Universidad Nacional de Educación a Distancia
Departamento de Informática y Automática
E.T.S. Ingeniería Informática

PLATAFORMAS INTERACTIVAS DE EXPERIMENTACIÓN VIRTUAL Y REMOTA: APLICACIONES DE CONTROL Y ROBÓTICA

Tesis Doctoral

M.Sc. Ernesto Fabregas Acosta
Ingeniero en Automática

Madrid, Abril 2013

Departamento Informática y Automática
E.T.S. de Ingeniería Informática

Título Plataformas Interactivas de Experimentación Virtual y Remota: Aplicaciones de Control y Robótica

Autor Ernesto Fabregas Acosta

Titulación Master en Sistemas Digitales
Ingeniero en Automática
Instituto Superior Politécnico
José A. Echeverría (CUJAE)
La Habana, Cuba

Directores Sebastián Dormido-Canto
Gonzalo Farias Castro

A mi familia...

Agradecimientos

En primer lugar, quisiera agradecer especialmente al profesor Sebastián Dormido Bencomo por sus consejos y apoyo. Durante todo el tiempo que ha durado esta investigación me ha dado la posibilidad de aprender muchas cosas que difícilmente se olvidan.

A mis supervisores Sebastián Dormido Canto y Gonzalo Farias Castro por su ayuda y su paciencia durante el desarrollo de esta investigación. Por las incontables horas que dedicaron a la revisión de este documento. Por sus siempre apreciables consejos y por ser tan "tiquismiquis" con el formato de este manuscrito.

También al Ministerio de Ciencia e Innovación de España por concederme la Ayuda Predoctoral de Formación de Personal Investigador BES-2008-004619 asociada al proyecto de investigación DPI2007-61068. Además por darme la posibilidad de realizar dos estancias de tres meses cada una en la Universidad de Gante en Bélgica.

Quiero agradecer muy especialmente a la secretaria de nuestro Departamento Pilar Riego, por su cariño y su total disposición a ayudar en todo momento, tanto en lo profesional como en lo personal.

A mis compañeros del Departamento de Informática y Automática de la UNED por su paciencia durante todo el tiempo que hemos pasado juntos: Alejandro Moreno, David Moreno, Luis de la Torre, Víctor Sanz, Dictino Chaos, Miguel Ángel Rubio, María Guinaldo. Y muy especialmente a Jesús Chacón "*jesschacn*", Héctor Vargas "*hvargas*" y Gonzalo Farias "*gfarias*" por saber ser, además de mis compañeros de trabajo, mis amigos.

A los profesores del Departamento de Informática y Automática por acogerme con tanto cariño: Natividad Duro, Raquel Dormido, José Manuel Díaz, José Luis Fernández, María Antonia Canto, Fernando Morilla, Sebastián Dromido Bencomo, José Sánchez y Sebastián Dormido-Canto.

Al Departamento EeSA (Electrical energy, Systems and Automation) de la Universidad de Gante, por permitirme realizar en sus instalaciones dos estancias de tres meses cada una.

A todos y cada uno de mis familiares, que aunque en la distancia siempre han estado pendientes de mí. Y por supuesto con especial mención a mis padres que han sabido guiarme siempre por el camino correcto, logrando hacer de mí todo lo que soy hoy.

A mi compañera y amiga Puri, por su comprensión y paciencia. TQM :-)

A su familia (Pura, Pedro, Paloma, Miguel Ángel y Miguel) por saber sustituir a la mía en todo momento.

Al Daniel "el Goro" y Naylín, por su incondicional ayuda desde el mismo primer día en que llegué a este país, esas cosas no se olvidan.

A mis amigos Yisnier, Juan David, del Toro, el Sionell "Bauer", Yohanna, Addel, Gaspar, Edwin "el yuma", Yulexys, Rafael "el wasa", Hugo "la berruga" Boss, Jorge Antonio Amador y Héctor "ShereKan" por ser mis amigos incondicionales y saber estar siempre ahí a pesar de la distancia y de los problemas con las comunicaciones... ;-)

Índice de Contenido

Lista de Símbolos y Abreviaturas

Símbolo	UM	Descripción	Página
$\tau(t)$	[Nm]	Torque del servomotor del sistema bola y aro	22
t	[s]	Tiempo	22
$\theta(t)$	[rad]	Posición angular del aro	22
$\phi(t)$	[rad/s]	Velocidad angular de la bola	22
$\psi(t)$	[rad]	Posición angular de la bola	22
$y(t)$	[m]	Posición de la bola	22
R	[m]	Radio del aro	22
M	[Kg]	Masa del aro	22
m	[Kg]	Masa de la bola	22
I_a	[Kg/m^2]	Momento de inercia del aro	22
I_b	[Kg/m^2]	Momento de inercia de la bola	22
g	[m/s^2]	Aceleración de la gravedad	22
b_b	[Nm/s]	Coeficiente de fricción de la bola	23
b_m	[Nm/s]	Coeficiente de fricción del servomotor	23
r	[m]	Radio de rodadura de la bola	23
r_b	[m]	Radio de la bola	23
s		Variable en la Transformada de *Laplace*	24
rad/s		Unidad de medida de frecuencia, radianes por segundo	24
dB		Unidad de medida de magnitud, decibelios	24
f	[rad/s]	Frecuencia	24

PI		Controlador Proporcional Integral	25
θ_{ref}	[rad]	Referencia de posición angular del aro	26
K_p		Constante proporcional del controlador	26
K_i		Constante integral del controlador	26
K_θ	[V/rad]	Escalado de posición angular del aro	26
K_ω	[V/rad/s]	Escalado de velocidad del servomotor	26
K_m		Ganancia del servomotor	26
$\dot{\theta}$	[rad/s]	Velocidad angular del aro	26
τ_m	[s]	Constante de tiempo del servomotor	26
K		Ganancia de desviación de la bola	26
A, B, C		Coeficientes de un sistema lineal de segundo orden	26
$x(s)$		Señal para observar la fase no mínima	28
K_s		Factor de ganancia escalar	28
V	[Volts]	Voltaje aplicado al servomotor	30
G	[rad/s/V]	Ganancia de estado estacionario del servomotor	31
H	[s]	Ganancia de estado estacionario del aro	31
ΔV	[Volts]	Variación de voltaje a la entrada del servomotor	31
$\Delta\omega$	[rad/s]	Variación de velocidad angular del servomotor	31
K_a	[rad/s/V]	Ganancia del servomotor	32
t_1	[s]	Tiempo para el 28.3 % de la salida del servomotor	32
t_2	[s]	Tiempo para el 63.2 % de la salida del servomotor	32
T_p	[s]	Parámetro de un sistema de 1^{er} orden $(1{,}5 \cdot (t_2 - t_1))$	32
T_0	[s]	Parámetro de un sistema de 1^{er} orden $(t_2 - T_p)$	32
y_e	[rad]	Valor de la respuesta en estado estacionario	33
b/a		Razón de amortiguamiento de un sistema de 2^{do} orden	33
t_0	[s]	Pseudoperíodo de oscilación de un sistema de 2^{do} orden	33
M_p		La máxima sobre-elongación respecto al valor final	33
t_p	[s]	El instante de máxima sobre-elongación	33
t_d	[s]	El tiempo de retardo	33
a_0		Parámetro de un modelo de 2^{do} orden	33

Lista de Tablas

Lista de Figuras

1

Introducción

1.1 La Educación a distancia

La educación a distancia es la modalidad educativa en la cual los estudiantes y el profesor no necesitan compartir el mismo espacio físico durante el proceso de enseñanza - aprendizaje. Este modelo de educación se ha venido empleando desde hace mucho tiempo atrás, cuando las personas que querían estudiar comenzaron a verse limitadas de asistir a las escuelas por falta de tiempo, la lejanía, el coste de transporte, la incompatibilidad del horario laboral con el escolar, etc. Fue entonces cuando surgió la idea de enseñar sin requerir de la presencia física del alumno en un aula de clases. Los primeros pasos se dieron en los Estados Unidos en el año 1728. El profesor de caligrafía Caleb Phillipps en un anuncio en la Gaceta de Boston, proponía enviar semanalmente por correspondencia varias lecciones a personas de todo el país que estuvieran interesadas

en aprender este arte. El curso que se denominaba "*The New Method of Short Hand*", incluía la posibilidad de tutorías por correo y "garantizaba el aprendizaje perfecto de la caligrafía" [6, 7]. Este método se denominó educación por correspondencia y se formalizó en 1840 en Inglaterra, donde Isaac Pitman planteó la enseñanza de la taquigrafía a través de la "*Phonografic Corresponding Society*" mediante tarjetas postales que contenían los principios fundamentales de la materia. Estas tarjetas eran enviadas por correo a los estudiantes para que tradujeran breves pasajes de la Biblia y las devolvieran para su corrección [8, 9]. Esta modalidad de enseñanza también apareció en Alemania en 1856, con un curso de aprendizaje del lenguaje por correspondencia. Más tarde, en 1890 surgió en Estados Unidos un curso por correspondencia sobre "Minería y prevención de accidentes mineros" dirigido por el periodista Thomas J. Foster. Su principal objetivo era capacitar a los obreros de las minas de Pennsylvania sobre la prevención de riesgos laborales. Posteriormente, aparecieron en esta ciudad las llamadas escuelas internacionales por correspondencia (*ICS*), de las cuales se crearon filiales en todos los continentes [10]. Su objetivo era brindar una oportunidad educativa a los trabajadores que debido a diferentes causas se veían imposibilitados de asistir a las escuelas convencionales [11].

Para finales del siglo XVIII y principios del siglo XIX la educación por correspondencia se fue haciendo cada vez más popular. En 1891 la Universidad de *Qeensland* en Australia, ofrecía programas a distancia muy completos en varias asignaturas [12]. Mientras que en 1892 el *Pennsylvania State College* comienza sus cursos de agricultura por correspondencia. Para 1906 la Universidad de *Wisconsin* ofrece su extensión a distancia. Más adelante, en 1915, se funda la *National University Continuing Education Association*. Mientras que en 1922 el *Pennsylvania State College* comienza con sus cursos por radio al igual que la Universidad de *Columbia*. A su vez, en 1925 hace lo propio la *State University of Iowa*. Al finalizar la Segunda Guerra Mundial, se produjo una expansión de esta modalidad para facilitar el acceso a los centros educativos en todos los niveles, especialmente en los países industrializados occidentales y del centro de Europa. Esto obedeció al incremento de la demanda de mano de obra cualificada registrada [13–15]. Este período, desde 1720 hasta 1950 se conoce como la primera

generación de la educación a distancia, basada fundamentalmente en la enseñanza por correspondencia. Luego de este período, la correspondencia empezó a ser sustituida por otros medios que comenzaron a estar disponibles debido a los adelantos tecnológicos. Por ejemplo en 1950, la *Ford Foundation* comienza con programas educativos por televisión. En 1965 la Universidad de *Wisconsin* imparte cursos basados en comunicación telefónica. Mientras que en 1968 la Universidad de *Stanford* crea una red educativa por televisión y en 1969 comienza la *Open University* de Londres. En este período existía bastante recelo en cuanto a estos métodos de enseñanza. Se alegaba que la educación tenía que ser interactiva, que en un solo sentido era imposible enseñar con calidad, ya que los profesores necesitaban recibir realimentación de sus estudiantes para comprobar la efectividad del proceso de enseñanza-aprendizaje. Fue así que en la década de los 70 se introdujo el concepto de interactividad en la enseñanza y se profundizó en el uso de medios electrónicos tales como las audiocintas, las videocintas, la radio y la televisión. A principios de esta década se crea en España la Universidad Libre a Distancia que luego derivó en la actual Universidad Nacional de Educación a Distancia (UNED) [16]. Su principal objetivo era llevar la educación superior a los núcleos de población, alejados de las grandes metrópolis. A finales de los años 70 se crea la Asociación Argentina de Educación a Distancia (*AAED*) que lleva a cabo encuentros anuales con el objeto de compartir las experiencias en esta modalidad [15]. En la década de los 80 la educación a distancia se extiende en organismos oficiales, instituciones privadas y también en la universidad, siendo varias las universidades que desarrollan programas de educación a distancia; es en este período cuando la educación a distancia evoluciona hacia una herramienta verdaderamente interactiva con el uso de los ordenadores y la aplicación de videoconferencias.

En la década de los 80 la UNED comienza con la creación de centros regionales (centros asociados) en casi todas las provincias españolas, que contribuyen a asentar la universidad en todo el territorio nacional. En estos centros los tutores actúan como guías y asesores de los estudiantes. Además, se comienza a trabajar en una estructura internacional que involucra a distintos países de América Latina. Es así que la UNED se implanta en algunos países y su modelo metodológico se "exporta", convirtiéndose en

líder de la *AIESAD* (Asociación Iberoamericana de Educación Superior a Distancia). Esta asociación es una entidad sin ánimo de lucro y está integrada por representantes de la UNED (España), la Universidad Estatal a Distancia (Costa Rica), la Universidad Nacional Abierta (Venezuela), la Universidad de la Sabana (Bogotá, Colombia), la Universidad de Brasilia (Brasil), la Universidad de la Habana (Cuba), la Universidad Nacional Autónoma de México , el Instituto Politécnico de Leiría y la Universidad Abierta (Portugal) [17], entre otras.

A partir de la década de los 90 con el desarrollo de las redes de comunicaciones e *Internet* se hace cada vez más patente la emergencia de un nuevo modelo de educación, que muchos creen que va a hacer converger la educación presencial o tradicional y la educación a distancia, influido directamente por las Tecnologías de la Información y las Comunicaciones (*TICs*). Al igual que otras muchas universidades, la UNED ha incorporado a su modelo de enseñanza el uso de las *TICs*. Esto unido a la continua renovación de sus métodos de estudio y su presencia en muchas partes del mundo, la han convertido en la actualidad en la universidad más grande de España. Teniendo más de 260 000 estudiantes y una oferta educativa que abarca 26 Títulos de Grado, 43 Másteres, más de 600 programas de Formación Continua, 12 Cursos de Idiomas, más de un centenar de Cursos de Verano y casi 400 actividades de Extensión Universitaria.

En el ámbito de la enseñanza de ingeniería, la educación a distancia también ha evolucionado a tal punto que en nuestros días es común encontrarse con especialidades de ingeniería que se imparten a distancia en varias universidades; tal es el caso de la Ingeniería de Control. La enseñanza de esta especialidad requiere que los conceptos teóricos, que se imparten a través de las clases, sean complementados por actividades prácticas en los laboratorios. A través de estos experimentos los estudiantes pueden observar en la práctica algunos contenidos teóricos que en muchas ocasiones se tornan complicados de asimilar por medio de los métodos tradicionales. En la enseñanza a distancia constituye un inconveniente incorporar la experimentación práctica, porque los estudiantes tienen que estar físicamente presentes en el laboratorio para desarrollar este tipo de actividades. Es por esto que la enseñanza a distancia de la Ingeniería de Control puede beneficiarse de las condiciones actuales que proveen las *TICs* [18–20]. En

este contexto los experimentos pueden ser adaptados para que los estudiantes puedan acceder a ellos de forma remota a través de *Internet*. De esta forma desaparece la desventaja que implica la presencia física de los estudiantes en la universidad para realizar las prácticas de laboratorio [21–23]. A su vez, la práctica de laboratorio puede ser complementada con una simulación del proceso al que el estudiante se enfrentará durante la experimentación con la planta real. Estas simulaciones propician que el estudiante conozca el proceso e interactúe con el sistema de forma virtual antes de enfrentarse con el equipamiento real. Este tipo de aplicaciones que poseen una simulación del proceso y posibilitan el acceso a plantas reales de forma remota, son conocidos en el mundo de la educación en ingeniería como laboratorios virtuales y remotos.

1.2 Plataformas para la enseñanza de Ingeniería de Control

Desde el punto de vista de la enseñanza y el aprendizaje de la Ingeniería de Control, como se mencionó anteriormente, la experimentación práctica es esencial para el desarrollo de los estudiantes como futuros ingenieros. En muchos casos, estos experimentos se realizan a través de los laboratorios tradicionales, los cuales permiten a los estudiantes afianzar conceptos fundamentales tanto desde el punto de vista teórico como práctico. Estos laboratorios consisten básicamente en una planta piloto conectada a un ordenador. Estas plantas representan procesos industriales y sistemas didácticos que son interesantes desde el punto de vista de la enseñanza de la Ingeniería de Control, y posibilitan la realización de experimentos en lazo abierto o lazo cerrado para diferentes tipos de controladores. Entre los procesos más comunes a controlar se encuentran el control de nivel de un líquido de un tanque o varios tanques acoplados, el control de temperatura de un líquido, el control de posición y velocidad de un motor, por mencionar solo algunos. En la literatura se pueden encontrar muchos ejemplos de estos laboratorios tradicionales, sobre todo en universidades donde la modalidad de la enseñanza es presencial como por ejemplo: la práctica de laboratorio desarrollada en el Instituto Politécnico de Milán [24, 25], que consiste en el control de temperatura de dos transistores de un circuito electrónico, a partir de la velocidad de un motor que tiene acoplado un ventilador; las prácticas de laboratorio [26, 27] con el sistema del

péndulo invertido, de las Universidades de *Montgomery* e *Illinois* respectivamente; y el de la Universidad de *Mercer* [28], que emplea para las prácticas un brazo robótico.

Con este tipo de plataformas se facilita el planteamiento de problemas que permiten a los estudiantes aplicar los conocimientos aprendidos en clase, entrenándose en la aplicación del método científico en el mundo real, lo que les permite desarrollar habilidades cognitivas [29]. Es innegable el hecho de que los laboratorios convencionales constituyen una excelente herramienta que permite a los estudiantes la experimentación con equipamiento real. Sin embargo, estos laboratorios presentan una serie de inconvenientes entre los que se pueden destacar el costo actual del equipamiento, la necesidad de mantenimiento constante, el consumo de energía, la necesidad de espacio, las limitaciones del tiempo de disponibilidad, etc. Además este tipo de prácticas necesitan de la supervisión del profesor o de la persona encargada del laboratorio, por lo que el número de estudiantes que pueden ser atendidos a la vez, se ve limitado. En el caso de universidades con método de enseñanza a distancia como la UNED, los estudiantes no pueden trasladarse al laboratorio para realizar las prácticas. También ocurre que en ocasiones los laboratorios no pueden estar habilitados en los horarios en los que los estudiantes disponen de tiempo para realizar los experimentos.

Para disminuir estos inconvenientes desde hace algunos años se comenzaron a desarrollar simulaciones de procesos reales basadas en sus modelos matemáticos. Según la bibliografía, los primeros indicios apuntan al año 1997 en el Centro de Investigación Académica de la Universidad Estatal a Distancia de Costa Rica [30]. A partir del año 2002 se puede notar un incremento en el desarrollo de este tipo de aplicaciones sobre todo en el área de las ciencias y la ingeniería. En los últimos años, particularmente en el ámbito de la enseñanza de Ingeniería de Control, estas simulaciones se han vuelto muy populares por su alto valor didáctico y educativo. Sin embargo, es común que muchas simulaciones necesiten de *softwares* especializado que, debido a su coste, sólo se encuentran disponibles en los ordenadores de la universidad. Es por ello que una de las soluciones propuestas ha sido la ejecución remota de simulaciones de ingeniería a través de *Internet*. Este tipo de simulaciones a las que se puede acceder de forma remota a través de un simple navegador *web*, se les denomina laboratorios virtuales

o simulaciones distribuidas. Estos laboratorios permiten simular el funcionamiento de un laboratorio convencional en el que los estudiantes llevan a cabo los experimentos siguiendo un procedimientos similar al que siguen en un laboratorio convencional.

A continuación se comentan las ventajas más importantes del uso de los laboratorios virtuales:

- Al ser aplicaciones de software, se reducen enormemente los costes de uso.
- Los estudiantes pueden experimentar sin riesgo de ocasionar daños irreparables como ocurre en el caso del trabajo con equipamiento real. Esto permite a los estudiantes observar comportamientos y situaciones que no podrían ser vistos en una planta real, por el riesgo que esto implica.
- Se puede ofrecer la visualización de instrumentos y datos mediante objetos dinámicos, gráficos, imágenes y animaciones. Brindando de esta forma un ambiente propicio para el auto-aprendizaje, en el cual los estudiantes tienen plena libertad de modificar variables, parámetros y configuración del sistema, ofreciendo una gran flexibilidad en la personalización de los experimentos. Esto permite una visión mucho más intuitiva de aquellos fenómenos que en los laboratorios convencionales no pueden ser observados con la suficiente claridad gráfica.
- Permite a los estudiantes acercarse a los laboratorios mediante el uso de un simple navegador con un horario completamente flexible para realizar las prácticas.
- Permite a un número mayor de estudiantes experimentar con un laboratorio de manera asíncrona sin importar que no coincidan en tiempo y espacio.

Es necesario señalar que estos laboratorios virtuales no pueden sustituir del todo a los laboratorios convencionales ya que esto influiría negativamente en la calidad del proceso de enseñanza-aprendizaje ya que como se ha mencionado, hay situaciones y fenómenos que solo pueden observarse enfrentándose al equipamiento real. Por lo que se considera que los laboratorios virtuales deben emplearse como valiosas herramientas complementarias de gran utilidad para los estudiantes de ingeniería. Cuando se usan laboratorios virtuales en el proceso de enseñanza, se debe tener en cuenta que, como en cualquier aplicación remota, los estudiantes pueden convertirse en simples espectadores. Esto atenta contra la calidad del proceso de aprendizaje, por lo que durante

el diseño de la aplicación se debe tener cuidado para que ésta sea lo suficientemente interactiva de tal forma que el estudiante no pierda la atención y se encuentre en constante actividad durante el desarrollo de la práctica. Para ello es recomendable que el estudiante disponga de una guía o manual, que pueda seguir durante la realización de los experimentos; además de una evaluación que muestre si se han cumplido los objetivos trazados antes de la práctica. También es importante tener en cuenta que estas simulaciones no son más que una virtualización de la realidad, por lo que pueden provocar que el estudiante pierda la noción del proceso real que estudia. En este sentido, también es cierto que al simular un proceso, la simulación puede resultar muy distante de la realidad, lo que implica que debe tenerse un cuidado especial en el diseño de cada una de ellas para que el resultado sea lo más parecido al proceso original.

En este sentido, desde hace algunos años surgió también la idea de operar con equipamiento real desde una localización diferente a donde éste se encuentra. Estas aplicaciones se denominan laboratorios remotos. Los primeros indicios que se tienen de trabajo con equipamiento real de forma remota se evidencian en el trabajo [31] del año 1991. El cual propuso un esquema de acceso remoto para que las universidades pudieran compartir su equipamiento de laboratorio. Lo que significó sin duda, una idea revolucionaria para su época. Mientras que desde el punto de vista educativo el trabajo [32] presentado en 1996 se considera como uno de los primeros laboratorios remotos a través de *Internet* dedicado a la enseñanza del Control Automático. Aunque en la bibliografía no queda claro si los laboratorios remotos surgieron antes o después que los virtuales, lo cierto es que son considerados como una evolución de estos últimos. Porque en este caso ya no se trata de una virtualización del proceso real, sino de la manipulación de dicho proceso a través de Internet. Esto posibilita que los estudiantes realicen experimentos con equipamiento real desde cualquier sitio a través de *Internet*, utilizando plantas que representan procesos industriales y que están físicamente localizadas en las universidades. Posibilitándoles interactuar con sensores, actuadores e instrumentos de medición que permiten la transferencia de información de manera bidireccional. Las ventajas de este tipo de laboratorios son muchas:

- Se pueden aprovechar los laboratorios convencionales existentes y adaptarlos para

convertirlos en laboratorios remotos.

- Se pueden compartir laboratorios remotos entre universidades como es el caso de la plataforma *AutomatL@bs* [33], que permite compartir recursos entre varias universidades españolas.

Al usar equipamiento real, el estudiante no pierde la perspectiva del proceso real, como podría llegar a ocurrir en un laboratorio virtual. Es evidente que resulta muy ventajoso para los estudiantes poder interactuar con un proceso real desde cualquier sitio, utilizando solamente un navegador *web* y una conexión a *Internet*. Por lo que los estudiantes de educación a distancia no tienen que asistir a la universidad para realizar las prácticas de laboratorio, con lo que disminuyen los costos de transporte y tiempo. Además el equipamiento se encuentra disponible durante las 24 horas, por lo que pueden usarlo cuando dispongan de tiempo sin tener que regirse por un horario de apertura y cierre del laboratorio [22, 34–37].

No se puede dejar de señalar que los laboratorios remotos también tienen algunas desventajas entre las que se destacan las siguientes: la experimentación en tiempo real requiere períodos de muestreo pequeños, lo que puede derivar en componentes de *hardware* especializado, que puede resultar costoso; dependiendo de la arquitectura que se emplee en el diseño, se debe tener muy en cuenta la latencia de la red, que en ocasiones resulta un gran inconveniente; como la aplicación será utilizada a través de *Internet*, se debe tener en cuenta la seguridad y el control de acceso, ya que se trabaja con equipamiento real y se pueden provocar daños irreparables por usuarios malintencionados; debe tenerse en cuenta la robustez del *hardware* que se emplea, para que los estudiantes no tengan problemas durante el desarrollo de las prácticas.

Desde hace algún tiempo varios grupos de investigación de diferentes universidades han colaborado en la tarea de desarrollar aplicaciones que por un lado, se beneficien de las ventajas que proveen las simulaciones y por el otro, aprovechen las ventajas que proporciona el acceso al equipamiento real. Combinando estos dos tipos de aplicaciones han surgido plataformas interactivas para la experimentación denominadas laboratorios virtuales y remotos. Estas plataformas que integran la experimentación de laboratorios virtuales y remotos, no son más que una simulación de un proceso que

permite a los estudiantes realizar experimentos con un modelo virtual de un proceso real y además, acceder a una planta real que representa el mismo proceso. Estas plataformas constituyen potentes herramientas que contribuyen a mejorar la calidad del proceso de enseñanza-aprendizaje [23, 38, 39]. Existen muchos ejemplos de este tipo de plataformas, en algunos casos se combinan varias herramientas conocidas en el ámbito del control automático como son *MATLAB/Simulink* y *LabVIEW*. Tal es el caso de los laboratorios *MeRLab* de la Universidad de *Leicester* [40] y Telelabs de la Universidad *Western Australia* [41], que presentan varios instrumentos virtuales desarrollados con *LabVIEW* que permiten el estudio de brazos robóticos y otros sistemas de control. Usando también *LabVIEW*, la plataforma *Connexions* presenta instrumentos virtuales para el procesamiento de señales mediante simulaciones interactivas [42]. Mientras que la Universidad de *Chattanooga* dispone de simulaciones para el estudio de la dinámica de sistemas integrando *LabVIEW* con *MATLAB/Simulink* [43]. Otro ejemplo es el "*Automatic Control Telelab (ACT)*" de la Universidad de *Siena* [44], con el cual se pueden realizar experimentos de control de robots móviles y sistemas multi-agente. En España existen varios grupos de investigación que se dedican al desarrollo de este tipo de herramientas. Tal es el caso de la Universidad de Almería, donde han desarrollado herramientas para la enseñanza de la robótica utilizando la plataforma *LEGO* [45, 46], o el caso de la Universidad de Alicante con la plataforma *RobUALab* empleada para la enseñanza de robótica industrial [47, 48], por solo citar algunos ejemplos. La UNED ofrece cursos de control automático a distancia para un buen número de estudiantes cada año. En el Departamento de Informática y Automática (DIA), desde hace algunos años se han venido desarrollando un grupo importante de laboratorios virtuales y remotos para el apoyo a la enseñanza [49–52]. Estos laboratorios se encuentran incluidos en la plataforma *AutomatL@bs* [33], que es usada en la actualidad por estudiantes de 7 universidades españolas. Todas las aplicaciones mencionadas anteriormente tienen en común que son herramientas interactivas con buen nivel de visualización, que en algunos casos garantizan el acceso al equipamiento real, proporcionando un ambiente de experimentación que facilita el proceso de enseñanza-aprendizaje en el ámbito de la Ingeniería de Control.

1.3 Objetivos

El objetivo general de esta Tesis es diseñar e implementar plataformas interactivas de experimentación para el desarrollo de prácticas de laboratorio de forma virtual y remota en el ámbito de la enseñanza de la Ingeniería de Control y la Robótica Móvil. Para ello se desarrollará un entorno virtual en cada uno de los casos, lo cual constituye una simulación de cada proceso que permite realizar experimentos de forma interactiva y sencilla. Posteriormente se desarrollará un entorno de experimentación que incluye el control local de cada sistema. Finalmente se establecerá la conexión del entorno virtual con el experimental en ambos casos, permitiendo de esta forma el acceso remoto al equipamiento real desde *Internet*. Muchos de los resultados obtenidos en este trabajo podrían ser aplicados a la enseñanza en otras especialidades de ingeniería, que requieran prácticas de laboratorio en la formación de los futuros profesionales. Los siguientes objetivos específicos son abordados en esta Tesis:

- Implementar un laboratorio virtual para la realización de experimentos de control automático basado en el sistema bola y aro.
- Desarrollar una aplicación de control para la planta piloto que representa al sitema bola y aro.
- Integrar en una plataforma experimental un laboratorio virtual y remoto del sistema bola y aro.
- Implementar un laboratorio virtual para la realización de experimentos de control automático basado en el sistema bola y plato.
- Integrar en una plataforma experimental un laboratorio virtual y remoto del sistema bola y plato.
- Desarrollar un laboratorio virtual que permita realizar simulaciones de Robótica Móvil, específicamente en el área del control de formación de robots móviles con evasión de obstáculos.
- Implementar entornos experimentales para el control de formación y evasión de obstáculos de robots móviles.
- Desarrollar una plataforma interactiva de experimentación virtual y remota para el control de formación y evasión de obstáculos de robots móviles.

Estos objetivos se desarrollan en las diferentes partes que conforman esta Tesis, como se describe a continuación.

1.4 Estructura de la Tesis

El trabajo se ha dividido en dos partes fundamentales: la primera parte consiste en la presentación y descripción de las plataformas para el desarrollo de prácticas de Ingeniería de Control. Mientras que la segunda parte aborda los detalles de las plataformas para el desarrollo de prácticas de laboratorio con robots móviles. Se ha decidido dividir el trabajo en estas dos partes debido a que a pesar de que las cuatro plataformas están vinculadas a la enseñanza del Control Automático y el desarrollo de habilidades por parte de los futuros ingenieros, existen entre ellas varias diferencias. Fundamentalmente en cuanto a la implementación, complejidad y especificidades de los entornos experimentales de cada una. Los detalles de cada uno de los capítulos se presenta a continuación.

Capítulo 2

En este capítulo se describen los detalles teóricos de las plantas que se emplean en la construcción de las plataformas, tanto para el del sistema bola y aro, como para los robots móviles. En el caso del sistema bola y aro, primeramente se presenta su modelo y los detalles de su dinámica, así como los diferentes experimentos que con esta planta se pueden llevar a cabo (control de posición del aro, control de posición de la bola, ceros de transmisión y comportamiento de fase no mínima). Luego se presenta el proceso de identificación de la planta real (parámetros del motor y dinámica de la bola en el interior del aro). Para esto se emplean distintos experimentos y se comparan los resultados obtenidos. Finalmente se calculan los parámetros del controlador que se utiliza en el lazo de control empleando varios métodos de ajuste de controladores e igualmente comparando los resultados obtenidos. En el caso de la plataforma con robots móviles, primeramente se presentan las características técnicas de los robots *Moway*. A continuación los destalles de su dinámica y el control de su posición, así como los experimentos

que se pueden realizar (control de formación con evasión de obstáculos) con una plataforma descentralizada (los robots analizan su entorno y toman decisiones). Por el tipo de formaciones que se llevan a cabo, los robots se comunican entre ellos usando una red inalámbrica basada en radio-frecuencia. Además necesitan conocer su posición por lo que se comunican con un ordenador que les sirve como sensor de posición absoluta, usando para ello la misma red inalámbrica. Finalmente se presentan los detalles de la identificación de estos robots y se analizan los resultados obtenidos. Para el caso de los robots *Surveyor SRV-1* se presentan sus características y los detalles de su control, y se propone una arquitectura descentralizada para el control de formación (los robots pueden tomar decisiones). Como el tipo de formación que se lleva a cabo es en línea recta (tomando como referencia el primer robot), los robots no necesitan recibir sus posiciones absolutas y además no necesitan comunicarse entre ellos. La herramienta de desarrollo para el laboratorio remoto es EJS ("*Easy Java Simulations*") y para la comunicación con los robots, *MATLAB* a través de una red inalámbrica (*wifi*).

Capítulo 3

En este capítulo se detallan los laboratorios virtuales y remotos desarrollados para ambas plataformas, comenzando con una descripción de la arquitectura empleada. En el caso de la plataforma para el sistema bola y aro, primeramente se presentan los detalles de las dos variantes desarrolladas (*LabVIEW* y *MATLAB*), así como las especificidades de cada una de las aplicaciones (cliente y servidor) en ambas variantes. Se hace especial hincapié en el entorno experimental por la complejidad que supone el trabajo con equipamiento real. También se describen los detalles de la comunicación a través de *Internet* usando el protocolo *TCP*. En todos los casos se describe el código fuente de cada una de estas aplicaciones. En ambos casos se culmina con una presentación de las ventajas y desventajas tanto para la variante en la que se empleó *LabVIEW*, como la variante en la que se empleó *MATLAB* como herramientas de desarrollo.

Para la plataforma de desarrollo de prácticas con robots móviles, se comienza presentando en primer lugar la arquitectura empleada y posteriormente se describen las aplicaciones cliente y servidor en ambas versiones, tanto para los robots *Moway* como

para los robots *Surveyor SRV-1*. En todos los casos se detalla el código fuente de estas aplicaciones tanto para el caso del cliente como el servidor. En ambos casos se resaltan los entornos experimentales debido a la complejidad que supone la obtención de la posición absoluta de los robots en este tipo de experimentos. Debido a los problemas de iluminación, los retardos en el procesamiento de imágenes y las pérdidas de datos en las comunicaciones inalámbricas, etc. Análogamente a lo realizado con la plataforma anterior, finalmente se presentan las ventajas y desventajas de estas dos variantes.

Capítulo 4

En este capítulo se presentan y discuten los resultados obtenidos al usar cada una de estas plataformas en sus diferentes versiones. Primeramente se presentan los resultados de los laboratorios virtuales y remotos con el sistema bola y aro y con el sistema bola y plato para los diferentes experimentos que con ellos se pueden realizar. Para el caso de las plataformas de experimentación con robots móviles, igualmente se presentan y discuten los resultados obtenidos para ambas versiones. En todas las plataformas se discuten y valoran los resultados obtenidos en los diferentes experimentos realizados.

Capítulo 5

En este capítulo se presentan las conclusiones generales, las conclusiones específicas y los trabajos futuros relacionados a las líneas de investigación de esta Tesis, así como las posibles mejoras a las plataformas que se pueden llevar a cabo.

1.5 Principales contribuciones

Las contribuciones de esta Tesis están divididas en dos aspectos: aplicaciones desarrolladas y publicaciones.

1.5.1 Aplicaciones desarrolladas

Los resultados de esta Tesis doctoral incluyen la creación de plataformas que permiten la realización de laboratorios virtuales y remotos en un ambiente interactivo.

Sus principales fines son pedagógicos y tienen un gran impacto en la enseñanza de ingeniería de control. Las plataformas desarrolladas son las siguientes:

- Plataforma para el desarrollo de experimentos de control: Consiste en una potente herramienta completamente operativa que proporciona una simulación del sistema bola y aro así como la conexión con una planta real que representa este sistema para su control a través de *Internet*.
- Plataforma para experimentos con robot móviles: Consiste en un laboratorio virtual y remoto disponible para realizar experimentos de control de formación de robots móviles con evasión de obstáculos y sistemas multi-agente.

Para el desarrollo de estas plataformas se han diseñado e implementado un conjunto de aplicaciones de *software* que interactúan entre sí y con el *hardware*, obteniéndose como resultado herramientas de gran utilidad para el desarrollo de prácticas de laboratorio. Estas aplicaciones se mencionan a continuación:

- Simulación interactiva del sistema bola y aro empleando *EJS*. Esta aplicación actúa como cliente en esta plataforma.
- Aplicación que controla la planta piloto bola y aro desarrollada en *LabVIEW*. Esta aplicación puede actuar como servidor en dicha plataforma.
- Aplicación que controla la planta piloto bola y aro desarrollada en *MATLAB*. Esta aplicación puede actuar como servidor en dicha plataforma.
- Simulación interactiva en 3D del sistema bola y plato empleando *EJS*. Esta aplicación actúa como cliente en esta plataforma.
- Interfaz de comunicación entre *Java* y *Visual C#* para la comunicación con la planta real del sistema bola y plato.
- Simulador para experimentos de control de formación de robots móviles. Esta aplicación actúa como cliente en esta plataforma.
- Aplicación *Moway Server* que permite obtener las posiciones de los robots y la envía inalámbricamente a cada uno de ellos. Esta aplicación actúa como servidor en esta plataforma.
- Programas que se ejecutan en cada uno de los robots móviles, dependiendo de su papel en la formación (maestro o esclavo). Estas aplicaciones también forman

parte del lado del servidor en esta plataforma.

- Interfaz entre el simulador y la aplicación *Moway Server*. Esta interfaz es la implementación del protocolo *TCP* para la comunicación entre *Java* y *Visual C#*.
- Aplicación cliente del laboratorio remoto con robots *Surveyor SRV-1* desarrollada en *EJS*. Permite la interacción con el laboratorio remoto de control de formación de robots.

1.5.2 Publicaciones

Durante el desarrollo de esta Tesis doctoral se han publicado varios artículos en revistas especializadas y en conferencias internacionales relacionadas con la ingeniería de control. Algunos de estos artículos han sido obtenidos como resultados de la realización de esta Tesis. Otros en cambio, han sido desarrollados en colaboración directa con autores de otros grupos de investigación. Como es el caso del grupo de investigación del Departamento de Energía Eléctrica, Sistemas y Automatización de la Universidad de Gante en Bélgica.

Artículos en Revistas

Los siguientes artículos han sido obtenidos como resultados de esta Tesis y se han publicado en revistas internacionales especializadas en materias directamente relacionadas con esta investigación:

- Fabregas E., Farias G., Dormido-Canto S., Dormido S., Esquembre F. "Developing a remote laboratory for engineering education", *Computers & Education*, vol. 57, no. 2, pp. 1686-1697, 2011, doi: 10.1016/j.compedu.2011.02.015, IF: 2.621.
- M. Guinaldo, G. Farias, E. Fabregas, J. Sánchez, S. Dormido-Canto, S. Dormido. "An Interactive Simulator for Networked Mobile Robots", *IEEE Network Special Issue*, vol. 26, no. 3, pp. 14-20, 2012, doi: 10.1109/MNET.2012.6201211, IF: 2.239.
- Clara M. Ionescu, Ernesto Fabregas, Stefana M. Cristescu, Sebastián Dormido, Robin De Keyser. "A Remote Laboratory as an Innovative Educational Tool for

Practicing Control Engineering Concepts", *IEEE Transactions on Education*, vol. PP, no. 99, 2013, doi: 10.1109/TE.2013.2249516, IF: 1.021.

- E. Fabregas, G. Farias, S. Dormido-Canto, M. Guinaldo, J. Sánchez, S. Dormido. "Virtual and real laboratory for teaching mobile robotic", *IEEE Transactions on Industrial Electronics*, 2013, IF: 5.16. (Enviado)
- M. Guinaldo, E. Fabregas, G. Farias, S. Dormido-Canto, D. Chaos, J. Sánchez, S. Dormido. "Mobile robots experimental environment with event-based wireless communications". *Sensors - Open Access Journal*, 2013, IF: 1.739. (Enviado)

Artículos en Conferencias

Los siguientes artículos se han obtenido como resultados de este trabajo y han sido presentados en conferencias internacionales relacionadas con la ingeniería de control:

- E. Fabregas, N. Duro, R. Dormido, S. Dormido-Canto, H. Vargas, S. Dormido. "Virtual and remote experimentation with the Ball and Hoop system". *14^{th} International IEEE Conference on Emerging Technologies and Factory Automation*. IEEE Reference MD-003301. Palma de Mallorca. España. September, 2009.
- Daniel V. Neamtu, E. Fabregas, R. Hordea, R. De Keyser. "Remote Laboratory for Leader-Follower Formation Control". *Jornadas de Estudio IUAP/PAI DYSCO*. Ghent. Belgium. May, 2010.
- E. Fabregas, G. Farias, S. Dormido-Canto, S. Dormido, F. Esquembre. "A Practical demonstration of reset control with the ball and hoop system". *9^{th} Portuguese Conference on Automatic Control*. Coimbra, Portugal, September, 2010.
- Daniel V. Neamtu, Ernesto Fabregas, Bart Wyns, Robin De Keyser, Sebastian Dormido, Clara M. Ionescu. "A Remote Laboratory for Mobile Robot Applications". *18^{th} IFAC World Congress*. Milan. Italy, September, 2011.

2

Plataformas de experimentación

En este capítulo se presentan y describen los sistemas que constituyen las plataformas de experimentación. Para ello en primer lugar, se comienza estudiando sus componentes y características físicas. A continuación, se procede al análisis de las propiedades que posibilitan su uso en la enseñanza de ingeniería de control y robótica. Para finalizar, se presentan los diferentes experimentos que pueden llevarse a cabo para demostrar aspectos teóricos fundamentales en la enseñanza de esta materia.

2.1 Descripción de la plataformas de experimentos de control

Para el desarrollo de las plataforma de experimentación de control automático se emplean las plantas piloto basadas en sistemas muy conocidos en el ámbito de la enseñanza de la Ingeniería de Control como son: el sistema bola y aro, y el sistema bola y plato.

2.1.1 Descripción del sistema bola y aro

Este sistema está compuesto por una esfera de acero que puede rodar en el interior de un aro metálico, que está montado verticalmente sobre el eje de un servomotor de corriente continua; por lo que puede girar entorno a éste. La Figura 2.1 muestra el *"Ball and Hoop Apparatus CE9"* de *TecQuipment* [53] que representa a este sistema.

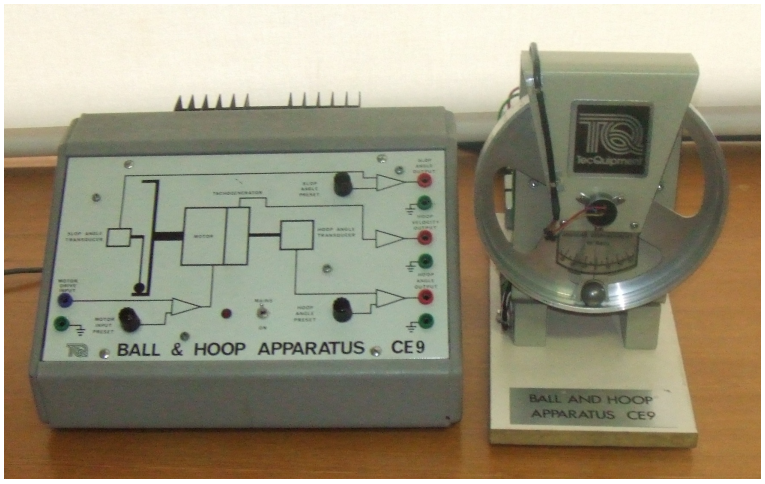


Figura 2.1. Planta real: *"Ball and Hoop Apparatus CE9"* de *TecQuipment*.

Partiendo de la posición de reposo, con la bola en su punto de equilibrio, cuando el aro gira, la bola tiende a moverse en la dirección de rotación del aro. En cierto punto, la gravedad vence las fuerzas de fricción entre el aro y la bola, tendiendo esta última a regresar a su estado de equilibrio, comportándose con un movimiento oscilatorio. El movimiento de la bola en el interior del aro es debido principalmente al par motor que provoca una aceleración angular en el aro y este a su vez, una desviación angular de la bola de su posición de equilibrio. La dinámica del movimiento de la bola en el interior del aro es análoga a la dinámica de los líquidos en el interior de un contenedor cilíndrico, como se observa en la Figura 4.2. Este sistema fue propuesto por el profesor Peter E. Wellstead para estudiar las oscilaciones de un líquido en el interior de un contenedor cilíndrico [54].

Este fenómeno es conocido como agitación de los líquidos y ocurre a menudo cuando se transportan grandes cantidades de líquido y éste se somete a variaciones bruscas de velocidad y dirección. Estas variaciones provocan oscilaciones en el líquido que pueden

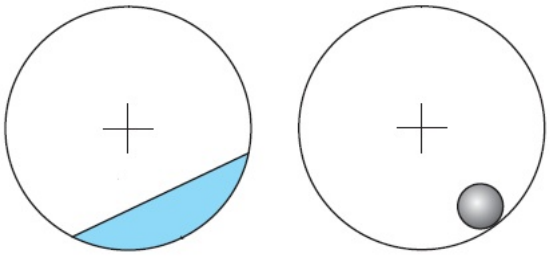


Figura 2.2. Analogía entre el movimiento de un líquido en el interior de un contenedor cilíndrico y la dinámica de la bola dentro del aro.

ocasionar el movimiento del contenedor lo que puede resultar indeseable y peligroso dependiendo del tipo de líquido; tal como sucede en los camiones que distribuyen mezcla de cemento, buques petroleros, cohetes con combustible líquido y vagones de ferrocarril, entre otros. La Figura 2.3 muestra algunos de estos ejemplos. Este sistema también es adecuado para demostrar algunos conceptos básicos de la dinámica de sistemas y teoría de control, como por ejemplo los ceros de transmisión, los sistemas de fase no mínima y el control de posición [55].

Figura 2.3. Transporte de líquidos.

2.1.1.1 Modelo del sistema bola y aro

El esquema del modelo del sistema se muestra en la Figura 2.4. Sus principales componentes son: el par motor (τ), el ángulo de la posición del aro (θ), el ángulo de desviación de la bola con respecto a la vertical (ψ),la velocidad angular de la bola (ϕ) y la posición de la bola con respecto al aro (y).

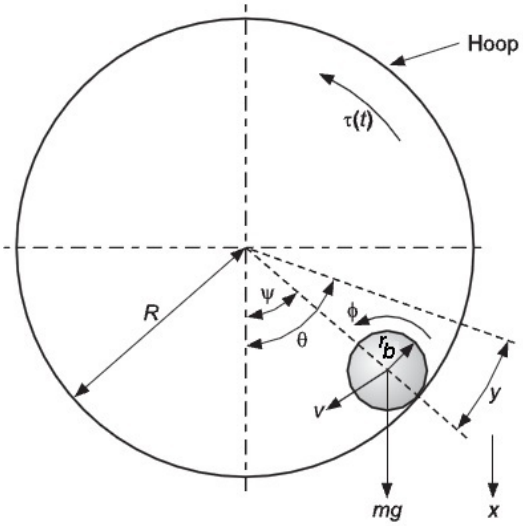


Figura 2.4. Esquema del modelo del sistema bola y aro.

El resto de las variables conocidas para el modelo son: el radio y la masa del aro (R, M) y el radio de la bola y la masa de la bola (r_b, m). El comportamiento dinámico del sistema se representa completamente en ecuaciones de movimiento de la posición angular del aro (θ) y la posición de la bola (y). Estas variables se emplean como coordenadas generalizadas en las ecuaciones de *Lagrange*. Después de varias transformaciones y sustituciones, se obtienen las ecuaciones que describen el comportamiento dinámico del sistema [55, 56]. La Ecuación 2.1 describe la dinámica de la bola en el interior del aro y la Ecuación 2.2 describe la dinámica del aro.

$$\left[\frac{I_b}{r_b^2} + \frac{m(R-r_b)^2}{R}\right]\ddot{y} + \frac{b_b}{r_b^2}\dot{y} - \frac{m\,(R-r_b)^2}{R}\ddot{\theta} = mg\frac{(R-r_b)}{R}\sin\psi \tag{2.1}$$

$$\left[I_a + m(R-r_b)^2\right]\ddot{\theta} + b_m\dot{\theta} - \frac{m(R-r_b)^2}{R}\ddot{y} = \tau\,(t) - mg\,(R-r_b)\sin\psi \tag{2.2}$$

Donde:

- I_a es el momento de inercia del aro.
- I_b es el momento de inercia de la bola.
- b_m es el coeficiente de fricción del servomotor.
- b_b es el coeficiente de fricción de la bola.
- g es la aceleración de la gravedad.

2.1.1.2 Dinámica del sistema bola y aro

La dinámica del sistema bola y aro es muy rica y compleja porque tiene un comportamiento oscilatorio y que varía constantemente. Por esta razón, su análisis y estudio son muy interesantes para los estudiantes de la especialidad de control automático desde el punto de vista didáctico. Las principales características que lo distinguen son la capacidad de demostrar los ceros de la transmisión [57], el fenómeno de la resonancia [58] y el comportamiento de fase no mínima del sistema [59]. Otros experimentos que se pueden implementar con este sistema son las estrategias de control de posición del aro y de control desviación de la bola de su posición de equilibrio [60].

Ceros de transmisión

Para demostrar el experimento de los ceros de transmisión se realizan algunas transformaciones en las Ecuaciones 2.1 y 2.2, que se describen a continuación. Primeramente se linealizan dichas ecuaciones considerando que para ángulos pequeños el seno del ángulo de la bola con respecto a la vertical se hace igual a dicho ángulo ($\sin\psi \cong \psi$). Se considera además que el radio del aro es mucho mayor que el radio de la bola ($R \gg r_b$); por lo que se desprecia este último frente al del aro. También se tiene en cuenta que el radio de la bola (r_b) es diferente al radio de rodadura (r). En la Figura 2.5 se observa la representación de un corte transversal de la bola y el aro [5].

Finalmente se combinan ambas ecuaciones y se aplica la *Transformada de Laplace* para obtener la función de transferencia del sistema que relaciona la posición de la bola con respecto al punto cero del aro *y(s)* (salida) y el ángulo de posición del aro *θ(s)* (entrada), como se muestra en la Ecuación (2.3).

La función de transferencia del sistema obtenida tiene dos ceros imaginarios puros

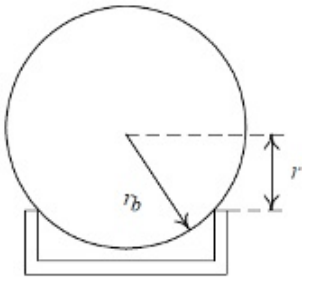


Figura 2.5. Sección transversal de la bola y el aro.

en $s = \pm\sqrt{\frac{g}{R}}i$. Esto significa que si a la posición del aro $\theta(s)$ se le aplica una señal sinusoidal con una frecuencia igual a $\sqrt{\frac{g}{R}}$ en *rad/s*, entonces habrá una respuesta de cero exacto en la posición de la bola $y(s)$. En términos físicos esto significa que la bola oscila en el interior del aro exactamente a la misma frecuencia que éste. Para un observador situado en el aro, la bola está detenida, porque la bola y el aro se mueven con un sincronismo exacto [54].

$$\frac{y(s)}{\theta(s)} = R\left[\frac{s^2 + \frac{g}{R}}{\left(\frac{2r_b^2}{5r^2} + 1\right)s^2 + \frac{b_b}{mr^2}s + \frac{g}{R}}\right] \tag{2.3}$$

Si se disminuye la frecuencia de la señal sinusoidal aplicada a la entrada hasta el valor de la frecuencia del polo resonante, cuando se alcanza el estado estacionario, las posiciones del aro y de la bola oscilan manteniéndose con amplitudes contantes. En este momento la bola y el aro no estarían sincronizados. Puede que sea necesario reducir la amplitud de la señal sinusoidal a la mitad para prevenir que el ángulo de la bola se haga demasiado grande [61].

En la Figura 2.6 se muestra el Diagrama de *Bode* de magnitud del sistema. En el cual el eje de las abscisas representa la frecuencia en *rad/s* y el eje de las ordenadas representa la magnitud en *dB*. Como se puede apreciar existe un pico resonante para la frecuencia *f=9.074 rad/s*. Además, la frecuencia a la que se obtienen los ceros de transmisión es *f=10.74 rad/s*.

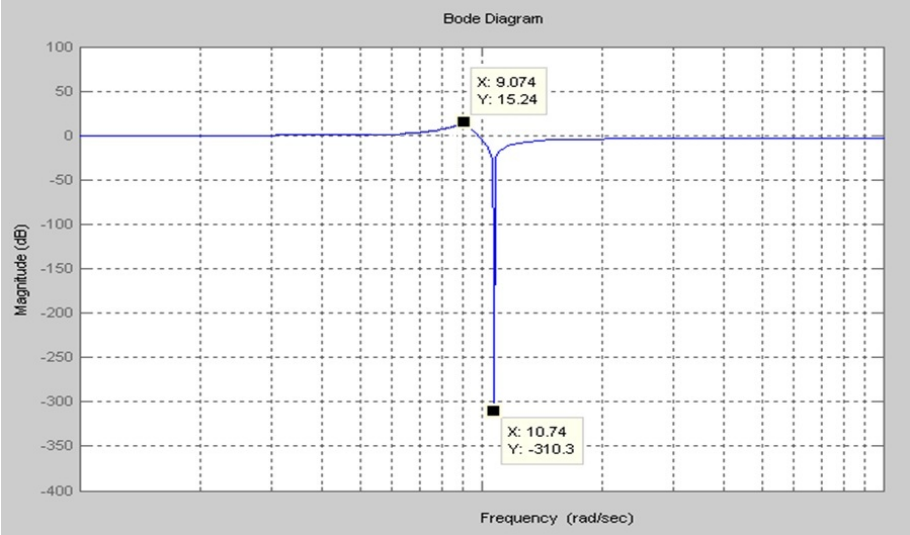


Figura 2.6. Diagrama de *Bode* de magnitud del sistema.

Control del ángulo de posición del aro

El experimento consiste en controlar el ángulo de posición del aro. Como el aro se encuentra unido al eje del servomotor, es necesario llevar a cabo el control del mismo. Para ello se diseña un lazo de control de realimentación como se observa en el diagrama de bloques representado en la Figura 2.7. Los bloques principales son: el controlador *PI*, un modelo de primer orden que representa al motor y un integrador para poder obtener a partir de la velocidad angular del motor, su ángulo de posición. Es necesario destacar que en teoría bastaría con un controlador Proporcional (P) pero se añade acción integral debido a la zona muerta del motor.

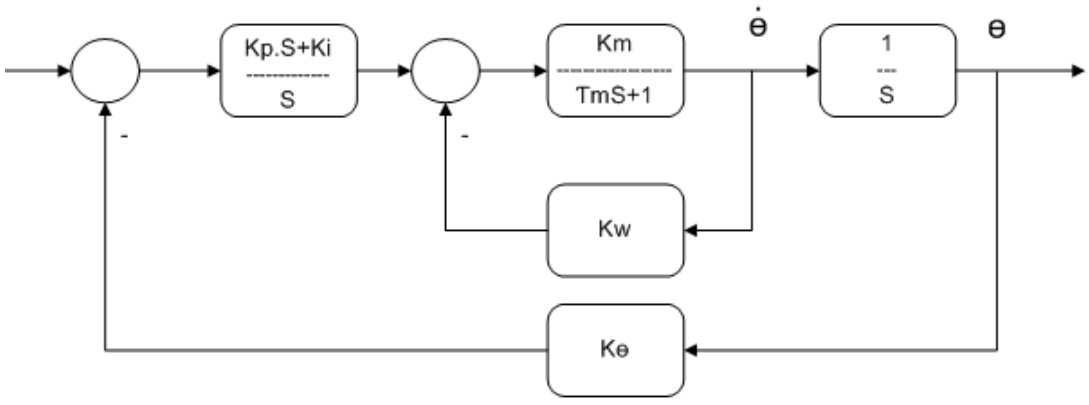


Figura 2.7. Diagrama de bloques del control de posición del aro.

Las variables y constantes representadas en la figura son las siguientes:

- θ_{ref} es la referencia de posición del aro.
- K_p es la constante proporcional del controlador *PI*.
- K_i es la constante integral del controlador *PI*.
- K_m es la ganancia del servomotor.
- $\dot{\theta}$ es la velocidad angular del aro.
- τ_m es la constante de tiempo del servomotor.
- θ es el ángulo de posición del aro.
- K_θ es la constante de realimentación de control de posición del servomotor.
- K_ω es la constante de realimentación de control de velocidad del servomotor.

El diagrama lo forman también un lazo de realimentación negativa de la posición del aro a través de la ganancia (K_θ), y un lazo interno de realimentación negativa de velocidad del aro a través de una ganancia (K_ω), para lograr una mejor estabilidad del sistema.

Tanto estas ganancias, como las constantes relacionadas con el servomotor y los parámetros del controlador *PI* empleado, se obtienen por métodos experimentales para la planta real, como se analizará posteriormente. Usando este lazo de control, al aplicar un salto en escalón a la entrada, el aro sigue a la referencia hasta alcanzar el estado estacionario. El movimiento del aro ocasiona un desplazamiento de la bola de su posición de equilibrio. El objetivo del siguiente experimento consiste en mover el aro, causando la menor desviación posible de la bola de su posición de reposo.

Control del ángulo de desviación de la bola

Para demostrar el control del ángulo de desviación de la bola de su posición de equilibrio, se añaden al diagrama anterior un bloque que representa la dinámica de la bola en el interior del aro y un lazo de realimentación negativa del ángulo de la bola hacia el ángulo de referencia del aro. Dicho lazo se incorpora a través de una ganancia (K) denominada ganancia de desviación, como se muestra en la Figura 2.8. Donde A, B y C son los coeficientes del modelo lineal de segundo orden.

El siguiente paso es seleccionar un valor apropiado para la ganancia de desviación

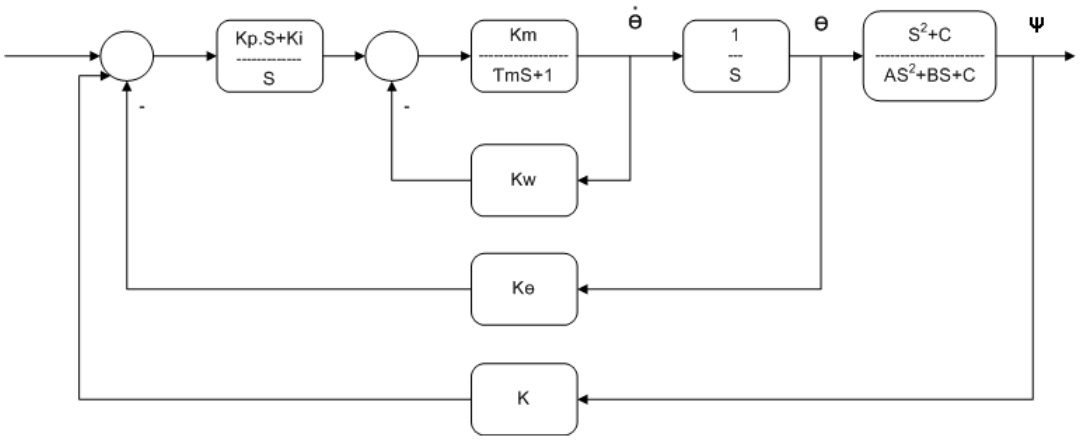


Figura 2.8. Diagrama de bloques del control del ángulo de desviación de la bola.

(K), lo que resulta en un compromiso de ingeniería, ya que se requiere que la posición angular del aro responda rápidamente a una entrada escalón en su referencia, causando la mínima desviación de la posición de equilibrio de la bola. En este contexto el sistema bola y aro se puede ver como un sistema de una entrada y dos salidas. Al aumentar el valor de K se fuerza al aro a moverse más despacio para mantener la desviación de la bola en un valor mínimo. Examinando el lugar geométrico de las raíces del sistema y variando K desde cero, se puede obtener su valor deseado. Para valores pequeños de K la dinámica del aro es dominante por lo que el aro responde rápidamente a cambios en la referencia ocasionando oscilaciones considerables en la posición de la bola. Para valores grandes de K la dinámica de la bola se hace dominante por lo que el aro responde lentamente a los cambios en la referencia, ocasionando una ligera desviación en la posición de equilibrio de la bola. La Figura 2.9 muestra el lugar geométrico de las raíces del sistema de lazo cerrado según se varía el valor de K.

La figura de la izquierda muestra la posición de los polos para valores pequeños de K. Mientras que la de la derecha muestra la posición de los polos para valores grandes. El valor adecuado se escoge de forma tal que los dos pares de polos complejos conjugados tengan la misma parte real, como se muestra en la figura del centro. Este valor garantiza que ninguna de las dos dinámicas sea dominante sobre la otra y por tanto se obtenga la mejor velocidad de respuesta del aro frente a cambios en la referencia. Esto origina la menor desviación posible de la bola de su posición de equilibrio. Como se observa

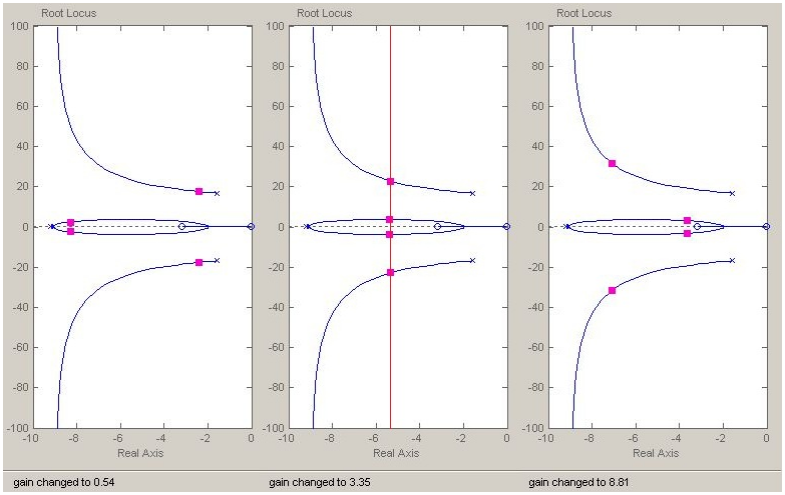


Figura 2.9. Lugar geométrico de las raíces para $K = 0.54$, $K = 3.35$ y $K = 8.81$.

en la Figura 2.9, en este caso el valor adecuado es $K = 3.35$.

Comportamiento de fase no mínima

Tomando en cuenta las dos salidas del sistema ($\theta(s)$ y $\psi(s)$), es posible construir una señal auxiliar $x(s)$ dada por:

$$x(s) = \theta(s) - K_s \psi(s) \tag{2.4}$$

K_s es un factor de ganancia escalar, cuando este coeficiente es la unidad, $x(s)$ es la variable $\frac{y(s)}{R}$ que se corresponde con la posición escalada de la bola en el interior del aro. Reescribiendo la Ecuación (2.4) como una función de transferencia se obtiene:

$$\frac{x(s)}{\theta(s)} = 1 - K_s \frac{\psi(s)}{\theta(s)} \tag{2.5}$$

En el lugar geométrico de las raíces para la función de transferencia de la Ecuación (2.5), al variar el factor K_s la parte del gráfico queda en el semi-plano derecho. Esto implica que para ciertos valores de dicha ganancia, el sistema tiene un comportamiento de fase no mínima. Lo que significa que inicialmente el sistema tenderá a ir en la dirección contraria a la dirección final, cuando se aplica un salto en escalón a la entrada en $t = 10s$.

En la Figura 2.10 [56] se observa el la señal $x(s)$ con respecto al tiempo para variaciones del valor del factor K_s entre 1 y 20.

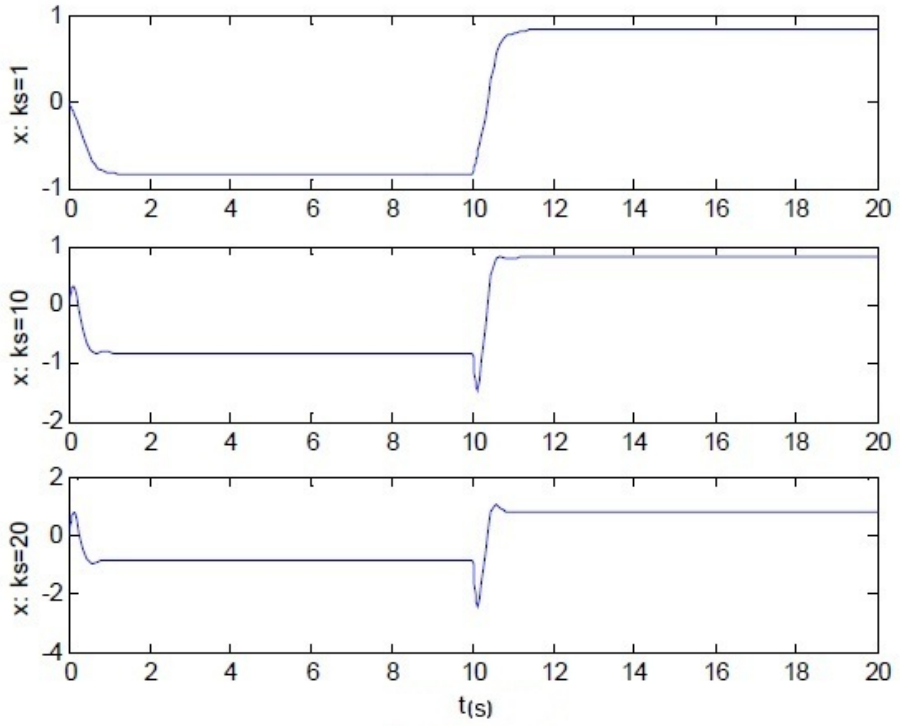


Figura 2.10. Comportamiento de fase no mínima para valores de K_s entre 1 y 20.

2.1.1.3 Planta real del sistema bola y aro

Una vez presentados los aspectos teóricos del sistema bola y aro, se procede a presentar e identificar la planta real con la cual se desarrolla el laboratorio remoto. La Figura 2.1 muestra *"Ball and Hoop Apparatus CE9"* de *TecQuipment* que es la planta real que representa a este sistema. Sus componentes físicos son los siguientes:

- Un aro metálico del cual se conocen su masa y su radio.
- Un servomotor de corriente continua que tiene montado verticalmente el aro en su eje y del cual no se conocen sus parámetros.
- Una esfera de acero de la cual se conocen su masa y su radio.
- Un sensor de posición angular del aro.
- Un sensor de posición angular de la bola.
- Un sensor de velocidad del aro.

Una vez descritos los componentes de la planta real, se procede a obtener los parámetros del sistema empleando diferentes variantes.

Obtención de los parámetros del servomotor

En la Figura 2.11 se muestra el diagrama de bloques del sistema bola y aro, el cual se puede separar en dos funciones de transferencia, tomando como entrada el voltaje aplicado al servomotor (V), el cual produce el ángulo de giro en el aro ($\theta(s)$), que a su vez provoca la desviación del ángulo de equilibrio en la posición de la bola ($\psi(s)$).

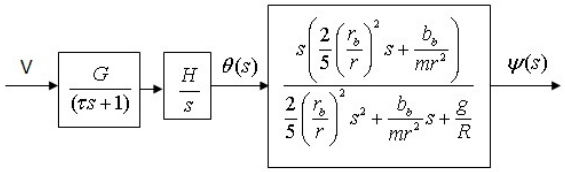


Figura 2.11. Diagrama de bloques del sistema bola y aro.

Los dos primeros bloques del diagrama corresponden a la dinámica del servomotor, sus parámetros se pueden calcular aplicando un salto en escalón a la entrada y registrando su velocidad con ayuda de una aplicación desarrollada en *LabVIEW* [62]. Esta aplicación almacena los datos del sensor de velocidad angular de la planta real en un fichero de texto. El conjunto de datos obtenido se carga en el espacio de trabajo de *MATLAB* [63] y se dibuja a través de las instrucciones que se muestran en el segmento de Código 2.1.

```
load Motor.txt;   %Se carga el fichero de datos
tac=Motor(:,4);
t=Motor(:,3);     %Se obtiene el tiempo
t=t-t(1);
ve=Motor(:,1);    %Se obtiene la velocidad
tau=Motor(:,2);   %Se obtiene la señal de control
plot(t,ve,t,tau,t,tac)    %Se dibuja el gráfico
```

Código 2.1. Instrucciones para cargar los datos del motor al espacio de trabajo de *MATLAB*

Para facilitar el cálculo de los parámetros del servomotor, se obtienen de manera gráfica los valores necesarios. La Figura 2.12 muestra la respuesta del sistema a una entrada escalón. Inicialmente el voltaje aplicado al servomotor (entrada) es 0.0 Voltios. A los dos segundos se cambia este valor a 5.0 Voltios. Como se observa la velocidad del servomotor (salida) comienza a variar su valor hasta pasados 2 segundos. A partir de este tiempo permanece constante aproximadamente en 3.13 rad/s.

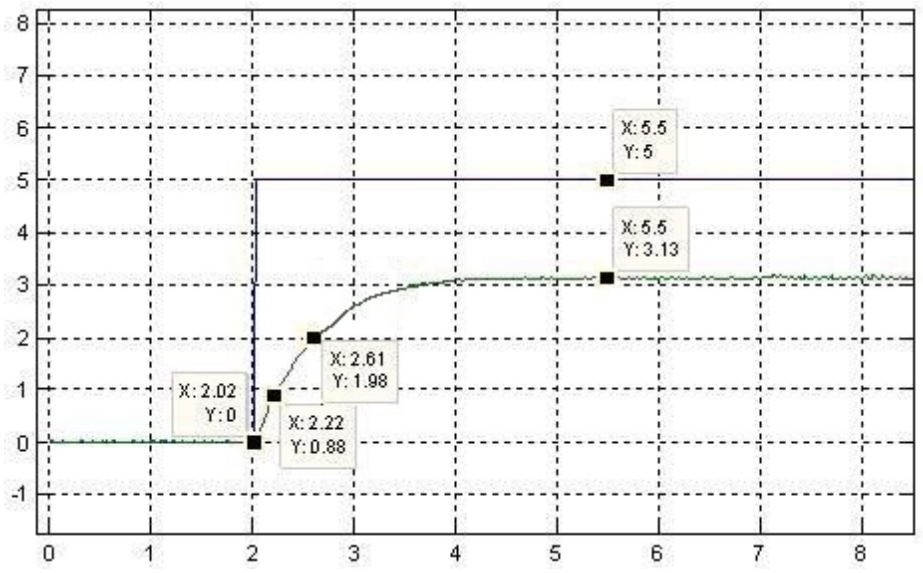


Figura 2.12. Respuesta del sistema a una entrada escalón.

El servomotor es un sistema de primer orden. Para obtener la constante de tiempo de dicho sistema se calcula el tiempo que tarda la salida en alcanzar el 63.2 % del valor final, al aplicar un salto en escalón en la entrada. En la Tabla 2.1 se resumen los valores que se necesitan para obtener la constante de tiempo.

% $y_e(t)$	$y_e(t)$	$t(s)$
0	0	2.02
28.3	0.88	2.22
63.2	1.98	2.61
100	3.13	5.5

Tabla 2.1. Salida del sistema al aplicar una entrada escalón.

Utilizando estos valores se obtiene:

$$\tau = t(63{,}2\,\%y_e(t)) - t(0) \quad \tau = 2,61s - 2,02s \quad \boxed{\tau = 0{,}59\ s} \tag{2.6}$$

Para obtener el resto de los parámetros del servomotor (representados en los dos primeros bloques Figura 2.11) se procede de la siguiente forma:

$$\Delta\omega = G\Delta V \qquad G = \frac{\Delta\omega}{\Delta V} \qquad G = \frac{3{,}13}{5{,}5} \quad \boxed{\text{G=0.56}} \tag{2.7}$$

$$V(t) = \Delta\omega H \quad H = \frac{V(t)}{\Delta\omega} \quad H = \frac{1{,}71}{9{,}91} \quad \boxed{\text{H=0.43}} \tag{2.8}$$

Para su posterior uso en las ecuaciones de ajuste del controlador se obtienen además gráficamente los siguientes datos:

$$K_a = \frac{3{,}13}{5{,}5} = 0{,}56 \qquad t_1 = 0{,}22 \qquad t_2 = 0{,}61 \qquad T_p = 0{,}58 \qquad T_0 = 0{,}03 \tag{2.9}$$

Donde:

- K_a es la ganancia (salida/entrada).
- t_1 es el tiempo para el 28.3 % del valor final de la salida del servomotor.
- t_2 es el tiempo para el 63.2 % del valor final de la salida del servomotor.
- T_p es $1.5{\cdot}(t_2 - t_1)$
- T_0 es $(t_2 - T_p)$

Dinámica de la bola en el interior del aro

Una vez analizada la dinámica del aro, se procede con el análisis de la dinámica de la bola. El movimiento de la bola en el interior del aro se corresponde con el comportamiento de un sistema de segundo orden. Para obtener los parámetros del modelo se emplea un método de aproximación gráfica. En la Figura 2.13, se aprecia la respuesta a una entrada escalón de un sistema de este tipo [64–68].

A partir del gráfico de la respuesta del sistema a una entrada escalón se pueden obtener las siguientes características:

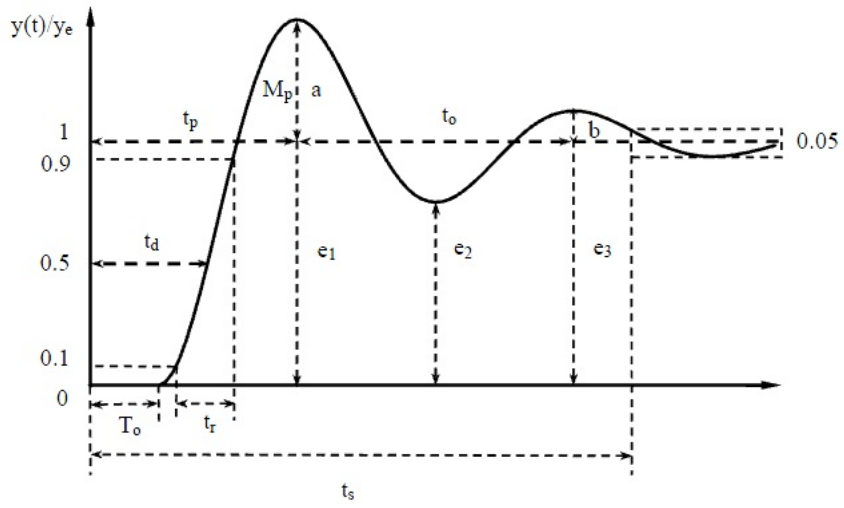


Figura 2.13. Respuesta de un sistema de segundo orden a una entrada escalón.

- y_e es el valor de la respuesta en el estado estacionario.
- $\frac{b}{a}$ es la razón de amortiguamiento, definida como el cociente entre dos sobre-elongaciones consecutivas del mismo signo.
- t_0 es el seudo-período de oscilación, definido como el tiempo transcurrido entre dos sobre-elongaciones consecutivas.
- M_p es la máxima sobre-elongación en (%) respecto al estado estacionario.
- t_p es el instante de máxima sobre-elongación, referido al instante en que se produjo el cambio en la entrada.
- t_d es el tiempo de retardo, definido como el instante en que la respuesta alcanza el 50 % de su valor final.
- a_0 es el área con signo, encerrada por la respuesta respecto a su valor final.

Un sistema de segundo orden se puede describir por la siguiente función de transferencia normalizada:

$$F(s) = \frac{K_a \omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2} \tag{2.10}$$

Donde:

- δ $(0 < \delta < 1)$ es el coeficiente de amortiguamiento.
- ω_n es la frecuencia natural.
- K_a es la ganancia del sistema.

En la Tabla 2.2 se muestra la relación entre las características de la respuesta a una entrada escalón de la Figura 2.13 y los parámetros del modelo 2.10.

y_e	K_a
$\frac{b}{a}$	$e^{-\frac{2\delta\pi}{\sqrt{1-\delta^2}}}$
t_0	$\frac{2\pi}{\omega_n\sqrt{1-\delta^2}}$
M_p	$100e^{-\frac{\delta\pi}{\sqrt{1-\delta^2}}}$
t_p	$T_0 + \frac{\pi}{\omega_n\sqrt{1-\delta^2}}$
t_d	$T_0 + \frac{0{,}2570\delta^2+0{,}3673\delta+1{,}0478}{\omega_n}$
a_0	$\left(\frac{2\delta}{\omega_n} + T_0\right)K$

Tabla 2.2. Relación de la respuesta con su función de transferencia.

Los diversos métodos de aproximación gráfica para un modelo de este tipo hacen uso de las relaciones de la Tabla 2.2. Todos ellos coinciden en que la ganancia K_a queda unívocamente determinada como el cociente entre el cambio observado en la salida y el cambio provocado en la entrada del proceso. No ocurre así con los otros parámetros del modelo, el coeficiente de amortiguamiento (δ), la frecuencia natural (ω_n) y el retardo (T_0), para los que se proponen diferentes métodos que se relacionan a continuación [64, 69, 70]:

- *Aproximación 1*: se basa en hacer que la respuesta del proceso y la del modelo tengan la misma razón de amortiguamiento ($\frac{b}{a}$), el mismo pseudoperíodo de oscilación (t_0), y que alcancen el mismo valor (el 50 % del valor en estado estacionario) en el mismo instante t_d.
- *Aproximación 2*: se basa en hacer que la respuesta del proceso y la del modelo tengan la misma sobre-elongación (M_p), en el mismo instante de tiempo t_p, y que alcancen el mismo valor (el 50 % del valor de estado estacionario) en el mismo instante t_d.
- *Aproximación 3*: se basa en hacer que la respuesta del proceso y la del modelo tengan la misma razón de amortiguamiento ($\frac{b}{a}$), el mismo pseudoperíodo de oscilación (t_0), y la misma área característica (a_0).
- *Aproximación 4*: se basa en hacer que la respuesta del proceso y la del modelo

tengan la misma sobre-elongación (M_p), en el mismo instante de tiempo t_p, y la misma área característica (a_0).

En la Tabla 2.3 se calculan los parámetros del modelo para las diferentes aproximaciones.

Aprox.	δ	ω_n	T_0
1	$\frac{1}{\sqrt{1+\left(\frac{2\pi}{ln\left(\frac{b}{a}\right)}\right)^2}}$	$\frac{2\pi}{t_0\sqrt{1-\delta^2}}$	$t_d - \frac{0{,}2570\delta^2+0{,}3673\delta+1{,}0478}{\omega_n}$
2	$\frac{1}{\sqrt{1+\left(\frac{\pi}{ln\left(\frac{Mp}{100}\right)}\right)^2}}$	$\frac{\frac{\pi}{\sqrt{1-\delta^2}}-\left(0{,}2570\delta^2+0{,}3673\delta+1{,}0478\right)}{t_p-t_d}$	$t_d - \frac{0{,}2570\delta^2+0{,}3673\delta+1{,}0478}{\omega_n}$
3	$\frac{1}{\sqrt{1+\left(\frac{2\pi}{ln\left(\frac{b}{a}\right)}\right)^2}}$	$\frac{2\pi}{t_0\sqrt{1-\delta^2}}$	$\frac{a_0}{K} - \frac{2\delta}{\omega_n}$
4	$\frac{1}{\sqrt{1+\left(\frac{\pi}{ln\left(\frac{Mp}{100}\right)}\right)^2}}$	$\frac{\frac{\pi}{\sqrt{1-\delta^2}}-2\delta}{t_p-\frac{a_0}{K}}$	$\frac{a_0}{K} - \frac{2\delta}{\omega_n}$

Tabla 2.3. Cuatro aproximaciones de los parámetros δ, ω_n y T_0 para $0 < \delta < 1$.

Dependiendo de la aproximación seleccionada, se escogen las fórmulas correspondientes de la Tabla 2.3 y se determinan los valores del coeficiente de amortiguamiento (δ), de la frecuencia natural (ω_n) y del retardo (T_0) del modelo.

Parámetros de la dinámica de la bola

Una vez obtenidos los parámetros de la dinámica del aro, se procede a obtener los parámetros de la dinámica de la bola. En la Figura 2.14 se muestra la función de transferencia que relaciona el ángulo del aro con la desviación angular de la bola respecto a la vertical.

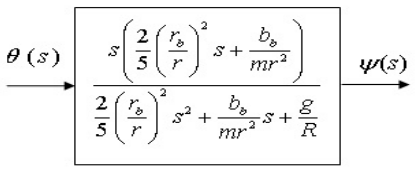


Figura 2.14. Función de transferencia que rige la dinámica de la bola dentro del aro.

De dicha función de transferencia se conocen además de la aceleración de la gravedad ($g = 9{,}81\ m/s^2$), la masa de la bola ($m = 0{,}02812\ kg$), el radio de rodadura de la bola ($r = 0{,}0094\ m$), el radio de la bola ($r_b = 0{,}0095\ m$) y el radio del aro ($R = 0{,}085\ m$). Por lo que solo es necesario calcular o estimar el valor de b_b (coeficiente de fricción de la bola con el aro). Inicialmente se trató de estimar dicho parámetro haciendo girar el aro en un sentido, suministrando un voltaje a la entrada del servomotor y midiendo el ángulo de la bola con respecto a la vertical. Con estos datos y haciendo uso de la herramienta de identificación de sistemas de *MATLAB* se estimó el modelo de la dinámica de la bola dentro del aro, pero se presentaron problemas en la planta por ser muy pequeño dicho coeficiente. Esto se debe a un calentamiento del aro o de la bola debido a la fricción, lo que provoca un funcionamiento incorrecto para dicho experimento. Los resultados obtenidos fueron muy diferentes a los alcanzados en la simulación con los mismos parámetros físicos, por lo que se concluyó que debería estar ocurriendo un funcionamiento anómalo. Posteriormente revisando entre las pruebas que proponen los fabricantes de la planta y el artículo del creador del sistema bola y aro [54–56], este experimento no aparece entre los que ellos realizan, por lo que es posible que sea normal un mal funcionamiento de la planta para dicho experimento. Por lo tanto se decidió realizar la identificación tomando datos de otros experimentos.

Experimento 1: Sistema de segundo orden

El Experimento 1 consiste en poner la planta en funcionamiento y con el aro detenido, variar el ángulo de desviación de la bola con la vertical hasta $-35°$ aproximadamente y liberarla, registrando el movimiento oscilatorio de la bola alrededor de la vertical [5]. Este comportamiento es análogo al presentado anteriormente y se corresponde con la dinámica de un sistema de segundo orden.

Cálculo de b_b con datos de la planta real (Experimento 1)

Se realizó el Experimento 1 y se obtuvieron los datos a través de la tarjeta de adquisición de datos y la aplicación desarrollada en *LabVIEW*. Estos datos fueron introducidos en el espacio de trabajo de *MATLAB* filtrados y se dibujó el correspondiente

gráfico. La Figura 2.15 muestra el resultado obtenido que representa la dinámica de la bola en el interior del aro para la planta real.

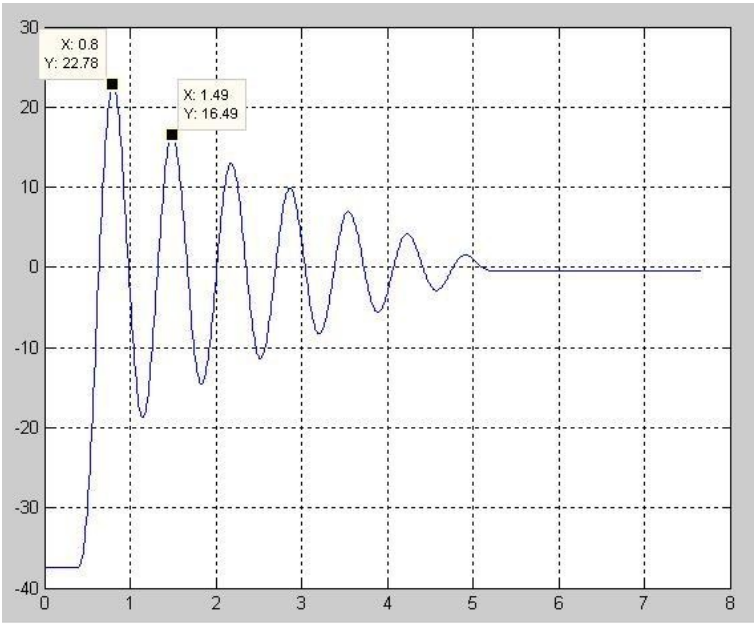


Figura 2.15. Característica de la dinámica de la bola en el aro para la planta real.

