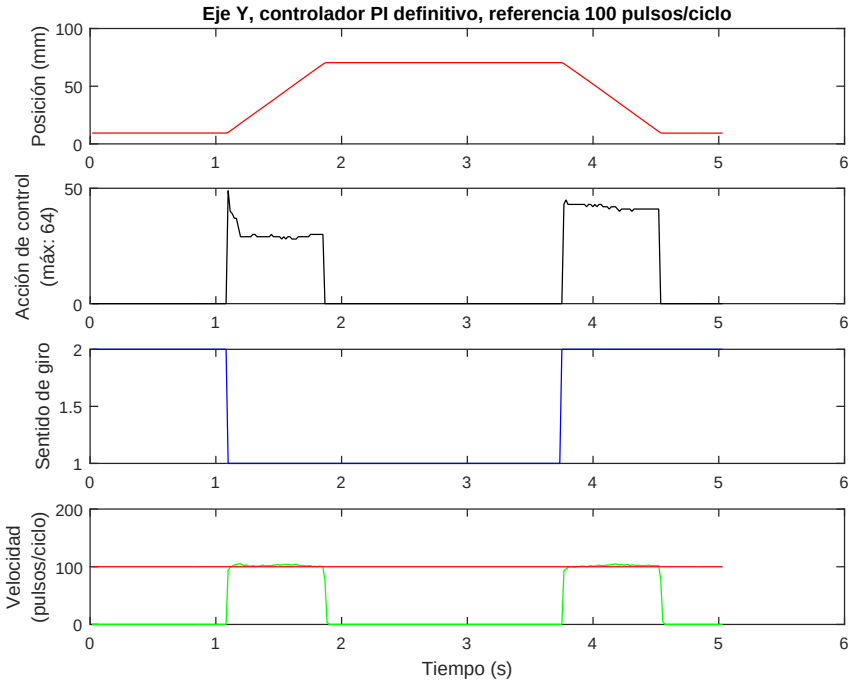
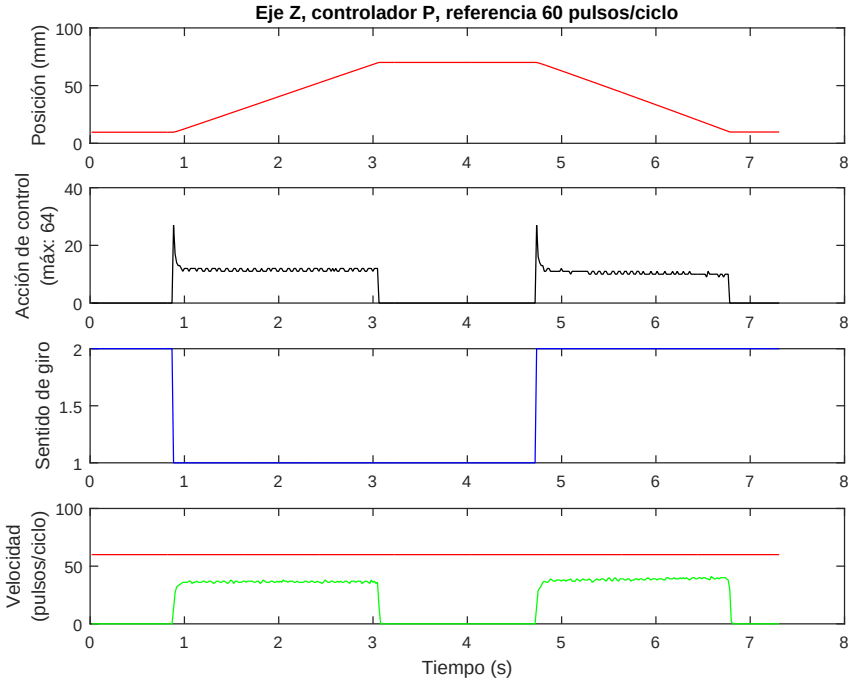
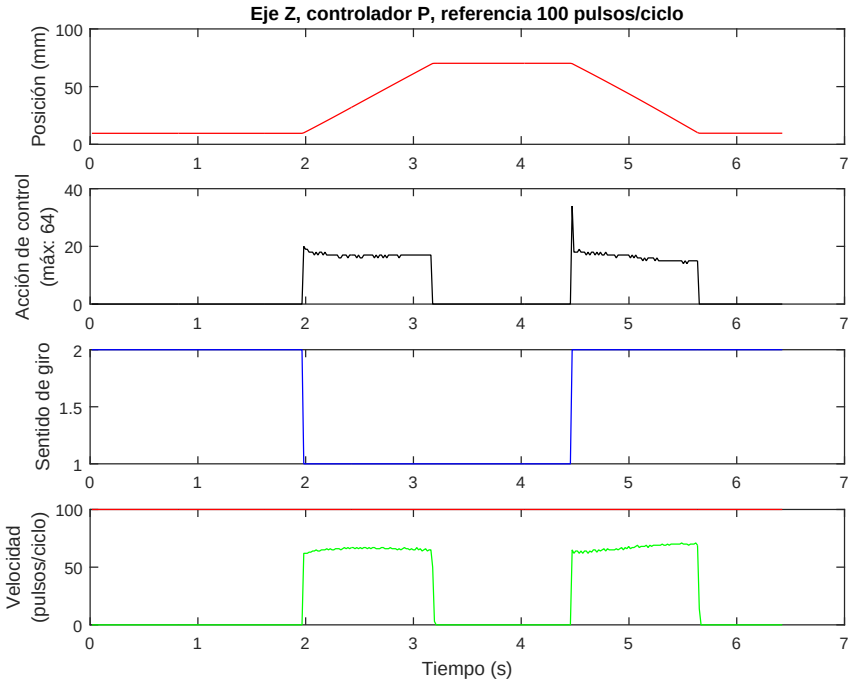


Figura 5.17: Controlador proporcional-integral definitivo para el eje Y con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 100 pulsos/ciclo.

5.3.3. Eje Z

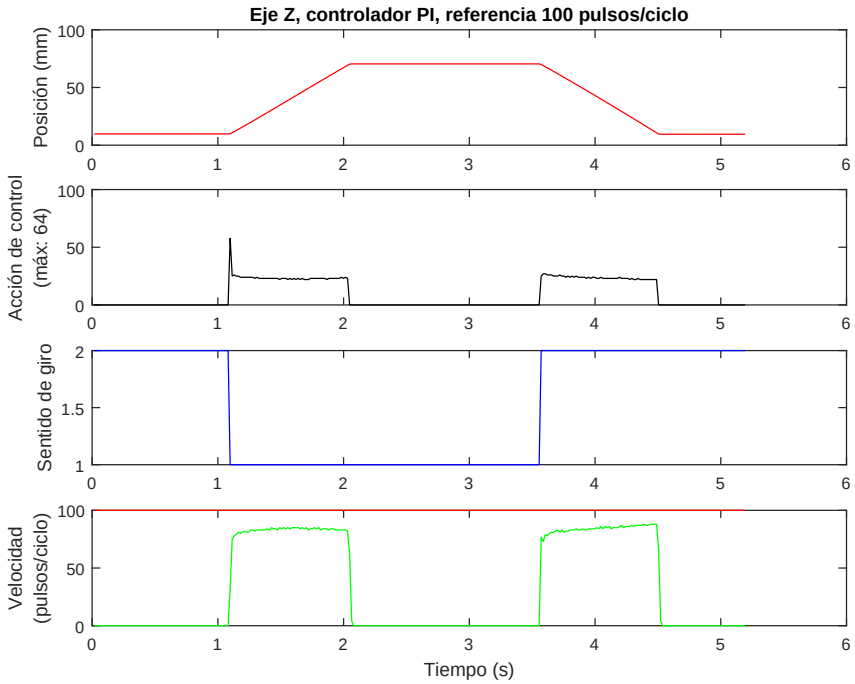


(a) Controlador proporcional con $K_c = 0{,}5$ para una referencia de velocidad de 60 pulsos/ciclo.



(b) Controlador proporcional con $K_c = 0{,}5$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.18: Ajuste del controlador de velocidad para el eje Z (parte 1).



