UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado

DESARROLLO DE UN CONTROLADOR EN PLAYER/STAGE PARA EL SEGUIMIENTO DE TRAYECTORIAS DE ROBOTS MÓVILES

Juan Alonso Álvarez

Enero 2014

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado

DESARROLLO DE UN CONTROLADOR EN PLAYER/STAGE PARA EL SEGUIMIENTO DE TRAYECTORIAS DE ROBOTS MÓVILES

Autor: Juan Alonso Álvarez
Directora: Elena López Guillén

TRIBUNAL:

Presidente: Manuel Ocaña Miguel

Vocal 1°: Julio Pastor Mendoza

Vocal 2°: Elena López Guillén

CALIFICACIÓN: ……..

FECHA: ……

AGRADECIMIENTOS

En primer lugar me gustaría expresar mi agradecimiento a Elena López Guillén, quien ha hecho posible que pudiera realizar este trabajo tan interesante y en el que tanto he aprendido. Su motivación por la enseñanza hace que el trabajo salga adelante gracias a la dedicación por su parte, tiempo y esfuerzo que han hecho posible la finalización de este proyecto. Ha sido muy instructivo y gratificante haber trabajado a su lado.

Por supuesto mis padres han sido un apoyo muy importante para mí a lo largo de mi vida universitaria, siempre me han ayudado y aconsejado de la mejor forma posible en todo momento.

Tampoco puedo olvidarme de mi hermano que siempre ha sido fuente de motivación, proporcionándome esa iniciativa que necesitaba para la realización y superación de nuevos retos.

Por último recuerdo a toda mi familia y amigos, a todos ellos agradezco su apoyo y el haberme aguantado todo este tiempo.

Índice

Lista de Figuras

Lista de Tablas

1. Resumen breve

Este proyecto presenta como objetivo dotar a un robot móvil de las herramientas software adecuadas para que sea capaz de realizar un recorrido atravesando puntos de paso introducidos por el usuario, y le permita navegar hasta el destino evitando colisiones con los obstáculos presentes en el entorno, ya sean estáticos o dinámicos.

Con esta finalidad se diseña y programa un controlador para un sistema multi-robot, que ajusta el movimiento mediante técnicas de control óptimo y borroso dando lugar a un comportamiento inteligente del robot en cada situación. Para realizar las tareas de desarrollo y simulación del sistema se emplea el popular entorno de programación de aplicaciones robóticas Player/Stage.

Palabras clave: Player-Stage, generación de trayectorias, evitación de obstáculos, control borroso, control óptimo.

2. Abstract

The main aim of this work is to provide a mobile robot with the appropriate software tools in order to travel through the trajectory marked by different points introduced by the user, allowing it to reach the target while avoiding collisions with any other robots or obstacles in the surroundings, both static and dynamic.

To achieve this goal, a controller for a multi-robot system has been designed and developed, which adjusts the motion by techniques such as optimal control and fuzzy control providing the robot with a smart behaviour for each situation. Developing and simulating tasks have been carried out within the popular robotic application programming environment Player/Stage.

Key words: Player-Stage, path generation, obstacles avoidance, fuzzy control, optimal control.

3. Resumen extendido

Este proyecto se ha desarrollado con el objetivo de diseñar, programar y simular un sistema que regula el desplazamiento de un robot móvil introduciendo diferentes técnicas de control para conseguir un comportamiento inteligente a lo largo de su recorrido.

Siempre se presenta una problemática a estudio en torno a la navegación del robot, convirtiéndose en una de las principales competencias requeridas en un robot móvil, la cual incluye la resolución de una serie de problemas, como son: "¿dónde estoy?", "¿hacia dónde me dirijo?", "¿cómo lo consigo?"…

Estos y otros problemas se enmarcan dentro de una de las líneas de investigación del Departamento de Electrónica de la UAH, y más concretamente este trabajo se centra en implementar un controlador para un sistema multi-robot, que abarca todas las etapas involucradas en la navegación de robots móviles; desde la interacción con el usuario para recopilar datos y mostrar información, la posterior generación de trayectorias a partir de puntos de paso deseados y el propio seguimiento del robot a lo largo del recorrido hasta que alcanza la posición final de destino.

En primer lugar es necesario preguntar al usuario dónde quiere ir y calcular una trayectoria acorde a sus necesidades. Este recorrido se genera mediante splines cúbicas a partir de puntos de paso intermedios. Una vez se ha generado la trayectoria, el robot comenzará a recorrer la ruta planificada hasta llegar a su destino adoptando un comportamiento inteligente que le hará reaccionar ante cualquier imprevisto, llegando incluso a detenerse si es necesario.

Para lograr el seguimiento de la trayectoria se han implementado distintos tipos de controles. Mientras se desplaza, el controlador actúa continuamente sobre la velocidad angular a través del algoritmo de control óptimo implementado. De este modo se consigue minimizar el error entre la posición real del robot y la teórica que se había planificado inicialmente.

Por otra parte, la velocidad lineal se ajusta mediante técnicas de control borroso. Se definen una serie de reglas de comportamiento y después de realizar los ajustes y pruebas correspondientes, se verifica que proporcionan la velocidad lineal apropiada en función de distintos parámetros. En este proyecto se han elegido como los parámetros más representativos para la reducción o aumento de la velocidad: el radio de la trayectoria, la distancia al destino y la distancia a un obstáculo

El parámetro de distancia al obstáculo evalúa el grado de peligrosidad que tiene cualquier acción para el robot, que tomará la decisión más apropiada en cada caso.

Por otro lado, el control de la distancia a la posición objetivo evita frenazos indeseables al final de la trayectoria, consiguiendo una parada suave al terminar el recorrido.

Estos controles en conjunto dotan al robot de un comportamiento inteligente que le permitirá desplazarse de un sitio a otro evitando choques con los distintos obstáculos que existan en el medio donde se encuentre, ya sean fijos o móviles.

Tras la implantación de este software de control a un robot, se realiza una extrapolación del mismo a n-robots para conseguir implementar finalmente el controlador multi-robot que se buscaba.

En el desarrollo de este último punto, se dota a cada robot con la misma lógica de comportamiento pero con diferente prioridad de avance. Este cambio posibilita que un robot con baja prioridad se frene ante la presencia de otro con mayor prioridad, facilitando el paso de éste último.

La robótica es un campo que genera un gran interés, sin embargo, la inversión necesaria para acceder al hardware resulta prohibitiva en ocasiones. La simulación por ordenador es una alternativa válida y de bajo coste para los que consideran el esfuerzo económico un impedimento o simplemente el proyecto se encuentra en una fase previa en la que todavía no es necesario el salto a las simulaciones reales. Incluso a parte de estas razones, son muchos más los motivos que hacen a la simulación robótica por ordenador una opción interesante; movilidad, adaptabilidad, gestión de recursos, prototipado, reusabilidad o control de tiempo.

Debido a la importancia de la simulación hoy día y con el objetivo de facilitar las que se vayan a realizar en el futuro, se implementa una interfaz gráfica de usuario que permite introducir trayectorias de forma rápida y sencilla, a la vez que muestra información e interactúa con el usuario durante el proceso de simulación.

Para poder trasladar estas herramientas software al soporte real se hará uso del entorno de desarrollo de aplicaciones robóticas Player/Stage, que dispone de un simulador y un servidor que también puede emplearse para la conexión al robot real. De este modo se dispone de una importante herramienta software capaz de realizar las simulaciones necesarias, pero con una gran flexibilidad para dar el salto de forma sencilla a la simulación real en el robot.

Una vez programada la herramienta, se realiza la puesta a punto del software para optimizar el comportamiento de los robots en las diferentes situaciones que se presenten y comprobar el correcto funcionamiento del sistema.

1. Introducción

1.1 Antecedentes

Los robots móviles son cada vez más empleados en todo tipo de situaciones en el interior de edificios para la ayuda al ser humano en diferentes tareas y también al aire libre para usos como la silvicultura, la minería, búsqueda y rescate, o la inspección de sitios peligrosos. La movilidad del robot es uno de los principales problemas que se presentan sobre todo en espacios interiores debido a que los movimientos están condicionados por la geometría del entorno en el que se encuentra.

El control automático de un sistema consiste en corregir éste de forma autónoma hasta conseguir el comportamiento deseado; por tanto, se trata de actuar automáticamente sobre el sistema.

Generalmente el control automático se realiza mediante un lazo de realimentación ("feedback") que utiliza la información sobre el comportamiento del sistema para actuar en uno o en otro sentido sobre él mismo hasta alcanzar el comportamiento deseado, esto quiere decir que se actúa sobre el sistema en función de la desviación (error) entre el efecto deseado y el efecto real observado.

Un control automático realimentado necesita básicamente tres componentes como se ve en la siguiente imagen: el controlador que actúa sobre el sistema en función de la desviación entre el efecto deseado y el real, el propio sistema que reacciona ante la acción del controlador y un sensor o medidor que observe el efecto real sobre el sistema y que realimente esta información al controlador.