Con estos datos se procede a calcular analíticamente el coeficiente b_b, para luego estimarlo con los mismos datos y establecer una comparación a modo de validación. A partir de la gráfica obtenida se pueden calcular los parámetros del modelo del sistema para la planta real. Haciendo uso de lo expuesto en la Sección 2.1.1.2 y estableciendo la analogía entre el comportamiento dinámico de la bola en el interior del aro (Figura 2.15) y la respuesta a una entrada escalón de un sistema de segundo orden (Figura 2.13), se pueden determinar los valores que se muestran en 2.11.

$$a = 22{,}78^\circ \quad \rightarrow \quad t_a = 0{,}8\ s \qquad b = 16{,}49^\circ \quad \rightarrow \quad t_b = 1{,}49\ s \tag{2.11}$$

Con estos valores se puede calcular t_0 como se muestra en 2.12.

$$t_0 = t_b - t_a \qquad \boxed{t_0 = 0{,}69\ s} \tag{2.12}$$

Partiendo de los datos anteriores (b,a) y haciendo uso de las ecuaciones mostradas en la Tabla 2.2 se calculan los parámetros del modelo (2.10) y se obtienen los resultados

mostrados en 2.13.

$$\frac{b}{a} = e^{-\frac{2\delta\pi}{\sqrt{1-\delta^2}}} \quad \rightarrow \quad \boxed{\delta = 0{,}0512} \tag{2.13}$$

Conocidos δ, t_0 y con la Ecuación 2.14 se obtiene $\omega_n = 9{,}114$.

$$t_0 = \frac{2\pi}{\omega_n\sqrt{1-\delta^2}} \quad \rightarrow \quad \boxed{\omega_n = 9{,}114} \tag{2.14}$$

Sustituyendo los valores obtenidos con las ecuaciones 2.13 y 2.14, el denominador del modelo 2.10 se muestra en la Ecuación 2.15.

$$s^2 + 2\delta\omega_n s + \omega_n^2 \qquad\qquad s^2 + 0{,}9332s + 83{,}06 \tag{2.15}$$

El denominador de la función de transferencia de la dinámica de la bola en el interior del aro presentado en la Figura 2.14 se puede transformar en la Ecuación 2.16.

$$s^2 + \frac{b_b}{mr^2\left(\frac{2}{5}\left(\frac{r_b}{r}\right)^2 + 1\right)}s + \frac{g}{R\left(\frac{2}{5}\left(\frac{r_b}{r}\right)^2 + 1\right)} \tag{2.16}$$

Se procede a calcular el coeficiente b_b a partir de los datos conocidos y los obtenidos en (2.15). Igualando los términos que acompañan a s en (2.15) y (2.16) se obtiene el resultado mostrado en 2.17.

$$\boxed{b_b = 3{,}26\text{x}10^{-6}} \tag{2.17}$$

A modo de comprobación se calcula el término independiente sustituyendo los valores conocidos, para comprobar si se acerca al del modelo.

$$\frac{g}{R\left(\frac{2}{5}\left(\frac{r_b}{r}\right)^2 + 1\right)} = \frac{9{,}81}{1{,}408 \cdot 0{,}085} = 81{,}96 \tag{2.18}$$

$$\boxed{81{,}96 \approx \omega_n^2 = 83,06} \tag{2.19}$$

Como se puede apreciar, el resultado es aceptable, ya que está en el orden de magnitud esperado y además el término independiente es bastante aproximado.

Estimación de b_b con datos de la planta real (Experimento 1)

A modo de comprobación se procede entonces a obtener el mismo coeficiente empleando los datos obtenidos con el Experimento 1, pero ahora empleando la herramienta de identificación de sistemas de *MATLAB*. Se realiza nuevamente el experimento para registrar nuevos datos que serán empleados en la validación del modelo obtenido. En la Figura 2.16 se observan los diferentes modelos obtenidos como resultado del uso de varias variantes en el proceso de identificación.

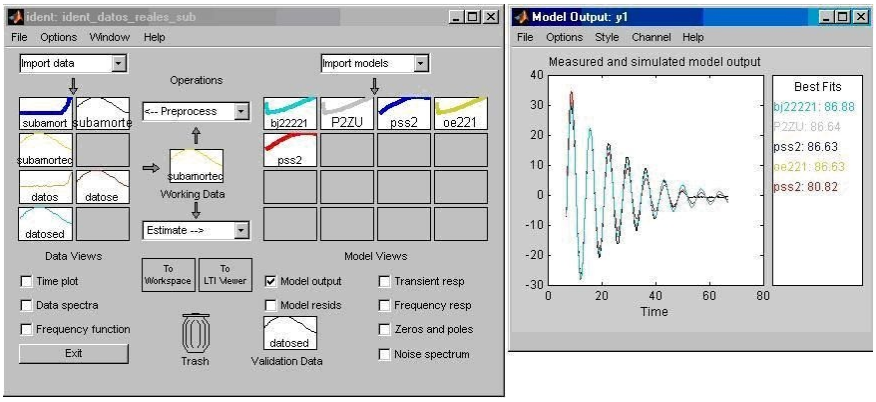


Figura 2.16. Modelos obtenidos con las diferentes variantes de estimación.

A partir de los modelos obtenidos se procede a calcular el coeficiente de fricción, se selecciona el modelo denominado *pss2*, resultante de aplicar la herramienta *State-space* con dos polos y un cero. Su función de transferencia se muestra en la Ecuación 2.20.

$$\frac{-0{,}441s + 0{,}4401}{s^2 - 1{,}982s + 0{,}99} \tag{2.20}$$

Para obtener el valor del coeficiente b_b se igualan los coeficientes del modelo obtenido a partir del uso de la herramienta de identificación de sistemas de *MATLAB* y los coeficientes de la función de transferencia del bola y aro, obteniéndose:

$$\frac{b_b}{mr^2\left(\frac{2}{5}\left(\frac{r_b}{r}\right)^2 + 1\right)} = 1,982 \quad \rightarrow \quad \boxed{b_b = 6{,}92\text{x}10^{-6}} \tag{2.21}$$

Este resultado no es bueno, ya que el signo del término que acompaña a s en el denominador es negativo y no se corresponde con el modelo, además, el término independiente no se aproxima al resultado calculado, lo único que puede tenerse en

cuenta, es que la magnitud del exponente (x10^{-6}), es del orden esperado. Se decide entonces realizar otro experimento en aras de poder determinar una mejor aproximación del coeficiente de fricción.

Experimento 2: Ceros de transmisión

Como se explicó en la Sección 2.1.1.2, este experimento consiste en suministrar una señal sinusoidal al servomotor con una frecuencia determinada, que es específica para cada planta en particular. Con dicha señal a la entrada (voltaje aplicado al motor) se obtiene como salida que la bola no se desplaza con respecto al aro, o sea, ambos oscilan a dicha frecuencia, en otras palabras, la frecuencia de la señal de entrada es transmitida a la salida. Esta frecuencia depende del radio del aro (R).

Estimación del coeficiente b_b con datos de la planta real (Experimento 2)

Los datos del Experimento 2 se obtienen de forma similar al experimento anterior (usando el *LabVIEW* y la tarjeta de adquisición de datos). Estos datos se cargan en el espacio de trabajo de *MATLAB* y luego se filtran (ver Código 2.2).

```
load pruebas.txt        %Se cargan los datos
theta2=pruebas(:,1);    %Se define el ángulo del aro
chi2=pruebas(:,3);      %Se define el ángulo de la bola
t=pruebas(:,4);         %Se define el tiempo
t=t-t(1);               %Se establece el instante inicial
chi1=filord1p(t,chi2,0.1);   %Se filtra al ángulo de la bola
theta1=filord1p(t,theta2,0.1);   %Se filtra el ángulo del aro
```

Código 2.2. Instrucciones para cargar los datos al espacio de trabajo.

A continuación se utiliza la herramienta de identificación de sistemas de *MATLAB* y se inicia el proceso de identificación. A partir del conjunto de datos se escogen dos secciones diferentes de los datos: uno para la estimación del modelo y otro para la validación del mismo. Después de tratar los datos, al igual que en los casos anteriores, se procede a iniciar el proceso de identificación como se muestra en la Figura 2.17.

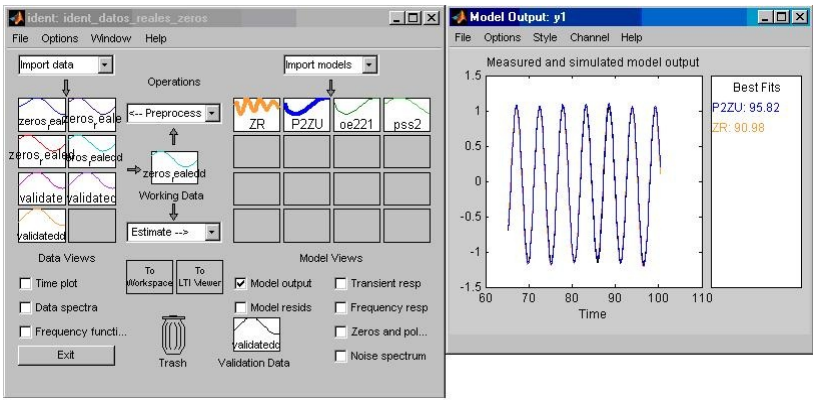


Figura 2.17. Proceso de identificación para el Experimento 2 con datos reales.

El modelo que más se ajusta se obtiene con la herramienta *Process Model* para dos polos y un cero. Con el objetivo de obtener los mejores resultados se fijó el parámetro conocido *Zeta=0.0512*. Posteriormente se transfiere al espacio de trabajo de *MATLAB* obteniéndose la función de transferencia que se muestra en la Ecuación 2.22.

$$\frac{24{,}33s - 206{,}9}{s^2 + 0{,}9078s + 78{,}62} \tag{2.22}$$

A partir del resultado obtenido con la herramienta de identificación e igualando los coeficientes con los de la función de transferencia del sistema bola y aro y despejando b_b se obtiene:

$$\frac{b_b}{mr^2\left(\frac{2}{5}\left(\frac{r_b}{r}\right)^2 + 1\right)} = 1,982 \quad \rightarrow \quad \boxed{b_b = 3{,}17\text{x}10^{-6}} \tag{2.23}$$

El resultado obtenido puede catalogarse de aceptable ya que tanto el coeficiente estimado, como el término independiente, son bastante aproximados a los cálculos iniciales.

Por último se realiza el proceso de estimación del coeficiente b_b con el Experimento 2 pero empleando en este caso, datos de la simulación implementada en *"Easy Java Simulations"* (*EJS*) [71, 72]. Es necesario señalar que esta simulación está basada en la planta piloto real, por lo que las propiedades físicas de la simulación (radio y masa de la bola, radio y masa del aro, etc) son las mismas que las de la planta real.

Estimación de b_b con datos de la simulación en *EJS* (Experimento 2)

EJS es una herramienta desarrollada en *Java* que es totalmente gratuita y de código abierto. Además está especialmente diseñada para crear simulaciones interactivas, obteniéndose muy buenos resultados en cuanto a la representación gráfica y al funcionamiento de modelos dinámicos sin necesitad de tener muchos conocimientos de programación.

Con los parámetros físicos de la planta, y utilizando *EJS* se desarrolla una aplicación para realizar diferentes experimentos con un coeficiente de fricción b_b arbitrario (suponiéndolo desconocido). Poniendo en marcha la simulación para el experimento de ceros de transmisión se obtienen los datos que se introducen al espacio de trabajo de *MATLAB* a través de las instrucciones mostradas en el segmento de Código 2.3.

```
load chi.txt;    %Se cargan los datos de la simulación
load theta.txt;
t=tau(:,2);      %Se define el vector tiempo
chi=chi(:,2);    %Valores del ángulo de la bola
theta=tau(:,1); %Valores del ángulo del aro
chi=chi(1:length(tau));
```

Código 2.3. Instrucciones para cargar los datos al espacio de trabajo.

A continuación se inicia el proceso de identificación utilizando la herramienta de identificación de sistemas de *MATLAB*. De este conjunto de datos se selecciona una parte para el proceso de identificación y otra parte para la validación, como se muestra en la Figura 2.18.

Con el objetivo de obtener los mejores resultados posibles, se fija uno de los parámetros conocidos $Zeta = 0{,}0512$. De los modelos obtenidos se selecciona el de mayor porcentaje de aceptación. Dicho modelo se presenta en el espacio de trabajo de *MATLAB*. La función de transferencia que se obtiene, se muestra en la Ecuación 2.24.

$$\frac{8{,}569s + 61{,}74}{s^2 + 0{,}9228s + 81{,}23} \tag{2.24}$$

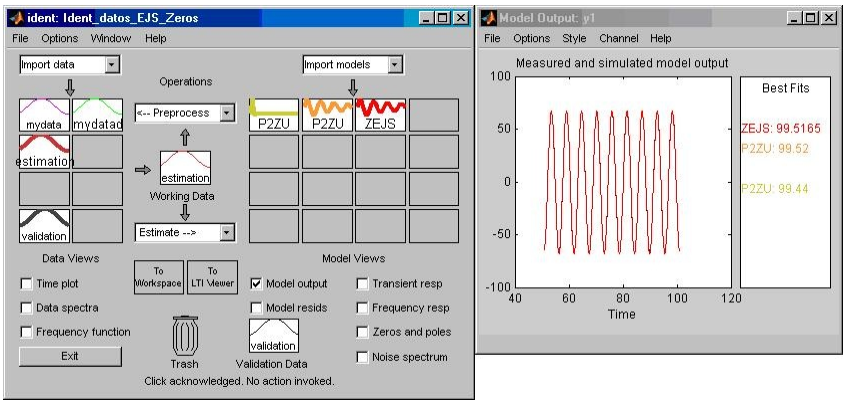


Figura 2.18. Proceso de identificación para el Experimento 2 con datos de *EJS*.

A partir del resultado obtenido con la identificación e igualando los coeficientes con la función de transferencia del sistema de bola y aro y despejando, se obtiene el coeficiente b_b que se muestra en 2.25.

$$\frac{b_b}{mr^2\left(\frac{2}{5}\left(\frac{r_b}{r}\right)^2+1\right)} = 0,9228 \quad \rightarrow \quad \boxed{b_b = 3{,}22\text{x}10^{-6}} \tag{2.25}$$

El resultado obtenido puede catalogarse de aceptable ya que al igual que en el caso anterior, tanto el coeficiente estimado como el término independiente, se aproximan en buena medida a los cálculos iniciales.

2.1.1.4 Análisis de los resultados obtenidos

A continuación se procede a establecer una comparación entre los resultados obtenidos con los diferentes métodos y datos empleados. La Tabla 2.4 muestra un resumen de los resultados para los diferentes experimentos, métodos y datos empleados en la obtención del coeficiente b_b.

Experimento	Segundo Orden		Ceros de Transmisión	
Datos	Datos Reales		Datos Reales	*EJS*
Método	Cálculo	*Ident*	*Ident*	*Ident*
Coeficiente b_b	$3{,}26\text{x}10^{-6}$	$6{,}92\text{x}10^{-6}$	$3{,}17\text{x}10^{-6}$	$3{,}22\text{x}10^{-6}$

Tabla 2.4. Resultados obtenidos para el coeficiente b_b.

Como se puede apreciar, los mejores resultados se obtuvieron con el experimento de

los ceros de transmisión, esto era de esperar, pues es el experimento que mejor funciona tanto para la planta real como en la simulación. Teniendo en cuenta los resultados obtenidos se puede decir que el coeficiente b_b de la planta real se encuentra entre 3,17x10^{-6} y 3,26x10^{-6}. Por lo que se escoge el valor 3,22x10^{-6}.

2.1.1.5 Diseño del controlador

Una vez conocidos los parámetros del sistema se procede a obtener los parámetros del controlador que se empleará en el lazo de control. Lo primero que se debe hacer es seleccionar el tipo de controlador, dependiendo de los conocimientos básicos sobre controladores *PID* (algoritmos y estructuras de control) y la experiencia que se tenga sobre el proceso a controlar. En este caso el proceso a controlar es el servomotor del sistema bola y aro. Haciendo uso de la Ecuación 2.9, se calcula la razón $\frac{T_0}{T_p} = 0{,}051$. Lo que indica que se está ante un proceso con constante de tiempo dominante y en consecuencia "fácil de controlar". El controlador recomendado para este tipo de sistemas es el controlador *PI* [73]. Solo queda entonces determinar los parámetros de control adecuados a este proceso y para ello se acude a las fórmulas de sintonía del *PID*. Existen muchos métodos de sintonía de controladores, algunos se basan en experimentos en lazo cerrado. Uno de los más conocidos es el método de la oscilación mantenida propuesto por Ziegler y Nichols [74]. Otro muy popular y bien conocido es el método del relé, propuesto por Åström y Hägglund [65, 75], que es una forma indirecta de automatizar el método de la oscilación mantenida. Bajo el nombre *AMIGO* (*Approximate M constrained Integral Gain Optimization*) Åström y Hägglund [76] propusieron unas fórmulas de sintonía que pretenden acabar con la hegemonía de las fórmulas de Ziegler y Nichols.

En este caso, para aprovechar el experimento en lazo abierto (respuesta a una entrada escalón) realizado con la planta real y mostrado en la Figura 2.13, hemos seleccionado los métodos más comunes que se pueden aplicar a experimentos en lazo abierto, entre los que se encuentran:

- Las fórmulas de Ziegler y Nichols para experimentos en lazo abierto [74], por ser las más representativas y porque dan buenos resultados para este tipo de

procesos, aunque en este caso, la razón $\frac{T_0}{T_p} = 0{,}051$ esté ligeramente fuera del rango de aplicación de estas fórmulas ($0{,}1 < \frac{T_o}{T_p} < 1$).

- Otras fórmulas que permiten más posibilidades son las propuestas por González en [77]. Con ellas se pretende que el sistema en lazo cerrado tenga un respuesta para cambio en el punto de consigna con las características de un sistema de segundo orden asociadas a su coeficiente de amortiguamiento (δ). El usuario tiene libertad para fijar el valor deseado de cualquier especificación (máxima sobre-elongación o razón de amortiguamiento) directamente relacionada con δ.
- Otro método que se puede emplear es el de Cohen y Coon conocido como método de la curva de reacción [78]. Al igual que los anteriores, los parámetros se obtienen de la respuesta del sistema a una entrada escalón en lazo abierto. Es una variación del método de Ziegler-Nichols que introduce algunas mejoras a las limitaciones de éste (sensibilidad a las variaciones de $\frac{T_o}{T_p}$).
- También son muy reconocidos los métodos basados en criterios integrales. El primero de estos métodos fue desarrollado por López [79]. El método define una función de coste que depende del error y del tiempo. Mientras menor sea el valor de esta función, mejor será el desempeño del sistema de control. Los criterios de desempeño son: Integral del error absoluto (IAE) e Integral del error absoluto por el tiempo ($ITAE$).
- Otro método muy utilizado es el propuesto por Chien-Hrones-Reswick, conocido como el método CHR [80]. Este método surge a partir de modificaciones sobre el método Ziegler-Nichols y tiene la ventaja de tener una mejor respuesta ante perturbaciones.
- Wang-Juang-Chan [81] proponen tres fórmulas para cada uno de los parámetros del controlador *PID*. Utiliza como criterio de desempeño la optimización *ITAE*. Este método está orientado a la estructura de control *PID-Ideal*.

En la Tabla 2.5 se muestran las diferentes fórmulas de obtención de los parámetros del controlador *PI* empleando estos métodos.

En la Tabla 2.6 se muestran los resultados obtenidos con datos de la planta real, usando cada uno de estos métodos. Los resultados obtenidos para los diferentes métodos

	Método	K_p	T_i
1	Ziegler-Nichols	$0.9\frac{T_p}{K_a T_0}$	$\frac{T_0}{0{,}3}$
2	González	$\frac{\omega_n^2 T_p T_0}{2K_a}$	$\frac{T_0}{2}$
3	Cohen-Coon	$\frac{T_p}{K_a T_0}\left(0{,}9+\frac{T_0}{12T_p}\right)$	$\frac{T_0(30T_p+3T_0)}{9T_p+20T_0}$
4	López-Miller (IAE)	$\frac{0{,}7}{Ka}\left[\frac{T_o}{T_p}\right]^{-0{,}861}$	$1{,}02-0{,}323\left[\frac{T_o}{T_p}\right]$
5	López-Miller (ITAE)	$\frac{0{,}586}{K_a}\left[\frac{T_o}{T_p}\right]^{-0{,}916}$	$1{,}03-0{,}165\left[\frac{T_o}{T_p}\right]$
6	Chien-Hrones-Reswick	$\frac{0{,}6T_p}{KT_0}$	T_p
7	Wang-Juang-Chan	$\frac{\left(0{,}7303+\frac{0{,}5307T_p}{T_0}\right)(T_p+0{,}5T_0)}{K_a(T_p+T_0)}$	$T_p+0{,}5T_0$

Tabla 2.5. Ajuste del controlador PI para los diferentes métodos.

pueden ser considerados aceptables teniendo en cuenta el orden de magnitud y los valores tanto de K_p como de T_i.

	1	2	3	4	5	6	7
K_p	34.52	0.15	31.21	16.01	15.77	20	19.15
T_i	0.1	0.015	0.09	1.0	1.021	0.58	0.595

Tabla 2.6. Resultados obtenidos para la planta real con los diferentes métodos.

El único método con el que se obtienen valores poco razonables es con el método 2 (González). Para tener más certeza en los resultados una opción sería calcular estos parámetros haciendo uso de los métodos existentes en lazo cerrado. Para ello habría que realizar un nuevo experimento, registrar las variables y realizar los cálculos. Esto se les podría proponer a los estudiantes a modo de comprobación durante el desarrollo de una práctica de laboratorio.

2.1.2 Descripción del sistema bola y plato

El sistema bola y plato está compuesto por una esfera que puede rodar sin deslizarse encima de un plato circular. El plato se encuentra unido a uno o varios motores de corriente continua que le permiten pivotar en el eje x y en el eje y. El objetivo del sistema es controlar la posición de la bola modificando el ángulo de inclinación del plato. Este sistema puede ser considerado como una extensión bidimensional del

sistema de bola y viga [82]. Los experimentos relacionados con el sistema de bola y plato se consideran desafiantes por la complejidad inherente al mismo. Debido a esto, este sistema ha despertado un especial interés académico en los últimos años y ha sido usado para el estudio de varios tópicos relacionados con la ingeniería de control tales como: control no lineal, análisis mecánico, identificación paramétrica, control inteligente, procesamiento de imágenes, seguimiento de trayectorias, entre otros [83]. Además, las plantas físicas que representan este sistema, pueden presentar influencias negativas en su comportamiento debido a la fricción entre la bola y el plato, retardos en las mediciones, etc. La Figura 2.19 muestra la planta real basada en el sistema bola y plato. Esta planta piloto ha sido construida en el Departamento de Energía Eléctrica, Sistemas y Automatización de la Universidad de Gante en Bélgica.

Figura 2.19. Planta real basada en el sistema de bola y plato.

El sistema bola y plato se emplea en la industria aeronáutica para el desarrollo de simuladores de vehículos aéreos y terrestres. La Figura 2.20 muestra algunos ejemplos de este tipo de simuladores.

Figura 2.20. Ejemplos de aplicaciones del sistema bola y plato.

2.1.2.1 Modelo del sistema bola y plato

El bola y plato es un sistema no lineal multi-variable. Para obtener el modelo matemático de este sistema se tienen en cuenta las suposiciones que se relacionan a continuación:

- La bola siempre está en contacto con el plato.
- La bola rueda sobre el plato sin deslizamiento.
- La bola es una esfera completamente simétrica y homogénea.
- Se desprecian todas las fricciones.

La Figura 2.21 muestra el esquema del sistema de bola y plato. El modelo del sistema se obtiene empleando las variables como las coordenadas generalizadas en las ecuaciones de *Euler-Lagrange*. Después de varias transformaciones y sustituciones, se obtienen las ecuaciones que describen el comportamiento dinámico del sistema como se muestra en las Ecuaciones 2.26 y 2.27 [84–86].

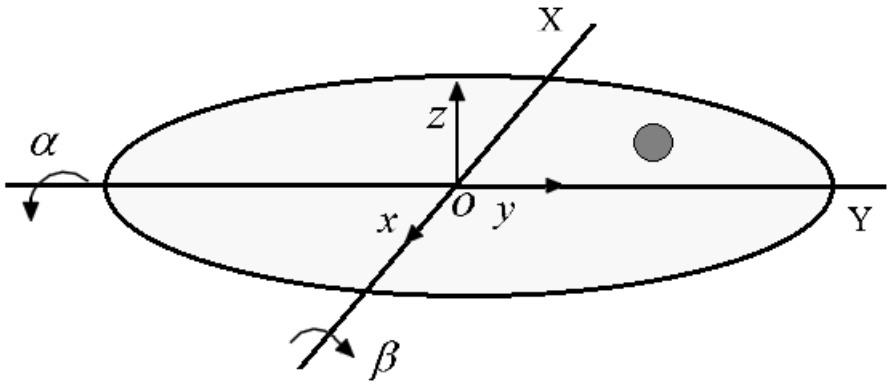


Figura 2.21. Esquema del sistema bola y plato.

$$\left(m_b + \frac{I_b}{r_b^2}\right)\ddot{x}_b - m_b\left(x_b\dot{\alpha}^2 + y_b\dot{\alpha}\dot{\beta}\right) + m_b g sin\alpha = 0 \tag{2.26}$$

$$\left(m_b + \frac{I_b}{r_b^2}\right)\ddot{y}_b - m_b\left(y_b\dot{\beta}^2 + x_b\dot{\alpha}\dot{\beta}\right) + m_b g sin\beta = 0 \tag{2.27}$$

Estas ecuaciones describen el movimiento de la bola en el plato y muestran como afecta la aceleración de la bola en relación con el ángulo y la velocidad angular de inclinación del plato. Las variables que describen el modelo son las siguientes:

- m_b, masa de la bola.
- r_b, radio de la bola.
- g, aceleración de la gravedad.
- I_b, momento de inercia de la bola.
- (x_b, y_b), posición de la bola con respecto al plato.
- (α,β), ángulos de inclinación del plato.

2.1.2.2 Dinámica del sistema bola y plato

La dinámica del sistema bola y plato es muy rica y compleja porque tiene un comportamiento oscilatorio que varía constantemente. Este sistema puede emplearse para implementar las estrategias de control de posición del plato y de posición de la bola. Para el experimento de control de posición de la bola, el sistema se pude ver como un sistema *SITO* semejante al sistema bola y aro descrito anteriormente. La entrada es la referencia de posición de la bola y las dos salidas son el ángulo de inclinación del plato y la posición de la bola. Una complejidad añadida que debe tenerse en cuenta en

este sistema es la relación entre los ángulos de inclinación del plato y los ángulos de los servomotores [87]. Esto está estrechamente vinculado a las características físicas de la planta piloto con la que se desee trabajar. El momento de inercia de una esfera sólida se calcula como se muestra en la Ecuación 2.28.

$$I_b = \frac{2}{5} m_b r_b^2 \tag{2.28}$$

Sustituyendo la Ecuación 2.28, las Ecuaciones 2.26 y 2.27 se pueden escribir como:

$$m_b \left[\frac{5}{7} \ddot{x}_b - \left(x_b \dot{\alpha}^2 + y_b \dot{\alpha} \dot{\beta} \right) + g sin\alpha \right] = 0 \tag{2.29}$$

$$m_b \left[\frac{5}{7} \ddot{y}_b - \left(y_b \dot{\beta}^2 + x_b \dot{\alpha} \dot{\beta} \right) + g sin\beta \right] = 0 \tag{2.30}$$

Asumiendo que para ángulos pequeños de inclinación del plato $\alpha \ll 0$ y $\beta \ll 0 \Longrightarrow$ $sen\alpha \simeq \alpha$ y $sen\beta \simeq \beta$, y que para valores pequeños de velocidad angular del plato $\dot{\alpha} \ll 0$ y $\dot{\beta} \ll 0 \Longrightarrow \dot{\alpha}\dot{\beta} \simeq 0$, $\dot{\alpha}^2 \simeq 0$, $\dot{\beta}^2 \simeq 0$. De las Ecuaciones 2.29 y 2.30 utilizando estas aproximaciones se obtienen las Ecuaciones 2.31 y 2.32.

$$\frac{5}{7} \ddot{x}_b + g\alpha = 0 \tag{2.31}$$

$$\frac{5}{7} \ddot{y}_b + g\beta = 0 \tag{2.32}$$

Haciendo uso de este modelo linealizado del sistema y tomando como salida la posición de la bola (x_b,y_b) y como entrada los ángulos de inclinación del plato (α, β), se obtienen las funciones de trasferencia del sistema para cada uno de los ejes como se muestra en las Ecuaciones 2.33, 2.34.

$$P_x(s) = \frac{x_b(s)}{\alpha(s)} = \frac{g}{\frac{5}{7}s^2} \tag{2.33}$$

$$P_y(s) = \frac{y_b(s)}{\beta(s)} = \frac{g}{\frac{5}{7}s^2} \tag{2.34}$$

Las funciones de transferencia que se obtienen para cada uno de los ejes, coinciden con la del sistema de bola y viga como se había mencionado anteriormente en esta

Sección. Por esta razón este sistema se analiza en la literatura como un sistema de bola y viga para cada uno de los ejes.

Control de posición de la bola en un punto del plato

El experimento consiste en controlar la posición de la bola en un punto sobre el plato. Como el plato se encuentra unido a varios servomotores, el control de la posición de la bola se lleva a cabo por medio del control de realimentación de posición de dichos motores que son los que provocan la inclinación del plato. Para ello se implementan los lazos de control de realimentación como se observa en el diagrama de bloques representado en la Figura 2.22.

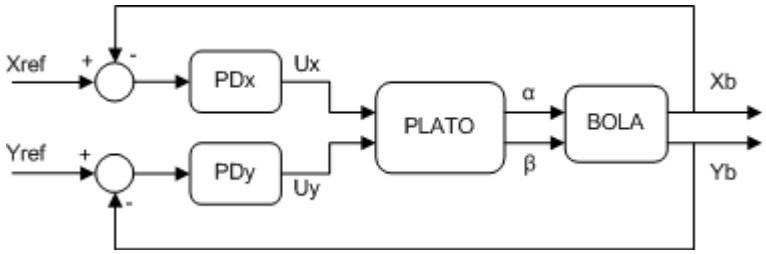


Figura 2.22. Diagrama de bloques del control de posición de la bola sobre el plato.

A partir del sistema linealizado de las Ecuaciones 2.33 y 2.34 se deduce que x_b depende únicamente de α, mientras que y_b depende únicamente de β. Por lo que el sistema pude ser tratado como dos sistemas independientes que funcionan simultáneamente con dos controladores independientes para controlar las dos coordenadas de posición de la bola.

Los bloques representados en el diagrama son: los controladores proporcional derivativo (PD), tanto para la coordenada x como para la coordenada y; el bloque correspondiente a la dinámica del plato y el bloque correspondiente a la dinámica de la bola. Mientras que las variables son las siguientes:

- X_{ref}, la coordenada x_b de la posición de referencia de la bola.
- Y_{ref}, la coordenada y_b de la posición de referencia de la bola.
- U_x, la señal de control coordenada x_b de la bola.
- U_y, la señal de control coordenada y_b de la bola.

2.1.2.3 Planta real del sistema bola y plato

Una vez presentados los aspectos teóricos del sistema bola y plato, se procede a presentar la planta real con la cual se desarrolla el laboratorio remoto [88]. Sus componentes físicos son los siguientes:

- Un plato plástico circular de 57,5 cm de diámetro.
- Una bola de billar de color amarillo claro.
- Seis servo-motores acoplados al plato por medio de seis varillas metálicas.
- Un circuito electrónico cuyo componente principal es un microcontrolador *PIC18F4550* de *Microchip*. Este circuito se conecta al ordenador a través del puerto USB y se usa para controlar los seis motores.
- Una cámara de vídeo (*Cam Logitech Pro 4000*) conectada al ordenador a través del puerto USB [89].

El microcontrolador se programa independientemente de la aplicación que controla la planta real. El programa que ejecuta el microcontrolador recibe las posiciones deseadas para los motores y les envía a éstos las órdenes correspondientes. Además mediante la cámara se obtienen las imágenes que son procesadas para obtener la posición de la bola sobre el plato. La posición de la bola, obtenida con la cámara es la variable que se usa como realimentación y que cierra el lazo de control.

El plato es de color negro para evitar reflejos y sombras que signifiquen perturbaciones en el tratamiento de imágenes. Por esta misma razón la bola es de color amarillo claro de forma tal que se pueda distinguir con claridad cuando esté sobre el plato.

La cámara se debe calibrar para poder usarla como sensor de posición de la bola. Para ello se usó la herramienta *Camera Calibration Toolbox for Matlab* desarrollada por Bouguet [90]. Además se implementó una aplicación en Visual C# usando la *Transformada de Hough* que se encuentra disponible en la librería *OpenCV* [91]. La Figura 2.23 muestra a la izquierda la imagen obtenida por la cámara y a la derecha el procesamiento de la imagen para la detección de bordes y la posición de bola. En la imagen de la izquierda la cruz amarilla representa los ejes de coordenadas mientras que la cruz y el círculo azul representan la posición de la bola y la cruz y el círculo rojos representan la referencia de posición de la bola.

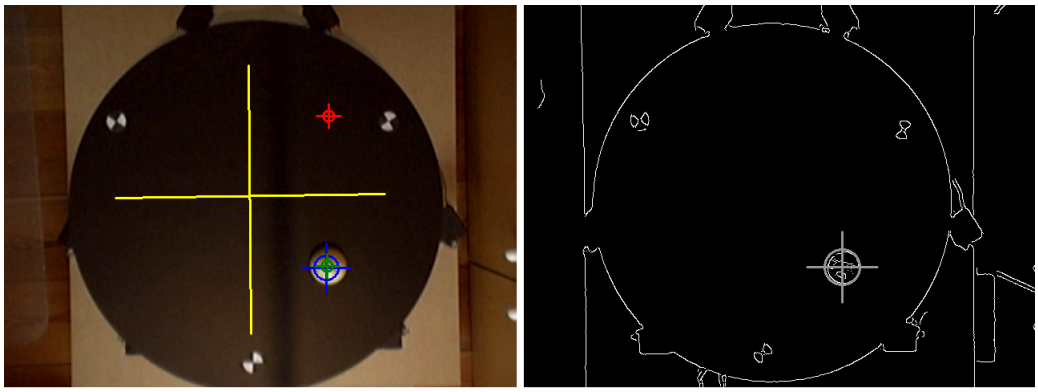


Figura 2.23. Imagen obtenida por la cámara y detección de la bola en el plato.

2.1.2.4 Ajuste de los controladores

Para controlar la planta real se propusieron dos algoritmos: un controlador *PD* y un controlador lineal cuadrático (*LQR*). El controlador *PD* se sintonizó con la herramienta *FR-tool* desarrollada en *MATLAB* [92]. Mientras que para el controlador *LQR* también se sintonizó usando la función *LQR* del "*Control System Toolbox*" de *MATLAB* [93] y luego se transformó en un controlador *PD* equivalente. En la Tabla 2.7 se muestran los valores obtenidos para ambos controladores.

Controlador	K_p	K_d
PD	-0.750	-0.514
LQR	-0.897	-0.580

Tabla 2.7. Parámetros obtenidos para los controladores *PD* y *LQR*.

Como se puede apreciar, los valores de las ganancias K_p y K_d están en el mismo orden de magnitud y son bastante similares. La Figura 2.24 muestra la respuesta de la planta real a un salto en escalón en la entrada de ambos controladores (*PD*-línea continua, *LQR*-línea discontinua). La gráfica de la parte superior de la figura muestra la coordenada x de la posición de la bola en milímetros mientras que la gráfica de la parte inferior de la figura muestra la señal de control (u), que en este caso es el ángulo de inclinación del plato en grados causado a su vez por el movimiento de los motores.

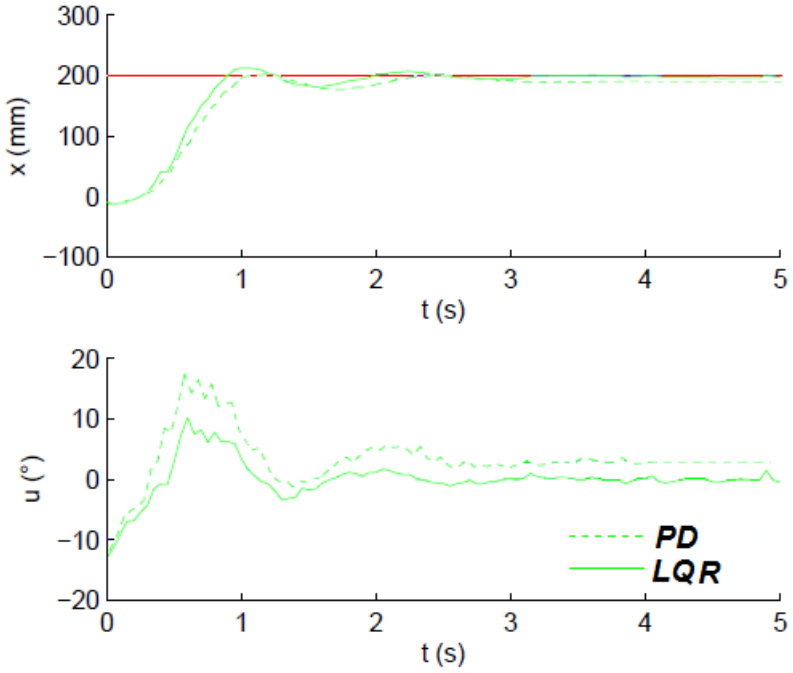


Figura 2.24. Respuesta de la planta real a un paso escalón por la entrada.

2.2 Descripción de las plataformas para la realización de experimentos de robótica

Dentro del estudio de la robótica móvil, los robots se clasifican en vehículos aéreos, terrestres y acuáticos [94]. A su vez, los robots terrestres se clasifican dependiendo del tipo de locomoción que utilizan, en tres grupos: robots de ruedas [95], robots de patas [96] y robots de orugas [97].

Para la realización de experimentos de robótica se han desarrollado dos plataformas: una con robots *Moway* (ruedas) y otra con robots *Surveyor SRV-1* (oruga). Estas plataformas han sido concebidas inicialmente para el desarrollo de experimentos de control de formación de robots móviles. Pero también se pueden utilizar para realizar otro tipo de experimentos, para lo cual solamente es necesario conocer la dinámica de los robots, cambiar la programación de los mismos y retardos en las comunicaciones inalámbricas, que pueden tener una gran influencia en el resultado de los experimentos. A continuación se describen los componentes de ambas plataformas.

2.2.1 Descripción de los robots *Moway*

A pesar del amplio estudio de los robots móviles de patas y de orugas, el desarrollo más significativo se ha dado en el estudio de los robots móviles con ruedas. Una ventaja principal por la que es más conveniente utilizar ruedas como medio locomotor es la facilidad para construirlas; otras ventajas son que sólo es necesario suministrar energía al eje de las ruedas motrices; se puede desplazar un peso mayor que usando patas; requieren menor cantidad de piezas; el control de las ruedas es menos complejo que la actuación de las patas o de las orugas; causan menor desgaste en la superficie en donde se mueven en comparación con las bandas de las orugas. Además, los problemas de balance no presentan gran dificultad, ya que el robot siempre se encontrará en contacto con una superficie, lo cual no siempre sucede con los robots actuados con patas u orugas.

Los *Moway* son pequeños robots móviles de ruedas. Su movimiento se basa en el modelo cinemático de su sistema motriz que le permite desplazarse en un determinado entorno, usando para ello, un sistema de tracción diferencial. La configuración por tracción diferencial se caracteriza porque el movimiento se consigue con dos ruedas acopladas cada una a su propio motor, por lo que el robot puede cambiar de dirección a través de la variación de la velocidad relativa entre las ruedas. De esta forma el robot puede avanzar en línea recta fijando ambos motores a la misma velocidad, también puede girar en una u otra dirección cuando se aplican velocidades diferentes, y se logra que gire sobre su propio eje cuando las velocidades tienen igual magnitud con sentidos opuestos. Por lo que no necesita ningún movimiento adicional de dirección para girar o avanzar. Los robots que usan este tipo de sistemas para su locomoción son conocidos en robótica como robots móviles con ruedas y tracción diferencial o simplemente robot diferencial. La Figura 2.25 muestra el robot *Moway*.

2.2.1.1 Modelo del robot diferencial con ruedas

Para obtener el modelo cinemático de un robot diferencial se tienen en cuenta varias suposiciones de diseño y operación: el robot se mueve sobre una superficie plana;

Figura 2.25. Robot *Moway*.

el deslizamiento en las ruedas izquierda y derecha es despreciable; el robot es rígido y no cuenta con partes flexibles; el robot tiene restricciones no holonómicas (su velocidad lineal no puede ser perpendicular a las ruedas) [98–100]. Además se considera que el movimiento del robot se realiza en el plano XY describiendo trayectorias circulares (de radio R) cuando gira, tanto a su derecha como a su izquierda. El radio R (ver Figura 2.28) de estas circunferencias varía desde un valor mínimo (cuando el robot gira sobre sí mismo), hasta un radio máximo que tiene valor infinito y es cuando el robot se mueve recto hacia adelante o hacia atrás dependiendo del signo de las velocidades. En la Figura 2.26 se muestra el diagrama del robot móvil, a partir del cuál se obtiene el modelo cinemático [101, 102].

Como se puede apreciar el punto $C(x_c,y_c)$ denota la posición del punto medio del eje que une las dos ruedas, la variable θ describe el ángulo que forma el eje de simetría del robot respecto al eje X positivo, las variables ω_i y ω_d son las velocidades angulares de las ruedas izquierda y derecha respectivamente. La variable $L/2$ es la distancia entre las ruedas y la variable r el radio de éstas. Sobre cada rueda, tanto izquierda como derecha, actúa una fuerza, $\overrightarrow{F}_i$ y $\overrightarrow{F}_d$ que proporcionan las velocidades $\overrightarrow{v}_i$ y $\overrightarrow{v}_d$, respectivamente. En la Figura 2.27.a y 2.27.b se muestran el diagrama de fuerzas y de velocidades respectivamente.

Si se toman como coordenadas generalizadas a (x, y y θ), se determina que la

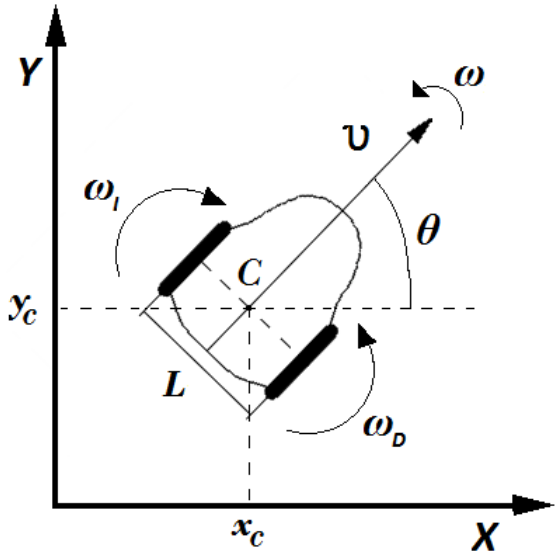


Figura 2.26. Diagrama del robot móvil.

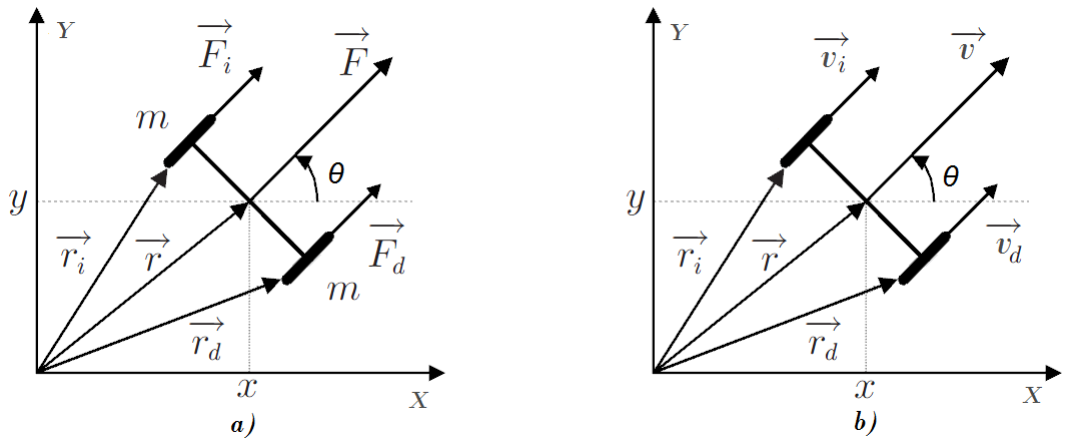


Figura 2.27. a) Diagrama de fuerzas b) Diagrama de velocidades.

posición de los puntos de contacto de cada una de las ruedas con el plano XY está dada por las Ecuaciones 2.35 y 2.36.

$$\overrightarrow{r_i} = (x - l\sin\theta)\,\hat{x} + (y + lcos\theta)\,\hat{y} \tag{2.35}$$

$$\overrightarrow{r_d} = (x + l\sin\theta)\,\hat{x} + (y - lcos\theta)\,\hat{y} \tag{2.36}$$

Tomando m como la masa de cada una de las ruedas, la energía cinética del sistema

viene dada por la Ecuación 2.37.

$$T = \frac{m}{2}\left(|\dot{\vec{r}_i}|^2 + |\dot{\vec{r}_d}|^2\right) = m\left(\dot{x}^2 + \dot{y}^2 + l^2\theta^2\right) \tag{2.37}$$

Después de varias transformaciones y haciendo uso de los multiplicadores de *Euler-Lagrange* [101–104] se obtienen las componentes de la velocidad lineal (ν) en el eje x (ν_x), y (ν_y) y la orientación del robot(θ) como se muestra en las Ecuaciones 2.38, 2.39 y 2.40 respectivamente.

$$\dot{x} = \frac{(w_d + w_i)\, r}{2} cos\theta \tag{2.38}$$

$$\dot{y} = \frac{(w_d + w_i)\, r}{2} sin\theta \tag{2.39}$$

$$\theta = \frac{(w_d - w_i)\, r}{2L} \tag{2.40}$$

Como las coordenadas del punto P (x_c, y_c) denotan la posición del centro de masa de las ruedas de las ecuaciones anteriores se obtiene la magnitud de velocidad lineal de este punto (ν) que viene dada por la Ecuación 2.41.

$$\nu = \sqrt{\dot{x}^2 + \dot{y}^2} = \frac{(w_d + w_i)\, r}{2} \tag{2.41}$$

Además $\dot{\theta}$ es la magnitud de la velocidad angular con la que el eje de simetría del robot rota respecto al sistema de referencia inercial XY, o simplemente la velocidad angular del robot(ω). Por tanto modelo cinemático del robot se puede escribir las ecuaciones representadas en 2.42.

$$\begin{cases} \dot{x}_c = \nu cos\theta \\ \dot{y}_c = \nu sin\theta \\ \quad \dot{\theta} = \omega \end{cases} \tag{2.42}$$

Este modelo solo tiene en cuenta la cinemática del robot. Es necesario destacar que existen otros modelos para este tipo de robots, que también tienen en cuenta la dinámica de los motores en la cuál se incluyen coeficientes que se calculan de manera experimental [105]. Estos modelos tienen en cuenta parámetros tales como: el deslizamiento de las ruedas, las fuerzas externas, las fuerzas de fricción, la inercia de las

ruedas y el centro de masa del robot en un punto distinto al centro de la línea que une las ruedas. Para poder usar estos modelos hay que conocer detalladamente el modelo dinámico de los motores, para lo que es necesario calcular los coeficientes de los que depende esta dinámica. En este caso, para estos robots no es posible usar estos modelos porque no se pueden medir los valores necesarios para implementarlo. Además, estos modelos no aportan muchas variaciones al funcionamiento del robot, por lo que se decidió usar el modelo que tiene solo en cuenta la cinemática del robot, con el que finalmente se obtuvieron buenos resultados.

2.2.1.2 Control de posición del robot

El objetivo del control es que el robot vaya desde la posición actual definida por su centro de masa C(x_c, y_c), hasta el punto P(x_p, y_p). Existen varias formas de implementar este control, en este caso, se ha escogido una de las más sencillas, que consiste en calcular la distancia (d) y el ángulo (α) entre estos dos puntos. Para ello se emplean las Ecuaciones 2.43.

$$d = \sqrt{(y_p - y_c)^2 + (x_p - x_c)^2} \qquad \alpha = \tan^{-1}\left(\frac{y_p - y_c}{x_p - x_c}\right) \tag{2.43}$$

Para alcanzar un punto determinado el robot debe recorrer el arco de circunferencia (de radio R) que une el punto en el que se encuentra y el punto al que desea llegar. Esto se debe a las restricciones descritas anteriormente. Para ello el robot realiza los cálculos de distancia y de ángulo en cada período de muestreo. Entonces trata de dirigirse al punto deseado haciendo el error del ángulo de orientación ($\theta_e = \alpha - \theta$) igual a cero al mismo tiempo que trata de acercarse al punto tratando de anular el error de distancia ($d = 0$). Este experimento es conocido como "*posture or point stabilization*" [106]. Con estos errores, el control se puede llevar a cabo manipulando la ν y la ω empleando las leyes de control representadas en las Ecuaciones 2.44. Donde ν' se obtiene usando el parámetro h_c el cuál a su vez, se obtiene del algoritmo de evasión de obstáculos que garantiza una disminución de la velocidad ante la cercanía a un obstáculo. Este parámetro será presentado posteriormente como parte de la descripción del algoritmo

de evasión de obstáculos utilizado. Por otra parte, ω es la velocidad angular actual y ω_{max} es la velocidad angular máxima que se alcanzará cuando $\theta_e = \pm 90°$.

$$\nu = \nu' \left(1 - \frac{\omega}{\omega_{max}}\right) \qquad \omega = \omega_{max} sen\,(\theta_e) \tag{2.44}$$

En la Figura 2.28 se muestra un esquema que representa las variables que se tienen en cuenta para llevar a cabo el control de posición. Donde, ν_i es la velocidad lineal de la rueda izquierda, ν_d es la velocidad lineal de la rueda derecha, R es el radio de giro y el punto ICC es el centro instantáneo de curvatura o de rotación.

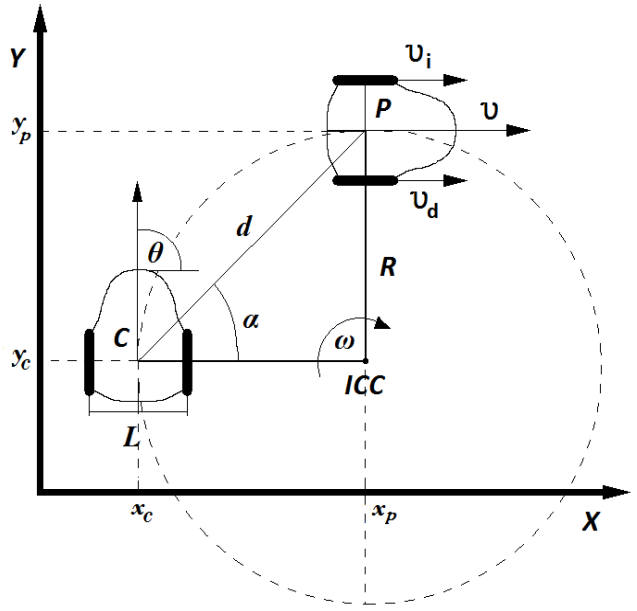


Figura 2.28. Control de posición del robot.

En este tipo de experimentos el robot necesita conocer su posición absoluta o relativa en cada momento. Más adelante se explicará cómo el robot obtiene su posición absoluta para realizar el experimento.

2.2.1.3 Control de formación con evasión de obstáculos

El control de formación de robots es una buena forma de introducir a los estudiantes en el mundo de la robótica móvil. En este caso se consideran formaciones del tipo maestro-esclavos. Este tipo de formaciones consisten en un robot que actúa como maestro del resto de los robots, los cuáles actúan como esclavos. La diferencia de este

tipo de formaciones con respecto a las de tipo líder-seguidores es que en el caso de la formación maestro-esclavos, los robots esclavos mantienen una formación entorno al maestro, por lo que todos los robots esclavos toman como referencia al robot maestro. Mientras que en la formación de tipo líder-seguidores, los robots seguidores siguen al robot líder para mantener una formación en línea recta a partir de la posición del maestro, por lo que a diferencia de la anterior, cada robot seguidor toma como referencia al robot que tiene delante, comenzando por el maestro.

De manera general el algoritmo funciona de forma tal que los esclavos tienen que tomar decisiones dependiendo de su entorno y de la información recibida del maestro. Por su parte el maestro tiene que tomar decisiones a partir de las variables de su entorno y a su vez, enviar la información que necesitan los esclavos para llevar a cabo su objetivo. Estas formaciones se pueden clasificar desde el punto de vista del tipo de control como: centralizado, descentralizado e híbrido. Dependiendo de la medida en que los robots sean capaces de tomar decisiones y actuar por sí solos teniendo en cuenta las variables de su entorno. En este caso el control es híbrido. Esto se explicará posteriormente. Para los experimentos se implementaron varios tipos de formaciones. Todas ellas consisten en hacer que el maestro vaya a un punto al mismo tiempo que envía continuamente su posición al resto de los robots. Los esclavos toman la posición del maestro como referencia para alcanzar un punto a una distancia constante de él. Para llevar a cabo la formación cada robot genera el "*point stabilization*". La Figura 2.29 muestra los diferentes tipos de formaciones desarrolladas donde se ha destacado en color gris el maestro en cada una de las formaciones.

Para este tipo de experimentos debe tenerse en cuenta la evasión de obstáculos. Existen varios algoritmos de evasión de obstáculos para robots móviles que son muy buenos y eficientes. La mayoría de estos algoritmos solo tienen en cuenta obstáculos estáticos. En este caso la idea es implementar uno o varios algoritmos de evasión de obstáculos. Con la dificultad añadida de que los robots se "*vean*" como obstáculos entre ellos. Lo que supone que todos los robots deberán evitar obstáculos móviles y estáticos.

Uno de los métodos más populares es el de los campos potenciales propuesto por Khatib [107, 108]. La idea de este método es que los obstáculos ejerzan una fuerza

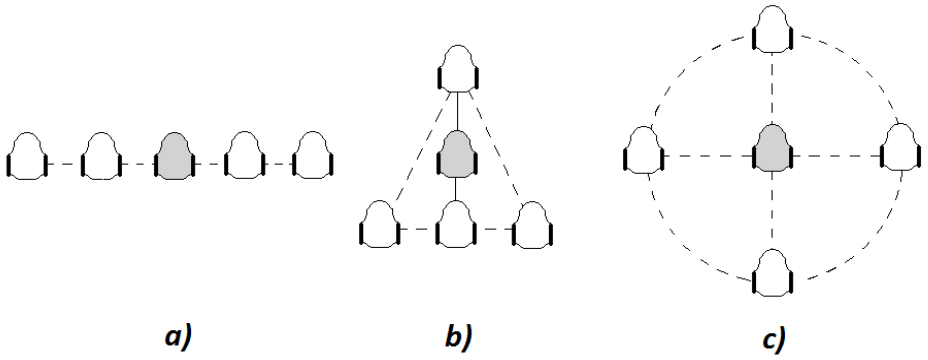


Figura 2.29. Tipos de formación: a) línea b) triángulo c) círculo.

repulsiva virtual sobre los robots al mismo tiempo que el punto de destino ejerza una fuerza atractiva virtual sobre el robot. La magnitud de estas fuerzas depende de la distancia en ambos casos (repulsivas y atractivas). La suma de todas las fuerzas, la fuerza resultante, determina la dirección y velocidad del movimiento del robot. Es como si los obstáculos tuvieran la misma carga eléctrica que el robot al tiempo que el punto de destino tuviera carga eléctrica contraria. De esta manera el robot tiene que moverse por este "campo de potenciales" siendo repelido por los obstáculos y atraído por el punto de destino. El método funciona bastante bien, pero tiene algunas desventajas para entornos dinámicos y cambiantes porque el robot puede caer en situaciones de "*trampa*" y quedar atrapado sin haber llegado a su destino [109]. Este tipo de situaciones se produce cuando el robot se encuentra frente a un obstáculo que ejerce una fuerza repusliva virtual sobre él. Esta fuerza repulsiva virtual puede ser igual a la fuerza atractiva que también está ejerciendo el punto de destino sobre el robot. De manera tal que la fuerza virtual resultante puede ser cero, por lo que el robot puede quedar detenido, sin haber llegado a su destino. Otra desventaja son las oscilaciones que puede experimentar el robot ante la presencia de un obstáculo así como los problemas al tratar de sortear pasillos estrechos. Este tipo de limitaciones son estudiadas por Koren y Borenstein en [110].

Otro método con el que se obtienen muy buenos resultados es el "*VFF (Virtual Force Field")* propuesto por Koren y Borenstein [111]. Este método fue diseñado para evadir obstáculos en ambientes dinámicos. Este método usa la idea de las fuerzas

atractivas y repulsivas virtuales del método de los campos potenciales e incluye un histograma bidimensional del área alrededor del robot. Esta área, llamada "región activa", se divide en celdas y se le asigna un valor determinado a cada celda denominado "certeza" (h). Este valor depende de si la celda está ocupada o no por un obstáculo además de la distancia a la que se encuentra éste del robot. Es aquí donde usan el concepto de los campos potenciales, para asignarle un valor determinado a cada celda dependiendo de la distancia. El histograma se construye entonces con los valores de las celdas alrededor del robot. De tal forma, que al moverse el robot sabe en todo momento las celdas ocupadas y vacías, y por tanto sabe que sectores circulares de su entorno están libres de obstáculos, por lo que puede evadirlos. Este método permite evitar los obstáculos rápidamente siguiendo trayectorias suaves y continuas. Pero tiene las limitaciones inherentes del método de los campos potenciales. En pasillos estrechos el robot recibe fuerzas repulsivas virtuales en direcciones opuestas lo que ocasiona en el robot un movimiento inestable y oscilatorio. Además el robot necesita tener sensores que le indiquen en todo momento la presencia de obstáculos a su alrededor.

Los creadores del *VFF* desarrollaron otro método que denominaron *VFH* (*Vector Field Histogram*) [112]. Este método está basado en el anterior pero usa una estructura de datos intermedia denominada Histograma Polar H. Donde H es un vector de 36 sectores circulares de 10° de ancho (estos valores son variables y se pueden ajustar en función del ancho del robot). El área alrededor del robot en un radio determinado, se denomina celda activa. El radio de esta área está en correspondencia con la distancia máxima que puede medir el sensor de distancia. La celda activa se divide en una matriz de subceldas. El contenido de cada subcelda de la región activa en el histograma se transforma en el correspondiente sector circular del histograma polar. Como resultado en cada sector k del histograma polar se toma el valor de la celda correspondiente h. A este valor se le denomina (H_k) y se interpreta como la densidad polar del obstáculo que se encuentra en la dirección del sector circular k.

La Figura 2.30.a muestra una configuración típica de un experimento de evasión de obstáculos. En color amarillo con el centro rojo se muestran los obstáculos identificados con letras desde la A hasta la F. En color azul se muestra el robot y en color

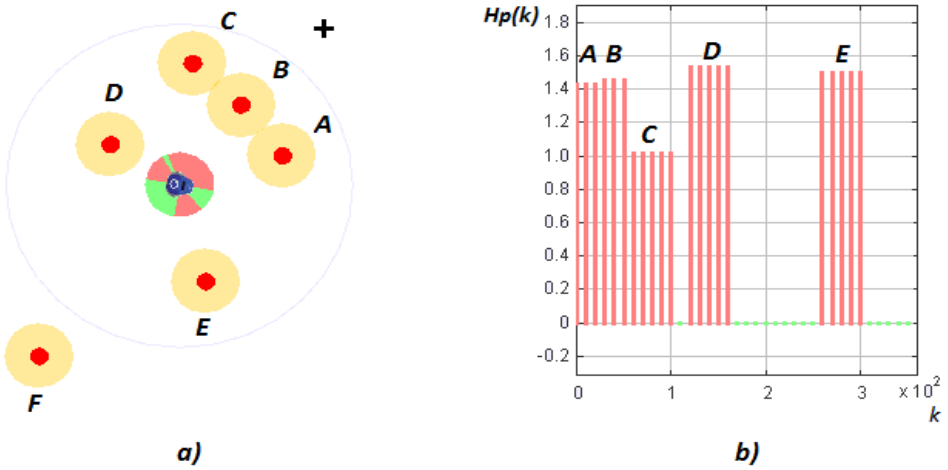


Figura 2.30. VFH: a) configuración para varios obstáculos b) histograma polar correspondiente.

negro una cruz que indica el punto de destino. El círculo mayor de color gris representa la región activa. Mientras que el círculo pequeño que rodea al robot representa el histograma polar, mostrando en color rojo claro los sectores ocupados por los correspondientes obstáculos y en color verde los sectores libres. La Figura 2.30.b representa el correspondiente histograma polar (Hp). Los valores del histograma de cada uno de los obstáculos se representan con la letra que le corresponde. El obstáculo representado por la letra F no se encuentra en el histograma porque está fuera de la región activa, por lo que no tiene influencia sobre el robot porque éste no lo puede "ver".

El método *VFH* elimina algunas de las limitaciones de los métodos basados en los campos potenciales. Por ejemplo, se minimiza la influencia de los errores de medición del sensor de distancia porque estos valores se promedian para calcular el histograma. Además se elimina la inestabilidad que provocan en el robot los pasillos estrechos a causa de las fuerzas repulsivas en sentidos opuestos. En este método no se usan directamente las fuerzas atractivas ni de repulsión, por lo que el robot no puede quedar atrapado a causa de que la fuerza resultante sea cero. Se trata de conducir al robot a través de un sector circular libre, sin importar si se está alejando del punto de destino.

Iwan y Borenstein desarrollaron una mejora del *VFH* a la cual denominaron *VFH+* [113]. Esta mejora incluye que el método tenga en cuenta las dimensiones del robot y además agregan un margen de seguridad a los obstáculos que se tiene en cuenta cuando

se calculan los sectores ocupados. Partiendo del histograma polar construyen un nuevo histograma que denominan histograma polar binario (Hb), en el cuál los sectores solo pueden tener dos valores que indican si están ocupados o no. Para seleccionar estos dos valores se seleccionan dos umbrales, sobre los cuales se decide si se asigna un uno o un cero al sector. Si el valor del sector es mayor que el umbral superior, ese sector tendrá valor uno. En cambio si su valor está por debajo del umbral inferior, ese sector tomará valor cero. El método tiene además en cuenta la trayectoria circular que describirá el robot. Sobre el histograma binario se construye otro histograma adicional que se denomina histograma enmascarado (Hm). Este histograma tiene la particularidad de marcar como ocupados los sectores que el robot no podrá alcanzar debido a que su trayectoria circular se encuentra bloqueada por un obstáculo.

El *VFH+* tiene la desventaja de que solo busca sectores libres sin importar si la elección está alejando al robot de su destino. En esta mejora se aplica una función de optimización que tiene en cuenta de entre todos los sectores libres del histograma enmascarado, los más cercanos a la dirección del punto de destino y las dimensiones del robot.

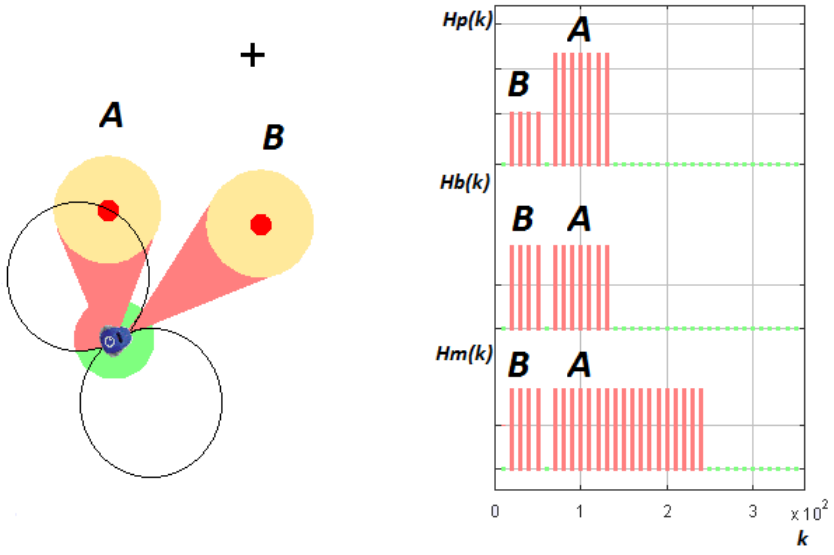


Figura 2.31. VFH+: configuración para dos obstáculos e histogramas polar, binario y enmascarado correspondientes.

La Figura 2.31 muestra una configuración típica para dos obstáculos. En color rojo claro se muestran los sectores circulares bloquedos por los obstáculos y en color

verde los sectores libres. En el histograma polar (*Hp*) se aprecian los diferentes valores de magnitud para cada uno de los sectores. Los sectores que ocupan el obstáculo *B* tienen un valor menor porque está más lejos del robot. En el histograma binario (*Hb*), los valores se establecen en solo dos niveles. En el histograma enmascarado (*Hm*), se aprecian los sectores bloqueados por el obstáculo *A* ya que está sobre el radio de la mínima circunferencia que puede describir el robot y por tanto todos los sectores a su izquierda y hasta $\theta + 180$ quedan bloqueados. En este caso el punto de destino queda entre los dos obstáculos, pero el espacio o pasillo es muy estrecho teniendo en cuenta el margen de seguridad del robot y de los obstáculos. El algoritmo seleccionará otro sector por el cual el robot pueda pasar sin problemas.

2.2.1.4 Características de los robots *Moway*

Para el desarrollo de esta plataforma se decidió seleccionar el robot *Moway*. Este robot ha sido diseñado por la compañía *Bizintek Innova* con fines educativos y de investigación [114]. Es autónomo, programable y concebido principalmente para realizar aplicaciones prácticas de robótica móvil. Está dotado de componentes que le permiten actuar de forma totalmente autónoma en un entorno real. El diseño es muy compacto y está preparado para que pueda moverse con agilidad. Está formado por cinco módulos fundamentales: el sistema de procesamiento, el sistema motriz, el sistema de sensores e indicadores, el sistema de alimentación y el bus de expansión; mediante el cual se puede conectar un módulo de comunicaciones inalámbricas, una cámara de vídeo, una tarjeta de prototipos o cualquier otro dispositivo que se considere interesante para el desarrollo de una tarea. En este caso en el módulo de expansión se conecta un módulo para la comunicación por radiofrecuencia.

Los componentes fundamentales de estos módulos son los siguientes [115–117]:

- *Sistema de procesamiento*: microcontrolador *PIC18F86J50* de la compañía *Microchip Technology* que trabaja a 4MHz [118].
- *Sistema motriz*: está formado por dos servomotores cuyos ejes están conectados a dos ruedas que garantizan el movimiento del robot en diferentes entornos.

- *Sistema de sensores e indicadores*: facilitan la obtención de información del entorno y la interacción con éste.
- *Sistema de alimentación*: batería recargable de *Li-Po* (Litio-Polímero). Esta batería de *3.5V/320mAh* le proporciona al robot una autonomía aproximada de dos horas. La carga se realiza a través del puerto *USB*.
- *Sistema de expansión*: que permite conectar otros módulos en función de las necesidades de la aplicación.

El microcontrolador *PIC18F86J50* es la unidad central donde se procesa toda la información de los sensores y se envían las órdenes a los actuadores. Para ello ejecuta el programa que el usuario ha grabado previamente. Tiene conectado a sus puertos de entrada/salida todos los periféricos que conforman el resto de los módulos del robot. Posee $3MB$ de *SRAM* de memoria de datos, $128KB$ de memoria *Flash* de programa, un reloj de 48 *MHz* y comunicación I^2C.

Por su parte, el sistema motriz está formado por una parte mecánica y otra electrónica. La parte motriz la constituyen los motores y las ruedas, y es la encargada de garantizar el movimiento del robot en diferentes entornos. Mientras que la parte electrónica se encarga principalmente de: a) controlar la velocidad de cada motor por separado; b) controlar el tiempo que dura cada orden aplicada a los motores con una precisión de 100 ms; c) controlar la distancia recorrida en cada orden con una precisión de 1 mm; d) medir la distancia recorrida desde el inicio de las órdenes; e) controlar el ángulo cuando se produce una rotación del robot. Cuando el microcontrolador necesita enviar una orden a los motores, éste envía un comando I^2C al sistema motriz que controla los motores y por lo tanto el microcontrolador queda libre para poder llevar a cabo otras tareas.

El control de velocidad se realiza mediante control en lazo cerrado gracias a la señal de los sensores de velocidad y de recorrido de los motores. La rotación de la rueda se monitoriza por medio de un encoder sobre uno de los engranajes del sistema y un sensor infrarrojo. El microcontrolador analiza esta señal y actúa sobre los motores. De esta manera, el robot puede controlar su velocidad y la distancia recorrida. Cuando desde el microcontrolador principal se quiere que el robot realice un desplazamiento

sólo se debe enviar una orden de movimiento con sus parámetros mediante I^2C. Para ello existen librerías en ensamblador y en lenguaje C con las que esta comunicación queda simplificada por unas funciones que se encargan de la misma.

El sistema de sensores y actuadores está formado por dos sensores de línea (octoacopladores de reflexión *CNY70* de la compañía *Vishay* [119]) situados en la parte inferior delantera del robot. Estos dispositivos tienen una fuente emisora de luz y un detector que está dispuesto en la misma dirección para poder detectar mediante el uso de los rayos infrarrojos la luz reflejada en el suelo. Estos dos sensores se conectan a dos de los puertos del microcontrolador de manera que no sólo se pueden detectar contrastes fuertes, sino que también es posible distinguir entre diferentes tonalidades.

Al igual que los sensores de línea, los sensores detectores de obstáculos utilizan también la luz infrarroja para detectar objetos situados en la parte delantera del robot. El sensor está compuesto por dos fuentes de luz infrarroja (*KPA3010-F3C* de la compañía *Kingbright* [120]) y cuatro receptores (*PT100F0MP* de *Sharp* [121]) colocados en ambos extremos delanteros del robot, que permiten detectar tanto el objeto como calcular la distancia a la que se encuentra. El funcionamiento de este sensor es similar al sensor de línea. El emisor de luz genera un pulso de una duración de 70 micro-segundos que, en caso de existir un obstáculo, se refleja contra él y es captado por el receptor.

El módulo de expansión permite la conexión de otros módulos comerciales o circuitos electrónicos que la aplicación pueda necesitar. En este caso se conecta el módulo *RF Moway BZI-RF2GH4* [122] que está basado en el transceptor *nRF24L01* de la compañía *Nordic Semiconductors* [123]. Este módulo utiliza la radiofrecuencia para facilitar la comunicación inalámbrica del robot con otros *Moway* y con un ordenador, posibilitando el desarrollo de complejas aplicaciones colaborativas. Sus principales características son: bajo consumo, frecuencia de trabajo de 2.4 GHz, potencia de emisión entre -18 y 0 dBm, velocidad de transmisión entre 1 y 2 Mbps y 128 canales de transmisión.

Como se mencionó anteriormente, estos robots se programan usando el lenguaje *C*. Para mover los motores, se emplean distintas instrucciones dependientes de la velocidad, radio de curvatura y sentido del movimiento, por ejemplo: *MOT_ROT(80, FWD,*

CENTER, RIGHT, TIME, 100). Esta instrucción indica rotación hacia la derecha en relación al centro al 80 % de la velocidad durante 10 segundos (100 ms x 100).

Para obtener un valor de velocidad del robot en cm/s, es necesario multiplicar dicho valor por una ganancia para obtener el valor en % que se necesita enviar al motor. Con este objetivo se realizó el proceso de identificación para obtener dicha ganancia. Se le enviaron varias órdenes al robot con valores consecutivos (variando de cinco en cinco) y se midió la velocidad manualmente obteniéndose el gráfico de la Figura 2.32.

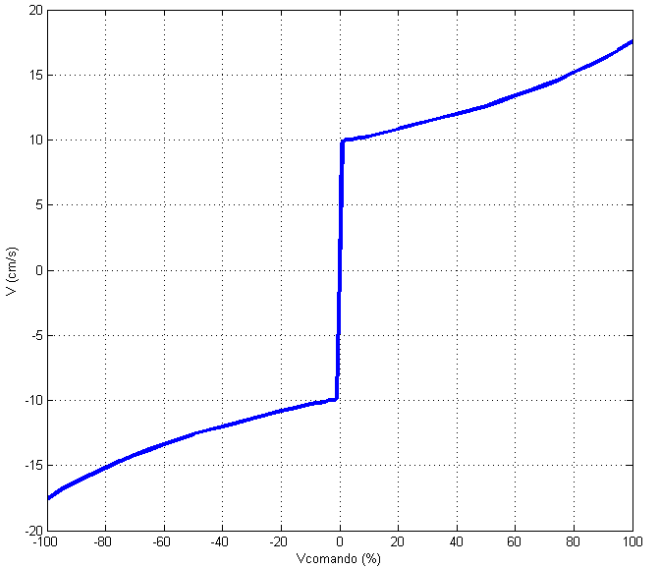


Figura 2.32. Identificación de la velocidad de los robots *Moway*.

Como se puede apreciar, fuera de la zona muerta del motor el comportamiento de la curva es bastante lineal. La velocidad máxima (para la instrucción con valor 100 %) es de 17 cm/s y la mínima es 0 cm/s. Con estos valores se calculó la ganancia tanto para el sentido positivo (hacia adelante) como para el negativo (hacia detrás). Hay que señalar que para la orden con valor 0 % el robot se mantiene detenido, pero para un valor de 1 % la velocidad es de aproximadamente 10 cm/s. Esta considerable discontinuidad o zona muerta (ver Figura 2.32) hay que tenerla en cuenta a la hora de ajustar los parámetros del controlador de velocidad lineal.

2.2.1.5 Ajuste de los controladores de los robots *Moway*

Como el modelo empleado para los robots es no lineal y la velocidad solamente se puede medir de forma manual, los parámetros para el control se sintonizaron manualmente luego de múltiples pruebas. El principal objetivo que se persiguió fue que el robot alcanzara la posición deseada lo más pronto posible. Para esto a la ley de control utilizada (Ecuación 2.44) se le agregó una ganancia que proporciona valores adecuados tanto para la velocidad lineal como para la angular, en cuanto a rapidez en alcanzar el punto de destino y por tanto la formación deseada. Fue necesario añadir estas ganancias debido a que la ley de control usada no tiene en cuenta la dinámica del robot, solo la cinemática. Por tanto, estas ganancias varían en función de la masa del robot, el rozamiento de las ruedas con la superficie y la dinámica de los motores. Ambas ganancias se obtuvieron de forma experimental. Al ser robots diferenciales, las velocidades angular y lineal están relacionadas, por lo que fue necesario prestar atención a los valores de la velocidad lineal con respecto a la angular. Si como resultado de la ley de control se disminuye mucho la velocidad lineal, esto puede provocar que el robot se mueva en forma de zigzag debido a que la velocidad lineal se hace muy pequeña cuando se está cerca del punto de destino o en frente de un obstáculo. Por otra parte, si se aumenta mucho la velocidad lineal, de tal forma que la velocidad angular resulte demasiado pequeña frente a ésta, el resultado será que el robot no gire cuando tenga que hacerlo y se desplace en línea recta debido al valor demasiado pequeño de la velocidad angular.

También se tuvo en cuenta el retardo de la comunicación inalámbrica y el hecho de que para los esclavos se agrega la complejidad de tener que esperar a recibir la posición del maestro para poder calcular su ley de control. Es necesario señalar que aunque todos los robots tienen la misma estructura y similares propiedades físicas, para iguales órdenes de velocidad, sin embargo no respondían exactamente igual. Esta particularidad también se tuvo en cuenta a la hora de obtener los parámetros de los controladores para los diferentes robots. Salvando las diferencias entre los valores obtenidos para el maestro y para los esclavos, se considera que los resultados son buenos, ya que la formación se alcanza con bastante rapidez, no existe sobre-elongación y los

robots reaccionan bien ante las perturbaciones como puede ser el cambio repentino de la posición dentro de la formación, el cambio de formación o un cambio en la posición del maestro.

2.2.2 Descripción de los robots *Surveyor SRV-1*

Para realizar otro tipo de experimentos también con robots móviles, se implementó otra plataforma en colaboración con el Departamento de Energía Eléctrica, Sistemas y Automatización de la Universidad de Gante, en Bélgica. Este grupo de investigación había analizado los robots existentes en el mercado tales como: *LEGO Mindstorm, Khepera, Pionneer* y *Surveryor SRV-1*. Finalmente la elección fue utilizar varios *Surveyor SRV-1*. La decisión se debió básicamente a la relación precio - prestaciones [124]. Estos pequeños robots son móviles y su forma es la de un tanque (con tracción de tipo oruga). Han sido diseñados por la compañía *Surveyor* con propósitos educativos y de investigación [125]. La Figura 2.33 muestra el *Surveyor SRV-1*.

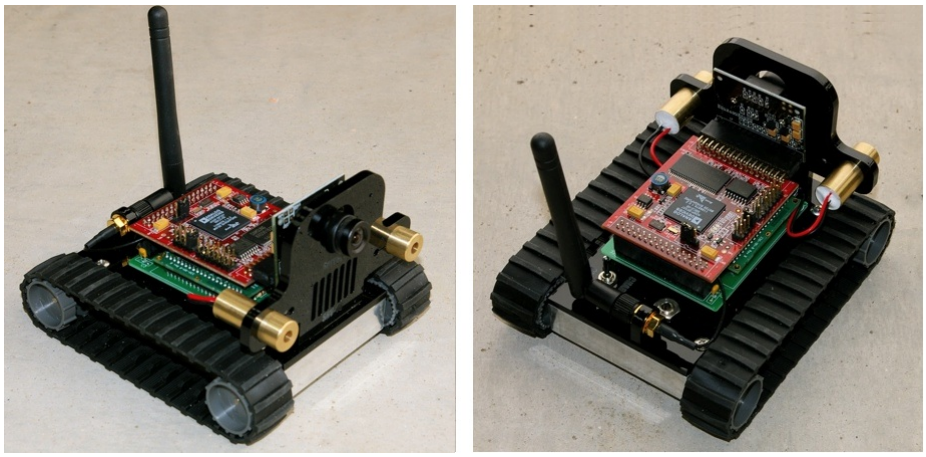


Figura 2.33. Robot *Surveyor SRV-1*.

Los robots *Surveryor SRV-1* están formados por tres módulos secundarios y uno principal. Los módulos son los siguientes:

1. *Módulo de Locomoción*: que consiste en un par de esteras o tracciones de tipo oruga, impulsadas diferencialmente por cuatro motores de corriente continua (dos a cada lado). Entre el eje del motor y la rueda que mueve la estera hay un reductor con una relación de reducción de 100:1.

2. *Módulo de Visión*: que consiste en una cámara digital de vídeo. A través del procesamiento de las imágenes de la cámara se puede calcular distancia.
3. *Módulo de Comunicaciones*: que consiste en un transmisor inalámbrico. A través de éste los robots se pueden comunicar con un ordenador.
4. *Módulo de Procesamiento*: que consiste en un micro-controlador *ARM7*. Éste es el módulo principal por lo que todos los módulos secundarios están conectados a él. Este módulo es el encargado de procesar las imágenes de la cámara, enviar órdenes a los motores y garantizar la comunicación inalámbrica. De esta forma, pueden ejecutar a bordo programas escritos en código C interpretado con herramientas de código abierto. Pueden operar así de forma autónoma, o manipularse de forma remota desde *Windows*, *Mac OS/X* o *Linux*.

El módulo de comunicaciones se puede configurar en modo *ad-hoc* en el cual el adaptador de red actúa como un servidor para el resto de los dispositivos con los cuáles se establezca la comunicación (red inalámbrica descentralizada). También se puede configurar en modo infraestructura, en el cual el adaptador de red se conectará a una red inalámbrica especificada. En este caso cada robot se configura en modo infraestructura con *IP* estática. Algunas de sus especificaciones de *Hardware* y *Software* son:

- 4 Motores eléctricos.
- Procesador embebido: *Analog Devices Blackfin BF537* [126].
- Cámara integrada: *Omnivision OV9655* [127].
- Tarjeta de red inalámbrica embebida: *Lantronix Matchport 802.11b/g WiFi*.
- Dos punteros laser.
- Velocidad: 20cm/s - 40cm/s (aproximadamente 1 pie/s o 0.5 millas/h).
- Dimensiones: 120mm de largo x 100mm de ancho x 80mm de altura.
- Peso: 350g (12oz).
- *Firmware*: Está escrito en lenguaje *C* y se puede actualizar fácilmente bajo licencia *GPL Open Source*.
- Intérprete para el lenguaje *small C* con instrucciones específicas para el robot, que permite ejecutar programas de usuario desde la memoria flash de a bordo.
- Herramientas de desarrollo de código abierto *GNU* [128].

Un inconveniente importante de estos robots es la falta de codificadores que puedan medir la velocidad de rotación. Esto unido a las diferencias entre las características dinámicas de los motores puede provocar que las dos esteras giren a velocidades distintas, lo que a su vez puede ocasionar que el robot no se mueva en una trayectoria recta cuando se envía la misma orden a las dos parejas de motores. Por otra parte, la velocidad tiene una característica no lineal ya que presenta una zona muerta al iniciar el movimiento y vencer la inercia estando en reposo. Todos estos inconvenientes hacen que los robots sean difíciles de controlar.

El microprocesador *Blackfin BF537* diseñado por *Analog Devices* tiene $32MB$ de *SDRAM*, $4MB$ de memoria *Flash*, un reloj de 500 *MHz* y comunicación I^2C. Este microprodesador es lo suficientemente rápido para desarrollar segmentación por colores en 16 ms para una imagen de 160x120 píxeles y 70 ms para una imagen de resolución 320x240. Uno de sus puertos serie (*USART0*) está conectado al adaptador inalámbrico. El puerto serie *USART1* no está conectado y se puede utilizar para conectar otro microcontrolador o un módulo Zigbee para la comunicación con el resto de los robots.

La cámara integrada en el robot es la *Omnivision OV9655* de 1.3 mega píxeles producida por *OmniVision Technologies*. Tiene una resolución de entre 160x128 y 1280x1024 píxeles. Esta cámara posee algunas características importantes como son: control de exposición automático, balance de blancos automático y calibración del nivel de blanco automático. Estas funciones pueden habilitarse/deshabilitarse en el *firmware* del robot. Además soporta varios formatos de colores *RGB*, *YUV* y *YCbCr*. En este caso se emplea el formato *YUV* ya que el *RGB* es muy popular debido a su idoneidad para pantallas a color, pero no es bueno para la segmentación y el análisis de color de imágenes, debido al alto grado de correlación entre los tres colores. Esto significa que si la intensidad de los colores cambia debido a la intensidad de la luz, el valor de estos tres componentes también cambiará.

El adaptador inalámbrico integrado es el *Lantronix Matchport WLAN 802.11g*. Este módulo tiene dos canales serie los cuales pueden enviar datos a una velocidad de 921 *Kbps*. Este dispositivo soporta dos estándares de modulación: 802.11b, 802.11g y puede implementar varios protocolos de comunicación: *ARP, UDP, TCP, Telnet, ICMP,*

SNMP, DHCP, BOOTP, Auto IP, HTTP, SMTP y *TFTP*. En el *SRV-1* el canal 1 de esta tarjeta está conectado al primer puerto serie del procesador *Blackfin*. Este canal está configurado para el protocolo *TCP*. Mientras que el segundo canal no está conectado. Este segundo canal podría utilizarse también para la comunicación entre los robots, conectándolo al puerto *USART1* del microprocesador y configurándolo con el protocolo *UDP*.

2.2.2.1 Modelo de los robots *Surveyor SRV-1*

Como se mencionó anteriormente, para esta plataforma los robots empleados son los *Surveyor SRV-1*. Estos son robots móviles pequeños estilo tanque de guerra u oruga. Su movimiento está basado en dos esteras que pueden rotar independientemente por medio de dos parejas de motores. Pueden así variar la dirección a través de la variación de la rotación relativa entre las esteras y, no necesitan ningún mecanismo adicional para girar. La velocidad lineal es perpendicular al eje de las esteras y se considera que estas no deslizan. Estas condiciones imponen restricciones no holonómicas a este tipo de sistemas. La Figura 2.34 muestra el modelo geométrico del robot.

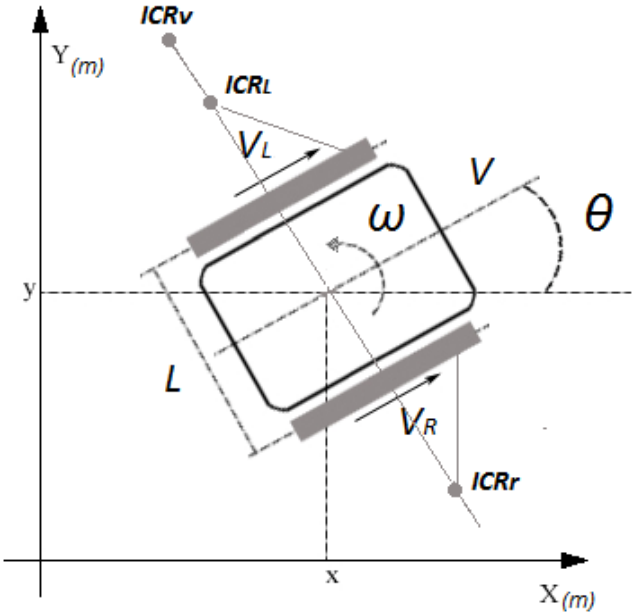


Figura 2.34. Modelo geométrico del robot *Surveyor SRV-1*.