(a) Controlador proporcional-integral con $K_c = 0{,}5$ y $T_i = 0{,}05$ para una referencia de velocidad de 100 pulsos/ciclo.

Figura 5.19: Ajuste del controlador de velocidad para el eje Z (parte 2).

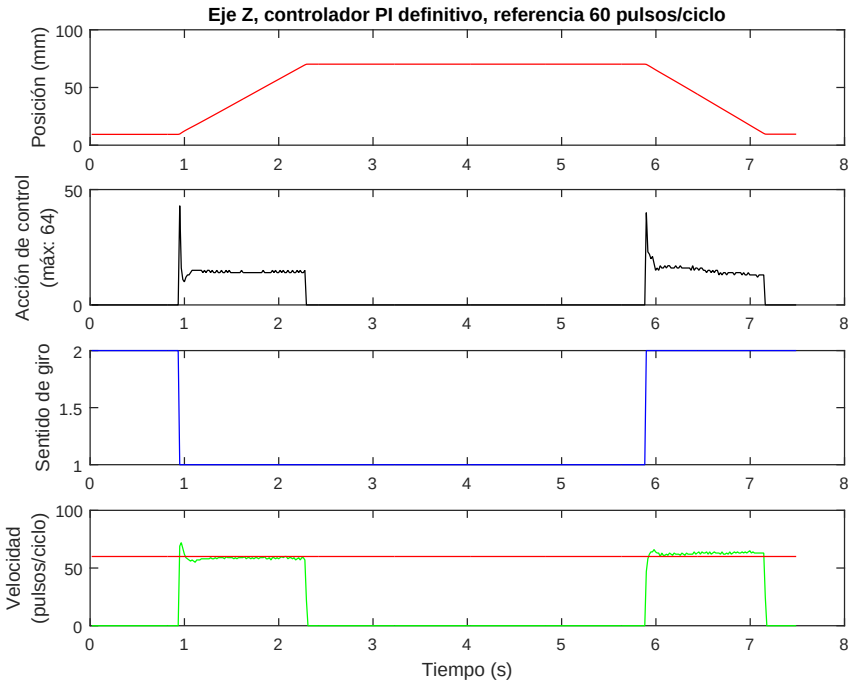


Figura 5.20: Controlador proporcional-integral definitivo para el eje Z con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 60 pulsos/ciclo.

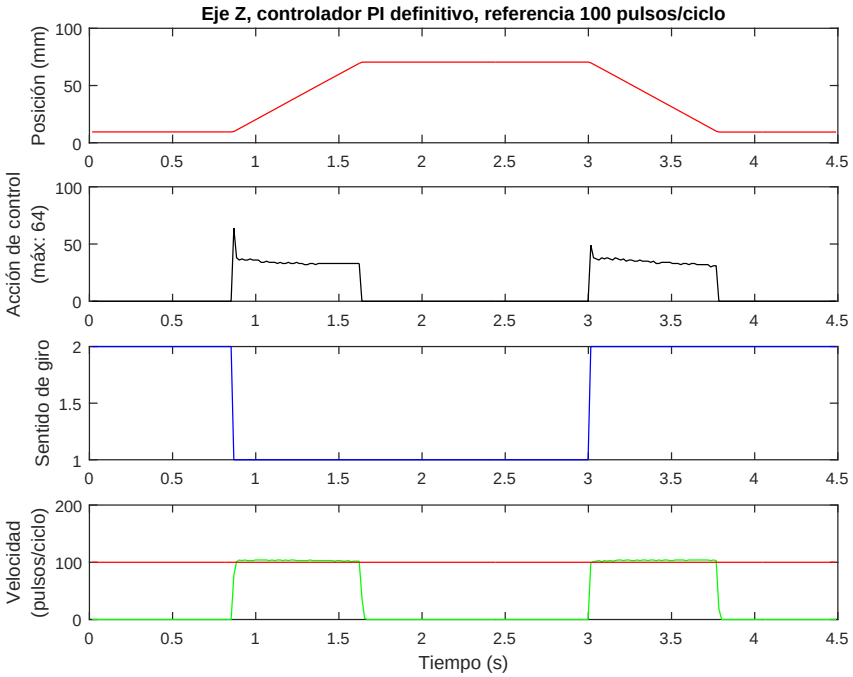


Figura 5.21: Controlador proporcional-integral definitivo para el eje Z con $K_c = 0{,}5$ y $T_i = 0{,}01$ para una referencia de velocidad de 100 pulsos/ciclo.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Trabajo Fin de Grado

Simulación Virtual, Reconfiguración Electrónica y Programación Real del Movimiento del Robot *Hexapod II*

Manual del usuario avanzado

Autor:
Guillermo Oliver Peiró

Tutor:
Dr. Ing. Ind. José Luís Oliver Herrero

Universidad Politécnica de Valencia
Valencia, ESPAÑA

Curso 2014-2015

Parte IV

Manual del usuario avanzado

Índice general

Índice de figuras

Capítulo 1

Introducción

Este es el *manual de usuario avanzado* del robot *Hexapod II*. A lo largo de este documento se detallarán los pasos para la utilización del robot mediante todo el sistema de comunicación implementado utilizando la conexión a través de Internet. El control del robot se puede realizar de varias formas distintas: utilizando el cliente para Windows, utilizando un dispositivo Android, o utilizando un mando a distancia de PlayStation3.

El primer paso para utilizar el robot es realizar todas las conexiones pertinentes para su correcto funcionamiento. Hay que asegurarse que la polaridad sea la correcta en las conexiones de baterías realizadas, estando los cables de color rojo y negro alineados correctamente en las conexiones, para esto se ha intentado realizar el montaje de manera que la configuración de los conectores esté hecha para que sea físicamente muy difícil realizar la conexión de manera incorrecta. Una vez hecho esto se iniciará el servidor del ordenador de a bordo.

Una vez realizadas las conexiones el robot y el servidor funcionando, el robot está preparado para empezar a recibir órdenes para realizar su control telemático. Hay ciertas limitaciones de uso entre la conexión por Internet y la conexión por puerto serie, dejando las opciones avanzadas de desarrollador como el modo monitor y el cambio de parámetros del PID para la conexión por puerto serie, destinada a ser utilizada por desarrolladores con un nivel de conocimientos técnicos superior.

Se han desarrollado programas clientes en distintas plataformas para permitir el control del robot una vez iniciado el servidor en su ordenador de a bordo. Cada uno de estos programas tiene sus peculiaridades y sus opciones de control; siendo el cliente para ordenadores con Windows el más completo y con opciones un poco más avanzadas, y el mando el más sencillo de utilizar.

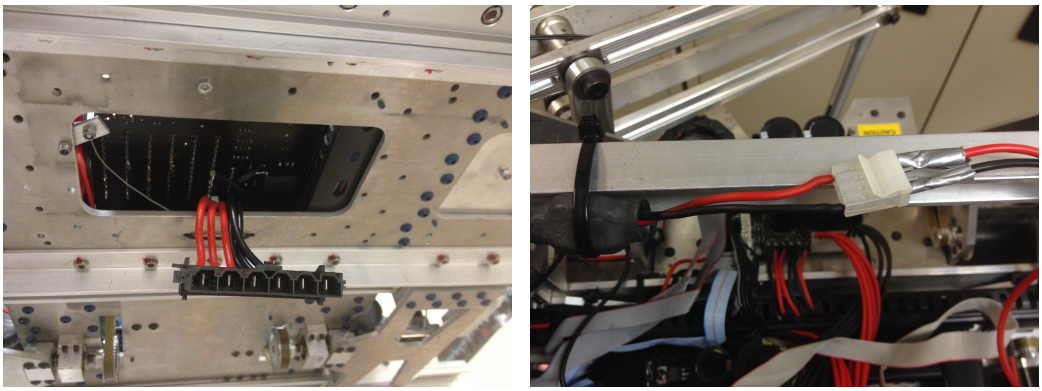
Capítulo 2

Conexiones y puesta a punto

Antes de empezar a usar el robot, se han de realizar las correctas conexiones para alimentar los distintos módulos electrónicos que lo componen. Las baterías de 24V suministrarán la potencia necesaria a los motores de corriente continua, además de alimentar el ordenador de a bordo a través del convertidor de corriente continua con salida USB. La electrónica de control tiene alimentación por separado, además de requerir una conexión por USB de baja potencia. Por lo que las necesidades de potencia son las siguientes:

- Baterías de 24V con alta capacidad y corrientes de pico elevadas para los motores de continua.
- Batería de 14-18V para alimentar la electrónica de control.