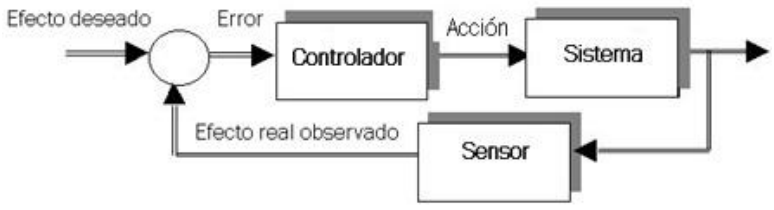


Figura 1.1. Esquema controlador básico.

A un nivel más global, el control y automatización de robots es algo que está a la orden del día, en la actualidad existen numerosos grupos de investigación en universidades y empresas punteras y referentes de todo el mundo que diseñan, investigan y desarrollan nuevos sistemas de control para aplicaciones complejas. Desde robots dotados con instrumentación avanzada (sensores de visión, de fuerza, etc.) que son controlados de forma autónoma o de forma remota para llevar a cabo operaciones médicas (laparoscopia) u operaciones en medios peligrosos (manipulación en centrales nucleares) o actuaciones en espacios físicos reducidos (robots limpiando o reparando tuberías de distribución de agua).

Uno de los futuros retos del control automático es la implantación de controladores robustos para el guiado autónomo de vehículos en las vías de tráfico convencionales, de forma que la acción del conductor sobre el volante, freno y acelerador quedará reemplazada por el sistema de control automático, con la comodidad que supone el hecho de no ser necesaria la intervención del conductor, permitiendo incrementar el número de vehículos que circulan a la vez y reducir el número de accidentes al mismo tiempo.

Este proyecto más concretamente pretende seguir con una de las líneas de trabajo del Departamento de Electrónica de la UAH, basándose en proyectos anteriores como "*Aplicación de controladores óptimo y Borroso al seguimiento de Trayectorias de una silla de ruedas*."(1), que aplica metodologías de control óptimo y borroso a la navegación por el entorno de una silla de ruedas; posteriormente se han realizado trabajos en el departamento que estudian la generación de trayectorias, como es el caso de "*Generador de trayectorias para sistemas multirobot mediante splines cúbicas*."(2); por otro lado se aborda el mapeado para robots, permitiendo de este modo que el robot conozca determinadas características del entorno en el que se mueve "*Implementación de métodos de localización y mapeado para robots Pioneer en entorno Player/Stage*."(3), y nuevamente la generación de trayectorias sencillas es motivo de estudio en el proyecto "*Generación de trayectorias con Player para la navegación de un robot móvil en espacios inteligentes*."(4).

Aquí se recogen algunas de estas ideas, de tal forma que se implementan sistemas de control óptimo y borroso particularizados al seguimiento de un robot móvil que deberá evitar obstáculos en su desplazamiento por el entorno. La trayectoria que sigue el robot se obtiene mediante splines cúbicas como sucedía en trabajos anteriores, de modo que aquí los coeficientes que delimitan cada tramo de la spline son parámetros de entrada al sistema de control. En este proyecto también se tiene en cuenta la información que proporciona el mapa de la zona, y se incorpora como elemento condicionante dentro de los factores de decisión al clasificar los diferentes tipos de obstáculos que puede encontrar el robot a su paso.

1.2 Objetivos del proyecto

Este trabajo se encuentra dentro de los proyectos de navegación de robots móviles realizados en el Departamento de Electrónica utilizando robots Pioneer y el entorno Player/Stage. Más concretamente, se centra en el diseño y programación de un controlador basado en diferentes técnicas:

- Generación de Trayectorias mediante Splines cúbicas. La trayectoria que recorrerá el robot será calculada de forma automática a partir de puntos intermedios de paso proporcionados por el usuario. Al emplear splines cúbicas se garantiza suavidad y continuidad en los cambios de tramo, evitando cualquier brusquedad indeseada a lo largo del recorrido.

- Control Óptimo. Ajusta la velocidad angular del robot en cada instante para seguir de forma precisa la trayectoria deseada. El cálculo de la velocidad angular intenta minimizar la desviación del robot atendiendo al error lateral y de orientación con respecto a la posición de consigna que se obtiene a partir de los datos proporcionados por la odometría (GPS).

- Control Borroso. Una vez el robot es capaz de recorrer la trayectoria, se introduce como mejora un control inteligente que ajusta la velocidad lineal en función de otros parámetros adicionales como son la distancia al objetivo, el radio de curvatura y la distancia al obstáculo más cercano, ya sea estático o dinámico.

Además, en el proyecto se extrapolan estas técnicas de control para su aplicación en múltiples robots, siendo posible abordar el problema de generación de trayectorias en un entorno con varios robots al añadir reglas de prioridad que evitarán choques entre ellos.

Adicionalmente se almacena el mapa de la zona por la que se moverá el robot para determinar los posibles obstáculos estáticos que deberá evitar durante su recorrido. De esta forma se consigue controlar un espacio con múltiples robots, donde cada uno recorre su propia trayectoria a la vez que evalúa su entorno para evitar colisiones entre ellos y cualquier obstáculo inesperado que aparezca.

El diseño y pruebas de la aplicación se realizará en el entorno Player/Stage. Esta plataforma permite realizar la simulación del sistema y posteriormente trasladar el código generado para su ejecución con los robots reales.

1.3 Estructura del documento

En este apartado se explicará la estructura del documento, que trata de exponer todo lo necesario para comprender el trabajo realizado, así como las conclusiones extraídas de la interpretación de resultados.

En el punto 2 se hará una breve introducción al entorno Player/Stage, explicando su funcionamiento, los diferentes elementos que lo componen y las posibilidades que ofrece para el desarrollo de aplicaciones robóticas.

El tercer punto de la memoria aporta una visión global del sistema completo, muestra un esquema con las diferentes partes que intervienen, explica su funcionalidad y define las relaciones entre ellas.

El punto número 4 comprende la generación de trayectorias mediante splines cúbicas. Aquí se exponen los detalles del algoritmo que se ha desarrollado para calcular la trayectoria, se describe el diseño adoptado, la comunicación con el controlador y finalmente se incluyen ejemplos ilustrativos con los resultados obtenidos.

A lo largo del quinto punto se explica a fondo el proceso de diseño del controlador. Abordando por separado los controles Óptimo y el Borroso, se describe y justifica la aplicación de los fundamentos teóricos al desarrollo del controlador para dar respuesta a los objetivos que se persiguen.

El punto 6 trata el desarrollo de la aplicación. Este apartado incluye los detalles técnicos de la implementación, el enfoque de desarrollo adoptado, la conexión de las diferentes partes que componen la aplicación y su integración en el sistema principal.

En el séptimo punto se explican los cambios que han sido necesarios para ampliar el sistema para N robots. Se expone la solución adoptada para establecer la configuración y puesta en marcha del sistema con un número variable de robots.

Durante el octavo punto se analizan los resultados obtenidos en cada uno de los controles por separado, realizando simulaciones con distinta configuración a fin de comparar la respuesta del robot en cada escenario.

Finalmente se incluyen las conclusiones, planos, el presupuesto y planificación, un manual de usuario para el manejo de la aplicación desarrollada, la bibliografía que se ha consultado para la realización de este trabajo y un anexo con los fundamentos teóricos de lógica borrosa aplicados en el diseño del controlador.

2. Entorno Player/Stage

Este trabajo se desarrolla en el entorno de programación de aplicaciones robóticas Player/Stage. A continuación se exponen brevemente los conceptos básicos para comprender cómo funciona y los pasos que se han de seguir para utilizarlo.

2.1 Player

Player es un servidor que permite controlar los dispositivos de un robot y obtener información de sus sensores. Es una capa de software que abstrae los detalles del hardware del robot, independizándolos del mismo. Los algoritmos de control del robot funcionarán como clientes de Player (a través de sockets TCP/IP). Así podemos controlar el robot enviando mensajes que sigan el protocolo de comunicación de Player o llamando a funciones de las librerías de Player (en nuestro caso libplayerc, ya que programaremos en lenguaje C), que nos abstraen de los detalles de comunicación. La distribución actual de player incluye librerías en diversos lenguajes como son C, C++, Java, Phyton, etc.

Puesto que Player usa el modelo cliente/servidor basado en sockets TCP, la aplicación cliente son los programas de control escritos por el diseñador que se pueden escribir en diversos lenguajes de programación y pueden ser ejecutados en cualquier computadora con conectividad a la red del robot. Además Player soporta múltiples conexiones concurrentes de clientes a los dispositivos, creando nuevas posibilidades de control distribuido y colaborativo.

Player trabaja creando varias capas de abstracción, ocultando el control de alto nivel del cliente con la implementación de bajo nivel específica del hardware. Al hacer esto se puede crear código modular, intercambiable entre hardware y proyectos. Esto se logra utilizando conceptos tales como cliente, servidor y proxies.

Los elementos básicos en Player se muestran en la Figura 2.1:

- Interfaz: Es una especificación de cómo interactuar con ciertas clases de sensores, actuadores o algoritmos de robots. Las interfaces definen la sintaxis y semántica de los mensajes que se puede intercambiar con entidades de la misma clase.

- Driver: es una pieza de software que se comunica con los sensores, actuadores o dispositivos del robot, traduciendo sus entradas y salidas

para adaptarse a una o más interfaces. El driver esconde la especificación de cualquier entidad haciéndolo parecer el mismo que cualquier otra entidad de su misma clase.