Al igual que en los robots diferenciales de ruedas, en los robots de tipo oruga el

control se rige por dos variables fundamentales (V y ω). Donde la velocidad traslacional (V) se puede descomponer en velocidad izquierda (V_L) y velocidad derecha (V_R), refiriéndose a las velocidades de las respectivas esteras. Mientras que ω_z es la velocidad angular. La Ecuación 2.45 representa la relación entre estas variables, donde V_x y V_y son las componentes de la velocidad traslacional en los respectivos ejes del plano.

$$(v_x, v_y, \omega_z) = f_d\,(V_L, V_R) \tag{2.45}$$

Si se considera el robot como un cuerpo rígido con un movimiento bidimensional, entonces el centro de rotación instantánea (ICR_v) se define como el punto en el plano horizontal donde el movimiento del robot se puede representar como una rotación sin ocurrencia de traslación. Este punto se puede designar en coordenadas locales como ($ICR_v = X_{ICRv}; Y_{ICRv}$). También se tiene en cuenta el movimiento de ambas esteras y su contacto con la superficie. Una estera se puede modelar como un cuerpo rígido con un grado extra de libertad que es su velocidad de rodadura. De esta forma el movimiento del sistema viene caracterizado por el movimiento del robot y la rodadura de la estera. Por esta razón el *ICR* de la estera es diferente del *ICR* del vehículo. Los *ICRs* para la estera derecha e izquierda pueden definirse respectivamente como: $ICR_l = X_{ICRl}; Y_{ICRl}$ e $ICR_r = X_{ICRr}; Y_{ICRr}$ respectivamente. La componente Z de la velocidad angular de las dos esteras en el plano, es la misma que la del robot, ya que el vehículo no puede girar en el eje Z. Las coordenadas locales para el robot y los *ICRs* de las esteras se pueden obtener geométricamente en función de las velocidades angular y traslacional del robot tal como se muestra en las Ecuaciones 2.46 y 2.47 [129].

$$X_{ICRv} = \frac{-v_y}{\omega_z} \quad ; \quad X_{ICRl} = \frac{V_L - V_y}{\omega_z} \quad ; \quad X_{ICRr} = \frac{V_R - V_y}{\omega_z} \tag{2.46}$$

$$Y_{ICRv} = Y_{ICRl} = Y_{ICRr} = \frac{V_x}{\omega_z} \tag{2.47}$$

Si se obtienen las funciones inversas de las ecuaciones anteriores, se deducen las velocidades traslacional y rotacional que representan la cinemática del robot.

$$V_x = \frac{V_R - V_L}{X_{ICRr} - X_{ICRl}} Y_{ICRv} \tag{2.48}$$

$$V_y = \frac{V_R + V_L}{2} - \frac{V_R - V_L}{X_{ICRr} - X_{ICRl}} \left(\frac{X_{ICRr} + X_{ICRl}}{2} \right) \tag{2.49}$$

$$\omega_z = \frac{V_R - V_L}{X_{ICRr} - X_{ICRl}} \tag{2.50}$$

2.2.2.2 Control de formación de robots (líder-seguidores)

Para demostrar el uso de esta plataforma experimental de robótica móvil, se ha decidido implementar una experiencia de control de formación en la modalidad líder-seguidores. En el cual hay uno o varios líderes de la formación y uno o varios seguidores. En este caso, por simplicidad en la implementación, se ha considerado un solo líder y varios seguidores, de forma tal que la formación se mueve en columna. El líder debe seguir una trayectoria predefinida, mientras que la tarea de los seguidores es mantener una posición predefinida (distancia y orientación) con respecto al robot que tienen en frente. En la Figura 2.35 se muestra una formación en columna similar a la que se desea realizar, en este caso con un líder y dos seguidores [124, 130].

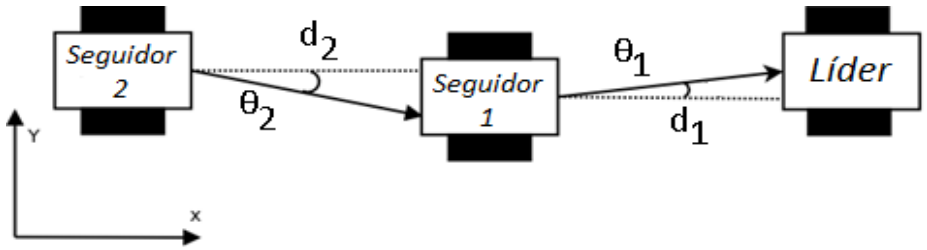


Figura 2.35. Formación de tipo líder-seguidores.

El robot del centro (seguidor 1) es el maestro para el robot de la izquierda (seguidor 2). El seguidor 1 tiene que mantener su posición con respecto al robot de la derecha (líder), manteniendo la distancia (d_1) y la orientación (θ_1) predefinidas. Por su parte, el objetivo del seguidor 2 es mantener su posición con respecto al seguidor 1 con las coordenadas (d_2) y (θ_2). Para esta implementación se han considerado las distancias $d_1 = d_2 = 30\ cm$. De la misma forma se ha asumido que $\theta_1 = \theta_2 = 0°$.

2.2.2.3 Segmentación del color

Para determinar la posición del robot en el control de la formación, cada robot usa la cámara que tiene incorporada. En este caso el algoritmo es distribuido ya que cada uno de los robots realiza sus propias mediciones y toma sus propias decisiones. Para poder llevar a cabo su tarea de seguir al robot que tiene delante, el seguidor tiene que poder medir la distancia y la orientación, tal como se mostró en la Figura 2.35. Todos los seguidores hacen esto usando la cámara de a bordo. La información de los colores en las imágenes adquiridas por la cámara se representa en el espacio de colores YUV. Este modelo define un espacio de colores en términos de una componente de luminancia o brillo del color (Y) y dos componentes de crominancia o color (UV) y codifica una imagen o vídeo en color teniendo en cuenta la percepción humana. Por lo que permite utilizar un ancho de banda reducido para los componentes de crominancia. De esta forma se logra que se oculten los errores de transmisión o las imperfecciones de compresión más eficientemente que cuando se emplean una representación RGB [131].

En la Figura 2.36 se muestra una representación de los componentes UV. En esta figura el rango de colores está escalado entre -0.5 y 0.5 mientras que en los robots los colores se representan con valores de 8 bits de 0 a 255.

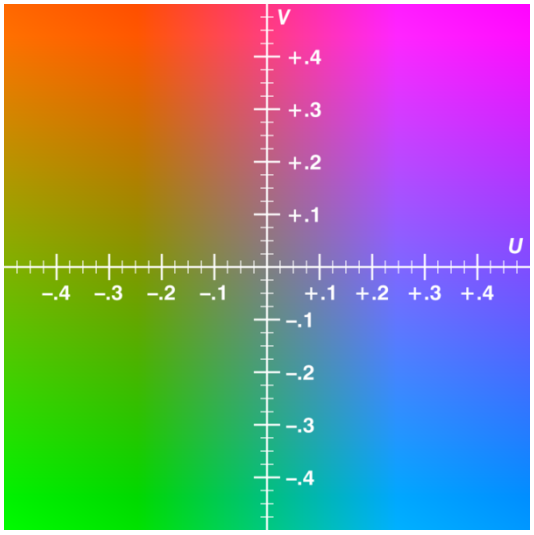


Figura 2.36. Representación de colores UV.

Para determinar si un punto con unos valores (Y, U, V) pertenece a un color o no,

estos valores tienen que estar en el rango de intervalos que definen el espacio de colores para las tres componentes. Si alguno de los componentes no está en el rango, entonces el color está fuera del espacio de colores *YUV*. En este caso, para el componente Y el rango del intervalo fue extendido al máximo, entre 0 y 255, con el fin de compensar las variaciones en la intensidad de la luz ambiente. Desafortunadamente, esto hará que los colores se superpongan, por lo que la cámara detectará algunos píxeles extraviados del fondo, como píxeles válidos. Este problema se resolvió escogiendo un intervalo fijo para la componente U pero variando el intervalo para la componente V, de manera que se adapte sobre la base de los valores medios de todos los píxeles de la imagen. Por ejemplo, para el marcador rojo que los seguidores tienen que detectar se usaron los siguientes valores: $0 \leq Y \leq 255$; $50 \leq U \leq 140$; $V_{mean} + 40 \leq V \leq 255$. Estos valores se seleccionaron a través del análisis de los valores de los píxeles en varias imágenes para diferentes condiciones de luz.

Después de determinar los intervalos de los colores, la imagen obtenida de la cámara se procesa mediante la función *vblob*, que está definida en el *firmware* del *SRV-1*. Esta función realiza el siguiente algoritmo:

1. Determina el píxel que coincide con el intervalo de color especificado.
2. Filtra los píxeles extraviados.
3. Determina las coordenadas y el número de píxeles de cada rectángulo.
4. Organiza los rectángulos en orden descendente de acuerdo a sus áreas.

El filtrado de los píxeles extraviados se hace considerando un píxel dado y los píxeles cercanos a éste (un total de ocho). Si al menos cuatro de los píxeles adyacentes a él en vertical, horizontal o diagonal, se encuentran en el intervalo especificado, este píxel se considera como válido, en cualquier otro caso se desecha.

En la parte izquierda de la Figura 2.37 se muestra una imagen capturada de la cámara. El objetivo es determinar el área del marcador amarillo fijado a la parte trasera del robot que se encuentra delante. El resultado después de realizar la segmentación de color se muestra a la derecha en la misma figura. La función *vblob* devolverá también las coordenadas del vértice superior izquierdo (x1, y1) y del vértice inferior derecho (x2, y2) del rectángulo. Donde el eje de las abscisas (X) es horizontal y está orientado

de izquierda a derecha y el eje de las ordenadas (Y) es vertical con dirección de arriba hacia abajo.

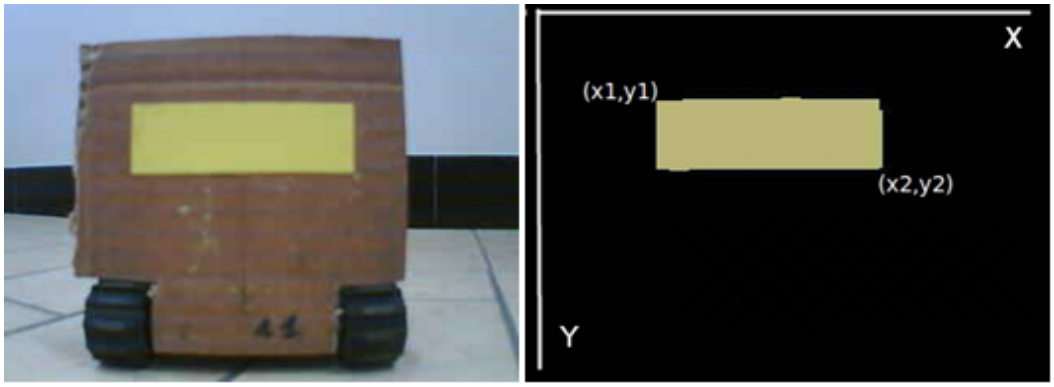


Figura 2.37. Imagen obtenida con la cámara y resultado del proceso de segmentación.

2.2.2.4 Determinación de las coordenadas del robot líder

Con el área y las dimensiones del rectángulo, el robot seguidor tiene que obtener la posición del robot que se encuentra delante en coordenadas reales. El ángulo entre los robots se puede relacionar con las coordenadas del centro del rectángulo en el eje de las abscisas. Estas coordenadas se pueden calcular usando la Ecuación 2.51.

$$x_c = \frac{x_1 + x_2}{2} \tag{2.51}$$

Donde x_c es la coordenada x del centro del rectángulo, $x1$ es la coordenada x del punto superior izquierdo del rectángulo, mientras que $x2$ es la coordenada x del punto inferior derecho del rectángulo.

Para este experimento que consiste en mantener una formación de tipo fila, el cálculo del ángulo entre los robots no es necesario. Si el centro del rectángulo coincide con el centro de la imagen, los robots están alineados. El segundo parámetro que se necesita determinar es la distancia entre los robots. Para hacer esto con una cámara simple, se necesitan conocer las dimensiones del objeto que se quiere detectar y entonces relacionar las dimensiones reales del objeto con las dimensiones del objeto en la imagen, medidas en píxeles. Con este objetivo se realizó un experimento que consistió en situar un robot en frente de otro a varias distancias (en centímetros) y medir la altura (en

píxeles) del rectángulo correspondiente para obtener una tabla que relaciona estos dos parámetros [124].

2.2.2.5 Algoritmo de control

Para mantener la formación, el robot seguidor tiene que calcular la distancia al robot que se encuentra delante y generar las órdenes de los motores necesarias para mantener la posición relativa. Las acciones del algoritmo de control se pueden dividir en varias etapas, correspondiéndose con las acciones que el robot debe llevar a cabo en modo autónomo.

Las acciones son las siguientes:

- Leer los parámetros para los controladores *PID* usados en el seguimiento del robot de delante.
- Calcular la posición relativa respecto del robot de delante. Éste es el proceso más lento debido a que lleva implícito el procesamiento de imágenes.
- Detener los motores si detecta un robot delante y muy cerca ($d < 20\ cm$).
- Calcular la acción de control para un movimiento longitudinal usando un controlador *PID*. Esta acción se realizará cuando el robot de delante esté a una distancia mayor que la mínima ($d > 20\ cm$) y menor que la máxima ($d < 45\ cm$).
- Calcular la acción de control para un movimiento lateral usando un controlador *PID*. Esta acción se llevará a cabo teniendo en cuenta el error de ángulo del robot de delante.
- Establecer una velocidad para los motores. Esta acción está asociada al cálculo de la acción de control longitudinal o lateral. Se envían las órdenes a los motores como señales *PWM*.
- Guardar los datos. Se almacenan en la memoria los valores de la distancia, la orientación, las órdenes laterales y longitudinales, el tiempo y los valores de las señales *PWM*, para descargarlos al final del experimento.
- Obtener mensajes del usuario. Los mensajes que puede enviar el usuario son: detener el algoritmo de control, enviar la información del estado actual o enviar la imagen que está almacenada actualmente en la memoria.

Cuando un robot detecta que hay otro robot delante se calculan las órdenes para el movimiento lateral y longitudinal usando dos controladores *PID*. En la Sección 3.5 se presentaron las ecuaciones que describen el modelo cinemático de este tipo de robots. Estas ecuaciones consideran los centros instantáneos de rotación de las esteras (x_ICRr y x_ICRl) y además el centro instantáneo de rotación del robot (x_ICRv). La relación cinemática inversa está dada por las Ecuaciones 2.52.

$$V_L = V_y + X_{ICRl}\omega_z \qquad V_R = V_y + X_{ICRr}\omega_z \tag{2.52}$$

En estas ecuaciones los signos de las coordenadas de los centros instantáneos de rotación en el eje x son diferentes, el centro de la izquierda tiene signo negativo y el de la derecha tiene signo positivo. El primer término V_y hará que el robot se mueva al frente, mientras que los dos últimos términos provocarán que el robot gire a la izquierda o a la derecha. Para obtener una versión simplificada de estas dos ecuaciones se pueden utilizar dos señales intermedias, que desacoplan las dinámicas lateral y longitudinal. Las Ecuaciones 2.53 muestran esta versión simplificada.

$$V_L = u_{lon} - u_{lat} \qquad V_R = u_{lon} + u_{lat} \tag{2.53}$$

El término *u_lon* provocará un movimiento al frente del robot, mientras que el término *u_lat* compensará las perturbaciones laterales. Estos dos términos se calculan en lazo cerrado con la información medida por los sensores. El control en lazo cerrado es necesario en ambos movimientos (tanto en el lateral como en el longitudinal), debido a las perturbaciones provocadas por el movimiento del robot a las esteras que rotan a velocidades angulares diferentes debido a las diferentes características de los motores, la fricción, etc. Todos estos factores unidos pueden provocar que el robot seguidor pierda rápidamente su referencia si no se usa control por realimentación. La tarea de los controladores longitudinal y lateral es: calcular los valores de las velocidades angulares de las esteras derecha e izquierda para minimizar los errores longitudinal y lateral. La Figura 2.38 muestra el diagrama de bloques del lazo de control. El movimiento del líder se considera como una perturbación a la salida para el seguidor. El desplazamiento

longitudinal del líder actuará como una perturbación (D_lon) en la distancia entre los dos robots. Mientras que el desplazamiento lateral del líder actuará a su vez, como una perturbación (D_lat) para la orientación entre ambos robots.

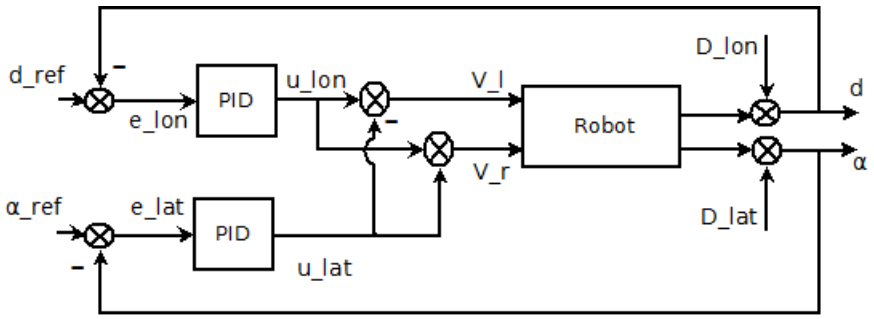


Figura 2.38. Diagrama de bloques del lazo de control.

2.2.2.6 Ajuste de los controladores de los robots *Surveyor SRV-1*

Debido a la falta de un modelo lineal de los robots y a no poder medir su velocidad, los controladores se sintonizaron manualmente haciendo múltiples ensayos. Además de buscar parámetros del controlador que garanticen un buen rendimiento, también se deseaba que estos permitan que el robot alcance rápidamente la formación, que no oscilen en la formación y que tengan un sobre-impulso inicial pequeño. También debe ser un controlador robusto adecuado para el resto de los robots.

A pesar de que los robots tienen la misma estructura en su módulo de locomoción (esteras, motores, etc), las diferencias en la velocidad son bastante significativas para órdenes semejantes de los motores y los parámetros de las cámaras de a bordo pueden ser diferentes, lo que da lugar a errores en la estimación de la distancia.

Los parámetros del controlador *PID* longitudinal se obtuvieron de forma experimental tratando de garantizar el mejor rendimiento. Además, todos los ensayos fueron realizados en línea recta para minimizar las perturbaciones que ocasiona el movimiento del líder en el controlador lateral. La referencia del controlador longitudinal fue establecida en 30 cm y los límites en 25 cm para el mínimo y 60 cm para el máximo. Los parámetros obtenidos para el controlador longitudinal son $K_p = 4$ y $K_i = 0{,}3$ con 0 en el término derivativo. Por su parte para el controlador lateral los mejores

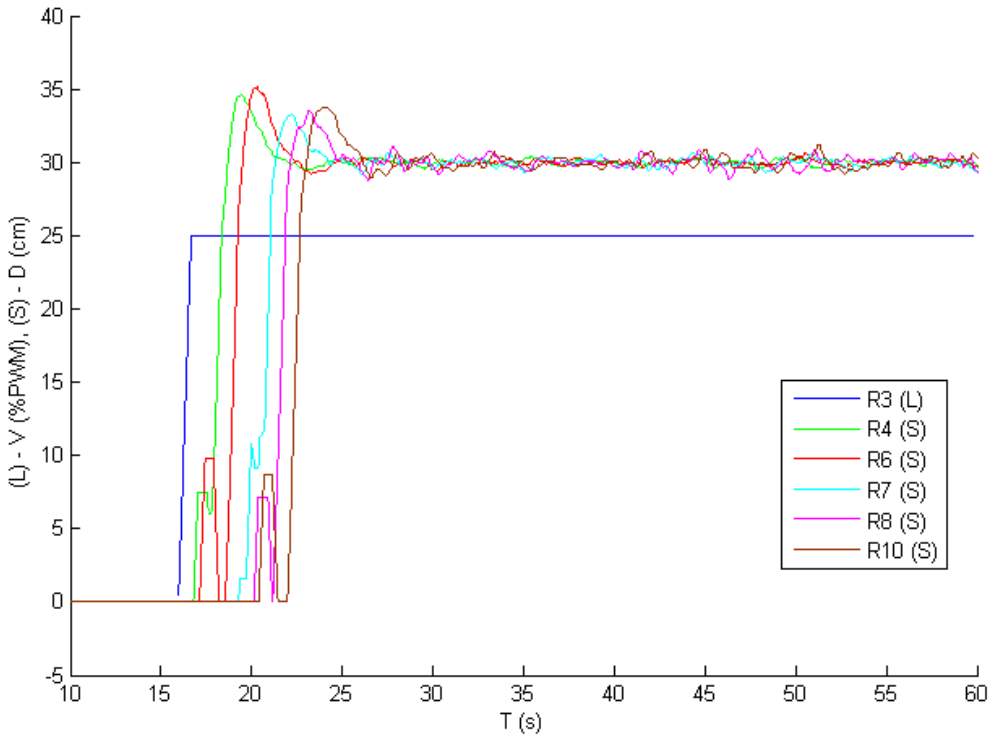


Figura 2.39. Respuesta de los robots para los parámetros $K_p = 2$ y $K_i = 0{,}7$.

resultados se obtuvieron para un controlador proporcional con una ganancia de 0.1. Al comienzo, el seguidor no reaccionaba instantáneamente, cuando el líder comenzaba a despegarse pasaba un tiempo hasta que éste se ponía en marcha. Sin embargo, el seguidor se mantenía en la formación pero no alcanzaba la distancia deseada porque la acción integral era muy pequeña. Al aumentar este coeficiente hasta 0.9 se obtuvieron mejores resultados. El seguidor alcanzó la distancia requerida pero el rechazo a perturbaciones no era muy bueno y oscilaba entorno al estado estacionario. Se obtuvo una mejora al disminuir ambos coeficientes hasta $K_p = 2$ y $K_i = 0{,}7$. Estos valores resultan ser aceptables para casi todos los robots. Todos alcanzan la posición deseada antes de los 4 s. Aunque algunos presentaban algunas pequeñas oscilaciones por las cuestiones planteadas anteriormente, se puede aceptar que mantienen la formación. La Figura 2.39 muestra en color azul marino la velocidad constante del robot líder (R3) (25 % de su velocidad máxima) y la distancia con respecto al líder del resto de los robots (R4, R6, R7, R8, R10). Como se puede apreciar, cada robot seguidor debe permanecer a una distancia de 30 cm con el robot que tiene en frente.

Al incluir una acción derivativa en el controlador el seguidor responde más rápidamente en el instante inicial. Como consecuencia la distancia entre los robots no aumentó mucho al comienzo del experimento. Hay que tener en cuenta que debido al ruido en las mediciones la acción derivativa no es muy beneficiosa a menos que se implemente a través de un filtro.

3

Laboratorios virtuales y remotos

En este capítulo se presentan las arquitecturas y el funcionamiento de los laboratorios virtuales y remotos desarrollados para la realización de prácticas de automática y robótica. Para la plataforma de control con el sistema bola y aro y con el sistema bola y plato, se presentan los laboratorios virtuales y remotos implementados para cada uno de estos sistemas. Se detallan las aplicaciones desarrolladas con las diferentes herramientas de *Software* en cada uno de los casos: sistema bola y aro (*EJS-MATLAB/Simulink* y *EJS-LabVIEW*) y sistema bola y plato (*EJS-Visual C#*). Por otra parte se presentan además los laboratorios virtuales y remotos desarrollados utilizando los dos tipos de robots presentados en el Capítulo 2: *Moway* y *Surveyor SRV-1*.

3.1 Sistema bola y aro (*EJS-MATLAB/Simulink*)

En esta sección se describen las características del laboratorio virtual y remoto con el sistema bola y aro en la versión *EJS-MATLAB/Simulink*. Su arquitectura está basada en una estructura cliente - servidor [44, 52, 132–135]. En la Figura 3.1 se muestra una representación de esta arquitectura.

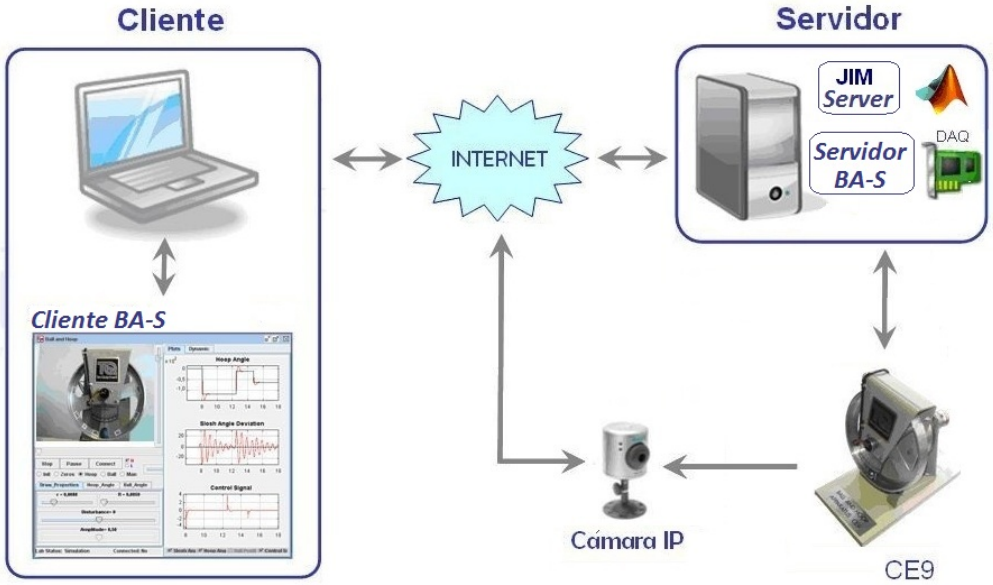


Figura 3.1. Arquitectura de la plataforma empleando *EJS-MATLAB/Simulink*.

Como se puede apreciar, la arquitectura está formada por componentes de *hardware* y *software* que interaccionan entre sí. Los principales componentes de *hardware* en cuanto a la arquitectura son dos ordenadores: uno en la parte del cliente y otro en la parte del servidor. En el ordenador del lado del cliente se ejecuta una aplicación que bien puede ser, un *Applet Java* [136] o una aplicación ejecutable independiente o autónoma ("*standalone*"), también desarrollada en *Java*, dependiendo de la decisión del programador. Si lo que se tiene es un *Applet* se necesitará un servicio *web* en la parte del servidor y un navegador *web* en la parte del cliente para que éste se pueda ejecutar. Esto se debe a que los *Applets* a pesar de ser código *Java*, no pueden ejecutarse de forma autónoma ya que carecen del método *main* y necesitan de un *contenedor* para ejecutarse (ejemplo: una página web). Mientras que si se tiene una aplicación independiente o autónoma *Java* (cuya extensión es *.jar*), ésta se puede ejecutar directamente sobre

la máquina virtual de java que se encuentra activa por defecto en el ordenador del usuario. En este último caso, esta aplicación se ha denominado *Cliente BA-S* y se ha desarrollado como una aplicación *Java* ejecutable utilizando el *Software EJS*. La aplicación *Cliente BA-S* tiene dos estados de trabajo: virtual y remoto.

En el primer estado de trabajo se permite a los usuarios realizar experimentos con una simulación interactiva basada en el modelo del sistema bola y aro. El usuario puede interaccionar con la simulación a través de la interfaz gráfica de usuario (*GUI*), con la que es posible modificar interactivamente los parámetros de la simulación y observar al instante los resultados por medio de las salidas gráficas correspondientes.

En el segundo estado de trabajo la aplicación se conecta al servidor para controlar la planta real; lo que permite realizar experimentos con ésta. La conexión se realiza usando el protocolo *TCP/IP* a través de *Internet*; por lo que los estudiantes no tienen que desplazarse hasta la universidad para realizar las prácticas de laboratorio con equipamiento real. En este modo de trabajo la animación de la simulación se sustituye por la señal de vídeo de una cámara *IP* que permite observar el experimento. Durante la realización del mismo, se muestran en las gráficas los datos de la planta real. La Figura 3.2 muestra la estructura de la comunicación que se establece entre las aplicaciones del cliente y del servidor.

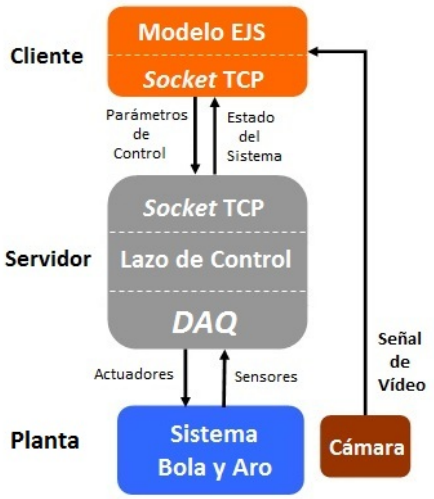


Figura 3.2. Estructura de la comunicación entre las aplicaciones del cliente y del servidor.

Para soportar la comunicación entre el cliente y el servidor se utiliza un programa

desarrollado en el Departamento de Informática y Automática de la UNED, denominado *JIM Server* (*Java-Internet-MATLAB Server*) [137–139]. Esta herramienta garantiza la comunicación a través de *Internet* entre aplicaciones desarrolladas en *Java* y en *MATLAB*. Esta aplicación se ejecuta en el servidor de forma permanente y está a la "escucha" para atender las solicitudes que se realicen a *MATLAB* a través de la conexión *TCP*. Para llevar a cabo el control de la planta real desde *MATLAB* se ha desarrollado una aplicación *Simulink* denominada *Servidor BA-S*. La interacción con la planta real se realiza a través de una tarjeta de adquisición de datos (*DAQ* en la Figura 3.2). A continuación se describen con cierto detalle la aplicación *Cliente BA-S* y la aplicación *Servidor BA-S*.

3.1.1 La aplicación *Cliente BA-S*

La aplicación *Cliente BA-S* se ha implementado usando *EJS*. Esta herramienta consta de tres secciones fundamentales:

1. *Descripción*: en esta sección se muestra un enunciado con las principales características de la simulación y las correspondientes instrucciones para su correcto uso.
2. *Vista*: esta sección se corresponde con la animación interactiva de la simulación.
3. *Modelo*: en esta sección se recogen las variables, las relaciones entre éstas, las ecuaciones diferenciales del modelo y el código *Java* requerido para la inicialización de la simulación.

Usando estas tres secciones se pueden desarrollar simulaciones interactivas sin necesidad de tener muchos conocimientos de programación. El comportamiento y las funciones de los objetos que se colocan en la vista, se manipula a través de la sección *Modelo*.

La Figura 3.3 muestra el diagrama de flujo de la aplicación *Cliente BA-S*. Como se puede apreciar, la mayoría de las funciones se encuentran implementadas en la subsección *Custom* que a su vez se encuentra en la sección *Modelo*. Estas funciones son invocadas desde la *Inicialización*, *la Evolución* y las *Relaciones Fijas*, como se explica a continuación.

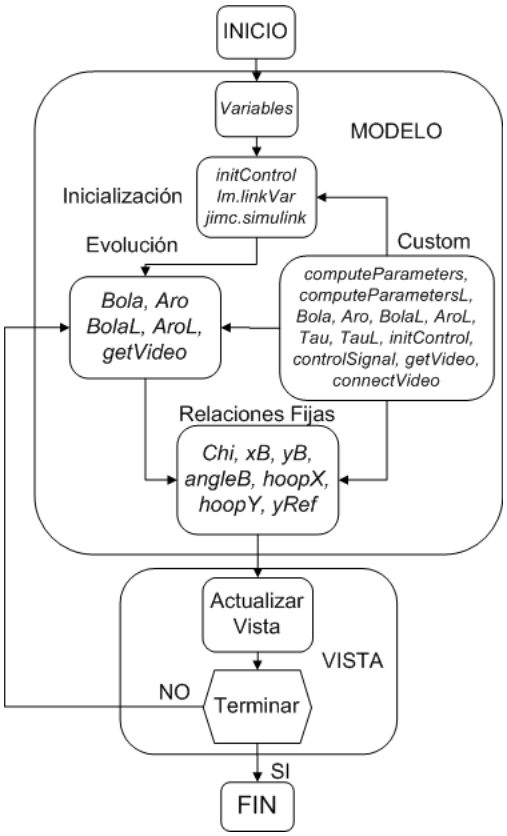


Figura 3.3. Diagrama de flujo de la aplicación *Cliente BA-S*.

La Figura 3.4 muestra la vista de *EJS* de la aplicación *Cliente BA-S* en tiempo de diseño. Como se puede apreciar, la vista se construye con una estructura de árbol, utilizando los objetos predefinidos que brinda *EJS* para la construcción de la simulación. Estos componentes tienen varias propiedades asociadas, las cuales rigen su comportamiento en tiempo de ejecución. Entre las más destacadas se encuentran sus variables y funciones asociadas, así como sus propiedades visuales: color, forma, tamaño, etc.

En la sección correspondiente al modelo se encuentran las variables, como se mencionó anteriormente. En este caso concreto, la simulación posee un gran número de variables, las más destacadas son las relacionadas directamente con los modelos lineal y no lineal del sistema: ángulo del aro, ángulo de la bola, velocidades angulares de la bola y el aro, entre otras. En esta sección, también se encuentran las ecuaciones diferenciales en las que se basa la simulación, tal como se muestran en la Figura 3.5.

Las funciones *Bola* y *Aro* (Código 3.1) se corresponden con las ecuaciones del modelo no lineal del sistema representadas en el capítulo anterior en las Ecuaciones 2.1 y 2.2

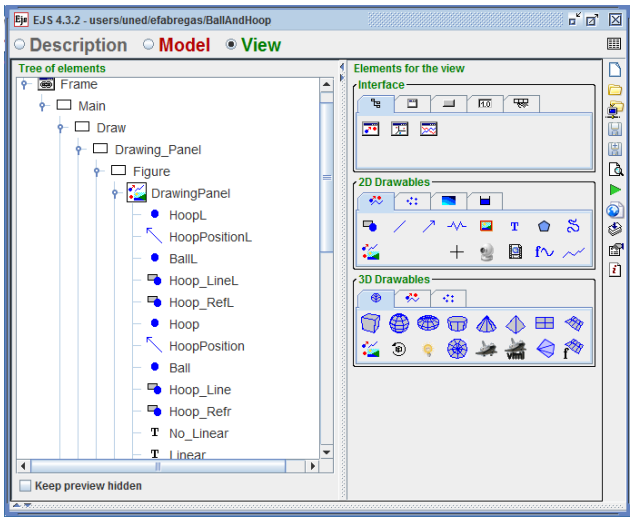


Figura 3.4. Vista de *EJS* de la aplicación *Cliente BA-S* en tiempo de desarrollo.

respectivamente. Mientras que las funciones *BolaL* y *AroL* representan las ecuaciones del modelo lineal del sistema se representan en el Código 3.2.

```
//Ecuación correspondiente a la dinámica de la bola en el modelo no lineal
public double Bola(double t, double aH, double vH, double y, double vy); {
 return(a22*(-bm*vH-c1*sin(aH-y/R)+tau(t))-a12*(-bb*vy/(r*r)-c2*sin(aH-y/R)))/Det;
}
//Ecuación correspondiente a la dinámica del aro en el modelo no lineal
public double Aro(double t, double aH, double vH, double y, double vy); {
 return(a11*(-bb*vy/(r*r)-c2*sin(aH-y/R))-a21*(-bm*vH-c1*sin(aH-y/R)+tau(t)))/Det;
}
```

Código 3.1. Funciones que corresponden al modelo no lineal del sistema.

```
//Ecuación correspondiente a la dinámica de la bola en el modelo lineal
public double BolaL(double t, double aHL, double vHL, double yL, double vyL); {
 return(a22*(-bm*vHL-c1*(aHL-yL/R)+tauL(t))-a12*(-bb*vyL/(r*r)-c2*(aHL-yL/R)))/Det;
}
//Ecuación correspondiente a la dinámica del aro en el modelo lineal
public double AroL(double t, double aHL, double vHL, double yL, double vyL); {
 return(a11*(-bb*vyL/(r*r)-c2*(aHL-yL/R))-a21*(-bm*vHL-c1*(aHL-yL/R)+tauL(t)))/Det;
}
```

Código 3.2. Funciones que corresponden al modelo lineal del sistema.

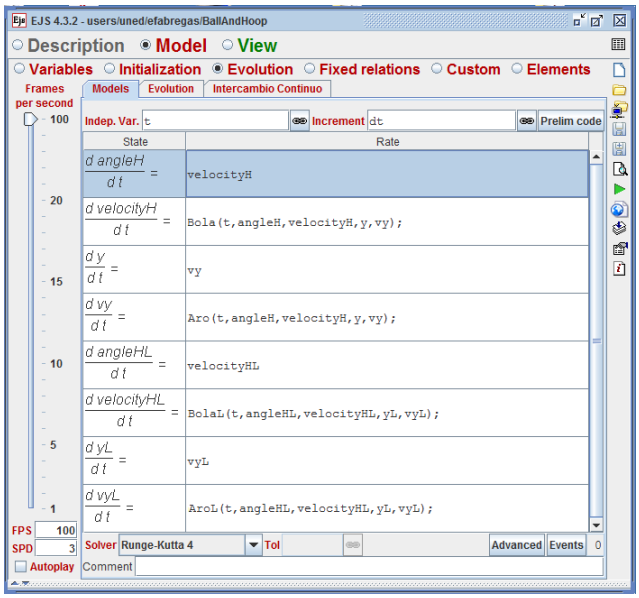


Figura 3.5. Modelo de *EJS* de la aplicación *Cliente BA-S* (ecuaciones diferenciales de la simulación).

Todas estas funciones reciben como parámetros las siguientes variables en este orden: tiempo (t), ángulo de posición del aro (aH), velocidad angular del aro (vHL), posición de la bola (y) y velocidad lineal de la bola (vy). Para el modelo lineal los parámetros son las mismas variables del modelo anterior, distinguidas con una L mayúscula al final del nombre de la variable que significa que son variables del modelo lineal. La inercia del aro I_a, la inercia de la bola I_b y los coeficientes *a11*, *a12*, *a21*, *a22*, *c1* y *c2* se calculan en la función *computeParameters* para el modelo no lineal de la forma mostrada en el Código 3.3.

```
public void computeParameters(){//Modelo no lineal
  Ia=M*R*R;                  //Inercia del aro
  Ib=2*m*r*r/5;              //Inercia de la bola
  a11=Ia+m*(R-r)*(R-r);      //Coeficiente inherente al modelo no lineal
  a12=-m*(R-r)*(R-r)/R;      //Coeficiente inherente al modelo no lineal
  a21=a12;                   //Coeficiente inherente al modelo no lineal
  a22=Ib/(r*r)+m*(R-r)*(R-r)/(R*R); //Coeficiente inherente al modelo no lineal
  Det=a11*a22-a12*a21;       //Determinante
  c1=m*g*(R-r);              //Coeficiente inherente al modelo no lineal
  c2=-m*g*(R-r)/R;           //Coeficiente inherente al modelo no lineal
}
```

Código 3.3. Parámetros que se usan en la evolución de la simulación.

Para el modelo lineal se calculan estos parámetros de la misma forma, en este caso con una función denominada *computeParametersL*. Estas funciones se invocan en la sección de inicialización. En esta sección luego se crea el objeto *videocam* que se utiliza posteriormente durante la conexión remota para obtener las imágenes de la cámara en red. A continuación se llama a la función *initControl* tanto para el modelo lineal como para el no lineal, tal como se observa en el Código 3.4.

```
//Parámetros de control de posición del aro (modelo no lineal)
initControl(pidContVH, old_v, aRefH, Kp_VH, tS, Ti_VH, Td_VH, 10*Ti_VH, 10, 1, TMin, TMax);
//Parámetros de control de posición del aro (modelo lineal)
initControl(pidContVHL, old_vL, aRefH, Kp_VH, tS, Ti_VH, Td_VH, 10*Ti_VH, 10, 1, TMin, TMax);
```

Código 3.4. Funciones que inicializan las variables del lazo de control.

Esta función es la encargada de la inicialización de la variable *pidContVH*, que contiene todos los parámetros relacionados con el lazo de control. En este caso los parámetros que recibe esta función son los siguientes:

- *pidContVH* se utiliza en la implementación del lazo de control y contiene todos los valores referentes al algoritmo de control.
- *old_value* es el valor anterior de la variable controlada (en este caso el ángulo de posición del aro).
- *aRefH* es la referencia en el lazo de control (en este caso del ángulo de posición del aro).
- *Kp_VH* es la constante proporcional del controlador *PI*.
- *tS* es el tiempo de muestreo del sistema.
- *Ti_VH* es la constante de tiempo integral del controlador *PI*.
- *Td_VH* es la constante de tiempo derivativo (en este caso es 0 porque el controlador solo tiene acciones Proporcional e Integral).
- *TMin* es el límite inferior de saturación de la señal de control.
- *TMax* es el límite superior de saturación de la señal de control.

Además durante el proceso de inicialización de la aplicación se establecen las condiciones para la comunicación con *MATLAB/Simulink*. Para ello se usa la librería *Jimc* [137] vinculada al *JIM_Server*.

```
//Crear la conexión con Simulink (objeto lm para la conexión con Simulink)
lm=new jimc.simulink("<matlab:62.204.199.192:2055>BallandHoop.mdl");
lm.setVarContext(this); //Contexto de las variables Java
//Conexión entre las variables Java y bloques Simulink
lm.linkVar("t","BallandHoop","param","t");                    //t-->t
lm.linkVar("aH","BallandHoop/ControlLoop/Sum","in","1");      //aRefH_r --> Sum in
lm.linkVar("Tau_r","BallandHoop/ControlLoop/Add","out","1");  //Tau_r --> Add out
lm.linkVar("aH_r","BallandHoop/ControlLoop/KCita","out","1"); //aH_r --> KCita out
lm.linkVar("chi_r","BallandHoop/ControlLoop/K3","out","1");   //chi_r --> K3
lm.linkVar("K","BallandHoop","param","K");                    //K --> K
```

Código 3.5. Vínculo de las variables de *EJS* con las de *MATLAB*.

Como se puede apreciar en el Código 3.5, primeramente se crea el objeto *lm* que posteriormente se usará para establecer la conexión con *MATLAB/Simulink* durante el funcionamiento de la aplicación en modo remoto. La función que crea este objeto, recibe como parámetros la *IP* y el puerto del ordenador que actúa como servidor además de el nombre del modelo *Simulink* que se debe ejecutar. Seguidamente se establece el vínculo entre las variables de la aplicación de *EJS* y los bloques del modelo *Simulink* mediante la función *linkVar*. Esta función recibe como parámetros la variable local de *EJS*, seguidamente el bloque del modelo *Simulink* donde se encuentra la variable remota a vincular, si esta es entrada o salida de ese bloque y finalmente el número que le corresponde en el bloque. En la sección de relaciones fijas se calculan algunas variables y parámetros que se usan en la aplicación y que están relacionadas directamente. Entre estas variables se encuentran la posición de la bola, el ángulo de la bola con respecto a la vertical, y la posición del aro entre otras. En el Código 3.6 se muestran estas relaciones tanto para el modelo no lineal como para el lineal.

En modo simulación, se obtiene la señal de control en cada paso de evolución a través de la función *tau* durante el funcionamiento de la aplicación, tal como se muestra en el Código 3.7.

Esta función calcula el valor de la señal de control dependiendo del tipo de control que esté seleccionado. Si la acción de control seleccionada es *zeros* entonces la señal de control que se obtiene es la señal sinusoidal para el experimento de los Ceros de Transmisión. Si la acción seleccionada es *init* la señal de control que se obtiene es 0 porque se necesita que el aro esté detenido. Si la acción de control seleccionada es *hoop* indica que se quiere controlar la posición del aro por lo que construye el lazo de

```
//Ecuaciones inherentes al modelo no lineal
chi=aH-y/R;          //Ángulo de inclinación de la bola en el modelo no lineal
xB=(R-r)*sin(chi);   //Coordenada x de la posición de la bola en el modelo no lineal
yB=-(R-r)*cos(chi);  //Coordenada y de la posición de la bola en el modelo no lineal
angleB=y/r;          //Velocidad angular de la bola en el modelo no lineal
hoopX=R*sin(aH);     //Coordenada x de la posición del aro en el modelo no lineal
hoopY=-R*cos(aH);    //Coordenada y de la posición del aro en el modelo no lineal
yRef=R*aRefH;
//Ecuaciones inherentes al modelo lineal
chiL=aHL-yL/R;          //Ángulo de inclinación de la bola en el modelo lineal
xBL=(R-r)*sin(chiL);    //Coordenada x de la posición de la bola en el modelo lineal
yBL=-(R-r)*cos(chiL);   //Coordenada y de la posición de la bola en el modelo lineal
angleBL=yL/r;           //Velocidad angular de la bola en el modelo lineal
hoopXL=R*sin(aHL);      //Coordenada x de la posición del aro en el modelo lineal
hoopYL=-R*cos(aHL);     //Coordenada y de la posición del aro en el modelo lineal
yRefL=R*aRefH;
```

Código 3.6. Variables que se calculan en cada paso de evolución de la aplicación.

```
public double tau(double t){
if(acc=="zeros")return amp*(sin(2*PI*f*t));              //Ceros de transmisión
  else if(acc=="init") return 0;                         //Inicio
  else if(acc=="hoop") return (contSig(pidContVH,aH+K*chi,"PI")-0.1*vH); //Aro
  else if(acc=="man") return constantTau;                //Manual
  else return (contSig(pidCon_BP,Kc*chi,"PI"))-Kv*vH;    //Bola
 }
```

Código 3.7. Cálculo de la señal de control.

control a través de la función *contSig*. Si la opción seleccionada es *man* esto indica que el control es manual por tanto la señal de control toma el valor del control deslizante correspondiente en la *GUI*. Mientras que si la opción seleccionada es *ball* entonces se construye el lazo de control para controlar la posición de la bola.

En modo remoto, luego de haber establecido la comunicación con *MATLAB/Simulink* si se desean modificar los parámetros del controlador, se usa la función *setControlParameters* como se muestra en el Código 3.8.

```
//Función para establecer los parámetros del controlador en la aplicación remota
private void setControlParameters(){
  lm.eval("set_param('BallandHoop/PI','numerator','["+Kp_VH+","+ Ti_VH+"]')");
}
```

Código 3.8. Función para enviar datos a *MATLAB*.

Esta función recibe como parámetros el bloque de *Simulink* donde se encuentra el parámetro a modificar, y seguidamente el valor del parámetro. En el Anexo A.1 se

muestra el código completo de la aplicación.

La vista de la simulación mostrada anteriormente en la Figura 3.4 da como resultado la interfaz gráfica de usuario de la aplicación *Cliente BA-S*, que se muestra en la Figura 3.6. Esta *GUI* consiste en una ventana principal que está dividida en tres paneles fundamentales: No. 1 Animación, No. 2 Parámetros y No. 3 Gráficos.

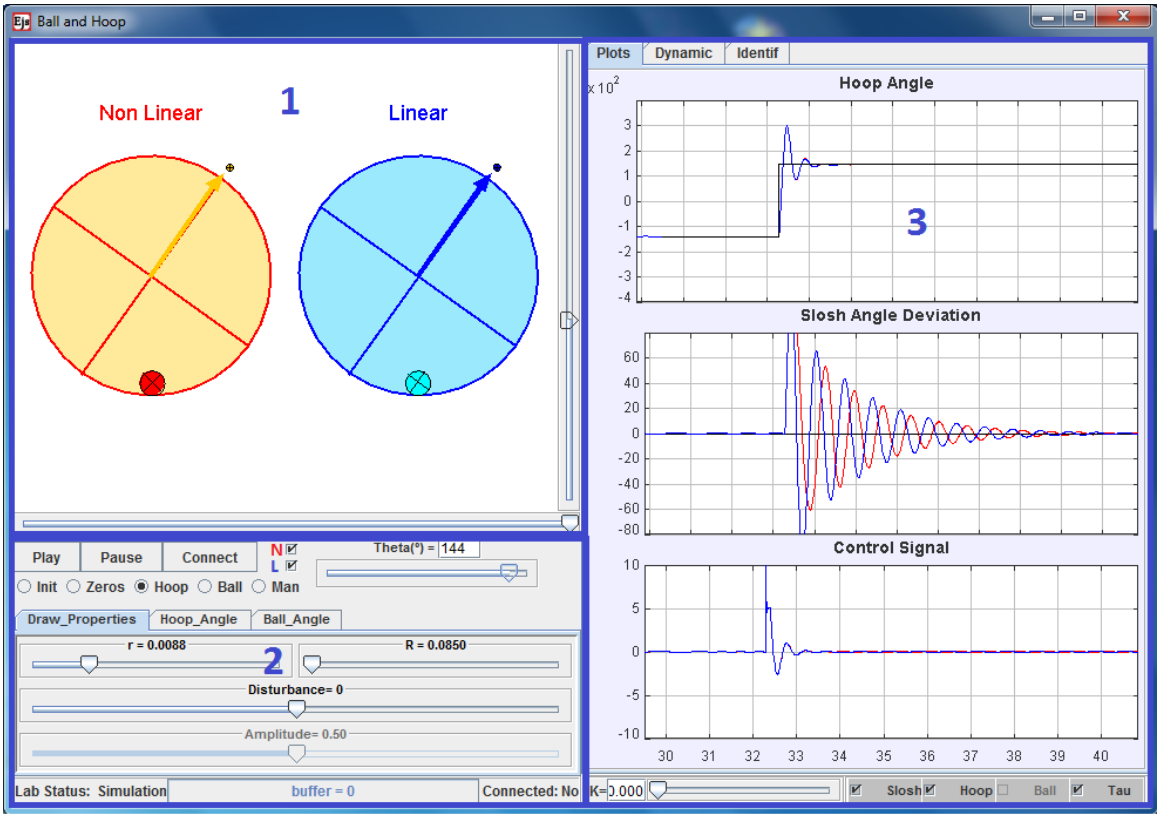


Figura 3.6. Ventana principal de la aplicación *Cliente BA-S*.

Panel No. 1

El panel No. 1 denominado Animación, muestra la vista interactiva de la simulación del sistema bola y aro. Esta animación posee dos modelos, el lineal (a la derecha) y el no lineal (a la izquierda). Ambos modelos solo difieren en las ecuaciones de su dinámica, el resto de sus propiedades físicas son las mismas que las de la planta real. El usuario tiene la posibilidad de mostrar u ocultar convenientemente cada uno de los modelos, dependiendo del experimento que desee realizar. Además, se pueden hacer algunas modificaciones a las propiedades de la simulación directamente sobre la animación

interactiva, como son:

- Cambiar el tamaño de la animación por medio de un control deslizante para visualizar con más detalle el modelo. Dicho control se encuentra situado en el borde derecho de este panel.
- Modificar el radio del aro en ambos modelos, arrastrando y soltando directamente sobre la animación.
- Variar la distancia entre ambos modelos hasta superponerlos para establecer comparación visual de su comportamiento, partiendo de la distancia inicial como se muestra en la Figura 3.6 hasta diminuir la distancia como se muestra la Figura 3.7. Esto se puede llevar a cabo por medio de un control deslizante situado en la parte inferior del dibujo.

Hay que tener en cuenta que si estas modificaciones se realizan en las propiedades físicas del modelo (radio del aro, radio de la bola, etc) esto influirá directamente en los datos de los experimentos obtenidos previamente. Por ejemplo, la frecuencia del experimento de los ceros de transmisión ya no sería la misma, por lo que habría que calcularla nuevamente.

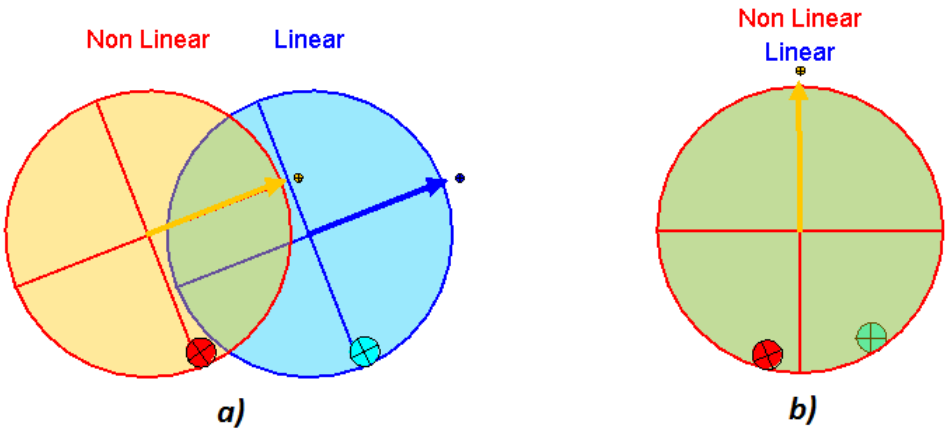


Figura 3.7. Variación de la distancia entre ambos modelos: a) d=7 b) d=0.

Panel No. 2

En el panel No. 2 denominado Parámetros, existen tres botones (*Play*,*Pause* y *Connect*) para manipular la simulación de cada uno de los experimentos. Con estos botones

se pueden ejecutar sobre la simulación cada una de las siguientes acciones: ponerla en marcha, detenerla o pausarla. También existe un botón cuya función es establecer el modo de trabajo remoto por medio de una conexión (*TCP/IP*) con la planta real situada en el laboratorio de la universidad. Mientras no se presione este botón la aplicación funcionará en modo virtual (simulación). En modo remoto, el usuario puede visualizar el experimento a través de una cámara *web*. La animación de la simulación se sustituye por la señal de vídeo de dicha cámara durante el tiempo que dure la conexión.

El experimento que se desee realizar se puede escoger por medio de unos botones de selección que se crearon para tal efecto. Estos botones se pueden usar tanto para el modo virtual como para el remoto. Al iniciar la aplicación la opción *Init* se encuentra seleccionada por defecto para establecer condiciones iniciales de operación. El resto de las opciones asociadas a los experimentos disponibles son las siguientes:

- *Zeros*: para el experimento de los ceros de transmisión.
- *Hoop*: para el experimento de control de posición de aro y desviación de la bola.
- *Ball*: para el experimento de control de posición de la bola.
- *Manual*: para el experimento de control manual de la planta.

Dependiendo del experimento que se seleccione, aparecerá un control deslizante que permitirá modificar varios parámetros asociados a dicha selección: la frecuencia de la señal de entrada para el experimento de los ceros de transmisión; la referencia de la posición para el experimento de control de posición del aro; la referencia para el experimento de control de la posición de la bola y el voltaje aplicado al motor en caso de que el control sea manual.

El panel denominado Parámetros (No. 2) posee además tres pestañas. A través de dichas pestañas el usuario puede modificar otros parámetros de la simulación tales como:

- La amplitud de la señal sinusoidal aplicada a la entrada en el experimento de los ceros de transmisión.
- Los radios del aro y la bola.
- Modificar los parámetros del controlador *PI* dependiendo del experimento.

- Variar el valor del factor de ganancia escalar (K_s) para demostrar el comportamiento de fase no mínima.
- Provocar una perturbación en la posición del aro en el experimento de control de la posición del aro.

Panel No. 3

El panel No. 3 denominado Gráficos, posee tres pestañas: *Plots*, *Dynamic* e *Ident*. En la pestaña *Plots* se muestran los gráficos de algunas de las variables de los modelos para el modo virtual. Mientras que para el modo remoto se muestran los gráficos de las variables de la planta real. El usuario puede seleccionar el gráfico que desea mostrar a través de un panel de selección que se encuentra en la parte inferior. La elección de los gráficos que se muestran depende del experimento que esté teniendo lugar, es decir, es posible que para algún experimento no sea necesario visualizar algunas de las variables que aparecen en los gráficos predefinidos. Este hecho se debe a que en cada experimento hay variables que resultan interesantes de observar mientras que otras no aportan mucha información al análisis del sistema. Por su parte, la pestaña *Dynamic* representa el gráfico del lugar geométrico de las raíces, que se utiliza para encontrar el valor apropiado de la ganancia K en el experimento del ángulo de desviación de la bola.

Se puede variar el valor de dicha ganancia a través de un control deslizante situado en la parte inferior de este panel. La posición de los polos complejos conjugados se representa por puntos azules en cada una de sus ramas. Al variar el valor de la ganancia, estos puntos se desplazan por sus correspondientes ramas, representando la posición de los polos para la ganancia correspondiente. Como se explicó anteriormente, el valor apropiado de esta ganancia es el que garantiza que la parte real de los polos complejos conjugados, sea la misma. Esto permite que el aro se mueva tan rápidamente como sea posible para alcanzar su posición final, causando la menor desviación posible en la posición de equilibrio de la bola. Al encontrar el valor apropiado, aparece una línea indicando que es el valor correcto, tal como se muestra en la Figura 3.8. La pestaña *Ident* muestra el gráfico de la entrada y la salida para el experimento de control manual con el

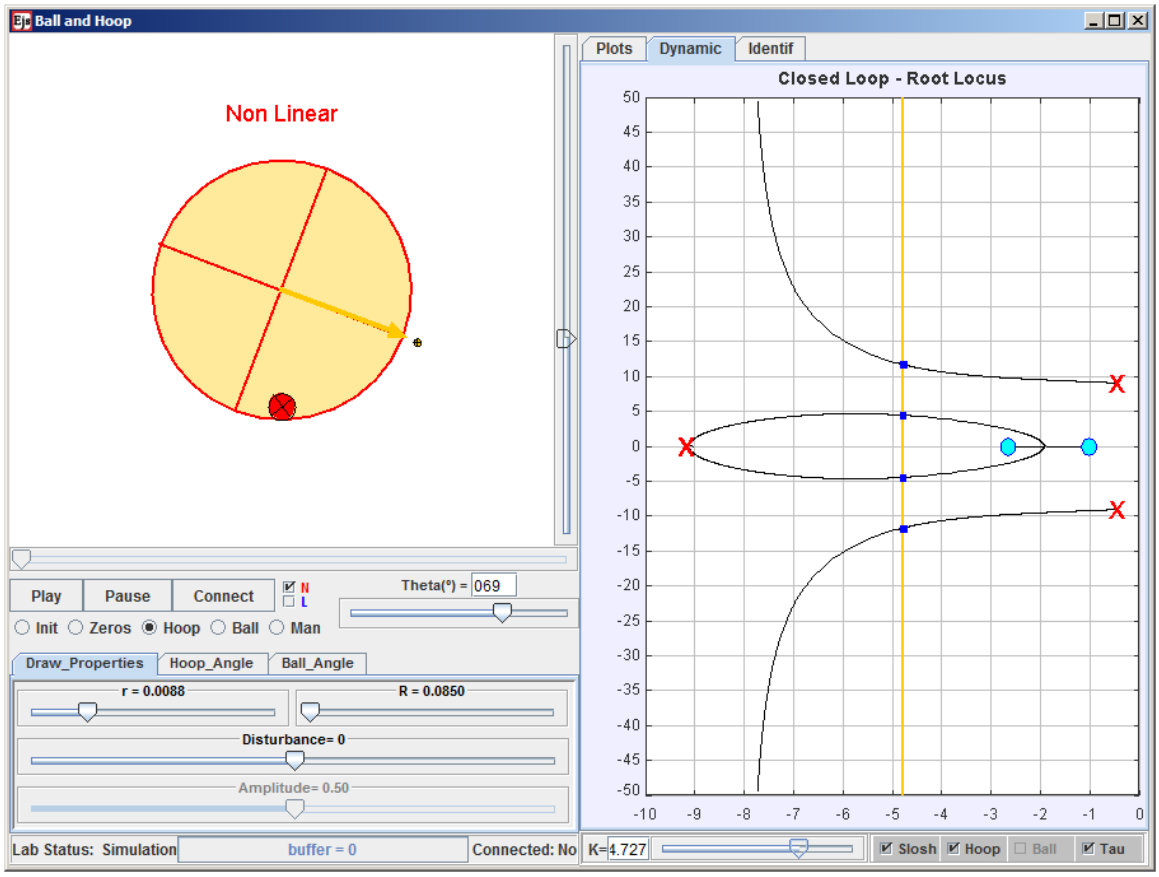


Figura 3.8. Lugar geométrico de las raíces que indica el valor correcto de la ganancia K.

objetivo de realizar la identificación del sistema con el fin de comprobar los resultados obtenidos con la planta real. En este caso se usan los valores de las propiedades físicas de la planta real y el resultado es $K = 4{,}73$.

3.1.2 La aplicación *Servidor BA-S*

La aplicación que se ejecuta en el servidor consiste en un modelo *Simulink* que controla la planta real. Esta aplicación se comunica con la planta real a través de la tarjeta de adquisición de datos *PCI-1711L* de *Advantech Corporation* y sus correspondientes controladores [140]. Además se emplea la librería *Simulink - Data Acquisition Toolbox* de *MATLAB* [141]. Esta última herramienta es un paquete de programas que provee un conjunto completo de utilidades para el trabajo con entradas/salidas analógicas y digitales. El *software* incluye una gran variedad de dispositivos de adquisición de datos. Además, permite la configuración del *hardware* externo así como la incorporación al

espacio de trabajo de *MATLAB*, los datos leídos para su inmediato análisis y procesamiento. También soporta el uso de *Simulink* con bloques que permiten la incorporación al modelo de elementos de *hardware* y datos en tiempo real [142].

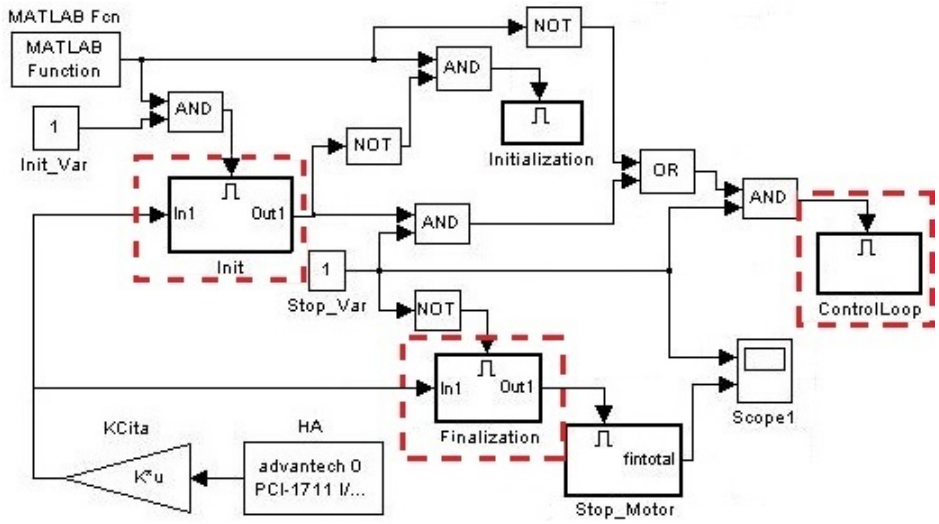


Figura 3.9. Modelo *Simulink* de la aplicación *Servidor BA-S*.

La Figura 3.9 muestra el modelo *Simulink* de la aplicación *Servidor BA-S*. Este modelo tiene la particularidad de que se utiliza para el control de una planta real. Por lo que se deben tener en cuenta ciertas consideraciones a las que normalmente no se les presta atención. Por ejemplo, el modelo debe tener un conjunto de bloques para el establecimiento de las condiciones iniciales durante la etapa inicial o arranque del experimento. Además debe poseer otro conjunto de bloques para establecer las condiciones necesarias una vez terminado el experimento. Esto no sucede cuando se trabaja solo en modo simulación ya que en dicho modo se pueden establecer las condiciones iniciales manualmente y además no se causa ningún daño a la planta real. En cambio, cuando se trabaja con el dispositivo de bola y aro, para iniciar un experimento, hay que tener en cuenta su estado inicial antes de realizarlo y su estado final al concluirlo, para no causar daños irreparables a la planta. Teniendo esto en cuenta la aplicación *Servidor BA-S* se ha diseñado para cumplir tres funciones fundamentales: establecer condiciones iniciales (bloque *Init*), controlar la planta real (bloque *ControlLoop*) y establecer las condiciones finales (bloque *Finalization*). Estos bloques se muestran resaltados en la Figura 3.9.

En la etapa inicial el bloque *Init* ejecuta un bucle de control de la posición del aro, garantizando que éste se encuentre en la posición inicial que requiere el experimento. En el caso de la bola, no es necesario establecer condiciones iniciales, puesto que cuando el aro está detenido, la bola se encuentra en su posición de equilibrio, que es la requerida para iniciar los experimentos. Una vez establecidas las condiciones iniciales, este bloque cede el control al bloque fundamental que es el *ControlLoop*. La Figura 3.10 muestra dicho bloque, que contiene el bucle de control representado en el diagrama de bloques de la Figura 3.9, para el experimento de control de la posición del aro y el de control de desviación de la bola de su posición de equilibrio. Por su parte, el bloque *Finalization* garantiza que el motor esta parado y con alimentación cero, al concluir el experimento.

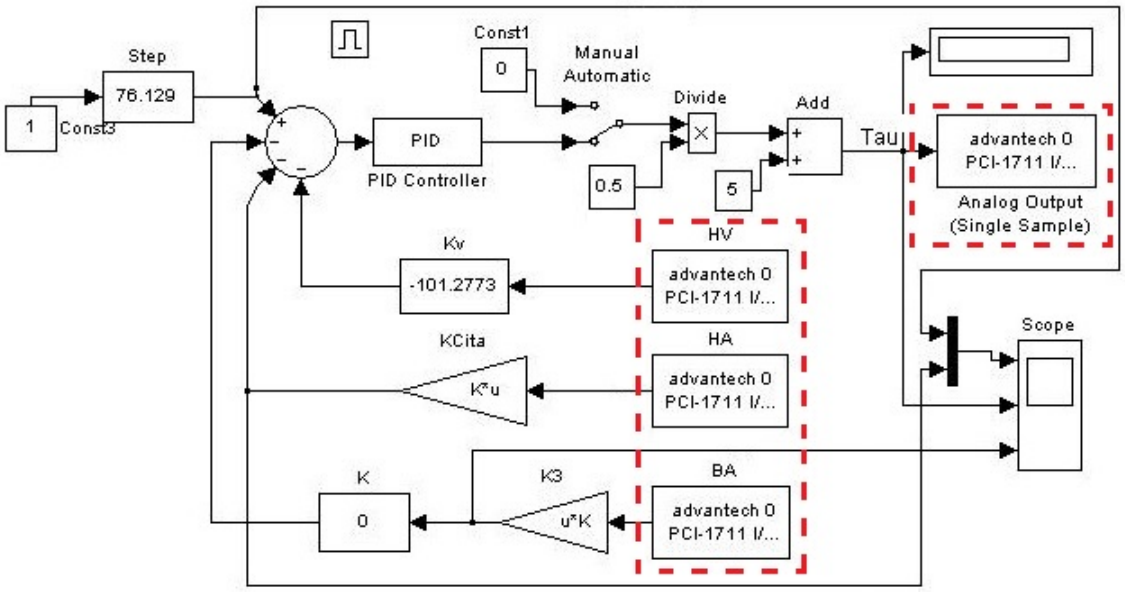


Figura 3.10. Implementación del bloque *ControlLoop* de la Figura 3.9.

El bloque *ControlLoop* es la implementación real del bucle de control presentado en el Capítulo 2 (ver Figura 2.8). Los bloques relacionados con la adquisición de datos son: velocidad del aro (HV), ángulo del aro (HA) y ángulo de la bola (BA). Estos bloques pertenecen al *Simulink - Data Acquisition Toolbox* de *MATLAB* y permiten obtener los datos de los sensores de la planta. La implementación del controlador *PI* se lleva a cabo con los valores obtenidos en la Sección 2.1.1.5. La señal de control obtenida a la salida del controlador se envía a la planta por medio del bloque *Analog Output*. Además, este bucle de control también tiene implementada la opción de control manual, que

está directamente relacionada con el control manual de la aplicación cliente.

3.1.3 Consideraciones prácticas de la plataforma con el sistema bola y aro (*EJS-MATLAB/Simulink*)

La plataforma consiste en una simulación interactiva del sistema bola y aro además de la conexión con una planta real de este sistema a través de *Internet*. La plataforma está basada en una arquitectura cliente - servidor. De la aplicación del lado del cliente se puede destacar que resulta muy interactiva y vistosa, por lo que atrae la atención y el interés de los usuarios. Obtener una simulación de un proceso tan dinámico como éste, resulta bastante complicado y se necesitan conocimientos avanzados de programación. Esto se evita en gran medida si se emplea *EJS* para el desarrollo de este tipo de aplicaciones. En cuanto a la aplicación del lado del servidor implementada en *Simulink*, se obtuvieron resultados que superaron las expectativas. Porque de antemano se sabía que tanto *MATLAB* como *Simulink* son herramientas enfocadas a cálculos matemáticos y simulaciones de control de procesos. Pero, utilizando los componentes adecuados para el trabajo con equipamiento real, se pueden obtener muy buenos resultados. Como *Simulink* es una potente herramienta bien conocida en el mundo del control automático, la implementación de bucles de control resulta bastante asequible. Si a esto se le añade el tratamiento con plantas reales, se aumenta su potencial y utilidad. Sin embargo, se debe destacar que los principales problemas que surgieron fueron exactamente con el *hardware*. En la adquisición de datos por ejemplo, se presentaron algunos problemas en cuanto a tiempo de muestreo, precisiones numéricas, etc. Esto se debe a que el tratamiento directo de *hardware* a bajo nivel, por parte de *MATLAB*, no está muy desarrollado y se puede decir que se encuentra en fase inicial comparado con otros *softwares* más especializados en el tema. Por lo que no se descarta que para un futuro no muy lejano, se puedan obtener resultados aún mejores con los productos de *MathWorks*.

3.2 Sistema bola y aro (*EJS-LabVIEW*)

En esta sección se describen las características fundamentales del laboratorio virtual y remoto con el sistema bola y aro con *EJS-LabVIEW*. Es necesario señalar que la

estructura y arquitectura de este laboratorio es similar a la anterior. Las principales diferencias se encuentran en el lado del servidor, donde para el control de la planta real se ha empleado *LabVIEW* y la misma tarjeta de adquisición de datos *PCI-1711L* de *Advantech*.

3.2.1 Arquitectura del laboratorio virtual y remoto

Para tratar de evitar los problemas mencionados en la aplicación servidor con *MATLAB/Simulink*, se decidió utilizar un programa más potente en cuanto a tratamiento de *hardware*. Entre las herramientas existentes dedicadas a la adquisición de datos y control de plantas reales destaca por su versatilidad y facilidad de manipulación *LabVIEW*. Esta herramienta resulta de gran utilidad para la interacción directa con el *hardware*, permitiendo la programación a muy bajo nivel. Lo que en este caso resulta ser una ventaja porque permite tener control total sobre los datos. La forma de programación se hace usando un diagrama de bloques donde se representan los diferentes elementos y funciones del programa. Además tiene una interfaz gráfica que es con la que interaccionará el usuario final de la aplicación. En este caso, al igual que en la variante anterior, la arquitectura del laboratorio virtual y remoto está basada en una estructura cliente - servidor. Esto significa que el ordenador que funciona como servidor también lo hace como controlador para la planta física. La Figura 3.11 muestra una representación detallada de esta arquitectura.

3.2.2 Aplicación *Cliente BA-L*

La aplicación cliente en este caso se denomina *Cliente BA-L* y es muy similar a la del laboratorio anterior. En el caso de la simulación (modo virtual) no hay ninguna diferencia. Las diferencias radican en el modo de funcionamiento con la planta real (modo remoto). La aplicación que se ejecuta en el servidor ha sido desarrollada en *LabVIEW*. Por lo que la comunicación con dicha aplicación se establece de la misma forma pero usando otra herramienta desarrollada en el Departamento de Informática y Automática de la UNED denominada *JIL Server (Java-Internet-LabVIEW Server)* [60, 133]. En este caso se crea el objeto *vi* a través del cual se establecerá la comunicación

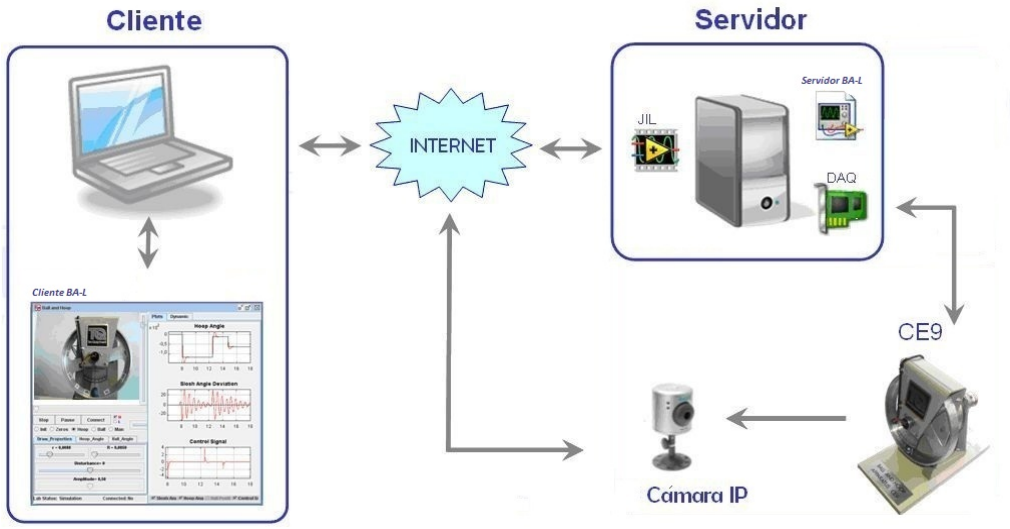


Figura 3.11. Arquitectura de la aplicación con *EJS-LabVIEW*.

con *LabVIEW* tal como se muestra en el código 3.9.

```
//Objeto vi que se usa en la comunicación remota
vi=new jil.Jil("<labview:62.204.199.192:2055>");
```

Código 3.9. Objeto *vi* que se usará posteriormente en la comunicación con *LabVIEW*.

Como se puede apreciar, esta función recibe como parámetro la *IP* y el puerto del ordenador que actúa como servidor para establecer la comunicación. En este servidor debe estar ejecutándose continuamente la aplicación *Jil_Server*. En este caso, las variables de la aplicación local están vinculadas con las variables de la aplicación remota de otra forma. En cada paso de evolución de la aplicación, se obtienen las variables de la aplicación remota tal como se muestra en el Código 3.10.

```
if(((jil.Jil)vi).isConnected()){          //Si el objeto vi está conectado
  buffer=((jil.Jil)vi).getDataAvailable(); //Se obtienen los datos del buffer
  bufferCB=((jil.Jil)vi).getDataAvailableCB();
  if(((jil.Jil)vi).isRunning()){          //Si el vi remoto se está ejecutando
   if(buffer>1){getValues();}             //Si el buffer tiene más de un dato se leen
  }
  getVideo();                             //Se obtiene una imagen de la cámara IP
}
```

Código 3.10. Segmento de código que obtiene los datos de *LabVIEW*.

Si la comunicación está establecida y hay datos disponibles, éstos se leen mediante la función *getValues* tal como se muestra en el Código 3.11. Esta función obtiene los valores de las variables vinculadas especificando el tipo de dato (*double*, *int*), etc.

```
public void getValues(){ //Función para obtener los valores de la planta real
  AngleH_r=((jil.Jil)vi).getDouble("HA(ind)"); //Obtiene el ángulo del aro
  chi_r=((jil.Jil)vi).getDouble("BA(ind)");    //Obtiene el ángulo de la bola
  Tau_r=((jil.Jil)vi).getDouble("U(ind)");     //Obtiene la señal de control
  aRefH_r=((jil.Jil)vi).getDouble("SP_Hoop_r(ind)"); //Obtiene la referencia del aro
}
```

Código 3.11. Función que vincula las variables de *EJS* con las de *LabVIEW*.

Por otra parte, si se desea modificar alguno de los parámetros de la aplicación remota, se realiza por medio de la función *setValues*, tal como se muestra en el Código 3.12. Esta función se puede usar tanto para modificar varios parámetros como para modificar solo un parámetro, dependiendo de lo que se necesite hacer.

```
public void setValues(){ //Función para establecer valores en la aplicación remota
    ((jil.Jil)vi).setValue("Ring(con)",sel); //Establece el experimento a realizar
    ((jil.Jil)vi).setValue("Frec(con)",f);   //Frecuencia de ceros de transmisión
}
```

Código 3.12. Función para establecer valor de controles en la aplicación de *LabVIEW*.

En este caso se muestra la función *setValues* para establecer los valores de frecuencia y del control de selección del experimento para los Ceros de Transmisión. Al igual que en el caso de la versión de *MATLAB/Simulink* del laboratorio virtual y remoto, los cálculos y la interacción con la planta real se realizan en el lado del servidor; quedando la aplicación cliente solo para mostrar los datos y para soportar la interacción del usuario con el laboratorio en sus dos modos de funcionamiento. En el Anexo A.2 se muestra el código completo de esta aplicación.

3.2.3 Aplicación *Servidor BA-L*

Como se mencionó anteriormente el lado del servidor es un ordenador que ejecuta una aplicación desarrollada en *LabVIEW* para controlar la planta real. Esta aplicación

se denomina *Servidor BA-L* y se comunica con la planta real a través de la tarjeta de adquisición de datos *PCI-1711L* de *Advantech* y sus correspondientes controladores [140]. La Figura 3.12 muestra la interfaz de usuario de dicha aplicación.

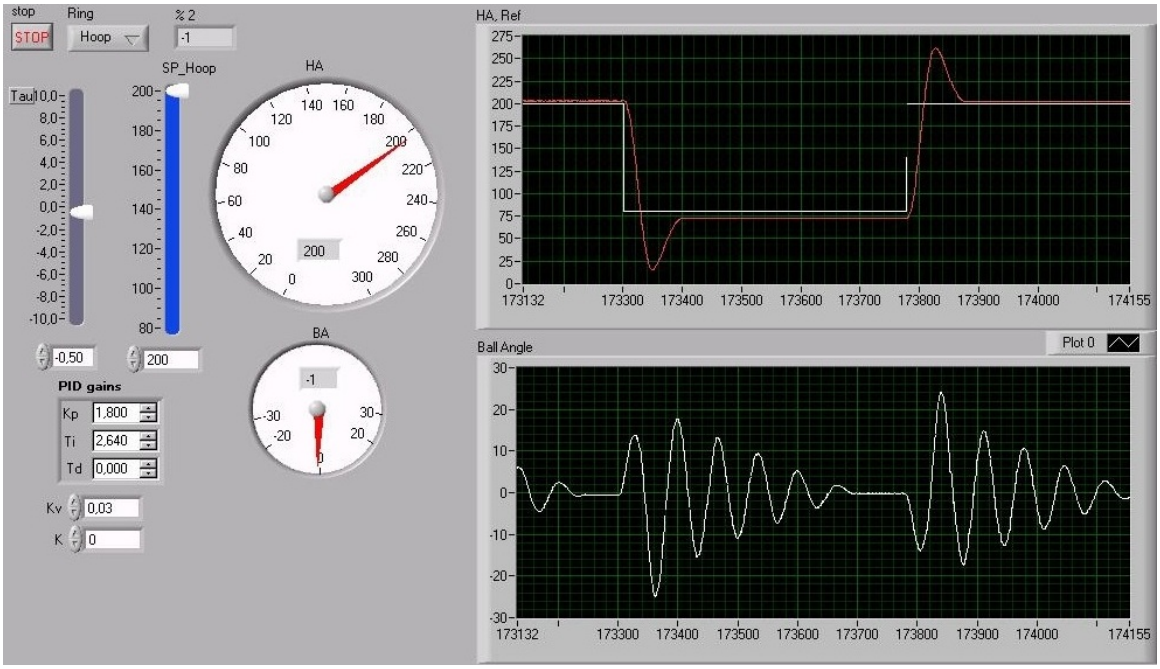


Figura 3.12. Interfaz de usuario de la aplicación que se ejecuta en el servidor.

En el panel frontal de esta aplicación se representan todas las variables del sistema. En el panel izquierdo se muestra un selector por medio del cual se selecciona el experimento que se desea realizar, el voltaje que se aplica al motor y la referencia de la posición del aro. Además de los parámetros del controlador *PI* y la ganancia *K* correspondiente al control de desviación de la bola de su posición de equilibrio. En el panel derecho de la aplicación se muestran los valores del ángulo de posición del aro y del ángulo de posición de la bola. Todas estas variables se encuentran conectadas con las variables de la aplicación *Cliente BA-L*. Por tanto, en modo remoto, cuando el usuario interaccione con la aplicación cliente, estas modificaciones se pasan a la aplicación *Servidor BA-L*. De esta forma no se necesita visualizar dicha interfaz, ya que toda la interacción que establece con esta aplicación la efectúa a través de la aplicación *Cliente BA-L*.

La Figura 3.13 muestra el diagrama de bloques de la aplicación *Servidor BA-L*. Como se puede apreciar al igual que en en la aplicación anterior, se necesitan un bloque

de inicialización y otro de finalización. Debido a que se trabaja con equipamiento real, es necesario establecer condiciones iniciales y finales de trabajo para los experimentos.

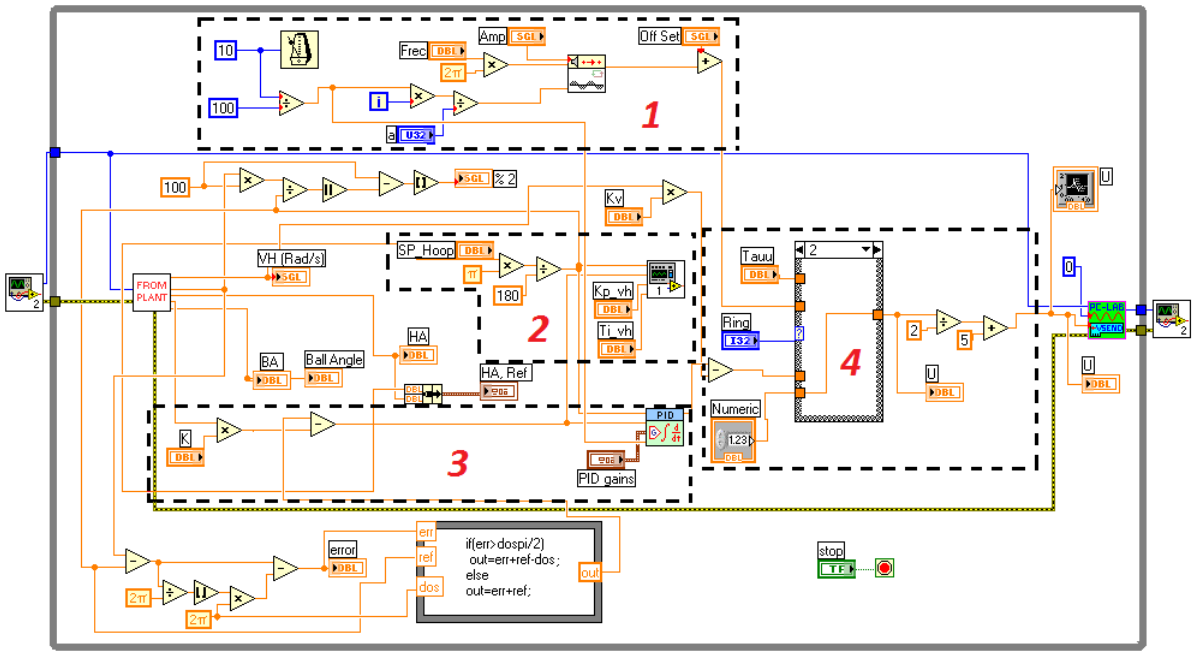


Figura 3.13. Diagrama de bloques de la aplicación *Servidor BA-L*.

Como se puede apreciar, el recuadro No. 1 de la Figura 3.13 muestra la implementación correspondiente al experimento de los ceros de transmisión. Para el cual se puede variar tanto la amplitud como la frecuencia de la señal sinusoidal que se aplica a la entrada del motor. Mientras que el recuadro No. 2 muestra el bucle de control correspondiente al experimento de control de la posición del aro con su respectivo controlador *PI*. En este caso, se implementa el controlador manualmente. Mientras que el recuadro No. 3 muestra el bucle de control correspondiente al experimento de control del ángulo de desviación de la bola. En este caso, se emplea un controlador *PI* perteneciente a las librerías de ingeniería de control que posee *LabVIEW*. El recuadro No. 4 muestra un control de selección del cuál dependerá la entrada que se le aplique al motor. Este selector está vinculado a los controles de selección de experimentos de la aplicación cliente. Dependiendo del valor de este selector se escogerá la salida de uno de los tres bucles de control, para aplicárselo a la planta como entrada. Es necesario señalar que tanto en este laboratorio como en el anterior (con *MATLAB*) el retardo de la red no influye en el bucle de control, ya que el controlador está implementado en el lado del

servidor. Así, cuando el usuario varía la referencia, este dato se transmite a la aplicación servidor, que calcula la señal de control correspondiente a esta referencia. El dato calculado se envía de vuelta a la aplicación cliente unido al valor de referencia (para visualizarlos) sin contar el tiempo de la red, es decir, el dato de referencia se devuelve como si el cambio hubiera sido local. Por esta razón el retardo de la red no influye ni en el control de la planta, ni en la visualización de los resultados. Para estudios futuros sería interesante implementar el controlador en el lado del cliente. De esta forma habría que tener en cuenta el retardo de la red en el bucle de control. Esto haría el problema mucho más interesante, pero además es otro tipo de problema mucho más complejo.

3.2.4 Consideraciones prácticas de la plataforma con el sistema bola y aro (*EJS-LabVIEW*)

Los cambios fundamentales de esta plataforma con respecto a la anterior se han realizado en la aplicación del servidor. Se han aprovechado todas las ventajas de la aplicación cliente en cuanto a interactividad y atracción de la atención e interés de los usuarios. De este modo, *EJS* ha contribuido de manera importante en el desarrollo de esta simulación interactiva, sobre todo, como se ha mencionado, porque no es necesario tener muchos conocimientos de programación para obtener una aplicación de alta calidad y fiabilidad. De la aplicación del lado del servidor, en contraste con la anterior, se puede decir que se obtuvieron los resultados esperados.