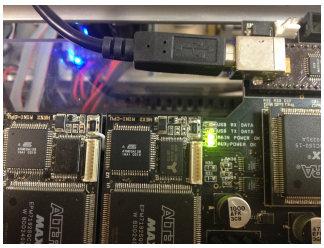
Partiendo del estado inicial que se asume que están todos los conectores desconectados, la primera conexión a realizar es la de las baterías de 24V, hallándose el conector en la parte inferior del robot. Después de esto conectaremos las baterías que alimentan la electrónica de control.



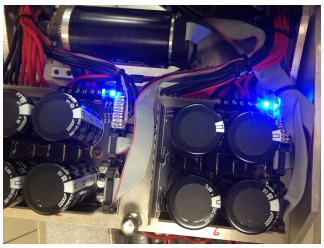
(a) Baterías de 24V para los motores. (b) Baterías para la electrónica de control.

Figura 2.1: Conectores de las baterías del robot.

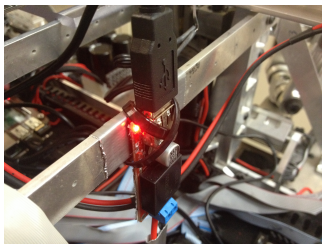
Una vez realizada la conexión de ambas baterías, los LEDs de las etapas de potencia de las patas del robot, el LED del convertidor DC-DC con salida USB, y el ventilador del convertidor DC-DC de la etapa de potencia se encenderán, indicando que la conexión se ha realizado satisfactoriamente. El siguiente paso es conectar el cable de alimentación USB del ordenador de a bordo a la salida USB del convertidor DC-DC.



(a) Electrónica de control.



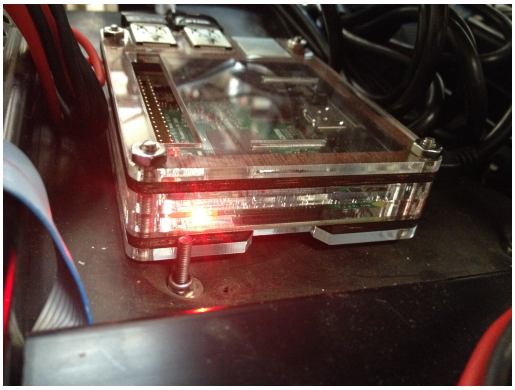
(b) Etapas de las patas.



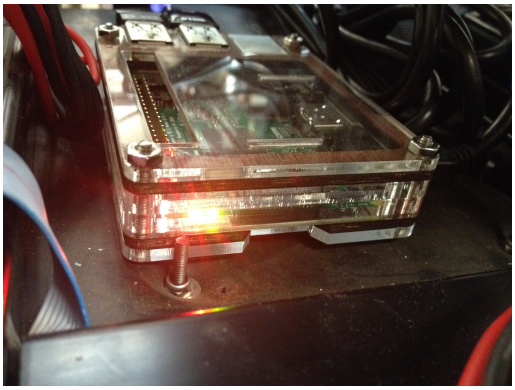
(c) Alimentación USB.

Figura 2.2: Luces de los LEDs del robot.

En cuanto tenga alimentación, se encederá un LED naranja en la placa del ordenador de a bordo. Poco después, el ordenador de a bordo procederá a ejecutar los comandos de inicio, pudiendo observarse un LED amarillo parpadeante que indica que se está realizando el proceso de arranque. Una vez terminado la luz amarilla dejará de parpadear y se quedará la luz roja fija.



(a) Luz roja fija.



(b) Luz amarilla parpadeante.

Figura 2.3: Luces de arranque del ordenador de a bordo.

Una vez finalizado el proceso de arranque, se ejecutarán los *scripts* de inicio, que permitirán al robot conectarse a la red WiFi de la universidad, y una vez conectado procederá al envío de un correo electrónico al usuario con la dirección IP adquirida por el robot para que pueda realizarse la conexión remota. El ordenador de a bordo se conecta a la red WiFi *UPVNET2G* de la Universidad Politécnica de Valencia, pudiendo así conectarse a él cualquier cliente que esté dentro de esta red, sea por conexión física a través de cable, por WiFi dentro de la universidad, o a través de la VPN desde cualquier otro lugar. La dirección IP a utilizar será la que se muestra

en la sección de conexión inalámbrica, *wlan0*, a continuación de la palabra *inet*.

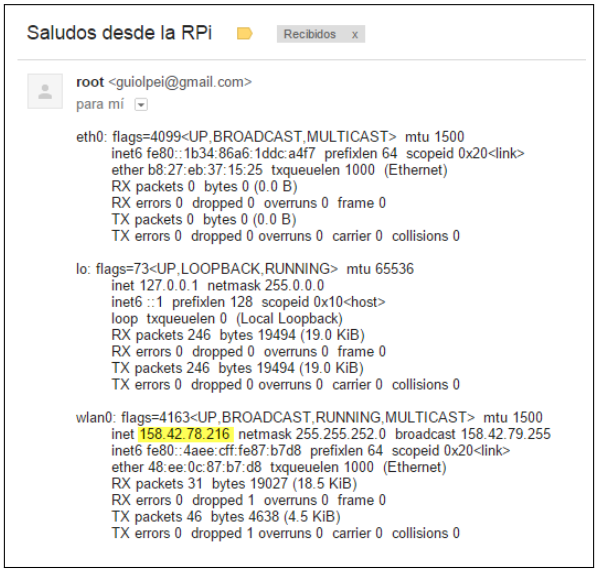


Figura 2.4: Formato del email recibido por parte del ordenador de a bordo con la IP.

Con la dirección IP del ordenador de a bordo, se procederá a realizar la conexión utilizando el protocolo SSH. Para ello puede utilizarse cualquier programa que permita realizar conexiones a través de este protocolo, en las páginas siguientes se utilizará el programa PuTTY[1] por lo que la configuración podría ser ligeramente distinta si se usa otro distinto. Para realizar la conexión simplemente se le indica a PuTTY la dirección IP y el puerto, que por defecto será el puerto 22, y una vez establecida la conexión se solicitará el usuario y la correspondiente contraseña que se quiere utilizar para identificarse en el sistema.

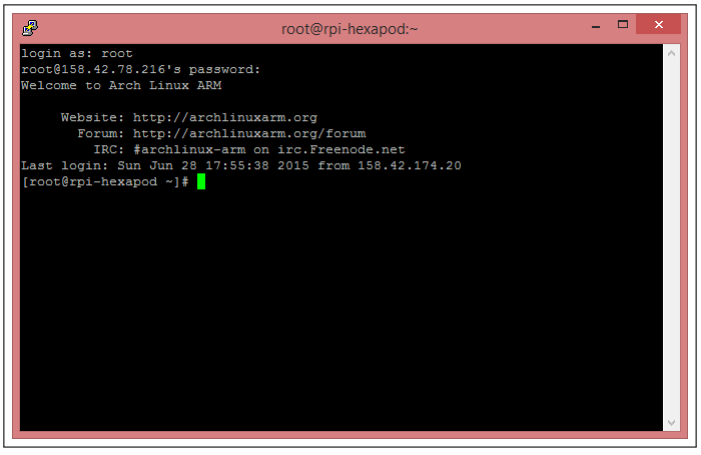


Figura 2.5: Conexión realizada con éxito por SSH.

[1] **PuTTY**, cliente Telnet/SSH gratuito y de código abierto. `http://www.chiark.greenend.org.uk/~sgtatham/putty/`

Capítulo 3

Uso del servidor del ordenador de a bordo

Cuando la conexión por SSH esté correctamente realizada, se tendrá acceso a la línea de comandos del sistema Linux del ordenador de a bordo, teniendo la posibilidad de ejecutar cualquier comando existente y pudiendo acceder al sistema de archivos interno. El siguiente paso a realizar será iniciar el programa servidor, denominado *HexapodRPi*, que está disponible directamente en la carpeta raíz del usuario. La inicialización del servidor requiere de la recepción de unos parámetros por la línea de comandos, mostrándose la pantalla de información si los parámetros recibidos no son los adecuados.