- Dispositivo: un driver sirve de límite a una interface, y tiene una dirección dada, dando lugar a un dispositivo de Player. Todos los mensajes en Player ocurren entre dispositivos a través de las interfaces. A los drivers, aunque realizan la mayoría del trabajo, nunca se acceden directamente.

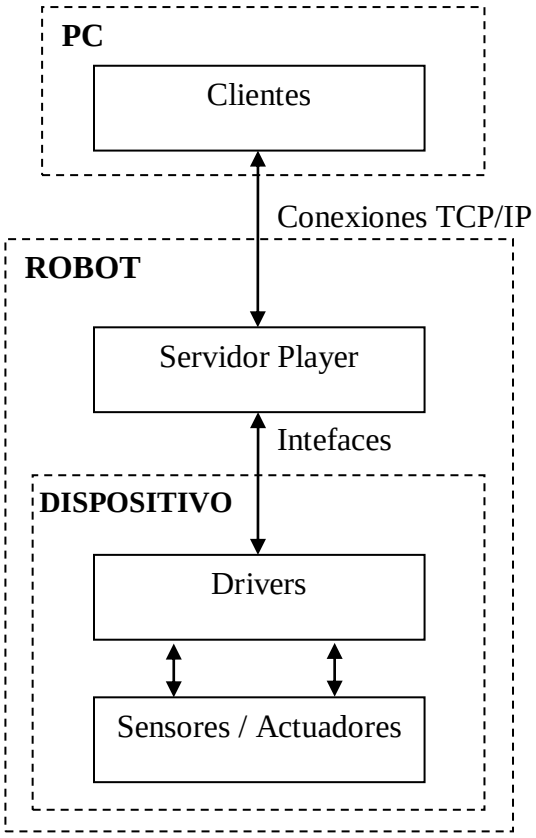


Figura 2.1. Esquema Player/Stage.

Player incluye varias interfaces predefinidas para los algoritmos y dispositivos en robótica. El objetivo de un driver es crear un vínculo entre los dispositivos físicos y la interface predefinida que mejor lo representa. Existen dos tipos de drivers en player, el driver normal de player y el driver plugin, el plugin tiene la ventaja de que no necesita recompilar player para compilar el driver. En el caso de driver plugin se enlaza con el servidor en tiempo de ejecución. El driver implementa métodos estándares para comunicarse con los proxies, un Proxy es un estándar de comunicación definido por Player para un objeto dado, tal como un sonar, el movimiento de un vehículo, etc.

Los programas realizados con Player normalmente siguen estos pasos:

- Establecer la conexión con el robot, a través de un objeto de la clase PlayerClient.
- Conectarse a los dispositivos con los que queremos interactuar.
- Entrar en un bucle en el que se leerá la información de los sensores y se enviarán instrucciones a los actuadores. Player se comunica con el robot a intervalos regulares, un número determinado de veces por segundo.

Los ficheros de configuración de Player (.cfg) contienen información sobre los dispositivos del robot. En caso de utilizar Stage, estos no serán reales, sino simulados.

Para la configuración de dispositivos del robot, Player distingue entre interfaz y driver. Una interfaz es una serie de funcionalidades que proporciona un determinado dispositivo, mientras que un driver es una implementación concreta de una interfaz.

La documentación de Player incluye una referencia de todas las interfaces y los drivers que vienen implementados en la distribución actual del software, aunque por supuesto el usuario puede definir los suyos propios. En el fichero .cfg aparecerán cada uno de los drivers del robot definidos con la palabra clave “driver” y seguidos de sus características. Un mismo driver puede implementar diferentes interfaces.

2.2 Stage

Stage es un simulador para múltiples robots móviles en 2D perteneciente al proyecto Player. Se puede usar en conjunción con Player para realizar pruebas iniciales de nuestros programas o si no disponemos de un robot real donde probarlos. El programa que realicemos es independiente de que trabajemos con el robot real o con el simulador Stage. Se emplea como plugin con Player, es la forma más común de usar Stage, o como una librería de C; En este último caso, ya no se usarían las interfaces del Player, sino a través de nuestros propios programas C.

Para ejecutar el simulador Stage como plugin de Player se utilizan dos archivos:

Un archivo de configuración de Player .cfg, contendrá la descripción para el control de los dispositivos. Otro archivo, referenciado al archivo de configuración, que contendrá la descripción del mundo (entorno) donde va a desplazarse el robot, el tamaño de la ventana visual, así como las referencias a los archivos de descripción de los dispositivos empleados, incluyendo al propio robot.

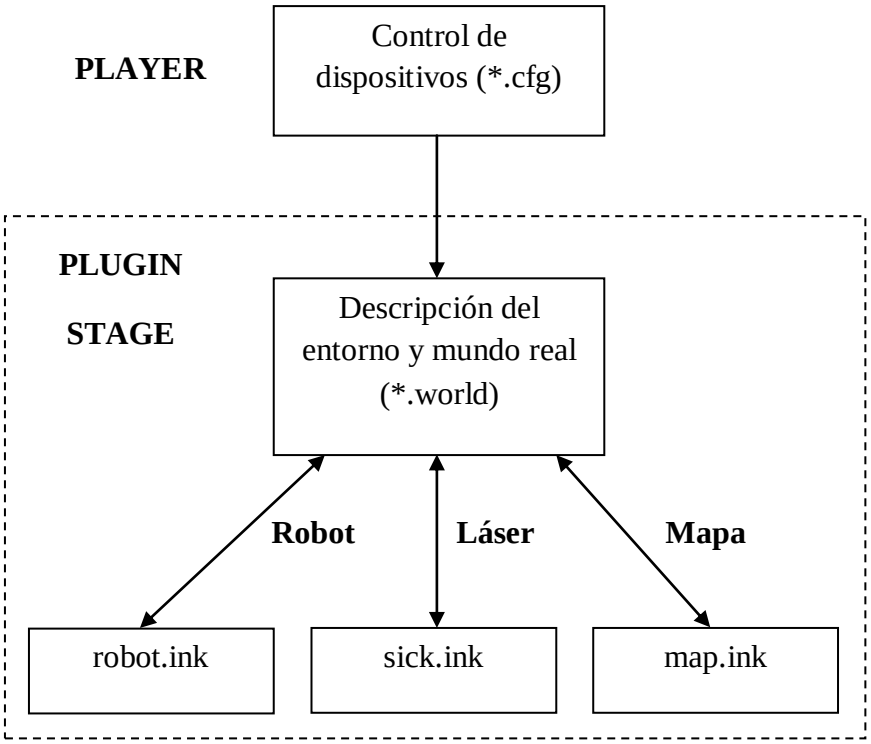


Figura 2.2. Plugin Stage.

De esta forma, Stage nos permite hacer uso de robots virtuales dotados de un amplio número de sensores y actuadores simulados que son capaces de interactuar con el entorno de simulación.

Stage es muy adecuado para investigación de sistemas autónomos multi-agente puesto que provee de modelos simples que conllevan un bajo consumo computacional, en lugar de emular cada dispositivo con gran fidelidad ya que la carga computacional en este caso sería elevada.

Para trabajar con una simulación en Stage se emplea un driver especial denominado “Stage”. Este driver deberá aparecer en el archivo de configuración (.cfg). Con esto se indicará que se hace uso de una simulación en lugar de robots reales. El archivo .world contendrá la configuración detallada del mundo (entorno) y del robot o robots a simular.

En Stage un modelo es un objeto, puede ser un robot, un dispositivo, un entorno, etc, cuyo comportamiento puede simular Stage. Todos los modelos de Stage derivan de uno básico; con un modelo basado en éste se puede representar desde un robot a un mapa de entorno, en función de las necesidades.

3. Visión general del sistema

El diseño del controlador comprende varios subsistemas que trabajan de forma coordinada para lograr el objetivo propuesto. En este proyecto se ha empleado el entorno Player/Stage para simular el sistema completo, sin embargo también es posible su aplicación en robots reales utilizando el servidor Player.

Por un lado se ha desarrollado un programa en Matlab/Octave para abordar la generación de trayectorias mediante splines cúbicas a partir de puntos de paso definidos por el usuario. Con las trayectorias calculadas, se almacenan los coeficientes de las splines en su correspondiente fichero para que sean recuperadas más adelante por el programa principal de control. Este programa, escrito en C, constituye la parte principal de la aplicación ya que será el encargado de realizar el seguimiento y control del robot hasta su llegada al destino.