El uso de *LabVIEW* como herramienta para la interacción con plantas reales a través de una tarjeta de adquisición de datos, proporciona importantes beneficios. Eliminando sobre todo los errores en las mediciones de las variables de la planta real. Además, al ser un programa creado para el desarrollo de aplicaciones de ingeniería, la programación con esta herramienta es muy sugerente e intuitiva, puesto que todo el código se desarrolla con bloques de construcción. *LabVIEW* tiene varias librerías específicas con bloques predefinidos que ayudan mucho en el desarrollo de aplicaciones de control de *hardware*. La implementación de bucles de control resulta pues relativamente sencilla. Por otra parte, aunque no hay que enfrentarse con la interfaz gráfica de usuario de esta aplicación, también resulta relativamente sencillo obtener una aplicación

totalmente operativa que funcione correctamente.

3.3 Sistema bola y plato (*EJS-Visual C#*)

En esta sección se describen las características fundamentales del laboratorio virtual y remoto con el sistema bola y plato. La arquitectura de este laboratorio es la misma que la del laboratorio con el sistema bola y aro (cliente-servidor), presentada en la Sección 3.1. Aunque la arquitectura es la misma, existen diferencias desde el punto de vista de implementación tanto en el lado del cliente como en el lado del servidor. La aplicación cliente es totalmente diferente debido a que el sistema no es el mismo y por tanto su simulación será distinta. Debido a que la aplicación servidor, que controla la planta real, fue desarrollada en *Visual C#*, fue necesario desarrollar la interfaz de comunicación entre *Visual C#* y *EJS* a través de *Internet*. Esta interfaz se ha denominado *JIC* (*Java Internet C#*).

3.3.1 Arquitectura del laboratorio virtual y remoto

Como se mencionó anteriormente en la Sección 2.1.2.3 la planta real y la aplicación local que la controla se desarrollaron en el Departamento de Energía Eléctrica, Sistemas y Automatización de la Universidad de Gante en Bélgica. Sobre la base de esta planta y su aplicación de control, se procedió a desarrollar un laboratorio virtual y remoto. La Figura 3.14 muestra una representación de la arquitectura del laboratorio.

Como se puede apreciar en el lado del cliente, al igual que en el caso anterior, la aplicación *Cliente BP-C* tiene dos modos de trabajo: virtual (simulación) y remoto (control de la planta real). Esta aplicación tiene implementada la interfaz *JIC* representada en la Figura 3.14 y la comunicación por el puerto *TCP*. Esta parte de la interfaz al estar incorporada en *EJS* está implementada en código *Java*. En modo local, la aplicación ejecuta una simulación en 3D del sistema y en modo remoto, cuando se conecta a la planta real a través de *Internet*, la animación de la simulación se sustituye por la señal de vídeo de la *Cámara IP* que visualiza el desarrollo de los experimentos. En el lado del servidor, la interfaz *JIC* se incorporó al código de la aplicación local de control

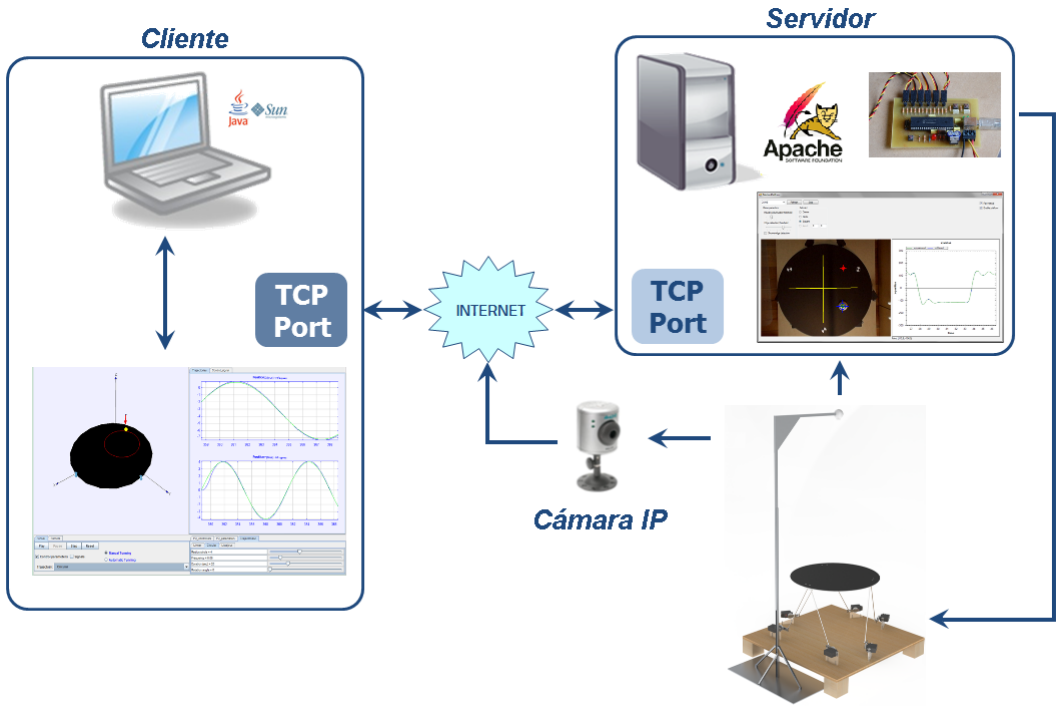


Figura 3.14. Arquitectura del laboratorio virtual y remoto con el sistema bola y plato.

de la planta, por lo que esta parte de la interfaz (representada por el puerto *TCP*) se desarrolló en *Visual C#*. Por eso se representa en color diferente, para indicar que es la implementación del *Socket TCP* en dos lenguajes de programación diferentes. Es necesario destacar que la aplicación de control de la planta real, hace un procesamiento de las imágenes de la cámara para obtener la posición de la bola. El procesamiento de imágenes consume muchos recursos computacionales del servidor, por lo que debe tenerse en cuenta este aspecto ya que puede influir introduciendo retardos en la comunicación y el control.

3.3.2 Aplicación *Cliente BP-C*

La aplicación *Cliente BP-C* ha sido desarrollada en *EJS* lo que implica que desde el punto de vista de implementación tiene una estructura similar a la aplicación *Cliente BA-S* descrita en la Sección 3.1.1 (Descripción, Vista y Modelo). La Figura 3.15 muestra la vista de *EJS* de la aplicación *Cliente BP-C* en tiempo de diseño. La vista tiene una estructura de árbol jerárquico de objetos predefinidos de *EJS* para la construcción

de la simulación.

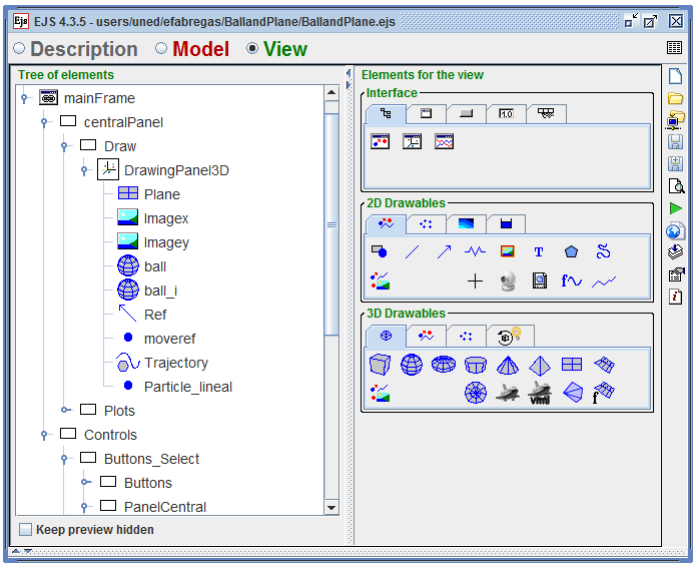


Figura 3.15. Vista en tiempo de diseño de la aplicación *Cliente BP-C*.

La principal diferencia con el resto de simulaciones creadas durante el desarrollo de esta tesis es que la animación de esta simulación es en tres dimensiones (3D). Esto hace que la implementación sea un poco más compleja, pero a la vez hace la animación mucho más vistosa y atractiva para los usuarios.

Al igual que en los demás casos, la simulación tiene un gran número de variables que se definen en la Sección *Modelo/Variables*. Las más destacadas son las coordenadas de la posición de la bola sobre el plato y los ángulos de inclinación del plato. En la sección *Modelo/Inicialización* se definen la posición inicial de la bola y la referencia de la posición de la bola tal como se muestra en el segmento de Código 3.13.

```
//Coordenadas del centro de la bola (ejes x,y,z)
xcb=(xcb0+xb)*cos(Alpha)-zcb0*sin(Alpha);
ycb=-(xcb0+xb)*sin(Beta)*sin(Alpha)+(ycb0+yb)*cos(Beta)-zcb0*sin(Beta)*cos(Alpha);
zcb=(xcb0+xb)*cos(Alpha)*sin(Alpha)+(ycb0+yb)*sin(Beta)+zcb0*cos(Beta)*cos(Alpha);
//Referencias de la posición de la bola (ejes x,y,z)
Refx=Refx0*cos(Alpha)-Refz0*sin(Alpha);
Refy=-Refx0*sin(Beta)*sin(Alpha)+Refy0*cos(Beta)-Refz0*sin(Beta)*cos(Alpha);
Refz=Refx0*cos(Beta)*sin(Alpha)+Refy0*sin(Beta)+Refz0*cos(Beta)*cos(Alpha);
```

Código 3.13. Inicialización de la aplicación *Cliente BP-C*.

La Figura 3.16 muestra la sección *Modelo/Evolución*. Como se puede apreciar, las ecuaciones implementadas en la simulación se corresponden con las ecuaciones del modelo linealizado del sistema presentado en las Ecuaciones 2.31 y 2.32.

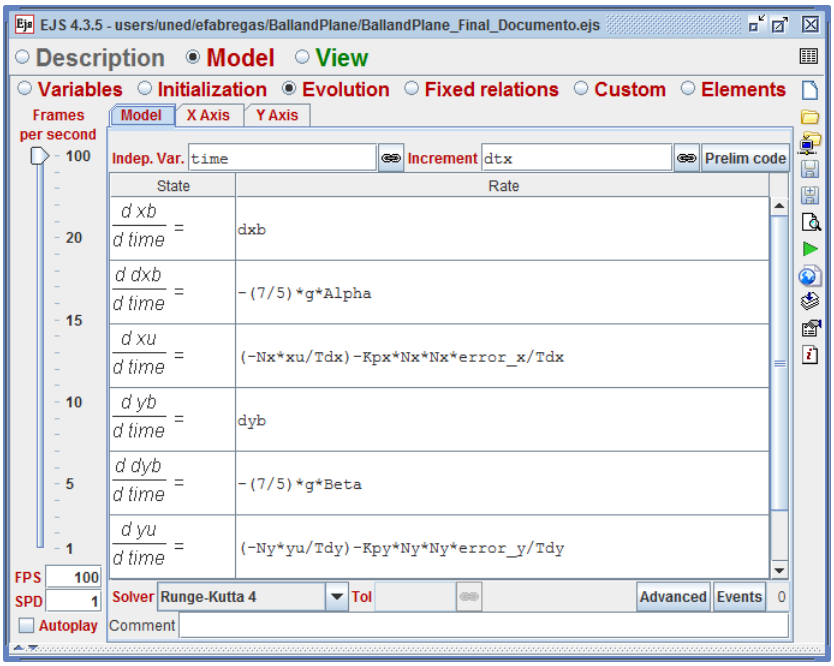


Figura 3.16. Modelo de *EJS* de la aplicación *Cliente BP-C* en tiempo de desarrollo.

Las variables relacionadas con el modelo son las siguientes:

- *xb* es la coordenada x de la posición de la bola (x_b).
- *dxb* es la velocidad de la bola en el eje x ($\dot{x}_b$).
- g es la aceleración de la gravedad.
- *Alpha* es el ángulo de inclinación del plato correspondiente al eje x (α).
- *yb* es la coordenada y de la posición de la bola (y_b).
- *dyb* es la velocidad de la bola en el eje y ($\dot{y}_b$).
- *Beta* es el ángulo de inclinación del plato correspondiente al eje y (β).

La tercera ecuación es la implementación del controlador *PD* para el eje x del plato mientras que la sexta ecuación es su homóloga para el eje y. Las variables que se relacionan en estas ecuaciones para cada uno de los ejes son las siguientes:

- *xu* e *yu*, señales de control de ángulos del plato.
- *Tdx* y *Tdy*, constantes de tiempo derivativo del controlador *PD*.

- *Kpx* y *Kpy*, constantes de acción proporcional del controlador *PD*.
- *error_x* y *error_y*, errores de posición de la bola.
- *Nx* y *Ny*, factores del filtro de las acciones derivativas del controlador *PD* [67].

En cada paso de evolución de la simulación también se realizan las siguientes acciones dependiendo de la trayectoria seleccionada:

- Se actualiza la posición del plato.
- Se calcula la referencia de posición de la bola para cada uno de los ejes.
- Se calcula el error de posición de cada uno de los ejes.
- Se calcula el ángulo de posición del plato para cada uno de los ejes.
- Se actualiza el tiempo de duración de la trayectoria y se verifica si el tiempo ha llegado a su fin.

El segmento de Código 3.14 muestra cómo se realizan estas acciones para el eje x en el caso del control de posición de la bola en un punto y en el caso de la trayectoria circular. El resto de las acciones para el eje y así como para el eje z se pueden observar en el Anexo A.3.

```
arreglo[0]=dirAplate[0]; //Actualización de la posición del plato
if(manual && TrajectorySelected==0){ //Punto de equilibrio
 error_x=-(Refx0-xball);         //Se obtiene el error
 Alpha=(Kpx*(Nx+1)*error_x+xu); //Ángulo del plato para x de la bola
}
else if(manual && TrajectorySelected==1){  //Trayectoria Lineal
 Refx0=pos_actual_x+(final_x-pos_actual_x)*time_traj_x/duration_lineal;
 final_x0=Refx0;                    //Se obtiene la referencia
 error_x=-(Refx0-xball);            //Se calcula el error
 Alpha=(Kpx*(Nx+1)*error_x+xu);     //Se calcula la señal de control
 time_traj_x=time_traj_x+dtx;       //Se calcula el tiempo de la trayectoria
 if(time_traj_x≥duration_lineal){
   _pause(); //Se detiene si el tiempo es mayor que el establecido
 }
}
```

Código 3.14. Evolución de la aplicación *Cliente BP-C*.

La implementación de la interfaz *JIC* en la aplicación *Cliente BP-C* se puede apreciar en el segmento de Código 3.15. En modo remoto la comunicación se establece

usando el protocolo *TCP* y funciona de forma tal que la aplicación remota está a la escucha permanentemente por el puerto 1238 (seleccionado aleatoriamente), para atender las solicitudes que se realicen desde la aplicación *Cliente BP-C*. Para ello se implementa la función *enviar* que se emplea cada vez que se necesita enviar datos (parámetros de los controladores, tipo de trayectoria, etc) a la aplicación de control de la planta real. La función *enviar* transfiere la cadena de datos y además especifica si se requiere una respuesta o no. Para ello ejecuta los siguientes pasos:

- Se crea el *Socket TCP*.
- Se crea el *buffer* de escritura/salida (*writer*).
- Se crea el *buffer* de lectura/entrada (*buf*).
- Se envía la cadena de datos (*dato*).
- Se "limpia" el *buffer* de escritura/salida.
- Si se necesita algún dato de vuelta (*receive=true*), se espera la respuesta del servidor durante un tiempo (50 ms).
- Se lee la cadena de datos enviada desde el servidor.
- Se divide la cadena por ";" que ha sido el símbolo que se ha seleccionado como delimitador de cadena.
- Si el primer valor de la cadena es 2, indica que se han enviado datos a la planta real para realizar la identificación y por tanto, se le informa al usuario mediante una ventana de mensajes, que el experimento terminará en breves instantes.
- Finalmente se cierran el *buffer* de escritura y el *Socket* de comunicaciones.

En la Figura 3.17 se muestra la interfaz gráfica de la aplicación *Cliente BA-S*. Esta *GUI* consiste en una ventana principal que está dividida en tres paneles fundamentales: No. 1 Animación, No. 2 Parámetros y No. 3 Gráficos.

Panel No. 1

El panel No. 1 denominado Animación, muestra la vista interactiva en 3D de la simulación del sistema bola y plato cuando la aplicación trabaja en modo virtual. En este caso, para el experimento de seguimiento de una trayectoria circular.

```
public void enviar(String dato, boolean receive){
 socket=new Socket("157.193.82.79",1238); //Socket que escucha por el puerto 1238
 writer=new PrintWriter(socket.getOutputStream(),true);
 buf=new BufferedReader(new InputStreamReader(socket.getInputStream()));
 writer.println(dato);    //Se envía el dato
 writer.flush();          //Se limpia el buffer de envío
 if(receive){             //Si se necesita respuesta
  Thread.sleep(50);       //Se espera la respuesta del servidor
  buf=new BufferedReader(new InputStreamReader(socket.getInputStream()));
  while(!buf.ready()){}  //Mientras haya datos disponibles
  cad=buf.readLine();  //Se leen dichos datos
  temp=cad.split(";"); //Se divide la cadena recibida
  if(Double.valueOf(temp[1])==2) //Si el segundo valor de la cadena es 2
   _view.alert("Experiment","Ball and Plate","Experiment will finish in seconds");
 }  //Se indica que se está enviando los datos del experimento a la planta real
  writer.close();  //Se cierra el buffer de escritura
  socket.close();  //Se cierra el Socket de comunicaciones
}
```

Código 3.15. Implementación de la interfaz *JIC* en la aplicación *Cliente BP-C*.

Las propiedades físicas de la simulación tales como diámetro del plato, masa y radio de la bola, tienen los mismos valores que estas propiedades de la planta real. El usuario puede hacer algunas modificaciones a las propiedades de la simulación directamente sobre la animación interactiva, como son:

- Rotar la animación en cualquiera de las tres dimensiones para visualizar con más detalle la animación.
- Modificar la referencia de posición de la bola.
- Modificar las propiedades de la trayectoria seleccionada.

Panel No. 2

El panel No. 2 denominado Parámetros, tiene tres pestañas (*Virtual*, *Remote* e *Identification*). Estas pestañas están vinculadas a los modos de operación de la aplicación y a la identificación de la planta remota. La pestaña *Virtual*, en su parte izquierda, tiene cuatro botones para controlar la ejecución de la simulación (*Play*, *Pause*, *Step* y *Reset*). Además tiene un menú de selección a través del cual el usuario puede escoger el experimento que desea realizar. Entre los experimentos que se pueden llevar a cabo se encuentran el control de posición de la bola en un punto (*Equilibrium Point*, que permite establecer la referencia para el control de posición de la bola) y el seguimiento de una trayectoria. Las trayectorias disponibles en la aplicación son:

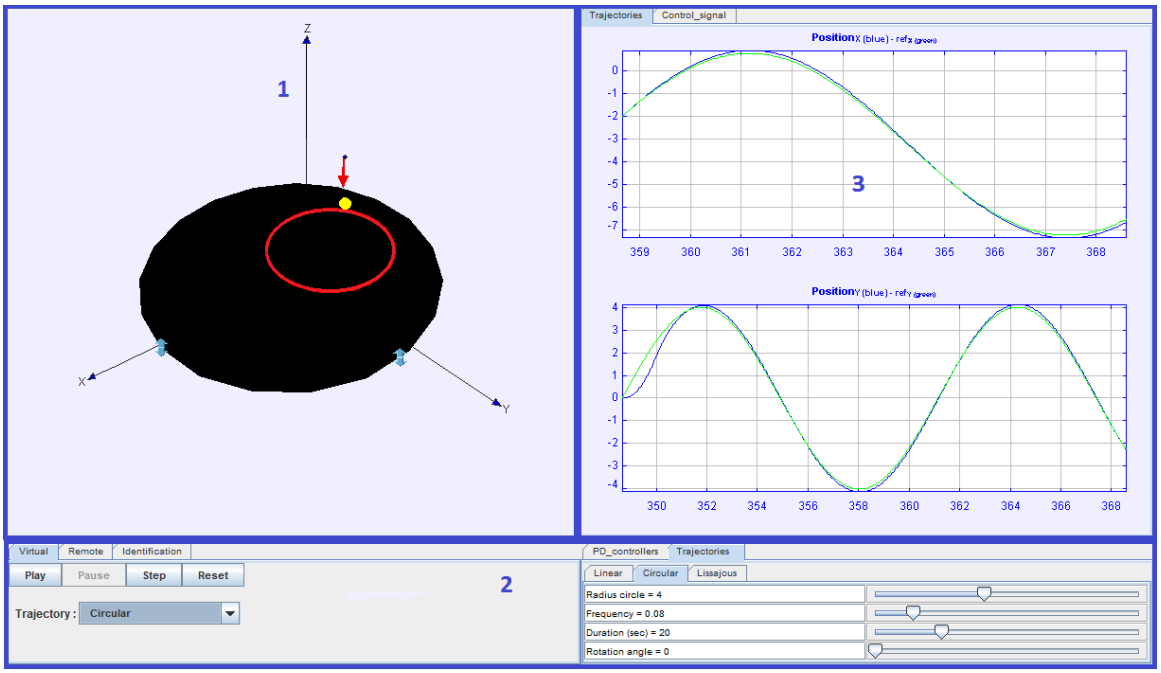


Figura 3.17. Interfaz gráfica de usuario de la aplicación *Cliente BP-C*.

- *Linear*: La bola debe moverse siguiendo una trayectoria rectilínea.
- *Circular*: La bola debe moverse siguiendo una trayectoria circular.
- *Lissajous*: La bola debe moverse siguiendo una trayectoria que representa una curva de *Lissajous*.

En la parte derecha de la pestaña aparece un panel que a su vez está dividido en dos pestañas (*PD_controllers* y *Trajectories*). A través de la pestaña *PD_controllers* el usuario puede modificar los parámetros (constantes proporcional y derivativa) de los dos controladores *PD*. Mientras que a través de la pestaña *Trajectories* se pueden modificar los parámetros de cada una de las trayectorias (radio del círculo, longitud de la recta, punto de referencia, frecuencias, etc), dependiendo de la trayectoria que se haya seleccionado.

A través de la pestaña *Remote* se conecta con la planta remota. Esta pestaña tiene cuatro botones: *Connect*, *Disconnect*, *Start* y *Stop*. Por medio de los cuales se puede conectar con la planta remota, iniciar la aplicación de control remota, detenerla, y desconectar la planta real. Además por medio de unos selectores (*Center*, *Circle*, *Square* y *Input*) se puede escoger el experimento que se quiere llevar a cabo con la planta

real. Seleccionando *Center* la aplicación remota controlará la posición de la bola en el centro del plato. Esta opción además de ser uno de los experimentos que normalmente se realizan con este sistema, también sirve para establecer la posición inicial de la bola para otros experimentos. Con las opciones *Circle* y *Square* la bola hará un seguimiento de una trayectoria circular y una cuadrada respectivamente. Mientras que con la opción *Input* se pueden introducir las coordenadas de un punto específico sobre el plato donde se desea posicionar la bola. Las coordenadas que se pueden introducir están restringidas a las dimensiones del plato, para evitar ocasionar daños en la planta real. La Figura 3.18 muestra esta pestaña.

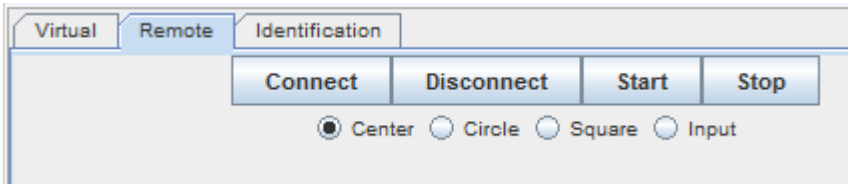


Figura 3.18. Pestaña *Remote* del Panel No. 2 de la aplicación *Cliente BP-C*.

Por su parte la pestaña *Identification* permite realizar el proceso de identificación de la planta real. Por medio del botón *Send File* es posible enviar un fichero en formato **.txt* generado con *MATLAB*. Este fichero debe contener la señal (*PRBS* o pulso) que se desea enviar a la planta para realizar el proceso de identificación. Una vez cargado el fichero por parte del usuario, se comprueba que los valores están en un rango determinado para no causar daños a la planta y se envía este fichero al servidor. Posteriormente hay que presionar el botón *Start Signal* para enviar la señal a la planta. Los datos de la señal de salida se almacenan en un fichero de tipo texto (**.csv*) al que se puede acceder una vez finalizado el envío de la señal.

Para acceder al fichero con los datos el usuario debe conectarse al servidor y descargarlo. Para ello se debe usar cualquier programa gratuito que utilice el protocolo *SFTP*(*Secure File Transfer Protocol*), que es el que por razones de seguridad se usa en la Universidad de Gante. Con el fichero en el ordenador se puede entonces realizar el proceso de identificación de la planta real, diseñar sus propios controladores y probarlos. Teniendo siempre la oportunidad de descargarse el fichero de texto con los datos de entrada y salida de la planta real.

Se debe tener en cuenta que esta planta es un poco complicada de identificar, por lo que los estudiantes deben estar lo suficientemente familiarizados con el proceso de identificación y con la planta para poder obtener buenos resultados.

Panel No. 3

El panel No. 3 denominado Gráficos, posee dos pestañas: *Trajectories* y *Control_Signal*. En la pestaña *Trajectories* se muestran los gráficos de las coordenadas x e y de la posición de la bola en el plato. Mientras que en la pestaña *Control_ Signal* se muestran las señales de control de la planta real.

3.3.3 Aplicación *Servidor BP-C*

La aplicación *Servidor BP-C* fue desarrollada en *Visual C#* en el Departamento de Energía Eléctrica, Sistemas y Automatización de la Universidad de Gante en Bélgica [88]. Esta aplicación es bastante compleja teniendo en cuenta que tiene implementado el procesamiento de imágenes para obtener la posición de la bola sobre el plato; los controladores *PD* que se usan para controlar la planta real; el código referente a la generación de trayectorias para la bola; la comunicación con el circuito que controla los motores a través del puerto USB y el almacenamiento de datos de la planta real. La Figura 3.19 muestra la *GUI* de esta aplicación cuando trabaja en modo local.

La *GUI* muestra la imagen de la cámara USB que se usa para obtener la posición de la bola. El usuario puede hacer clic sobre esta imagen para establecer la referencia en el experimento de control de posición de la bola. Puede seleccionar el tipo de trayectoria que desea que siga la bola (posicionarla en el centro, que siga un círculo o que siga un cuadrado). Además puede modificar algunos parámetros de la calibración en el proceso de detección de la bola. Se puede observar además la coordenada x de la posición de la bola en el plato.

Desde el punto de vista de desarrollo, esta aplicación consiste en una clase principal que tiene implementados varios métodos. El método más importante es un hilo que ejecuta infinitamente un bucle *while* que se encarga de leer las imágenes de la cámara, procesarlas y obtener la posición de la bola. En cada iteración este hilo llama al

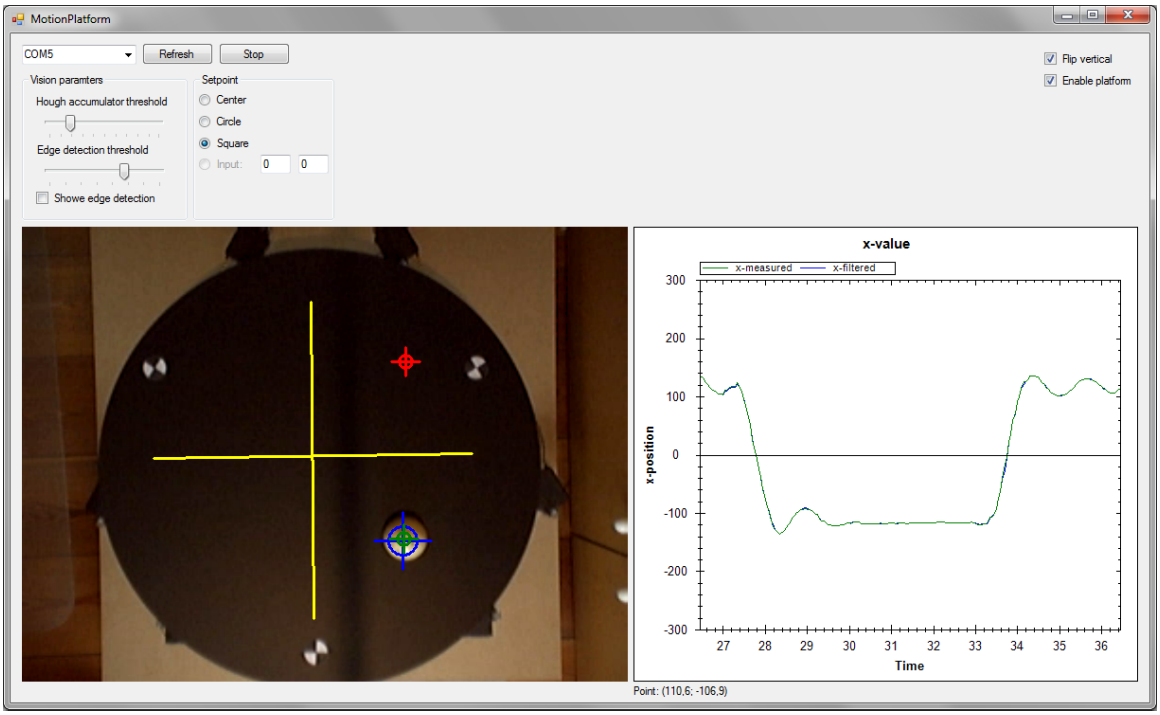


Figura 3.19. *GUI* de la aplicación *Servidor BP-C*.

temporizador *mmTimerTick*. Este temporizador calcula la referencia de posición de la bola y las señales de control (salidas del *PD*). Finalmente envía las posiciones de la plataforma en un rango de $-10°$ a $10°$. Los datos de la posición de la bola, señales de control y tiempo se almacenan en un fichero de texto, tal como se detallará más adelante. Además la posición de la bola se muestra en el gráfico de la *GUI*.

En este caso el desarrollo consiste en modificar esta aplicación para que pueda ser accedida remotamente desde *Internet*. Para que todas sus funcionalidades estén disponibles de forma remota fue necesario desarrollar la parte correspondiente al servidor de la interfaz *JIC*. Esta tarea consiste en la implementación del protocolo *TCP/IP* en el lenguaje *Visual C#*, para que la aplicación se pueda comunicar con *EJS* a través de *Internet*.

En el proceso de comunicación la aplicación de *EJS* actúa como cliente y la de *Visual C#* actúa como servidor. La modificación consiste en agregar cuatro funciones: *startServer*, *proccessEvent*, *readEJS* y *startChirp*. La función *startServer* es la encargada de crear e inicializar el *Socket TCP* que permanece "a la escucha" y atiende las

solicitudes del cliente *TCP* que se implementa en la aplicación *Cliente BP-C*. Esta función se invoca durante la inicialización de la aplicación, por lo que mientras está activa esta aplicación, atiende las solicitudes que se realicen por el puerto *TCP*. En el Anexo A.4 se puede ver el código completo de esta función.

La función *proccessEvent* es la encargada de procesar y dar respuesta a la solicitud recibida desde *EJS*. El Código 3.16 muestra un fragmento de esta función. En el Anexo A.4 se puede ver el código íntegro de esta función.

```
private void ProcessEvents(IAsyncResult asyn){
 Stream s=tcpClient.GetStream(); //Se crea el buffer de entrada
 if(myStream.CanRead){  //Si se lee alguna cadena de datos
   StreamReader readerStream=new StreamReader(myStream);
   string myMessage=readerStream.ReadLine();
   answer=readEJS(myMessage); //Se lee la cadena recibida
   if(tcpClient.Connected){   //Se envía respuesta al cliente
     if(answer==1){ //Puerto seleccionado
        byte[] toSend=Encoding.ASCII.GetBytes("0"+";"+"1"+";"+"\r\n");
        s.Write(toSend,0,toSend.Length);
     }
     else if(answer==2){ //Señal enviada a la planta
        byte[] toSend=Encoding.ASCII.GetBytes("0"+";"+"2"+";"+"\r\n");
        while(!trigger){};
        s.Write(toSend,0,toSend.Length);
      }
      else if(answer==3){ //Error
        byte[] toSend=Encoding.ASCII.GetBytes("0"+";"+"3"+";"+"\r\n");
        s.Write(toSend,0,toSend.Length);
      }
    }...
  }
}
```

Código 3.16. Función *ProccessEvent* incorporada a la aplicación *Servidor BP-C*.

Como se puede apreciar cuando se recibe una solicitud, esta función llama al método *readEJS* pasándole como parámetro la cadena de texto recibida en *myMessage*. La función *readEJS* es la que realmente procesa la cadena recibida y devuelve el resultado del procesamiento en la variable *answer*. En la cadena que se recibe *EJS* indica si quiere recibir respuesta a su solicitud o si solo ha enviado un dato y no necesita respuesta.

Con el valor devuelto por la función *readEJS* se forma una cadena de texto que se envía a *EJS* si éste solicitó respuesta. Los códigos correspondientes a las respuestas del procesamiento son: 1-puerto seleccionado, 2-señal enviada a la planta, 3-error. De esta manera la aplicación cliente puede saber si ha ocurrido un error o si se ha llevado a cabo con éxito su solicitud.

Por su parte la función *readEJS* recibe el paquete de datos y lo analiza. Lo primero que busca es si la cadena recibida es muy grande, lo que indica que se trata de los datos correspondientes a la señal que se debe enviar a la planta para iniciar el proceso de identificación. De ser cierto, crea un fichero de tipo texto y copia los datos en el fichero en la dirección *@"./studentData/Data/y.txt";* como se muestra en el segmento de Código 3.17. En esta misma dirección se almacenarán posteriormente los datos de la planta, por lo que esta carpeta es a la que accede el usuario cuando se conecta por *FTP* para descargar los datos.

```
if(cadena.Length>1000){
 bejs=Encoding.ASCII.GetBytes(cadena); //Obtiene los bytes de la cadena
 Writer=new BinaryWriter(File.OpenWrite(@"./studentData/Data/y.txt"));
 Writer.Write(bejs);  //Copia los datos leídos
 Writer.Flush();      //Limpia el buffer de lectura
 Writer.Close();      //Cierra el Socket
}
```

Código 3.17. Segmento de código de la Función *readEJS*.

Si la cadena es pequeña entonces indica que los datos son una acción a ejecutar. Entre las acciones a ejecutar se encuentran las siguientes:

- Conectar y desconectar la planta.
- Posicionar la bola en el centro del plato.
- Establecer las trayectorias a seguir por la bola (círculo, cuadrado, punto).
- Seleccionar el puerto USB por el que se envían los datos a la planta.
- Iniciar el lazo de control de la planta.
- Detener el control de la planta.
- Guardar los datos en un fichero de datos al terminar un experimento.
- Modificar los parámetros de ambos controladores.

El Código 3.18 muestra como se llevan a cabo algunas de estas acciones, el resto del código se puede consultar en el Anexo A.4.

```
string[] datos=cadena.Split(';'); //Se divide la cadena
if(cadena.Length>3){ //Si cadena es mayor que 3 indica datos recibidos
 if(Convert.ToSByte(datos[1])==1 && !isRunning){ //Botón Start (like btn.click())
  if(Convert.ToSByte(datos[2])==1){ //Case "Center": 021000000
    setpointType=SetpointType.center; //Referencia en el centro del plato
  }
  mmTimer.Start(); //Sustituye al botón Start
 }
 else if(Convert.ToSByte(datos[1])==0 && isRunning==true){
  mmTimer.Stop(); //Para activar el botón Stop 0000000000
  using (CsvWriter writer=new CsvWriter()){ //Guarda los datos del experimento
  writer.WriteCsv(csvFile,@"./studentData/Data/ExperimentData.csv",Encoding.Default);
 }
 }
 else if(Convert.ToSByte(datos[2])==1 && isRunning){
   setpointType=SetpointType.center; //Case "Centro": 021000000
 }
 else if(Convert.ToSByte(datos[2])==2 && isRunning==true){
  setpointType=SetpointType.circle; //Case "Círculo":022000000
 }
 else if(Convert.ToSByte(datos[2])==3 && isRunning==true){
   setpointType=SetpointType.square; //Case "Cuadrado":023000000
 }
 else if(Convert.ToSByte(datos[2])==4 && isRunning==true){
  setpointType=SetpointType.input; //Case "Entrada":024000000
 } // Set the parameters of the PD controllers
 else if(Convert.ToSByte(datos[3])==1 && isRunning==true){
  PID_x.Kp=(float.Parse(datos[4])/10000);    //PID_x.Kp
  PID_x.Kd=(float.Parse(datos[5])/10000);    //PID_x.Kd
 }...
}
```

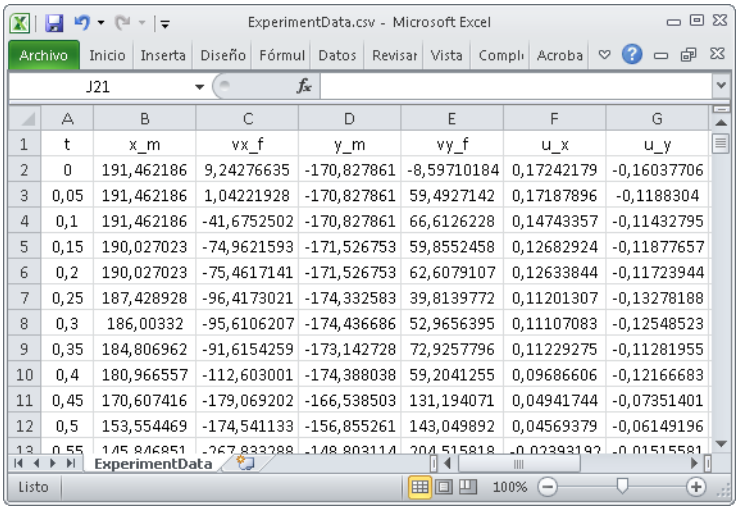
Código 3.18. Segmento de código de la Función *readEJS*.

La Figura 3.20 muestra el aspecto que tiene el fichero *ExperimentData* que contiene los datos almacenados para cada experimento.

Los datos almacenados que corresponden con las columnas del fichero son los siguientes:

- t es el tiempo.

- *x_m* es la coordenada x de la posición de la bola.
- *vx_f* es la coordenada x de la velocidad de la bola.
- *y_m* es la coordenada y de la posición de la bola.
- *vx_f* es la coordenada y de la velocidad de la bola.
- *u_x* es la señal de control para el eje x del plato;
- *u_y* es la señal de control para el eje y del plato;



	A	B	C	D	E	F	G
1	t	x_m	vx_f	y_m	vy_f	u_x	u_y
2	0	191,462186	9,24276635	-170,827861	-8,59710184	0,17242179	-0,16037706
3	0,05	191,462186	1,04221928	-170,827861	59,4927142	0,17187896	-0,1188304
4	0,1	191,462186	-41,6752502	-170,827861	66,6126228	0,14743357	-0,11432795
5	0,15	190,027023	-74,9621593	-171,526753	59,8552458	0,12682924	-0,11877657
6	0,2	190,027023	-75,4617141	-171,526753	62,6079107	0,12633844	-0,11723944
7	0,25	187,428928	-96,4173021	-174,332583	39,8139772	0,11201307	-0,13278188
8	0,3	186,00332	-95,6106207	-174,436686	52,9656395	0,11107083	-0,12548523
9	0,35	184,806962	-91,6154259	-173,142728	72,9257796	0,11229275	-0,11281955
10	0,4	180,966557	-112,603001	-174,388038	59,2041255	0,09686606	-0,12166683
11	0,45	170,607416	-179,069202	-166,538503	131,194071	0,04941744	-0,07351401
12	0,5	153,554469	-174,541133	-156,855261	143,049892	0,04569379	-0,06149196
13	0,55	145,846851	-267,833288	-148,803114	204,515818	-0,02393192	-0,01515581

Figura 3.20. Fichero *ExperimentData* que contiene los datos de un experimento.

La otra función incorporada es *startChirp*. Esta función se encarga de enviar a la planta real la señal utilizada para el proceso de identificación. Como se puede apreciar en el Código 3.19 esta función en primer lugar establece la referencia de posición de la bola en el centro del plato e inicia la función *mmTimer*.

```
private void startChirp(){
 setpointType=SetpointType.center; //Se establece la referencia en el centro
 //Se posiciona la bola en el centro del plato
 mmTimer.Start(); //Se inicia el controlador
 while(count<4000000000){count++;} //Se espera un tiempo
 mmTimer.Stop();  //Se detiene el controlador
 chirpTimer.Start(); //Se envía la señal del usuario a la planta
}
```

Código 3.19. Segmento de código de la Función *startChirp*.

El objetivo es controlar la posición de la bola en el centro del plato para establecer las condiciones iniciales. A continuación espera un tiempo, debido a que esta tarea

consume cierto tiempo hasta que se estabiliza la posición de la bola. El valor de esta demora se estableció por un método de prueba y error comprobando y asegurando que fuera suficiente para posicionar la bola en el centro. Seguidamente se detiene el lazo de control y se envía la señal de identificación a la planta por medio de la función *chirpTimer*. En el Anexo A.4 se puede ver el código completo de la función *startChirp*.

3.3.4 Consideraciones prácticas de la plataforma con el sistema bola y plato (*EJS- Visual C#*)

La plataforma consiste en una simulación interactiva en 3D del sistema bola y plato además de la conexión a través de *Internet* con una planta real de este sistema. La plataforma está basada en una arquitectura cliente - servidor. De la aplicación del lado del cliente se destaca el alto grado de interactividad que tiene, además al ser en 3D, resulta muy realista, por lo que despierta el interés y la atención de los usuarios. Obtener una simulación tan interactiva como ésta resulta muy laborioso, además de que se necesitan conocimientos avanzados de programación para lograrlo. Esto se evita en gran medida si se usa *EJS* para el desarrollo de este tipo de aplicaciones. En cuanto a la aplicación del lado del servidor, se añadieron cuatro funciones a la aplicación de control local que había sido desarrollada en *Visual C#*. Estas funciones se añadieron con el objetivo de poder acceder a todas las funcionalidades de esta aplicación desde *Internet*. Esta aplicación es bastante compleja y extensa, por lo que fue necesario examinar el código detenidamente para poder agregar las funciones mencionadas que permiten manipularla remotamente. La incorporación de estas funciones resultó un trabajo bastante complejo por las peculiaridades que representan la modificación de código de programa desarrollado por otro programador. Afortunadamente el código estaba bien organizado y con comentarios que resultaron de gran utilidad. Otro aspecto a señalar es que se decidió eliminar las gráficas de la *GUI* de la aplicación servidor puesto que la tarea de dibujar gráficos en cada instante de muestreo, consume muchos recursos del ordenador que actúa como servidor. Además, el usuario tendrá disponible todos los datos al finalizar cada experimento con lo que puede reproducir todas las gráficas que desee. También se puede mencionar que resultó muy beneficioso que la aplicación servidor

tuviera implementada la función para enviar una señal personalizada a la planta real, permitiendo utilizar esta funcionalidad en el proceso de identificación de la planta real. La principal modificación en este sentido consistió en facilitar que se pudiera enviar a la planta una señal generada por él mismo, restringiendo solo la amplitud de la misma para evitar daños físicos. También se debe aclarar que no fue necesario interaccionar directamente con el circuito y las funciones que controlan los motores, pues estas funciones están implementadas a alto nivel y se pueden usar perfectamente sin tener que modificarlas.

3.4 Plataforma de robots móviles: *Moway*

En esta sección se describen las características fundamentales de la plataforma para experimentos de control de formación de robots móviles con *Moway*. La Figura 3.21 muestra la estructura de la plataforma que está basada en una arquitectura cliente - servidor como en el caso anterior.

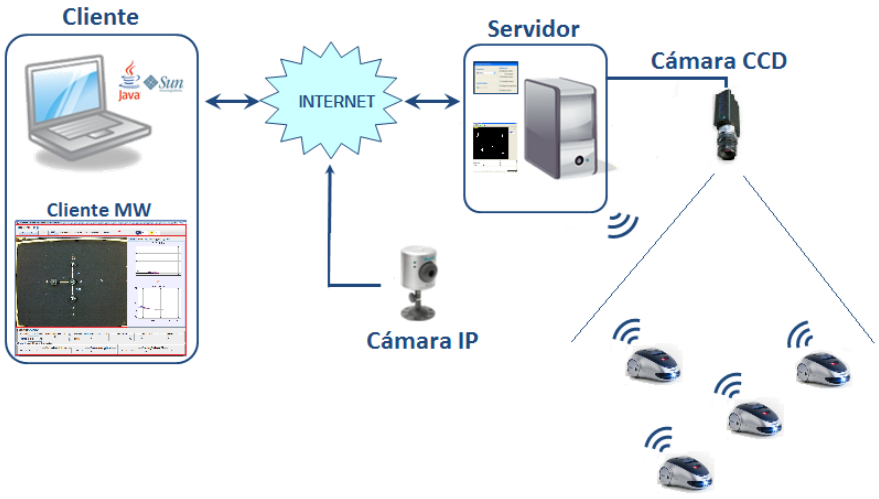


Figura 3.21. Arquitectura de la plataforma para experimentos de robótica móvil.

En este caso, la planta real o proceso lo constituyen un conjunto de robots móviles, que se comunican entre ellos y con el ordenador que actúa como servidor para realizar experimentos de control de formación. La aplicación del lado del cliente es un simulador desarrollado para el control de formación de robots móviles. Mientras que en el lado del servidor se combinan un conjunto de herramientas de *hardware* y *software* que

interaccionan entre sí para llevar a cabo los experimentos. Al igual que en el caso del sistema bola y aro, la plataforma tiene dos modos de trabajo: simulación y remoto. En modo simulación la aplicación trabaja en modo local representando una simulación del proceso de control de formación de robots móviles. Cuando se establece el modo de trabajo remoto, la aplicación cliente se conecta a un servidor para realizar los experimentos con robots reales. El usuario recibe realimentación de lo que está ocurriendo a través de las imágenes que se obtienen de una cámara *web* que muestra el desarrollo de los experimentos. A continuación se describen tanto la aplicación cliente como los componentes del servidor.

3.4.1 Aplicación *Cliente MW*

La aplicación cliente se ha denominado *Cliente MW* y ha sido desarrollada en *EJS* en forma de ejecutable *Java*. No se decidió generar la aplicación como un *Applet Java* para ejecutarla en un sitio *web* porque esto requeriría instalar un servicio *web* en el ordenador que actúa como servidor. Desde un inicio se intentó no sobrecargar de trabajo al servidor ya que además este ordenador tiene que realizar procesamiento de imágenes como se explicará posteriormente. El objetivo fundamental de la aplicación es controlar las posiciones de los robots tanto la del maestro como la de los esclavos para poder realizar formaciones evitando obstáculos.

Como se describió en el Capítulo 2 los algoritmos de evasión de obstáculos necesitan conocer en todo momento la posición de los obstáculos que se encuentran dentro del campo de visión del robot. El objetivo fundamental de conocer la posición de los obstáculos, es construir un histograma polar que indica los sectores circulares que están libres y ocupados en el entorno del robot; para de esta forma poder seleccionar un sector vacío que permita al robot moverse sin colisionar con los obstáculos.

La sección de inicialización, que se muestra en el Código 3.20, primeramente crea el objeto *videocam* y el *Socket TCP* que se usan en la conexión remota. Después calcula la cantidad de sectores circulares alrededor del robot maestro, dependiendo del ancho de cada sector alrededor del mismo. Esto se usa en los algoritmos de evasión de obstáculos. Luego se calculan las componentes x e y de las velocidades de los obstáculos para

cuando éstos sean móviles. Posteriormente se inicializan los sectores circulares alrededor del robot maestro en color verde. De esta forma inicialmente para el robot maestro el histograma polar está completamente en cero al igual que para el resto de los robots (con la función *InitSectors*).

```
//Objeto videocam que obtendrá las imágenes de la cámara
videocam=new webcam.VideoReceiver(otra,"app",isMJPEG,"HTTP",delay);
try {
 sktSwiss=new Socket("62.204.199.192",3000); //Se crea el Socket
 inFromServer=new BufferedReader(new InputStreamReader(sktSwiss.getInputStream()));
} //Se crea el buffer de lectura
catch(Exception e){ //Si se provoca una excepción, se captura
  System.out.print("Error: "+e); //Se muestra el error si lo hay
}
T=360/S; //Se obtiene el número de triángulos (sectores)
initHmTlTh(); //Se inicializa el Histograma Enmascarado
for(int i=0;i<numberOfObstacles;i++){ //Inicializa velocidad de los obstáculos
 velocityObstaclesX[i]=(i+1)*0.1; //Vx de los obstáculos
 velocityObstaclesY[i]=(i+1)*0.1; //Vy de los obstáculos
 flag[i]=false;
}
for(int b=0;b<T;b++){
 sectorcolor[b]=green; //Se dibujan los sectores en verde
 Hbkprev[b]=0.0; //Se inicializa en 0 el Histograma Binario
}
InitSectors(); //Inicializa los sectores
```

Código 3.20. Segmento de código que inicializa la aplicación.

Posteriormente se establecen las posiciones iniciales de los robots dependiendo de la formación seleccionada (*Círculo*, *Línea* o *Libre*), tal como se muestra en el Código 3.21. Si la formación seleccionada es un círculo, dependiendo del número de esclavos se calculan sus posiciones para formar un círculo tomando al maestro como centro del mismo.

Si la formación seleccionada es una línea, los robots esclavos se distribuyen formando una línea tomando al maestro como punto cero de la línea. Mientras que si la formación es libre, se toman las posiciones establecidas por el usuario y se almacenan como puntos finales para cada robot esclavo.

Además de calcular y establecer las posiciones iniciales de los robots, se calculan

```
for(int r=0;r<numberOfRobots;r++){//Posiciones iniciales de los robots
 visiblesetmoway[r]=true;
 if(circle==true){  //Círculo
   positionRobotsX[r]=R*cos(r*2*PI/numberOfRobots)+x1; //X=A=R*cos(r*360/NºAgentes)
   positionRobotsY[r]=R*sin(r*2*PI/numberOfRobots)+y1; //Y=B=R*sin(r*360/NºAgentes)
 }
 else if(row){                    //Línea horizontal (S4)---(S2)---(M)---(S1)---(S3)
   positionRobotsY[r]=y1;                    //Esclavos entorno al maestro
   if(r\%2==0){positionRobotsX[r]=x1+(r+2);}//Pares a la derecha
   else{positionRobotsY[r]=x1-(r+2);}        //Impares a la izquierda
 }
 else{  //Libre
   for(int b=0;b<numberOfRobots;b++){
   if(r<4){ //Posiciones arbitrarias (círculo)
     positionRobotsX[r]=R*cos(r*2*PI/4)+x1; //Tomando con offset la x del maestro
     positionRobotsY[r]=R*sin(r*2*PI/4)+y1; //Tomando con offset la y del maestro
   }
   else{ //Posiciones arbitrarias (círculo de mayor radio)
     positionRobotsX[r]=(R+2)*cos(r*2*PI/4)+x1; //Tomando con offset la x del maestro
     positionRobotsY[r]=(R+2)*sin(r*2*PI/4)+y1; //Tomando con offset la y del maestro
   }
 }
 //Actualiza las distancias y ángulos de refrencia de cada robot en la formación
 dref[r]=dgcalcule(x1,y1,positionRobotsX[r],positionRobotsY[r],0,0);
 angleref[r]=alphacalcule(x1,y1,positionRobotsX[r],positionRobotsY[r],0,0);
}
```

Código 3.21. Segmento de código que establece las posiciones iniciales de los robots.

los ángulos y distancias al punto de destino de cada robot esclavo tomando como referencia al maestro con las funciones (*dgcalculate* y *angleref*). Las ecuaciones que rigen la dinámica tanto para los robots como para los obstáculos se pueden ver en la Figura 3.22 y corresponden a la dinámica descrita en la Sección 2.2.1.1.

Para poder llevar a cabo el control de posición de los robots y realizar formaciones, en cada paso de evolución del programa es necesario calcular la distancia y el ángulo de cada robot a su punto final. Además como los robots se ven como obstáculos entre sí, también es necesario conocer en cada momento la distancia de cada robot al resto de los robots y a cada obstáculo que se encuentre en su radio de visión, tal como se muestra en el Código 3.22. Estos valores se utilizan posteriormente para construir los respectivos histogramas en los algoritmos de evasión de obstáculos descritos en el Capítulo 2.

Para el robot maestro al igual que para los esclavos, todos los robots representan obstáculos. Es por eso que en el algoritmo de evasión de obstáculos se debe incluir tanto la posición de los robots esclavos como la de los obstáculos. Para implementar el algoritmo primeramente se calcula el ángulo del punto de destino ya que también se

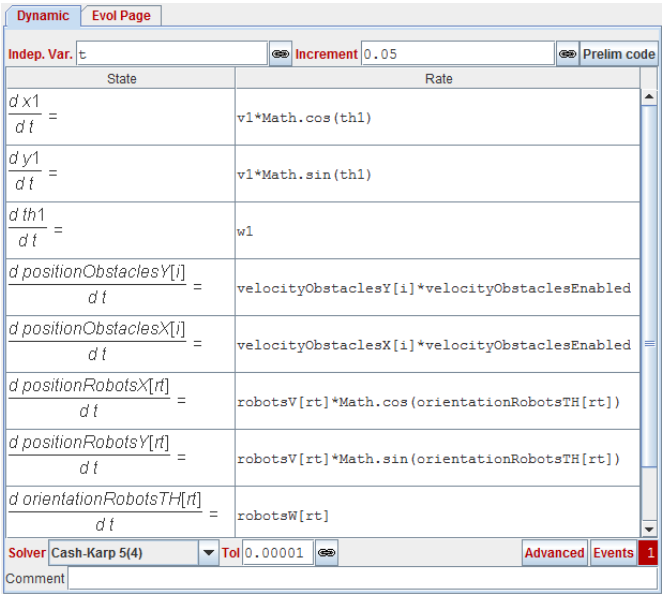


Figura 3.22. Ecuaciones de la dinámica de los robots y los obstáculos.

usará en dicho algoritmo, tal como se muestra en el Código 3.23.

A continuación se calculan las distancias a cada obstáculo y a cada robot que estén dentro del radio de visión ($distanceGlobal[i] < ws$). Dependiendo del algoritmo de evasión de obstáculos seleccionado se calcula el tamaño del sector circular que ocupa cada obstáculo alrededor del robot. Si el algoritmo seleccionado es el *VHF* solo se tiene también en cuenta el radio del obstáculo. Si el algoritmo seleccionado es el *VFH+*, entonces se tiene en cuenta en el cálculo el margen de seguridad del obstáculo. Posteriormente se obtienen el histograma polar primario, el histograma polar primario con histéresis, el histograma polar binario y el histograma enmascarado del robot maestro, tal como se muestra en el Código 3.24.

Posteriormente, dependiendo del resultado obtenido en el histograma se obtiene el sector candidato. El sector candidato es el sector libre lo suficientemente ancho para que quepa el robot y que a la vez está más cerca del sector donde se encuentra el punto de destino, tal como se muestra en el Código 3.25.

Por último se calcula la distancia al punto de destino y dependiendo de la ley de control seleccionada, se obtienen el valor de velocidad lineal y angular del robot. Obteniendo de esta forma la velocidad lineal ($v1$) y la velocidad angular ($w1$) que dictan el movimiento del robot teniendo en cuenta el punto de destino y su entorno en

```
for(int i=0;i<numberOfObstacles;i++){
 //Distancia y ángulo de cada robot a cada obstáculo
 distanceObstacles[i]=dgcalcule(x1,y1,positionObstaclesX[i],positionObstaclesY[i],0,0);
 angleObstacles[i]=alphacalcule(x1,y1,positionObstaclesX[i],positionObstaclesY[i],0,0);
}
for(int rbt=0;rbt<numberOfRobots;rbt++){ //Robots considerados como obstáculos
 orientationRobotsTH_PI4[rbt]=orientationRobotsTH[rbt]-PI/4;
 robotsAlPHA_PI4[rbt]=robotsAlPHA[rbt]-PI/4;
 //Distancia y ángulo de cada robot a cada robot
 distanceRobots[rbt]=dgcalcule(x1,y1,positionRobotsX[rbt],positionRobotsY[rbt],0,0);
 angleRobots[rbt]=alphacalcule(x1,y1,positionRobotsX[rbt],positionRobotsY[rbt],0,0);
}
for(int r=0;r<numberOfRobots+numberOfObstacles;r++){
 //Límites globales derecho e izquierdo para cada robot
 phiRightGlobal[r]=phircalc(th1,angleGlobal[r],posGlobalX[r],posGlobalY[r],x1,y1);
 phiLeftGlobal[r]=philcalc(th1,angleGlobal[r],posGlobalX[r],posGlobalY[r],x1,y1);
}
```

Código 3.22. Segmento de código que calcula parámetros usados en la evasión de obstáculos.

```
alpha1=alphacalcule(x1,y1,xg,yg,0,0);  //Ángulo del maestro al pto. destino
for(int i=0;i<numberOfObstacles+numberOfRobots;i++){
 if(distanceGlobal[i]>ws) continue; //Solo si está en el rengo de visión
 if(Method=="VFH+"){ //Tiene en cuenta el margen de los obstáculos
   gamma=asin(marginObstacles/distanceGlobal[i]); //Sectores que ocupa obstáculo A
 }
 else if(Method=="VFH"){ //NO tiene en cuenta el margen de los obstáculos
   gamma=asin(radioObstacle/distanceGlobal[i]);  //Sectores que ocupa obstáculo A
 }
 else{}   //Tiene que haber un método seleccionado siempre
 resta=0; //Inicialización sectores superiores al ocupado
 suma=0;  //Inicialización sectores inferiores al ocupado
 resta=angleGlobal[i]-gamma;     //Sectores inferiores al ocupado
 suma=angleGlobal[i]+gamma;      //Sectores superiores al ocupado
 if(suma>2*PI) suma=suma-2*PI; //Se establece el rango
 if(resta<0) resta=resta+2*PI;
```

Código 3.23. Segmento de código que obtiene los sectores ocupados por los obstáculos.

cuanto a obstáculos y robots, tal como se muestra en el Código 3.26.

Las leyes de control en el caso del consenso (*controlv1PlusConsensus* y *controlwConsensus*) reciben como parámetros la distancia al punto de destino (*dg1*), la orientación del robot (*th1*) y el ángulo al punto de destino (*alpha1*). Éste último tiene en cuenta el resultado del algoritmo de evasión de obstáculos.

Solo se ha detallado el proceso de evasión de obstáculos y el cálculo de la señal de control para el robot maestro porque para el resto de los robots, el procedimiento es el mismo. El procedimiento solamente se diferencia en los valores del punto final de cada uno, que depende de la posición que deben ocupar respecto al maestro en la formación.

En modo remoto si la aplicación está conectada a la aplicación servidor, en cada

```
if(resta<suma){
   for(int j=(int)(resta/K);j≤(int)(suma/K);j++){
     sectorcolor[j]=red;//Colorea de rojo los sectores inferiores ocupados
     Hpk[j]=(a-(b*pow(distanceGlobal[i],2))); //Histograma Polar Primario
     Hpgk[j]=2*Hpk[j]; //Para dibujar gráfico
   }
}else{
   for(int j=0;j≤(int)(suma/K);j++){
     sectorcolor[j]=red;//Colorea de rojo los sectores superiores ocupados
     Hpk[j]=(a-(b*pow(distanceGlobal[i],2))); //Histograma Polar Primario
     Hpgk[j]=2*Hpk[j]; //Para dibujar gráfico
}
   for(int j=0;j<T;j++){
     if(Hpk[j]≥TauH){ //Límite superior de la histérisis
       Hbk[j]=1.0;      //Estblece valor binario
   }                    //Hp Histograma Polar Primario con histéresis
else if(Hpk[j]≤TauL){
   Hbk[j]=0.0;          //Establece valor binario
}                       //Hb Histograma Polar Binario con histéresis
else{
   Hbk[j]=Hbkprev[j];   //Se obtiene el valor anterior
}
   Hbkprev[j]=Hbk[j];   //Se preserva al valor
}
  for(int j=0;j<T;j++){
   if(Hbk[j]==0){
     double th1aux=th1;
     if(th1aux<0)th1aux=th1aux+2*PI; //Sector candidato libre?
     if(pertenecelibre(K*j,phiRightGlobal[i],th1aux,phiLeftGlobal[i])) Hm[j]=0;
     else Hm[j]=1;     //Hm Histograma enmascarado con histéresis
   } else Hm[j]=1;     //Sector ocupado
  }
}
```

Código 3.24. Segmento de código que construye los histogramas del robot maestro.

```
for(int q=0;q<numberOfObstacles+numberOfRobots;q++){
 if(distanceGlobal[q]<ws){
  widsek=wide_valley(Hm,(int)(alpha1/K)); //Busca el sector candidato
  if(trapped){ //Si está en situación de trampa
   int randomIndex=generator.nextInt(20)-10; //Se genera sector aleatorio
   if((widsek+randomIndex<T)&&(widsek+randomIndex≥0)){widsek=widsek+randomIndex;}
  alpha1=(widsek*K); //Se establece el nuevo valor de alpha/candidato
  if(alpha1<0) break;
}
```

Código 3.25. Segmento de código que encuentra el sector libre más cercano al sector destino.

```
dg1=dgcalcule(x1,y1,xg,yg,0,0);  //Distancia al punto de destino
if(consensus){ //Leyes de control v y w del maestro para el consenso
   v1=controlv1PlusConsensus(dg1,th1,alpha1);//v
   w1=controlwConsensus(dg1,th1,alpha1);//w
}
else{ //Leyes de control v y w del maestro para VFH, VFH+
  v1=controlv1plus(dg1, Wmax, v1min, Hpk[(int)((180*th1/PI)/S)],w1,v1max); //v
  w1=control_w(dg1,th1,alpha1); //w
  if(abs(v1/w1)<minTurningRatio) w1=v1/minTurningRatio; //Limita radio de giro
}
```

Código 3.26. Segmento de código que calcula las velocidades lineal y angular del robot.

paso de evolución se lee el *Socket TCP* y se procesan los datos con las líneas que se muestran en el Código 3.27. En el Anexo A.5 se muestra el código completo de esta aplicación.

```
if(((webcam.VideoReceiver)videocam).isConnected()){ //Si el vídeo está conectado
  getVideo(); //Se obtiene el vídeo
   try {
    String  sentence=inFromServer.readLine();   //Datos de SWISTRACK
    processSwiss(sentence);  //Se procesa el paquete obtenido de  SWISTRACK
   }
   catch(Exception e){ //Si ocurre una excepción, se captura
    System.out.print("No se puede procesar el paquete de SwisTrack "+e);
   }
 }
```

Código 3.27. Segmento de código que obtiene los datos de *SwisTrack*.

La Figura 3.23 representa la ventana principal de la interfaz gráfica de la aplicación *Cliente MW*. Esta ventana principal está dividida en cinco paneles que se explican a continuación:

Panel No. 1

El panel No. 1 contiene un menú que permite guardar y cargar experimentos realizados. Esto proporciona al usuario la posibilidad de guardar configuraciones de experimentos determinados, para poder utilizarlos posteriormente. Este menú también tiene una opción de ayuda que proporciona algunos temas teóricos de control de formación de robots y de evasión de obstáculos, además de alguna ayuda para la utilización de la aplicación. Este panel también contiene los botones de control de la simulación a través de los cuales ésta se puede poner en marcha, detenerla, pausarla o reiniciar todos los valores del experimento. Con los controles de selección contenidos en el panel *Formation*, es posible seleccionar el tipo de formación que desea simular: *Free*, para formaciones libres; *Circular*, para formaciones circulares y *Line*, para formaciones en línea. Si se selecciona *Free* con la simulación en pausa, se puede modificar la formación fácilmente sin más que arrastrar y soltar los robots en las posiciones deseadas. En el resto de las formaciones, las posiciones están impuestas. Es necesario se tenga en

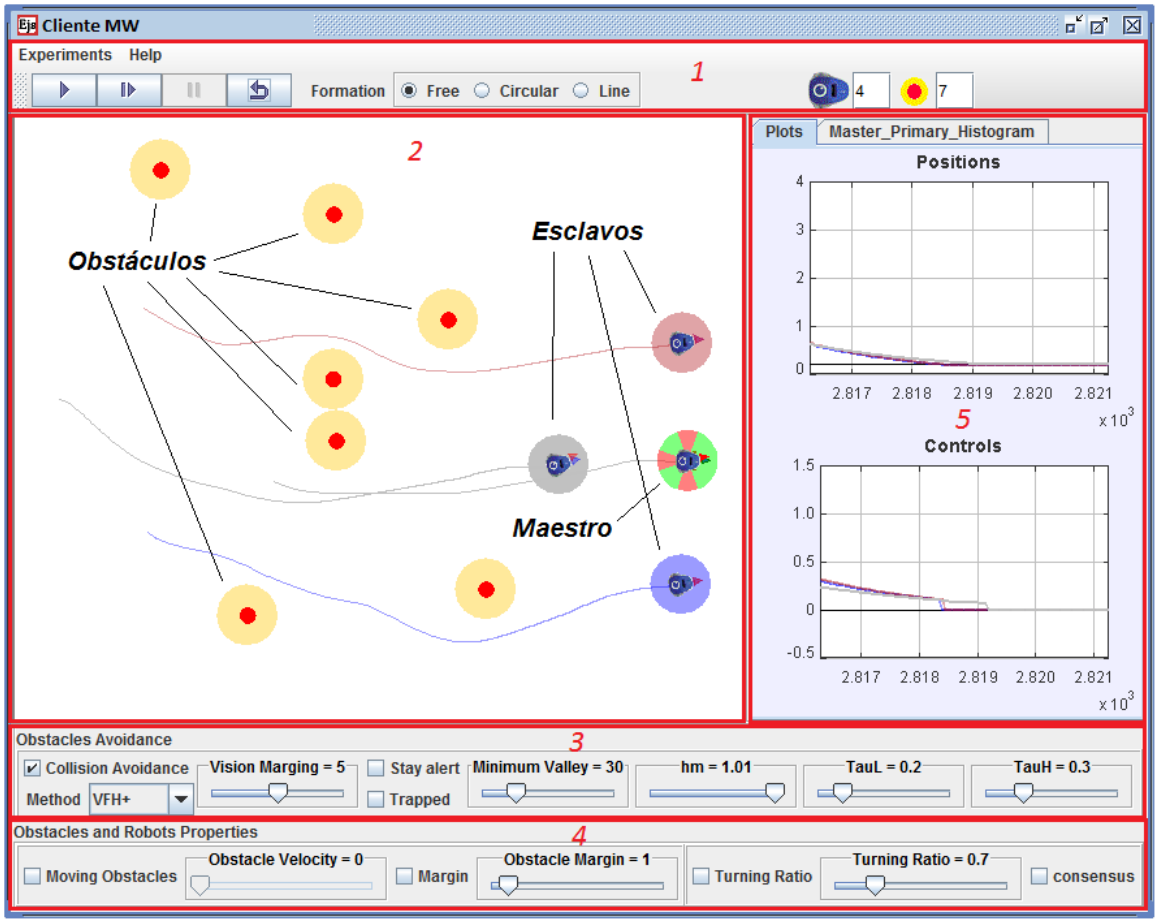


Figura 3.23. Interfaz gráfica de usuario de la aplicación *Cliente MW*.

cuenta que en este caso las formaciones son de tipo maestro-esclavo o de seguimiento a un líder, por lo que las formaciones siempre se realizarán entorno a la posición del maestro o líder. Por ejemplo, en el caso de la formación *Circular* los esclavos formarán un círculo tomando como centro la de la formación posición del maestro y un radio predefinido. En el caso de la Figura 3.23, la formación seleccionada es *Free* en la que los esclavos forman un triángulo alrededor del maestro. Al ser libre la formación a realizar, el usuario debe configurar la formación antes de iniciar el experimento, arrastrando y soltando los robots esclavos en las posiciones deseadas. Al final del experimento los esclavos deberán posicionarse en esta formación entorno al maestro.

En el panel No. 1 también se puede seleccionar la cantidad de robots (incluyendo al maestro) y la cantidad de obstáculos que se quiere que tenga el experimento. Tanto la cantidad de obstáculos como de robots no puede ser menor que uno, por lo que esa es

la mínima cantidad requerida de obstáculos y robots para un experimento. Al agregar obstáculos al experimento, éstos se posicionan aleatoriamente en los espacios vacíos del escenario de experimentación. Para ello se divide el escenario en una matriz donde cada elemento de la matriz es una posible posición para un robot o un obstáculo. En todo momento se conoce el estado de cada posición que puede estar ocupada o libre. Cuando se agrega un robot o un obstáculo, primeramente se obtiene aleatoriamente su posición en el escenario. Una vez obtenida la posición, se verifica si esta está libre u ocupada. Si está ocupada, se obtiene otra posición diferente; si está libre, se le asigna al robot u obstáculo que se agrega. Una vez agregados los obstáculos y robots, el usuario puede modificar las posiciones de los mismos arrastrándolos y soltándolos en la posición deseada.

Panel No. 2

El panel No. 2 muestra el escenario donde se realizan los experimentos. Al hacer clic en el escenario el usuario modifica la posición final que debe ser alcanzada por el robot maestro. Cuando el maestro inicia su movimiento, envía su posición al resto de robots, ya que éstos necesitan la posición del maestro para usarla como referencia, por lo que, también comienzan a moverse para tratar de mantener o alcanzar la formación establecida. Durante el movimiento tanto los esclavos como el maestro deberán sortear los obstáculos que se presenten. Es necesario tener en cuenta que para evitar colisiones entre los robots, en el algoritmo de evasión de obstáculos, se ha establecido que cada robot significa un obstáculo móvil para el resto. Esto le añade un alto grado de complejidad y variantes a los experimentos. Este hecho hay que tenerlo en cuenta a la hora de establecer las distancias entre los robots para las diferentes formaciones. A cada robot se le ha asociado una traza que indica la trayectoria que ha seguido durante el experimento, y una sombra circular a su alrededor, que representa el margen de seguridad definido en el algoritmo de evasión de obstáculos *VFH+*, tal como se presentó en la Sección 2.2.1.3. En el caso del maestro, su margen de seguridad además muestra los sectores libres y ocupados por obstáculos a su alrededor. Estos datos son los que se emplean para calcular el histograma en el algoritmo de evasión de obstáculos. Se

intentó dibujar también los sectores libres y ocupados para el resto de los robots, pero resultó que la simulación se volvía muy lenta al intentar dibujar todos estos objetos al mismo tiempo, por lo que se decidió dejar solamente los sectores dibujados para el maestro y de esta manera también se puede distinguir fácilmente de una manera visual el robot maestro del resto de los robots.

Panel No. 3

El panel No. 3 *Obstacles Avoidance* contiene los datos referentes a los algoritmos de evasión de obstáculos. Si el control de selección *Collision Avoidance* no está seleccionado, los robots no evitarán los obstáculos y colisionarán entre ellos. Esto le permite al usuario establecer comparaciones entre las situaciones en las que se usan los algoritmos y en las que no. Si este control de selección se encuentra marcado, con la lista desplegable *Method* se puede seleccionar uno de los métodos de evasión de obstáculos programados: *VFF*, *VFH* y *VFH+*. Esto permite probar el funcionamiento de los diferentes métodos y establecer comparaciones entre ellos. Con el control deslizante *Vision Margin*, se puede modificar el margen de visión de los robots. Cambiando de esta forma el radio del campo de "visión" de los robots en cuanto a los obstáculos. Mientras más pequeño sea el margen de visión más difícil será evitar las colisiones ya que el robot tiene menos margen de reacción. Si un obstáculo se encuentra dentro de este círculo de visión del robot, entonces se incluye en el algoritmo y por tanto es evitado por el robot. Cuanto antes el robot pueda "detectar" la presencia de un obstáculo, antes podrá reaccionar. Con la opción *Trapped* el robot maestro puede escapar de las situaciones de "trampa" en las que puede caer. Estas situaciones, como se mencionó anteriormete en la Sección 2.2.1.3, son las configuraciones de posiciones de obstáculos en las que el robot puede quedar atrapado y que los algoritmos no pueden resolver. La Figura 3.24 muestra una de estas posibles situaciones.

Como se puede apreciar, entre el robot y su punto de destino aparecen varios obstáculos (Figura 3.24.a). Cuando el robot intenta dirigirse a su punto de destino, se encuentra con los obstáculos (Figura 3.24.b). Debido a la posición de estos obstáculos (formando una especie de semicírculo), cuando el robot se acerca a ellos, todos sus

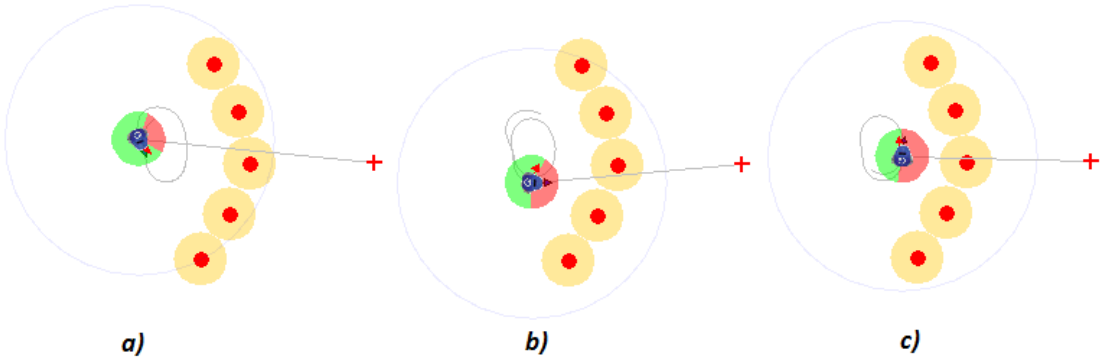


Figura 3.24. Posible situación trampa.

sectores en dirección al destino quedan bloqueados (Figura 3.24.c). Por lo que el robot gira y trata de alejarse de ellos usando los sectores libres más cercanos al sector del punto de destino. Cuando el robot se aleja de los obstáculos, vuelve a "ver" libres los sectores en dirección al destino. Por lo que trata nuevamente de ir en esa dirección y al aproximarse a los obstáculos, el proceso vuelve a repetirse cíclicamente. Esta situación de "trampa" es muy similar a la que ocurre en el método de evasión de obstáculos de los campos potenciales descrito en la Sección 2.2.1.3. En el caso de los campos potenciales lo que ocurre es que puede haber una configuración de obstáculos (muy parecida a esta), en la que el potencial resultante en el robot sea 0. Por lo que el robot puede quedarse detenido sin estar sobre su punto final. Como solución a esta situación de trampa se ha modificado el algoritmo para que el robot pueda escapar, seleccionando aleatoriamente sectores libres que le permitan salir.

El control deslizante *Minimum Valley* especifica el ancho de los sectores libres que requiere el robot para poder pasar. Esto está relacionado con el algoritmo *VFH+* que es el que tiene en cuenta el ancho del robot. La modificación de este valor puede implicar que el robot intente pasar por una abertura por la que no cabe o que exista un espacio suficiente y no se puede percatar de ello. Los valores *hm*, *TauL* y *TauH* están relacionados con el histograma polar primario. Donde *hm* es el coeficiente por el que se multiplica la distancia a un obstáculo para darle valor en el histograma. Mientras que *TauL* y *TauH* son los valores límite que se utilizan para obtener el histograma binario a partir del histograma polar primario. Si el valor de un sector en el histograma primario es mayor que *TauH* este sector tomará el valor 1 en el histograma binario, por el

contrario si es menor que *TauL*, entonces tomará el valor 0. El usuario puede modificar estos valores y observar su influencia en los algoritmos de evasión de obstáculos.

Panel No. 4

El panel No. 4 denominado *Obstacles and Robots Properties* contiene las propiedades físicas de los obstáculos y los robots. El control de selección *Moving Obstacles* define si los obstáculos son estáticos o móviles. Esto también añade complejidad a los experimentos y los hace aún más interesantes. Por medio de un control deslizante, se puede también establecer la velocidad a la que se moverán los obstáculos. El control de selección *Stay Alert* del panel No. 3 está relacionado con los obstáculos móviles. Si este control está seleccionado, cuando un robot está detenido porque ha alcanzado su posición final, el robot evitará el obstáculo móvil. Cuando un robot detecta que un obstáculo móvil se acerca, éste abandona su posición en la formación para evitar la colisión. Una vez que el obstáculo haya sobrepasado la posición del robot, éste vuelve a recuperar su posición en la formación. Si este control no está seleccionado, el obstáculo en movimiento colisionaría con el robot que se encuentra detenido en su posición final. Por medio del control deslizante *Obstacle Margin* se puede modificar el margen de seguridad de los obstáculos, que se tiene en cuenta en el algoritmo *VFH+*. En este panel también se puede especificar el radio de giro mínimo del robot, esta propiedad física es muy importante en la dinámica del robot y en los algoritmos de evasión de obstáculos. Por último en este panel también se puede definir la ley de control que seguirán los robots. Si no está seleccionado, los robots controlarán su posición de acuerdo a la ley de control presentada en la Sección 2.2.1.2. De lo contrario los robots implementarán una ley de control basada en la teoría del consenso [143, 144].

Panel No. 5

El panel No. 5 posee dos pestañas, una denominada *Plots* que es en la que se muestran dos gráficos. El gráfico denominado *Positions* muestra las distancias de los robots respecto a su posición final. Mientras que el gráfico denominado *Controls* muestra las señales de control de todos los robots. Para diferenciarlos, el color de la traza en los

gráficos corresponde con el color del margen de seguridad de cada robot. Por su parte, la pestaña *Master Primary Histogram* muestra los histogramas polar primario, binario y enmascarado para el robot maestro, similar a la Figura 2.31. Al igual que con los margenes de seguridad de los robots, se decidió solo dejar el histograma para el robot maestro puesto que al incorporar los demás histogramas, la aplicación se hacía muy lenta, ya que al ser elementos gráficos consumen muchos recursos.

Cuando la aplicación *Cliente MW* trabaja en modo remoto se conecta a un ordenador que hace de servidor y del cuál a continuación se describirán sus componentes.

3.4.2 Aplicación *Servidor MW*

Para realizar experimentos de control de formación de robots móviles en modo remoto es necesario conocer la posición absoluta o relativa de los robots en cada instante. En el caso de la simulación esto no constituye ningún problema, puesto que en todo momento las posiciones de los robots se conocen. Pero en el caso del escenario de experimentación con robots reales las cosas son muy diferentes. La obtención de las posiciones de objetos móviles siempre ha constituido un reto en la realización de experimentos con equipamiento real. Para ello se utilizan varias técnicas como el uso de *GPS* o señales de ultrasonido [145, 146]. En este caso el *GPS* no resulta adecuado porque se trabaja en un entorno bajo techo. Además, las mediciones de posición con *GPS*, suelen tener errores que en magnitud resultan inapropiados para esta plataforma. En el caso de la determinación de la posición por ultrasonidos, no se disponía de los componentes necesarios para su implementación. Otra opción que es bastante común, es el uso de una cámara de alta resolución e indicadores o códigos en la parte superior de los objetos que permitan identificarlos por medio del procesamiento de las imágenes de dicha cámara [44, 147–150].

Arquitectura

En este caso, se utilizó una cámara *CCD* en color de alta resolución de *Basler A631fc* [147, 151]. Aunque se trata de una cámara en color, se trabaja con ella en modo monocromo para aprovechar al máximo la resolución espacial y la velocidad de

disparo. En estas condiciones se obtienen imágenes con una resolución de *1388* x *1038* píxeles a una velocidad de *18.7 fps*. La cámara se encuentra montada en un soporte anclado al techo del laboratorio de tal modo que se realiza la observación del suelo desde una posición cenital. La cámara se conecta con el ordenador mediante un cable de enlace *FireWire* (*IEEE 1394a*) de *10m* de longitud. Se acondicionó una mesa de *2.50 m* x *1.70 m* con sus respectivos bordes para que los robots no cayeran al suelo. Con el área que ofrece la mesa, es suficiente para desarrollar los experimentos debido al tamaño pequeño y a la manejabilidad de los robots y al área de visión de la cámara. La Figura 3.25 muestra las interacciones entre los componentes de la arquitectura. Las líneas punteadas representan conexiones inalámbricas.

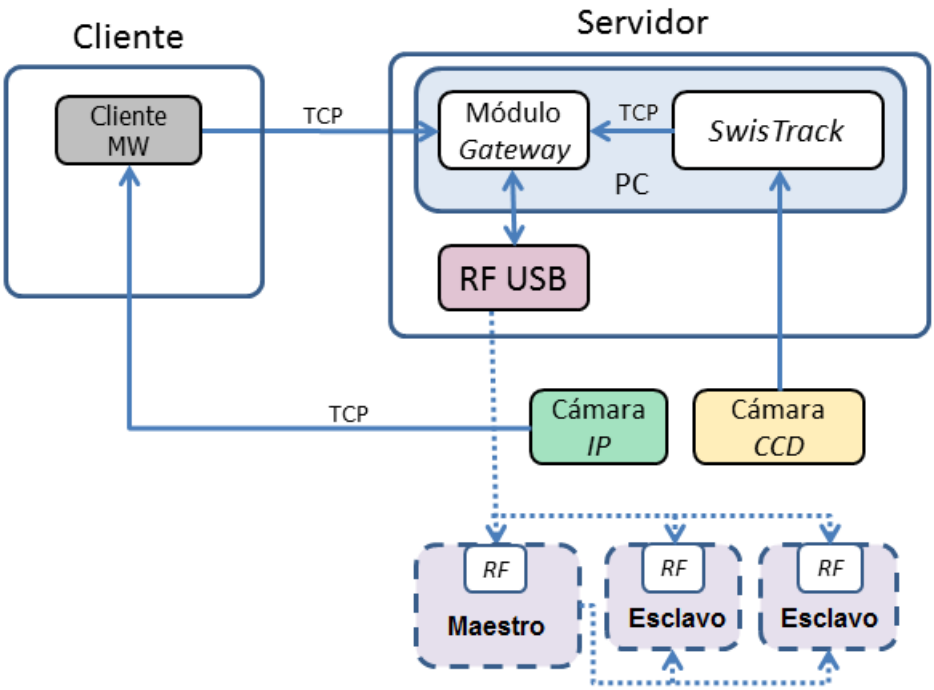


Figura 3.25. Interacción de los componentes que forman la plataforma.

Los robots reciben sus posiciones del ordenador y la usan para controlar su propia posición. En el caso del robot que actúa como maestro, toma la referencia también del ordenador y calcula su ley de control. Mientras que envía a los esclavos su posición, pues éstos la necesitan como referencia para calcular su ley de control. Al mismo tiempo, en el servidor se encuentra ejecutándose la aplicación *Cliente MW* en modo remoto. Esta aplicación se comunica con el *Módulo Gateway* cuando el usuario quiere realizar

alguna modificación en el experimento. Como por ejemplo, cambiar la referencia del robot que actúa como maestro, cambiar el tipo de formación, modificar el tipo de control o establecer una nueva configuración inicial.

La aplicación *Cliente MW* también recibe la señal de vídeo de la *cámara IP* a través del protocolo *TCP*. Por medio de esta señal se puede recibir realimentación del experimento que se está llevando a cabo.

SwisTrack

Las imágenes de la cámara *CCD* se procesan por una herramienta de código abierto desarrollada en la Escuela Politécnica Federal de Lausanne (*EPFL*) denominada *SwisTrack* [152, 153]. Para conocer las posiciones de los robots a cada uno se le coloca un identificador en su parte superior. Este identificador permite conocer la posición y orientación de cada uno de los robots con una buena precisión. Los identificadores pueden ser de diferentes tamaño y se pueden generar tantos como se desee, solo hay que cumplir con algunas restricciones que presentan los autores. Si se siguen sus indicaciones, se obtienen muy buenos resultados. Un ejemplo de los códigos que se pueden obtener se muestran en la Figura 3.26.

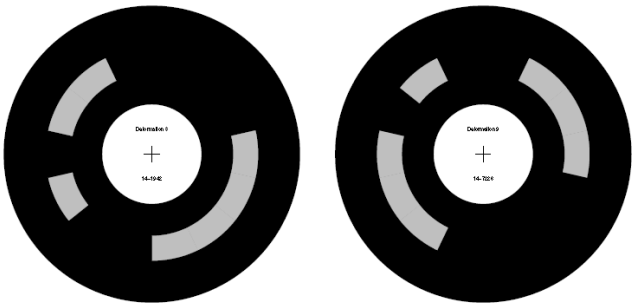


Figura 3.26. Códigos utilizados por *SwisTrack* para identificar los robots.

En este caso, para seguir las indicaciones de los autores, fue necesario construir un sistema de iluminación de forma tal que la luz que incidiera sobre la mesa fuera intensa pero a la vez difuminada. Al instalar el sistema de iluminación, se produjeron sombras en la mesa, que dificultaban el proceso de identificación de los códigos por parte del *SwisTrack*. Por lo que fue necesario entonces colocar un tapiz de color gris

para que desaparecieran las sombras. Esto dio muy buenos resultados. En un inicio se pensó que esto podría repercutir negativamente en el desplazamiento de los robots, pero el resultado es que el tapiz garantiza que los robots no se deslicen, por lo que resulta también muy beneficioso.