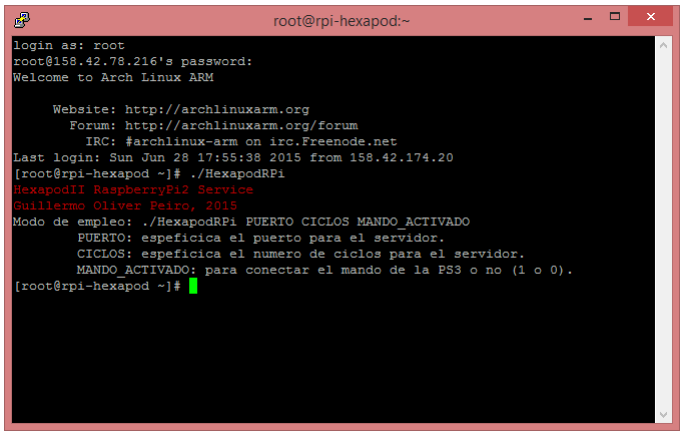


Figura 3.1: Mensaje informativo acerca del modo de empleo.

El primer parámetro a elegir será el puerto TCP por donde se realizará la conexión con el servidor, que ha de estar comprendido entre el 1 y el 65535. Se recomienda utilizar el puerto 1478, puesto que será el puerto por defecto en la configuración de los programas cliente. Lo siguiente que se tendrá que especificar es el número de ciclos que ha de realizar el programa, entendiéndose como realizado un ciclo cuando se completa un proceso de conexión-desconexión del cliente o tras una espera de conexión mayor de un tiempo establecido. El último parámetro necesario será si ha

de inicializarse el mando o no, introduciendo el valor 1 si desea realizarse el control utilizando el mando de PlayStation3 y el valor 0 en caso contrario.

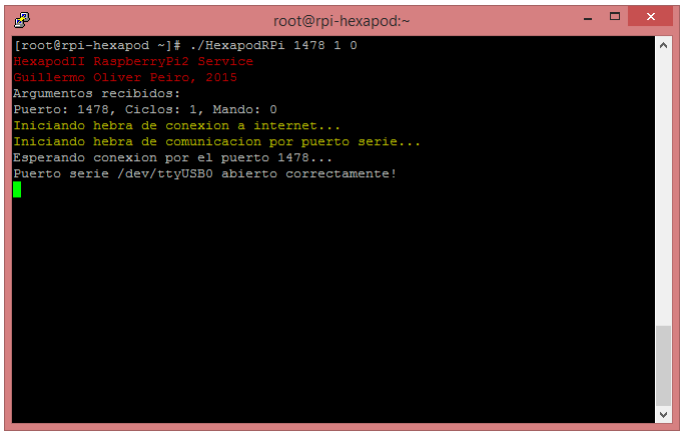


Figura 3.2: Funcionamiento esperando conexión.

Una vez iniciado el servidor, éste se pondrá a la espera de recibir una petición de conexión por Internet debidamente formulada desde el programa cliente. Cabe decir que si no se realiza esta petición después de un tiempo establecido, que por defecto toma el valor de veinte segundos, se dará por terminado el ciclo de conexión. Una vez realizados todos los ciclos del parámetro que se le ha introducido al servidor, se terminará su ejecución, siendo necesario ejecutar de nuevo el proceso desde línea de comandos.

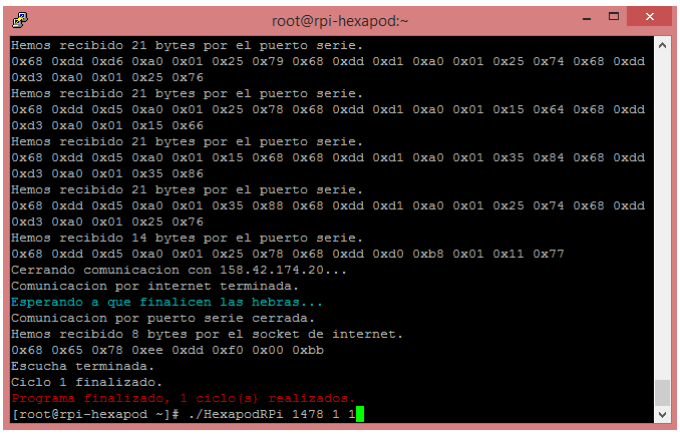


Figura 3.3: Fin de ejecución del programa servidor tras realizar los ciclos seleccionados.

Capítulo 4

Uso del cliente en ordenadores con Windows

El cliente para ordenadores con sistema operativo Windows es el más completo de los clientes desarrollados para el control del robot, teniendo disponibles todas las opciones avanzadas y mostrando por pantalla el registro de todas las órdenes enviadas y recibidas durante el período de control actual. Una vez seleccionado el tipo de comunicación a *Internet*, aparecerá la pantalla principal de control.

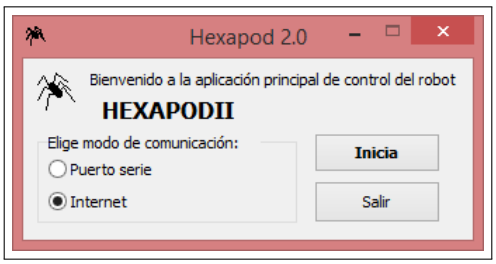


Figura 4.1: Selección del tipo de conexión.

La pantalla principal de control tiene tres cuadros de texto donde se mostrarán los mensajes recibidos desde el robot. En uno de los cuadros aparecerán los que provengan del micro principal y otros mensajes de índole general, en otro cuadro aparecerán los mensajes recibidos de las patas, y, por último, en el tercer cuadro aparecerán los datos que lleguen por el socket de Internet en codificación hexadecimal.

Lo primero que se ha de realizar es la conexión con el servidor, para ello bastará con introducir en el recuadro de *Conexión Internet* la IP recibida en el correo y el puerto seleccionado, así como la contraseña de autentificación. Una vez se pulse el botón de *Conecta!*, si los datos introducidos son correctos, en el recuadro principal de mensajes recibidos aparecerá la notificación correspondiente informando sobre el suceso.

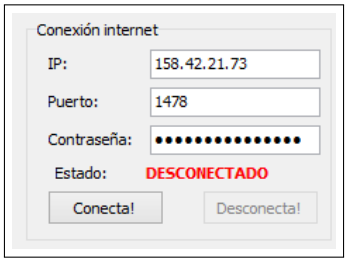


Figura 4.2: Recuadro de conexión a Internet.

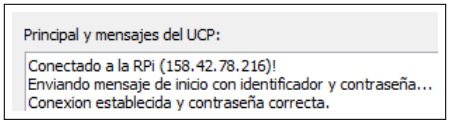


Figura 4.3: Conexión establecida correctamente con el servidor.

El control se realiza desde la zona establecida para ello, que se separa en varias zonas distintas. La primera zona es la de los comandos de ámbito general para el micro principal, donde se pueden encontrar los siguientes botones:

- *Ping UCP*: Enviará un comando de PING al micro principal, al que se responderá con un PONG, que aparecerá en el cuadro de mensajes principales.
- *Estado Robot*: Abrirá una nueva ventana donde se muestra el estado de los distintos parámetros del robot.
- *Reset UCP*: Provocará un reset del micro principal, volviendo a iniciarse el programa que éste contiene.
- *Test RAM UCP*: Enviará la orden de comprobar el estado de la RAM externa del micro principal, recibiendo la respuesta con el resultado del proceso de comprobación.

Desde aquí también se realiza el control de la etapa de potencia, pudiendo encenderla y apagarla. Para iniciar la potencia, que permitirá el suministro de potencia a los motores de las patas habrá que pulsar el botón *Enciende*. Una vez hecho esto se podrá comprobar el estado de la potencia observando el valor de la tensión de 34V en la ventana de estado del robot, un nivel alto corresponde con la etapa de potencia iniciada. Hacer clic sobre el botón *Apaga* desconectará la etapa de potencia, cortando el suministro de potencia a los motores. También se encuentra el botón de *Limpia*, que borrará los tres cuadros de texto donde se muestran los mensajes y los datos de la comunicación.