En la Figura 3.1 se aprecia la estructura global del sistema, que incluye al controlador y su relación con el resto de actores que intervienen:

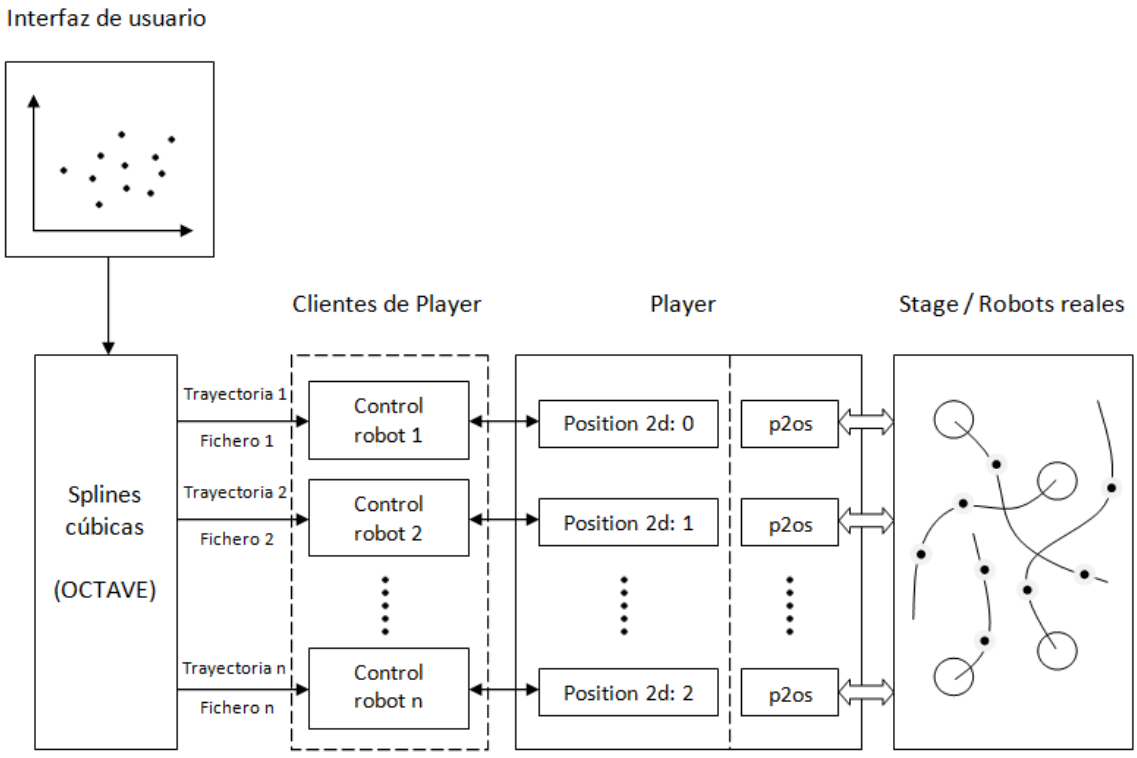


Figura 3.1. Esquema del sistema completo.

El programa cliente se comunica con el servidor Player a través de una conexión TCP y es el responsable de guiar al robot durante su recorrido. En primer lugar, este programa accede a los ficheros que contienen los coeficientes calculados para cada tramo de la spline. Teniendo en cuenta que la posición real del robot es conocida a través de GPS, podrá calcular la desviación que presenta en cada instante respecto a su posición de consigna y aplicará las correcciones oportunas para minimizar ese error.

Para que el robot comience a moverse y sea capaz de seguir el recorrido se han introducido dos controladores independientes, uno por cada variable a considerar; la velocidad lineal (V) se calculará aplicando técnicas de control borroso, y la velocidad angular (Ω) será obtenida mediante control óptimo.

El sistema también hace uso de la información proporcionada por los sensores del robot para evaluar la distancia a cualquier obstáculo. Además, diferencia entre las paredes o muros que presenta el entorno y cualquier otro punto de impacto que sea detectado durante el recorrido, lo que permite clasificar los obstáculos y conocer con más precisión la amenaza que representan en cada caso.

Ambos controladores se encuentran implementados dentro del programa cliente que aplica la lógica de control, realiza un seguimiento del robot a lo largo de la trayectoria, ajusta el movimiento para que la desviación sea mínima y garantiza una rápida reacción ante cualquier evento como podría ser la aparición de un obstáculo no esperado, llegando incluso a parar el robot si es necesario.

La aplicación admite varios clientes conectados simultáneamente al servidor Player, cada uno siguiendo su propia trayectoria. Para evitar colisiones se introduce una nueva configuración en la lógica del programa cliente que establece diferentes prioridades de avance para cada robot.

Dado que la aplicación consta de varios sistemas que funcionan de forma independiente, resultando tediosa la ejecución de cada uno en el orden establecido, era necesario encontrar una solución para que todo el proceso de configuración y puesta en marcha fuera transparente al usuario. Con esta finalidad se ha desarrollado un script que automatiza el proceso y recoge los datos necesarios a través de una interfaz gráfica de usuario.

El enfoque que se ha seguido en los distintos apartados, así como los detalles de implementación serán explicados a fondo en los siguientes capítulos de la memoria, que abordan por separado cada una de las partes implicadas en el desarrollo del controlador.

4. Generación de trayectorias

4.1 Introducción

Los diferentes robots que puedan existir, se desplazarán por un espacio común siguiendo un recorrido predeterminado; a este recorrido se le conoce como trayectoria. Una opción para calcularla es mediante splines cúbicas (2). Una spline cúbica, es una curva diferenciable definida en porciones mediante polinomios generalmente de tercer grado para las abscisas y las ordenadas. En los puntos de la spline que delimitan cada tramo, se establecen ciertas condiciones de continuidad impuestas a través de derivadas primeras y segundas. A través de las formas polinómicas, se puede describir igualmente el ángulo que tendrá el punto inicial de la trayectoria así como el del punto final.

En los problemas de interpolación, se utiliza a menudo la interpolación mediante splines porque da lugar a resultados similares a los que ofrecería un polinomio de orden elevado, requiriendo solamente el uso de polinomios de bajo grado y evitando así las oscilaciones, indeseables en la mayoría de las aplicaciones.

Para el ajuste de curvas, los splines se utilizan para aproximar formas complicadas. La simplicidad de la representación y la facilidad de cómputo de los splines los hacen populares para la representación de curvas simplificadas.

Las trayectorias diseñadas con splines, son formas polinómicas interpoladas, es decir, una aproximación a un caso ideal. El diseñador de las trayectorias podrá fijar una serie de puntos en el espacio por los que los robots deberán pasar forzosamente. Dichos puntos coincidirán con los puntos que limitan cada tramo de la spline. Además también se puede especificar la “brusquedad” del trazado de las curvas a través de un parámetro de suavidad que estará implícito a partir de las condiciones iniciales de cálculo como se verá más adelante.

4.2 Interpolación mediante splines cúbicas

Las trayectorias se generan mediante un algoritmo de interpolación basado en curvas parametrizadas, que representan una curva Q en un plano bidimensional (x,y) en función de un parámetro de avance denominado ‘u’, de tal forma que se seguirá una expresión de la forma:

$$Q(u) = (X(u), Y(u))$$

donde se puede apreciar que X(u) e Y(u) son función del parámetro 'u' mencionado. En concreto, X(u) representa la variación de las abscisas en función del parámetro 'u', mientras que Y(u) lo hace con las ordenadas.

En la siguiente figura se muestra de una forma gráfica la forma de Q(u) y descompuesta en X(u) e Y(u).

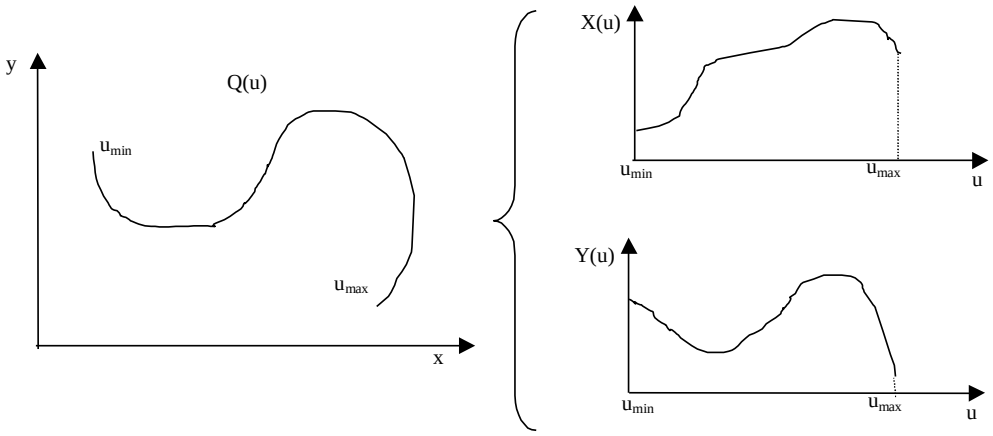


Figura 4.1. Parametrización en función del avance de la variable u.

Para implementar las funciones X(u) e Y(u) de una forma simple se usan polinomios. Teniendo en cuenta que es posible definir en la trayectoria tantos puntos intermedios como sean necesarios, se dividirá en varios tramos (uno por cada dos puntos consecutivos) estando definido cada uno por un polinomio de tercer grado.