Para obtener las posiciones de los marcadores el *SwisTrack* realiza un procedimiento de procesamiento de imágenes que consta de varios pasos. Cada paso puede realizarse por uno o más componentes del programa. La Tabla 3.1 muestra los diferentes pasos (componentes) que se llevan a cabo, así como sus funcionalidades.

No.	Componente	Funcionalidad
1	*"Trigger"*	Indica cuando procesar la próxima imagen.
2	*"Input"*	Adquisición de imagen de la cámara (por *"FireWire"*).
3	*"Background"*	Sustracción del fondo.
4	*"Thresholding"*	Conversión de la imagen a binario.
5	*"Particle detection"*	Detección de manchas en la imagen binaria.
6	*"Calibration"*	Transformación de coordenadas (píxeles a cm).
7	*"Output"*	Salida por un puerto TCP (mensajes NMEA 0183).

Tabla 3.1. Pasos que realiza *SwisTrack* en el procesamiento de imágenes.

El primer componente (*"Trigger"*) indica el momento en que se debe obtener la siguiente imagen, en este caso a través de un temporizador. Este componente establece la frecuencia de obtención de las imágenes para su procesamiento. El segundo componente *"Input"* adquiere la imagen directamente de la cámara, en este caso la cámara se encuentra conectada por el puerto *"FireWire"*. La imagen obtenida es en blanco y negro porque a pesar de que la cámara es en colores, se usa en modo monocromo para eliminar un paso del procesamiento. El componente *"Background"* elimina el fondo de la imagen, que ha sido definido previamente. El componente *"Thresholding"* convierte la imagen a binario usando un umbral que puede ser variado manualmente, dejando solo colores blanco y negro. Esto permite detectar los marcadores con mayor facilidad. El componente *"Particle detection"* detecta el centro de los marcadores dependiendo del radio de su círculo central y los identifica con sus respectivos códigos. El componente *"Calibration"* convierte las coordenadas de píxeles de la imagen a centímetros del mundo real usando para ello un modelo lineal de segundo orden. Finalmente el

componente *"Output"* crea un paquete con las posiciones de los marcadores detectados usando formato de mensajes *NMEA 0183* y lo coloca en un puerto *TCP* definido previamente. Al usar el protocolo *TCP/IP* dichos paquetes pueden ser leídos desde *Internet*. La Figura 3.27 muestra los resultados de aplicar los pasos del 3 al 6.

Figura 3.27. Etapas de procesamiento de un código de identificación.

Una de las mayores ventajas de esta aplicación es que la obtención de los datos se hace muy rápidamente, teniendo en cuenta que procesa las imágenes y devuelve los datos de todos los marcadores presentes en el escenario. Anteriormente se intentó hacer esto mismo usando *LabVIEW* y los resultados obtenidos no fueron buenos. Lo más importante a tener en cuenta para el uso de esta aplicación es la calidad de la cámara (resolución, *frames* por segundo y disparo externo); puesto que de estos parámetros depende la rapidez del procesamiento de las imágenes.

El paquete de datos recibido en formato *NMEA 0183* se muestra en la Figura 3.28. Analizando este mensaje se puede observar que están presentes 4 robots, para los cuales se indican sus posiciones y sus orientaciones.

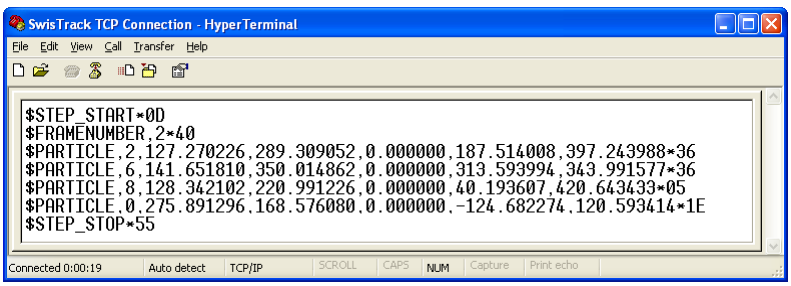


Figura 3.28. Paquete de datos que contiene la posición de los robots.

Módulo Gateway

Para procesar el paquete de datos y enviar las posiciones a los robots, se desarrolló una aplicación en *Visual C#*, que se denominó *Módulo Gateway*. Esta aplicación es la encargada de leer el paquete de datos del puerto *TCP* enviado por *SwisTrack*, procesarlo y enviar las respectivas posiciones a los robots. Además recibe las solicitudes de *EJS* de envío de la referencia para el robot maestro, para lo que usa la aplicación *JIC*, que se describió en la Sección 3.3.2. Estas funcionalidades las llevan a cabo un hilo y un temporizador. La Figura 3.29 muestra el diagrama de flujo de la aplicación *Módulo Gateway*.

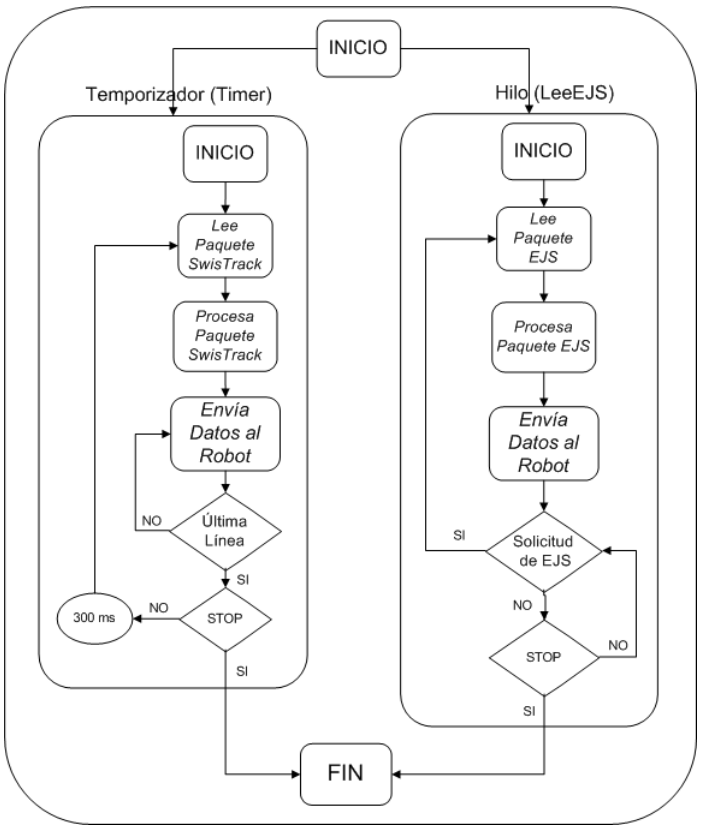


Figura 3.29. Diagrama de flujo de la aplicación *Módulo Gateway*.

Durante la inicialización de la aplicación se crean varios objetos e inician varios objetos que luego serán usados, como por ejemplo: el "*Socket TCP*" por el cuál se comunica ésta aplicación con *SwisTrack* y el "*Socket TCP*" por el que se comunica con *EJS* usando *JIC*, como se muestra en el Código 3.28.

```
//Cliente SWISTRACK
clientSocket=new System.Net.Sockets.TcpClient(); //Inicializa Socket TCP SwisTrack
clientSocket.Connect("127.0.0.1",3000);          //Socket por el puerto 3000
if(clientSocket.Connected){label8.Text="TCP Connected";}
else{label8.Text="TCP DisConnected"; }
//Servidor EJS
IPAddress ipAddress=IPAddress.Any;
TcpListener listener=new TcpListener(ipAddress,1238); //Inicializa Socket TCP de EJS
listener.Start(); //Se inicia la escucha por el puerto 1238
socketejs=listener.AcceptSocket(); //Solicitud de un cliente?, se acepta
```

Código 3.28. Segmento de código que crea los *Sockets* para la comunicación con *SwisTrak* y *EJS*.

```
myTimer=new System.Windows.Forms.Timer(); //Timer para SWISTRACK
myTimer.Tick += new EventHandler(TimerEventProcessor);
ejshilo=new Thread(new ThreadStart(leeEJS));  //Hilo para lectura de datos EJS
ejshilo.Start();       //Se inicia el hilo EJS
myTimer.Interval=300; //Intervalo del Timer en milisegundos
myTimer.Start();       //Se inicia el Timer SWISTRACK
```

Código 3.29. Temporizador que establece la frecuencia de muestreo y proceso que recibe de *EJS*.

Como se puede apreciar en el caso de la comunicación con *SwisTrack* el *Socket TCP* se crea como cliente. Puesto que en este caso el *SwisTrack* es el que actúa como servidor colocando los paquetes de datos en el puerto 3000. Al estar las dos aplicaciones ejecutándose en el mismo ordenador, la dirección *IP* que se usa para la comunicación es la *127.0.0.1* que indica que la conexión es local. Mientras que para la comunicación con *EJS* esta aplicación actúa como servidor y *EJS* actúa como cliente. Por lo que se crea el *Socket* por el puerto *1238* y se comienzan a escuchar las peticiones que se realicen por ese puerto. En este caso como *EJS* se comunicará con esta aplicación a través de *Internet*, se usa la dirección *IP* del servidor para crear el *Socket*. A continuación se crea y se inicializa el temporizador que establece el tiempo de muestreo y el proceso que recibe las solicitudes recibidas desde *EJS*, tal como se muestra en el Código 3.29.

Como se puede apreciar primeramente se crea el temporizador *myTimer* y se le asigna la función que se llama cada vez que tenga que ejecutarse. En este caso, se establece que el temporizador debe ejecutarse cada *300 ms*. Cada vez que transcurre este tiempo, la función *TimerEventProcessor* lee el paquete de datos de *SwisTrack*, lo procesa y envía a cada robot su posición y orientación. Durante la inicialización también se crea el hilo (*leeEJS*) que atiende las solicitudes realizadas desde *EJS*. Este

proceso tiene la particularidad que se ejecuta en paralelo con el temporizador.

Cuando desde la aplicación *Cliente MW* (*EJS*) el usuario interacciona con los robots de forma remota, estas acciones se convierten en solicitudes que atiende la aplicación *Módulo Gateway*. Entre las más importantes se encuentra enviar al robot maestro la referencia o punto final al que debe dirigirse tal como se muestra en el Código 3.30.

```
while(true){   //Leer desde EJS
  int k=socketejs.Receive(bejs); //Se leen los datos del Socket EJS
  cadena=ascii.GetString(bejs);  //Se obtiene la cadena de EJS
  string[] datos=cadena.Split(';'); //Se divide la cadena
  if(k>6){ //Construyendo paquete de datos
    Data[0]=CMD_SEND_RF; //Instrucción para envío de paquetes
    Data[1]=Convert.ToSByte(datos[2]); //Robot destino
    Data[2]=Convert.ToSByte(datos[0]); //Xp
    Data[3]=Convert.ToSByte(datos[1]); //Yp
    Data[4]=0;                         //Xref
    Data[5]=0;                         //Yref
    Data[6]=Convert.ToByte(datos[3]);  //1-form, 2-ref, 3-ctrl
    Data[9]=255; //Id paquete
    bzirf.Send(Data); //Se envía el paquete de datos
  }
}
```

Código 3.30. Función que atiende las solicitudes de *Cliente MW*.

Como se puede apreciar, esta función consiste en un bucle *While* que se ejecuta en un ciclo sin fin una vez iniciado el proceso de inicialización. Si se recibe alguna solicitud y la cadena recibida tiene más de 6 datos, ésta se procesa y se crea el paquete a enviar al robot destino. El resto de las funcionalidades de este proceso se pueden ver en el Anexo A.6. Por su parte, el temporizador *TimerEventProcessor* que se ejecuta cada 300 *ms* también realiza varias acciones. La más destacada es la de obtener el paquete de *SwisTrack* procesarlo y enviar las posiciones a los robots. El fragmento que se muestra en el Código 3.31 indica cómo se realiza esta tarea.

Una vez recibido el paquete, éste se divide en líneas y se busca en cada línea la cadena de caracteres *PARTICLE* que es la que indica el comienzo de los datos. En cada línea los datos están separados por coma según el formato *NMEA 0183*, por lo que se separan y se obtienen las posiciones. Primeramente se obtiene el número que

```
if(lines[i].StartsWith("PARTICLE,")){ //Si la línea empieza con PARTICLE
 string[] datos=lines[i].Split(','); //Divide la línea y guarda los string en data
 byte robot=Convert.ToByte(datos[1]); //Obtiene el número del robot
 if(datos[5].IndexOf('.')>0){
   x_sh=Convert.ToInt16(datos[5].Remove(datos[5].IndexOf('.'))); //Obtiene x
 }
 else{
   x_sh=Convert.ToInt16(datos[5]); //Convierte x a Short
 }
 if(datos[6].IndexOf('.')>0){
   y_sh=Convert.ToInt16(datos[6].Remove(datos[6].IndexOf('.'))); //Obtiene y
 }
 else{
   y_sh=Convert.ToInt16(datos[6].Remove(datos[6].IndexOf('*'))); //Convierte y a Short
 }
 double th_dou=Convert.ToDouble(datos[4].Replace(".", ",")); //Obtiene th
 th_dou=RadianToDegree(th_dou);          //Convierte th a grados
 th_sh=Convert.ToInt16(th_dou);          //Convierte th en grados a Short
 robotArray[numberOfRobots,0]=robot;     //Actualiza el robot en cuestión
 robotArray[numberOfRobots,1]=x_sh;      //Actualiza la coordenada x
 robotArray[numberOfRobots,2]=y_sh;      //Actualiza la coordenada y
 robotArray[numberOfRobots,3]=th_sh;     //Actualiza la coordenada th
 robotArray[numberOfRobots,6]=numberOfRobots;  //Índice del Robot
 if(robotArray[numberOfRobots,3]≥360){
  robotArray[numberOfRobots,3]=(short)(robotArray[numberOfRobots,3]-(short)360);
  }
 numberOfRobots++;
}
```

Código 3.31. Función que lee los paquetes de *SwisTrack*.

identifica al código que por lo tanto coincide con la dirección del robot al que se enviarán los datos. Posteriormente se obtienen las coordenadas x e y que indican la posición del robot, así como su orientación (th). Seguidamente se construye el paquete de datos y se le envía al robot. Estos pasos se siguen para cada línea del paquete que comience con la cadena de caracteres *PARTICLE*, es decir, para cada código detectado, o lo que es lo mismo, para cada robot. Por otra parte, cabe destacar que, los robots no pueden saber la posición de los obstáculos en su entorno puesto que solo tienen un sensor de distancia en frente, por lo que si tienen un obstáculo a un costado o detrás, no pueden detectarlo. Ante esta imposibilidad, se decidió implementar el algoritmo de evasión de obstáculos en el *Módulo Gateway*. Esta aplicación en todo momento maneja la información de las posiciones de todos los robots, por lo que puede calcular perfectamente para cada uno de ellos los sectores libres y enviar esta información a los robots. Esto centraliza un poco el proceso, cosa que siempre se quiso evitar, pero en este caso no hubo alternativa por las restricciones intrínsecas de los propios robots. Inicialmente se pensó enviar a cada robot las posiciones del resto de ellos, pero esto en la práctica no funcionó muy

bien, debido a los retardos introducidos en la red por el incremento del tráfico de información. En el Anexo A.6 se puede ver el código completo de esta aplicación donde se implementan el resto de las funcionalidades.

Programación de los robots Moway

La programación de los robots *Moway* se puede hacer de varias formas incluyendo el lenguaje Ensamblador. Pero en este caso se realiza en lenguaje C usando *MPLAB* con un compilador de lenguaje C. Al implementarse un control de posición tanto para los robots esclavos como para el robot maestro, las leyes de control son muy similares. Las mayores diferencias se encuentran en el proceso de comunicación y obtención de las respectivas referencias como se ha explicado anteriormente. A continuación se describen los principales detalles de estas aplicaciones.

Como se mencionó con anterioridad el robot maestro recibe su posición, obtenida en el servidor por el *SwisTrack* y enviada por el *Módulo Gateway*, mediante el módulo *Moway RF USB* conectado al servidor. El programa que se ejecuta en cada uno de los robots es un bucle infinito que chequea periódicamente si se ha recibido algún paquete de datos. Este paquete de datos dependiendo de donde proceda contiene diferente tipo de información, tal como se muestra en el Código 3.32.

```
if(ret!=2 && data_in_dir==8){  //Recibe de la cámara 8
 LED_TOP_RED_ON_OFF();         //LED Rojo (indica recepción de la cámara)
 x=((float)(data_in[0]))-84.0; //x + 84 x en un solo byte desde la cámara
 y=((float)(data_in[1]))-64.0; //y + 64 x en un solo byte desde la cámara
 thbytes[0]=data_in[2];        //Dos bytes correspondientes al theta del robot
 thbytes[1]=data_in[3];
 alphabytes[0]=data_in[4];     //Dos bytes del ángulo al target
 alphabytes[1]=data_in[5];
 hc=(float)(data_in[6]/10.0);  //Valor del histograma en el sector del ángulo th
```

Código 3.32. Función que lee los paquetes enviados por el Módulo *Gateway*.

Al recibir un paquete de datos, si éste procede de la cámara, entonces su contenido es la posición y la orientación del robot. En este caso se guardan estos valores en las variables x e y que almacenan la posición actual del robot así como su orientación

en la variable *th*. Además se almacenan también los valores de *alpha* y *hc* que serán utilizados para calcular la ley de control teniendo en cuenta el algoritmo de evasión de obstáculos. A continuación el robot maestro envía su posición al resto de los robots, tal como se muestra en el Código 3.33.

```
data_out[0]=data_in[0];  //x+84 del robot maestro
data_out[1]=data_in[1];  //y+64 del robot maestro
data_out[2]=0;           //No usados
data_out[3]=0;           //No usados
data_out[4]=0;           //No usados
data_out[5]=0;           //No usados
data_out[6]=0;           //No usados
data_out[7]=123;         //Etiqueta del paquete
var=RF_SEND(1,data_out);   //Envío al robot 1
var=RF_SEND(4,data_out);   //Envío al robot 4
var=RF_SEND(3,data_out);   //Envío al robot 3
```

Código 3.33. Segmento de código que envía la posición del maestro al resto de robots.

Los robots esclavos reciben la posición del maestro después de haber recibido su propia posición desde la cámara. Con ambas posiciones (la suya y la del maestro) calculan entonces su ley de control. Si el paquete recibido por el robot maestro no es de la cámara, entonces es la referencia (su posición final deseada). Con este paquete de datos el robot maestro recibe además el *offset*, tal como se muestra en el Código 3.34. Estos valores indican la distancia en x e y que debe mantener el robot con respecto al punto final, dependiendo de la formación. Para el maestro estos valores son 0 porque la formación la realizan los esclavos entorno a él.

```
LED_TOP_GREEN_ON_OFF(); //LED verde indica recepción del maestro
xp=((float)(data_in[0]))-84.0; //x referencia
yp=((float)(data_in[1]))-64.0; //y referencia
xrefbyte=data_in[2];   //x offset
yrefbyte=data_in[3];   //y offset
contOrConf=data_in[4]; //Configuración o formación?
```

Código 3.34. Segmento de código que obtiene la referencia del maestro.

Con los datos de su posición actual y el punto final al que debe dirigirse, el robot calcula entonces las señales de control. Estas señales de control son valores de velocidades

```
d=sqrt(pow(yp-(y-ymref),2)+pow(xp-(x-xmref),2)); //Calcula la distancia al pto final
error=alpha-th;      //Calcula el error
omega=Kp*(error);    //Calcula la w
if(hc>hm) hcpp=hm;   //Mínimo entre hc y hm es hcpp
if(hm>hc) hcpp=hc;
Vp=Vmax*(1-(hcpp/hm));     //Calcula la velocidad a partir de hcpp y hm
if(Vp*(1-(omega/omega_max))+9.2≥Vmax){Vcm=Vmax;} //Vcm en función de hcpp y Wmax
else{Vcm=Vp*(1-(omega/omega_max))+9.2;}  //Escalado de velocidad en cm
Vicm=(2.0*Vcm+omega*6.5)/2.0;          //Vcm izquierdo y derecho
Vdcm=(2.0*Vcm-omega*6.5)/2.0;
if(Vicm≥9.2) Vi=(Vicm-9.2)/0.0788;  //Vi=(Vicm-9.2)/0.0788; Vi de Cm a Volt
else if(Vicm≥4.5) Vi=1;              //Límite zona muerta motor
else Vi=0;
if(Vdcm≥9.2) Vd=(Vdcm-9.2)/0.0788;  //Vd=(Vdcm-9.2)/0.0788; Vd de Cm a Volt
else if(Vdcm≥4.5)Vd=1;               //Límite zona muerta motor
else Vd=0;
if(d>5){ //Si la distancia al puto es >5cm se envían las v a los motores
 MOT_CHA_VEL((unsigned char)Vi,fwdbacki,LEFT,TIME,0);  //Motor izquierdo
 MOT_CHA_VEL((unsigned char)Vd,fwdbackd,RIGHT,TIME,0); //Motor derecho
}
else{MOT_STOP();}
```

Código 3.35. Segmento de código que calcula las señales de control.

lineal y angular, tal como se muestra en el Código 3.35.

Como se puede apreciar, para el robot maestro primeramente se calcula la distancia y el error de ángulo con respecto al punto de destino (que depende de la orientación actual y la deseada). Con estos valores y teniendo en cuenta el algoritmo de evasión de obstáculos, se obtienen la velocidad angular (*omega*) y la velocidad lineal (*Vp*). La velocidad lineal calculada se transforma a velocidad del mundo real (*Vcm* en cm/s) para adaptarla a las condiciones reales del robot (zona muerta, discontinuidad, rozamientos, etc). Esta velocidad lineal se transforma en velocidades lineales para los motores derecho e izquierdo, que finalmente se envían como órdenes a dichos motores (con la instrucciones *MOT_CHA_VEL*) si la distancia al punto es mayor que *5 cm*. De lo contrario, se detienen los motores (*MOT_STOP*) porque el robot ya está en un entorno muy cercano al punto de destino.

En el caso de los robots esclavos el programa que se ejecuta es bastante similar con algunas diferencias. Por ejemplo cuando los esclavos reciben su posición no tienen que enviarla a otro robot. Además obtienen su referencia para el control en el paquete recibido del maestro. Esto añade una complejidad extra porque cuando el servidor envía las posiciones a los robots, en el caso de los esclavos, éstos tienen que recibir inmediatamente la posición enviada desde el maestro. Debido a que necesitan ambos

datos para calcular sus respectivas leyes de control, los retardos en las comunicaciones pueden influir negativamente y con gran peso en el control de formación. En el Anexo A.7 se puede ver el código completo de los programas que se ejecutan tanto para el robot maestro como para uno de los robots esclavos.

Ejemplo de control de formación

La Figura 3.30 muestra los resultados del experimento representado en la Figura 3.23 en modo simulación, pero con los robots reales. Se muestra al maestro con la letra M y a los esclavos con la letra S y un número para identificarlos (S_1, S_2, S_3). Como se puede apreciar, inicialmente los robots se encuentran en una formación *Free* seleccionada por el usuario. Al hacer clic en el escenario, se le modifica la referencia del maestro por lo que éste comienza a desplazarse. Transcurridos 3 segundos, la formación se ha perdido, debido a los retardos de la red y a que los esclavos tienen que esperar a que el maestro reciba y envíe. Pasados 6 segundos el maestro ha llegado a su posición final y los esclavos también. Lográndose alcanzar la formación que tenían al inicio del experimento pero en una nueva posición.

3.4.3 Consideraciones prácticas de la plataforma con robots *Moway*

Tanto la simulación como la experimentación con robots móviles resulta de gran interés para los estudiantes. Al interaccionar con estos tipos de plataformas se sienten muy motivados debido a que se enfrentan a problemas novedosos y esto despierta su curiosidad por aprender. Desde este punto de vista la plataforma resulta de gran utilidad en el proceso de aprendizaje. Además, tanto el control de formación, como la evasión de obstáculos constituyen materias con las que se pueden demostrar aspectos teóricos de ingeniería de control. En cuanto a la implementación de la plataforma, se puede decir que al igual que en los casos anteriores, la versatilidad de *EJS* resultó de gran ayuda en el desarrollo de la aplicación cliente para el trabajo en modo simulación. En los inicios del desarrollo se pensó que habría problemas con la implementación del sistema para obtener las posiciones absolutas de los robots. Fue necesario realizar algunas adaptaciones debido a las características de los robots, para poder usar *SwisTrack*. Por

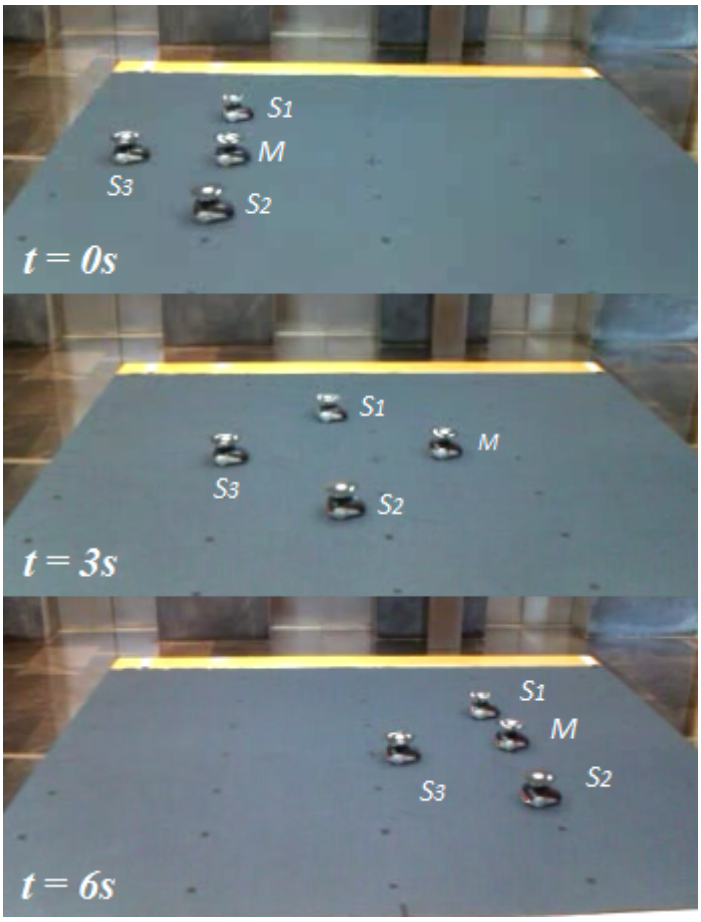


Figura 3.30. Resultados de experimento con los robots.

ejemplo los códigos identificadores para los robots se diseñaron más pequeños acorde al tamaño de los mismos.

En cuanto al retardo que introducen tanto el procesamiento de imágenes de la cámara de vídeo, como el envío y recepción a través de la red inalámbrica, se puede decir que es aceptable, teniendo en cuenta que la dinámica de los robots es bastante lenta. Lo que resulta beneficioso en este caso ya que permite tener un período de muestreo mayor. En un futuro se pretenden diseñar experimentos más complejos, donde el maestro tenga en cuenta la posición de los esclavos en su ley de control. Esto permitirá clasificar la plataforma como un sistema multi-agente, donde todos los elementos cooperan entre sí para lograr un objetivo común. Para este tipo de experimentos habrá que tener en cuenta la capacidad de procesamiento de los robots, que sería el principal cuello de botella para la implementación de experimentos más complejos.

3.5 Plataforma de robots móviles: *Surveyor SRV-1*

En esta sección se presenta el laboratorio remoto desarrollado con los robots *SRV-1*. Al igual que en los casos anteriores la estructura del laboratorio está basada en un arquitectura cliente - servidor [139, 154]. La Figura 3.31 muestra una representación de esta estructura.

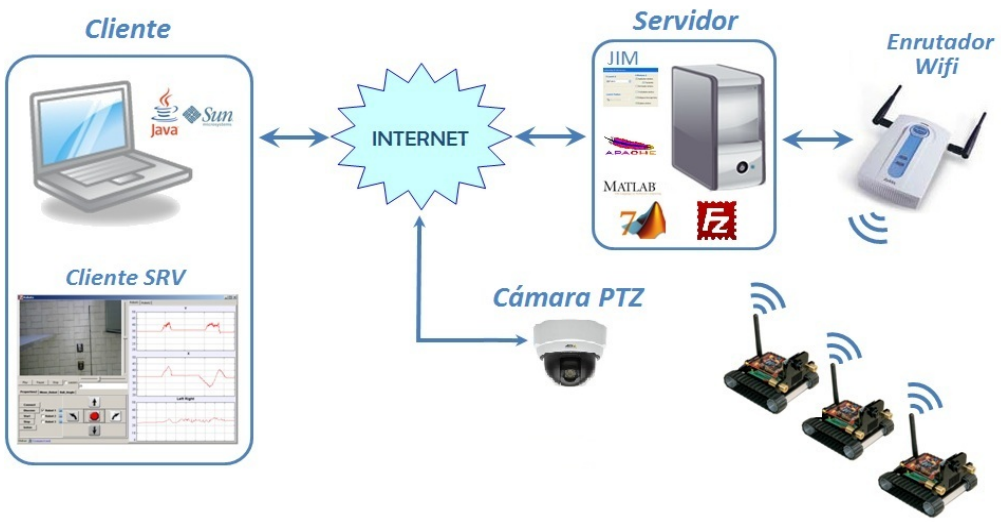


Figura 3.31. Arquitectura del laboratorio remoto.

En este caso, el lado del cliente consiste en una aplicación que se ha denominado *Cliente SRV*. La cual es una interfaz gráfica de usuario en forma de *Applet* que ha sido desarrollada en *EJS*, que como se ha explicado anteriormente, facilita la interactividad del usuario con el laboratorio. Además, resulta una gran ventaja porque se puede usar en cualquier ordenador con conexión a *Internet* y sin necesidad de instalar *Software* adicional. Solo es necesario un navegador, que como se conoce, en nuestros días éstos vienen incluidos en los sistemas operativos actuales. Como la aplicación cliente es un *Applet Java*, del lado del servidor se necesitan otros componentes, como se mencionó anteriormente. Además, la conexión con los robots *SRV-1* se establece a través de una red inalámbrica y usando *MATLAB*. Lo que añade una complejidad extra al laboratorio.

3.5.1 Aplicación *Cliente SRV*

Los experimentos que se pueden realizar con esta plataforma consisten en el control de formación de robots de tipo maestro - esclavos. Para llevar a cabo esta clase de experimentos es necesario establecer las posiciones iniciales de los robots. Una vez iniciado el experimento, se manipula al maestro desde la aplicación cliente, mientras que los esclavos ejecutan la ley de control que tienen implementada. En este caso, los robots tienen dos modos de trabajo: autónomo o descentralizado, cuando el robot ejecuta la ley de control que tiene programada y centralizado, cuando el robot ejecuta las órdenes que se le envían desde el servidor. Antes de iniciar un experimento todos los robots trabajan en modo centralizado, para el establecimiento de sus posiciones de trabajo. En este momento solo se selecciona un robot y se le envían órdenes de movimiento desde la aplicación cliente para establecer su posición inicial. Para iniciar el experimento se ordena a los esclavos ejecutar la ley de control que tienen programada, por lo que trabajan de forma autónoma; mientras que se sigue manipulando al maestro de forma centralizada para establecer su trayectoria de forma manual. En ese momento el maestro se moverá siguiendo las órdenes del usuario, mientras que el resto de los robots se moverán siguiendo al maestro. Una vez terminado el experimento se podrán guardar los datos resultantes para su posterior análisis. Para llevar a cabo estas tareas la aplicación *Cliente SRV* implementa las siguientes funcionalidades:

- Mostrar las imágenes de la cámara *PTZ ("Pan-Tilt-Zoom")* y permitir acceso a diversas funcionalidades de la cámara. A través de estas funciones el usuario puede cambiar el área de enfoque de la cámara para no perder de vista a los robots en ningún momento. Además puede realizar un acercamiento ("*zoom*") para observar si se requieren detalles.
- Mostrar los robots disponibles para establecer la conexión.
- Permitir establecer el modo de trabajo de los robots (autónomo o centralizado).
- Permitir establecer los parámetros de los controladores para cada uno de los robots.
- Mover los robots individualmente para establecer las posiciones iniciales.
- Visualizar los datos de los robots durante el desarrollo de un experimento.

- Visualizar las imágenes de la cámara de abordo de los robots.
- Guardar los datos resultantes de un experimento.

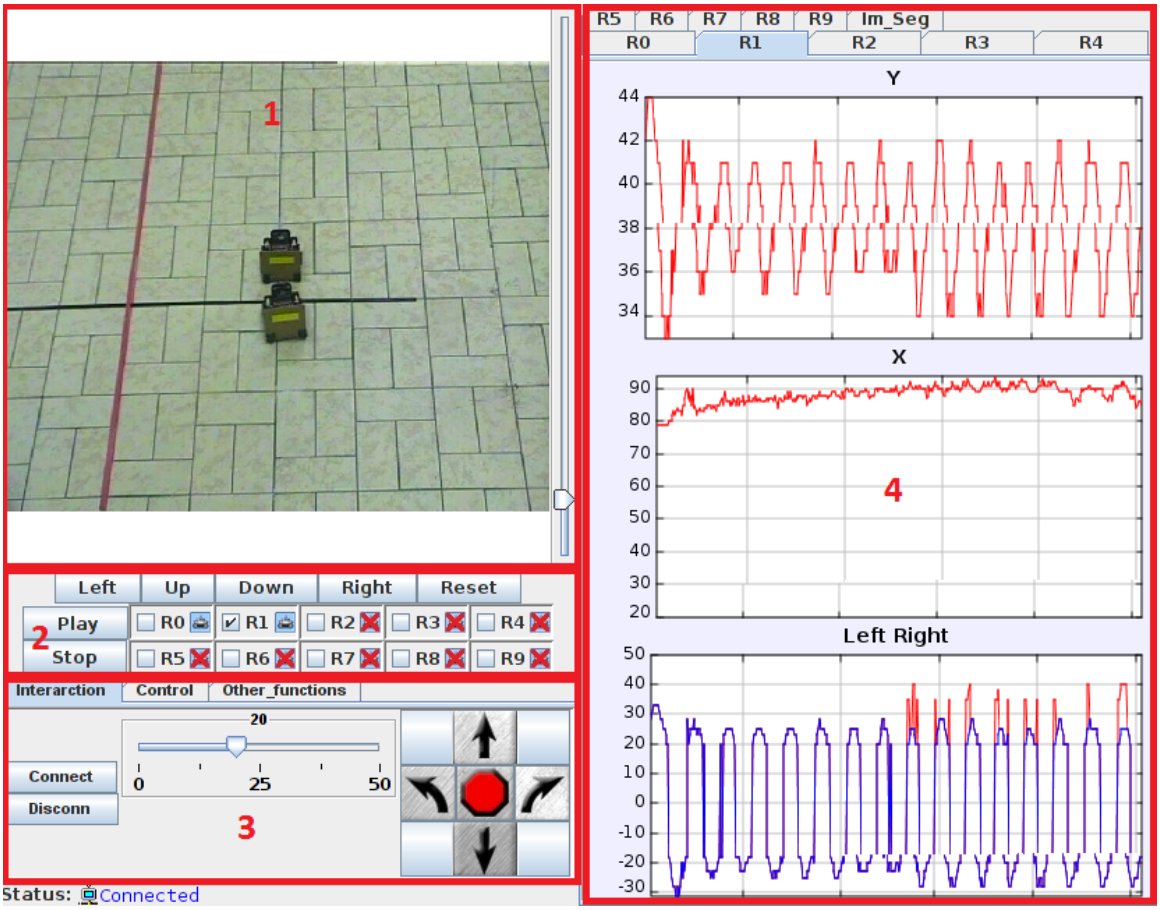


Figura 3.32. Interfaz gráfica de la aplicación *Cliente SRV*.

En la Figura 3.32 se observa la interfaz gráfica de usuario de la aplicación *Cliente SRV*. Como se puede apreciar, en el panel central se muestran las imágenes de la cámara (No. 1 en la Figura 3.32). En la parte derecha de este panel hay un control deslizante que permite controlar el "*zoom*" de la misma. Mediante los botones (arriba, abajo, derecha e izquierda) del panel No. 2, se puede modificar el área de enfoque de la cámara para no perder de vista a los robots durante el desarrollo de los experimentos. Mientras que con el botón *Reset*, se puede establecer la posición inicial de la cámara. Este panel también se emplea para seleccionar los diferentes robots y enviarles órdenes en modo centralizado. A su vez, los iconos muestran el estado de los robots: desconectados (una X de color rojo) o conectados (icono de color azul). Cuando un robot está desconectado,

no se puede seleccionar.

El panel No. 3 de la Figura 3.32 tiene tres pestañas. Con la primera pestaña se puede iniciar o detener el algoritmo de control. Además, con los botones se pueden mover los robots en todas las direcciones según indican las flechas que éstos poseen. También, por medio del control deslizante que aparece a la derecha de este panel, se puede establecer la velocidad del robot seleccionado, cuando éste trabaja en modo centralizado. A través de la segunda pestaña se pueden modificar los parámetros del controlador para el robot seleccionado con el panel anterior. La tercera pestaña sirve para otras funcionalidades como guardar los datos de un robot al finalizar un experimento. El panel No. 4 de la Figura 3.32 tiene dos pestañas. La primera muestra tres gráficos con los datos provenientes de los robots cuando éstos trabajan en modo autónomo. El robot cuyos datos se deseen visualizar se puede seleccionar por medio de los controles de selección que aparecen en la parte baja de este panel. Las tres gráficas muestran los siguientes datos:

1. Distancia en centímetros al robot de enfrente en el caso de los esclavos. Si el maestro está seleccionado esta gráfica mostrará la velocidad del maestro.
2. La orientación del robot con respecto al robot de enfrente. esto indica cómo de desalineado está el robot con respecto al centro del rectángulo pegado en la parte posterior del robot de delante.
3. Las dos señales PWM enviadas a los motores de la derecha y de la izquierda.

La segunda pestaña muestra las imágenes de la cámara del robot seleccionado con el panel No. 5, tal como se observa en la Figura 3.33.

El panel No. 1 de la Figura 3.33, muestra una imagen tomada de la cámara de a bordo del robot seleccionado, cuando se selecciona la opción *Normal* en el panel No. 4. El panel No. 2 de la Figura 3.33 muestra una imagen tomada de la cámara del robot seleccionado cuando en el panel No. 4 se selecciona la opción *Segmentación*. El panel No. 3 de la Figura 3.33 muestra la imagen obtenida cuando se selecciona en el panel No. 4 la opción de *Bordes*. Solo se puede obtener una imagen a la vez, la aplicación solo actualizará la imagen seleccionada con el panel No. 4. Las funciones de las opciones que se pueden seleccionar con el panel No. 4 son las siguientes:

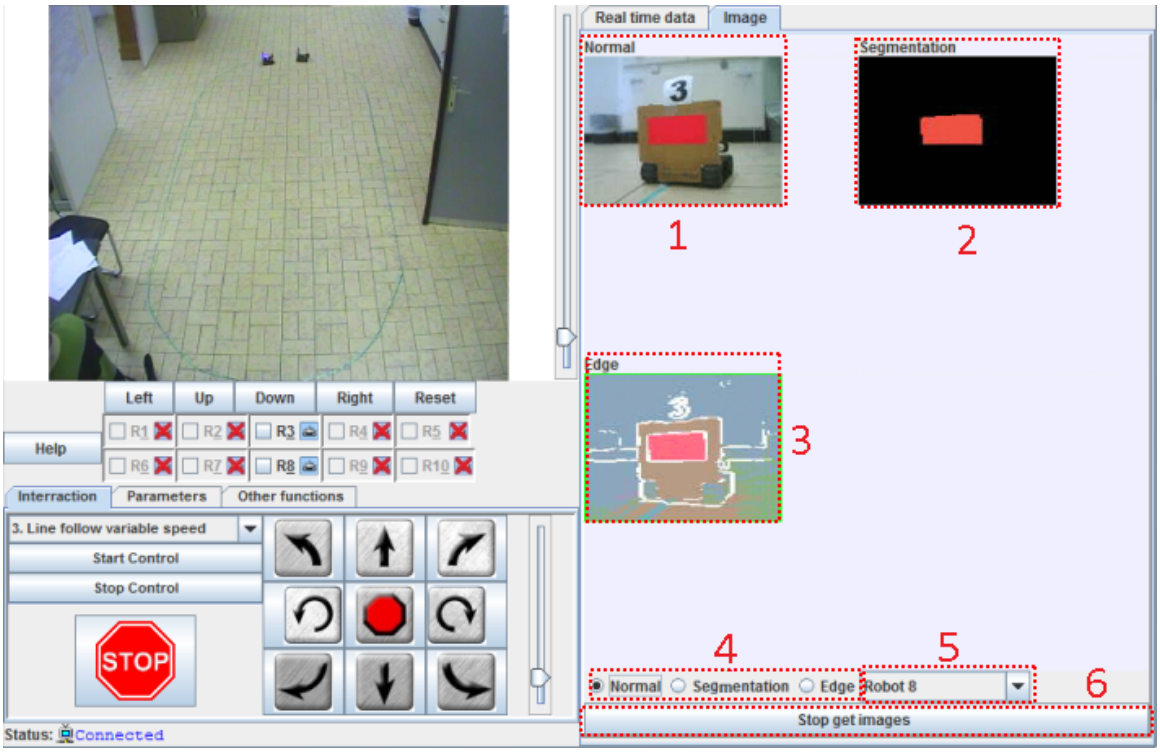


Figura 3.33. Imágenes tomadas por la cámara de a bordo del robot.

- *Normal*: con esta opción se obtiene una imagen de la cámara sin procesar.
- *Segmentación*: con esta opción se obtiene una imagen resultante después de aplicar un algoritmo de segmentación en busca de un color específico.
- *Bordes*: con esta opción se obtiene una imagen resultante después de aplicar el algoritmo de detección de bordes.

El panel No. 6 es un botón que se emplea para iniciar y detener la obtención de las imágenes provenientes del robot. Hay que tener en cuenta que no es posible obtener imágenes del robot en modo autónomo.

3.5.2 Aplicación *Servidor SRV*

Del lado del servidor se encuentran un conjunto de componentes de *software* y *hardware* que proporcionan las diferentes funcionalidades que requiere la aplicación cliente. La Figura 3.34 muestra el esquema de las interacciones que se establecen entre los componentes del cliente y el servidor, así como las conexiones internas entre los componentes del servidor.

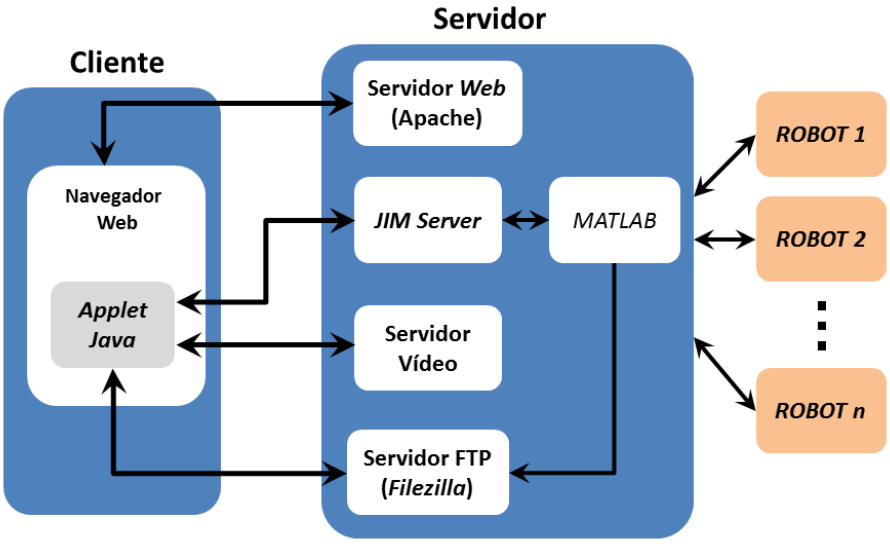


Figura 3.34. Interacción entre los diferentes componentes *software* y *hardware*.

Como se había mencionado anteriormente, la aplicación cliente es un *Applet Java* por lo que se necesita un servicio *web* que se ejecute en el servidor, en este caso *Apache*, que es de libre distribución. Este servicio *web* hospeda el sitio *web* que contiene el *Applet*. Cuando el usuario visita el sitio *web* la aplicación cliente se descarga a su navegador y se inicia. Al iniciarse, esto da lugar a que se ejecute una instancia de *MATLAB* en el servidor ya que la aplicación cliente se conecta con *JIM Server* para que ordene iniciar *MATLAB*. El envío de órdenes a los robots en los dos modos de trabajo, se realiza a través de *MATLAB*, por lo que se puede decir que las órdenes enviadas desde la aplicación cliente, se "traducen" a instrucciones de *MATLAB* que son los que finalmente obedecen los robots. Al mismo tiempo en el *Applet* se recibe el vídeo de la cámara *PTZ* que también se encuentra conectada a la red. Si el usuario presiona el botón para guardar los datos, esto enviará al robot una instrucción de *MATLAB* que le indica que envíe los datos para almacenarlos. Estos datos se almacenan en un fichero en una localización conocida en el servidor. Los datos pueden descargarse posteriormente usando el protocolo *FTP*. Para hacer esto se encuentra ejecutándose en el servidor un servicio *FTP* garantizado por el programa *Filezilla*, que también es de libre distribución. La comunicación entre *MATLAB* y los robots se realiza mediante un enrutador inalámbrico al cual se encuentra conectado el servidor, tal como se muestra

en la Figura 3.31. Las direcciones *IP* tanto para el servidor como para los robots son estáticas.

Por su parte, los robots *Surveyor SRV-1* pueden controlarse enviándoles órdenes desde un ordenador al micro-programa "*firmware*" que se ejecuta a bordo del robot. El programa implementado en el robot se ejecuta como un bucle infinito que está contínuamente recibiendo órdenes del usuario y respondiendo consecuentemente. Las órdenes se pueden dividir en varios grupos [155]. Algunos de estos grupos y sus funcionalidades son los siguientes:

- Órdenes principales: conducir el robot, cambiar la resolución, leer y escribir en la memoria del robot, encender y apagar los *lasers*, obtener las imágenes y establecer las velocidades de los motores directamente, entre otras.
- Órdenes especiales: reiniciar el microprocesador, leer los acelerómetros, etcétera
- Órdenes de visión: habilitar la segmentación de color, habilitar la detección de bordes, cambiar las opciones de la cámara (ganancia automática, balance de blancos y exposición), búsqueda de manchas, obtención de los valores de color para un píxel en particular, etcétera.
- Intérprete de C: ejecuta el código en lenguaje C implementado y almacenado en la memoria del robot.

Las órdenes son caracteres *ASCII* que representan el nombre de la orden y algunas de ellas además pueden recibir parámetros. Estas órdenes se pueden enviar desde cualquier programa que pueda establecer conexión *Telnet*.

En este caso para la implementación del laboratorio remoto el robot se controla usando estas órdenes para llevar el robot a la posición deseada, tomar las imágenes de la cámara así como establecer sus propiedades, etc. En etapas anteriores del desarrollo del laboratorio se implementó el modo autónomo transfiriendo las imágenes al ordenador que actúa como servidor, para su procesamiento y luego enviar al robot las órdenes. La ventaja de esta variante es que se añade al sistema una alta capacidad de procesamiento de imágenes usando por ejemplo, la librería *OpenCV* [156]. Este enfoque fue desechado porque además de centralizar el proceso, introducía demoras importantes por el retardo de la red inalámbrica.

3.5.3 Consideraciones prácticas de la plataforma con *Surveyor SRV-1*

La plataforma permite acceder remotamente a un laboratorio que consiste en controlar la formación de dos o más robots que siguen a un maestro. La estructura del laboratorio remoto es cliente - servidor, similar a la de los casos anteriores. Las principales diferencias están en el lado del servidor ya que la comunicación en este caso se establece a través de *JIM Server* con *MATLAB*. Además la comunicación entre el servidor y los robots se realiza inalámbricamente a través de un enrutador inalámbrico. En el servidor, dentro del directorio de trabajo de *MATLAB* se encuentran las funciones necesarias para interaccionar con los robots. De esta forma las órdenes transmitidas desde la aplicación cliente, se traducen a órdenes que se envían a los robots. El programa que ejecutan los robots se graba en la memoria interna que estos poseen antes de iniciar el experimento. La plataforma resulta ser una herramienta interactiva que permite desarrollar prácticas de laboratorio con robótica móvil. Las principales dificultades se presentaron en la medición de las velocidades de los robots que es necesaria para su control. Estos robots no poseen sensores de velocidad, lo que dificulta su control de posición. Además fue necesario tener en cuenta los retardos en la comunicación de la red inalámbrica. Otra limitante que condiciona el desarrollo de nuevos experimentos es el hecho de que los robots no se pueden comunicar entre ellos. Para poder mejorar la implementación del control de posición sería interesante añadir a los robots comunicación inalámbrica entre ellos, por ejemplo *XBee*. Esto proporcionaría a la plataforma la posibilidad de realizar otro tipo de experimentos de control de formación con sistemas multi-agente. También se podría incorporar a la plataforma una cámara cenital para facilitar la obtención de las posiciones de los robots para el control de formación. Para todo esto habría que tener en cuenta la capacidad de procesamiento de los robots y los retardos de la red inalámbrica, que pueden repercutir en los algoritmos de control.

4

Experiencias prácticas de control y robótica

En este capítulo se presentan los resultados obtenidos para diferentes experiencias realizadas con las plataformas de experimentación de control automático y robótica móvil para cada uno de los sistemas estudiados en los capítulos anteriores.

Para la plataforma del sistema de bola y aro, se analizan los resultados obtenidos para los experimentos de ceros de transmisión, control de posición del aro, control del ángulo de desviación de la bola y comportamiento de fase no mínima del sistema. Mientras que para el caso de la plataforma del sistema de bola y plato, se presentan los resultados obtenidos para los experimentos de control de posición de la bola en un punto del plato y de seguimiento de trayectorias: circular y cuadrada.

En el caso de la plataforma de los robots *Moway* se presentan los resultados para las experiencias de control de posicionamiento en un punto y control de formaciones de tipo maestro-esclavos con evasión de obstáculos. Mientras que para la plataforma de los

robots *Surveyor SRV-1* se muestran los resultados de los experimentos de control de formación de tipo líder-seguidores. Como se explicó anteriormente en la Sección 2.2.1.3, la diferencia entre estos dos tipos de formaciones radica en la referencia tanto para los robots seguidores como para los robots esclavos ya que los robots esclavos toman como referencia la posición de su líder, mientras que los robots seguidores toman como referencia al robot que tienen en frente.

4.1 Experimentos con la plataforma del sistema bola y aro

En esta sección se describen los resultados obtenidos para los experimentos realizados con la plataforma implementada para el sistema de bola y aro.

4.1.1 Simulación de los ceros de transmisión del sistema

Como se describió en la Sección 2.1.1.2, este experimento consiste en aplicar una señal sinusoidal a la entrada del sistema, en este caso el voltaje del motor, para provocar un movimiento oscilatorio a la salida, en este caso la posición del aro. Este movimiento oscilatorio provoca a su vez, un movimiento oscilatorio en la posición de la bola.

En la ventana principal de la aplicación cliente (ver Fig. 4.1), inicialmente por defecto se encuentra selecciona la opción *Init*. Esta opción se utiliza para establecer condiciones iniciales (ángulo del aro ($\theta = 0$), voltaje del motor ($\tau = 0$) y posición de la bola en el interior del aro ($y = 0$).

Para iniciar este experimento, el estudiante debe seleccionar la opción *Zeros* y luego presionar el botón *Play* para poner en marcha la simulación. Para iniciar cada uno de los experimentos restantes el procedimiento es el mismo, cambiando solamente la selección del experimento que se desee realizar. Al ejecutar la simulación de los ceros de transmisión, el aro y la bola comienzan a oscilar, pero se observa que no están sincronizados, ya que inicialmente la frecuencia de la señal sinusoidal aplicada al motor es arbitraria. El estudiante por tanto, debe variar la frecuencia de la señal sinusoidal hasta encontrar la frecuencia de los ceros de transmisión del sistema calculada previamente, que sincroniza la oscilación del aro y la bola. Nótese que el valor de la

frecuencia de los ceros de transmisión se obtiene a partir de la Ecuación 2.3.

Al suministrar al sistema la señal sinusoidal con la frecuencia de los ceros de transmisión, se observa que la bola y el aro oscilan a la misma frecuencia, como si la bola estuviera "fija" en el aro. La Figura 4.1 muestra un ejemplo de este experimento cuando se aplica al sistema la señal con la frecuencia de los ceros de transmisión. Nótese que los gráficos se han escalado convenientemente con el objetivo de observar tanto las oscilaciones del aro como las de la bola.

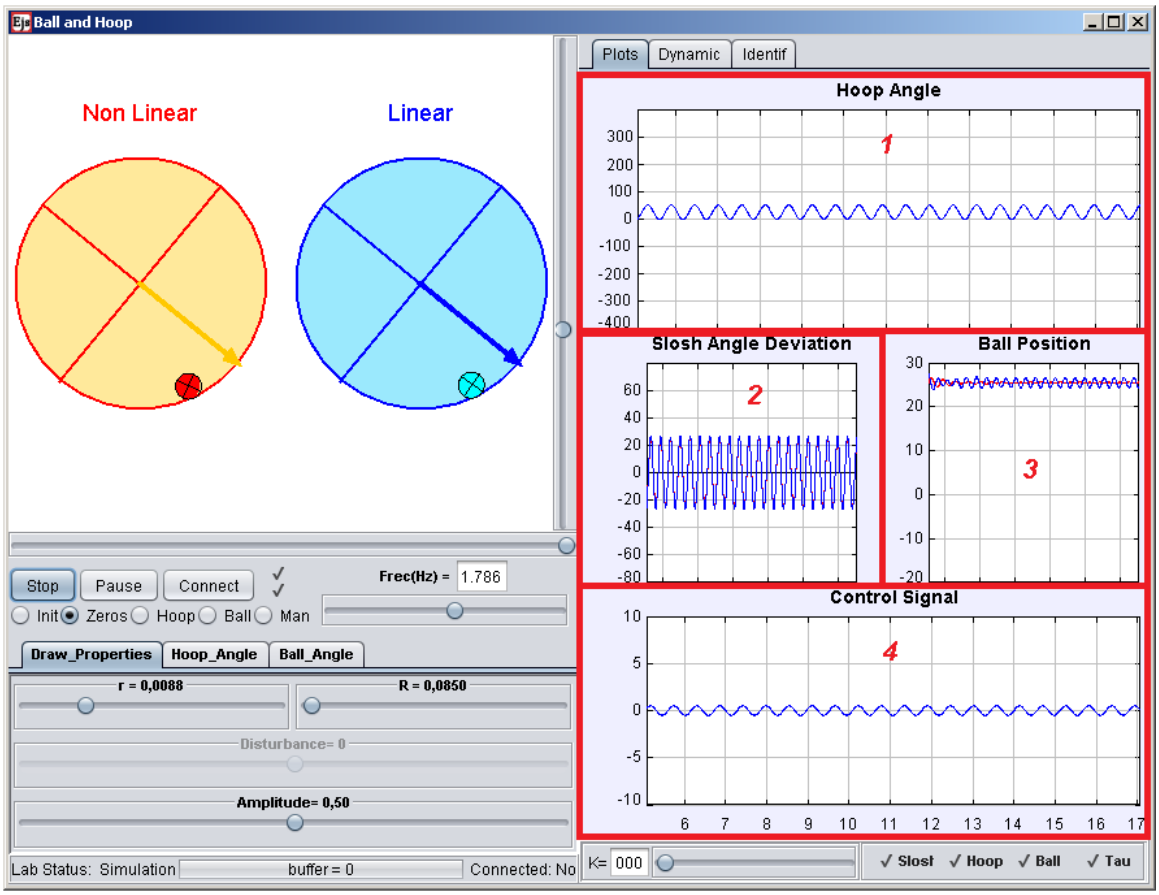


Figura 4.1. Experimento de los ceros de transmisión.

Como se puede apreciar, en los gráficos situados en el lado derecho de la ventana principal, el ángulo de posición del aro (Panel No. 1), describe un movimiento oscilatorio que se corresponde con la señal sinusoidal que se aplica a la entrada del motor (señal de control, Panel No. 4). En ambos casos, aunque se dibujan tanto la señal para el modelo lineal, como para el modelo no lineal, como ambas coinciden, se observa solo la señal de color azul, que se corresponde con el modelo lineal del sistema. La amplitud de esta

señal sinusoidal es un valor pequeño, lo que provoca que las oscilaciones del ángulo de posición de la bola (Panel No. 2 *Slosh Angle Deviation*), sean a su vez pequeñas. Al ser pequeñas estas oscilaciones, ambos modelos del sistema (el lineal y el no lineal), se comportan de forma parecida; debido a que una de las consideraciones que se tienen en cuenta en la linealización del sistema es que para valores pequeños de este ángulo se cumple que: $\sin\psi \approx \psi$. Por este motivo las diferencias entre ambos modelos son prácticamente imperceptibles. En cambio, en el gráfico correspondiente con la posición de la bola en el interior del aro (Panel No. 3) no ocurre lo mismo. Para el modelo no lineal la posición de la bola en el interior del aro es constante, lo que indica que la bola está "fija" en el aro (respuesta de cero exacto en la posición de la bola). Esto demuestra un buen funcionamiento del modelo no lineal para este experimento. Mientras que para el modelo lineal se aprecian en el gráfico pequeñas variaciones en la posición de la bola. Estas variaciones son solamente perceptibles en el gráfico, pues en la animación de la simulación no se observan, lo que indica que son muy pequeñas. Esto ocurre porque el modelo lineal está basado en las aproximaciones descritas anteriormente, que hacen que los dos modelos no se comporten exactamente igual.

Usando este experimento también se puede observar el fenómeno de la ocurrencia de resonancia. Para demostrarlo, la frecuencia se debe disminuir hasta que la posición de la bola en el interior del aro y la posición del aro, oscilen a amplitudes constantes. En este momento tiene lugar el pico resonante del Diagrama de Bode que se describió en la Figura 2.6. La frecuencia de la señal sinusoidal de la entrada puede ser modificada por un control deslizante que se encuentra debajo de la animación. El valor de la frecuencia (Hz) seleccionada se puede observar en el campo editable situado encima del control deslizante.

4.1.2 Control de la posición del aro en modo remoto

Este experimento consiste en aplicar un salto escalón a la entrada (de 60° a 200°), que se corresponde con la referencia del ángulo de posición del aro y observar el comportamiento del sistema. El experimento comienza al seleccionar la opción *Hoop* de la aplicación y modificar la referencia del ángulo del aro. Una vez iniciado el experimento,

el bucle de control hará que el aro siga a la referencia, sin tener en cuenta el movimiento oscilatorio que esto provocará en la posición de la bola, desviándola de su posición inicial de equilibrio. Obsérvese que en la Figura 4.2 se ha realizado la conexión con la planta real a través del botón *Connect*.

El estudiante puede modificar la referencia del aro a través de un control deslizante que se encuentra debajo de la animación. Un campo de edición muestra el valor de la referencia en grados. Los parámetros del controlador *PI* pueden ser modificados en la pestaña *HoopAngle*.

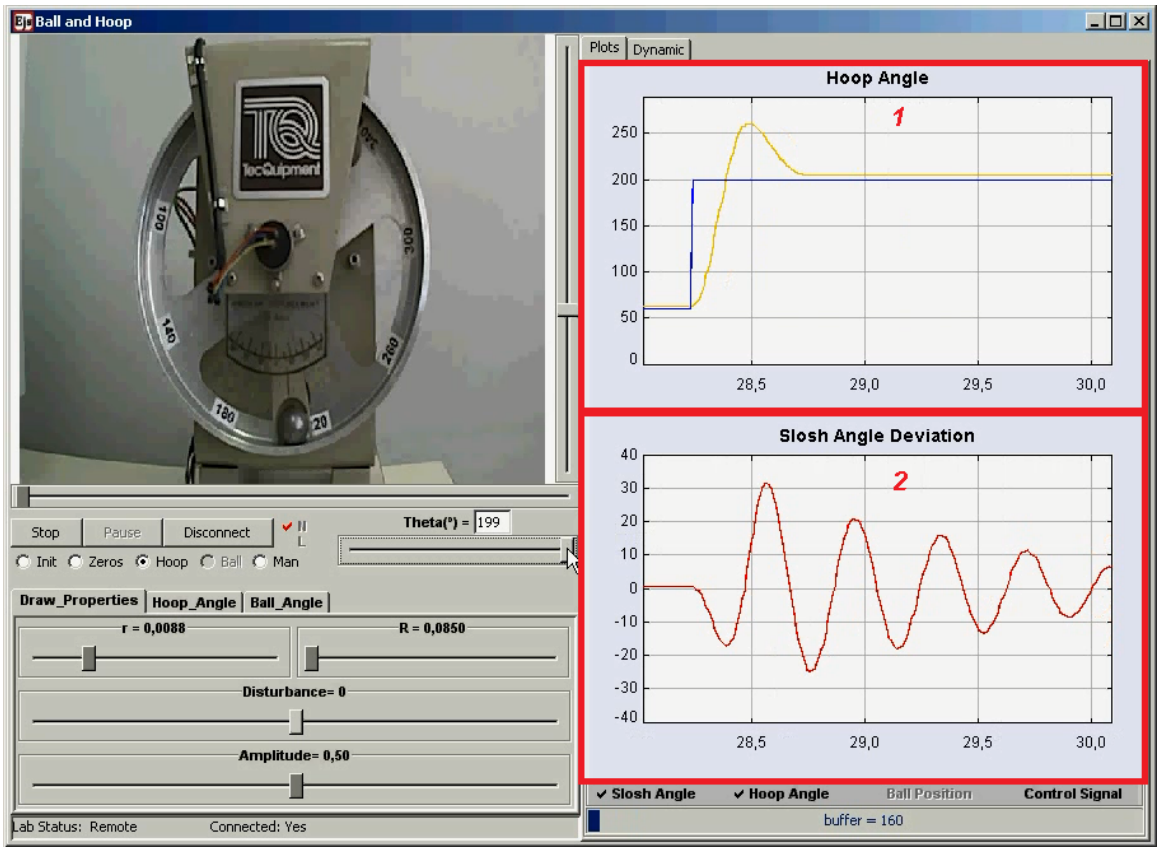


Figura 4.2. Control de posición del aro.

El Panel No. 1 (*Hoop Angle*) de la Figura 4.2 muestra el gráfico de la referencia del ángulo del aro y el ángulo del aro (θ). Mientras que el Panel No. 2 (*Slosh Angle Deviation*) muestra el gráfico del ángulo de desviación de la bola (ψ). En este caso, a diferencia del experimento anterior, en los gráficos solo se muestran los resultados de la planta real, puesto que la plataforma se encuentra en funcionando en modo remoto

(conexión a través de *Internet* con la planta real).

Como se puede apreciar el control de posición del aro funciona correctamente teniendo en cuenta que el error de estado estacionario que se observa es debido a la zona muerta del motor. Como se explicó anteriormente, en este experimento el control de posición del aro no tiene en cuenta la posición de la bola, por lo que el movimiento brusco de éste, ocasiona un movimiento oscilatorio bastante pronunciado en la posición de la bola (hasta 30°). Transcurrido un tiempo, la bola vuelve a su posición de equilibrio. Para evitar este comportamiento brusco de la bola se debe realimentar el ángulo de posición de ésta, para que se tenga en cuenta al controlar la posición del aro. Los resultados obtenidos al realimentar la posición de la bola se presentan en la próxima sección.

4.1.3 Control de la posición del aro en modo simulación

En el experimento de control de posición del aro en modo simulación, para mostrar el comportamiento de fase no mínima del sistema, un control deslizante permite modificar el valor de la ganancia K_s de la Ecuación 2.5. Cuando se incrementa el valor de la ganancia, el valor de la señal escalada $x(s)$ se muestra en el gráfico y se puede observar como para un cambio en la referencia inicialmente va en la dirección contraria (comportamiento de fase no mínima). El Panel No. 1 de la Figura 4.3 (*Hoop Angle*) muestra el gráfico de la señal escalada $x(s)$.

Es necesario señalar que este sistema en su modo de funcionamiento real no presenta este comportamiento para ninguno de los experimentos que con él se realizan. En modo de simulación la señal escalada $x(s)$ se construye tomando como salidas la posición del aro ($\theta(s)$) y la desviación de la bola de su posición de equilibrio ($\psi(s)$) como se describió anteriormente en la Ecuación 2.4. Esta señal permite a los estudiantes observar el comportamiento de fase no mínima.

4.1.4 Control del ángulo de desviación de la bola en modo remoto

En este experimento se emplea el lazo de control presentado anteriormente para controlar la posición del aro. Agregándose la realimentación de la posición de la bola a

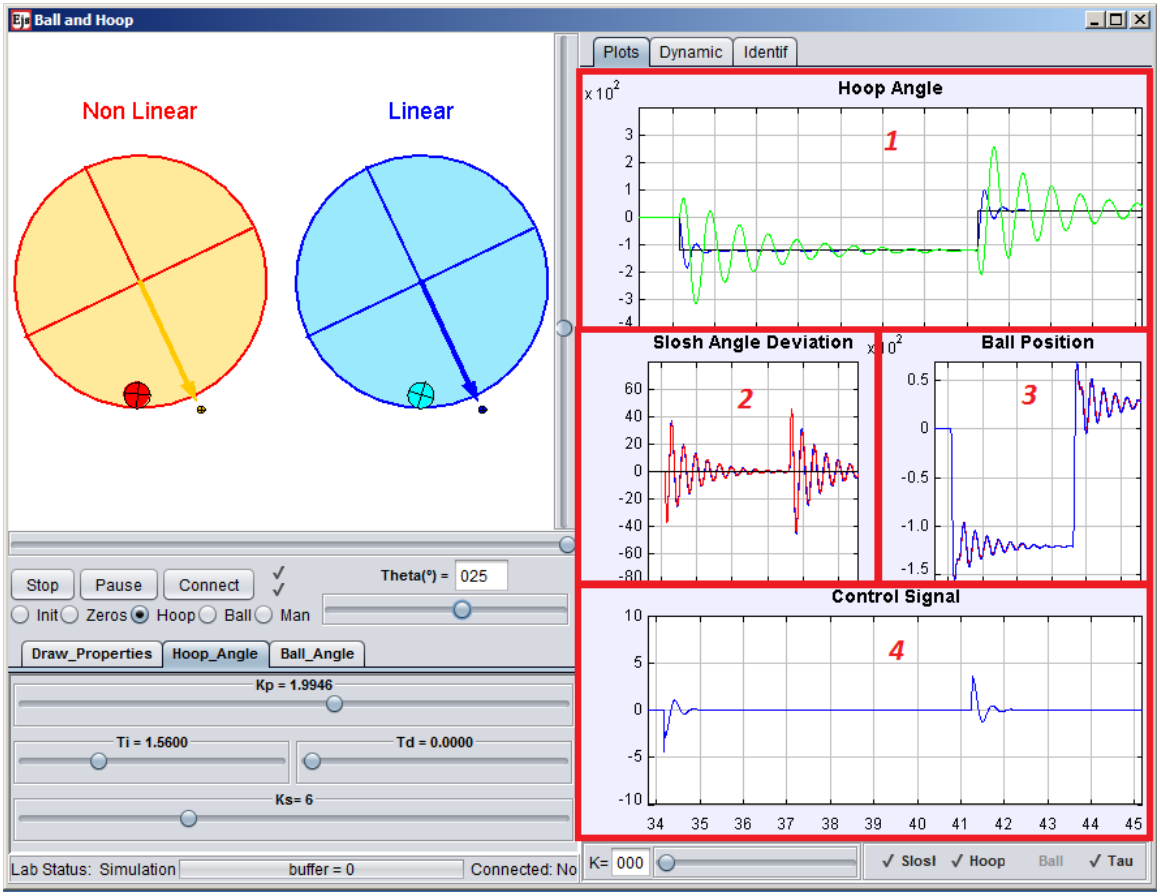


Figura 4.3. Comportamiento de fase no mínima.

través de la ganancia de realimentación de la posición de la bola (K), para demostrar el control de desviación de su posición de equilibrio. El valor de la ganancia K se puede modificar a través de un control deslizante.

Al incrementar este valor de la ganancia K la dinámica del aro se ve afectada, moviéndose éste más lentamente, provocando una menor desviación de la bola con respecto su posición de equilibrio. Para valores grandes de K el aro se mueve muy lentamente hasta alcanzar su estado estacionario, mientras que la bola casi no se desplaza con respecto a su posición inicial de equilibrio. Para valores pequeños de K, el aro responde rápidamente ante un cambio en la referencia llegando al estado estacionario en un tiempo menor que el anterior. Sin embargo, esto causa una mayor desviación de la posición de equilibrio de la bola.

La Figura 4.4 muestra los resultados para este experimento. Como se observa el aro tarda más en alcanzar su posición final, lo que indica que su dinámica es un poco

más lenta (Panel No. 1). Mientras que el ángulo de desviación de la bola (Panel No. 2) experimenta oscilaciones menos pronunciadas (la mayor desviación no alcanza los 20°), debido a que el control de posición del aro tiene en cuenta la posición de la bola en mayor o menor medida dependiendo de la ganancia K.

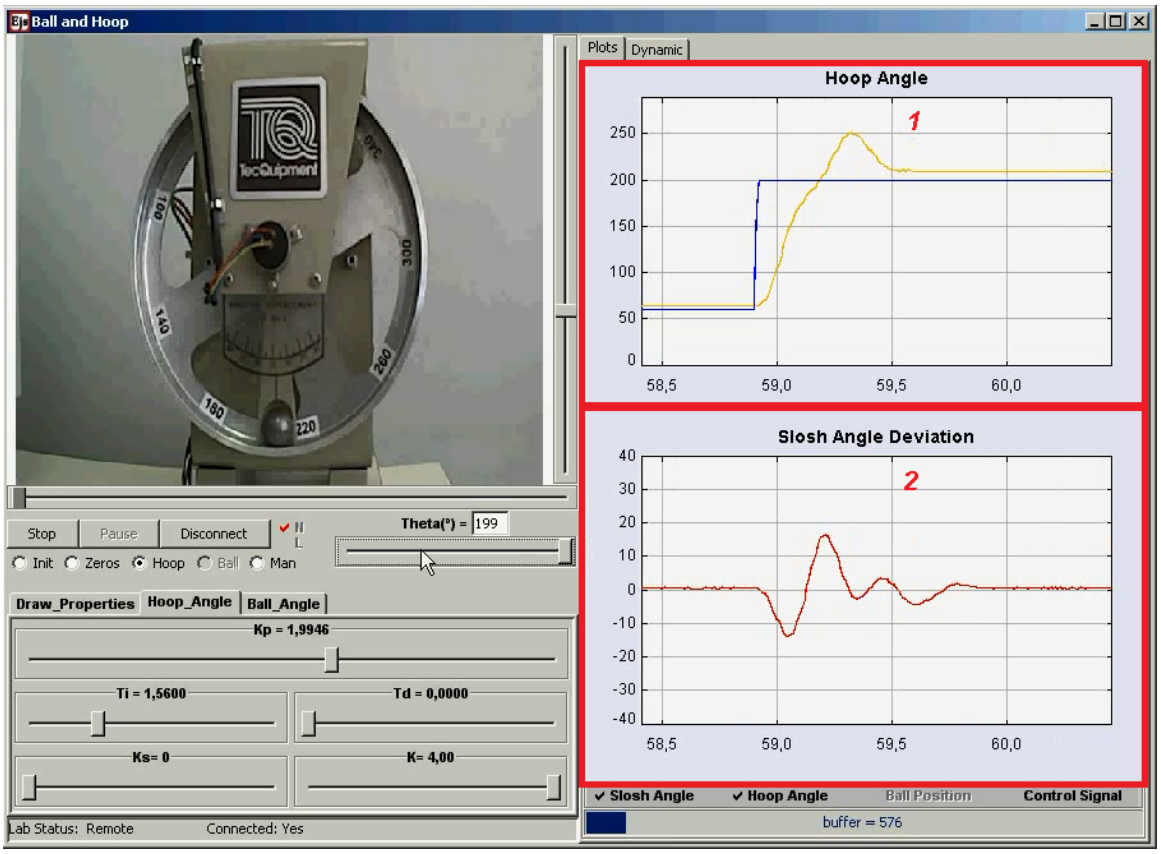


Figura 4.4. Control de ángulo de desviación de la bola.

Los resultados de la Figura 4.4 muestran el comportamiento del sistema para el valor recomendado de la ganancia K. Como se mencionó anteriormente en la Sección 2.1.1.2, el valor de la ganancia K se obtiene analizando el lugar geométrico de las raíces para el lazo cerrado. Se escoge dicho valor para el cual los dos pares de polos complejos conjugados (los que corresponden a la dinámica del aro y los que corresponden a la dinámica de la bola), tengan la misma parte real. De esta forma se garantiza que la influencia de ambas dinámicas sobre el sistema sea la misma. Físicamente esto implica que para este valor de K el aro se va a mover a la mayor velocidad posible causando la menor desviación posible en la posición de la bola, por lo que ésta recupera más

rápidamente su posición inicial de equilibrio, lo cuál es el objetivo fundamental para el que fue diseñado este sistema.

4.1.5 Análisis de los resultados obtenidos

En el caso del funcionamiento en modo simulación, el comportamiento de la plataforma es el esperado. Los resultados obtenidos han sido satisfactorios para todos los experimentos realizados (control de posición del aro, control de ángulo de desviación de la bola de su posición de equilibrio, ceros de transmisión, frecuencia de resonancia y comportamiento de fase no mínima). Además, estos resultados se han mantenido en los valores esperados al modificar las propiedades físicas del sistema en el modelo matemático de la animación (radio del aro, radio de la bola, etc). Esto indica que desde el punto de vista de diseño se ha logrado obtener una buena simulación de este sistema que presenta una complejidad añadida debido específicamente a la interacción entre la bola y el aro.

En el caso del funcionamiento en modo remoto (conexión a través de *Internet* con la planta real), los resultados también pueden catalogarse de satisfactorios para todos los experimentos realizados. En estos resultados ha influido principalmente que la señal de control, se calcula en el ordenador que actúa como servidor y al que se encuentra conectada la planta real, por lo que la señal de control se envía directamente a la planta a través de la tarjeta de adquisición de datos. Por este motivo, los posibles problemas de latencia de la red en el control de la planta, no afectan al funcionamiento de la plataforma ya que la planta solo recibe a través de la red la referencia de posición del aro. Esto indudablemente constituye un beneficio sustancial para el buen funcionamiento de la plataforma en modo remoto. La implementación del control a través de la red es un problema diferente con la complejidad añadida que supone al pérdida de información.

4.2 Experimentos con la plataforma del sistema bola y plato

En esta sección se describen los resultados obtenidos para los experimentos realizados con la plataforma implementada para el sistema de bola y plato.

4.2.1 Control de posición de la bola en modo simulación

Este experimento consiste en marcar un punto sobre el plato que representa la referencia en el bucle de control de posición. Usando esta referencia y la posición actual de la bola sobre el plato, el bucle de control calcula la señal de control, que representa el voltaje necesario a aplicar a los motores; para que el plato se incline causando a su vez un desplazamiento de la bola hacia el punto señalado.

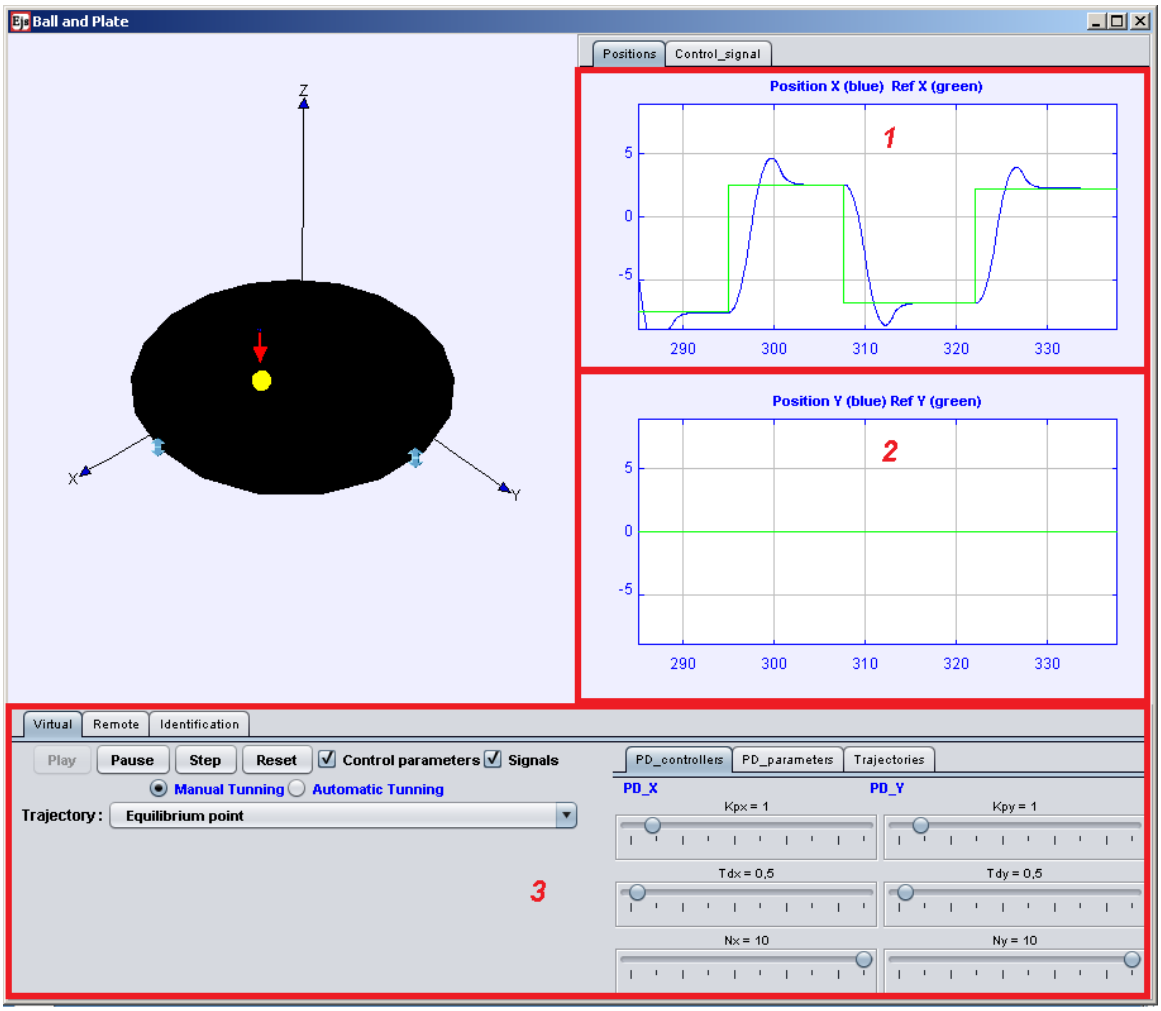


Figura 4.5. Control de posición de la bola en modo local.

La Figura 4.5 muestra el funcionamiento de la plataforma en modo simulación. El experimento comienza al seleccionar la opción *Equilibrium Point* en el menú desplegable *Trajectories* de la pestaña *Virtual* (Panel No. 3). Una vez seleccionadas estas características se inicia el experimento presionando el botón *Play*. El estudiante debe

modificar la referencia de posición de la bola mediante la flecha roja que se aprecia en la animación.

En la parte derecha de *GUI* de la aplicación se encuentran los gráficos correspondientes a la posición de la bola sobre el plato: el Panel No.1 corresponde con la coordenada X y el Panel No. 2 con la coordenada Y. Como se puede apreciar, la referencia de la coordenada X de la posición de la bola (color verde) se varía introduciendo varios saltos escalón. Como se observa, la coordenada X de la posición de la bola (color azul) sigue a la referencia. Por otra parte, también se puede apreciar que la coordenada Y de la referencia y de la posición de la bola permanecen constantes, esto indica que la referencia se establece manteniendo una de las dos coordenadas fija, por lo que la bola debe seguir una trayectoria rectilínea paralela al eje x desde su posición actual hasta la referencia establecida.

4.2.2 Control de posición de la bola en modo remoto

Para cambiar a modo remoto el estudiante debe seleccionar la pestaña *Remote*. Una vez en esta pestaña el estudiante debe presionar el botón *Connect* para conectar con la planta real. Al establecer la conexión con la planta real la imagen de la animación se sustituye por la señal de vídeo de la cámara *IP* (Panel No. 1) que muestra el funcionamiento de la planta real.

La Figura 4.6 muestra la aplicación funcionando en modo remoto. En el caso particular de esta aplicación los datos de posición de la bola no se envían de vuelta a la aplicación cliente. Como se explicó anteriormente, esto es debido a que el procesamiento de imágenes para la obtención de la posición de la bola, consume muchos recursos al igual que la conexión remota con la aplicación cliente. Por estos motivos el bucle de control local podría verse afectado si se agregara la funcionalidad de enviar los datos a la aplicación cliente. En su lugar los datos se almacenan en el ordenador que actúa como servidor. El estudiante puede acceder posteriormente al fichero que contiene los datos a través de *Internet* usando una conexión *FTP*. Una vez obtenido el fichero de datos, el estudiante puede exportar los datos al espacio de trabajo de *MATLAB* para tratarlos.

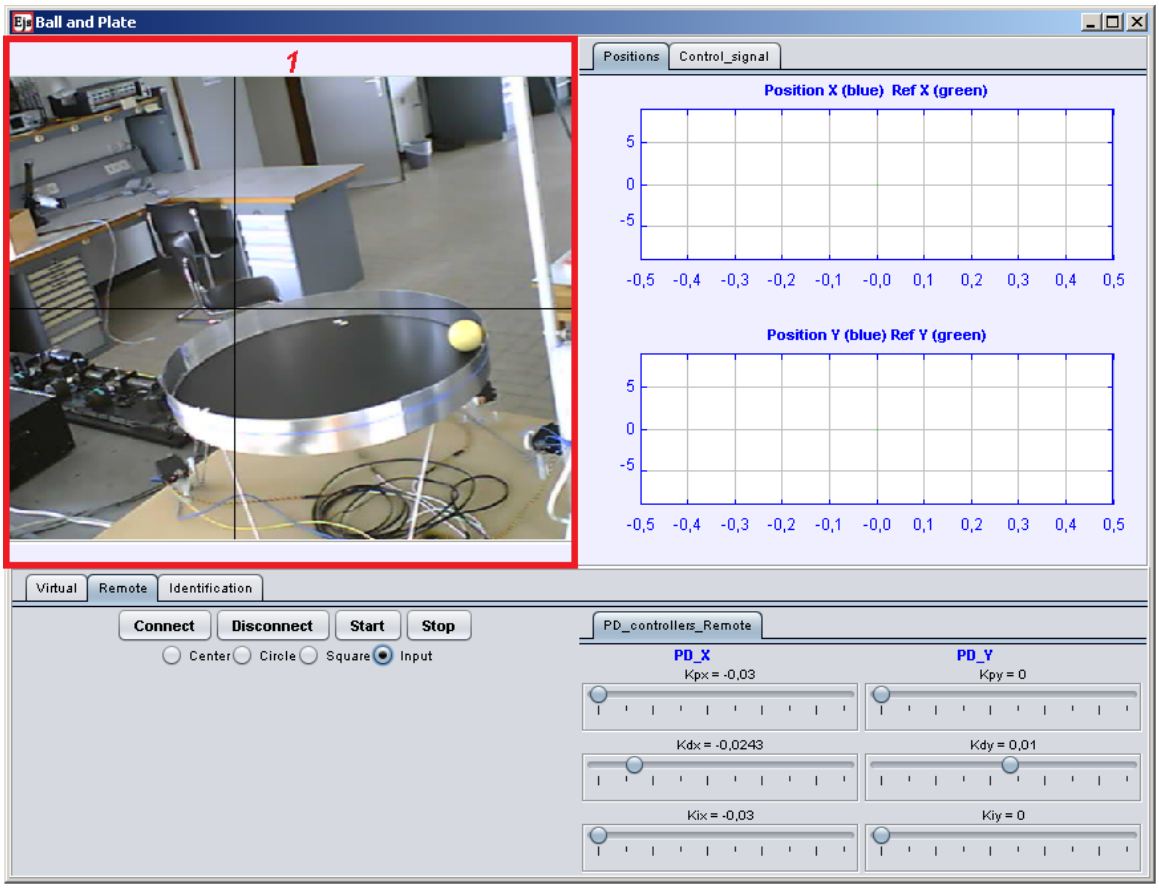


Figura 4.6. Experimento de control de posición de la bola en modo remoto.

La Figura 4.7 muestra los datos de posición de la bola para el experimento de control de posición de la bola en un punto del plato. Como se puede apreciar la referencia de posición de la bola es el punto $(0; 0)$, que se representa con la cruz de color rojo en el centro de la imagen. Mientas que la posición de la bola se representa con los puntos de color azul. Como se observa la bola parte aproximadamente desde la posición $(190; -175)$ y termina en una posición cercana al punto $(0; 0)$. Se aprecia que existe un pequeño error en estado estacionario que no resulta importante debido a que los valores están medidos en milímetros.