Para realizar el control se dispone de una selección de modo para elegir entre el panel de control del robot y el panel de control independiente de las patas. Según se seleccione una pestaña u otra, se procederá, bien a controlar el robot en su totalidad, pudiendo efectuar movimientos preprogramados y cambiar parámetros generales, o

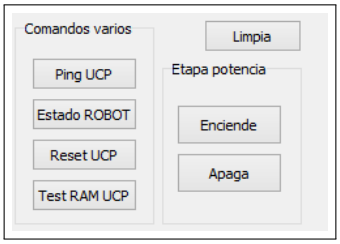


Figura 4.4: Comandos generales para el robot.

bien a controlar las patas por separado, donde se podrá mover cada una de las patas de manera independiente y visualizar otros datos acerca de su funcionamiento. Cada uno de los comandos que se envíen provocarán una respuesta confirmando su recepción, y, si procede, una respuesta confirmando la correcta ejecucción del comando. Cabe destacar que antes de realizar cualquier movimiento con el robot en su conjunto es necesario hacer un reset del robot, de modo que se puedan poner todas las patas a cero, y lo mismo sucede si se quiere mover una pata de manera independiente, de lo contrario la respuesta será de error, informando acerca de la necesidad de realizar el *reset* correspondiente.

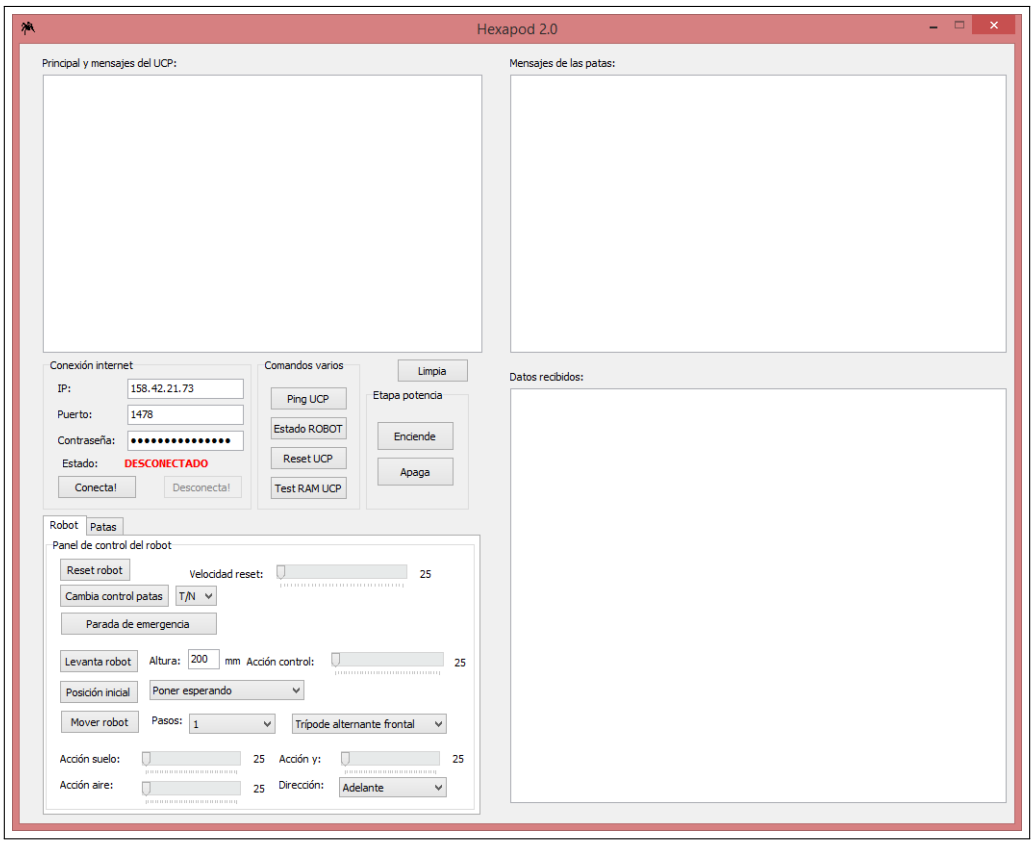


Figura 4.5: Vista general de la pantalla principal.

4.1. Panel de control del robot

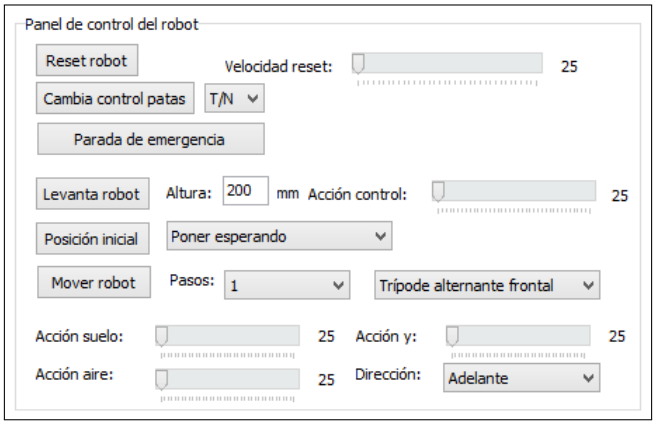


Figura 4.6: Panel de control del robot.

El *panel de control del robot* tiene una serie de botones disponibles, según se pulse uno u otro se enviarán distintos comandos. Además las barras para la selección de las de acciones de control cambiarán por otras barras de selección de velocidad si se cambia al modo controlador PID. Los botones y las distintas acciones de cada uno de ellos se detallan a continuación:

- *Reset robot*: Según la velocidad seleccionada de la barra *Velocidad reset*, se enviará un mensaje para realizar el reset del robot completo. El reset se hace mediante el control todo/nada, por lo que este comando no sufrirá modificaciones en su configuración cuando se cambie de tipo de control. Siempre que se vuelva a iniciar el programa del micro principal será necesario realizar el reset antes de enviar una orden de movimiento al robot.

- *Cambia control patas*: Desde aquí se podrá elegir el tipo de control que se desee según el menú desplegable situado a la derecha del botón. Este botón envía el comando de cambiar el tipo de control en todas las patas, teniendo constancia también el micro principal del tipo de control seleccionado. Cambiar del control todo/nada (T/N) al de controlador tipo PID (PID) provocará la actualización de las barras de selección de acción de control por las de selección de velocidad de movimiento en el panel de control del robot.

- *Parada de emergencia*: Hacer clic sobre este botón enviará un mensaje de alta prioridad que parará cualquier proceso que esté ejecutando el micro principal provocando el reinicio de éste y de todos los micros de las patas. Tras enviar este comando habrá que volver a realizar el reset del robot para volver a poder hacer cualquier movimiento.

- *Levanta robot*: Enviará el comando para realizar la modificación de la distancia al suelo del robot, pudiendo elegir la altura y la acción de control o la velocidad a la que se ha de realizar el movimiento. Los límites introducidos en el diseño de la función de movimiento son un mínimo de 80 mm y un máximo de 320 mm.

- *Posición inicial*: Eligiendo del menú desplegable la posición deseada y utilizando al misma barra de selección de acción de control o velocidad de movimiento que para *Levanta robot*, al pulsar este botón se enviará el comando pertinente para llevar el robot a una posición inicial determinada.

- *Mover robot*: El caminar del robot se controlará desde esta sección ubicada en la parte inferior del panel de control. Desde aquí se seleccionará la cantidad de pasos a realizar, la dirección de movimiento, y las acciones de control o velocidades del movimiento. Se pueden configurar las acciones de control o velocidades de tres movimientos distintos: el movimiento de las patas que estén tocando el suelo para avanzar (*suelo*), el movimiento de las patas que están avanzando en el aire (*aire*), y el movimiento de las patas al subir y bajar (*y*).