Para conseguir que cada tramo sea independiente del parámetro 'u', habrá que normalizar en cada tramo dicho parámetro con un valor inicial 0 y un valor final 1. Haciéndolo de este modo, cada tramo estará definido por dos polinomios de tercer grado del tipo:

$$X_i(u) = a_{ix} + b_{ix} \cdot u + c_{ix} \cdot u^2 + d_{ix} \cdot u^3 \quad \text{Con } 0 \le u \le 1$$

$$Y_i(u) = a_{iy} + b_{iy} \cdot u + c_{iy} \cdot u^2 + d_{iy} \cdot u^3 \quad \text{Con } 0 \le u \le 1$$

De forma gráfica, teniendo en cuenta la división en tramos y las anteriores expresiones de X(u) e Y(u):

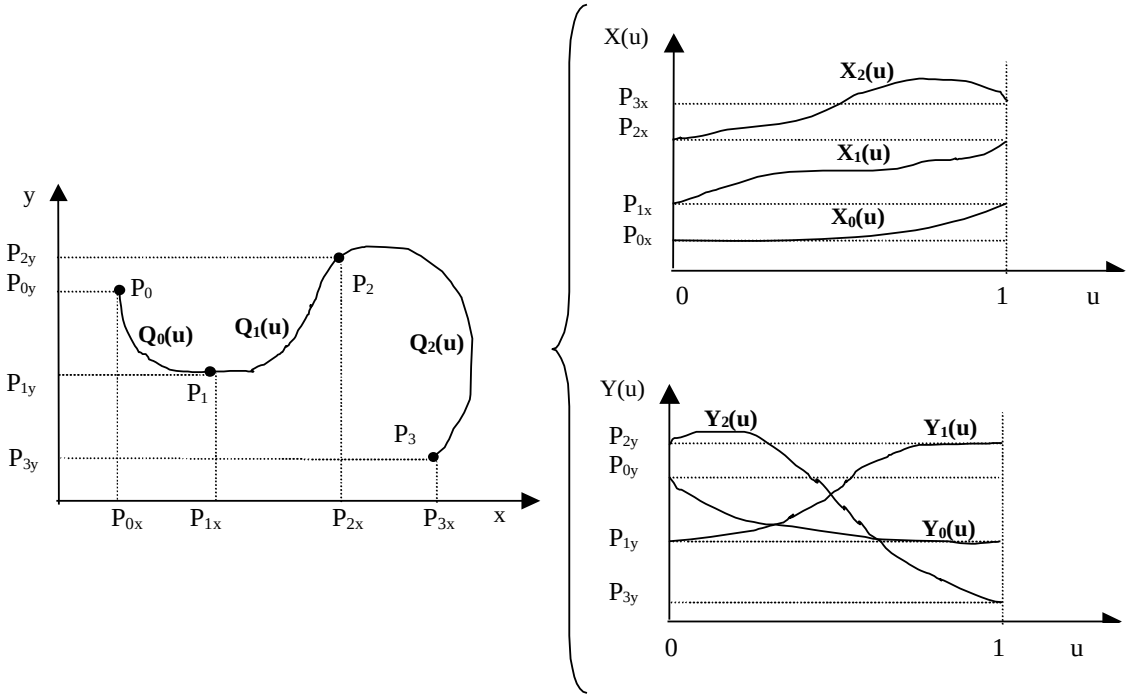


Figura 4.2. Parametrización en tramos de la trayectoria.

Para realizar el cálculo de los coeficientes de los polinomios que definirán cada tramo de las expresiones de X(u) e Y(u), se deben imponer una serie de condiciones que en este caso serán continuidad en la primera y segunda derivada en todos los puntos intermedios de la spline asegurando además una determinada pendiente (derivada primera) de la curva para sus puntos extremos, que se hace coincidir con la orientación inicial y final deseada del robot.

Se expone a continuación el desarrollo del algoritmo necesario para calcular los cuatro coeficientes de cada tramo de las curvas tomando como ejemplo el desarrollo de Y(u), siendo idéntico al de X(u).

El polinomio para el tramo 'i' de la curva Y(u) tiene la expresión:

$$Y_i(u) = a_{iy} + b_{iy} \cdot u + c_{iy} \cdot u^2 + d_{iy} \cdot u^3 \qquad \text{para } 0 \le u \le 1$$

Si se tienen en cuenta las condiciones mencionadas anteriormente y que se deben cumplir, resulta el siguiente desarrollo:

La trayectoria debe pasar por los puntos que enlaza, es decir P_{iy} y $P_{(i+1)y}$:

$$Y_i(0) = P_{iy} = a_{iy}$$

$$Y_i(1) = P_{(i+1)y} = a_{iy} + b_{iy} + c_{iy} + d_{iy}$$

Las derivadas primeras en los puntos que enlazan deben tomar un determinado valor que asegure continuidad en primera y segunda derivada en los puntos intermedios o bien un valor determinado en los puntos extremos:

$$Y_i^{(1)}(0) = D_{iy} = b_{iy}$$

$$Y_i^{(1)}(1) = D_{(i+1)y} = b_{iy} + 2c_{iy} + 3d_{iy}$$

A partir de las cuatro ecuaciones obtenidas, es posible obtener los cuatro coeficientes del polinomio buscado. El resultado es el siguiente:

$$a_{iy} = P_{iy}$$

$$b_{iy} = D_{iy}$$

$$c_{iy} = 3(P_{(i+1)y} - P_{iy}) - 2D_{iy} - D_{(i+1)y}$$

$$d_{iy} = 2(P_{iy} - P_{(i+1)y}) + D_{iy} + D_{(i+1)y}$$

Una vez obtenidos estos coeficientes, únicamente queda por determinar las derivadas primeras en los puntos por los que pasa el polinomio. Para esto, también se deberán cumplir una serie de condiciones.

En los puntos intermedios, las derivadas primeras deberán tomar el valor apropiado para asegurar continuidad en la segunda derivada de los polinomios que enlazan. Es decir, los polinomios $Y_{i-1}(u)$ y $Y_i(u)$, enlazados por el punto P_i sean:

$$Y_{i-1}^{(2)}(1) = Y_i^{(2)}(0)$$

$$2c_{(i-1)y} + 6d_{(i-1)y} = 2c_{iy}$$

Si se sustituyen los coeficientes obtenidos anteriormente en estas expresiones, tenemos:

$$D_{(i-1)y} + 4D_{iy} + D_{(i+1)y} = 3(P_{(i+1)y} - P_{(i-1)y})$$

Por otro lado, en los puntos extremos hay que imponer sus pendientes. Para lograrlo, se parte de un valor de orientación inicial de la curva y un valor de orientación final de la misma. Estas orientaciones inicial y final deben ser traducidas a pendientes (o derivadas) inicial y final de los polinomios $X_0(u)$, $Y_0(u)$ y $X_{(n-1)}(u)$ y $Y_{(n-1)}(u)$, siendo 'n' el número de tramos de la spline.

Según todo lo anterior, en el primer punto P_0 se deduce que la pendiente de la curva $Q_0(u)$ deberá ser la tangente de la orientación inicial θ_{ini}:

$$tg(\theta_{ini}) = \left.\frac{dy}{dx}\right|_{P_0} = \left.\frac{dy/du}{dx/du}\right|_{P_0} = \frac{\mu \cdot \sin(\theta_{ini})}{\mu \cdot \cos(\theta_{ini})}$$

donde cuanto mayor sea el parámetro μ, la pendiente impuesta afectará a más valores de 'u' (será más suave).

De la misma forma que para el primer punto, en el último punto P_n se sabe que la pendiente de la curva debe ser la tangente de la orientación final θ_{fin}:

$$tg(\theta_{fin}) = \left.\frac{dy}{dx}\right|_{P_n} = \left.\frac{dy/du}{dx/du}\right|_{P_n} = \frac{\mu \cdot \sin(\theta_{fin})}{\mu \cdot \cos(\theta_{fin})}$$

Por tanto, al sistema de ecuaciones para determinar los coeficientes del polinomio $Y_0(u)$ de la spline, hay que añadir el valor de la primera derivada en el punto P_0:

$$D_{0y} = \mu \cdot \sin(\theta_{ini})$$

Y de la misma forma, al sistema de ecuaciones para determinar los coeficientes del polinomio $Y_{(n-1)}(u)$ de la spline hay que añadir el valor de la primera derivada en el punto P_n:

$$D_{ny} = \mu \cdot \sin(\theta_{fin})$$

Como existen (n+1) puntos, habrá que determinar sus derivadas mediante un sistema de (n+1) ecuaciones (visto anteriormente), resumidas matricialmente de la siguiente manera:

$$\begin{bmatrix} 1 & & & & & & & \\ 1 & 4 & 1 & & & & & \\ & 1 & 4 & 1 & & & & \\ & & 1 & 4 & 1 & & & \\ & & & . & . & . & & \\ & & & & 1 & 4 & 1 & \\ & & & & & & 1 & \end{bmatrix} \cdot \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ . \\ . \\ . \\ D_n \end{bmatrix} = \begin{bmatrix} \mu \cdot \sin(\theta_{ini}) \\ 3 \cdot (P_{2y} - P_{0y}) \\ . \\ . \\ . \\ 3 \cdot (P_{ny} - P_{(n-2)y}) \\ \mu \cdot \sin(\theta_{fin}) \end{bmatrix}$$

Empezando por arriba, y eliminando el primer valor de cada fila utilizando la fila inmediatamente anterior y escalando la diagonal:

$\gamma_0=0$

for i=1 step 1 until (n-1) do

$\gamma_i=1/(4-\gamma_{(i-1)})$

endfor

Ahora, realizando las mismas operaciones en el vector de términos independientes:

$\delta_0 = \mu \text{ sen}(\theta ini)$

for i=1 step 1 until (n-1) do

$\delta_i=(3(P_{(i+1)y}-P_{(i-1)y})- \delta_{i-1}) \gamma_i$

endfor

$\delta_n = \mu \text{ sen}(\theta fin)$

El sistema de ecuaciones queda de la siguiente forma:

$$\begin{bmatrix} 1 & \gamma_0 & & & & & \\ & 1 & \gamma_1 & & & & \\ & & 1 & \gamma_2 & & & \\ & & & 1 & \gamma_3 & & \\ & & & & . & . & \\ & & & & & 1 & \gamma_{n-1} \\ & & & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ . \\ . \\ . \\ D_n \end{bmatrix} = \begin{bmatrix} \delta_0 \\ \delta_1 \\ . \\ . \\ . \\ . \\ \delta_n \end{bmatrix}$$

Y se resuelve fácilmente mediante el siguiente bucle:

$D_n = \delta_n$

for i=(n-1) step -1 until 0 do

$D_i = \delta_i - \gamma_i \; D_{(i+1)}$

endfor

4.3 Implementación

Dentro de las diferentes tareas que han de ser abordadas por el controlador, la generación de trayectorias es el punto de partida en el que se elige y planifica la ruta a seguir. Como se ha visto, una de las principales características que se persiguen en este tipo de aplicaciones es la continuidad en los puntos que están próximos entre sí, con el objetivo de facilitar el desplazamiento del robot.

Para este cometido se introduce un primer programa desarrollado en el entorno Matlab/Octave que se encarga de recoger los puntos de paso requeridos por el usuario y en base a éstos genera la trayectoria mediante splines cúbicas.

El programa aborda el cálculo de la trayectoria en varios pasos. En primer lugar se crea la ventana de trabajo mostrando el mapa de la simulación sobre un eje de coordenadas.

Para fijar un punto de partida se recurre al archivo de texto *"Posicion.txt"* que contiene los datos de posición (x,y) y orientación iniciales del robot (θ). El script almacena la información y dibuja el robot en el lugar que le corresponde (Figura 4.3).

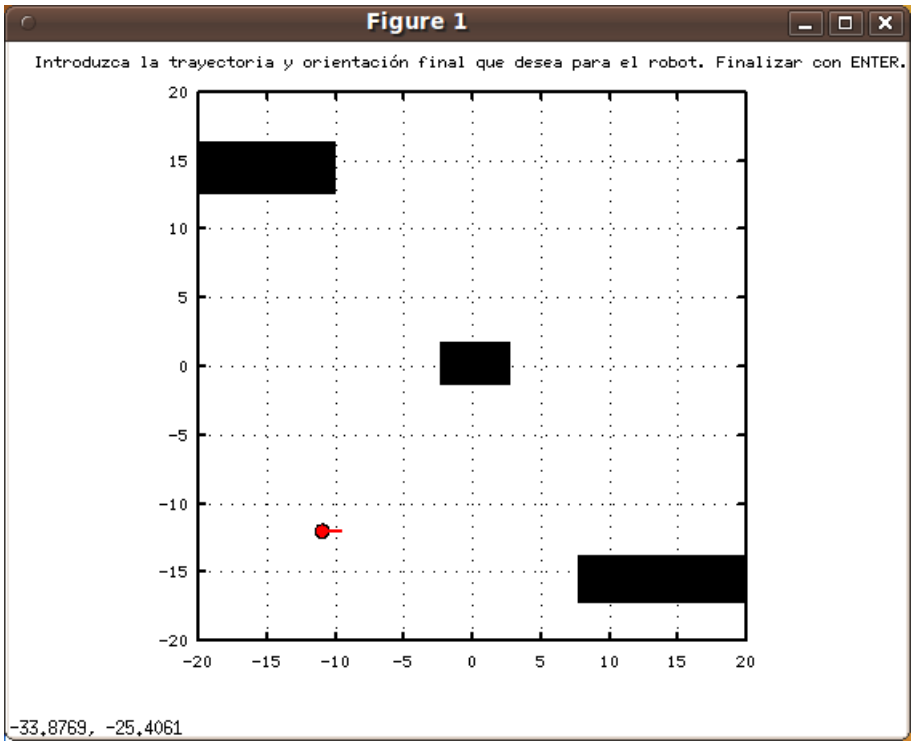


Figura 4.3. Robot en el mapa del entorno con su posición y orientación inicial.

A continuación el programa solicitará los puntos intermedios de paso y la orientación final del robot, que deben ser introducidos pinchando sobre la imagen recogiendo las coordenadas deseadas en ese mismo instante.

Con los datos recopilados por la interfaz de usuario, se dibuja sobre el mapa los puntos intermedios de la trayectoria. El último elemento se deja apartado, ya que sirve para definir la orientación final y por tanto no se trata de un punto que pertenezca a la trayectoria. Seguidamente se procede a generar la trayectoria en base a los conjuntos de puntos (x,y) ordenados, las orientaciones inicial y final del robot, y un parámetro encargado de ajustar la suavidad deseada para la curva resultante.

El script sigue los pasos expuestos para calcular la curva de abscisas y ordenadas en función de "u", devuelve los coeficientes para cada tramo y finalmente dibuja la trayectoria sobre el eje de coordenadas que contiene el mapa. El robot en la posición inicial aparece en color rojo al igual que la línea de orientación, mientras que la posición de destino se marca en azul.

Por otro lado, el sistema que se encarga de controlar el robot necesita conocer la trayectoria que se ha generado, de tal forma que los coeficientes en función de "u" calculados se almacenan en un fichero de texto denominado *"Coeficientes.txt"* que será consultado más tarde por el cliente (Figura 4.4).

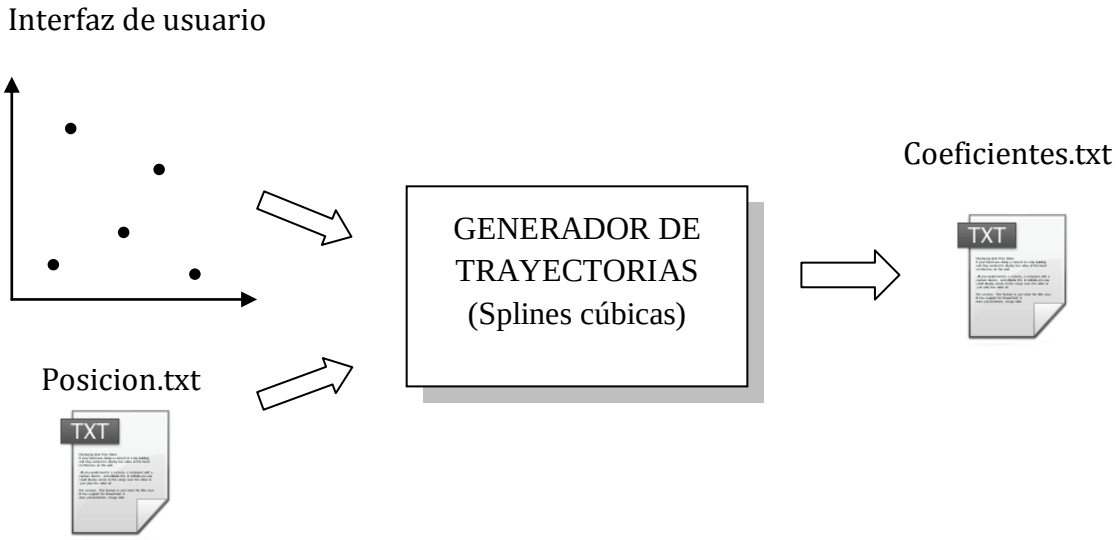


Figura 4.4. Elementos que intervienen en el Generador de Trayectorias.

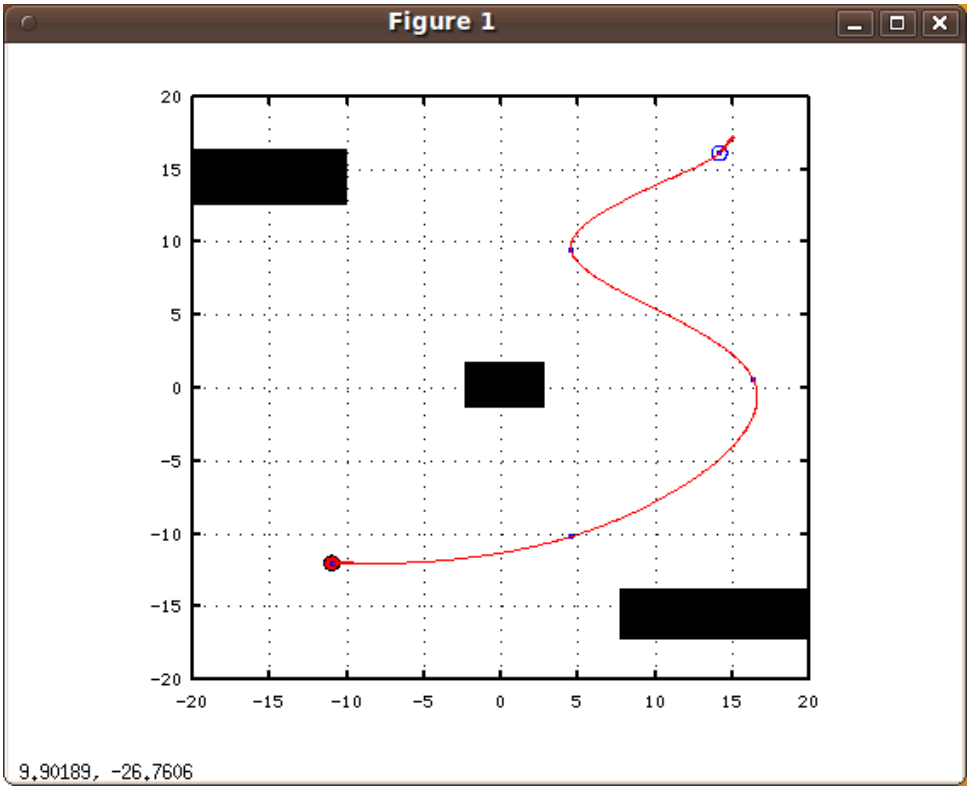


Figura 4.5. Trayectoria generada.