4.2.3 Control de seguimiento de trayectorias

En esta sección se muestran los resultados obtenidos para los experimentos de seguimiento de trayectorias. Estos experimentos tienen implícito el experimento anterior de control de posición de la bola en un punto. Pero en este caso las trayectorias que se

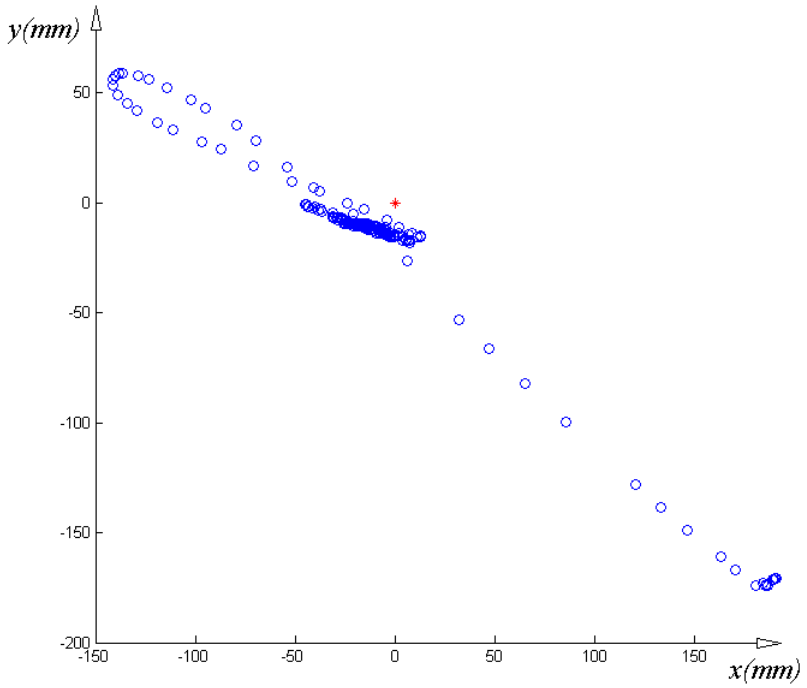


Figura 4.7. Posición de la bola para la planta real.

usan como referencia se separan en puntos y se envían de forma dinámica a la planta.

4.2.3.1 Trayectoria Circular en modo simulación

En el caso del modo de funcionamiento simulación, este experimento se inicia cuando el estudiante presiona el botón *Play* y selecciona la trayectoria *Circular* en la pestaña *Virtual* de la ventana principal de la aplicación *Cliente BP-C* (Panel No. 2). En la pestaña *Trajectories/Circular* (Panel No. 3), el estudiante puede variar las propiedades geométricas de la trayectoria seleccionada (*Radio del círculo*, *Frecuencia*, *Duración de la trayectoria* y *Ángulo de rotación de la trayectoria seleccionada sobre el plato*). Así como los parámetros de los controladores incluidos en los bucles de control.

La Figura 4.8 muestra el funcionamiento de la plataforma para este experimento en modo simulación. En la parte derecha de la ventana principal (Panel No. 1), los gráficos muestran las referencias (color verde) y las coordenadas X e Y de la posición de la bola (color azul). Como se puede apreciar ambas tienen una forma sinusoidal con un desfase aproximado de 90° entre ambas coordenadas, lo que indica que realmente la bola sigue una trayectoria circular.

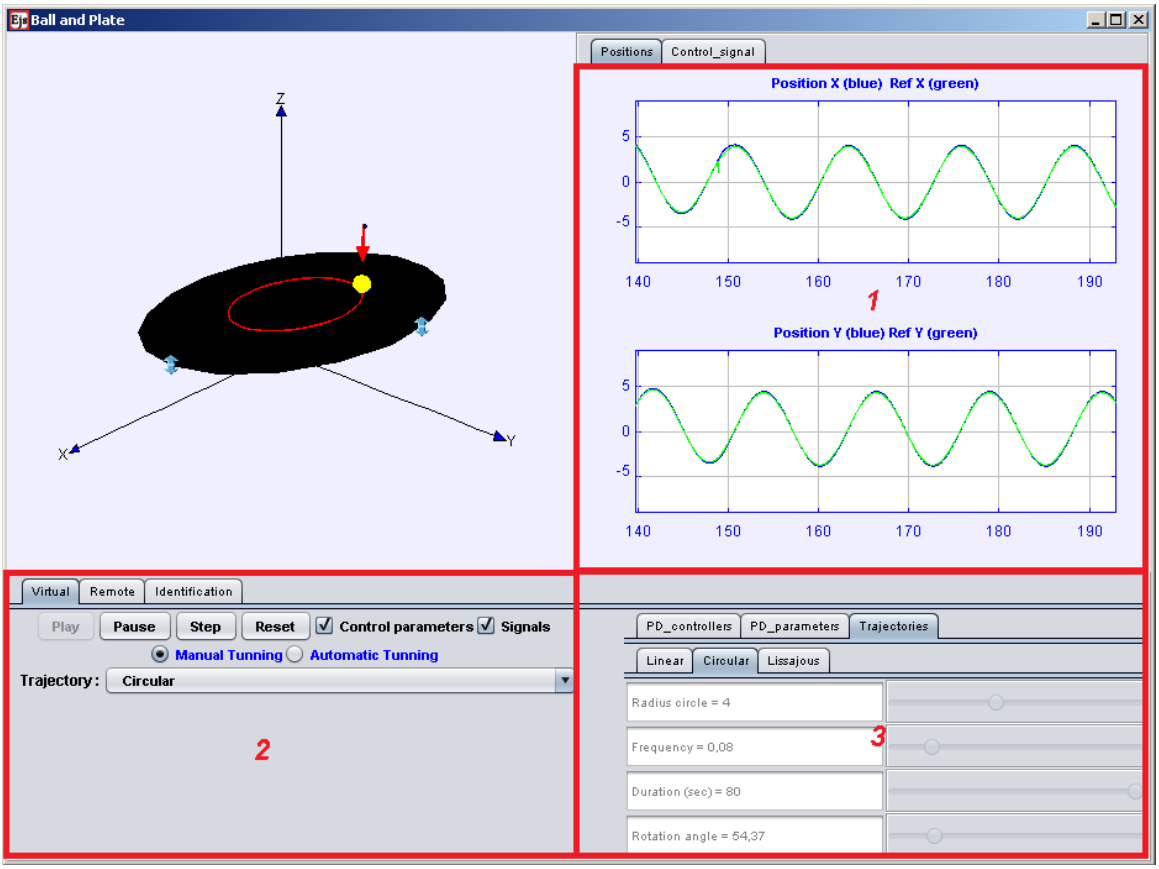


Figura 4.8. Trayectoria circular en modo local.

4.2.3.2 Trayectoria Circular en modo remoto

En el caso de este mismo experimento para el funcionamiento en modo remoto, el experimento comienza cuando el estudiante selecciona este modo, para conectar con la planta real y escoge la trayectoria *Circle* en la pestaña *Remote*. La Figura 4.9 muestra los datos de la planta real para este experimento.

Al igual que en el caso anterior la referencia se muestra con cruces de color rojo, mientras que la posición de la bola se representa con puntos de color azul. En este caso no se permite al estudiante modificar las propiedades geométricas de la trayectoria para evitar ocasionar daños en los motores de la planta real debido al cálculo de una señal de control errónea. Como se aprecia también existe un error en estado estacionario debido a la zona muerta de los motores, al retardo introducido en el lazo de control por el tratamiento de las imágenes de la cámara y a la comunicación de la aplicación

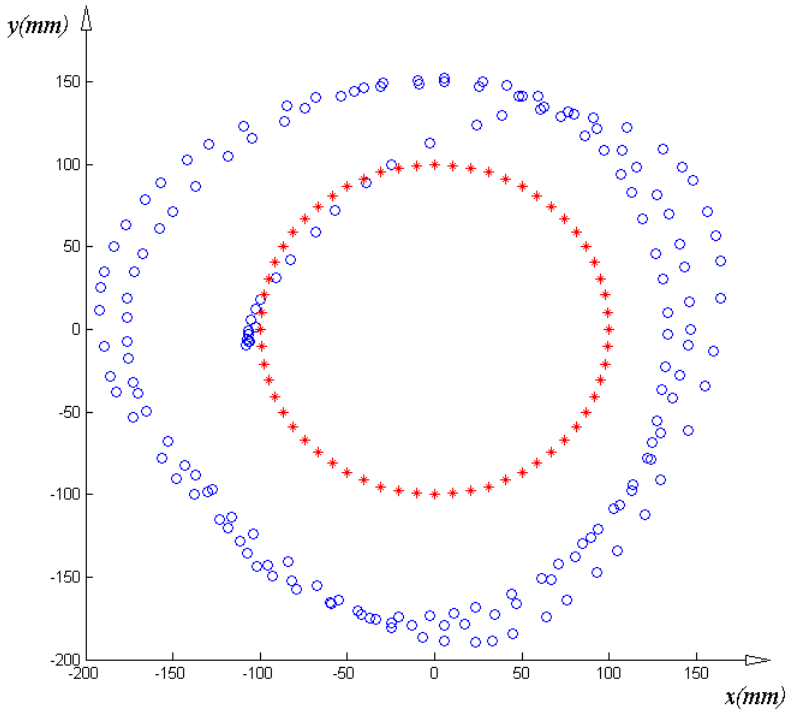


Figura 4.9. Trayectoria circular en modo remoto.

local (servidor) con la aplicación remota (cliente). Variando los parámetros de los controladores en ambos ejes el error se puede disminuir considerablemente. Esta es una de las tareas asignadas a los estudiantes durante el desarrollo de este experimento.

4.2.3.3 Otras Trayectorias

Tanto para el caso del funcionamiento en modo local como remoto, la plataforma implementa otras trayectorias. Por ejemplo, para el modo de funcionamiento local: una línea y una trayectoria de Lissajous simple. Mientras que para el modo remoto: una trayectoria cuadrada. El funcionamiento de la plataforma para estas trayectorias se describe a continuación.

Trayectoria recta en modo simulación

En el caso del funcionamiento en modo simulación para la trayectoria recta, el experimento se inicia cuando el estudiante presiona el botón *Play* y selecciona la opción *Linear* en el menú desplegable *Trajectories* (Panel No. 2) de la Figura 4.10.

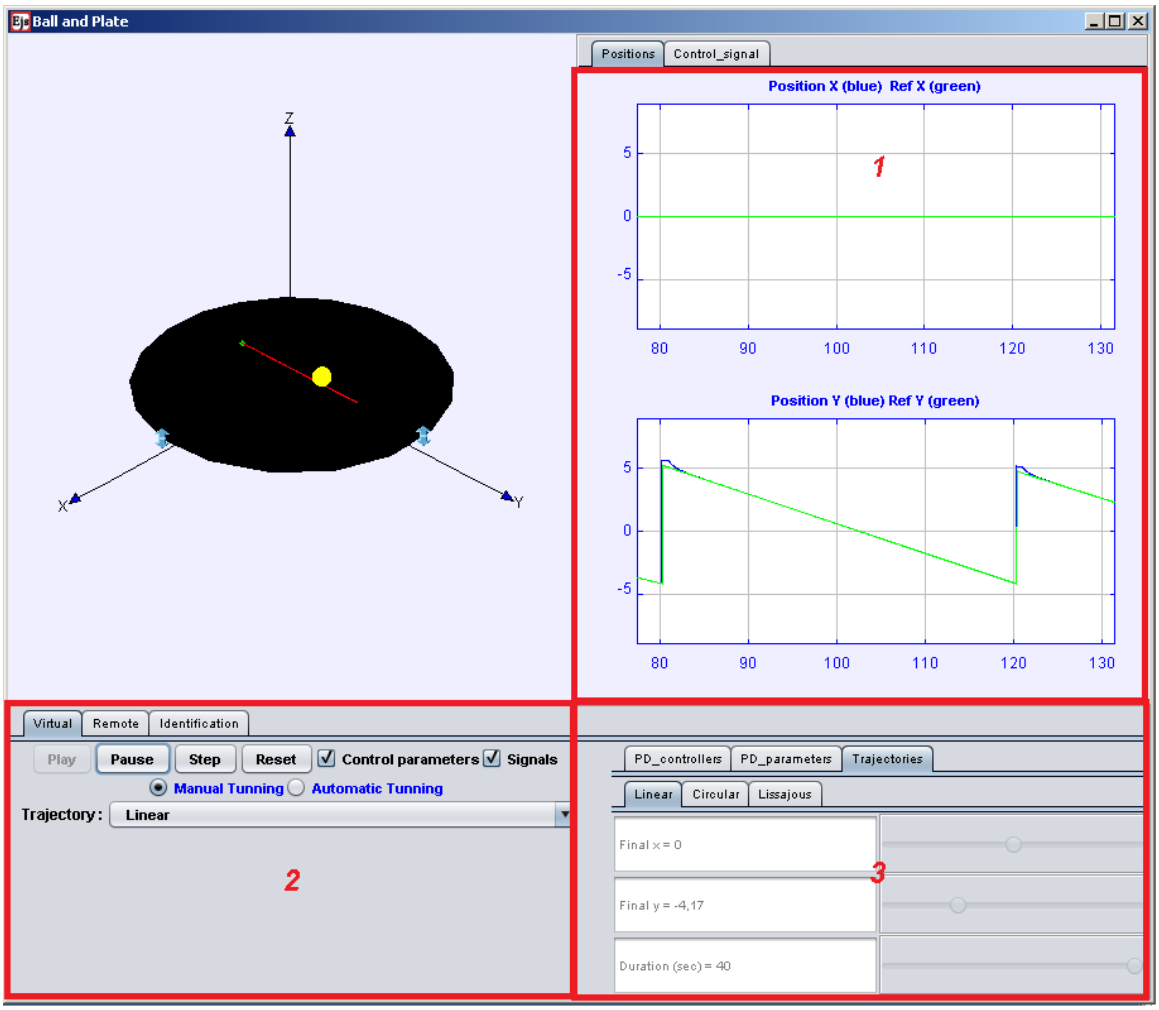


Figura 4.10. Trayectoria lineal en modo simulación.

Al igual que en los casos anteriores en modo simulación el estudiante puede modificar las propiedades de la trayectoria recta en la pestaña *Trajectories/Linear* (Punto final y duración de la misma) (Panel No. 3). Al seleccionar un punto, la línea se establece desde la posición actual de la bola (punto inicial de la trayectoria) hasta la posición indicada por el usuario (punto final de la trayectoria). Mientras que el tiempo representa la medida de la rapidez con que este punto se alcanza siguiendo esta trayectoria. Además el estudiante puede modificar la posición actual de la bola (arrastrando y soltado) a modo de perturbación para el sistema. En este caso los gráficos muestran que la coordenada X de la posición de la bola permanece constante mientras que la coordenada Y de la posición de la bola varía (Panel No. 1). Esto indica que la trayectoria recta seleccionada es sobre el eje Y del plato.

Trayectoria de *Lissajous* en modo simulación

La Figura 4.11 muestra el funcionamiento de la plataforma del experimento de control de seguimiento de una trayectoria de *Lissajous* en modo simulación.

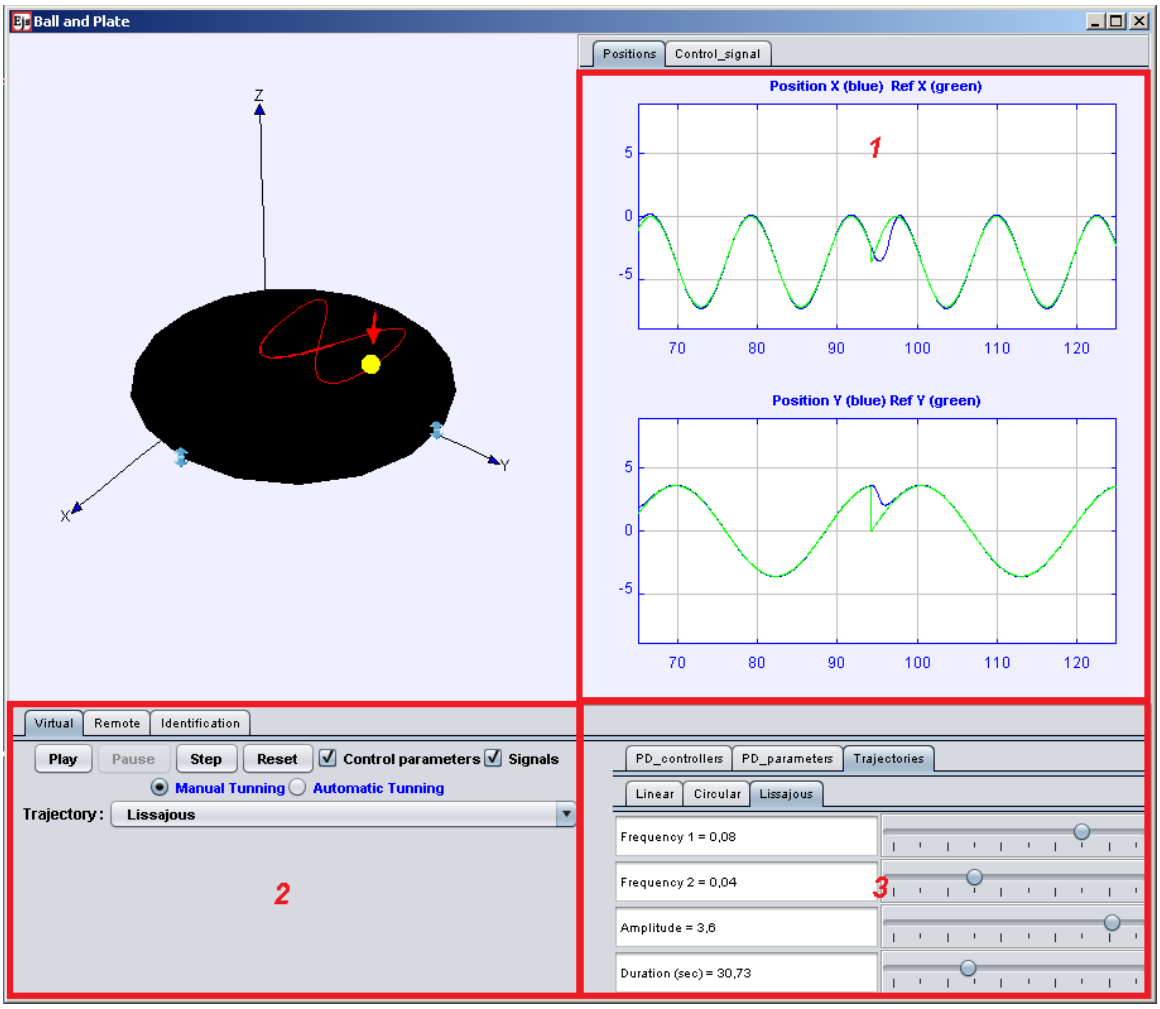


Figura 4.11. Trayectoria de Lissajous en modo simulación.

Este experimento se inicia cuando el estudiante selecciona la opción *Lissajous* del menú desplegable *Trajectories* en la pestaña *Virtual* (Panel No. 2). Al igual que en los casos anteriores en modo simulación, el estudiante puede modificar las propiedades geométricas de dicha trayectoria en la pestaña *Trajectories/Lissajous* (Panel No. 3). Las propiedades de la trayectoria que se pueden modificar son las siguientes: *Frecuencia* de ambas señales, *Amplitud* de ambas señales y *Duración* del experimento. Los gráficos muestran las referencias y las coordenadas X e Y de la posición de la bola sobre el

plato (Panel No. 1).

Trayectoria cuadrada en modo remoto

En el caso del funcionamiento de la plataforma en modo remoto se implementa también el control de seguimiento de una trayectoria cuadrada. Este experimento se inicia cuando el estudiante selecciona *Square* en la pestaña *Remote* y presiona el botón *Start*. La Figura 4.12 muestra los datos de la planta real para este experimento. Al igual que en los casos anteriores la trayectoria de referencia se representa con cruces de color rojo, mientras que la posición de la bola se representa con puntos de color azul. Como se puede apreciar el experimento se inicia con la bola en el centro del plato, lo que indica que antes de comenzar el experimento se controla la bola en el centro del plato.

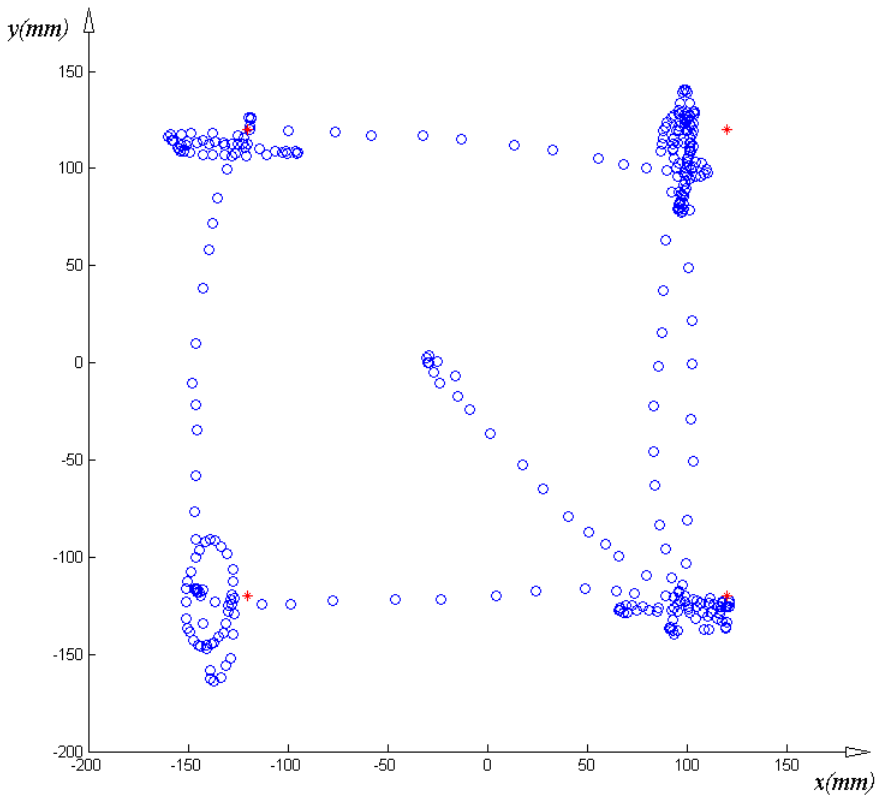


Figura 4.12. Datos de la planta real para la trayectoria cuadrada.

Al iniciar el experimento de seguimiento de una trayectoria cuadrada, se envían los puntos de referencia al bucle de control introduciendo un tiempo de espera entre envíos.

Este retardo se introduce para dar tiempo a que la bola se estabilice en el punto recién enviado. Una vez estabilizada la posición, se envía un nuevo punto. Como se aprecia la bola recorre los cuatro puntos asignados como referencia.

4.2.4 Análisis de los resultados obtenidos

De manera general se puede afirmar que el funcionamiento de la plataforma tanto para el modo local como para el modo remoto es bueno. En el caso de la simulación el hecho de que la animación sea en *3D* y totalmente interactiva, despierta en los estudiantes un mayor interés a la hora de usar la plataforma. Además al ser una simulación permite al estudiante modificar algunos de los parámetros del bucle de control sin ocasionar daños en la planta real. Los resultados obtenidos para los experimentos de control de posición como para los de seguimiento de trayectorias pueden catalogarse de favorables. El hecho de que sea una simulación en la que solo se utilizan componentes de *software*, brinda mucha flexibilidad en cuanto a implementación de experiencias de control. Debido a las potencialidades que brinda el *EJS* como herramienta de desarrollo de simulaciones con elevadas prestaciones visuales e interactivas. También este tipo de aplicaciones permiten al estudiante comprobar sus soluciones de una manera rápida y directa para detectar posibles fallos de forma inmediata en los parámetros de control seleccionados.

En el caso del funcionamiento de la plataforma en modo remoto, el hecho de tratarse de *hardware* real supuso una complejidad adicional. También influyó en gran medida en el bucle de control el tratamiento de imágenes que hace la aplicación servidor para obtener la posición de la bola sobre el plato. Al incorporar a esta aplicación la comunicación remota a través de *Internet* usando el protocolo *TCP*, la aplicación también se vio afectada en cuanto a su funcionamiento. Esto se aprecia con mayor énfasis en el seguimiento de la trayectoria circular ya que el establecimiento de la referencia (los puntos que forman la trayectoria), se realiza de forma continuada y al incorporar la comunicación remota, esto incluyó un retardo adicional en el bucle de control que afectó el funcionamiento de la aplicación que controla la planta. Debido a esto, se decidió almacenar los datos de la planta en el servidor en lugar de enviarlos a la aplicación cliente

como en casos anteriores, para evitar comportamientos no deseados. En cambio, en el caso del seguimiento de la trayectoria cuadrada, el envío de los puntos de referencia se realiza más lentamente (se establece el punto de referencia y se"espera" a que la bola se estabilice en ese punto). Por este motivo, la comunicación remota no supuso anomalías en el funcionamiento del lazo de control.

Finalmente se debe indicar que la obtención de buenos resultados para los experimentos dependen en gran medida de una adecuada calibración inicial que debe llevarse a cabo antes de realizar cualquier experiencia. De no realizar una buena calibración, en ocasiones se pueden obtener errores de estado estacionario considerables aunque la plataforma funcione correctamente. No obstante estos inconvenientes, los resultados obtenidos con la planta real se pueden catalogar de aceptables considerando la complejidad que supone el trabajo con esta planta.

4.3 Experimentos con la plataforma de los robots *Moway*

En esta sección se describen los resultados obtenidos para los experimentos realizados con la plataforma implementada con los robots *Moway*. En primer lugar se muestran los resultados obtenidos para el caso del control de posición de un solo robot y posteriormente el control de formación incluyendo varios robots.

4.3.1 Control de posición de robots móviles en modo simulación

En modo de trabajo simulación la plataforma implementa un simulador de control de posición de robots móviles. La Figura 4.13 muestra el funcionamiento de la plataforma para el experimento de control de posición de un robot. El Panel No. 1 muestra el espacio de trabajo donde se realizan los experimentos. En el espacio de trabajo se indica con una cruz de color rojo el punto de destino del robot. Mientras que el robot se representa con una imagen de un *Moway* de color azul. En color gris se muestra la traza que deja el robot a su paso para representar la trayectoria seguida hasta el punto de destino. El Panel No. 2 muestra los gráficos correspondientes al experimento en cuestión. El primero muestra el error de distancia al punto de destino mientras que

el segundo muestra la velocidad del robot.

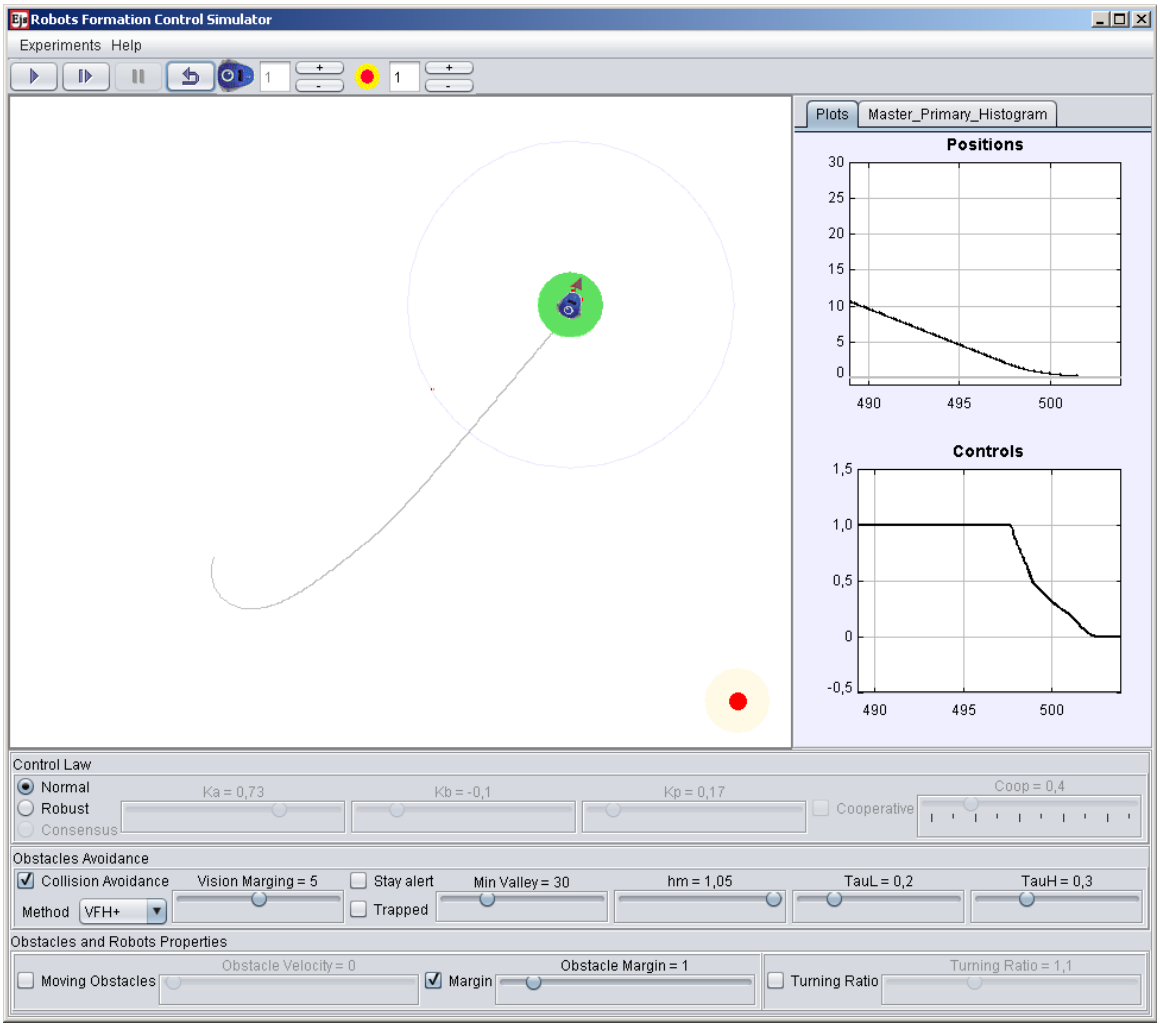


Figura 4.13. Control de posición de un robot móvil.

Como se puede apreciar en el primer gráfico, el error de posición del robot (que coincide con la distancia al punto de destino) disminuye. Esto indica que efectivamente el robot se acerca al punto final. Cuando este error se hace cero, indica que el robot está sobre el punto de referencia. Por su parte, el segundo gráfico muestra la velocidad lineal del robot. Como se puede apreciar cuando el robot está alejado del punto de destino, la velocidad lineal permanece en su valor máximo (saturación de la señal de voltaje que admiten los robots) ya que el bucle de control de la velocidad lineal es un control proporcional con saturación que depende de la distancia. Mientras que al acercarse a una distancia prudencial del punto de destino la velocidad entra en la zona

lineal y comienza a disminuir hasta hacerse cero, lo que implica que el robot se detenta cuando está sobre el punto de destino.

A este experimento se pueden añadir obstáculos para demostrar el algoritmo control de posición del robot móvil con evasión de obstáculos. La Figura 4.14 muestra una configuración típica para este tipo de experimentos con nueve obstáculos. El Panel No. 1 representa el espacio de trabajo en el cuál, con círculos de color rojo se muestran los obstáculos con sus respectivos márgenes de seguridad en color amarillo. La cruz de color roja muestra el punto de destino del robot y la línea recta de color negro muestra la trayectoria recta desde la posición actual del robot hasta el punto de destino. Al igual que en el caso anterior la trayectoria seguida por el robot se muestra en color negro.

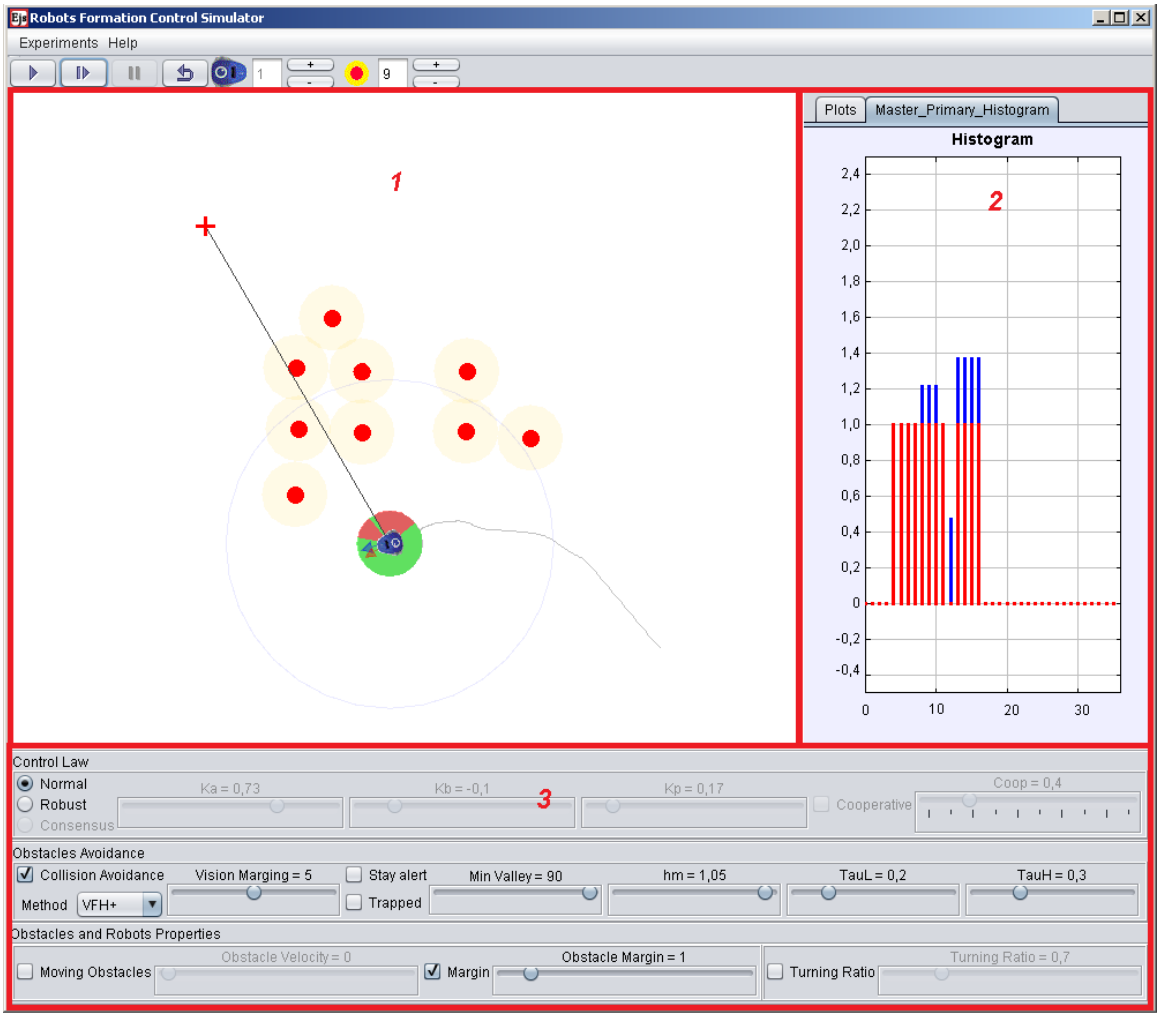


Figura 4.14. Control de posición de un robot móvil con evasión de obstáculos.

La circunferencia de color gris muestra el campo de visión del robot. El campo de visión es el área que el robot "ve". Los obstáculos que se encuentan dentro de esta área, son los que se tienen en cuenta en el algoritmo para evitarlos. El Panel No. 2 muestra el histograma polar primario (color azul) y el histograma binario enmascarado (color rojo). Estos histogramas se construyen en cada paso de simulación y como los sectores ciruclares están divididos en ángulos de 10°, para los 360° alrededor del robot hay 36 sectores circulares representados en los histogramas. Sus valores se calculan como una proporción inversa de la distancia a la que se encuentra cada obstáculo del robot. Lo que indica que un valor grande implica que el obstáculo está muy cerca del robot.

Los valores cero del histograma indican que en esos sectores no hay obstáculos, por lo que se consideran como sectores libres, por los que el robot puede pasar. Como se aprecia en el Panel No. 2, para la configuración del ejemplo considerado, los sectores ocupados van desde el sector 5 (50°) hasta el 17 (170°). A su vez el círculo que rodea al robot representa el histograma enmascarado actual del robot con los sectores libres (color verde) y los sectores ocupados (color rojo). El círculo alrededor del robot describe el histograma binario enmascarado en sentido contrario a las agujas del reloj comenzando los cero grados en el eje positivo de las X. El Panel No. 3 permite al estudiante modificar los parámetros de la simulación relacionados con la ley de control de los robots, el algoritmo de evasión de obstáculos y las propiedades físicas y geométricas de los obstáculos.

4.3.2 Control de posición de robots móviles en modo remoto

En el caso del funcionamiento de la plataforma en modo remoto para este experimento, la aplicación cliente se conecta al servidor donde se encuentran ejecutándose las aplicaciones que facilitan la comunicación con los robots. Cuando se establece la comunicación, la animación de la simulación se sustituye por la señal de vídeo que se recibe de la cámara *IP* instalada en el techo del laboratorio. Al hacer click sobre el vídeo que proporciona la cámara, se le envía al robot las coordenadas del punto representado con el círculo rojo, el cuál sirve como referencia de posición para dicho robot; análogamente a como se hace en el modo de trabajo local con el simulador.

Figura 4.15. Control de posición de un robot móvil en modo remoto.

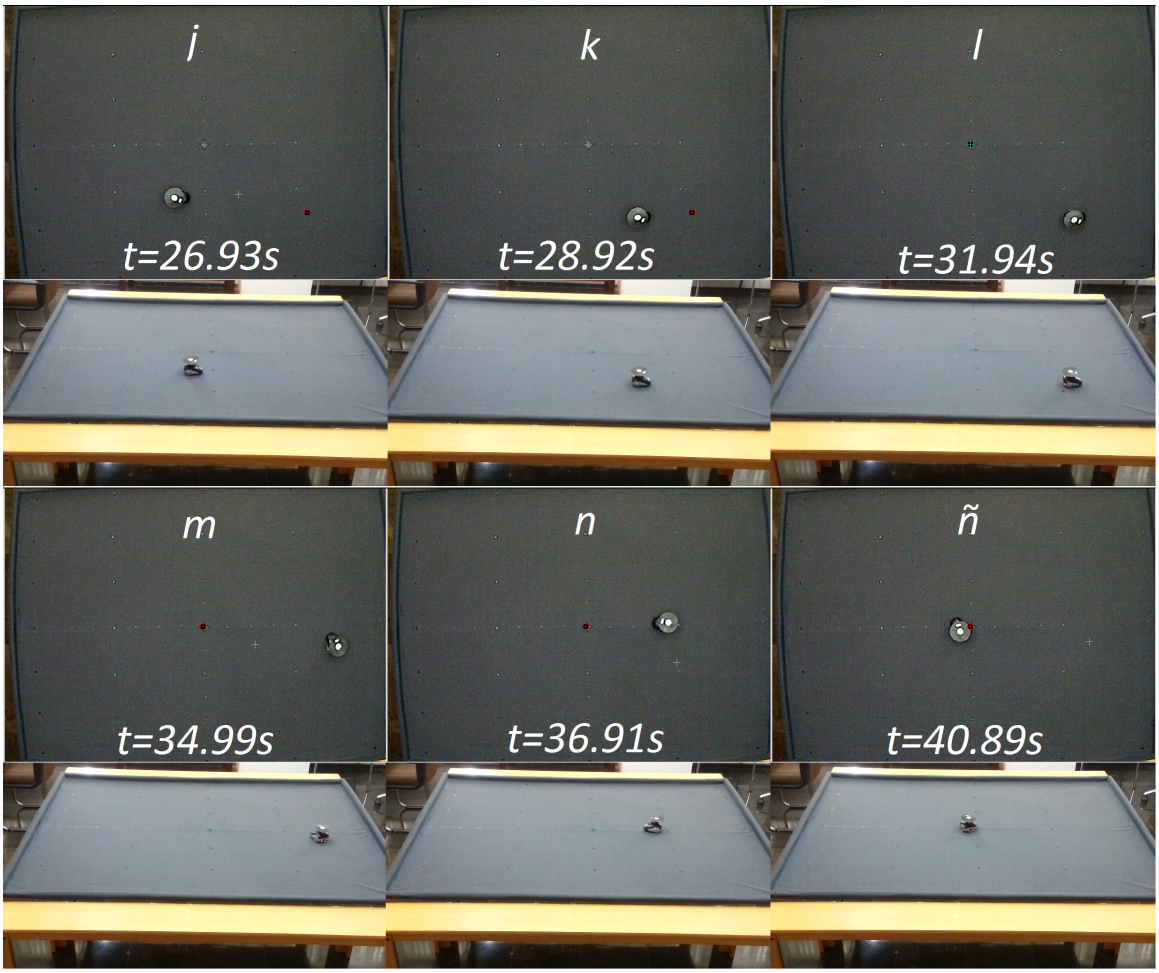


Figura 4.16. Control de posición de un robot móvil en modo remoto.

Las Figuras 4.15 y 4.16 muestran una secuencia de imágenes de lo que sucede cuando el estudiante hace click sobre el vídeo. La secuencia se compone de una matriz de imágenes, el la cual las filas impares de imágenes muestran lo que ve el estudiante y en las filas pares se observan las imágenes de un vídeo realizado para mostrar el desarrollo del experimento. En el instante inicial (t=0s) el robot se encuentra en la posición (0;0) (*imagen a*) al hacer click sobre el vídeo, se le envía el punto de referencia (57;37). Pasado un tiempo t=5.86s, el robot arriba al punto de destino (*imagen c*). Seguidamente se le envía el punto de referencia (-37;36) al que arriba pasado un tiempo t=16.78s (*imagen f*). A continuación se le envía el punto de referencia (-44;-31) al que llega pasados t=22.99s (*imagen i*). Seguidamente se cambia la referencia al punto (47;-32) al que llega pasado un tiempo t=31.94s (*imagen l*). Finalmente se repite el

experimento enviándole al robot la referencia del punto origen (0;0), al que llega pasado un tiempo t=40.89s.

La Figura 4.17 muestra los datos de posición del robot almacenados en el servidor para este experimento. La cruz de color rojo muestra los diferentes puntos de referencia que se le envían al robot durante el experimento y corresponden con las coordenadas del círculo rojo de la Figura 4.15 cada vez que se hace click sobre el vídeo. Mientras que los puntos de color azul representan las coordenadas de la posición del robot y las letras se corresponden con las letras de las secuencias de las Figuras 4.15 y 4.16. Como se puede apreciar cada vez que se cambia la referencia, el robot sigue una trayectoria suave desde el punto de inicio hasta el punto de destino y permanece finalmente sobre éste. Inicialmente el robot se mueve a su máxima velocidad lineal posible y cuando se acerca al punto de destino comienza a disminuir su velocidad hasta que alcanza el punto y se detiene.

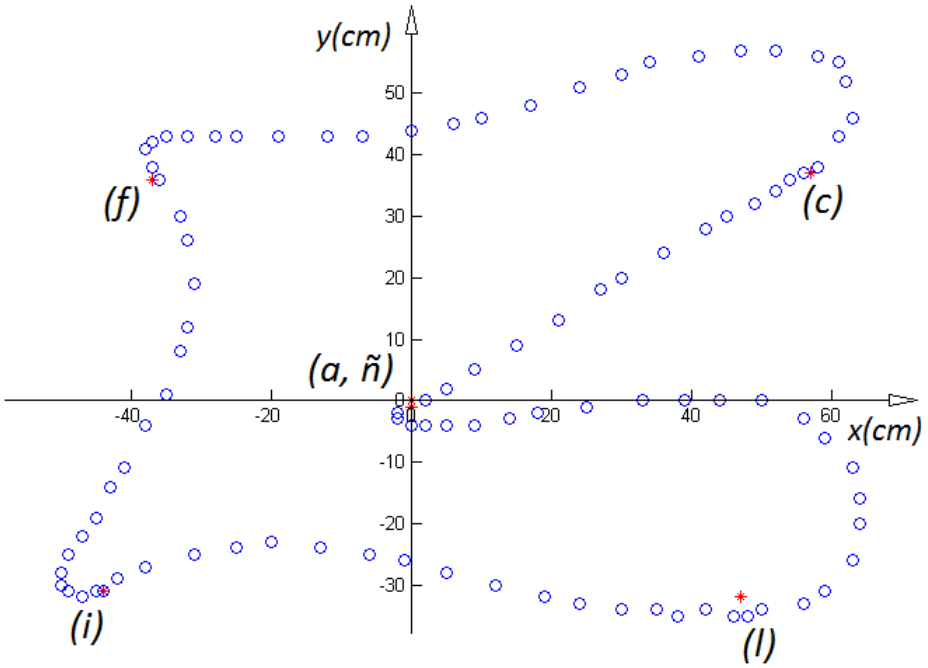


Figura 4.17. Datos para el experimento de control de posición de un robot en modo remoto.

Para demostrar el funcionamiento del algoritmo de evasión de obstáculos implementado, se selecciona la configuración de dos obstáculos mostrada en la Figura 4.18. Al igual que en los casos anteriores, en la parte superior de la figura se muestra lo que

ve el estudiante y en la parte inferior una imagen que para mostrar el desarrollo del experimento.

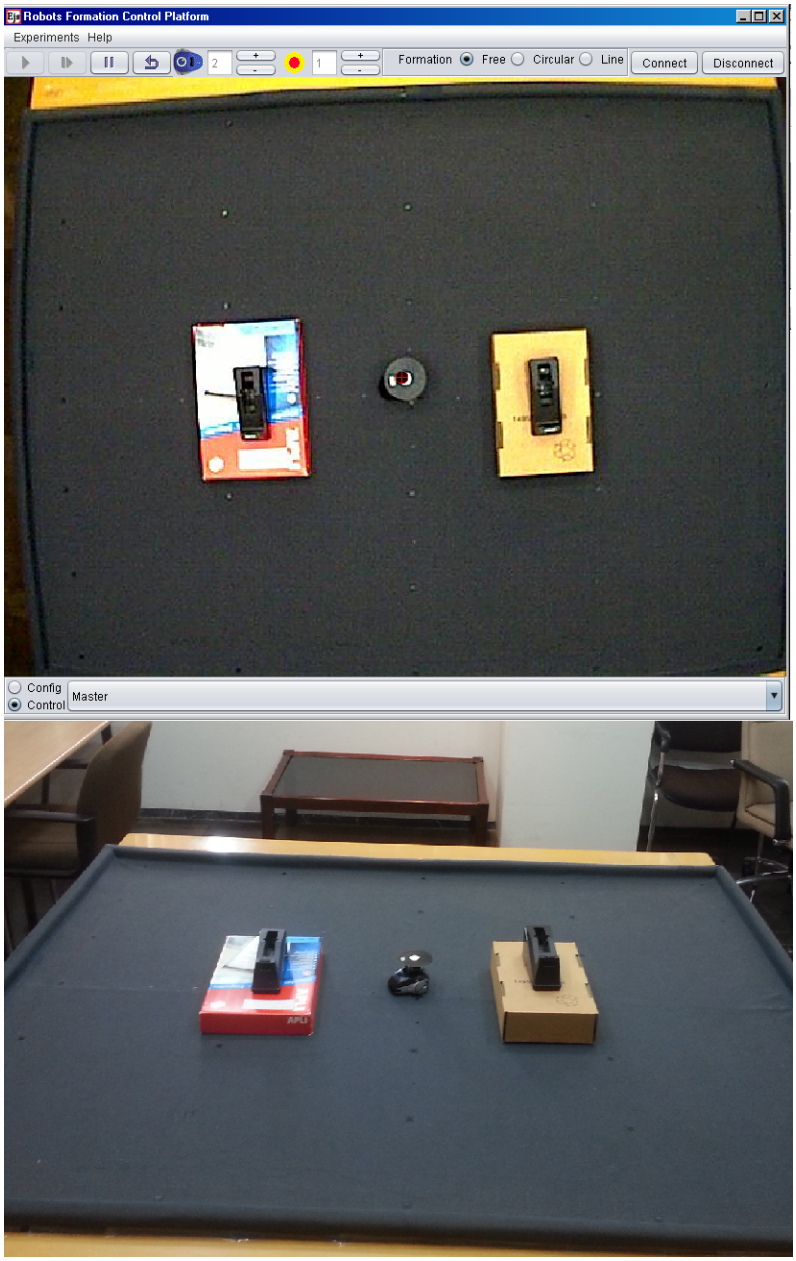


Figura 4.18. Configuración típica con dos obstáculos en modo remoto.

El experimento consiste en enviarle al robot varios puntos de referencia, de forma similar al experimento anterior, para observar su comportamiento en presencia de los obstáculos. La Figura 4.19 muestra los datos obtenidos para este experimento. Las cruces de color rojo muestran los puntos de referencia enviados al robot. Mientras que

los puntos de color azul muestran la posición del robot.

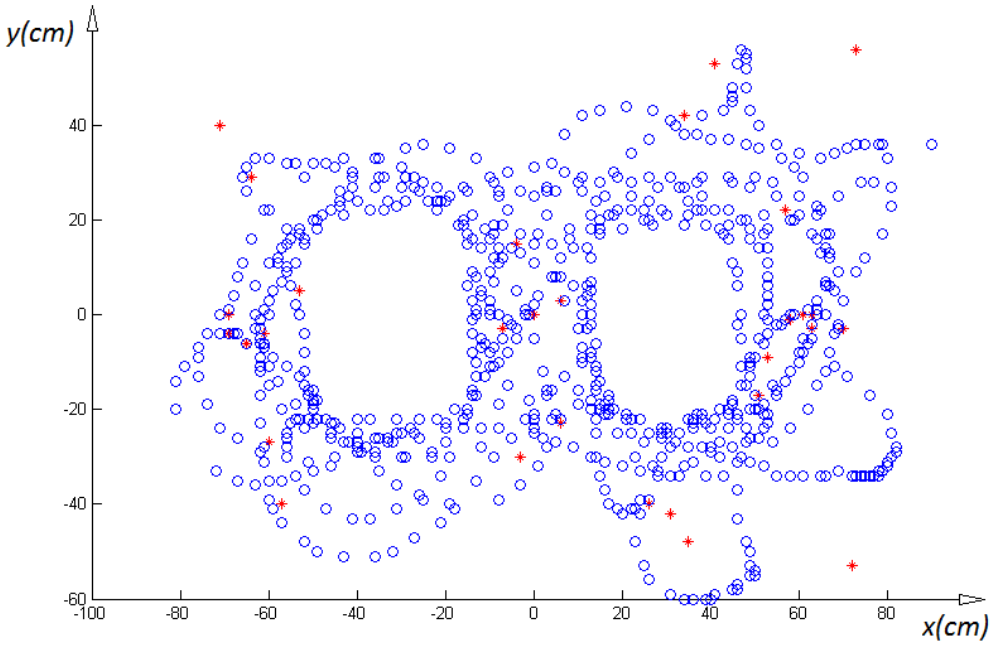


Figura 4.19. Resultados del algoritmo de evasión de obstáculos en modo remoto.

Como se puede apreciar, el robot intenta alcanzar los diferentes puntos de referencia suministrados evitando los obstáculos presentes en el espacio de trabajo.

4.3.3 Experimento de control de formación de tipo maestro-esclavos en modo simulación

Para realizar este tipo de experimentos con más de un robot (maestro-esclavos), el estudiante debe agregar más robots al espacio de trabajo y escoger el tipo de formación que se desea realizar. La Figura 4.20 muestra el desarrollo de un experimento de este tipo en el que se seleccionan un robot maestro y cinco robots esclavos.

Como se puede apreciar a través de las trazas de los robots, éstos inician el experimento desde posiciones aleatorias arbitrariamente seleccionadas por medio de arrastrar y soltar. Durante el desarrollo del experimento el robot maestro trata de alcanzar su punto de destino. Mientras que los robots esclavos tratan de alcanzar y mantener su posición en la formación, en este caso un círculo con respecto al maestro. La gráfica de las posiciones muestra la distancia al punto de destino para cada uno de los robots (el color de las líneas corresponde con el color del círculo que rodea a cada robot para

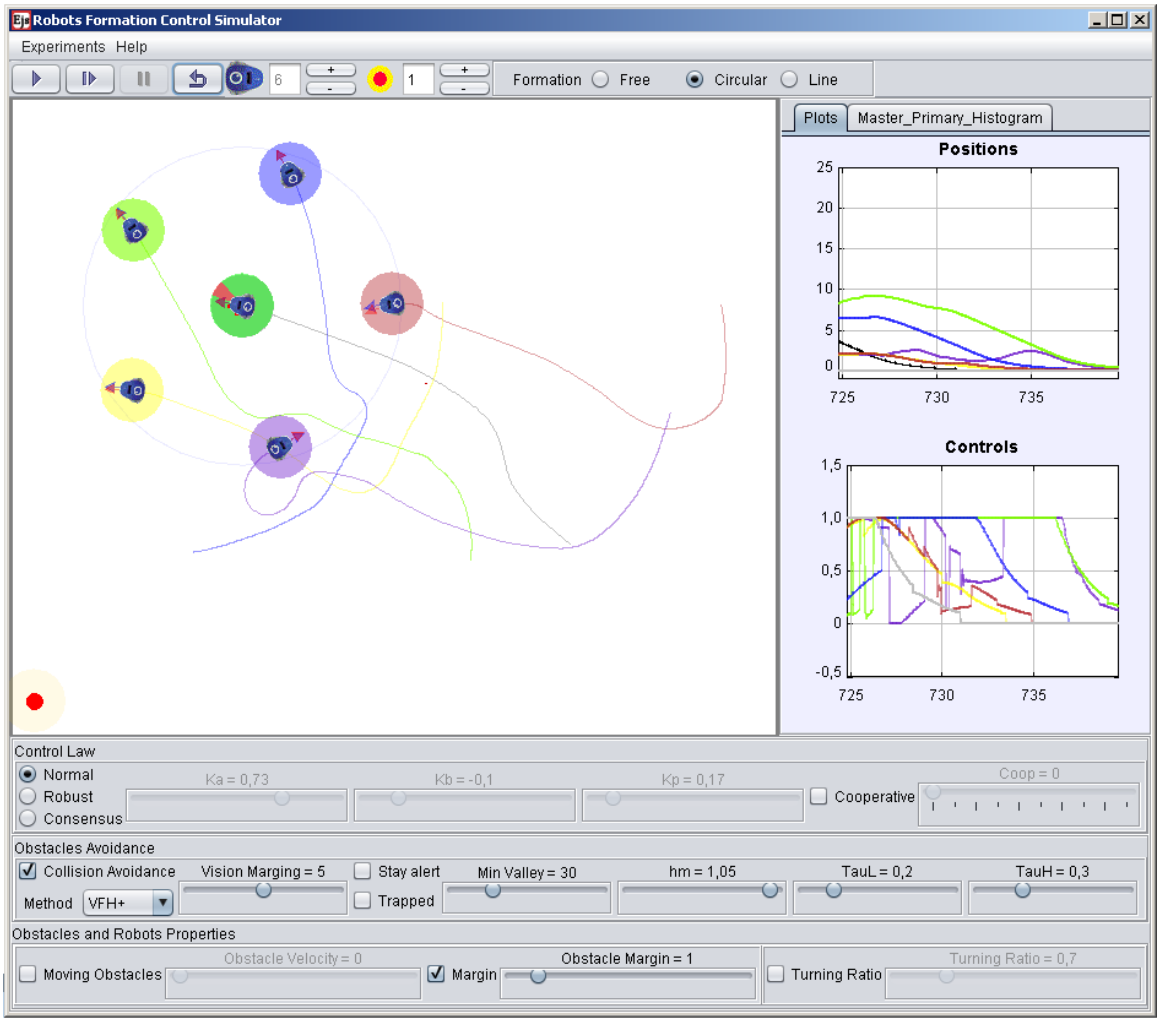


Figura 4.20. Control de formación de robots móviles.

hacer más fácil la identificación). Como se aprecia la distancia al punto de destino de cada robot disminuye en la medida en que cada uno de ellos se acerca a su posición en la formación. En cuanto a las velocidades, experimentan variaciones debido a los giros que los robots realizan. Al igual que las distancias al llegar los robots a su posición final en la formación, las velocidades se hacen cero, lo que indica que los robots finalmente se detienen al alcanzar su posición final. Durante el experimento los robots no se mantienen en formación ya que el robot maestro no tiene en cuenta los errores de posición de los esclavos en el control de su velocidad. Por lo que se puede decir que el control no es de tipo cooperativo. Los robots esclavos utilizan la posición del maestro para situarse en una posición entorno a este, dependiendo de la formación seleccionada.

Experimento de control cooperativo en modo simulación

Para llevar a cabo el control cooperativo es necesario que el robot maestro tenga en cuenta la posición de los robots esclavos en la formación. De forma tal que controle su velocidad en función de su propio error de posición (distancia hacia su punto de destino) más la suma del error de posición de los robots esclavos en la formación, que a su vez es el error de posición de los robots esclavos. Para realizar este experimento se multiplican ambos errores por un factor que es el que determina la importancia que tiene cada error en el control de la velocidad del robot maestro.

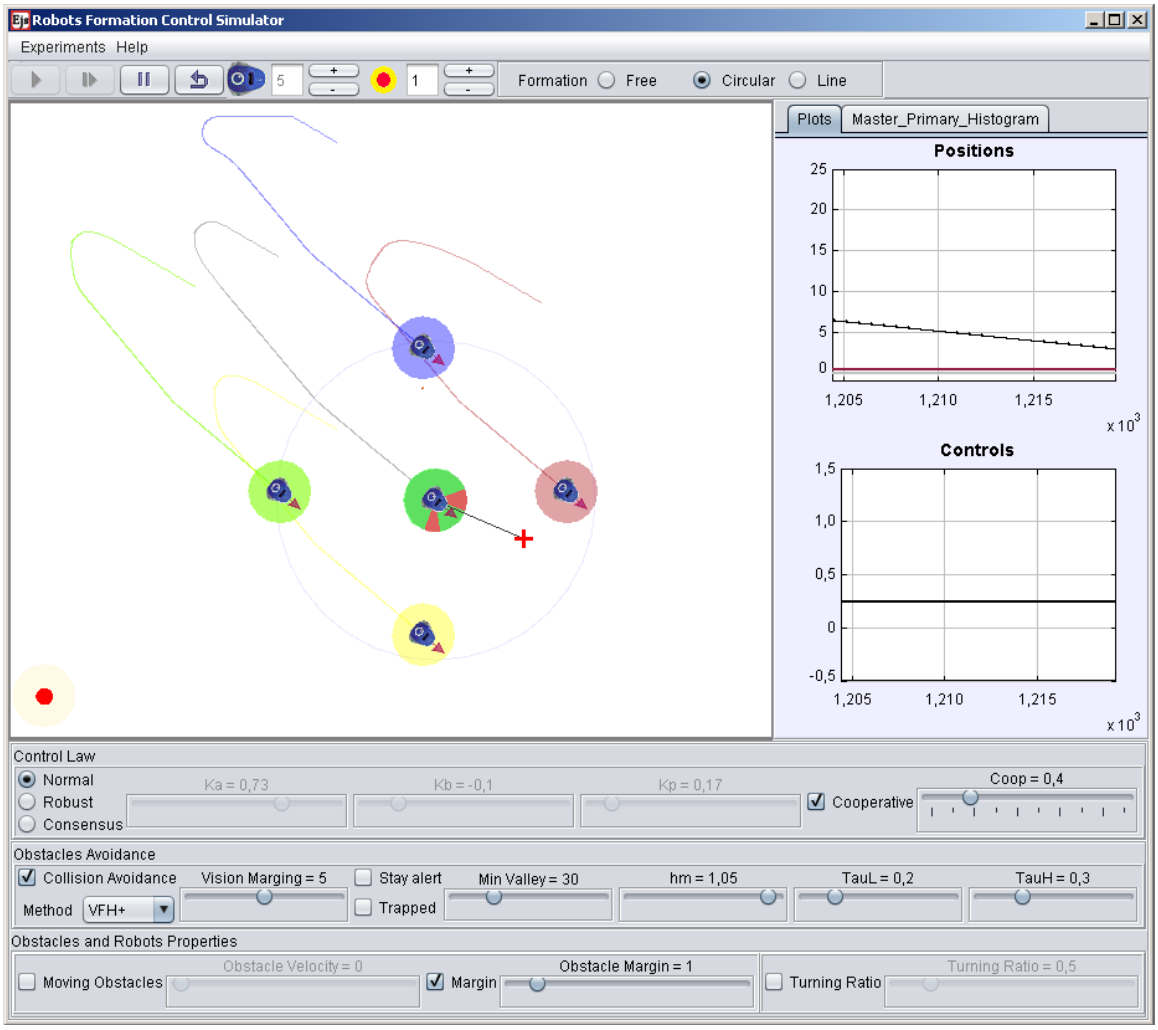


Figura 4.21. Control de formación de robots móviles.

La Figura 4.21 muestra un ejemplo de control de formación de tipo cooperativo. En este caso se han utilizado cuatro robots esclavos. Como se puede observar en la gráfica

de posiciones de los robots, la distancia del robot maestro (color negro) es la que mayor error presenta y va disminuyendo. Mientras el error de posición de los robots esclavos es casi cero. Esto indica que el robot maestro está alejado de su punto de destino, marcado por la cruz de color rojo en el espacio de trabajo. Los robots esclavos están en su punto de destino (su posición en la formación entorno al maestro). En la gráfica de los controles se aprecia que las velocidades de todos los robots son iguales y constantes. Esto indica que los robots esclavos se mueven a la misma velocidad que el maestro, ya que sus posiciones finales dependen de la posición del robot maestro. Si éste se mueve tratando de alcanzar su punto de destino, los robots esclavos lo siguen, lo que indica que los robots están coordinados y mantienen la formación.

Este experimento se puede realizar para cualquier otra formación que se seleccione. Lo que cambia de una formación a otra es la posición de los esclavos con respecto al líder. Para todas las formaciones implementadas en modo simulación los resultados son similares en todos los casos. Si durante el desarrollo del experimento se saca a alguno de los robots esclavos de la formación a modo de perturbación, el robot maestro se detiene y "espera" al robot rezagado ya que el error de la formación aumenta y esto hace disminuir considerablemente la velocidad del robot maestro.

4.3.4 Experimento de control de formación de tipo maestro-esclavos en modo remoto

En este experimento cuando la plataforma trabaja en modo remoto se conecta al servidor para enviar las órdenes a los robots *Moway*, como se explicó anteriormente en la Sección 3.4. La imagen de la animación se sustituye por las imágenes de la cámara *IP* fijada en el techo del laboratorio, como se explicó anteriormente. La Figura 4.22 muestra el funcionamiento de este experimento en una matriz de imágenes en las que se observa una secuencia de lo que ocurre desde el instante inicial *imagen a* hasta el instante final *imagen i*. De forma similar al experimento anterior en modo remoto las filas impares muestran las imágenes de un vídeo tomado para observar el experimento. Mientras que las filas pares muestran las imágenes de la cámara situada en el techo del laboratorio.

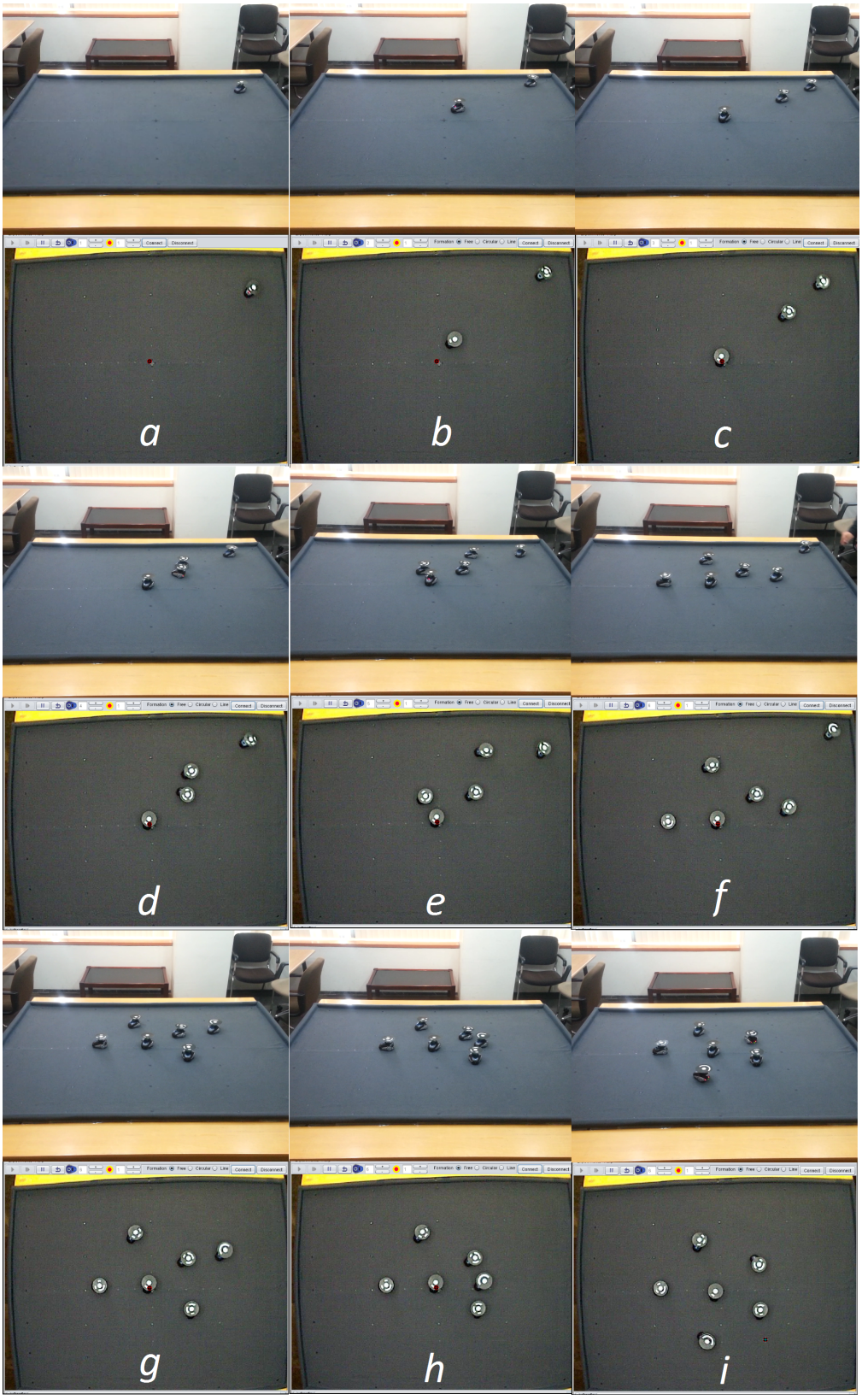


Figura 4.22. Control de formación de robots móviles en modo remoto.

El experimento consiste en fijar el punto de referencia de posición del robot maestro en el origen (0;0) y a continuación incluir de forma paulatina el resto de los robots. El objetivo es que al finalizar el experimento, los robots esclavos se posicionen formando un círculo respecto al robot maestro. Al iniciar el experimento el robot maestro inicia su movimiento hacia el origen como se puede apreciar en la *imagen a*. A continuación se comienzan a incluir el resto de los robots (*imagen b* hasta la *imagen f*). Como se puede apreciar, a medida que los robots se van incluyendo, éstos inician su movimiento hacia la formación final que alcanzan como se observa en la *imagen i*.

La Figura 4.23 muestra los datos de las posiciones de los robots para este experimento. Al igual que en los casos anteriores los puntos representan las posiciones de los robots (color negro para el robot maestro y resto de los colores para los robots esclavos). La cruz de color rojo representa la referencia del robot maestro, que como se observa se sitúa en el origen (0;0). Se ha representado además con líneas discontinuas la circunferencia aproximada que tienen que formar los robots esclavos entorno al robot maestro.

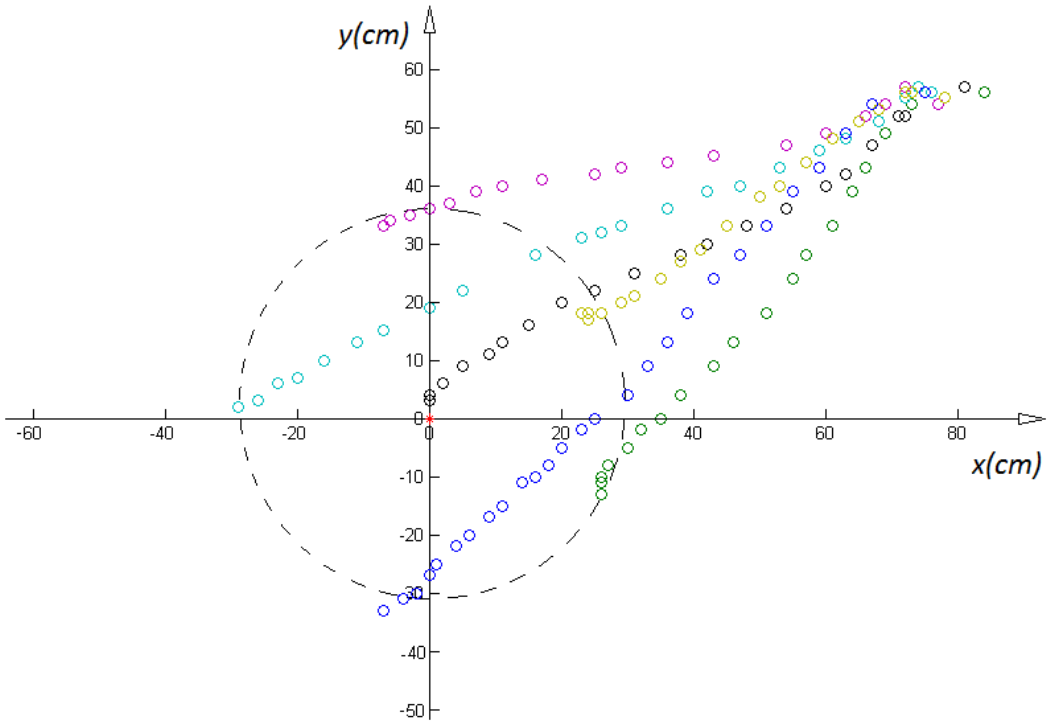


Figura 4.23. Control de formación de robots móviles en modo remoto.

Como se mencionó anteriormente los robots necesitan conocer su posición en cada instante para este tipo de experimentos. Dicha posición se les envía de forma inalámbrica por medio de paquetes de datos una vez que se han obtenido de la cámara las posiciones de todos los robots presentes en el experimento. El tiempo mínimo de refrescamiento de la posición de los robots se encuentra entorno a los $300ms$. Este tiempo influye en el control de posición como un retardo importante que se tiene que tener en cuenta al ajustar los parámetros de los controladores de los robots. Por otra parte, el tamaño de los paquetes de datos que pueden recibir los robots cada vez es de 8 *bytes*. Por esta razón resulta imposible enviar de una vez más datos al robot que su posición, su orientación y su referencia (usando 2 *bytes* para cada uno de estos datos, ya que incluyen datos negativos menores que -127, por lo que no se puede usar un solo byte). Una alternativa es enviar paquetes de datos más grandes, lo que implica enviar más paquetes de 8 *bytes* cada vez, pero esto requiere de más tiempo de envío e influye en el lazo de control. Otra opción es calcular la velocidad del robot en el ordenador que actúa como servidor, ya que en cada instante todos los datos de posición de todos los robots están disponibles; pero esto centraliza la operación del sistema perdiéndose de esta forma la propiedad de la plataforma de ser descentralizada (que los robots sean capaces de tomar sus propias decisiones a través del cálculo de su acción de control).

4.3.5 Análisis de los resultados obtenidos

Los resultados obtenidos con esta plataforma han sido satisfactorios, tanto para el funcionamiento en modo local como en modo remoto para todos los experimentos realizados: control de posición o estabilización de un robot en un punto; control de formación de robots de tipo maestro-esclavos. En el caso de la simulación, la animación totalmente interactiva del sistema resulta de interés para los estudiantes. La interacción con el simulador es muy dinámica ya que se permite introducir nuevos obstáculos y nuevos robots a los experimentos. Además, permite establecer manualmente la posición de los obstáculos y decidir que éstos sean estáticos o móviles. Y en el caso de los robots, permite establecer sus posiciones iniciales y el tipo de formación. Otra ventaja es que permite probar diferentes leyes de control e interactuar con sus parámetros

para observar el comportamiento de los robots, sin ocasionar daños físicos a éstos. Otro aspecto importante y que resulta muy ventajoso es que en todo momento se conocen las posiciones de todos los obstáculos y todos los robots que intervienen en un experimento, lo que facilita el funcionamiento del algoritmo de evasión de obstáculos implementado. En plataformas reales tanto la obtención de la posición de cada robot, como la detección de obstáculos y la comunicación entre los robots, constituyen un gran desafío. En este sentido el simulador tiene grandes ventajas sobre las plataformas reales.

En el caso del funcionamiento de la plataforma en modo remoto, los resultados obtenidos pueden catalogarse de aceptables. Para todos los experimentos la comunicación inalámbrica entre el servidor y los robots resulta de vital importancia. Esta comunicación se realiza a través de *RF* (Radiofrecuencia) en la banda de radio *ISM* ("*Industrial Scientific Medical*" a 2,4 GHz). Este tipo de comunicación es muy utilizada en comunicaciones inalámbricas. Su gran desventaja es que también se usa para las redes *Wifi*, *Bluetooth*, *Xbee*, etc. Por tanto puede sufrir interferencias de otras señales inalámbricas con el consecuente efecto sobre el experimento que se esté realizando en cada momento. Para disminuir las interferencias de otras señales a la plataforma se escogió un canal de comunicaciones a una frecuencia lo más alejada posible de los canales que usan las redes Wifi activas como por ejemplo *EDUROAM*. No obstante, se debe tener en cuenta que en todo momento pueden ocurrir pérdidas de datos debido a estas interferencias. Estas pérdidas de datos tienen una mayor influencia en el caso de los experimentos con varios robots, ya que los robots esclavos usan como referencia la posición del robot maestro. Por lo que si un robot esclavo se ve afectado por estas pérdidas la repercusión en la formación no será muy considerable. En cambio si es el robot maestro el que sufre estas pérdidas, la formación en su totalidad puede verse afectada.

Otro aspecto que influye directamente en el funcionamiento de la plataforma y que tiene que ver también con las comunicaciones y el procesamiento de la señal de vídeo de la cámara, es el tiempo mínimo de envío de paquetes de información a los robots. Después de numerosas pruebas realizadas se determinó que este tiempo se encuentra entorno a los $300ms$, lo que en comparación con la velocidad máxima que pueden

alcanzar los robots, es un retardo considerable. Este retardo se tiene en cuenta al calcular los parámetros de los controladores de los robots. Además al incluir varios robots, este tiempo puede aumentar, hecho que también se debe tener en cuenta en el diseño de los controladores. Otro factor que influye directamente en el funcionamiento de la plataforma es el tamaño del paquete de información que se le puede enviar a los robots cada vez. El tamaño máximo de dicho paquete es de 8 *bytes* lo que constituye una limitación importante de estos robots. Esta limitación impide enviar a los robots información acerca de la posición de todos los robots y de los obstáculos, lo que influye directamente en el control de formaciones, debido a que el robot maestro necesita conocer en todo momento la posición del resto de robots de la formación. Esto se podría eliminar si se calculara la velocidad del robot maestro en el ordenador que actúa como servidor, pero esto centralizaría el control y la plataforma perdería su esencia que es precisamente que sea descentralizada (que los robots tomen sus propias decisiones calculando su señal de control).

En el caso del algoritmo de evasión de obstáculos, la plataforma presenta limitaciones en modo remoto ya que todos los robots necesitan conocer en todo momento la posición de los robots restantes y de todos los obstáculos. Por este motivo se implementó un algoritmo de evasión de obstáculos diferente usando los 4 sensores de obstáculos que tienen los robots. Los resultados obtenidos son aceptables y los robots son capaces de detectar y esquivar tanto a los otros robots como los bordes del espacio de trabajo. Los sensores de detección de obstáculos que poseen los robots funcionan por ultrasonidos, enviando una señal que rebota en los obstáculos y es recibida nuevamente en el robot. El uso de estos sensores presenta dos limitaciones fundamentales: 1.- el número de sensores que posee el robot y su ubicación; 2.- el funcionamiento de dichos sensores.

Cada robot solo posee cuatro sensores de obstáculos ubicados en su parte frontal, lo que no permite detectar obstáculos que estén a su espalda. Debido al funcionamiento de estos sensores, la superficie de los obstáculos que pueden detectar tiene que ser uniforme y plana. Pero la forma de los robots no es plana por lo que en ocasiones cuando el obstáculo a detectar es otro robot, los sensores no funcionan correctamente.

Por estos dos motivos, la detección de obstáculos puede fallar en algunas situaciones que se dan durante el desarrollo de los experimentos.

4.4 Experimentos con la plataforma de los robots *Surveyor SRV-1*

En esta sección se describen los resultados obtenidos para los experimentos realizados con la plataforma implementada con los robots *Surveyor SRV-1*.

4.4.1 Experimento de control de formación de robots móviles de tipo líder-seguidores

Este experimento consiste en colocar un conjunto de robots a una distancia determinada unos de otros, formando una fila. El robot denominado líder debe estar posicionado al inicio de la fila. Para poner en funcionamiento el experimento, se puede hacer de dos formas: el líder puede seguir una línea de color dibujada en el suelo o puede ser manipulado remotamente, haciendo que éste se mueva según las órdenes del estudiante. En el caso de una trayectoria dibujada en el suelo, éstas pueden ser curvas ya que como se mencionó anteriormente ne la Sección 3.5, el control implementado en los robots también corrige su posición lateral. La Figura 4.24 muestra el experimento con un robot líder (R0) y un seguidor (R1).

Como se puede apreciar, el robot líder se encuentra siguiendo una línea roja dibujada en el suelo, mientras que el robot seguidor intenta mantener una distancia constante con el líder. Para mantener la distancia constante, el robot seguidor toma como referencia el área del rectángulo de color amarillo que se encuentra en la parte trasera del robot líder. Los tres gráficos de la *GUI* muestran los datos enviados desde el robot seguidor. El gráfico Y (Panel No. 1) muestra la altura del marcador rectangular de la parte trasera el líder expresada en píxeles. La altura del marcador está relacionada con la distancia con respecto al robot maestro. Mientras que el gráfico X (Panel No. 2) muestra la posición del marcador rectangular en el eje horizontal expresada en píxeles. La posición del marcador está relacionada con la alineación del robot seguidor con respecto al robot líder. El gráfico *Left Right* (Panel No. 3) muestra las órdenes enviadas a los motores

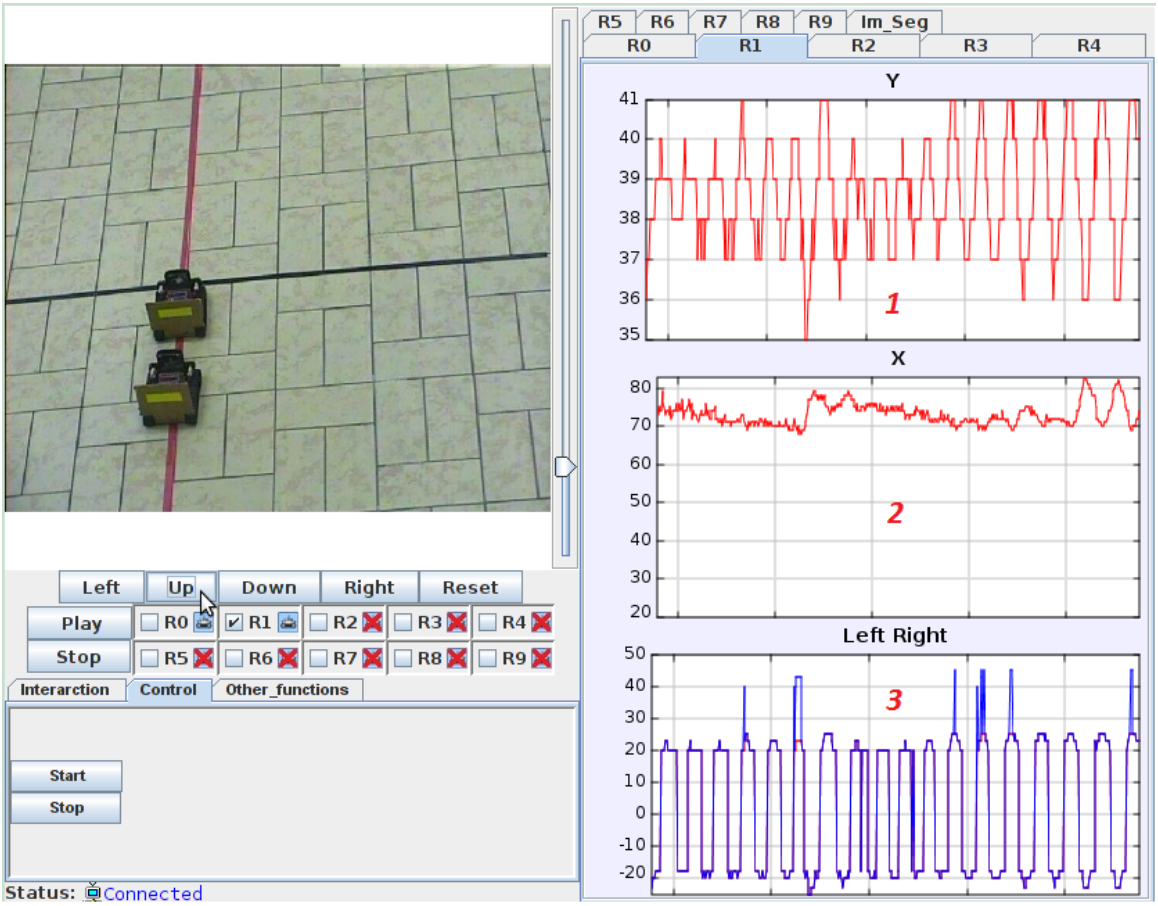


Figura 4.24. Control de posición de tipo líder-seguidores (el líder sigue una línea).

izquierdo y derecho expresadas en unidades a escala (% de la velocidad) entre -100 (velocidad máxima hacia atrás) y 100 (velocidad máxima hacia delante). Como muestra la Figura 4.24, para que el robot seguidor se encuentre a la distancia deseada del robot líder (30 cm), la altura del marcador debe estar en torno a los 38 píxeles. Mientras que para estar alineado la posición del marcador en el eje horizontal debe estar entorno a los 70 píxeles.

La Figura 4.25 muestra el mismo experimento, pero en este caso el robot líder es controlado remotamente por el estudiante. En este caso, el experimento consiste en enviar órdenes al robot líder que hacen que éste se mueva según las órdenes recibidas. Las órdenes se envían a través de los botones (representados con flechas) que se observan en la interfaz gráfica de usuario. Estas flechas indican el sentido en que se desea que el robot se mueva. Mientras que la velocidad se corresponde con el control deslizante que

se encuentra junto a estos botones (Panel No. 4).

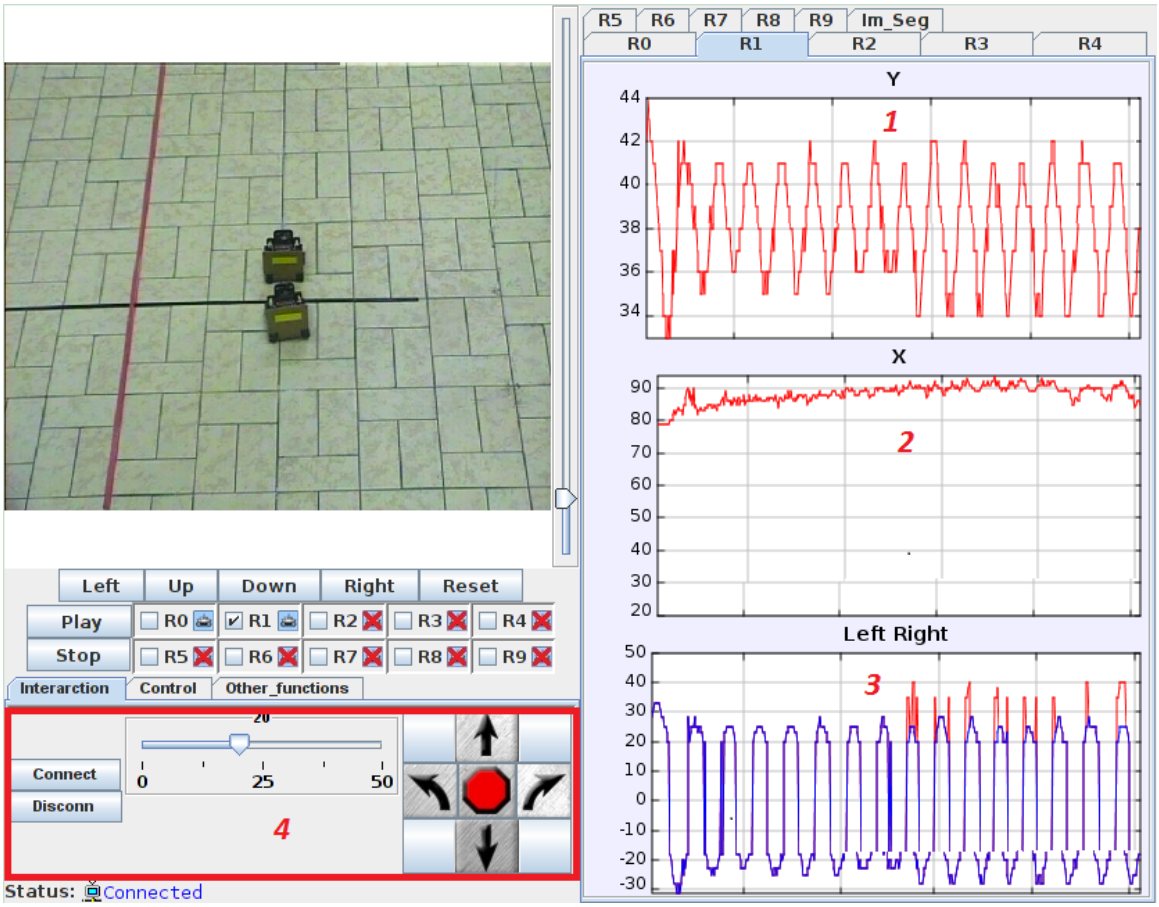


Figura 4.25. Control de posición de tipo líder-seguidores (el líder es controlado remotamente).