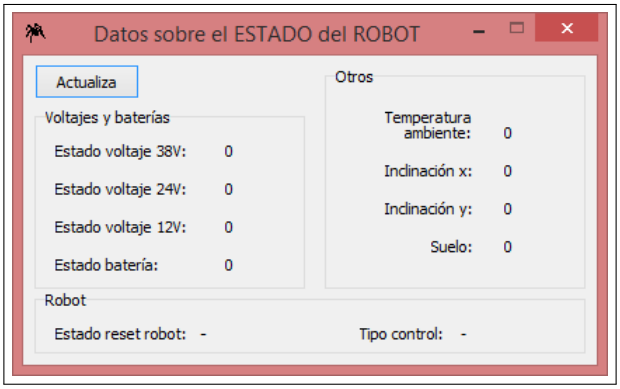


Figura 4.7: Información del estado del robot.

La pantalla que aparece al pulsar el botón *Estado Robot* muestra datos acerca del funcionamiento interno del robot. Para actualizar los datos bastará con hacer clic sobre el botón *Actualiza*. Aquí se muestran las tensiones de la alimentación principal de 24V, de la tensión de 12V proveniente de la principal, y de la tensión de 34V que está disponible cuando se activa la etapa de potencia. También se podrá consultar en el futuro el estado de la batería de litio que alimenta la electrónica de control, así como otros parámetros que se implementarán más adelante. Cobra gran importancia el recuadro inferior, que sirve para mostrar si el robot ha sido reseteado en su conjunto y el tipo de control de movimiento actual del robot entero, todo/nada o PID.

4.2. Panel de control de las patas

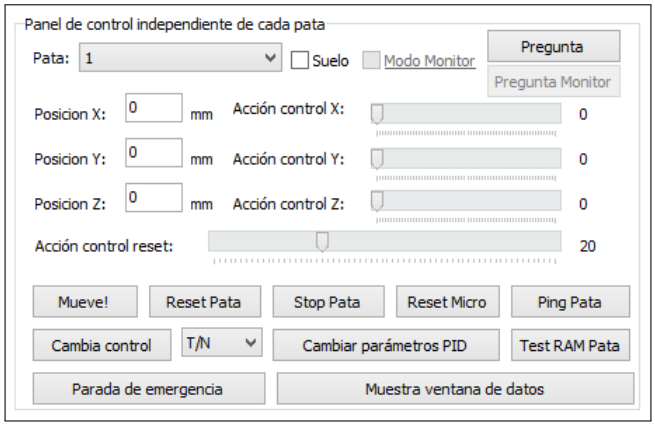


Figura 4.8: Panel de control de las patas.

Desde el *panel de control independiente de cada pata* podrán realizarse movimientos de las patas bajo demanda eligiendo previamente la pata a controlar desde el menú desplegable. Como se puede observar, el *modo monitor* está desactivado para su uso en la conexión a través de Internet. El funcionamiento de cada uno de los botones se detalla a continuación:

- *Mueve!*: Eligiendo previamente la posición a la que se quiere mover en cada uno de los tres ejes y la acción de control o velocidad de movimiento correspondiente para cada uno de ellos, pulsar este botón enviará el comando de movimiento a la pata seleccionada. Además utilizando la opción de suelo se podrá elegir si se ha de seguir el movimiento al tocar el pie de la pata el nivel del suelo si está marcada la opción, o se ha de parar al tocar, si está desmarcada. Para que no se produzca el movimiento en un eje, bastará con indicar una acción de control o velocidad de movimiento cero en la barra de selección. Antes de realizar cualquier movimiento es necesario que la pata seleccionada esté reseteada.

- *Reset Pata*: Enviará el comando de hacer el reset de la pata seleccionada, siempre que se reinicie el programa del micro será necesario hacer el reset, o puesta a cero, de la pata. El reset finalizará cuando la pata toque el suelo, activándose el sensor de suelo situado al final del recorrido del tramo telescópico del pie de la pata.

- *Stop Pata*: Pulsar este botón provocará el envío de un mensaje de alta prioridad que parará cualquier movimiento que esté realizando la pata, quedándose en la situación actual a la espera de un nuevo comando.

- *Reset Micro*: Este botón enviará la orden para reiniciar el micro de la pata, volviendo a tener que realizarse el reset de la pata cuando se quiera mover de nuevo.

- *Ping Pata*: Enviará un comando de PING para comprobar si la comunicación con el micro de la pata está abierta correctamente, la respuesta será un comando de PONG que aparecerá en el cuadro de texto reservado para los mensajes de las patas.
- *Cambia control*: Previamente seleccionando el tipo de control del menú desplegable, pulsando este botón se enviará la orden de cambiar el control de movimiento en el micro de la pata, bien sea a todo/nada o a tipo PID. Cambiar el tipo de control, como en el caso del panel de control del robot, provocará la actualización de las barras de selección de acción a selección de velocidad si se cambia el todo/nada por el PID, y viceversa.
- *Cambiar parámetros PID*: Pulsar este botón abrirá una nueva ventana donde se podrán consultar los valores del controlador implementado en cada una de las patas, pero no pudiendo cambiar sus parámetros al estar haciéndose la conexión a través de Internet.
- *Test RAM Pata*: Enviará la orden de comprobar el estado de la RAM externa del micro de la pata, recibiendo la respuesta con el resultado del proceso de comprobación.
- *Parada de emergencia*: Pulsar este botón provocará el envío de un mensaje de alta prioridad que parará cualquier movimiento que esté realizando la pata, realizándose además un reinicio del programa del micro. Tras realizar esta acción habrá que enviar el comando de reset pata de nuevo.
- *Muestra ventana de datos*: Abrirá una nueva ventana donde se podrán solicitar distintos datos internos acerca del movimiento de la pata y donde se mostrarán por pantalla debidamente organizados.
- *Pregunta*: Enviará la solicitud para recibir todos los mensajes que se han ido produciendo y almacenando en la pila de salida del micro de la pata. Estos mensajes contienen información acerca del funcionamiento interno, conteniendo lo avisos de la necesidad de realizar el reset antes de solicitar un movimiento, los datos solicitados de los parámetros del controlador PID, etc.

La ventana de *datos sobre las patas* permite solicitar información acerca de los parámetros internos de movimiento de la pata seleccionada. Pulsando sobre el texto que reza *Click para solicitar info* se puede solicitar los datos correspondientes al estado de los sensores, las lecturas de los encoders, y el estado de posición de la pata elegida. Dentro del apartado de *Sensores* se muestra el estado de los finales de carrera de los tres ejes, así como el sensor de suelo, estando éstos a 1 al estar pulsados o a 0 si no lo están. En la sección de *Encoders* se muestran los datos de realimentación que devuelven los encoders acerca del funcionamiento de los motores, destacando *avance pulsos* que mide la velocidad de movimiento en pulsos por ciclo, *pulsos reales* que muestra la cuenta total de pulsos tomando como origen el final de carrera, *sentido encoder* que muestra hacia qué lado está girando el motor, y *motor moviéndose* que indica si el motor está en movimiento en ese preciso instante. Por último, también se pueden consultar *datos internos* acerca de las acciones de movimiento de la pata,

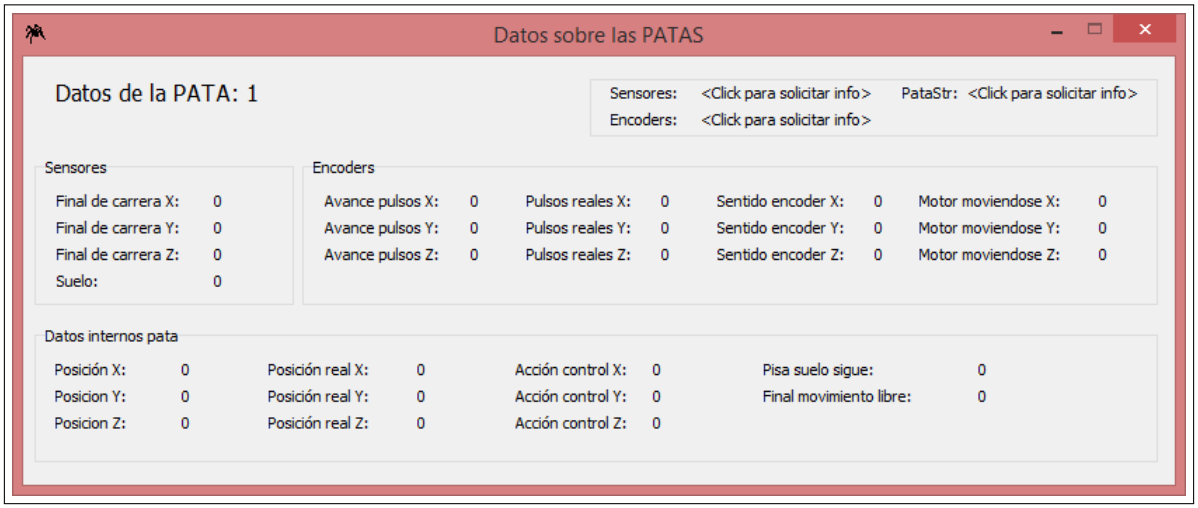


Figura 4.9: Datos internos de la pata.