Algunos ejemplos adicionales a continuación (Figura 4.6 y Figura 4.7):

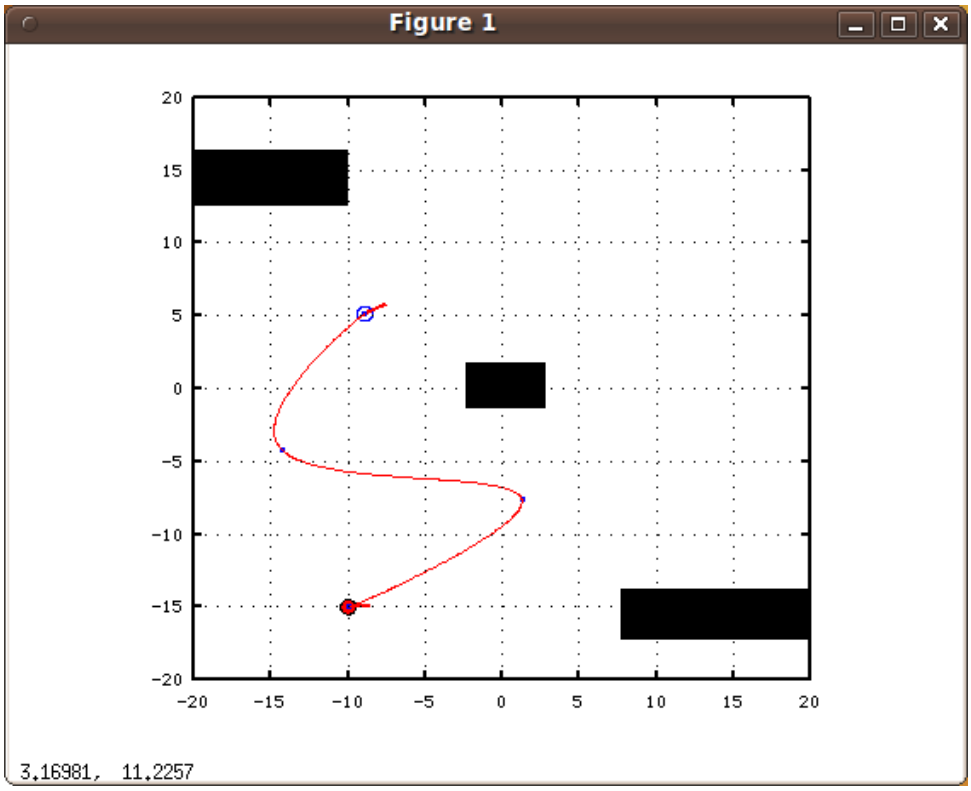


Figura 4.6. Ejemplo de trayectoria (I).

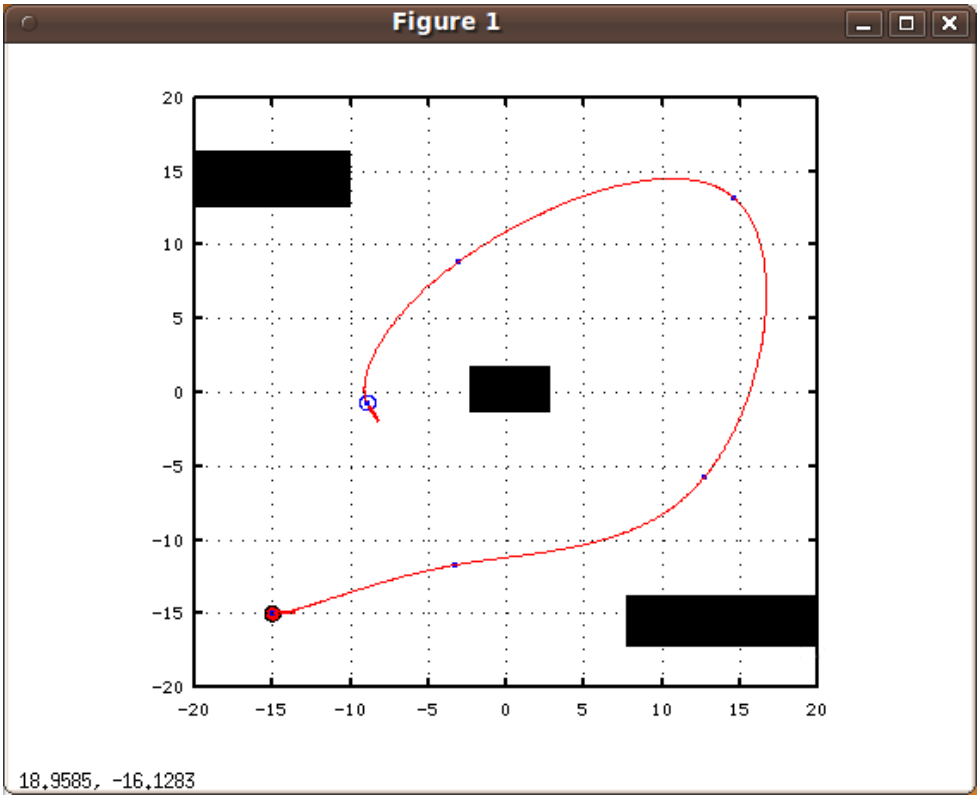


Figura 4.7. Ejemplo de trayectoria (II).

5. Diseño del controlador

5.1 Introducción

A lo largo de este capítulo se explica el proceso de diseño del controlador. Como se ha comentado anteriormente, el movimiento se consigue a través de dos controles independientes; la velocidad angular será obtenida mediante técnicas de control óptimo, y la velocidad lineal del robot se ajustará en función de las condiciones del entorno a través del control borroso.

En los apartados siguientes se detallan los pasos que se han seguido para abordar el diseño de cada sistema por separado.

5.2 Control Óptimo en V.V.E.E.

5.2.1 Introducción

Tras disponer de un método para el cálculo de trayectorias, será necesario introducir un sistema de control que capacite al robot para seguir el recorrido previsto con la mínima desviación posible. En primer lugar se abordará el diseño de un controlador que determine la velocidad angular suponiendo que el robot se mueve con una velocidad lineal constante. Una vez alcanzado este objetivo, se añadirá un control inteligente que varíe la velocidad lineal atendiendo a diferentes parámetros como son el radio de la curvatura, la distancia al destino o la distancia al obstáculo proporcionada por los sensores sonar del robot.

5.2.2 Procedimiento de diseño del controlador

Para la realización del controlador de velocidad angular (Ω) se supondrá que la posición real del robot es conocida en todo momento a través de los datos que proporciona la odometría, y la velocidad lineal (V) se considera una variable ya controlada, representando un parámetro de entrada al sistema que no es necesario calcular. De este modo la única variable de entrada a la planta desde el punto de vista de diseño es la velocidad angular (Ω). Por otro lado, las variables de salida a controlar

(x, y, θ) serían aquellas que determinan la posición del robot, como se puede observar en el diagrama de planta que se muestra en la Figura 5.1.

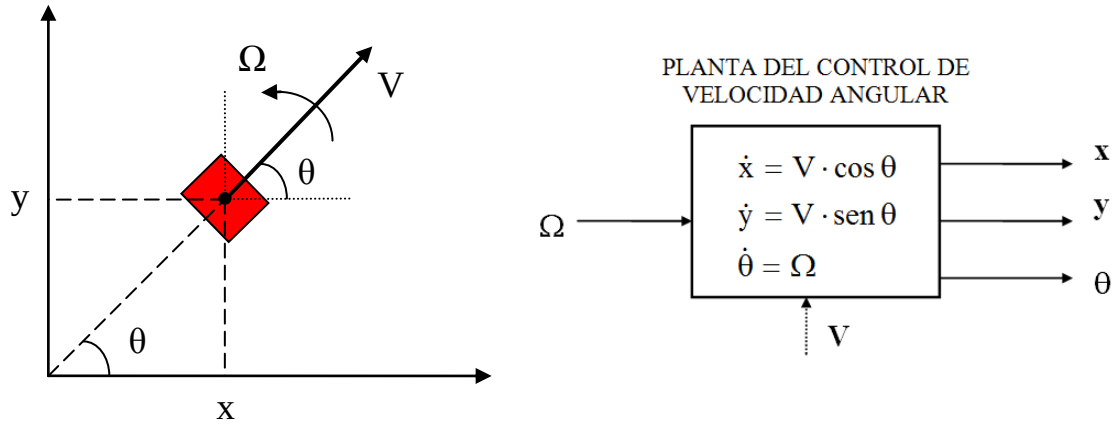


Figura 5.1. Planta para controlador de velocidad angular.