Como se aprecia en la Figura 4.25, al igual que en el caso anterior, el robot seguidor mantiene la distancia (30 cm) con el robot líder, por lo que la altura del marcador está entorno a los 38 píxeles (Panel No. 1). Mientras que en el caso de la alineación con el robot líder, a partir de la imagen de la Figura 4.25 es fácil observar que el robot seguidor no está totalmente alineado. Por lo que en el gráfico (Panel No. 2) de la posición del marcador en el eje horizontal se encuentra entre los 80 y 90 píxeles cuando para estar totalmente alineado debería estar en los 70 píxeles.

La Figura 4.26 muestra este mismo experimento pero en este caso con 5 robots. Al igual que en el caso anterior, el robot líder es manipulado remotamente por el estudiante. Los gráficos muestran los datos enviados por el robot seguidor (R6) que es el último de la fila.

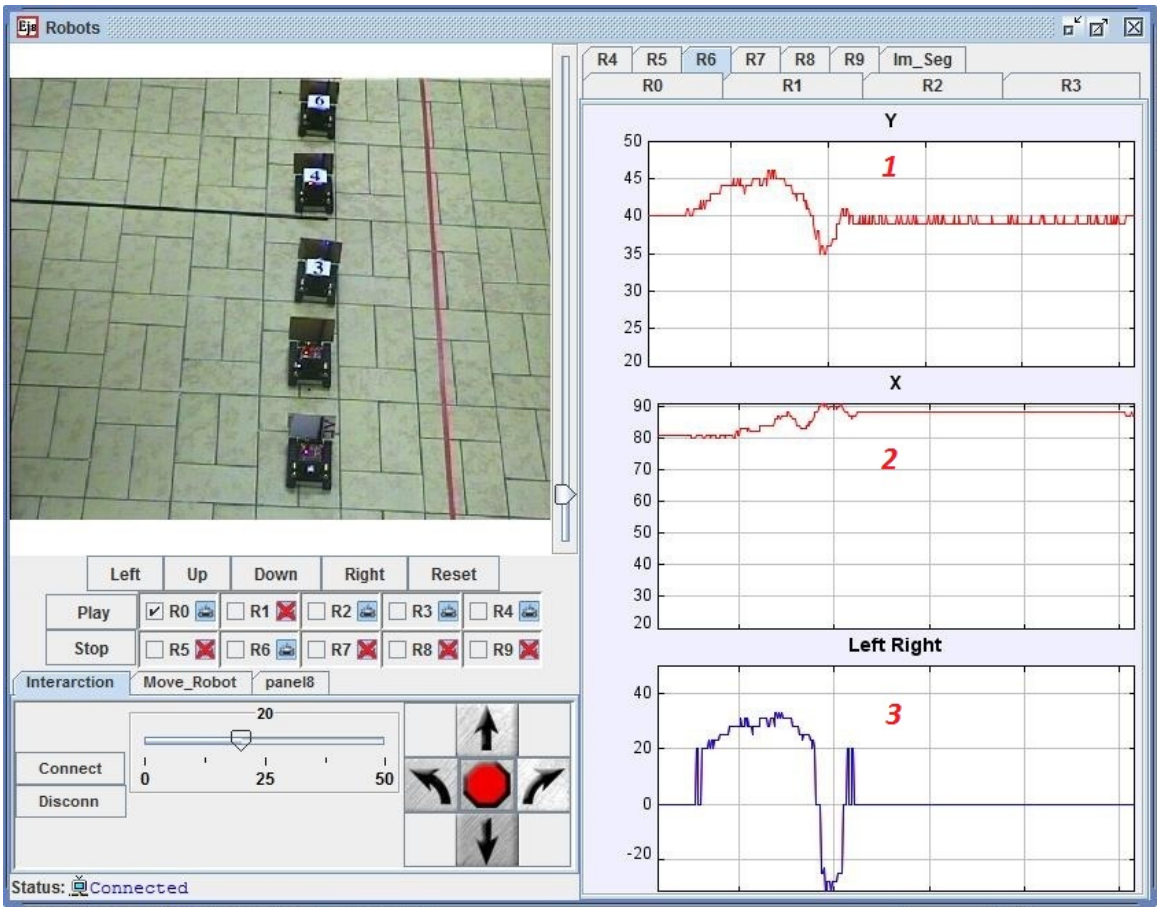


Figura 4.26. Control de posición de tipo líder-seguidores (el líder es controlado remotamente).

Como se puede apreciar en el gráfico (Panel No. 1), la altura del marcador se mantiene entorno a los 38 píxeles, lo que indica que la distancia con el robot que le precede es de aproximadamente 30 cm. El gráfico (Panel No. 2) muestra que el robot no está totalmente alineado por lo que el valor de la posición del marcador en el eje horizontal se encuentra entre los 80 y 90 píxeles, cuando debería estar entorno a los 70 píxeles. En este caso, a diferencia de los dos experimentos anteriores en el momento de tomar la imagen los robots se encuentran detenidos, por lo que el gráfico (Panel No. 3) muestra que los valores de las órdenes enviadas a los motores derecho e izquierdo son cero.

La Figura 4.27 muestra el mismo experimento para dos robots pero en este caso el robot líder sigue una trayectoria curva dibujada en el suelo. Este experimento permite demostrar el funcionamiento del control lateral en el robot seguidor. Los paneles 1, 2

y 3 muestran las imágenes enviadas desde el robot seguidor: (Panel No. 1), imagen obtenida con la cámara, (Panel No. 2) imagen después del proceso de segmentación y (Panel No .3) resultado del proceso de detección de bordes.

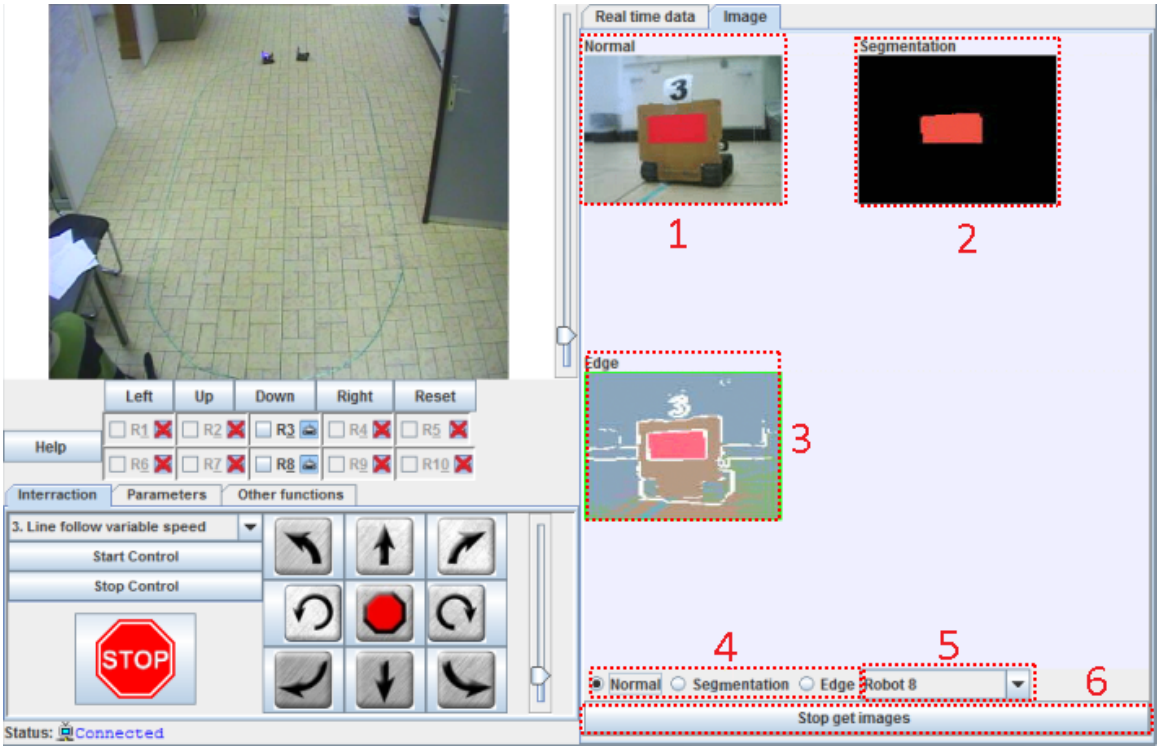


Figura 4.27. Control de posición de tipo líder-seguidores (el líder sigue una trayectoria curva).

4.4.2 Análisis de los resultados obtenidos

Los resultados obtenidos con esta plataforma han sido satisfactorios para todos los experimentos realizados: control de formación de tipo líder seguidores (el líder sigue una trayectoria recta o curva dibujada en el suelo y el líder es manipulado remotamente). Todos los experimentos dependen en gran medida de la comunicación inalámbrica entre el ordenador que actúa como servidor y los robots (tanto para el caso del envío de órdenes a los robots como para recibir los datos de éstos). Esta comunicación se establece usando una red *Wifi* que es muy estable y no experimenta muchas pérdidas de datos, por lo que desde el punto de vista de las comunicaciones la plataforma no se ve muy afectada. Durante el desarrollo de los experimentos no se mantiene comunicación entre el ordenador y los robots seguidores, ya que éstos ejecutan el programa que llevan

a bordo en el cual se encuentra implementado el algoritmo de control. En cambio, en el caso del robot líder, para el experimento en que éste es manipulado remotamente, la comunicación se mantiene activa durante todo el experimento. Sin embargo, durante el desarrollo de ambos experimentos si se desea establecer una nueva posición para cualquiera de los robots, se establece comunicación con éste y se le envía una señal para que detenga el algoritmo de control implementado a bordo y ejecute las órdenes que se le envían desde la ventana principal de la aplicación de usuario. Por este motivo es muy importante que la comunicación entre los robots y el ordenador que actúa como servidor, sea estable y esté disponible cuando se necesite.

Por otra parte hay que destacar que tanto para el robot líder como para los robots seguidores, el algoritmo de control depende en gran medida del tratamiento de imágenes usando la cámara de a bordo. En el caso de los robots seguidores, la referencia se obtiene a partir de un rectángulo de color que aparece en la parte posterior de cada robot. Mientras que en el caso del robot líder, la referencia se obtiene de la línea de color dibujada en el suelo. A pesar de la complejidad que implica el tratamiento de imágenes, debido al alto coste computacional que presupone, se obtuvieron muy buenos resultados, lo que demuestra que los robots tienen buenas prestaciones desde el punto de vista de procesamiento de imágenes de la cámara de a bordo.

5

Conclusiones y Líneas Futuras

En este capítulo se describen las conclusiones generales y específicas de la investigación realizada en este trabajo de tesis y se proponen además, algunas líneas futuras de investigación.

5.1 Conclusiones Generales

En la enseñanza de Ingeniería de Control los conceptos que se imparten a través de las clases, a menudo son complementados por actividades prácticas en los laboratorios. La experimentación práctica con plantas reales es un inconveniente en la modalidad de enseñanza a distancia. En este sentido, los laboratorios virtuales y remotos constituyen una importante alternativa para eliminar esta desventaja, ya que los estudiantes pueden acceder a los experimentos de forma remota a cualquier hora y desde cualquier lugar.

Sin embargo, el proceso de transformación de un experimento clásico basado en una planta piloto en un experimento interactivo y remoto a través de *Internet*, es una tarea compleja desde el punto de vista de la programación y las limitaciones inherentes a la comunicación a distancia.

Esta tesis describe la implementación de cuatro plataformas totalmente interactivas para el desarrollo de prácticas de laboratorio de control automático y robótica móvil. Estas plataformas están basadas en una arquitectura Cliente-Servidor con dos modos de trabajo: local (simulación) y remoto (conexión a través de *Internet* con la planta real). La estructura de esta arquitectura consiste en una aplicación del lado del cliente (desarrollada en *Easy Java Simulations*) que trabaja en modo local cuando se ejecuta en modo de simulación. Mientras que para trabajar en modo remoto se establece una comunicación a través de *Internet* con una aplicación del lado del servidor que controla la planta real situada en el laboratorio. Para establecer la comunicación a través de *Internet* se usa una interfaz que depende del lenguaje en que se haya programado la aplicación local que controla la planta real: *JIM-Server* para *MATLAB*, *JIL-Server* para *LabVIEW* y *JIC-Server* para *Microsoft Visual C#*.

5.2 Conclusiones Específicas

A continuación se resumen las principales aportaciones de esta tesis, detallando los componentes implementados y utilizados para cada uno de los laboratorios que conforman las plataformas; así como los experimentos que dichas plataformas permiten llevar a cabo.

1. La plataforma de control automático con el sistema de bola y aro permite desarrollar los siguientes dos experimentos: control de posición del aro, control del ángulo de la bola de su posición de equilibrio, cálculo de los ceros de transmisión, calcular la frecuencia de resonancia y observar el comportamiento de fase no mínima. Para ello se diseñaron e implementaron varios componentes que se describen a continuación:

    - Se ha desarrollado en *EJS* una simulación interactiva del sistema bola y aro

denominada *Cliente BA-L* que se puede comunicar con la aplicación *Servidor BA-L* desarrollada en *LabVIEW*. Esta aplicación permite además comparar los modelos lineal y no lineal del sistema bola y aro a través de la realización de los experimentos mencionados anteriormente. Permite además modificar las propiedades geométricas del sistema tales como: el radio del aro, el radio de la bola y la frecuencia de la señal de entrada. También es posible ajustar los parámetros del controlador seleccionado.

- Se ha desarrollado en *EJS* la simulación interactiva del sistema bola y aro denominada *Cliente BA-S* que se puede comunicar con la aplicación *Servidor BA-S* desarrollada en *MATLAB/Simulink*. Cuando se trabaja en modo remoto se ha implementado la comunicación de *EJS* con *MATLAB/Simulink*.
- Se ha desarrollado la aplicación *Servidor BA-L* usando *LabVIEW* para controlar de forma local la planta piloto "*Ball and Hoop System CE9*" a través de la tarjeta de adquisición de datos *PCI-1711L* de *Advantech Corporation*.
- Se ha desarrollado la aplicación *Servidor BA-S* usando *MATLAB/Simulink* para controlar de forma local la planta real a través de la tarjeta de adquisición de datos *PCI-1711L* de *Advantech Corporation*.
- Se ha utilizado la interfaz *JIL-Server* desarrollada en el Departamento de Informática y Automática de la UNED, para la comunicación a través de *Internet* entre la aplicación local de control de la planta (desarrollada en *LabVIEW*) y la aplicación cliente (desarrollada en *EJS*).
- Se ha utilizado la interfaz *JIM-Server* desarrollada en el Departamento de Informática y Automática de la UNED, para la comunicación a través de *Internet* entre la aplicación del lado del servidor que controla la planta (desarrollada en *MATLAB/Simulink*) y la aplicación cliente (desarrollada en *EJS*).

2. La plataforma de experimentos de control automático con el sistema de bola y plato permite desarrollar los siguientes experimentos: control de posición de la bola en un punto del plato y seguimiento de trayectorias (lineal, circular, cuadrada

y figura de *Lissajous*). En este caso, tanto la planta piloto como la aplicación del lado del servidor que controla la planta real habían sido construidas en la Universidad de Gante, por lo que el desarrollo de este laboratorio se adaptó a los componentes ya existentes. Partiendo de la planta piloto y su aplicación de control desarrollada en *Microsoft Visual C#*, se implementaron varios componentes que se describen a continuación:

- Se ha diseñado e implementado en *EJS* una simulación interactiva en tres dimensiones (3D) del sistema de bola y plato denominada *Cliente BP-C* (modo local). Esta aplicación permite llevar a cabo varios experimentos, modificar los parámetros de los controladores, conectar con la planta real, realizar la identificación de dicha planta y controlarla remotamente a través de *Internet*. Esta aplicación tuvo que ser adaptada para permitir acceder a las funcionalidades disponibles en la aplicación de control de la planta real.
- Se ha desarrollado e implementado la interfaz entre *Java* y *Microsoft Visual C#* denominada *JIC-Server*. Esta interfaz se utiliza para establecer la comunicación a través de *Internet* entre la aplicación *Cliente BP-C*(*EJS*) y la aplicación de control de la planta piloto (C#). Esta interfaz se implementó en ambos lenguajes usando el protocolo *TCP/IP* para la comunicación a través de *Internet*.
- Se ha implementado en la aplicación *Cliente BP-C* la posibilidad de enviar a la planta real, un fichero con una señal de entrada generada con *MATLAB* (*PRBS* o pulso binario pseudo-aleatorio) que permite la identificación de ésta. El fichero se envía desde el cliente a la entrada de la planta piloto y el resultado se almacena en un servidor *FTP* para que el estudiante pueda descargar dichos datos y realizar el proceso de identificación de la planta real.

3. La plataforma de experimentos de robótica móvil con los robots *Moway* permite desarrollar los siguientes dos tipos de experimentos: control de posición o estabilización en un punto de un robot móvil con evasión de obstáculos, control de

formación de robots móviles de tipo maestro-esclavos con evasión de obstáculos. Para ello se implementaron los componentes que se describen a continuación:

- Se ha diseñado e implementado la aplicación *Cliente MW* usando *EJS*. Esta aplicación cuando funciona en modo simulación es un simulador que permite la realización de experimentos de control de formación de robots móviles. Cuando funciona en modo remoto permite la conexión con los robots *Moway* situados en el laboratorio a través de *Internet* usando la interfaz *JIC-Server* y la aplicación de control *Servidor MW*.
- Se ha diseñado e implementado en *Microsoft Visual C#* la aplicación *Servidor MW* cuyas dos funciones principales son: capturar y enviar la posición absoluta a los robots obtenida desde la cámara y enviar a los robots las órdenes recibidas a través de *Internet* desde la aplicación *Cliente MW*, cuando ésta trabaja en modo remoto. Para la obtención de la posición de los robots se utiliza también una comunicación *TCP/IP* con la aplicación *SwisTrack* que es la que procesa las imágenes de vídeo de la cámara y determina la posición de cada robot.
- Se han implementado las funciones correspondientes para almacenar en un fichero (con extensión **.csv*) los datos de las posiciones de los robots para todos los experimentos. Dicho fichero se almacena en el en el ordenador que actúa como servidor para que los estudiantes puedan disponer de ellos una vez finalizados los experimentos.
- Se ha diseñado e implementado el *software* de control de los robots móviles usando para ello las comunicaciones inalámbricas por radiofrecuencia entre éstos.
- La implementación del método de evasión de obstáculos con los robots reales se incorporó en el *software* de control de éstos, usando los sensores disponibles ya que la comunicación inalámbrica presenta varias limitaciones como el tamaño del paquete de datos a transmitir.
- La implementación del *Software* de control y comunicación de los robots en lenguaje C.

4. La plataforma de experimentos de control de robots móviles *Surveyor SRV-1* permite tele-operar remotamente experimentos de control de formación de robots móviles de tipo líder-seguidores. En este caso, al igual que para el sistema bola y plato la aplicación de control local y el *software* que se ejecuta en los robots se desarrollaron en la Universidad de Gante. Por lo que la aplicación *Cliente SRV* se adaptó para mantener disponibles todas las funcionalidades del laboratorio. La comunicación con los robots se establece utilizando *MATLAB* a través de una red inalámbrica. Partiendo de estas condiciones se implementaron varios componentes que se describen a continuación:

   - La aplicación cliente *Cliente SRV* desarrollada en *EJS* trabaja en modo remoto y es la encargada de enviar a la aplicación *Servidor SRV* las órdenes para la interacción con los robots durante la realización de los experimentos. La comunicación a través de *Internet* se realiza usando la interfaz *JIM-Server* al igual que en el caso de la plataforma del sistema de bola y aro.
   - En el lado del servidor el *JIM-Server* es el encargado de "traducir" las órdenes recibidas desde la aplicación *Cliente SRV* en órdenes de *MATLAB* que ejecutan los robots. Estas órdenes permiten interactuar con los robots que a su vez ejecutan a bordo el *software* de control.
   - Se ha implementado del lado del servidor las funciones de *MATLAB* que permiten almacenar los datos de los experimentos en un servidor *FTP* al que los estudiantes pueden acceder una vez finalizados los experimentos.

Como resultado de esta tesis se han obtenido cuatro plataformas totalmente interactivas para el desarrollo de prácticas de control automático y robótica móvil. Dos de estas plataformas han sido incluidas en el proyecto *AutomatL@bs* que es una red de laboratorios virtuales y remotos para la enseñanza de la automática que se constituye mediante la integración de los recursos compartidos entre varias universidades españolas que participan en este proyecto liderado por el Departamento de Informática y Automática de la UNED.

5.3 Trabajos Futuros

Los sistemas utilizados para el desarrollo de estas plataformas son muy versátiles en cuanto a los experimentos que se pueden realizar. Una línea de trabajo a seguir en el futuro para mejorar estas plataformas es incorporar nuevas capacidades desde el punto de vista de la enseñanza del control. A continuación se proponen algunas de estas mejoras que pueden ser implementadas para cada una de las plataformas desarrolladas:

1. Las mejoras que se podrían añadir a la plataforma del sistema bola y aro son las siguientes:

   - La implementación de otros tipos de control como por ejemplo el *Reset Control* [1]. Este tipo de control introduce una no linealidad en el lazo de control y permite el estudio de otros fenómenos como es el caso de los Ciclos Límite utilizando el Método de la Función Descriptiva [2] y para su observación el método de los Mapas de Poincaré [3, 4].
   - La obtención de los parámetros del controlador usando métodos de algoritmos genéticos y lógica difusa [5]. Estas son alternativas para controlar este sistema de una forma más novedosa e interesante desde el punto de vista educativo.
   - La incorporación de realidad aumentada a la aplicación *Cliente BA-L*. Esto podría aumentar la interactividad del estudiante con la plataforma y la haría más atractiva.

2. En el caso de la plataforma del sistema bola y plato las sugerencias y mejoras que se pueden implementar son las siguientes:

   - La continua evolución de *EJS* permite en ocasiones mejorar las aplicaciones desarrolladas. Tal es el caso de la aplicación *cliente BP-C* que puede ser mejorada con la utilización de la funcionalidad recién incluida *Java 3D*. Esto sin dudas mejoraría la interfaz de la simulación desde el punto de vista visual y de funcionamiento de la misma.

- Desde el punto de vista de programación, la aplicación *Servidor BP-C* también se puede mejorar ya que la interacción con la aplicación *Cliente BP-C* a través de *Internet* introduce retardos que influyen en el lazo de control. La modificación de dicho código usando tiempo real, hilos o temporizadores, podría mejorar el rendimiento de dicha aplicación.

3. En cuanto a la plataforma de robots *Moway* los posibles trabajos futuros que se podrían introducir son las siguientes:

   - La comunicación inalámbrica con los robots ya que la limitación del tamaño del paquete de datos influye notablemente en la implementación de los diferentes experimentos.
   - Eliminar la confirmación de recepción que tal y como está configurada actualmente por defecto no se puede eliminar y esto produce un tráfico de datos elevado. Esta configuración es parte del *Firmware* de los robots, pero seguramente puede eliminarse por parte del fabricante para garantizar menor tráfico de datos y con ellos enviar más datos a los robots en un menor tiempo.
   - Utilizar otro método de comunicación inalámbrica como es el caso de una red *Wifi* que sin duda mejoraría las comunicaciones entre el servidor y los robots además de que permitiría la incorporación de otro tipo de robots móviles (sistema heterogéneo e robots) a la plataforma para hacerla mucho más interesante.
   - La herramienta empleada para la obtención de la posición absoluta de los robots (*SwisTrack*) permite la incorporación de otra cámara. Esto amplía el área de trabajo, lo que permitiría un mayor espacio para el desarrollo de experimentos más complejos y la incorporación de más robots a la plataforma.

4. En el caso de la plataforma de los robots *Surveyor SRV-1* las mejoras que se pueden implementar son las siguientes:

   - El hecho de que la comunicación inalámbrica que se usa en la plataforma

es la *Wifi*, permite el envío de paquetes de datos de tamaño suficiente para realizar los experimentos. Sin embargo, la limitación fundamental es que los robots no pueden comunicarse entre ellos ya que su *Firmware* no lo permite hasta el momento de realización de esta tesis. Sin duda que si esta limitación pudiera superarse, otro tipo de experimentos más vistosos y con mayores funcionalidades se podrían implementar.

- Estos robots permiten la incorporación de otros módulos a su *hardware* como por ejemplo *GPS*. Con nuevos componentes de *hardware* diferentes experimentos que consideren la localización en un área de mayor tamaños pueden implementarse, tratando siempre de despertar la curiosidad y el interés de los estudiantes.

En general para todas las plataformas se podría implementar el controlador del lado del cliente, para estudiar los efectos del retardo de la red en el control de las plantas piloto. También se podría incorporar la implementación de la super-computación con el fin de lograr un funcionamiento de las plataformas más versátil en tiempo real. Finalmente sería interesante incluir la implementación del control basado en eventos para mejorar la eficiencia en el funcionamiento de las plataformas.

Bibliografía

[1] J. C. Clegg. A nonlinear integrator for servomechanisms. *Transaction of the American Institute of Electrical Engineers*, vol. 77, pp. 41–42, 1958.

[2] O. Beker, C. V. Hollot, and Y. Chait. Forced oscillations in reset control systems. *Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 41–42, 2000.

[3] S. Seydel. *Practical Bifurcation and Stability Analysis*. Springer, third edition, 1994. ISBN 978-1-4419-1739-3.

[4] J. M. Gonçalves. *Constructive Global Analysis of Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[5] I. Griffin. On-line PID Controller Tuning using Genetic Algorithms. Master's thesis, Dublin City University, Dublín, 9 Ireland, Aug. 2003.

[6] R. W. Battenberg. The Boston Gazette. *Epistolodidaktika*, no. 1, pp. 44–45, 1971.

[7] B. L. Bower and K. P. Hardy. From correspondence to cyberspace: Changes and challenges in distance education. *New Directions for Community Colleges*, no. 128, pp. 5–12, 2004.

[8] J. M. Suárez and D. Anaya Nieto. Educación a distancia y presencial: diferencias

en los componentes cognitivo y motivacional de estudiantes universitarios. *Revista Iberoamericana de Educación a Distancia*, vol. 7, pp. 65–75, 2004.

[9] J. E. Padula. Una introducción a la Educación a Distancia. *Fondo de Cultura Económica*, 2003.

[10] G. Varelo. Historia de la Educación a Distancia. *Timetoast*, 2012. URL: http://www.timetoast.com/timelines/historia-de-la-educacion-a-distancia.

[11] I. R. Alfonso. La educación a distancia. *ACIMED*, vol. 11, no. 1, pp. 3–4, 2003. ISSN 1561-2880. URL: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352003000100002&lng=es&nrm=iso&tlng=es.

[12] Time Rime. Evolución de la Educación a Distancia, 2012. URL: http://timerime.com/en/timeline/1302824/Historia+de+la+educacin+a+distancia/.

[13] L. García. Historia de la Educación a Distancia. *Revista Iberoamericana de Educación a Distancia*, vol. 2, no. 1, pp. 8–27, 1999.

[14] M. Crichlow and D. Sánchez. Educación a Distancia. *Universidad Tecnológica de Panamá*, 1999.

[15] R. F. Bacallao and M. F. Bacallao. La Educación a Distancia, sus retos y posibilidades. *Eticanet*, no. 1, 2003. ISSN 1695-324X. URL: http://www.ugr.es/~sevimeco/revistaeticanet/Numero1/Articulos/EaDretos.pdf.

[16] Universidad Nacional de Educación a Distancia (UNED). Nuestra historia, 2012. URL: http://portal.uned.es/portal/page?_pageid=93,499271,93_20500119&_dad=portal&_schema=PORTAL.

[17] AIESAD. La Asociación Iberoamericana de Educación Superior a Distancia. Más de tres décadas de intercambio y cooperación, 2012. URL: http://portal.uned.es/portal/page?_pageid=375,2898784&_dad=portal&_schema=PORTAL.

[18] B. S. Heck. Special report: Future directions in control education. *IEEE Control System Magazine*, vol. 19, no. 5, pp. 35–58, 1999.

[19] S. Dormido. Control learning present and future. *IFAC Annual Reviews in Control*, vol. 28, pp. 115–136, 2005.

[20] I. Calvo, E. Zulueta, U. Gangoiti, and J. M. López. Laboratorios remotos y virtuales en enseñanzas técnicas y científicas. *IKASTORRATZA. e-Revista de Didáctica*, no. 3, 2009. URL: `http://www.ehu.es/ikastorratza/3_alea/laboratorios.pdf`.

[21] D. Gillet, G. Fakas, Y. Rekik, K. Zeramdini, F. Geoffroy, and S. Ursulet. The Cockpit. An Effective Metaphor for Remote Experimentation in Engineering Education. *International Journal of Engineering Education*, vol. 19, no. 3, pp. 389–397, 2003.

[22] R. Dormido, H. Vargas, N. Duro, J. Sánchez, S. Dormido-Canto, G. Farias, F. Esquembre, and S. Dormido. Development of a web-based control laboratory for automation technicians: The three-tank system. *IEEE Transaction on Education*, vol. 51, no. 1, pp. 35–44, 2008.

[23] J. Andújar and T. Mateo. Diseño de laboratorios virtuales y/o remotos. Un caso práctico. *Revista Iberoamericana de Automática e Informática Industrial*, vol. 7, no. 1, pp. 64–72, 2012.

[24] A. Leva. A hands-on experimental laboratory for undergraduate courses in automatic control. *IEEE Transactions on Education*, vol. 46, no. 2, pp. 263–272, May. 2003. ISSN 0018-9359.

[25] A. Leva. An experimental laboratory on control structures. In *2004 American Control Conference*, vol. 4, pp. 3227–3232, Jun. 2004. ISBN 0743-1619.

[26] C. Chandrasekara and A. Davari. Inverted pendulum: an experiment for control laboratory. In *Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory*, pp. 570–573, Mar. 2004.

[27] M. W. Spong and D. J. Block. The Pendubot: a mechatronic system for control research and education. In *Proceedings of the 34th IEEE Conference on Decision and Control*, pp. 555–556, Dec. 1995.

[28] R. Radharamanan and H. E. Jenkins. Robot Applications in Laboratory-Learning Environment. In *Proceedings of the Thirty-Ninth Southeastern Symposium on System Theory*, pp. 80–84, Mar. 2007. ISBN 0094-2898.

[29] A. P. Medina, G. Hermida, J. Hernández, and E. Ladrón de Guevara Durán. Los Laboratorios Virtuales y Laboratorios Remotos en la Enseñanza de la Ingeniería. *Revista Internacional de Educación en Ingeniería*, vol. 4, pp. 24–30, 2011.

[30] J. Monge, M. Rivas, and V. H. Méndez. La evolución de los laboratorios virtuales durante una experiencia de cuatro años con estudiantes a distancia, 2002.

[31] M. Aburdene, E. Mastascusa, and R. Massengale. A proposal for a remotely shared control systems laboratory. In *ASEE, Frontiers in Education Conference*, pp. 589–592, 1991.

[32] B. Aktan and C. A. Bohus. Distance learning applied to control engineering laboratories. *IEEE Transactions on Education*, vol. 39, no. 3, pp. 320–326, 1996.

[33] Universidad Nacional de Educación a Distancia (UNED). Automatl@bs Red de Laboratorios de Automática, 2012. URL: `http://lab.dia.uned.es/automatlab/`.

[34] H. A. Latchman, Ch. Salzmann, D. Gillet, and H. Bouzekri. Information technology enhanced learning in distance and conventional education. *IEEE Transactions on Education*, vol. 42, no. 4, pp. 247–254, Nov. 1999. ISSN 0018-9359.

[35] D. Gillet, S. El Helou, Y. Ch. Man, and C. Salzmann. Turning Web 2.0 Social Software into Versatile Collaborative Learning Solutions. In *Advances in Computer-Human Interaction*, pp. 170–176, Feb. 2008.

[36] L. Gomes and S. Bogosyan. Current Trends in Remote Laboratories. *IEEE Transactions on Industrial Electronics*, vol. 56, no. 12, pp. 4744–4756, 2009.

[37] A. Leva and F. Donida. Multifunctional Remote Laboratory for Education in Automatic Control: The CrAutoLab Experience. *IEEE Transactions on Industrial Electronics*, vol. 55, no. 6, pp. 2376–2385, Jun. 2008. ISSN 0278-0046.

[38] J. L. Guzmán, H. Vargas, J. Sánchez, M. Berenguel, S. Dormido, and F. Rodríguez. Education Research in Engineering Studies: Interactivity, Virtual and Remote Labs. *Distance Educations Research Trends*, 2007.

[39] N. Aliane. Limitaciones Pedagógicas de los Laboratorios Remotos de Control. In *XXIX Jornadas de Automática CEA-IFAC*, 2008.

[40] M. Pipan, T. Arh, and B. J. Blazic. Advanced evocational Education of Mechatronic Professions. *Journal of Education and Information Technologies*, vol. 3, no. 1, pp. 12–19, 2009.

[41] S. Rae. Using telerobotics for remote kinematics laboratories, 2004.

[42] E. Luther. Developing Interactive Simulations with the LabVIEW Player. *Connexions*, 2006. URL: `http://cnx.org/content/m14131/latest/`.

[43] J. Henry and R. Zollars. Experiments and Local Simulations: Student Experiences, Satisfaction and Suggestions. In *Annual Meeting of American Society for Engineering Education*, 2003.

[44] M. Casini, A. Garulli, A. Giannitrapani, and A. Vicino. A MATLAB-based remote lab for Multi-Robot Experiments. In 8^{th} *IFAC Symposium on Advances in Control Education*, 2009.

[45] F. Torres, F. A. Candelas, S. T. Puente, J. Pomares, P. Gil, and F. G. Ortiz. Experiences with Virtual Environment and Remote Laboratory for Teaching and Learning Robotics at the University of Alicante. *International Journal of Engineering Education. Special Issue on Robotics Education*, vol. 22, no. 6, 2006.

[46] F. J. Martínez, R. González, F. Rodríguez, and J. L. Guzmán. Laboratorio Virtual y Remoto para la Enseñanza de Robotica Paralela. *CEA Jornadas de Enseñanza de la Ingeniería de Sistemas y Automática*, 2010. URL: `http://www.ual.es/personal/rgonzalez/papers/eiwisa.pdf`.

[47] C. A. Jara, F. A. Candelas, and F. Torres. Laboratorios Virtuales y Remotos para la enseñanza de Robótica Industrial. In *XXVIII Jornadas de Automática*, 2007.

[48] C. A. Jara, F. A. Candelas, and F. Torres. Robolab.ejs: a new tool for robotics e-learning. In *Remote Engineering and Virtual Instrumentation (REV 2008)*, 2008.

[49] S. Dormido, S. Dormido-Canto, R. Dormido, J. Sánchez, and N. Duro. The Role of Interactivity in Control Learning. *International Journal of Engineering Education*, vol. 21, no. 6, pp. 1122–1133, 2005.

[50] H. Vargas, R. Dormido, N. Duro, and S. Dormido-Canto. Creación de laboratorios virtuales y remotos usando Easy Java Simulations y LabVIEW: El sistema heatflow como un caso de estudio. In *XXVII Jornadas de Automática*, pp. 1182–1188, 2006.

[51] S. Dormido, H. Vargas, J. Sánchez, R. Dormido, N. Duro, S. Dormido-Canto, and F. Morilla. Developing and Implementing Virtual and Remote Labs for Control Education: The UNED pilot experience. *17th IFAC World Congress*, pp. 8159–8164, 2008.

[52] N. Duro, R. Dormido, H. Vargas, S. Dormido-Canto, J. Sánchez, G. Farias, F. Esquembre, and S. Dormido. An Integrated Virtual and Remote Control Lab: The Three-Tank System as a Case Study. *Computing in Science & Engineering*, vol. 10, pp. 50–59, 2008. ISSN 1521-9615.

[53] TecQuipment. *Ball and Hoop Apparatus CE9*, 2010.

[54] P. E. Wellstead. The Ball and Hoop System. *Automática*, vol. 19, no. 4, pp. 401–403, 1983.

[55] P. E. Wellstead. *Ball and Hoop 1: Basics*, 2000. URL: `http://www.control-systems-principles.co.uk/whitepapers/ball-and-hoop1.pdf`.

[56] P. E. Wellstead and M. Readman. *Ball and Hoop 2: Control and analysis*, 2000. URL: `http://www.control-systemsprinciples.co.uk/whitepapers/ball-andhoop2.pdf`.

[57] J. D. Aplevich. *Implicit Linear Systems (Lecture Notes in Control and Information Sciences)*. Springer, 1991.

[58] B. C. Kuo. *Sistemas de Control Automático*. Pearson Education, 1996.

[59] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. Wiley, 2005.

[60] I. Calvo, F. López, E. Zulueta, and J. Pascual. Laboratorio de control remoto de un sistema de Ball & Hoop. In *JAT08 XXIX Jornadas de Automática*, 2008. ISBN 978-84-691-6883.-7. URL: `http://jata08-events.urv.cat/files/298.pdf`.

[61] S. Kanthalakshmi and V. Manikandan. Genetic Algorithm Based Self Tuning Regulator. *International Journal of Engineering Science and Technology*, vol. 2, no. 12, pp. 7719–7728, 2012.

[62] National Instruments. *Learning with LabVIEW 8 Student Edition Software*. Prentice Hall, 2008. ISBN 0138004609.

[63] MathWorks. *MATLAB/Simulink - Dynamic Simulation for MATLAB*, 2010.

[64] F. Morilla and I. López. Curso de Doctorado: Apuntes sobre identificación no paramétrica, 2009.

[65] K. J. Åström and T. Hägglund. *Automatic tuning of PID Controllers*. Instrument Society of America, 1988. ISBN 1-55617-081-5.

[66] K. J. Åström and T. Hägglund. *PID Controllers: Theory, Design and Tuning*. Instrument Society of America, 2 edition, 1995.

[67] K. J. Åström and T. Hägglund. *Advanced PID Control*. Research Triangle Park, NC ISA The Instrumentation, Systems, and Automation Society, 2005.

[68] P. A. Padilla, F. Chang, M. Torres, and H. Dominguez. *Métodos de identificación dinámica*, 2008.

[69] F. Morilla, A. González, and N. Duro. Auto-Tuning PID controllers in terms of relative damping. In IFAC 2000, editor, *Proceedings of IFAC Workshop on Digital Control*, 2000.

[70] V. Alfaro. Identificación de procesos sobreamortiguados utilizando técnicas de lazo cerrado. *Ingeniería*, vol. 11, no. 1, pp. 27–40, 2001.

[71] F. Esquembre. Easy Java Simulations: A software tool to create scientific simulations in Java. *Computer Physics Communications*, vol. 156, pp. 199–204, 2004.

[72] F. Esquembre. *Creación de Simulaciones Interactivas en Java*. Prentice Hall, 2005.

[73] F. Morilla. Curso de Doctorado: Ajuste empírico de controladores PID, 2009.

[74] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Transactions of the American Society of Mechanical Engineers*, vol. 64, pp. 759–768, 1942.

[75] K. J. Åström and T. Hägglund. Automatic tuning and adaptation for PID controllers - a survey. *Control Engineering Practice*, vol. 1, no. 4, pp. 699–714, 1993. ISSN 0967-0661. URL: `http://www.sciencedirect.com/science/article/pii/6706619391394C`.

[76] K. J. Åström and T. Hägglund. Revisiting the Ziegler-Nichols Step Response Method for PID Control. *Journal of Process Control*, vol. 14, no. 6, pp.

635–650, 2004. URL: http://www.sciencedirect.com/science/article/pii/S0959152404000034.

[77] G. López, J. Amable, A. Correas, J. Ignacio, and G. de la Vega. *Controlador PID*. Barcelona Tiempo Real, 2da edition, 1994.

[78] G. H. Cohen and G. A. Coon. Theoretical consideration of retarded control. *Transaction of American Society of Mechanical Engineers*, vol. 75, pp. 827–834, 1953.

[79] A. M. López, J. A. Miller, P. W. Murrill, and C. L Smith. Tuning Controllers With Error-Integral Criteria. *Instrumentation Technology*, vol. 14, no. 11, pp. 57–62, 1967.

[80] K. L. Chien, J. A. Hrones, and J. B. Reswick. On the automatic control of generalized passive systems. *Transaction of American Society of Mechanical Engineers*, pp. 74–175, 1952.

[81] F. S. Wang, W. S. Juang, and C. T. Chan. Optimal Tuning of PID Controllers for Single and Cascade Control Loops. *Chemical Engineering Communications*, vol. 132, pp. 15–34, 1995.

[82] J. Hauser, S. Sastry, and P. Kokotovic. Nonlinear control via approximate input-output linearization: the ball and beam example. *IEEE transactions on Automatic control*, vol. 37, no. 7, pp. 392–398, 1992.

[83] B. Ming and T. Yantao. A Nonlinear Switching Controller for a Ball and Plate System. In *24th Chinese Control Conference*, pp. 940–943, 2005.

[84] M. Bai, Y. Tian, J. Su, and J. Zhao. A nonlinear switching controller for a ball-and-plate system. In *24th Chinese Control Conference*, pp. 940–943, 2005.

[85] W. Hongrui, T. Yantao, F. Siyan, and S. Zhen. Nonlinear Control for Output Regulation of Ball and Plate System. In *27th Chinese Control Conference*, pp. 382–387, 2008.

[86] D. Yuan and Z. Zhang. Modelling and control scheme of the ball-plate trajectory-tracking pneumatic system with a touch screen and a rotary cylinder. *Control Theory & Applications*, vol. 4, no. 4, pp. 573–589, 2010.

[87] W. Rekdalsbakken. The use of artificial intelligence in controlling a 6 DOF motion. In *21st European Conference on Modelling and Simulation*, pp. 249–254, 2007.

[88] M. De Paepe. Design of a ball and plate system with a 6 DoF motion platform and vision-based feedback. Master's thesis, Ghent Univestity. Faculty of Engineering and Architecture. EeSA-department of Electrical energy Systems and Automation, 2011.

[89] Logitech. *Logitech Quick Cam Pro 4000 Manual*, 2013. URL: `http://logitech-en-amr.custhelp.com/ci/fattach/get/9596/0/filename/QuickCam+Pro+4000.pdf`.

[90] J. Y. Bouguet. Camera Calibration Toolbox for MATLAB, 2011.

[91] G. Bradski and A. Kaehler. *Learning OpenCV: computer vision with the OpenCV library*. O'Reilly, 2008.

[92] R. De Keyser and C. Ionescu. A MATLAB Interactive Tool for Computer Aided Control Systems Design in Frequency Domain: FRTool, MATLAB - Modelling, Programming and Simulations, 2010.

[93] MathWorks. Linear-Quadratic Regulator (LQR) design, 2012. URL: `http://www.mathworks.es/es/help/control/ref/lqr.html`.

[94] J. Angeles. *Fundamentals of Robotic Mechanical Systems. Theory, Methods and Algorithms*. Springer-Verlag, 2007.

[95] Y. Chang, S. Ma, H. Wang, and D. Tan. A kinematic modeling method for a wheeled mobile robot. In *International Conference on Mechatronics and Automation ICMA 2009*, pp. 1100–1105, Aug. 2009.

[96] J. Liu, J. Jiang, W. Chen, and W. Chen. The intelligent leg design for cockroach robots based on differential gear structure. In *IEEE 6th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1654–1659, Jun. 2011.

[97] H. Kwon, H. Shim, D. Kim, S. Park, and J. Lee. A development of a transformable caterpillar equipped mobile robot. In *International Conference on Control, Automation and Systems. ICCAS '07*, pp. 1062–1065, Oct. 2007.

[98] P. Muir and Ch. P. Neuman. Kinematic Modeling of Wheeled Mobile Robots. Technical Report CMU-RI-TR-86-12, Robotics Institute, Pittsburgh, PA, Jun. 1986.

[99] F. Tang. Kinematics and design of a wheeled mobile robot, 2002.

[100] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J. P. Laumond. Stabilization of trajectories for systems with nonholonomic constraints. *IEEE Transactions on Automatic Control*, vol. 39, no. 1, pp. 216–222, Jan. 1994.

[101] R. Silva, M. A. Molina, V. M. Hernández, G. Silva, M. Marciano, and E. A. Portilla. Modelado y control de un robot móvil tipo Newt en la tarea de seguimiento de trayectoria. *Télématique*, vol. 7, no. 20, pp. 129–145, 2008.

[102] R. Silva-Ortigoza, G. Silva-Ortigoza, V. M. Hernandez-Guzmán, V. R. Barrientos-Sotelo, J. M. Albarran-Jiménez, and V. M. Silva-García. Trajectory Tracking in a Mobile Robot without Using Velocity Measurements for Control of Wheels. *IEEE Latin America Transactions*, vol. 6, no. 7, pp. 598–607, Dec. 2008.

[103] J. Shao, G. Xie, J. Yu, and L. Wang. Leader-Following Formation Control of Multiple Mobile Robots. *Proceedings of the 2005 IEEE International Symposium on Mediterranean Conference on Control and Automation Intelligent Control*, vol. 62, pp. 808–813, 2005.

[104] D. Chwa, S. Hong, and B. Song. Robust posture stabilization of wheeled mobile

robots in polar coordinates. *Proceedings of the* 17^{th} *International Symposium on Mathematical Theory of Network and Systems*, 2006.

[105] Y. Zhang, D. Hong, J. H. Chung, and S. A. Velinsky. Dynamic model based robust tracking control of a differentially steered wheeled mobile robot. In *Proceedings of the 1998 American Control Conference*, vol. 2, pp. 850–855, Jun. 1998.

[106] A. P. Aguiar and A. Pascoal. Stabilization of the Extended Nonholonomic Double Integrator Via Logic-Based Hybrid Control. 6^{th} *International IFAC Symposium on Robot Control. SYROCO'00*, 2000.

[107] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, Apr. 1986. ISSN 0278-3649. URL: `http://dx.doi.org/10.1177/027836498600500106`.

[108] V. J. González, R. Parkin, M. López, J. M. Dorador, and M. J Guadarrama. A wheeled mobile robot with obstacle avoidance capbility. *Ingeniería Mecánica. Tecnología y Desarrollo*, vol. 1, no. 005, pp. 159–166, 2004.

[109] J. L. Susa and D. A. Ramos. ABEO. Algorimo Bioinspirado de Evasión de Obstáculos. *Tecnura*, vol. 13, no. 25, 2009.

[110] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1398–1404, 1991.

[111] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environment. *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 572–577, 1990.

[112] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[113] I. Ulrich and J. Borenstein. VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *1998 IEEE International Conference on Robotics and Automation*, pp. 1572–1577, May. 1998. URL: http://dblp.uni-trier.de/db/conf/icra/icra1998-2.html#UlrichB98.

[114] Bizintek Innova. *Moway Robots*, 2012. URL: http://www.moway-robot.com.

[115] Bizintek Innova. *Manual de Usuario de Moway v 1.0.0*, 2009. URL: http://www.superrobotica.com/download/Moway/Manual%20Moway%20v1.0.0.pdf.

[116] Bizintek Innova. *Manual de Usuario de Moway v 2.1.0*, 2011. URL: http://www.adrirobot.it/moway/pdf/mOway%20User%20Manua%202.1.0.pdf.

[117] Bizintek Innova. *Moway's Beginners Manual v 3.0.4*, 2012. URL: http://moway-robot.com/wp-content/files_mf/manualusuariomoway.pdf.

[118] Microchip Technology. *PIC18F87J50 Family Data Sheet*, 2009. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/39775c.

[119] Vishay Semiconductors. *CNY70 Reflective Optical Sensor with Transistor Output*, 2012. URL: http://www.vishay.com/docs/83751/cny70.pdf.

[120] Kingbright. *KPA-3010*, 2012. URL: http://pdf1.alldatasheet.com/datasheet-pdf/view/107244/KINGBRIGHT/KPA-3010.html.

[121] Sharp Corporation. *OPTO Optoelectronics*, 2012. URL: http://www.sharp-world.com/products/device/lineup/selection/pdf/opto201201_e.pdf.

[122] Bizintek Innova. *Manual de usuario del módulo BZI-RF2GH4*, 2007. URL: http://coolab.umh.es/moway_web/moway/Manual%20RF.pdf.

[123] Nordic Semiconductors. *nRF24L01 Single Chip 2.4GHz Transceiver Product Specification*, 2012. URL: http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01.

[124] D. V. Neamtu. Leader-Follower formation control for mobile robots in a Remote Laboratory. Master's thesis, Ghent Univestity. Faculty of Engineering and Architecture. EeSA-department of Electrical energy Systems and Automation, 2011.

[125] Surveyor Corporation. Surveyor SRV-1 Blackfin Robot. *Surveyor Robotics Journal*, 2010. URL: `http://surveyor.com/company/company.html`.

[126] Analog Devices. *Blackfin Embedded Processor ADSPBF537*, 2010. URL: `http://www.analog.com/static/importedfiles/datasheets/ADSBF537.pdf`.

[127] OmniVision. *OV9655 1.3 MPixel product brief.* 4275 Burton Drive Santa Clara, California 95054. Unite States, 2010. URL: `http://www.ovt.com/downloaddocument.php`.

[128] Analog Devices. *Analog Devices Open Source Koop*, 2012. URL: `http://blackfin.uclinux.org`.

[129] J. L. Martínez, A. Mandow, Morales J., S. Pedraza, and A. García-Cerezo. Approximating Kinematics for Tracked Mobile Robots. *The International Journal of Robotics Research*, vol. 24, no. 10, pp. 867–878, 2005.

[130] G. Sequeira. Vision based Leader-Follower formation control for mobile reobots. Master's thesis, University of Missouri-Rolla, 2007.

[131] Ch. Poynton. *Digital video and HDTV algorithms and interfaces.* Morgan Kaufmann, 2003.

[132] J. Sánchez, S. Dormido, F. Morilla, and R. Pastor. A Java/MATLAB-Based Environment for Remote Control System Laboratories: Illustrated With an Inverted Pendulum. *IEEE Transactions on Education*, vol. 3, pp. 321–329, 2004.

[133] H. Vargas, J. Sánchez, N. Duro, R. Dormido, S. Dormido-Canto, G. Farias, S. Dormido, F. Esquembre, Ch. Salzmann, and D. Gillet. A systematic two-layer approach to develop web-based experimentation environments for control

engineering education. *Intelligent Automation and Soft Computing*, vol. 14, no. 4, pp. 505–524, 2008.

[134] H. Vargas, J. Sánchez, C. A. Jara, F. A. Candelas, F. Torres, and S. Dormido. A Network of Automatic Control Web-Based Laboratories. *IEEE Transactions on Learning Technologies*, vol. 4, pp. 197–208, 2011. ISSN 1939-1382.

[135] A. Sayouti, A. Lebbat, H. Medromi, and F. Q. Aniba. Remote laboratory for teaching mobile systems. *International Journal of Computer and Network Security*, vol. 2, no. 3, pp. 28, 2010.

[136] Sun Microsystems. Lesson: Java Applets, 2012. URL: `http://docs.oracle.com/javase/tutorial/deployment/applet/index.html`.

[137] G. Farias. *JIM Server*, 2010. URL: `http://lab.dia.uned.es/rmatlab`.

[138] G. Farias. *Adding Interactive Human Interface to Engineering Software.* PhD thesis, Departamento de Informática y Automática. Escuela Escuela Téncica Superior de Ingeniería Informática. Universidad Nacional de Educación a Distancia (UNED), Madrid. España, 2010.

[139] G. Farias, R. De Keyser, S. Dormido, and F. Esquembre. Developing networked control labs: a MATLAB and Easy Java Simulations Approach. *IEEE Transactions on Industrial Electronics*, vol. 57, no. 10, pp. 3266–3275, 2010.

[140] Advantech. *PCI-1711/L PCI Multifunction Card*, 2009. URL: `http://www.advantech.com/products`.

[141] MathWorks. *MATLAB/Simulink - Data Acquisition Toolbox 2 User's Guide*, 2009.

[142] MathWorks. *Data Acquisition Toolbox 2.15 MATLAB*, 2010. URL: `http://www.mathworks.es/products/daq/`.

[143] W. Ren, W. Beard, and E. M. Atkins. Information consensus in multi-vehicle cooperative control. *IEEE Control Systems Magazine*, vol. 27, no. 2, pp. 71–82, 2007.

[144] W. Ren, W. Beard, and Randal. *Distributed Consensus in Multi-vehicle Cooperative Control*. Springer, 2008.

[145] H. Kang, J. Kawk, Ch. Kim, and K. Jo. Multiple robot formation keeping and cooperative localization by panoramic view. In *ICROS-SICE International Joint Conference 2009*, pp. 3509–3513, Aug. 2009.

[146] J. Bisson, F. Michaud, and D. Letourneau. Relative positioning of mobile robots using ultrasounds. In *International Conference on Intelligent Robots and Systems*, vol. 2, pp. 1783–1788, Oct. 2003.

[147] D. Chaos. *Control no lineal de vehículos marinos subactuados no-holonómicos*. PhD thesis, Departamento de Informática y Automática. Escuela Téncica Superior de Ingeniería Informática. Universidad Nacional de Educación a Distancia (UNED), Madrid. España, 2010.

[148] E. G. Hernández-Martínez and E. Aranda-Bricaire. Experimental comparison of non-collision strategies in multi-agent robots formation control. In *52nd IEEE International Midwest Symposium on Circuits and Systems*, pp. 321–324, Aug. 2009.

[149] J. A. Hernandez, A. Pustowka, E. F. Caicedo, and E. B. Bacca. Formation control of cooperative robots with limited sensing using a virtual robot as reference. In *6th Latin American Robotics Symposium (LARS)*, pp. 1–7, Oct. 2009.

[150] D. Benedettelli, M. Casini, A. Garulli, A. Giannitrapani, and A. Vicino. A LEGO Mindstorms experimental setup for multi-agent systems. In *IEEE Control Applications Intelligent Control*, pp. 1230–1235, Jul. 2009.

[151] Basler. *Camera A631fc*, 2010. URL: `http://www.baslerweb.com/products/A600.html?model=306&language=en`.

[152] N. Correll, G. Sempo, Y. López de Meneses, J. Halloy, J. Deneubourg, and A. Martinoli. SwisTrack: A Tracking Tool for Multi-Unit Robotic and Biological Systems. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2185–2191, 2006.

[153] T. Lochmatter, P. Roduit, Ch. Cianci, N. Correll, J. Jacot, and A. Martinoli. SwisTrack - A Flexible Open Source Tracking Software for Multi-Agent Systems. In *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems (IROS 2008)*, pp. 4004–4010, 2008.

[154] E. Fabregas, N. Duro, R. Dormido, S. Dormido-Canto, H. Vargas, and S. Dormido. Virtual and Remote Experimentation with the Ball and Hoop System. In *14^{th} IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1061–1068, Sep. 2009. ISBN 978-1-4244-2727-7.

[155] Surveyor Corporation. Definition of the SRV-1 Control Protocol (Blackfin Version), 2010. URL: `http://www.surveyor.com/SRV_protocol.html`.

[156] R. Igual and C. Medrano. *Tutorial de OpenCV*, 2008. URL: `http://docencia-eupt.unizar.es/ctmedra/tutorial_opencv.pdf`.

A

Anexos

A.1 Código *Java* de la aplicación *Cliente BA-S*

Inicialización.

```
computeParameters(); //Calcular los parámetros inherentes al modelo no lineal
computeParametersL();//Calcular los parámetros inherentes al modelo lineal
visible_hoop=false;  //Línea de referencia de posición del aro
//Objeto videocam que obtiene las imágenes de la cámara IP
videocam=new webcam.VideoReceiver(stringURL, "app", isMJPEG, "HTTP", delay);
//Alta prioridad de ejecución para este hilo
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
//Parámetros de control de posición del aro (modelo no lineal)
initControl(pidContVH,old_v,aRefH,Kp_VH,tS,Ti_VH,Td_VH,Ti_VH,10,1,TMin,TMax);
//Parámetros de control de posición del aro (modelo lineal)
initControl(pidContVHL,old_vL,aRefH,Kp_VH,tS,Ti_VH,Td_VH,Ti_VH,10,1,TMin,TMax);
//Parámetros de control de posición de la bola (modelo lineal)
```

```
initControl(pidContBP,old_v,angleRefB,Kp_BP,tS,Ti_BP,Td_BP,Ti_BP,10,1,TMin,TMax);
//Parámetros de control de posición de la bola (modelo no lineal)
initControl(pidContBPL,old_vL,angleRefB,Kp_BP,tS,Ti_BP,Td_BP,Ti_BP,10,1,TMin,TMax);
//Objeto lm para la conexión con Simulink
lm=new jimc.simulink("<matlab:62.204.199.192:2055>BallandHoop.mdl");
lm.setVarContext(this); //Contexto de las variables Java
//Conexión entre las variables Java y bloques Simulink
lm.linkVar("t","BallandHoop","param","t");                    //t-->t
lm.linkVar("aH","BallandHoop/ControlLoop/Sum","in","1");      //aRefH_r ---> Sum in
lm.linkVar("Tau_r","BallandHoop/ControlLoop/Add","out","1");  //Tau_r ---> Add out
lm.linkVar("aH_r","BallandHoop/ControlLoop/KCita","out","1"); //aH_r ---> KCita out
lm.linkVar("chi_r","BallandHoop/ControlLoop/K3","out","1");   //chi_r ---> K3
lm.linkVar("K","BallandHoop","param","K");                    //K ---> K
```

Evolución.

```
if(acc=="hoop") sel=1;      //Seleccionar el experimento a desarrollar
else if(acc=="ball") sel=4; //Control de posición de la bola
else if(acc=="man") sel=3;  //Control manual
else if(acc=="init") sel=2; //Condiciones iniciales
else sel=0;
if(lm.isConnected()){ //Si existe conexión con Simulink
  lm.step(1);         //Se lleva a cabo un paso de simulación
  getVideo();         //Se obtiene una imagen de la cámara IP
}
```

Relaciones Fijas.

```
//Ecuaciones inherentes al modelo no lineal
chi=aH-y/R;          //Ángulo de inclinación de la bola en el modelo no lineal
xB=(R-r)*sin(chi);   //Coordenada X de la posición de la bola en el modelo no lineal
yB=-(R-r)*cos(chi);  //Coordenada Y de la posición de la bola en el modelo no lineal
angleB=y/r;          //Velocidad angular de la bola en el modelo no lineal
hoopX=R*sin(aH);     //Coordenada X de la posición del aro en el modelo no lineal
hoopY=-R*cos(aH);    //Coordenada Y de la posición del aro en el modelo no lineal
yRef=R*aRefH;
//Ecuaciones inherentes al modelo lineal
chiL=aHL-yL/R;         //Ángulo de inclinación de la bola en el modelo lineal
xBL=(R-r)*sin(chiL);   //Coordenada X de la posición de la bola en el modelo lineal
yBL=-(R-r)*cos(chiL);  //Coordenada Y de la posición de la bola en el modelo lineal
angleBL=yL/r;          //Velocidad angular de la bola en el modelo lineal
```

```
hoopXL=R*sin(aHL);     //Coordenada X de la posición del aro en el modelo lineal
hoopYL=-R*cos(aHL);    //Coordenada Y de la posición del aro en el modelo lineal
yRefL=R*aRefH;         //Coordenada Y de la referencia del aro en el modelo lineal
p1x=P1[kxx][0];  //Coordenada X del P1 del LGR
p1y=P1[kxx][1];  //Coordenada Y del P1 del LGR
p2x=P2[kxx][0];  //Coordenada X del P2 del LGR
p2y=P2[kxx][1];  //Coordenada Y del P2 del LGR
p3x=P3[kxx][0];  //Coordenada X del P3 del LGR
p3y=P3[kxx][1];  //Coordenada Y del P3 del LGR
p4x=P4[kxx][0];  //Coordenada X del P4 del LGR
p4y=P4[kxx][1];  //Coordenada Y del P4 del LGR
```

Funciones personalizadas.

```
public void computeParameters(){     //Modelo no lineal
  Ia=M*R*R;                          //Inercia del aro
  Ib=2*m*r*r/5;                      //Inercia de la bola
  a11=Ia+m*(R-r)*(R-r);              //Coeficiente inherente al modelo no lineal
  a12=-m*(R-r)*(R-r)/R;              //Coeficiente inherente al modelo no lineal
  a21=a12;                           //Coeficiente inherente al modelo no lineal
  a22=Ib/(r*r)+m*(R-r)*(R-r)/(R*R); //Coeficiente inherente al modelo no lineal
  Det=a11*a22-a12*a21;               //Determinante
  c1=m*g*(R-r);                      //Coeficiente inherente al modelo no lineal
  c2=-m*g*(R-r)/R;                   //Coeficiente inherente al modelo no lineal
}
public void computeParametersL(){ //Modelo lineal
  double IaL=M*R*R;                //Inercia del aro
  double IbL=2*m*r*r/5;            //Inercia de la bola
  a11L=IaL;                        //Coeficiente inherente al modelo lineal
  a22L=(IbL/m*r*r)+1;              //Coeficiente inherente al modelo lineal
  c1L=m*g*R;                       //Coeficiente inherente al modelo lineal
  c2L=-m*g;                        //Coeficiente inherente al modelo lineal
}
//Ecuación correspondiente a la dinámica de la bola en el modelo no lineal
public double Bola(double t, double aH, double vH, double y, double vy){
 return(a22*(-bm*vH-c1*sin(aH-y/R)+tau(t))-a12*(-bb*vy/(r*r)-c2*sin(aH-y/R)))/Det;
}
//Ecuación correspondiente a la dinámica del aro en el modelo no lineal
public double Aro(double t, double aH, double vH, double y, double vy){
 return(a11*(-bb*vy/(r*r)-c2*sin(aH-y/R))-a21*(-bm*vH-c1*sin(aH-y/R)+tau(t)))/Det;
}
//Ecuación correspondiente a la dinámica de la bola en el modelo lineal
public double BolaL(double t, double aHL, double vHL, double yL, double vyL){
```

```
 return(a22*(-bm*vHL-c1*(aHL-yL/R)+tauL(t))-a12*(-bb*vyL/(r*r)-c2*(aHL-yL/R)))/Det;
}
//Ecuación correspondiente a la dinámica del aro en el modelo lineal
public double AroL(double t, double aHL, double vHL, double yL, double vyL){
 return(a11*(-bb*vyL/(r*r)-c2*(aHL-yL/R))-a21*(-bm*vHL-c1*(aHL-yL/R)+tauL(t)))/Det;
}
public double tau(double t){ //Señal de control del modelo no lineal
  if(acc=="zeros")return amp*(sin(2*PI*f*t));            //Ceros de transmisión
  else if(acc=="init") return 0;                         //Inicio
  else if(acc=="hoop") return(contSig(pidContVH,angH+K*chi,"PI")-0.1*vH); //Aro
  else if(acc=="man") return constantTau;                //Control manual
  else return(contSig(pidContBP,Kc*chi,"PI"))-Kv*vH;     //Posición de la bola
}
public double tauL(double t){ //Señal de control del modelo lineal
  if(acc=="zeros")return amp*(sin(2*PI*f*t));            //Ceros de transmisión
  else if(acc=="init") return 0;                         //Inicio
  else if(acc=="hoop") return(contSig(pidContVHL,angHL+K*chiL,"PI")-0.1*vHL); //Aro
  else if(acc=="man") return constantTau;                //Control manual
  else return(contSig(pidContBPL,Kc*chiL,"PI")-Kv*vHL);  //Posición de la bola
}
//Variable booleana 'pidCont' que indica si se usará acción integral o no
private void initControl(double control[], double init, double ref, double kp, ...
    double ts, double ti, double td, double tt, double n, double b, double min, ...
    double max){
  //Parámetros de entrada
  control[0]=init;  //Valor anterior de la variable controlada
  control[1]=ref;   //Referencia para la variable controlada
  control[2]=kp;    //Constante proporcional del controlador PID
  control[3]=b;     //Acción Proporcional: Kp*(b*ref-value)
  control[4]=min;   //Límite de saturación inferior
  control[5]=max;   //Límite de saturación superior
  //Variables calculadas
  control[6]=kp*ts/ti;          //Bi=Kp*Ts/Ti;
  control[7]=ts/tt;             //Ar=Ts/Tt;
  control[8]=td/(td+n*ts);      //Ad=Td/(Td+N*Ts);
  control[9]=kp*n*control[8];   //Bd=Kp*N*Ad;
  control[10]=0.0;              //I=0.0
  control[11]=0.0;              //D=0.0
}
private double contSig(double control[],double value,String pControll){
  //Acción Proporcional: Kp*(b*ref-value)
  pControl=pControll;
  double P=control[2]*(control[3]*control[1]-value);
  //Acción Derivativa: D=ad*D-Bd*(h-h_old);
  control[11]=control[8]*control[11]-control[9]*(value-control[0]);
```

```
  //Acción Integral
  double output;
  //pControl es una variable global que selecciona la acción de control
  if(pControl=="PD") output=P+control[11];       //P+D
  else if(pControl=="PI") output=P+control[10]; //P+I
  else output=P+control[10]+control[11];         //P+I+D
  //Saturación de la señal de control
  double outputSat;
  if(output<control[4])outputSat=control[4];
  else if(output>control[5]) outputSat=control[5];
  else outputSat=output;
  //Actualiza la acción integral:I=I+Bi*(Hr-h)+Ar*(v-u)
  control[10]=control[10]+control[6]*(control[1]-value)+control[7]*(outputSat-output);
  control[0]=value; //Mantiene el valor para la próxima iteración
  return outputSat;
}
//Función para cambiar los límites de saturación
private void changeSaturation(double control[],double min,double max){
  control[4]=min; //Límite inferior de saturación
  control[5]=max; //Límite superior de saturación
}
//Función para establecer los parámetros de control en la aplicación remota
private void setControlParameters(){
  lm.eval("set_param('BallandHoop/PI','numerator','["+Kp_VH+","+ Ti_VH+"]')");
}
//Función que calcula las raíces del sistema dados el radio del aro y de la bola
public void ComputeRoots(double Ratio, double ratio){
  ReP1=-5*bb/(14*m*ratio*ratio); //Parte real del polo
  ImP1=sqrt(abs((0.5218e-4/ratio*ratio-54.936/Ratio)))/2.8; //Parte imaginaria polo
  ReZ1=0;                        //Parte real del cero
  ImZ1=sqrt(g/Ratio);            //Parte imaginaria del cero
}
public void resizeHoop(){              //Cambiar tamaño del aro modelo no lineal
  R=sqrt(hoopX*hoopX+hoopY*hoopY);     //Nuevo radio del aro
  if(R<6*r) R=6*r;                     //Radio no puede ser menor que 6*r bola
  angH=atan2(hoopX,-hoopY);            //Ángulo de inclinación del aro
  if(v_zeros) changeFrec(R);           //Frecuencia de ceros de transmisión
  else if(v_hoop)changeangleRef(aRefH); //Ángulo de referencia del aro
  else changeBallRef(angleRefB);       //Ángulo de referencia de la bola
}
public void resizeHoopL(){             //Cambiar tamaño del aro modelo lineal
  R=sqrt(hoopXL*hoopXL+hoopYL*hoopYL); //Nuevo radio del aro
  if(R<6*r) R=6*r;                     //Radio no puede ser menor que 6*r bola
  angH=atan2(hoopXL,-hoopYL);          //Ángulo de inclinación del aro
  if(v_zeros) changeFrec(R);           //Frecuencia de ceros de transmisión
```

```
  else if(v_hoop)changeangleRef(aRefH); //Ángulo de referencia del aro
  else changeBallRef(angleRefB);        //Ángulo de referencia de la bola
}
public void moveRef(double xx,double yy){           //Cambiar referencia aro
  if(atan2(yy,xx)≤0) aRefH=PI/2-abs(atan2(yy,xx)); //Ángulo referencia aro
  else aRefH=PI/2+abs(atan2(yy,xx));
}
public void changeBallRef(double angleReff){ //Cambiar referencia bola
  xrB=(R+0.01)*sin(angleReff);   //Coordenada X referencia bola modelo no lineal
  yrB=-(R+0.01)*cos(angleReff);  //Coordenada Y referencia bola modelo no lineal
  xrBL=(R+0.01)*sin(angleReff);  //Coordenada X referencia bola modelo lineal
  yrBL=-(R+0.01)*cos(angleReff); //Coordenada Y referencia bola modelo lineal
}
public void changeangleRef(double angle){ //Cambiar referencia del aro
  xrH=(R+0.01)*sin(angle);   //Coordenada X de referencia del aro modelo no lineal
  yrH=-(R+0.01)*cos(angle);  //Coordenada Y de referencia del aro modelo no lineal
  xrHL=(R+0.01)*sin(angle);  //Coordenada X de referencia del aro modelo lineal
  yrHL=-(R+0.01)*cos(angle); //Coordenada Y de referencia del aro modelo lineal
}
public void changeFrec(double Rat){ //Cambiar frecuencia de ceros de transmisión
  fini=sqrt(g/Rat)/(2*PI);   //Frecuencia de inicio
  fmax=sqrt(g/Rat)/(2*PI)+1; //Frecuencia máxima
  fmin=sqrt(g/Rat)/(2*PI)-1; //Frecuencia mínima
  f=fini;
}
public void ResizeLetters(double d){ //Cambiar distancia entre las letras
  if(d≤0.027){
    PosY_lettersL=R+0.02; //Coordenada Y de las letras modelo lineal
    PosY_letters=R+0.04;  //Coordenada Y de las letras modelo no lineal
  }else{
    PosY_lettersL=R+0.03; //Coordenada Y de las letras modelo lineal
    PosY_letters=R+0.03;  //Coordenada Y de las letras modelo no lineal
  }
}
public void getVideo(){ //Obtiene imágenes de la cámara
  if(((webcam.VideoReceiver)videocam).isConnected()){
    _view.Imagen.setImage(((webcam.VideoReceiver)videocam).getImage());
  }
}
public void connectVideo(){ //Establece conexión con la cámara
  if(!((webcam.VideoReceiver)videocam).isConnected()){
   ((webcam.VideoReceiver)videocam).connect(); //Conectar la cámara
  }
}
public void disconnectVideo(){ //Cierra conexión con la cámara
```

```
  if(((webcam.VideoReceiver)videocam).isConnected()){
    ((webcam.VideoReceiver)videocam).disconnect(); //Desconectar la cámara
  }
}
if(_isPlaying()){//Botón Play
  if(lm.isConnected()){ //Si hay conexión con aplicación remota
    lm.disconnect();    //Se desconecta la aplicación remota
    connected=false;    //Label conexión en falso
    disconnectVideo();  //Se desconecta el vídeo
  }
  _reset();             //Resetea aplicación local
}else{                  //Si no se está ejecutando
  play_test="Stop";     //Botón play = stop
  _play();              //Se inicia la aplicación local
}
//Botón Conectar/Desconectar
if(!lm.isConnected()){    //Si existe conexión con la app remota
  liminftheta=0;          //Se establecen los límites del ángulo del aro
  limsuptheta=250;        //Entre 0-250
  lm.connect();           //Se conecta con la aplicación remota
  connected=true;         //Variable connected = true
  conn_text="Yes";        //Label conectado =  true
  sim_text="Remote";      //Label local o remoto = Remoto
  connectVideo();         //Concetar el vídeo
  conbutton="Disconnect"; //Texto del botón Conectar/Desconectar
  lm.eval("init_ref=get_param('BallandHoop/ControlLoop/Step','gain')"); //Referencia
  angH=lm.getDouble("init_ref");  //Recibe ángulo del aro
  aRefH=(lm.getDouble("init_ref"))*(PI/180);//Obtiene la referencia
  angleRefHmax=3.48;      //200º Referencia máxima aplicación remota
  angleRefHmin=1.047;     //60º  Referencia mínima aplicación remota
}else{//Desconectar
  lm.eval("fintotal=-1"); //Establecer condiciones finales en la aplicación remota
  //Detener aplicación remota
  lm.eval("set_param('BallandHoop/Stop_Var','value','0')");
  disconnectVideo();      //Desconectar vídeo
  do{lm.step(1);          //Último paso de simulación de la app remota
     auxfintotal=lm.getDouble("fintotal");
   }
   while(auxfintotal<=0);
   lm.disconnect();       //Desconetar aplicación remota
   connected=false;       //Variable connect=false
   _reset();              //Resetea aplicación local
}
```

A.2 Código *Java* de la aplicación *Cliente BA-L*

Inicialización

```
computeParameters();  //Calcular los parámetros del modelo no lineal
computeParametersL(); //Calcular los parámetros del modelo lineal
vi=new jil.Jil("<labview:62.204.199.192:2055>"); //Objeto de conexión remota
//Objeto cámara
videocam=new webcam.VideoReceiver(stringURL,"app",isMJPEG,"HTTP",delay);
//Inicialización de los parámetros para las señales de control
initControl(pidContVH,old_v,aRefH,Kp_VH,tS,Ti_VH,Td_VH,Ti_VH,10,1,TMin,TMax);
initControl(pidContVHL,old_vL,aRefH,Kp_VH,tS,Ti_VH,Td_VH,Ti_VH,10,1,TMin,TMax);
initControl(pidContBP,old_v,angleRefB,Kp_BP,tS,Ti_BP,Td_BP,Ti_BP,10,1,TMin,TMax);
initControl(pidContBPL,old_vL,angleRefB,Kp_BP,tS,Ti_BP,Td_BP,Ti_BP,10,1,TMin,TMax);
```

Evolución

```
if(((jil.Jil)vi).isConnected()){ //Si hay conexión con la aplicación remota
 buffer=((jil.Jil)vi).getDataAvailable();       //Obtiene los datos disponibles
 bufferCB=((jil.Jil)vi).getDataAvailableCB();  //Lee los datos y los almacena
 if(((jil.Jil)vi).isRunning()){
   if(buffer>1){
     getValues();                                //Obtiene los valores
   }
 }
 getVideo();                                     //Obtiene el vídeo
}
```

Funciones Personalizadas

```
//Modelo no lineal
public double tau(double t){
  if(acc=="zeros")return amp*(sin(2*PI*f*t)); //Ceros de transmisión
  else if(acc=="init") return 0;              //Inicio
  else if(acc=="hoop") return(contSig(pidContVH,angH+K*chi,"PI")-0.1*vH);
  else if(acc=="man") return constantTau;     //Control manual
  else return(contSig(pidContBP,Kc*chi,"PI"))-Kv*vH; //Bola
}
//Modelo lineal
```

```
public double tauL(double t){
  if(acc=="zeros")return amp*(sin(2*PI*f*t)); //Ceros de transmisión
  else if(acc=="init") return 0;              //Inicio
  else if(acc=="hoop") return(contSig(pidContVHL,angHL+K*chiL,"PI")-0.1*vHL);
  else if(acc=="man") return constantTau;     //Control manual
  else return(contSig(pidContBPL,Kc*chiL,"PI")-Kv*vHL); //Bola
}
public void changeBallRef(double angleReff){
  xrB=(R+0.01)*sin(angleReff);   //Coordeada X de referencia de la bola no lineal
  yrB=-(R+0.01)*cos(angleReff);  //Coordeada Y de referencia de la bola no lineal
  xrBL=(R+0.01)*sin(angleReff);  //Coordeada X de referencia de la bola lineal
  yrBL=-(R+0.01)*cos(angleReff); //Coordeada Y de referencia de la bola lineal
}
public void setValues(){
  ((jil.Jil)vi).setValue("Ring(con)",sel); //Establecer experimento remoto
  ((jil.Jil)vi).setValue("Frec(con)",f);   //Establecer frecuencia remota
}
public void getValues(){
  angH_r=((jil.Jil)vi).getDouble("HA(ind)"); //Obtener ángulo del aro remoto
  chi_r=((jil.Jil)vi).getDouble("BA(ind)");  //Obtener ángulo de la bola remoto
  Tau_r=((jil.Jil)vi).getDouble("U(ind)");   //Obtener señal de control remota
  aRefH_r=((jil.Jil)vi).getDouble("SP_Hoop_r(ind)"); //Referencia del aro remoto
}
public void getVideo(){ //Obtener señal de vídeo
  if(((webcam.VideoReceiver)videocam).isConnected()){
    _view.Imagen.setImage(((webcam.VideoReceiver)videocam).getImage());
  }
}
public void connectVideo(){ //Concetar el vídeo
  if(!((webcam.VideoReceiver)videocam).isConnected()){
   ((webcam.VideoReceiver)videocam).connect();
  }
}
public void disconnectVideo(){ //Desconectar el vídeo
  if(((webcam.VideoReceiver)videocam).isConnected()){
    ((webcam.VideoReceiver)videocam).disconnect();
  }
}
//Botón Conectar/Desconectar
if(!connected){   //Si la aplicación remota no está conectada
do{
  ((jil.Jil)vi).connect(); //Conectar la aplicación remota
    connected=false;
  }while(!((jil.Jil)vi).isConnected()); //Repite el intento si no se logra
  connected=true;          //Conectada
```

```
  conn_text="Yes";          //Label conectada = yes
  sim_text="Remote";        //Modo remoto
  connectVideo();           //Conecta el vídeo
  conbutton="Disconnect"; //Botón concetar
  aRefH=(((jil.Jil)vi).getDouble("HA(ind)"))*(PI/180);
  angleRefHmax=3.48;        //200º Ref Max Remote Lab
  angleRefHmin=1.047;       //60º  Ref Min Remote Lab
  liminftheta=0;
  limsuptheta=280;
  if(((jil.Jil)vi).isConnected()){ //Si hay conexión?
   //Intercambio de variables
   ((jil.Jil)vi).openVI("ball/BallandHoop.vi",exchangedVariables1);
   ((jil.Jil)vi).runVI(false);
   ((jil.Jil)vi).setValue("Ring(con)",sel); //Establecer experimento
  }
}
else{ //Desconectar
 if(((jil.Jil)vi).isConnected()){ //Si hay conexión?
  ((jil.Jil)vi).stopVI();          //Detiene el VI remoto
  ((jil.Jil)vi).closeVI();         //Cierra el VI remoto
  ((jil.Jil)vi).disconnect();      //Desconecta el VI remoto
  disconnectVideo();               //Desconecta el vídeo
  _reset();                        //Resetea la simulación local
 }
}
```

A.3 Código *Java* de la aplicación *Cliente BP-C*

Inicialización.

```
videocam=new webcam.VideoReceiver(stringURL, "app", isMJPEG, "HTTP", delay);
xcen0=0.0;
ycenb0=0.0;
zcenb0=axisAball/2;
//Proyección de la posición de la bola en función de los ángulos
xcenb=(xcenb0+xball)*cos(Alpha)-zcenb0*sin(Alpha);
ycenb=-(xcenb0+xball)*sin(Beta)*sin(Alpha)+(ycenb0+yball)*cos(Beta)-zcenb0 ...
    *sin(Beta)*cos(Alpha);
zcenb=(xcenb0+xball)*cos(Beta)*(Alpha)+(ycenb0+yball)*sin(Beta)+zcenb0 ...
    *cos(Beta)*cos(Alpha);
xu=0;           //Señal de control inicial, eje X
error_x=0;      //Error inicial eje X
```

```
yu=0;           //Señal de control inicial, eje Y
error_y=0;      //Error inicial eje Y
//Posición final de la bola de la referencia lineal
final_x0=0.0;  //Coordenada X
final_y0=0.0;  //Coordenada Y
final_z0=0.0;  //Coordenada Z
xiball=xcendb;
yiball=ycenb;
ziball=zcenb;
//Valores iniciales de la referencia
Refx0=0; //Coordenada X
Refy0=0; //Coordenada Y
Refz0=-sizeRefz+axisCball+plateSizeA/18; //Coordenada Z
Refx=Refx0*cos(Alpha)-Refz0*sin(Alpha);
Refy=-Refx0*sin(Beta)*sin(Alpha)+Refy0*cos(Beta)-Refz0*sin(Beta)*cos(Alpha);
Refz=Refx0*cos(Beta)*sin(Alpha)+Refy0*sin(Beta)+Refz0*cos(Beta)*cos(Alpha);
```

Evolución.

```
if(connected==true){ //Si está conectada la aplicación remota
  getVideo();        //Se obtiene el vídeo
}
arreglo[0]=dirAplate[0]; //Actualización de la posición del plato
arreglo[1]=dirAplate[1];
arreglo[2]=dirAplate[2];
arreglo[3]=dirBplate[0];
arreglo[4]=dirBplate[1];
arreglo[5]=dirBplate[2];
if(manual && TrajectorySelected==0){ //Punto de equilibrio
  error_x=-(Refx0-xball);            //Error en X
  Alpha=(Kpx*(Nx+1)*error_x+xu);     //Ángulo en X
}
else if(manual && TrajectorySelected==1){  //Trayectoria Lineal
  Refx0=pos_actual_x+(final_x-pos_actual_x)*time_traj_x/duration_lineal;
  error_x=-(Refx0-xball);            //Error en X
  Alpha=(Kpx*(Nx+1)*error_x+xu);     //Ángulo en X
  time_traj_x=time_traj_x+dtx;       //Duración de la trayectoria
  if(time_traj_x≥duration_lineal){   //Si la duración ha terminado
    _pause();                        //Detener la simulación
  }
}
else if(manual && TrajectorySelected==2){  //Trayectoria Circular
  Refx0_singiro=radiuscircle*cos(2*freqcircle*PI*time_traj_x)-radiuscircle;
```

```
  Refx0=Refx0_singiro*cos(a_giro)+Refy0_singiro*sin(a_giro)+pos_actual_x;
  error_x=-(Refx0-xball);                    //Error en X
  Alpha=(Kpx*(Nx+1)*error_x+xu);             //Ángulo en X
  time_traj_x=time_traj_x+dtx;               //Duración de la trayectoria
  if(time_traj_x≥duration_circular){        //Si la duración ha terminado
   _pause();                                 //Detener la simulación
  }
  paintCircular();                           //Dibuja la trayectoria
}
else if(manual && TrajectorySelected==3){ //Trayectoria de Lissajous
  Refx0=amplitudeLissa*sin(2*freqLissa_1*PI*time_traj_x)-amplitudeLissa;
  error_x=-(Refx0-xball);                   //Error en X
  Alpha=(Kpx*(Nx+1)*error_x+xu);            //Ángulo en X
  time_traj_x=time_traj_x+dtx;              //Duración de la trayectoria
  if(time_traj_x≥duration_lissajous){    //Si la duración ha terminado
    _pause();                               //Detener la simulación
  }
  paintLissajous();                         //Dibuja la trayectoria
}
```