entre los que se encuentran: la posición destino que se le he fijado, la posición real en la que se encuentra la pata, la última acción de control aplicada, si ha de seguir el movimiento del eje Y al pisar suelo, y si el motor queda libre al final del movimiento.

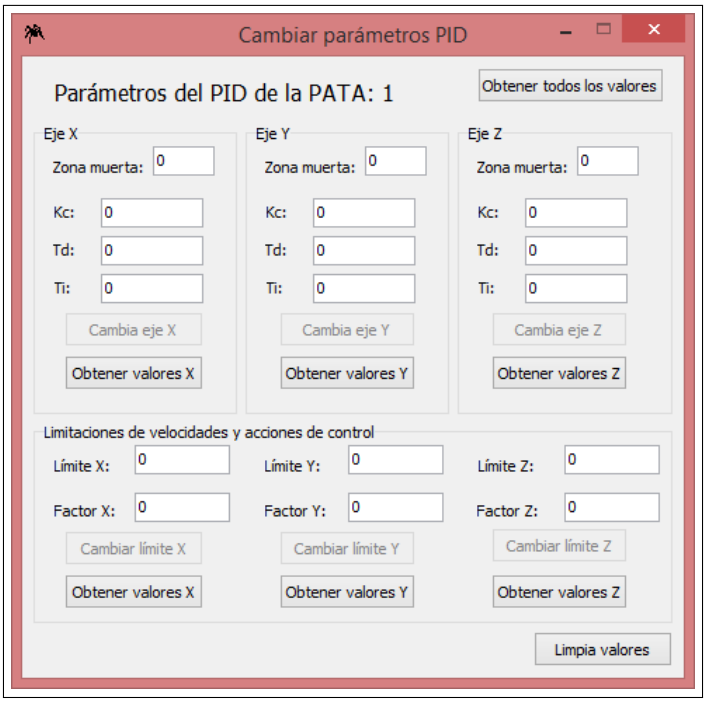


Figura 4.10: Parámetros del control PID de la pata.

La ventana de *parámetros del PID* da información acerca del controlador tipo PID para la velocidad que tiene integrado cada uno de los micros de las patas, que por defecto se trata de un PI. Desde la conexión por Internet se pueden modificar los parámetros del controlador PID, puesto que los botones de cambiar valores están

desactivados, pero si que se pueden consultar los valores actuales que tiene registrados el micro. Pulsando los botones de *obtener valores* se pueden solicitar los valores para cada eje en concreto o directamente para todos. Una vez pulsado uno de esos botones habrá que preguntar a la pata por los mensajes almacenados en la pila de salida para obtener la respuesta. Para cada uno de los ejes se muestran los siguientes valores:

- K_c: Valor del parámetro de la ganancia proporcional. El valor por defecto es 0,3 para el eje X y 0,5 para los ejes Y y Z.
- T_d: Valor del tiempo derivativo en segundos. El valor por defecto es 0 para los tres ejes.
- T_i: Valor del tiempo integral en segundos. El valor por defecto es 0,01 para los tres ejes.
- *Zona muerta*: Valor de compensación de la zona muerta.
- *Límite*: Límite para la acción de control.
- *Factor de escala*: Factor de escala para multiplicar las acciones de control resultantes de aplicar el controlador PID.

Los valores de zona muerta, límite de acción de control, y del factor de escala fueron utilizados durante las primeras pruebas con el control tipo PID de posición, y en la versión actual no son de aplicación para el control de velocidad.

Capítulo 5

Uso del cliente en dispositivos Android

El programa cliente para dispositivos Android está dividido en varias pantallas o actividades, siendo necesario salir de una para poder pasar a la siguiente. El control del robot está más limitado que en el caso del cliente para Windows, pero se siguen mostrando algunos mensajes de confirmación de recepción y ejecución de tareas. Todas las pantallas tienen un botón de parada de emergencia que provoca la parada del movimiento inmediato en el caso de estar moviendo las patas independientemente o la parada absoluta del robot y el reinicio de los micros en el caso de estar en las pantallas de movimiento del robot en su conjunto.

Cuando se inicia el programa se parte de la pantalla *Principal*, que es la presentación a la aplicación. En esta pantalla el usuario ha de introducir la dirección IP de la máquina donde está el servidor a la escucha (el puerto asignado por defecto es el 1478) y la contraseña para acceder a él. Si se pulsa el botón de conectar, se procede al envío de un mensaje al servidor que contiene el comando de inicio de comunicación, la identificación del tipo de cliente, y la contraseña. Una vez comprobada la contraseña en la parte del servidor, la identificación se da por válida y se establece la conexión. Si hay un error en la comunicación o la contraseña es incorrecta, se le comunica al usuario para que lo corrija.

Cuando está la comunicación establecida, se procede a la siguiente pantalla, denominada *Modo*, aquí el usuario puede elegir entre los distintos modos de operación que tiene el robot: acceso a las funciones del sistema, mover las patas independientemente, o empezar un movimiento de caminar por pasos en una dirección concreta. Pulsar cada uno de esos botones conducirá a la aparición de una nueva pantalla, y también se le enviará un mensaje al servidor informando acerca de la elección de modo de control que se está haciendo. Por lo contrario, se mandará el comando de desconexión en caso de pulsar el botón de desconectar o el botón de volver integrado en la aplicación.

(a) Pantalla principal. (b) Pantalla de selección de modo.

Figura 5.1: Primeras pantallas del cliente Android.

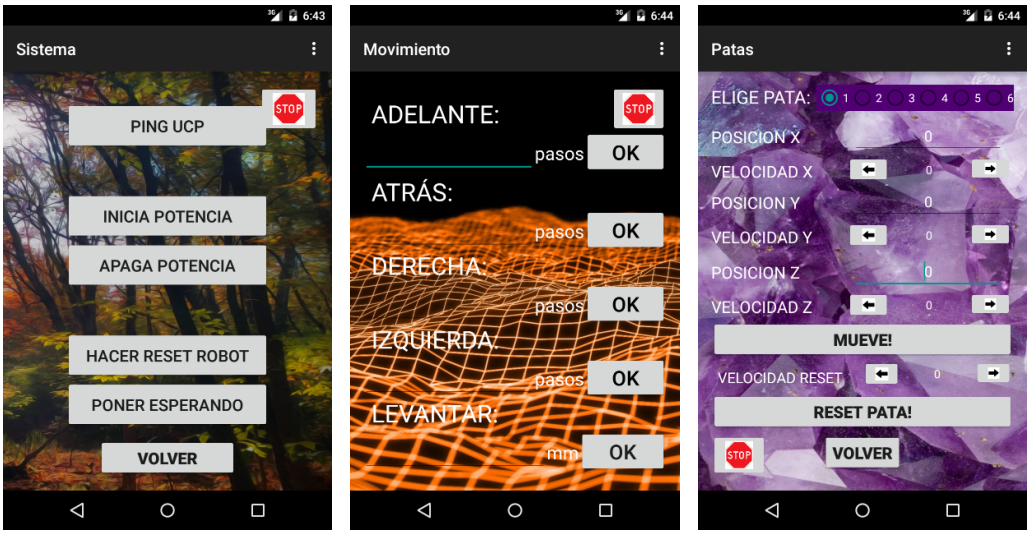
En la pantalla de *Sistema* se pueden mandar una serie de ordenes básicas al micro principal, así como realizar algunos movimientos especiales. Se puede enviar el comando del PING al micro principal, de manera que se pueda comprobar si la comunicación con éste se está realizando correctamente. También se tiene control sobre la etapa de potencia, pudiendo activar o desactivar el suministro de potencia a los motores de las patas pulsando los botones colocados para ello. El reset o puesta a cero del robot se realiza también desde esta pantalla, una vez se haya iniciado la potencia, así como se puede llevar el robot a la posición de espera.

La siguiente opción del menú inicia la pantalla *Movimiento* permite el movimiento del robot de manera más o menos parametrizada, de manera que elegimos una dirección y una cantidad de pasos, y se enviará el comando pertinente al robot. También nos permitirá mandar el comando de llevar el robot a una altura determinada en milímetros, que será elegida usando el recuadro de introducción de texto. Existen una serie de restricciones para este movimiento parametrizado: máximo 5 pasos y una altura entre 80 y 320 mm, y los recuadros de los parámetros están restringidos a valores numéricos. La velocidad de movimiento está definida según se esté utilizando el control todo/nada o PID desde el programa del propio micro.

La última opción del menú de modo es el movimiento independiente de cada una de las patas que está implementado en la pantalla *Patas*. En esta pantalla se da la opción para manejar de manera independiente cada una de las patas, se elige la

pata deseada y sus valores de posición y velocidad en los tres ejes de movimiento que posee. Una vez elegido, si se pulsa el botón se mandarán los datos en un mensaje al servidor, recibiéndose un mensaje de confirmación de recepción. Esta vez se ha usado un cuadro de texto numérico para introducir los datos de posición y unos botones para incrementar y disminuir el valor para introducir los datos de velocidad. Los datos de posición y de velocidad están limitados a los límites de movimiento físicos de cada uno de los ejes.

Cuando el usuario decida salir de una actividad, bien mediante el botón de volver integrado en Android o el correspondiente a la actividad, se enviará un mensaje informativo al servidor para que finalice el modo que había sido seleccionado. De este modo se finalizará todo movimiento que pudiera estar realizando y pasará a hallarse de nuevo a la espera del nuevo comando. Por regla general, en todos los casos que exista comunicación, se informa al usuario mediante un pequeño bocadillo con un mensaje de texto claro y conciso que aparece en la parte inferior de la pantalla, ya sea confirmando que se ha trasmitido el comando de manera satisfactoria o informando acerca de un error de transmisión.



(a) Funciones de sistema del robot.

(b) Control de movimiento del robot.

(c) Movimiento independiente de las patas.

Figura 5.2: Modos de operación del cliente Android.

Capítulo 6

Uso del mando de PlayStation3

La utilización del mando de PlayStation3 para controlar el robot permite realizar el movimiento de la manera más sencilla posible. Se han implementado distintas acciones para cada uno de los botones del mando, estando fijas las velocidades de movimiento a unos valores que se han comprobado realizan satisfactoriamente los movimientos según el control elegido.

Para poder utilizar el mando es necesario vincularlo a través de la línea de comandos del ordenador de a bordo, utilizando para esto la utilidad *sixad*, instalada en el sistema operativo de éste y que está especialmente diseñada para su uso con mandos de PlayStation3. Esto se realiza a través de los siguientes comandos: *hciconfig hci0 up*, para activar el dispositivo adaptador Bluetooth USB conectado; *hciconfig hci0 piscan*, para hacer el dispositivo visible y permitir conexiones; y, por último, *sixad –start*, para vincular el mando con el ordenador y poder recibir datos de él.

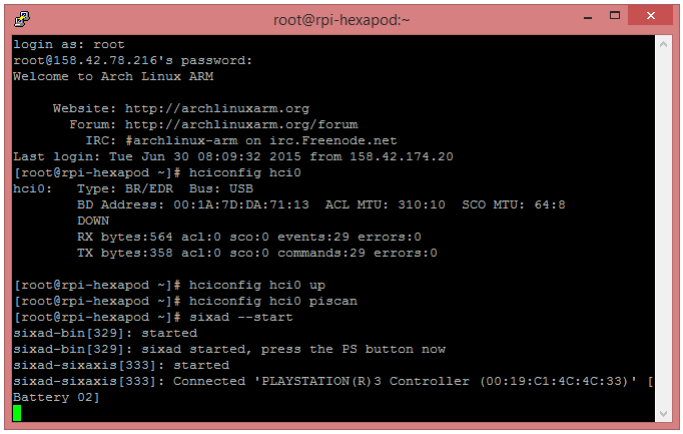


Figura 6.1: Configuración del mando por Bluetooth.

Es necesario utilizar el cliente para ordenadores con Windows como supervisor de la conexión y el movimiento realizado con el mando. Esto es una medida de

seguridad, puesto que el mando no tiene la opción de mostrar ningún mensaje de respuesta, ha de ser el cliente para PC el que vaya mostrando los mensajes enviados por el robot y comprobar su correcto funcionamiento. El usuario ha de iniciar el servidor con la opción de utilización de mando activada.

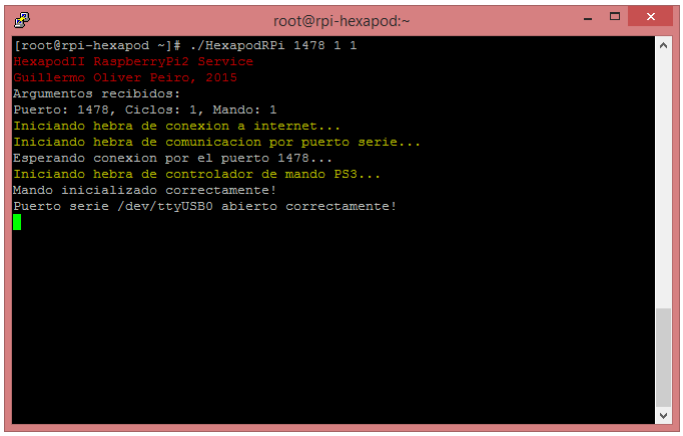


Figura 6.2: Funcionamiento del servidor con el mando iniciado.

Una vez vinculado y el servidor debidamente iniciado, los comandos disponibles se detallan a continuación:

- *Inicia potencia*: Activa la alimentación para los motores de las patas.
- *Reset robot*: Realiza el reset o puesta a cero de todas las patas del robot.
- *Control PID*: Cambia el control de todas las patas a tipo PID.
- *Control todo/nada*: Cambia el control de todas las patas a todo/nada.
- *Posición esperando*: Lleva el robot a posición inicial esperando.
- *Bajar robot*: Baja el robot a una altura de 100mm del suelo.
- *Levantar robot*: Sube el robot a una altura de 200mm del suelo.
- *Posición inicial frontal*: Lleva el robot a posición inicial para realizar el movimiento de trípode alternante frontal.
- *Paso adelante*: Avanza un paso hacia adelante en modo de movimiento trípode alternante frontal.
- *Paso atrás*: Avanza un paso hacia atrás en modo de movimiento trípode alternante frontal.
- *Posición inicial lateral*: Lleva el robot a posición inicial para realizar el movimiento de trípode alternante lateral.
- *Paso derecha*: Avanza un paso hacia la derecha en modo de movimiento trípode alternante lateral.

- *Paso izquierda*: Avanza un paso hacia la izquierda en modo de movimiento trípode alternante lateral.
- *Apaga potencia*: Desactiva la alimentación para los motores de las patas.
- *Terminar conexión*: Cierra la comunicación con el servidor.
- *Parada de emergencia*: Para el movimiento del robot y realiza el reinicio del programa de los micros.

Antes de realizar el movimiento de trípode alternante frontal o lateral, es necesario llevar el robot a la posición inicial correspondiente. Los movimientos siguen una secuencia cíclica, iniciando y terminando el movimiento en la misma posición, que se corresponde con la posición inicial del trípode alternante correspondiente.

Figura 6.3: Controles implementados en el mando.