Relaciones Fijas

```
//Actualización de la posición del plano y de las flechas
updatePlaneAndArrows();
updateSurface();
//La posición de la bola está limitada al tamaño del plato. La velocidad de la ...
    bola es cero cuando esta alcanza el borde del plato
if(xball>xmax){xball=xmax; dxb=0;}     //Vxball=0 si xball=xmax
if(yball>ymax){yball=ymax; dyb=0;}     //Vyball=0 si yball=ymax
if(xball<-xmax){xball=-xmax; dxb=0;}  //Vxball=0 si xball<xmin
if(yball<-ymax){yball=-ymax; dyb=0;}  //Vyball=0 si yball<ymin
xcenb=(xcenb0+xball)*cos(Alpha)-zcenb0*sin(Alpha);
ycenb=-(xcenb0+xball)*sin(Beta)*sin(Alpha)+(ycenb0+yball)*cos(Beta) ...
    -zcenb0*sin(Beta)*cos(Alpha);
zcenb=(xcenb0+xball)*cos(Beta)*sin(Alpha)+(ycenb0+yball)*sin(Beta) ...
    +zcenb0*cos(Beta)*cos(Alpha);
xiball=xcenb;
yiball=ycenb;
ziball=zcenb;
//La referencia está limitada al tamaño del plato
if(Refx>xmax){Refx=xmax;}
if(Refy>ymax){Refy=ymax;}
if(Refx<-xmax){Refx=-xmax;}
```

```
if(Refy<-ymax){Refy=-ymax;}
//Ligaduras para la altura de la Ref
Refz=Refx0*cos(Beta)*sin(Alpha)+Refy0*sin(Beta)+ Refz0*cos(Beta)*cos(Alpha);
//La referencia lineal está limitada al tamaño del plato
if(final_x>xmax){final_x=xmax;}
if(final_y>ymax){final_y=ymax;}
if(final_x<-xmax){final_x=-xmax;}
if(final_y<-ymax){final_y=-ymax;}
//Ligaduras para la altura y proyección de la Partícula de referencia Lineal
final_z=final_x*cos(Beta)*sin(Alpha)+final_y*sin(Beta);
//Pseudo-friction (Fricción de rodamiento de la bola en el plato se simula ...
    estableciendo la velocidad de la bola a cero cuando la velocidad de la bola y ...
    del plato son suficientemente pequeñas)
if((abs(Alpha)<0.02)&&(abs(dxb)<0.002)){
  dxb=0;
}
if((abs(Beta)<0.02)&&(abs(dxb)<0.002)){
  dxb=0;
}
//Proyección de la trayectoria lineal
if(TrajectorySelected==1){
 ΔT=1.0/NpointsTrajectory;
 for(int i=0;i<NpointsTrajectory;i++){
  traj_x[i]=traj_x0[i]*cos(Alpha)-traj_z0[i]*sin(Alpha);
  traj_y[i]=-traj_x0[i]*sin(Beta)*sin(Alpha)+traj_y0[i]*cos(Beta)-traj_z0[i] ...
      *sin(Beta)*cos(Alpha);
  traj_z[i]=traj_x0[i]*cos(Beta)*sin(Alpha)+traj_y0[i]*sin(Beta)+traj_z0[i] ...
      *cos(Beta)*cos(Alpha);
 }
}
//Proyección de la trayectoria circular
if(TrajectorySelected==2){
 ΔT=duration_circular/NpointsTrajectory;
 for(int i=0; i<NpointsTrajectory; i++){
  traj_x[i]=traj_x0[i]*cos(Alpha)-traj_z0[i]*sin(Alpha);
  traj_y[i]=-traj_x0[i]*sin(Beta)*sin(Alpha)+traj_y0[i]*cos(Beta)-traj_z0[i] ...
      *sin(Beta)*cos(Alpha);
  traj_z[i]=traj_x0[i]*cos(Beta)*sin(Alpha)+traj_y0[i]*sin(Beta)+traj_z0[i] ...
      *cos(Beta)*cos(Alpha);
  auxt=auxt+ΔT;
 }
}
//Proyección de la trayectoria de Lissajous
if(TrajectorySelected==3){
 NpointsTrajectory=NpointsTrajectory;
```

```
 ΔT=duration_lissajous/NpointsTrajectory;
 for(int i=0; i<NpointsTrajectory; i++){
  traj_x[i]=traj_x0[i]*cos(Alpha)-traj_z0[i]*sin(Alpha);
  traj_y[i]=-traj_x0[i]*sin(Beta)*sin(Alpha)+traj_y0[i]*cos(Beta)-traj_z0[i] ...
      *sin(Beta)*cos(Alpha);
  traj_z[i]=traj_x0[i]*cos(Beta)*sin(Alpha)+traj_y0[i]*sin(Beta)+traj_z0[i] ...
      *cos(Beta)*cos(Alpha);
  auxt=auxt+ΔT;
 }
}
```

Funciones Personalizadas

```
//Función que construye la cadena de valores a enviar a la aplicación remota
public String chain_builder(int a, int StStp, int Path, int KpxFlag, int Kpxx, ...
    int KdxFlag, int Kdxx, int KpyFlag, int Kpyy, int KdyFlag, int Kdyy, int ...
    ChirpFlag, int EqPt, int StartmyTimer, int SendPulse, int RefFlag, int xref, ...
    int yref, int Ampp){
  String chain;
  chain=IntToSt(a)+";"+IntToSt(StStp)+";"+IntToSt(Path)+";"+IntToSt(KpxFlag) ...
      +";"+IntToSt(Kpxx)+";"+IntToSt(KdxFlag)+";"+IntToSt(Kdxx)+";" ...
      +IntToSt(KpyFlag)+";"+IntToSt(Kpyy)+";"+IntToSt(KdyFlag)+";" ...
      +IntToSt(Kdyy)+";"+IntToSt(ChirpFlag)+";"+IntToSt(EqPt)+";" ...
      +IntToSt(StartmyTimer)+";"+IntToSt(SendPulse)+";" ...
      +IntToSt(RefFlag)+";"+IntToSt(xref)+";"+IntToSt(yref)+";"+IntToSt(Ampp)+";";
  System.out.println(chain);
  return chain;
}
public void ChangePlateAnglex(){ //Modifica el ángulo del plato en el eje X
  Alpha=atan2(zArrowx, xArrowx);
  if(Alpha>thetaxmax) Alpha=thetaxmax;
  if(Alpha<-thetaxmax) Alpha=-thetaxmax;
}
public void ChangePlateAngley(){ //Modifica el ángulo del plato en el eje Y
  Beta=atan2(zArrowy, yArrowy);
  if(Beta>thetaymax) Beta=thetaymax;
  if(Beta<-thetaymax) Beta=-thetaymax;
}
public void updatePlaneAndArrows(){ //Actualiza la posición del plato
  if(Alpha>thetaxmax) Alpha=thetaxmax;  //Límite superior de Alpha
  if(Alpha<-thetaxmax) Alpha=-thetaxmax;//Límite inferior de Alpha
  if(Beta>thetaymax) Beta=thetaymax;    //Límite superior de Beta
  if(Beta<-thetaymax) Beta=-thetaymax;  //Límite inferior de Beta
```

```
  dirAplate[0]=cos(Alpha);
  dirAplate[1]=0;
  dirAplate[2]=sin(Alpha);
  dirBplate[0]=0;
  dirBplate[1]=cos(Beta);
  dirBplate[2]=sin(Beta);
  xArrowx=dirAplate[0]*plateSizeA/2;  //Flecha x en el eje x
  yArrowx=dirAplate[1]*plateSizeA/2;  //Flecha x en el eje y
  zArrowx=dirAplate[2]*plateSizeA/2;  //Flecha x en el eje z
  xArrowy=dirBplate[0]*plateSizeB/2;  //Flecha y en el eje x
  yArrowy=dirBplate[1]*plateSizeB/2;  //Flecha y en el eje y
  zArrowy=dirBplate[2]*plateSizeB/2;  //Flecha y en el eje z
  plateX=-dirAplate[0]*plateSizeA/2-dirBplate[0]*plateSizeB/2;
  plateY=-dirAplate[1]*plateSizeA/2-dirBplate[1]*plateSizeB/2;
  plateZ=-dirAplate[2]*plateSizeA/2-dirBplate[2]*plateSizeB/2;
}
public void ChangeBallPos(){  //Cambia la posición de la bola
  xball=xiball;
  yball=yiball;
  xb=xiball/-9.81*(5.0/7.0);
  yb=yiball/-9.81*(5.0/7.0);
}
public void ChangeReference(){ //Cambia la referencia
  Refx0=Refx;
  Refy0=Refy;
}
public String IntToSt(int i){ //Tipo de dato de entero a String
  String a=Integer.toString(i);
  return a;
}
```

A.4 Código *Visual C#* de la aplicación *Servidor BP-C*

Función *startServer*

```
private bool StartServer(){  //Inicia la aplicación servidor
  IPAddress ipAddress=Dns.GetHostEntry("localhost").AddressList[0];
  try{  //"Escucha" por el puerto TCP 1238
    tcpListener=new TcpListener(IPAddress.Any,1238);
    tcpListener.Start(); //Inicia la "Escucha"
    tcpListener.BeginAcceptTcpClient(new ...
        AsyncCallback(this.ProcessEvents),tcpListener);
```

```
  }
  catch(Exception e){ //Si ocurre una excepción
    Console.WriteLine(e.ToString()); //Se muestra el mensaje correspondiente
    return false;      //Servidor no inciado
  }
  return true;  //Servidor iniciado
}
```

Función *ProcessEvents*

```
//Función que procesa el paquete recibido de EJS
private void ProcessEvents(IAsyncResult asyn){
 trigger=false;
 int answer=0;
 try{
  TcpListener processListen=(TcpListener)asyn.AsyncState;
  TcpClient tcpClient=processListen.EndAcceptTcpClient(asyn); //Acepta cliente
  NetworkStream myStream=tcpClient.GetStream(); //Obtiene el paquete por TCP
  Stream s=tcpClient.GetStream();
  if(myStream.CanRead){  //Lee la cadena de datos
    StreamReader readerStream=new StreamReader(myStream);
    string myMessage=readerStream.ReadLine();
    answer=readEJS(myMessage);  //Procesa la cadena de datos recibida
    if(tcpClient.Connected){    //Si el cliente precisa de respuesta?
      if(answer==1){ //Planta conectada, aplicación remota iniciada
        byte[] toSend=Encoding.ASCII.GetBytes("0"+";"+"1"+";"+"\r\n");
        s.Write(toSend,0,toSend.Length); //Envío de la respuesta
      }
      else if(answer==2){  //Señal enviada a la planta real
        byte[] toSend=Encoding.ASCII.GetBytes("0"+";"+"2"+";"+"\r\n");
        while(!trigger){};
        s.Write(toSend,0,toSend.Length); //Envío de la respuesta
      }else if(answer==3){  //Error
        byte[] toSend=Encoding.ASCII.GetBytes("0"+";"+"3"+";"+"\r\n");
        s.Write(toSend,0,toSend.Length); //Envío de la respuesta
      }
    }
    readerStream.Close();  //Se cierra el objeto de comunicaciones TCP
    myStream.Close();
    tcpClient.Close();      //Se cierra el Cliente TCP
    tcpListener.BeginAcceptTcpClient(new ...
        AsyncCallback(this.ProcessEvents),tcpListener);
  }
```

```
}
catch(Exception e){
  Console.WriteLine(e.ToString());
 }
}
```

Función *readEJS*

```
//Función que analiza la cadena recibida de EJS
public int readEJS(string cadena){
int response=0;
portSelected=false;
byte[] bejs=new byte[1024*10]; //Reserva espacio para recibir la cadena
BinaryWriter Writer=null;
string Name=@"./studentData/Data/y.txt"; //Path para de la señal de identificación
if(cadena.Length>1000){ //Si la cadena tiene más de 1000 caracteres
  counter2=0; //Indica que se está recibiendo la señal de identificación
  SetControlPropertyValue(txtChirpCounter,"Text",counter2.ToString());
  try{
   SetControlPropertyValue(txtDebug,"Text"," ");
   bejs=Encoding.ASCII.GetBytes(cadena); //Recibe la cadena en bytes
   Writer=new BinaryWriter(File.OpenWrite(Name));
   Writer.Write(bejs);  //Guarda los bytes recibidos
   Writer.Flush();      //Limpia el buffer de recepción
   Writer.Close();      //Cierra el buffer de recepción
  }
  catch{} //Lee fichero para identificación.
  StreamReader tr=new StreamReader(@"./studentData/Data/y.txt");
  int i=0;
  int indexOfTab=0;
  string number="",filestring="";
  chirparray=new double[1024*10];
  filestring=tr.ReadToEnd(); //Lee los bytes recibidos
  while(indexOfTab!=-1){
   indexOfTab=filestring.IndexOf('\t');
    if(indexOfTab!=-1){
     number=filestring.Substring(0,indexOfTab);
     chirparray[i]=Convert.ToDouble(number);
     SetControlPropertyValue(txtDebug,"Text",txtDebug.Text+ ...
         chirparray[i].ToString()+"\r\n");
     filestring=filestring.Substring(indexOfTab+1,filestring.Length-indexOfTab-1);
     i++;
    }
```

```
  }
}else{ //Cadena recibida (establecer valores de control)
    string[] datos=cadena.Split(';'); //Divide la cadena
    if(cadena.Length>3){   //La cadena recibida tiene que ser > 3
     if(Convert.ToSByte(datos[1])==1&&!isRunning){//1000(like btn.click())
      if(Convert.ToSByte(datos[2])==1){ //Referencia "Center": 021000000
       setpointType=SetpointType.center;
      }
      if(!portSelected){ //Lee los puertos COM disponibles
        String[] Ports=System.IO.Ports.SerialPort.GetPortNames();
        for(inti=0;i<Ports.Length;i++){
         if(Ports[i]!="COM4")continue; //El puerto COM tiene que ser el 4
          platform.setComPort("COM4");
          portSelected=true;  //Puerto seleccionado con éxito
          break;
         }
        } //Botón Inicio
      SetControlPropertyValue(this.buttonStartController,"Text","Stop");
      mmTimer.Start(); //Iniciar el Timer porque la planta está detenida
      checkBoxEnablePlatform.Checked=true;
      platform.enablePlatform(true);
      platform.setDeltaZ(-46); //Ángulo inicial del plato en Z
      isRunning=true;
    }    //Botón Stop 0000000000
    elseif(Convert.ToSByte(datos[1])==0 && isRunning==true){
      SetControlPropertyValue(this.buttonStartController,"Text","Start");
      mmTimer.Stop(); //Detiene el Timer de control
      platform.setThetaX(0);
      platform.setThetaY(0);
      isRunning=false;
      using(CsvWriter writer=new CsvWriter()){  //Guarda los Datos del experimento
        writer.WriteCsv(csvFile,@" ...
            ./studentData/Data/ExperimentData.csv",Encoding.Default);
      }
    }
    //Referencia "Centro": 021000000
    else if(Convert.ToSByte(datos[2])==1&&isRunning){
      setpointType=SetpointType.center;
    }
    //Referencia "Círculo": 022000000
    else if(Convert.ToSByte(datos[2])==2&&isRunning==true){
      setpointType=SetpointType.circle;
    }
    //Referencia "Cuadrado": 023000000
    else if(Convert.ToSByte(datos[2])==3&&isRunning==true){
```

```
    setpointType=SetpointType.square;
  }
  //Referencia "Punto de entrada": 024000000
  else if(Convert.ToSByte(datos[2])==4&&isRunning==true){
    setpointType=SetpointType.input;
  }
  //Establece parámetros del PID.x
  else if(Convert.ToSByte(datos[3])==1 && isRunning==true){
    PID_x.Kp=(float.Parse(datos[4])/10000);  //Kp.x
    PID_x.Kd=(float.Parse(datos[5])/10000);  //Kd.x
    PID_x.Ki=(float.Parse(datos[6])/10000);  //Ki.x
  }
  //Establece parámetros del PID.y
  else if(Convert.ToSByte(datos[7])==1 && isRunning==true){
    PID_y.Kp=(float.Parse(datos[8])/10000);  //Kp.y
    PID_y.Kd=(float.Parse(datos[9])/10000);  //Kd.y
    PID_y.Ki=(float.Parse(datos[10])/10000); //Ki.y
  }
  //Enviar señal de identificación
  else if(Convert.ToSByte(datos[11])==1){
    if(!chirpTimer.IsRunning){
      String[] Ports=System.IO.Ports.SerialPort.GetPortNames();
      for(inti=0; i<Ports.Length; i++){ //Selecciona el puerto
       if(Ports[i]!="COM4")continue;
          platform.setComPort("COM4");
          portSelected=true;
          break;
        }
        //Si el puerto está seleccionado envía la señal de identificación
        if(portSelected==true){startChirp();response=2;}
        else{response=3;}
      }
  }else if(Convert.ToSByte(datos[12])==1){ //Connectar
    String[] Ports=System.IO.Ports.SerialPort.GetPortNames();
    for(int i=0; i<Ports.Length; i++){
      if(Ports[i]!="COM4")continue;
        platform.setComPort("COM4");
        portSelected=true;
        break;
      }
      if(portSelected){response=1;}
      else{portSelected=false;response=3;} //Error
  }else if(Convert.ToSByte(datos[15])==1){  //Enviar referencia
    try{
       double x=Convert.ToDouble(textBoxXsetpoint.Text);
```

```
          double y=Convert.ToDouble(textBoxYsetpoint.Text);
          Geometry.Point point=new ...
              Geometry.Point(double.Parse(datos[16])/10,double.Parse(datos[17])/10);
          double r=Geometry.Point.distance(point,new Geometry.Point(0,0));
          if(r<200){ //Si el cursor está dentro del plato
            //Obtiene las coordenadas del punto
            lastMouseClick=new ...
                Geometry.Point(double.Parse(datos[16])/10,double.Parse(datos[17])/10);
          }
         }
         catch(Exception){ //Si ocurre una excepción
          MessageBox.Show("Input a decimal number");
         }
      }
   }
}
Array.Clear(bejs,0,bejs.Length);//"Limpia" el buffer de recepción
cadena="";
return response; //Devuelve el resultado de la operación
}
```

Función *startChirp*

```
//Función que envía la señal de identificación a la planta real
private void startChirp(){
 long count=0;
 setpointType=SetpointType.center; //Posiciona la bola en el centro del plato
 SetControlPropertyValue(this.setpointCenterRadioButton,"Checked",true);
 SetControlPropertyValue(this.setpointSquareRadioButton,"Checked",false);
 SetControlPropertyValue(this.setpointCircleRadioButton,"Checked",false);
 SetControlPropertyValue(this.setpointInputRadioButton,"Checked",false);
 SetControlPropertyValue(this.buttonStartController,"Text","Stop");//Stop button;
 mmTimer.Start(); //Inicia el controlador de posición de la bola
 while(count<4000000000){count++;} //Espera a que el control termine
 mmTimer.Stop();  //Detiene el controlador
 SetControlPropertyValue(this.buttonStartController,"Text","Start");//Start button;
 chirpTimer.Start(); //Envía la señal de identificación
}
```

A.5 Código *Java* de la aplicación *Cliente MW*

Inicialización

```
//Crea el Socket
sktSwiss=new Socket("62.204.199.192",3000); //Crea Socket por el puerto 3000
inFromServer=new BufferedReader(new InputStreamReader(sktSwiss.getInputStream()));
String[] Method={"VFH+","VFH","VFF"}; //Métodos de evasión de obstáculos
T=360/S;  //Cantidad de sectores alrededor del maestro
K=2*PI/T;
C=W*T/360;
initHmTlTh(); //Inicializacion del histograma
//Inicialización de las velocidades de los obstáculos
for(int i=0;i<numberOfObstacles;i++){
  velocityObstaclesX[i]=(i+1)*0.1;
  velocityObstaclesY[i]=(i+1)*0.1;
  flag[i]=false;
}
//Coloreo de los sectores en verde
for(int b=0;b<T;b++){
 Sk[b]=b*(360/T);
 sectorcolor[b]=green;
 Hbkprevi[b]=0;
 Hbkprev[b]=0.0;
}
InitSectors(); //Inicializar los sectores
setPosition("Obstacles"); //Posiciones iniciales de los obstáculos
//Posiciones iniciales de los robots
for(int r=0;r<numberOfRobots;r++){
  visiblesetmoway[r]=true;
  if(circle==true){ //Si la posición inicial es Círculo
    positionRobotsX[r]=R*cos(r*2*PI/numberOfRobots)+x1; //X=A=R*cos(r*360/Nº Agentes)
    positionRobotsY[r]=R*sin(r*2*PI/numberOfRobots)+y1; //Y=B=R*sin(r*360/Nº Agentes)
  }else if(row){ //Línea Horizontal (S4)---(S2)---(M)---(S1)---(S3)
     positionRobotsY[r]=y1;
     if(r%2==0){positionRobotsX[r]=x1+(r+2);}
     else{positionRobotsY[r]=x1-(r+2);}
  }else{ //Si la posición inicial es Free
    for(int b=0;b<numberOfRobots;b++){
      if(r<4){ //Menos de 4 robots
        positionRobotsX[r]=R*cos(r*2*PI/4)+x1;
        positionRobotsY[r]=R*sin(r*2*PI/4)+y1;
      }else{ //Más de 4 robots
        positionRobotsX[r]=(R+2)*cos(r*2*PI/4)+x1;
```

```
        positionRobotsY[r]=(R+2)*sin(r*2*PI/4)+y1;
      }
    }
  }
  //Distancias y ángulos de referencia iniciales
  dref[r]=dgcalcule(x1,y1,positionRobotsX[r],positionRobotsY[r],0,0);
  angleref[r]=alphacalcule(x1,y1,positionRobotsX[r],positionRobotsY[r],0,0);
  }
```

Evolución

```
if(((webcam.VideoReceiver)videocam).isConnected()){
  getVideo(); //Obiene la señal de vídeo
   try{
     String  sentence=inFromServer.readLine(); //Obtiene los datos de SwisTrack
     processSwiss(sentence);
   }
   catch(Exception e){ //Si ocurre un error
     System.out.print("No se pudo procesar el SwisTrack");
   }
}else{ //Si no hay conexión
 if(cantRobots<=2)formationVisible=false;
 else formationVisible=true;
 th1=th1%(2*PI); //Reduce th1 a 0-360
 if(th1<0){th1=th1+2*PI;} //Si th1<0 lo convierte a positivo
 //Ángulo y distancia desde los robots hasta los obstáculos
 for(int i=0;i<numberOfObstacles;i++){
 distanceObstacles[i]=dgcalcule(x1,y1,positionObstaclesX[i],positionObstaclesY[i],0,0);
 angleObstacles[i]=alphacalcule(x1,y1,positionObstaclesX[i],positionObstaclesY[i],0,0);
 }
 for(int rbt=0;rbt<numberOfRobots;rbt++){
  orientationRobotsTH_PI4[rbt]=orientationRobotsTH[rbt]-PI/4;
  robotsAlPHA_PI4[rbt]=robotsAlPHA[rbt]-PI/4;
  distanceRobots[rbt]=dgcalcule(x1,y1,positionRobotsX[rbt],positionRobotsY[rbt],0,0);
  angleRobots[rbt]=alphacalcule(x1,y1,positionRobotsX[rbt],positionRobotsY[rbt],0,0);
 }
 //Construyendo un array Global con todas las posiciones, ángulos....
 System.arraycopy(positionObstaclesX,0,posGlobalX,0,numberOfObstacles);
 System.arraycopy(positionObstaclesY,0,posGlobalY,0,numberOfObstacles);
 System.arraycopy(angleObstacles,0,angleGlobal,0,numberOfObstacles);
 System.arraycopy(distanceObstacles,0,distanceGlobal,0,numberOfObstacles);
 System.arraycopy(positionRobotsX,0,posGlobalX,numberOfObstacles,numberOfRobots);
 System.arraycopy(positionRobotsY,0,posGlobalY,numberOfObstacles,numberOfRobots);
```

```
System.arraycopy(angleRobots,0,angleGlobal,numberOfObstacles,numberOfRobots);
System.arraycopy(distanceRobots,0,distanceGlobal,numberOfObstacles,numberOfRobots);
for(int r=0;r<numberOfRobots+numberOfObstacles;r++){
 phiRightGlobal[r]=phircalc(th1,angleGlobal[r],posGlobalX[r],posGlobalY[r],x1,y1);
 phiLeftGlobal[r]=philcalc(th1,angleGlobal[r],posGlobalX[r],posGlobalY[r],x1,y1);
}
for(int j=0;j<T;j++){
 sectorcolor[j]=green; //Inicializa los sectores, el histograma y los sectores
 Hpk[j]=0;  //Histograma primario
 Hbk[j]=0;  //Histograma binario
 Hbki[j]=0; //Histograma binario con histéresis
 Hm[j]=0;   //Histograma enmascarado
}
alpha1=alphacalcule(x1,y1,xg,yg,0,0); //Ángulo del Robot 1 al Goal
if(activarenmascarado){ //Si está habilitada la evasión de obstáculos
 //Para esquivar los obstáculos y los robots
 for(int i=0;i<numberOfObstacles+numberOfRobots;i++){
  if(distanceGlobal[i]>ws) continue;
  if(Method=="VFH+"){
   gamma=asin(marginObstacles/distanceGlobal[i]); //Sectores ocupados
  }else if(Method=="VFH"){
   gamma=asin(radioObstacle/distanceGlobal[i]);  //Sectores ocupados
  }
  resta=0; //Inicializa sectores superiores
  suma=0;  //Inicializa sectores inferiores
  for(int j=0;j<T;j++){
  //Inicializa los sectores, el histograma y el color de los sectores en verde
  sectorcolor[j]=green; //Sector libre en verde
  }
  resta=angleGlobal[i]-gamma; //Sectores inferiores
  suma=angleGlobal[i]+gamma;  //Sectores superiores
  if(suma>2*PI) suma=suma-2*PI;
  if(resta<0) resta=resta+2*PI;
  if(resta<suma){
   for(int j=(int)(resta/K);j<=(int)(suma/K);j++){
    sectorcolor[j]=red; //Sector ocupado en rojo
    Hpk[j]=(a-(b*pow(distanceGlobal[i],2))); //Histograma Polar Primario
    Hpgk[j]=2*Hpk[j];   //Para dibujar el gráfico
   }
   }else{
    for(int j=0;j<=(int)(suma/K);j++){
     sectorcolor[j]=red; //Sector ocupado en rojo
     Hpk[j]=(a-(b*pow(distanceGlobal[i],2))); //Histograma Polar Primario
     Hpgk[j]=2*Hpk[j];   //Para dibujar el gráfico
    }
```

```
    for(int j=(int)(resta/K);j<T;j++){
     sectorcolor[j]=red;  //Sector ocupado en rojo
     Hpk[j]=(a-(b*pow(distanceGlobal[i],2))); //Hp Histograma Polar Primario
     Hpgk[j]=2*Hpk[j];     //Para dibujar el gráfico
    }
   }
    for(int j=0;j<T;j++){
     //Hp Histograma Polar Primario con histéresis
     if(Hpk[j]≥TauH){Hbk[j]=1.0;Hbki[j]=1;}
     //Hb Histograma Polar Binario con histéresis
     else if(Hpk[j]≤TauL){Hbk[j]=0.0;Hbki[j]=0;}
     //Recupera los histogramas de la iteración anterior
     else{Hbk[j]=Hbkprev[j]; Hbki[j]=Hbkprevi[j];}
     //Guarda ambos histogramas para la siguiente iteración
     Hbkprev[j]=Hbk[j];
     Hbkprevi[j]=Hbki[j];
    }
    for(int j=0;j<T;j++){
     if(Hbk[j]==0){
     double th1aux=th1;
     if(th1aux<0) th1aux=th1aux+2*PI;
     if(pertenecelibre(K*j,phiRightGlobal[i],th1aux,phiLeftGlobal[i])) Hm[j]=0;
     else Hm[j]=1;  //Hm Histograma enmascarado con histéresis
    }else Hm[j]=1;  //Resto de sectores ocupados
   }
}
    for(int j=0;j<T;j++){
     Hmk[j]=(double)Hm[j];
     if(Hm[j]==1) sectorcolor[j]=red; //Sector ocupado
     else sectorcolor[j]=green;       //Sector libre
    }
    for(int q=0;q<numberOfObstacles+numberOfRobots;q++){
     if(distanceGlobal[q]<ws){ //Si está dentro del rango de visión?
     widsek=wide_valley(Hm,(int)(alpha1/K)); //Busca el sector candidato
     if(trapped){ //Si condición de atrapado habilitada?
       int randomIndex=generator.nextInt(20)-10; //Sector aleatorio
       if((widsek+randomIndex<T)&&(widsek+randomIndex≥0)){
         widsek=widsek+randomIndex; //Asigna sector aleatorio
       }
     }
     alpha1=(widsek*K); //Ángulo destino = sector candidato aleatorio
     if(alpha1<0) break;
    }
   }
}
```

```
//Distancia del Robot 1 al Pto destino
dg1=dgcalcule(x1,y1,xg,yg,0,0);
//Ley de control w
if(consensus){ //Control para consenso
  v1=controlv1plus_consensus(dg1,th1,alpha1);
  w1=control_w_consensus(dg1,th1,alpha1);
} else{ //Control para ley de control VFH
  v1=controlv1plus(dg1, Wmax, v1min, Hpk[(int)((180*th1/PI)/S)],w1,v1max);
  w1=control_w(dg1,th1,alpha1);
  if(abs(v1/w1)<minTurningRatio) w1=v1/minTurningRatio;
}
for(int rw=0;rw<numberOfRobots;rw++){
  orientationRobotsTH[rw]=orientationRobotsTH[rw]%(2*PI);
  if(orientationRobotsTH[rw]<0){
    orientationRobotsTH[rw]=orientationRobotsTH[rw]+2*PI;
  }
}
//Distancia y ángulo de cada robots a cada obstáculo
for(int ro=0;ro<numberOfRobots;ro++){
 for(int ob=0;ob<numberOfObstacles;ob++){
  distanceObstacless[ob][ro]=dgcalcule(positionRobotsX[ro], ...
      positionRobotsY[ro],positionObstaclesX[ob],positionObstaclesY[ob],0,0);
  angleObstacless[ob][ro]=alphacalcule(positionRobotsX[ro], ...
      positionRobotsY[ro],positionObstaclesX[ob],positionObstaclesY[ob],0,0);
 }
}
//Distancia y ángulo de cada robots a cada robot
for(int rot=0;rot<numberOfRobots;rot++){
 for(rott=0;rott<numberOfRobots;rott++){
  if(rot==rott) continue;
  else{
    distanceRobotss[rott][rot]=dgcalcule(positionRobotsX[rot], ...
        positionRobotsY[rot],positionRobotsX[rott],positionRobotsY[rott],0,0);
    angleRobotss[rott][rot]=alphacalcule(positionRobotsX[rot], ...
        positionRobotsY[rot],positionRobotsX[rott],positionRobotsY[rott],0,0);
  }
 }
 distanceRobotss[rott][rot]=dgcalcule(positionRobotsX[rot], ...
     positionRobotsY[rot],x1,y1,0,0);
 angleRobotss[rott][rot]=alphacalcule(positionRobotsX[rot], ...
     positionRobotsY[rot],x1,y1,0,0);
}
//Uniendo las dos matrices.
distanceGlobalss=append(distanceObstacless,distanceRobotss);
angleGlobalss=append(angleObstacless,angleRobotss);
```

```
for(int b=0;b<numberOfRobots+numberOfObstacles+1;b++){
  for(int h=0;h<numberOfRobots;h++){
    if((b-numberOfObstacles)==h) continue;
      phiRightGlobalss[b][h]=phircalc(orientationRobotsTH[h], ...
          angleGlobalss[b][h],posGlobalX[b],posGlobalY[b], ...
          positionRobotsX[h],positionRobotsY[h]);
      phiLeftGlobalss[b][h]=philcalc(orientationRobotsTH[h], ...
          angleGlobalss[b][h],posGlobalX[b],posGlobalY[b], ...
          positionRobotsX[h],positionRobotsY[h]);
    }
  }
 for(int b=0;b<numberOfRobots;b++){
  for(int h=0;h<T;h++){
   //Inicializa los sectores, el histograma y el color de los sectores en verde
   sectorcolors[b][h]=green; //Sector libre
   Hpks[b][h]=0;    //Histograma polar primario
   Hbks[b][h]=0;    //Histograma polar binario
   Hbkis[b][h]=0;   //Histograma binario con histéresis
   Hms[b][h]=0;     //Histograma enmascarado
  }
 }
 for(int rt=0;rt<numberOfRobots;rt++){
  //Ángulo y distancia al pto destino
  robotsAlPHA[rt]=alphacalcule(positionRobotsX[rt],positionRobotsY[rt],x1,y1, ...
      dref[rt]*cos(angleref[rt]),dref[rt]*sin(angleref[rt]));
  robotsDg[rt]=dgcalcule(positionRobotsX[rt],positionRobotsY[rt],x1,y1, ...
      dref[rt]*cos(angleref[rt]),dref[rt]*sin(angleref[rt]));
 }
 for(int rob=0;rob<numberOfRobots;rob++){ //FOR todos los robots y obstáculos
  for(int h=0;h<numberOfObstacles+numberOfRobots+1;h++){
   if(distanceGlobalss[h][rob]>ws) continue;
   if((h-numberOfObstacles)==rob) continue;
   if(Method=="VFH+"){ //Método de evasión de obstáculos
    gammas[rob][h]=asin(marginObstacles/distanceGlobalss[h][rob]);
  }else if(Method=="VFH"){ //Método de evasión de obstáculos
  //Sector que ocupa el obstáculo
  gammas[rob][h]=asin(radioObstacle/distanceGlobalss[h][rob]);
  }
  restas[rob][h]=0; //Sectores inferiores
  sumas[rob][h]=0;  //Sectores superiores
  restas[rob][h]=angleGlobalss[h][rob]-gammas[rob][h];
  sumas[rob][h]=angleGlobalss[h][rob]+gammas[rob][h];
  if(sumas[rob][h]>2*PI) sumas[rob][h]=sumas[rob][h]-2*PI;
   if(restas[rob][h]<0) restas[rob][h]=restas[rob][h]+2*PI;
   if(restas[rob][h]<sumas[rob][h]){
```

```
        for(int j=(int)(restas[rob][h]/K);j≤(int)(sumas[rob][h]/K);j++){
        //Histograma Polar Primario
        Hpks[rob][j]=(a-(b*pow(distanceGlobalss[h][rob],2)));
       }
      }else{
      for(int j=0;j≤(int)(sumas[rob][h]/K);j++){
        //Histograma Polar Primario
        Hpks[rob][j]=(a-(b*pow(distanceGlobalss[h][rob],2)));
      }
      for(int j=(int)(restas[rob][h]/K);j<T;j++){
       //Histograma Polar Primario
       Hpks[rob][j]=(a-(b*pow(distanceGlobalss[h][rob],2)));
      }
      }
      for(int j=0;j<T;j++){
        if(Hpks[rob][j]≥TauH){
          Hbks[rob][j]=1.0;Hbkis[rob][j]=1;
        } //Histograma Polar Primario con histéresis
        else if(Hpks[rob][j]≤TauL){
          Hbks[rob][j]=0.0;Hbkis[rob][j]=0;
        } //Histograma Polar Binario con histéresis
        else{
          Hbks[rob][j]=Hbkprevs[rob][j]; Hbkis[rob][j]=Hbkprevis[rob][j];
        }
        //Preserva los histogramas actuales para la siguiente iteración
        Hbkprevs[rob][j]=Hbks[rob][j];
        Hbkprevis[rob][j]=Hbkis[rob][j];
        }
        for(int j=0;j<T;j++){
         if(Hbks[rob][j]==0){
          if(pertenecelibre(K*j, phiRightGlobalss[h][rob], ...
              orientationRobotsTH[rob], phiLeftGlobalss[h][rob])) Hms[rob][j]=0;
          else Hms[rob][j]=1;  //Histograma enmascarado con histéresis
         }else Hms[rob][j]=1;
        }
        for(int j=0;j<T;j++){
         Hmk[j]=(double)Hm[j];
         if(Hm[j]==1) sectorcolor[j]=red; //Sector ocupado
         else sectorcolor[j]=green;       //Sector libre
        }
      }
       //Calcular alpha nuevo que considera obstáculos y robots
       for(int ob=0;ob<numberOfObstacles+numberOfRobots+1;ob++){
        if(distanceGlobalss[ob][rob]<ws){
          if(robotsAlPHA[rob]<0)robotsAlPHA[rob]=robotsAlPHA[rob]+2*PI;
```

```
          //Busca el sector candidato si no está atrapado
          widseks[rob]=wide_valley(Hms[rob],(int)(robotsAlPHA[rob]/K));
          if(trapped){
            int randomIndex=generator.nextInt(20)-10;
            if((widseks[rob]+randomIndex<T)&&(widseks[rob]+randomIndex≥0)){}
          }
         }
        }
        robotsAlPHA[rob]=(widseks[rob]*K);
       }
      for(int jj=0;jj<numberOfRobots;jj++){
       if(consensus){
        robotsV[jj]=controlv1plus_consensus(robotsDg[jj],orientationRobotsTH[jj], ...
            robotsAlPHA[jj]);
        robotsW[jj]=control_w_consensus(robotsDg[jj],orientationRobotsTH[jj], ...
            robotsAlPHA[jj]);
       }else{
        robotsV[jj]=controlv1plus(robotsDg[jj],Wmax,v1min,Hpks[jj][(int) ...
            ((180*orientationRobotsTH[jj]/PI)/S)],robotsW[jj],v1max);
        robotsW[jj]=control_w(robotsDg[jj],orientationRobotsTH[jj], robotsAlPHA[jj]);
        if(abs(robotsV[jj]/robotsW[jj])<minTurningRatio) ...
            robotsW[jj]=robotsV[jj]/minTurningRatio;
       }
      }
}
```

Relaciones Fijas

```
if(consensus){
 //Radio de giro mínimo del maestro
 turningRatio=(abs(v1)/abs(w1))*signum(v1/w1);
}else{
 turningRatio=v1/w1;
 if(abs(turningRatio)<minTurningRatio) ...
     turningRatio=minTurningRatio*signum(turningRatio);
 if(abs(turningRatio)<minTurningRatio) ...
     System.out.println("turningRatio="+turningRatio);
}
T=360/S;
K=2*PI/T;
C=W*T/360;
//La arena es de 24x20 y se deja un margen de 0.5 por cada lado
if(xg>11.5)xg=11.5;   //Límite superior del X Goal
```

```
if(xg<-11.5)xg=-11.5; //Límite inferior del X Goal
if(yg>9.5)yg=9.5;     //Límite superior del Y Goal
if(yg<-9.5)yg=-9.5;   //Límite inferior del Y Goal
//La arena es de 24x20 y se deja un margen de 0.5 por cada lado
if(x1>11.5)x1=11.5;   //Límite superior del X maestro
if(x1<-11.5)x1=-11.5; //Límite inferior del X maestro
if(y1>9.5)y1=9.5;     //Límite superior del Y maestro
if(y1<-9.5)y1=-9.5;   //Límite inferior del X maestro
//Los robots no pueden salir de la arena
for(int es=0;es<numberOfRobots;es++){
  if(positionRobotsX[es]≥11.5) positionRobotsX[es]=11.5;   //Límite superior X
  if(positionRobotsX[es]≤-11.5) positionRobotsX[es]=-11.5; //Límite inferior X
  if(positionRobotsY[es]≤-9.5) positionRobotsY[es]=-9.5;   //Límite inferior Y
  if(positionRobotsY[es]≥9.5) positionRobotsY[es]=9.5;     //Límite superior Y
}
//Restringir ancho del robot W=10,30,50,70,90
if(W≥80) W=90;                //Valor máximo
else if(W<20) W=10;           //Valor mínimo
else if(W≥20 && W<40) W=30; //Impar 30
else if(W≥40 && W<60) W=50; //Impar 50
else W=70;
```

Funciones Personalizadas

```
//Obtener señal de Vídeo
public void getVideo(){
 if(((webcam.VideoReceiver)videocam).isConnected()){
   _view.Imagen.setImage(((webcam.VideoReceiver)videocam).getImage());
 }
}
//Conectar con la cámara
public void connectVideo(){
 if(!((webcam.VideoReceiver)videocam).isConnected()){
  ((webcam.VideoReceiver)videocam).connect();
 }
}
//Desconectar la cámara
public void disconnectVideo(){
 if(((webcam.VideoReceiver)videocam).isConnected()){
   ((webcam.VideoReceiver)videocam).disconnect();
 }
}
//TCP Socket por el puerto 1238
```

```
void cliente(){
 try{
   skt=new Socket("localhost",1238);
   ps=new PrintStream(skt.getOutputStream(),true,"UTF8");
 }
 catch(Exception e){ //Si ocurre un error
   System.out.println("Error:"+e);
 }
}
//Enviar cadena de caracteres
void enviar(String dato){
 try{
   ps.println(dato);
   System.out.println("Enviando...:"+dato);
   Thread.sleep(100);
 }
 catch(Exception e){ //Si ocurre un error
   System.out.println("Error:"+e);
 }
}
//Reiniciar posición de los robots si cambia el tipo de formación
public void ReInitRobots(int cantRobots){
  for(int r=0;r<numberOfRobots;r++){
    visiblesetmoway[r]=false;
  }
  //Variables relacionadas con los robots dependiendo de la nueva cantidad
  numberOfRobots=cantRobots;
  positionRobotsX=new double[numberOfRobots];
  positionRobotsY=new double[numberOfRobots];
  orientationRobotsTH=new double[numberOfRobots];
  robotsAlPHA=new double[numberOfRobots];
  robotsDg=new double[numberOfRobots];
  robotsW=new double[numberOfRobots];
  robotsV=new double[numberOfRobots];
  distanceRobots=new double[numberOfRobots];
  phiRightRobots=new double[numberOfRobots];
  phiLeftRobots=new double[numberOfRobots];
  angleRobots=new double[numberOfRobots];
  widseks=new double[numberOfRobots];
  angleGlobal=new double[numberOfRobots+numberOfObstacles+1];
  distanceGlobal=new double[numberOfRobots+numberOfObstacles+1];
  phiRightGlobal=new double[numberOfRobots+numberOfObstacles+1];
  phiRightGlobalss=new double[numberOfRobots+numberOfObstacles+1][numberOfRobots];
  phiLeftGlobalss=new double[numberOfRobots+numberOfObstacles+1][numberOfRobots];
  phiLeftGlobal=new double[numberOfRobots+numberOfObstacles+1];
```

```
  posGlobalX=new double[numberOfRobots+numberOfObstacles+1];
  posGlobalY=new double[numberOfRobots+numberOfObstacles+1];
  distanceObstacless=new double[numberOfObstacles][numberOfRobots];
  distanceRobotss=new double[numberOfRobots+1][numberOfRobots];
  angleObstacless=new double[numberOfObstacles][numberOfRobots];
  angleRobotss=new double[numberOfRobots+1][numberOfRobots];
  angleGlobals=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
  distanceGlobals=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
  distanceGlobalss=new double[numberOfObstacles+numberOfRobots+1][numberOfRobots];
  angleGlobalss=new double[numberOfObstacles+numberOfRobots+1][numberOfRobots];
  gammas=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
  sumas=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
  restas=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
  //Nuevas posiciones de los robots dependiendo de la formación y la cantidad
  for(int r=0;r<numberOfRobots;r++){
    visiblesetmoway[r]=true;
    if(circle==true){
      //X-A=R*cos(r*360/Nº Agentes)
      positionRobotsX[r]=R*cos(r*2*PI/numberOfRobots)+x1;
      //Y-B=R*sin(r*360/Nº Agentes)
      positionRobotsY[r]=R*sin(r*2*PI/numberOfRobots)+y1;
    }else if(row){  //Lína horizontal   (S4)---(S2)---(M)---(S1)---(S3)
      positionRobotsY[r]=y1;
      if(r%2==0){positionRobotsX[r]=x1+(r+2);}
      else{positionRobotsY[r]=x1-(r+2);}
    }else{//X-A=R*cos(r*360/Nº Agentes)
      positionRobotsX[r]=(R+r)*cos(r*2*PI/numberOfRobots)+x1;
      //Y-B=R*sin(r*360/Nº Agentes)
      positionRobotsY[r]=(R+r)*sin(r*2*PI/numberOfRobots)+y1;
    }
    dref[r]=dgcalcule(x1,y1,positionRobotsX[r],positionRobotsY[r],0,0);
    angleref[r]=alphacalcule(x1,y1,positionRobotsX[r],positionRobotsY[r],0,0);
  }
   _view.resetElements();
}
//Reinicia las propiedades de los obstáculos dependiendo de la nueva cantidad
public void ReInitObstacles(int cantObstacles){
   numberOfObstacles=cantObstacles;
   positionObstaclesX=new double[numberOfObstacles];
   positionObstaclesY=new double[numberOfObstacles];
   distanceLeftObstaclesX=new double[numberOfObstacles];
   distanceLeftObstaclesY=new double[numberOfObstacles];
   distanceRightObstaclesX=new double[numberOfObstacles];
   distanceRightObstaclesY=new double[numberOfObstacles];
   distanceObstacles=new double[numberOfObstacles];
```

```
    angleObstacles=new double[numberOfObstacles];
    velocityObstaclesX=new double[numberOfObstacles];
    velocityObstaclesY=new double[numberOfObstacles];
    angleGlobal=new double[numberOfRobots+numberOfObstacles+1];
    distanceGlobal=new double[numberOfRobots+numberOfObstacles+1];
    phiRightGlobal=new double[numberOfRobots+numberOfObstacles+1];
    phiRightGlobalss=new double[numberOfRobots+numberOfObstacles+1][numberOfRobots];
    phiLeftGlobalss=new double[numberOfRobots+numberOfObstacles+1][numberOfRobots];
    phiLeftGlobal=new double[numberOfRobots+numberOfObstacles+1];
    posGlobalX=new double[numberOfRobots+numberOfObstacles+1];
    posGlobalY=new double[numberOfRobots+numberOfObstacles+1];
    distanceObstacless=new double[numberOfObstacles][numberOfRobots];
    distanceRobotss=new double[numberOfRobots+1][numberOfRobots];
    angleObstacless=new double[numberOfObstacles][numberOfRobots];
    angleRobotss=new double[numberOfRobots+1][numberOfRobots];
    angleGlobals=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
    distanceGlobals=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
    distanceGlobalss=new double[numberOfObstacles+numberOfRobots+1][numberOfRobots];
    angleGlobalss=new double[numberOfObstacles+numberOfRobots+1][numberOfRobots];
    gammas=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
    sumas=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
    restas=new double[numberOfRobots][numberOfObstacles+numberOfRobots+1];
    for(int i=0;i<numberOfObstacles;i++){
      velocityObstaclesX[i]=(i+1)*0.1;
      velocityObstaclesY[i]=(i+1)*0.1;
     }
    setPosition("Obstacles"); //Establece las nuevas posiciones de los obstáculos
}
//Inicializa los sectores alrededor del maestro
public void InitSectors(){
 Px[0][0]=0;
 Px[0][1]=1;
 Px[0][2]=getDecimal(4,cos((S*PI/180)*(1)));
 Py[0][0]=0;
 Py[0][1]=0;
 Py[0][2]=getDecimal(4,sin((S*PI/180)*(1)));
 for(int q=1;q<T;q++){  //Calcula las coordenadas de cada triángulo
   Px[q][0]=0;
   Px[q][1]=Px[q-1][2];
   Px[q][2]=getDecimal(4,cos((S*PI/180)*(q+1)));
   Py[q][0]=0;
   Py[q][1]=Py[q-1][2];
   Py[q][2]=getDecimal(4,sin((S*PI/180)*(q+1)));
 }
 for(int g=0;g<T;g++){
```

```
    sectorx[g][0]=Px[g][0];
    sectorx[g][1]=Px[g][1];
    sectorx[g][2]=Px[g][2];
    sectory[g][0]=Py[g][0];
    sectory[g][1]=Py[g][1];
    sectory[g][2]=Py[g][2];
  }
for(int b=0;b<T;b++){ //Inicializa los sectores en verde
 Sk[b]=b*(360/T);
 sectorcolor[b]=green;
 Hbkprevi[b]=0;      //Histogramas en cero
 Hbkprev[b]=0.0;
}
}
//Establece las nuevas posiciones tanto para robots como para obstáculos
public void setPosition(String type){
 //Para los obstáculos busca celdas vacías
 if(type=="Obstacles"){
  if(numberOfObstacles>numberOfCells){
   numberOfObstacles=numberOfCells;
   for(int idx=0;idx<numberOfObstacles;idx++){
    arrayCells[idx]=1;
    positionObstaclesX[idx]=-12+2*marginObstacles*(idx%(round(0.5* ...
        width/marginObstacles)));//Columna
    positionObstaclesY[idx]=-10+2*marginObstacles*(idx/(round(0.5* ...
        width/marginObstacles)));//Fila
   }
  return;
  }
  int randomInt;
  for(int idx=0;idx<numberOfObstacles; idx++){
  do{
    randomInt=randomGenerator.nextInt(numberOfCells);
  }while(arrayCells[randomInt]!=0);
  arrayCells[randomInt]=1;
  //Coordenada X de la posición
  positionObstaclesX[idx]=-12+2*marginObstacles*(randomInt%(round(0.5* ...
      width/marginObstacles)));
  //Coordenada Y de la posición
  positionObstaclesY[idx]=-10+2*marginObstacles*(randomInt/(round(0.5* ...
      width/marginObstacles)));
  }
 }else{
  if(numberOfRobots>numberOfCells){
   numberOfRobots=numberOfCells;
```

```
    for(int idx=0;idx<numberOfRobots; idx++){
    arrayCells[idx]=1;
    //Coordenadas X de la posición
    positionRobotsX[idx]=-12+2*marginObstacles*(idx%(round(0.5* ...
        width/marginObstacles)));
    //Coordenadas Y de la posición
    positionRobotsY[idx]=-10+2*marginObstacles*(idx/(round(0.5* ...
        width/marginObstacles)));
   }
    return;
 }
  int randomInt;
  for(int idx=0;idx<numberOfRobots; idx++){
   do{
     randomInt=randomGenerator.nextInt(numberOfCells);
   }while(arrayCells[randomInt]!=0);
   arrayCells[randomInt]=1;
   positionRobotsX[idx]=-12+2*marginObstacles*(randomInt%(round(0.5* ...
        width/marginObstacles)));
   positionRobotsY[idx]=-10+2*marginObstacles*(randomInt/(round(0.5* ...
        width/marginObstacles)));
  }
 }
}
//Ley de control W para el consenso
public double control_w_consensus(double d, double th, double alpha){
 double w;
 w=0.05*(-((sin(th))*(d*cos(alpha)))/0.5+((cos(th))*(d*sin(alpha)))/0.5);
 if(w>Wmax)w=Wmax*signum(w);
 return w;
}
//Ley de control V para el consenso
public double controlv1plus_consensus(double d, double th, double alpha){
 double v=0.10*(cos(th)*(d*cos(alpha))+sin(th)*(d*sin(alpha)));
 if(v>v1max) v=v1max*signum(v);
 return v;
}
//Ley de control V
public double control_w(double d, double th, double alpha){
 double w;
 w=(v1max/minTurningRatio)*sin(alpha-th);
 if(d≥0.2){ return w;}
 else return 0;
}
//Ley de control W
```

```
public double controlv1plus(double d,double _wmax,double Vmin,double hc,double ...
    w,double vmax){
 double hcpp=min(hc,hm);
 double Vp=vmax*(1-(hcpp/hm));
 if(d<=0.2)return 0;
 else{
  if(abs(d*0.5*(Vp*(1-(w/_wmax))+Vmin))>vmax){
   return vmax*signum(d*0.5*(Vp*(1-(w/_wmax))+Vmin));
  }
  else{return d*0.5*(Vp*(1-(w/_wmax))+Vmin);}
 }
}
//Distancia al punto de destino
public double dgcalcule(double xorig,double yorig,double xdest,double ...
    ydest,double xref,double yref){
 double dgc;
 dgc=sqrt(pow(xdest-(xorig-xref),2)+pow(ydest-(yorig-yref),2));
 return dgc;
}
//Ángulo al punto de detino
public double alphacalcule(double xorig,double yorig,double xdest,double ...
    ydest,double xref,double yref){
 double alphac;
 alphac=atan2(ydest-(yorig-yref),xdest-(xorig-xref));
 if(alphac<0){alphac=alphac+2*PI;}    //Convierte de -pi;pi a 0-360
 return alphac;
}
//Ángulo límite por la derecha
public double phircalc(double thd,double aOd,double xOd,double yOd, double xr, ...
    double yr){
 double phid=thd+PI;
 if(thd<0){thd=thd+2*PI;phid=thd+PI;phid=(phid)%(2*PI);}
 if(aOd<0){aOd=aOd+2*PI;}
 double phir=phid;
 double dxr=xr+minTurningRatio*sin(thd);
 double dyr=yr-minTurningRatio*cos(thd);
 if(compAngles(aOd,thd)=='r' && compAngles(aOd,phir)=='l'){
  if(pow(dgcalcule(dxr,dyr,xOd,yOd,0,0),2)<minTurningRatio+ ...
      marginObstacles){phir=aOd;}
 }
 return phir;
}
//Ángulo límite por la izquierda
public double philcalc(double thi,double aOi,double xOi,double yOi,double xr, ...
    double yr){
```

```
 double phii=thi+PI;
 if(thi<0){thi=thi+2*PI; phii=thi+PI;  phii=(phii)%(2*PI);}
 if(aOi<0){aOi=aOi+2*PI;}
 double phil=phii;
 double dxl=xr-minTurningRatio*sin(thi);
 double dyl=yr+minTurningRatio*cos(thi);
 if(compAngles(aOi,thi)=='l' && compAngles(aOi,phil)=='r'){
  if(pow(dgcalcule(dxl,dyl,xOi,yOi,0,0),2)<minTurningRatio+ ...
      marginObstacles){phil=aOi;}
 }
 return phil;
}
//Comparación entre ángulos
public char compAngles(double a1, double a2){
 char direction;
 double a3=(a2+PI)%(2*PI);
 if(a1==a2) direction='r';
 if(a2≤PI){
  if(a1>a2 && a1<a3) direction='l'; //Izquierda
  else direction='r';               //Derecha
 }else{
  if(a1<a2 && a1>a3) direction='r';
  else direction='l';
 }
 return direction;
}
//Indica si el sector candidato está libre o no
public boolean pertenecelibre(double _ki, double _phir, double _th, double _phil){
 if(_phir==_phil) return true;
 if(_phir>_phil){
  if(0≤_ki&&_ki≤_phil) return true;
  else if(_phir≤_ki && _ki≤2*PI) return true;
  else return false;
 }else{
   if((_phir≤_ki&&_ki≤_th)||(_ki>_th&&_ki≤_phil)) return true;
   else return false;
 }
}
//Indica si el sector es suficientemente ancho para que el robot pase
public int wide_valley(int[] _Hp, int kT){
 int con=0, cont=0;
 fu1=false;
 fd1=false;
 int kup1,ku1=0,kd1=0,kn1=0;
 int ln=_Hp.length;
```

```
 int kdwn1=(kT+1)%ln;
 //Si al frente esta libre con suficiente espacio, seguir por ahí
 if(_Hp[kT]==0){
  if(_Hp[(kT+1)%ln]==0){if(_Hp[(kT-1+ln)%ln]==0){return kT;} }
 }
 kup1=(kT-1+ln)%ln;
 for(int v=0;v<T/2;v++){  //Cuantos sectores necesito para obtener 60º
  kup1=(kup1+1)%ln;
  if(_Hp[kup1]==0 && fu1==false){
   con++;
   if(con==C){fu1=true; ku1=(kup1-(int)(C/2)+ln)%ln; break;}}
   else con=0;
  }
  for(int ka=0;ka<T/2;ka++){ //Buscando el sector libre
   kdwn1=((kdwn1-1)+ln)%ln;
   if(_Hp[kdwn1]==0 && fd1==false){
   cont++;
   if(cont==C){
    fd1=true;
    kd1=(kdwn1+(int)(C/2))%ln;
    break;
   }
  }
  else cont=0;
}
  if(fu1 && fd1){
   if(((kT-kd1)+ln)%ln≥((ku1-kT)+ln)%ln){kn1=ku1;}
   else{kn1=kd1;}
  }
  else if(fu1){kn1=ku1;}
  else if(fd1){kn1=kd1;}
  else return -1;
  return kn1;
}
//Histéresis de los Histogramas
public void initHmTlTh(){
  b=1.0/(ws*ws-0.25*(ws-1)*(ws-1));
  a=b*(pow((ws-1)/2,2))+1;
  HmMax=1+b*0.25*(ws-1)*(ws-1)-b*(2*minTurningRatio*marginObstacles ...
      +marginObstacles*marginObstacles);
  TauHmax=HmMax;
  TauLmax=HmMax;
}
//Limitar cantidad de decimales en un double
public double getDecimal(int numeroDecimales,double decimal){
```

```
  decimal=decimal*(java.lang.pow(10, numeroDecimales));
  decimal=java.lang.round(decimal);
  decimal=decimal/java.lang.pow(10, numeroDecimales);
  return decimal;
}
public double nearest(double a1, double a2, double thh){
  double d1=abs(a1-thh);
  double d2=abs(a2-thh);
  if(d1>PI){d1=abs(d1-2*PI);}
  if(d2>PI){d2=abs(d2-2*PI);}
  if(d1≤d2) return a1;
  else return a2;
}
public  double[][] append(double[][] a, double[][] b){
  double[][] result=new double[a.length+b.length][];
  System.arraycopy(a, 0, result, 0, a.length);
  System.arraycopy(b, 0, result, a.length, b.length);
  return result;
 }
```

A.6 Código *C#* de la aplicación *Servidor MW*

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;
using lib_nrf24lu1_RF;
using lib_prog_mow2;
using System.Net.Sockets;
using System.Globalization;
using System.Net;
namespace InterruptTransferTest{
 public partial class Form1:Form{
  byte[] Data=new byte[10];
  Thread ejshilo;
  //16 robots, 12 datos de su posición(x,y,th)
```

```
short[,] RobotsStates={{180,180,0},{180,180,0},{180,180,0},{180,180,0}, ...
    {180,180,0},{180,180,0},{180,180,0},{180,180,0}, ...
    {180,180,0},{180,180,0},{180,180,0},{180,180,0}};
short[,] robotArray=new short[16,7];
double A,B,K,minTurningRatio=5.0,marginObstacles=10,TauH=0.8,TauL=0.5;
double HmMax,robotAlPHA=0,robotDg=0,hc,dref,angleref;
bool fu1=false,fd1=false;
short numberOfRobots=0,T,S=10,widesek,C,W=30,alpha=0,ws=40,x1=0,y1=0;
sbyte xOutsByte,yOutsByte;
System.Net.Sockets.TcpClient clientSocket;
Bzirf2Manager bzirf;
BziprogManager2 bziprog;
delegate void ParameterDelegate(int progress);
NetworkStream serverStream;
System.Windows.Forms.Timer myTimer;
byte CMD_SEND_RF=0xFF;  //Comando de envío (Moway)
Socket socketejs;       //Socket EJS (Servidor)
byte[] bejs=new byte[25];
byte xOut=0,yOut=0;
public Form1(){    //Ventana principal (GUI)
 T=(short)(360/S); //Cantidad de sectores (360/ancho de cada sector)
 K=RadianToDegree(2*PI/T);
 //Coeficientes para calcular el valor máximo del Histograma
 C=(short)(W*T/360);
 B=1.0/(ws*ws-0.25*(ws-1)*(ws-1));
 A=B*(Pow((ws-1)/2,2))+1;
 //Valor máximo del histograma
 HmMax=1+B*0.25*(ws-1)*(ws-1)-B*(2*minTurningRatio*marginObstacles ...
     +marginObstacles*marginObstacles);
 InitializeComponent();
 bzirf=new Bzirf2Manager();
 this.Controls.Add(bzirf);
 //Para tratar el ACK de los robots
 bzirf.Read+=new NewBziRF2DataEventHandler(bziRFReadEvent);
 bziprog=new BziprogManager2();
 if(bzirf.USBdongleconected && ...
     !bzirf.RFIsRunning()&&!bziprog.ProgIsRunning()&&textCanal.Text.Length>0 && ...
     textDirPropia.Text.Length>0){
  byte canal=byte.Parse(textCanal.Text, ...
      System.Globalization.NumberStyles.AllowHexSpecifier);
  byte dir=byte.Parse(textDirPropia.Text, ...
      System.Globalization.NumberStyles.AllowHexSpecifier);
  if(bzirf.Start(canal, dir)) label2.Text="RF Connected";
  else label2.Text="Imposible conectar RF";
 }
```

```
  clientSocket=new System.Net.Sockets.TcpClient();  //Inicializa socket TCP
  clientSocket.Connect("127.0.0.1",3000);
  if(clientSocket.Connected){label8.Text="TCP Connected";}
  else{label8.Text="TCP DisConnected";}
  IPAddress ipAddress=IPAddress.Any;  //Servidor EJS
  TcpListener listener=new TcpListener(ipAddress, 1238);
  listener.Start();   //"Escucha" iniciada
  socketejs=listener.AcceptSocket();
  myTimer=new System.Windows.Forms.Timer();
  myTimer.Tick+=new EventHandler(TimerEventProcessor);
  //La lectura de los datos del Socket EJS se hace en un Thread(leeEJS)
  ejshilo=new Thread(new ThreadStart(leeEJS));
  ejshilo.Start();
  myTimer.Interval=300; //500 ok para 2 robots(5 y 3)
  myTimer.Start();
 }
 public void leeEJS(){ //Función que lee paquete de EJS
  sbyte rob_code;       //Id de SwisTrack
  short R=30;           //Radio del círculo
  byte TargOrConf,formation;
  string cadena;
  ASCIIEncoding ascii=new ASCIIEncoding();
  while(true){          //Leer desde EJS
   int k=socketejs.Receive(bejs);
   cadena=ascii.GetString(bejs);
   string[] datos=cadena.Split(';');
   if(k>6){//Si el tamaño del paquete es > 6
    xOutsByte=Convert.ToSByte(datos[0]);
    yOutsByte=Convert.ToSByte(datos[1]);
    rob_code=Convert.ToSByte(datos[2]);
    //1-conf formación, 2-conf referencias, 3-ctrl
    TargOrConf=Convert.ToByte(datos[3]);
    //Formación: 1-línea, 2-círculo, 3-libre
    formation=Convert.ToByte(datos[4]);
    xOut=(byte)(xOutsByte+84);
    yOut=(byte)(yOutsByte+64);
    //Construyendo paquete de configuración
    Data[0]=CMD_SEND_RF;
    Data[1]              //Robot destino
    Data[2]=xOut;        //Xp
    Data[3]=yOut;        //Yp
    Data[4]=0;           //Xref
    Data[5]=0;           //Yref
    Data[6]=TargOrConf; //1-conf formación, 2-conf referencias, 3-cotrl
    Data[7]=0;
```

```
    Data[8]=0;          //Id paquete
    Data[9]=255;
   }
  }
}
//Procesa paquete de SwisTrack y envía a robots y a EJS
private void TimerEventProcessor(Object myObject, EventArgs myEventArgs){
 short x_sh,y_sh,th_sh;
 string[] lines;
 int beginFrame=-1;
 TimeSpan stop;
 TimeSpan start=new TimeSpan(DateTime.Now.Ticks);
 serverStream=clientSocket.GetStream();
 int z=(int)clientSocket.ReceiveBufferSize;
 byte[] inStream=new byte[z];
 string returndata=null;
 do{
   serverStream.Read(inStream, 0,(int)clientSocket.ReceiveBufferSize);
   returndata=returndata+System.Text.Encoding.ASCII.GetString(inStream);
 }while(clientSocket.Available!=0);
   lines=returndata.Split('\n');
   int size=lines.Length;
   for(int b=size-1; b>=0; b--){
    if(lines[b].StartsWith("FRAMENUMBER,")){
     beginFrame=b;
     break;
    }
   }
   if(beginFrame<0) return;
   numberOfRobots=0;
   for(int i=beginFrame+1; i<size-1; i++){ //Para cada paquete
    if(lines[i].StartsWith("PARTICLE,")){  //Si la línea empieza con PARTICLE
     //Divide la línea por comas y guarda los String en data
     string[] datos=lines[i].Split(',');
     byte robot=Convert.ToByte(datos[1]);
     //Obtiene X y lo convierte a Short
     if(datos[5].IndexOf('.')>0) ...
         x_sh=Convert.ToInt16(datos[5].Remove(datos[5].IndexOf('.')));
     else x_sh=Convert.ToInt16(datos[5]);
     //Obtiene Y y lo convierte a Short
     if(datos[6].IndexOf('.')>0) ...
         y_sh=Convert.ToInt16(datos[6].Remove(datos[6].IndexOf('.')));
     else y_sh=Convert.ToInt16(datos[6].Remove(datos[6].IndexOf('*')));
     //Obtiene Th y lo convierte a Double
     double th_dou=Convert.ToDouble(datos[4].Replace(".", ","));
```

```
    th_dou=RadianToDegree(th_dou);          //Th a grados
    th_sh=Convert.ToInt16(th_dou);          //Th en grados a Short
    robotArray[numberOfRobots,0]=robot;     //Robot destino
    robotArray[numberOfRobots,1]=x_sh;      //Actualiza la coordenada X
    robotArray[numberOfRobots,2]=y_sh;      //Actualiza la coordenada Y
    robotArray[numberOfRobots,3]=th_sh;     //Actualiza la coordenada Th
    robotArray[numberOfRobots,6]=numberOfRobots;  //Índice del Robot
    if(robotArray[numberOfRobots,3]≥360){robotArray[numberOfRobots,3]= ...
        (short)(robotArray[numberOfRobots,3]-(short)360); }
    numberOfRobots++;
   }
  }
  for(int b=0;b<numberOfRobots;b++){
   if(robotArray[b,0]==5){
    x1=robotArray[b,1];
    y1=robotArray[b,2];
   }
  }
  for(short b=0; b<numberOfRobots; b++){//Analiza Array de posiciones de Robots.
   if(robotArray[b,0]==5){
    short xO=(short)(xOutsByte); //Referencia recibida del EJS en byte con signo
    short yO=(short)(yOutsByte);
    robotAlPHA=alphacalcule(robotArray[b,1],robotArray[b,2],xO,yO,0,0);
    robotDg=dgcalcule(robotArray[b,1],robotArray[b,2],xO,yO,0,0);
   }
   else if(robotArray[b,0]==1){
    //Ángulos y distancias de cada robot al punto de destino
    robotAlPHA=alphacalcule(robotArray[b,1],robotArray[b,2],x1,y1,-30,0);
    robotDg=dgcalcule(robotArray[b,1], robotArray[b,2],x1,y1,-30,0);
   }
   short alpha=calcularAlphaNew(b, robotAlPHA, robotDg);
   //Enviar alpha y posiciones a robots
   //Obtiene los bytes correspondientes a X
   byte[] x_bytes=BitConverter.GetBytes(robotArray[b,1]);
   //Obtiene los bytes correspondientes a Y
   byte[] y_bytes=BitConverter.GetBytes(robotArray[b,2]);
   //Obtiene los bytes correspondientes a TH
   byte[] th_bytes=BitConverter.GetBytes(robotArray[b,3]);
   byte[] alpha_bytes=BitConverter.GetBytes(alpha);
   //Obtiene los bytes correspondientes a Alpha
   if(bzirf.RFIsRunning()){
    Data[0]=CMD_SEND_RF;                    //RFUSB Comando "send"
    Data[1]=(byte)robotArray[b, 0];         //Robot destino
    Data[2]=(byte)(robotArray[b, 1]+84); //X+84 para enviarla en 1 byte
    Data[3]=(byte)(robotArray[b, 2]+64); //Y+64 para enviarla en 1 byte
```

```
        Data[4]=th_bytes[0];                    //th-byte 0
        Data[5]=th_bytes[1];                    //th-byte 1
        Data[6]=alpha_bytes[0];                 //alpha-byte 0
        Data[7]=alpha_bytes[1];                 //alpha-byte 1
        Data[8]=(byte)hc;                       //Valor del histograma
        Data[9]=(byte)20;                       //Id del paquete
        bzirf.Send(Data);
      }
    }
  }
//Calcula Alpha
private short calcularAlphaNew(short b, short robotALPHA1, double robotDg){
  short[] distanceRobotss=new short[numberOfRobots];
  short[] angleRobotss=new short[numberOfRobots];
  short[] phiRightGlobalss=new short[numberOfRobots];
  short[] phiLeftGlobalss=new short[numberOfRobots];
  double[] Hpks=new double[T];        //Histograma polar
  double[] Hbks=new double[T];        //Histograma binario
  double[] Hbkprevs=new double[T];    //Histograma binario previo
  short[] Hms=new short[T];           //Histograma enmascarado
  short[] gammas=new short[numberOfRobots];
  short[] sumas=new short[numberOfRobots];
  short[] restas=new short[numberOfRobots];
  alpha=robotALPHA1;
  for(int u=0;u<T;u++){ //Inicializa los sectores, su color en verde
   Hpks[u]=0;              //Histogramas en 0
   Hbks[u]=0;
   Hms[u]=0;
  }
  //Distancia de cada robots a cada robot
  for(short rott=0;rott<numberOfRobots;rott++){
   if(b==rott) continue;
   else{
    //Distancia y ángulo al punto de destino para todos los robots esclavos
    distanceRobotss[rott]=dgcalcule(robotArray[b,1],robotArray[b,2], ...
        robotArray[rott,1],robotArray[rott,2],0,0);
    angleRobotss[rott]=alphacalcule(robotArray[b,1],robotArray[b,2], ...
        robotArray[rott,1],robotArray[rott,2],0,0);
   }
  }
  for(short h=0;h<numberOfRobots;h++){
   if(b==h) continue;
    //Límites derecho e izquierdo para todos los robots esclavos
    phiRightGlobalss[h]=phircalc(robotArray[b,3],angleRobotss[h],robotArray[h,1], ...
        robotArray[h,2],robotArray[b,1],robotArray[b,2]);
```

```
   phiLeftGlobalss[h]=philcalc(robotArray[b,3],angleRobotss[h],robotArray[h,1], ...
       robotArray[h,2],robotArray[b,1],robotArray[b,2]);
  }
 for(short g=0;g<numberOfRobots;g++){
  if(g==b) continue;
  if(distanceRobotss[g]>ws) continue;
   gammas[g]=(short)(RadianToDegree(Asin(marginObstacles/distanceRobotss[g])));
   restas[g]=0; //Reinicia sectores inferiores
   sumas[g]=0;  //Reinicia sectores superiores
   restas[g]=(short)(angleRobotss[g]-gammas[g]); //Sectores inferiores
   sumas[g]=(short)(angleRobotss[g]+gammas[g]);  //Sectores superiores
   if(sumas[g]≥360) sumas[g]=(short)(sumas[g]-360);
   if(restas[g]<0) restas[g]=(short)(restas[g]+360);
   if(restas[g]<sumas[g]){
   for(short j=(short)(restas[g]/K);j≤(short)(sumas[g]/K);j++){
     if(A-(B*Pow(distanceRobotss[g], 2))>Hpks[j]){
      Hpks[j]=(A-(B*Pow(distanceRobotss[g], 2))); //Histograma Polar Primario
       if(Hpks[j]<0) Hpks[j]=0;
   }
  }
  }else{
    for(short j=0;j≤(short)(sumas[g]/K);j++){
     if(A-(B*Pow(distanceRobotss[g], 2))>Hpks[j]){
      Hpks[j]=(A-(B*Pow(distanceRobotss[g], 2))); //Histograma Polar Primario
       if(Hpks[j]<0) Hpks[j]=0;
     }
    }
    for(short j=(short)(restas[g]/K);j<T;j++){
     if(A-(B*Pow(distanceRobotss[g], 2))>Hpks[j]){
      Hpks[j]=(A-(B*Pow(distanceRobotss[g], 2))); //Hp Histograma Polar Primario
       if(Hpks[j]<0) Hpks[j]=0;
     }
    }
  }  //Si el ángulo se pasa de 360
   if(robotArray[g,3]≥360){robotArray[g,3]=(short)(robotArray[g,3]-(short)360);}
   //Se multiplica por 10 para el envío, luego se divide por 10 en el robot
   hc=(short)((10*Hpks[(short)(robotArray[b,3]/S)])*10);
   for(short j=0;j<T;j++){
    if(Hpks[j]≥TauH) Hbks[j]=1.0;  //Hpks Histograma Polar Primario con histéresis
    else if(Hpks[j]≤TauL) Hbks[j]=0.0; //Hbks Histograma Polar Binario con histéresis
    else Hbks[j]=Hbkprevs[j];
    Hbkprevs[j]=Hbks[j];
   }
   for(short j=0;j<T;j++){
    if(Hbks[j]==0){
```

```
      if(pertenecelibre((short)(K*j),phiRightGlobalss[g],robotArray[b,3], ...
          phiLeftGlobalss[g]))Hms[j]=0;
      else Hms[j]=1; //Hms Histograma enmascarado con histéresis
     }else Hms[j]=1;
   }
    if(robotArray[b, 0]==5){
     for(short j=0; j<T; j++){
      if(Hpks[j]!=0){ }
     }
    }
    for(short ob=0;ob<numberOfRobots;ob++){
     if(ob==b) continue;
     if(distanceRobotss[ob]<ws){ //Sector candidato
      if(robotALPHA1<0){robotALPHA1=(short)(robotALPHA1+360);}
       widesek=wide_valley(Hms,((short)(robotALPHA1/K)));
      }
     }
     alpha=(short)(widesek*K);
    }
     return(short)alpha;
}
//Función que calcula la distancia a un punto
private short dgcalcule(short xorig,short yorig,short xdest,short ydest,short ...
    xref,short yref){
 double dgc=0;
 dgc=Sqrt(Pow(xdest-(xorig-xref),2)+Pow(ydest-(yorig-yref),2));
 return(short)dgc;
}
//Función que calcula el ángulo a un punto
private short alphacalcule(short xorig,short yorig,short xdest,short ydest,short ...
    xref,short yref){
 double alphac=0;
 alphac=Atan2(ydest-(yorig-yref),xdest-(xorig-xref));
 if(alphac<0){alphac=alphac+2*PI;}        //Convierte de -pi;pi a 0-360
 return(short)(RadianToDegree(alphac));  //Valor convertido a grados
}
//Función que calcula el ángulo límite derecho
private short phircalc(short thd,short aOd,short xOd,short yOd,short xr,short yr){
 short phid=(short)(thd+180.0);
 if(thd<0){thd=(short)(thd+360.0); phid=(short)(thd+360); phid=(short)((phid)%(360));}
 if(aOd<0){aOd=(short)(aOd+360.0);}
 short phir=phid;
 short dxr=(short)(xr+minTurningRatio*Sin(thd));
 short dyr=(short)(yr-minTurningRatio*Cos(thd));
 if(compAngles(aOd,thd)=='r'&&compAngles(aOd,phir)=='l'){
```

```
  if(Pow(dgcalcule(dxr,dyr,xOd,yOd,0,0),2)<minTurningRatio+marginObstacles){phir=aOd;}
 }
 return phir;
}
private void botonDesconectarRF_Click(object sender, EventArgs e){
 myTimer.Stop();           //Detiene el controlador
 if(bzirf.RFIsRunning()){ //Desconecta el módulo RF
  if(bzirf.Stop()) label2.Text="RF Disconnected";
  else label2.Text="Impossible to disconect RF";
 }
 if(clientSocket.Connected){
  clientSocket.Close(); //Cierra el Socket
  if(!clientSocket.Connected){
   label8.Text="TCP Disconnected";
  }
 }
}
```

A.7 Código en lenguaje C para los robots *Moway*.

Maestro Moway

```
 #include "p18f86j50.h"      //Microcontrolador de Moway
 #include "lib_rf2gh4.h"     //Librería de RF
 #include "lib_sen_moway.h" //Librería de sensores
 #include "lib_mot_moway.h" //Librería de motores
 #include "h"
 #include "delays.h"
 void main(){
  float x,y,th,alpha,d,alfa,omega,Vcm,Vi,Vd,Vicm,Vdcm,xp,yp,hcpp,hc,Vp,hm=11.8,error;
  unsigned char data_in[8],data_out[8],xbyte ybyte,thbytes[2];
  unsigned char alphabytes[2],xrefbyte,yrefbyte,contOrConf,flagPos=0;
  char data_in_dir, ret, var;
  unsigned char fwdbacki=0,fwdbackd=0; //fwd=1 back=0
  float omegamax=Vmax/6.5, Vmax=12.0;
  if(SEN_BATTERY()<50) SEN_SPEAKER(140,20,SPEAKER_TIME);
  RF_CONFIG_SPI();
  RF_CONFIG(0x00,0x05); //Maestro dir 0x05, canal 0
  RF_ON();
  while(1){
   ret=RF_RECEIVE(&data_in_dir,&data_in[0]) ;
   if(ret!=2 && data_in_dir==8){  //Recibe de la cámara 8
```

```
 if(data_in[7]==0){
  LED_TOP_RED_ON_OFF();
  LED_BRAKE_ON_OFF();
  x=((float)(data_in[0]))-84.0; //x+84 x en un solo byte desde la cámara
  y=((float)(data_in[1]))-64.0; //y+64 y en un solo byte desde la cámara
  thbytes[0]=data_in[2];        //2 bytes correspondientes al theta del robot
  thbytes[1]=data_in[3];
  alphabytes[0]=data_in[4];     //2 bytes del ángulo al target
  alphabytes[1]=data_in[5];
  hc=(float)(data_in[6]/10.0); //Valor histograma en el sector del ángulo theta
  data_out[0]=data_in[0];       //x+84 del robot maestro
  data_out[1]=data_in[1];       //y+64 del robot maestro
  data_out[7]=123;
  var=RF_SEND(1,data_out);  //Envío al robot 1
  var=RF_SEND(4,data_out);  //Envío al robot 4
  var=RF_SEND(3,data_out);  //Envío al robot 3
  //Convierte th de 2 bytes a entero
  if(thbytes[1]≥128){th=(float)(-(¬thbytes[1]+1)*256+thbytes[0]); }
  else{th=(float)(thbytes[1]*256+thbytes[0]);}
  //Convierte alpha de 2 bytes a entero
  if(alphabytes[1]≥128){alpha=(float)(-(¬alphabytes[1]+1)*256+alphabytes[0]);}
  else{alpha=(float)(alphabytes[1]*256+alphabytes[0]);}
  flagPos=1; //Marca la bandera para indicar que se recibió la posición del robot
 }else{
  LED_TOP_GREEN_ON_OFF();
  xp=((float)(data_in[0]))-84.0;
  yp=((float)(data_in[1]))-64.0;
  xrefbyte=data_in[2];
  yrefbyte=data_in[3];
  contOrConf=data_in[4]; //Control(3) o Configuración (1,2)
  flagPos=1;
 }
}else{
 LED_TOP_RED_OFF();
 LED_BRAKE_OFF();
}
if(flagPos==1){                          //Si recibe de la cámara
d=sqrt(pow(yp-(y),2)+pow(xp-(x),2)) ;  //Calcula la distancia al punto
if(d>0){
 th=th*3.14/180.0;
 alpha=alpha*(3.14/180.0);
 error=alpha-th;
 if(error<-3.14){error=6.28+error;}
 else if(error>3.14){error=-6.28+error;}
 omega=1.0*(error);
```

```
 if(hc>hm) hcpp=hm;  //Mínimo entre hc y hm
 if(hm>hc) hcpp=hc;
 Vp=Vmax*(1-(hcpp/hm));
 if(Vp*(1-(omega/omega_max))+9.2 ≥Vmax){Vcm=Vmax;}
 else{Vcm=Vp*(1-(omega/omega_max))+9.2;}
 Vicm=(2.0*Vcm+omega*6.5)/2.0;
 Vdcm=(2.0*Vcm-omega*6.5)/2.0;
 if(Vicm>0){ //Módulo de la Velocidad
 Vicm=Vicm;
 fwdbacki=1; //Velocidad positiva FWD=1
 }else{
 Vicm=-Vicm; //Velocidad negativa BACK=0
 fwdbacki=0;
 }
 if(Vdcm>0){ //Módulo de la Velocidad
 Vdcm=Vdcm;
 fwdbackd=1; //Velocidad positiva FWD=1
 }else{
 Vdcm=-Vdcm; //Velocidad negativa BACK=0
 fwdbackd=0;
 }
 if(Vicm≥9.2)Vi=(Vicm-9.2)/0.0788;
 else if(Vicm≥4.5)Vi=1;  //Límite zona muerta motor
 else Vi=0;
 if(Vdcm≥9.2)Vd=(Vdcm-9.2)/0.0788;
 else if(Vdcm≥4.5)Vd=1;  //Límite zona muerta motor
 else Vd=0;
 if(Vi>100)Vi=100;       //Saturación de velocidades
 if(Vi<0)Vi=0;
 if(Vd<0)Vd=0;
 if(Vd>100)Vd=100;
 if(d>5){
  MOT_CHA_VEL((unsigned char)Vi,fwdbacki,LEFT,TIME,0) ;
  MOT_CHA_VEL((unsigned char)Vd,fwdbackd,RIGHT,TIME,0) ;
 }else{
  MOT_STOP();
  LED_BRAKE_ON();
 }
}else{ //Calcula alfa y lo convierte a grados
alfa=(atan2(yp-(y-ymref),xp-(x-xmref)))*(180/3.14);
if(th>alfa+180.0){th=-(360.0-th);} //Th de (0,360] a (-180,180]
error=alfa-th;
if(error<0){merror=-error;}
else{merror=error;}
if(error<0){SENT=0;}                //Gira por la izquierda
```

```
      else{SENT=1;}                          //Gira por la derecha
      angulo=(unsigned char)((merror)*3.33/12.0); //Valor a girar (angulo°x3.33)/12°
      if(merror>5&&d>10){
       MOT_ROT(1,1,1,SENT,0,angulo);
        while(!MOT_END){}
      }else{MOT_STOP();}
      if(d>10){MOT_STR(5,FWD,DISTANCE,0);}
      else{MOT_STOP();}
      }
      flagPos=0;
    }
  }
}
```

Esclavo Moway

```
#include "p18f86j50.h"          //Microcontrolador de Moway
#include "lib_rf2gh4.h"         //Librería de RF
#include "lib_sen_moway.h"      //Librería de sensores
#include "lib_mot_moway.h"      //Librería de motores
#include "h"
#include "delays.h"
void main(){
 unsigned char flagRef=0,flagPos=0;
 float x,y,th,alpha,d,xmref,ymref,omega,Vcm,Vi,Vd,Vicm,Vdcm,xp,yp;
 unsigned char data_in[8],xbyte,ybyte,data_out[8],thbytes[2],alphabytes[2];
 char data_in_dir, ret;
 unsigned char xrefbyte,yrefbyte,fwdbacki=0,fwdbackd=0; //fwd=1 back=0
 float Vmax=10.0,omega_max=17.0/6.5,hcpp,hc,Vp,hm=11.8,error;
 if(SEN_BATTERY()<50) SEN_SPEAKER(140,20,SPEAKER_TIME);
  RF_CONFIG_SPI();
  RF_CONFIG(0x00,0x01); //Esclavo dir 0x01, Canal 0
  RF_ON();
  while(1){
   ret=RF_RECEIVE(&data_in_dir,&data_in[0]) ;
   if(ret!=2){  //Recibe de la cámara (dir 8)
    if(data_in_dir==8){
    LED_TOP_RED_ON_OFF();
    LED_BRAKE_ON_OFF();
    x=(float)(data_in[0])-84.0;  //x+84 x en un solo byte desde la cámara
    y=(float)(data_in[1])-64.0;  //y+64 x en un solo byte desde la cámara
    thbytes[0]=data_in[2];       //2 bytes correspondientes al theta del robot
    thbytes[1]=data_in[3];
```

```
 alphabytes[0]=data_in[4];     //2 bytes del ángulo al target
 alphabytes[1]=data_in[5];
 hc=(float)(data_in[6]/10.0); //Valor histograma en el sector del ángulo theta
 if(thbytes[1]≥128){th=(float)(-(¬thbytes[1]+1)*256+thbytes[0]);}
 else{th=(float)(thbytes[1]*256+thbytes[0]);} //Th a entero
 if(alphabytes[1]≥128){ alpha=(float)(-(¬alphabytes[1]+1)*256+alphabytes[0]);}
 else{alpha=(float)(alphabytes[1]*256+alphabytes[0]);} //Alpha a entero
 flagPos=1; //Marca la bandera para indicar que se recibió la posición del robot
}else{ //Recibe la referencia del maestro si no está en config
 LED_TOP_GREEN_ON_OFF();
 xp=(float)(data_in[0])-84.0;
 yp=(float)(data_in[1])-64.0;
 flagPos=1;
}
}else{LED_FRONT_ON_OFF();}
if(flagPos==1){    //Para controlar debe recibir de la cámara y del maestro
 xmref=-30.0;
 ymref=0.0;
 d=sqrt(pow(yp-(y-ymref),2)+pow(xp-(x-xmref),2)); //Distancia al punto
 if(d>0){
  th=th*3.1415/180.0;          //Th a radianes
  alpha=alpha*(3.1415/180.0); //Alpha a radianes
  error=alpha-th;
  if(error<-3.14){error=-(6.28+error);}
  else if(error>3.14){error=6.28-error;}
  omega=1.0*error;       //Ka*e+Kb*B Kb<0, Kp>0, Ka-Kp>0
  if(hc≥hm) hcpp=hm;    //Mínimo entre hc y hm
  else hcpp=hc;
  Vp=Vmax*(1-(hcpp/hm));
  if(((Vp*(1-(omega/omega_max))+0))>Vmax){Vcm=Vmax;}
  else{Vcm=(Vp*(1-(omega/omega_max))+0);}
  Vicm=(2.0*Vcm+omega*6.5)/2.0;
  Vdcm=(2.0*Vcm-omega*6.5)/2.0;
  if(Vicm>0){      //Módulo de la Velocidad
   Vicm=Vicm;
   fwdbacki=1;     //Velocidad positiva FWD=1
  }else{
    Vicm=-Vicm;    //Velocidad negativa BACK=0
    fwdbacki=0;
   }
   if(Vdcm>0){     //Módulo de la Velocidad
    Vdcm=Vdcm;
    fwdbackd=1;    //Velocidad positiva FWD=1
   }else{
    Vdcm=-Vdcm;    //Velocidad negativa BACK=0
```

```
        fwdbackd=0;
       }
       if(Vicm≥9.2) Vi=(Vicm-9.2)/0.0788; //Vi de Cm a Volt
       else if(Vicm≥4.5)Vi=1;              //Límite zona muerta motor
       else Vi=0;
       if(Vdcm≥9.2) Vd=(Vdcm-9.2)/0.0788; //Vd de Cm a Volt
       else if(Vdcm≥4.5)Vd=1;               //Límite zona muerta motor
       else Vd=0;
       if(Vi>100)Vi=100; //Saturación de velocidades de los motores
       if(Vi<0)Vi=0;
       if(Vd<0)Vd=0;
       if(Vd>100)Vd=100;
       if(d>10){ //Si la distancia al punto final es > 10 cm
        MOT_CHA_VEL((unsigned char)Vi,fwdbacki,LEFT,TIME,0) ;
        MOT_CHA_VEL((unsigned char)Vd,fwdbackd,RIGHT,TIME, 0) ;
       }
     }
     flagPos=0; //Reinicia bandera de recepción de posición
     flagRef=0; //Reinicia bandera de recepción de referencia
     }
   }
}
```