Para el diseño de este controlador se utilizarán métodos en el espacio de estados. Una de las características más atractivas de este método de diseño es que consiste en una secuencia de pasos independientes y sistemáticos, que se resumen a continuación (1).

- Representación de la planta en el espacio de estados.
- Obtención de la ley de control
- Diseño de un observador de estados.
- Combinación de la ley de control y el estimador
- Introducción de la entrada de referencia.
- Determinación de los parámetros del controlador

5.2.3 Representación de la planta en el espacio de estados

El primer paso es obtener una representación de la planta en variables de estado, de la forma:

$$\dot{\chi} = A \cdot \chi + B \cdot u \qquad \Rightarrow \text{ Ecuación de estado.}$$

$$y = C \cdot \chi + D \cdot u \qquad \Rightarrow \text{ Ecuación de salida.}$$

donde χ representa los estados del sistema, u las entradas de la planta, e y las salidas de la misma.

Las ecuaciones de la planta son:

$$\dot{x} = V \cdot \cos\theta$$
$$\dot{y} = V \cdot \operatorname{sen}\theta$$
$$\dot{\theta} = \Omega$$

En primer lugar, se debe determinar el número de variables de estado necesarias para describir el sistema. Se observa fácilmente que el número de integradores es tres, siendo éste por tanto el número de variables de estado requeridas.

A continuación se deben elegir las variables de estado. Dadas las características del sistema bajo estudio, las componentes del vector de estado serán [x y θ]', con lo cual las variables de estado coincidirán con las salidas del sistema.

Por tanto, la entrada, salidas, y variables de estado son las siguientes:

$$u \Rightarrow \Omega \quad ; \quad \chi \Rightarrow \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad ; \quad y \Rightarrow \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Por último, se deben determinar las matrices A, B, C y D que permiten relacionar las variables del sistema de la siguiente forma:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = A \cdot \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + B \cdot \Omega$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = C \cdot \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + D \cdot \Omega$$

La ecuación de salida es fácil de determinar, puesto que, como es obvio:

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad y \qquad D = 0$$

Sin embargo, la ecuación de estado no se puede expresar de la forma indicada, ya que las derivadas de los estados no son combinación lineal de los propios estados. Esto es debido a la presencia de funciones trigonométricas, que no son lineales. Además, no es posible encontrar otro vector de estado a partir de combinaciones

lineales de las variables de estado indicadas para el cual no aparezca también este problema.

Por lo tanto, ya en el primer paso de diseño aparece la dificultad de una planta no lineal, que no puede representarse de forma sencilla en el espacio de estados. Sin embargo, se continuará con el proceso de diseño suponiendo que se dispone de dicha representación, y más adelante se verá cómo introduciendo la entrada de referencia de la forma adecuada, se puede llegar a linealizar el sistema.

5.2.4 Obtención de la ley de control

En este apartado se hará un breve repaso de la técnica básica de diseño de reguladores en la teoría de control moderna, que consiste en la realimentación de los estados hacia la entrada.

La misión de todo controlador es modificar la situación de los polos de un sistema de forma que la respuesta del mismo sea la deseada.

Los polos de una determinada planta coinciden con los autovalores λ de la matriz A de su ecuación de estados, que a su vez son las soluciones de la siguiente ecuación:

$$|\lambda \cdot I - A| = 0$$

La ley de control consiste en realimentar los estados de la forma indicada en la Figura 5.2, es decir:

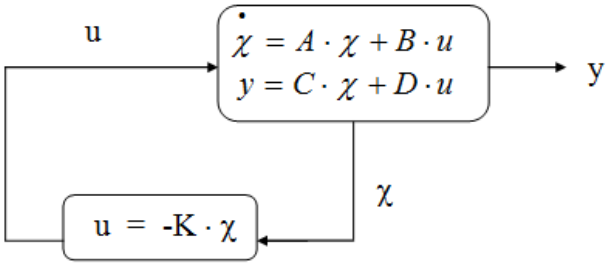


Figura 5.2. Control mediante realimentación de los estados.

$$u = - K \cdot \chi = - [\, k_1 \;\; k_2 \;\; \dots \;\; k_n \,] \cdot \begin{bmatrix} \chi_1 \\ \chi_2 \\ \vdots \\ \chi_n \end{bmatrix}$$

La nueva ecuación de estados del lazo cerrado será:

$$\dot{\chi} = A \cdot \chi + B \cdot (-K \cdot \chi) = (A - B \cdot K) \cdot \chi$$

De esta forma, los nuevos autovalores del sistema coincidirán con las raíces de:

$$|\lambda \cdot I - (A - B \cdot K)| = 0$$

con lo cual , es posible ajustar K para que los autovalores, y por tanto los polos, tengan el valor deseado según las especificaciones de diseño.

5.2.5 Introducción de la entrada de referencia. Linealización y discretización del sistema

En el apartado anterior se ha visto que la ley de control viene dada por $u=-K\cdot\chi$. Obviamente, esto no constituye la estructura completa del controlador, pues es necesario introducir la entrada de referencia del sistema, que en este caso es proporcionada por el generador de trayectorias.

Una de las estructuras más utilizadas para la construcción de un servosistema cuya salida siga el valor de una entrada de referencia es la mostrada en la siguiente figura:

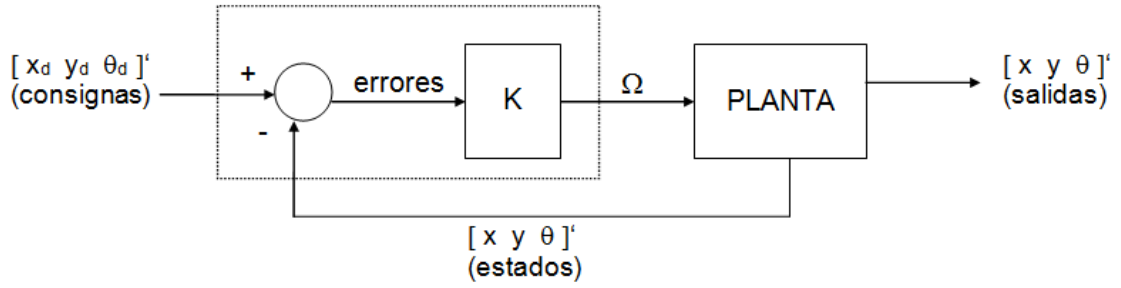


Figura 5.3. Estructura y disposición del control de velocidad angular (Ω).

Para conseguir error nulo en el seguimiento de la referencia es necesario que el sistema sea de tipo 1, lo cual en este caso se cumple con un simple control proporcional puesto que la planta ya contiene un elemento integrador.

De este modo, la introducción de la entrada de referencia debe realizarse según la figura anterior, en la que se observa que la estructura interna del controlador queda reducida a dos bloques:

- Un restador cuya función es detectar los errores del sistema
- Una matriz de ganancias K, a través de la cual se realimentan los errores hacia la entrada, y de cuyo valor dependerá la respuesta del controlador.

Llegados a este punto, existe la posibilidad de realizar un cambio de variables de estado que además permitirá linealizar el sistema de una forma muy sencilla. Dicho cambio consiste en tomar como nuevo vector de estado χe:

$$\chi_e = \chi - r = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} - \begin{bmatrix} x_d \\ y_d \\ \theta_d \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix}$$

Este nuevo vector de estado está formado por los errores del sistema, cuyos valores serán próximos a cero, de tal forma que es posible linealizar todas las ecuaciones en torno al origen, resolviendo así la no linealidad de la planta.

De esta forma, el control quedará reducido a una ley del tipo:

$$u = -K \cdot \chi_e$$

cuya misión será la de minimizar los errores del sistema, que ahora son las variables de estado.

ERRORES DEL SISTEMA: ERROR DE ORIENTACIÓN Y ERROR LATERAL

Por tanto, el objetivo del control de velocidad angular es minimizar una serie de errores en la ubicación del robot, que además han sido elegidos como variables de estado con el fin de linealizar el sistema.

$$\chi_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix}$$

donde: $x_e = x - x_d$ es el error de abscisas.

$y_e = y - y_d$ es el error de ordenadas.

$\theta_e = \theta - \theta_d$ es el error de orientación.

Antes de buscar la ecuación de estados para estas nuevas variables, existe la posibilidad de reducir el orden del sistema reuniendo los errores x_e e y_e en uno solo, correspondiente a la distancia desde la ubicación actual del robot hasta el punto de la trayectoria más cercano, que se ha llamando (x_d,y_d). De esta forma, no sólo se reduce el orden del sistema, sino que desaparece el problema del tratamiento de las distintas combinaciones de signos que surgen al trabajar con x e y como variables independientes. A este error se le denominará en adelante error lateral d_e, y junto con el error de orientación θ_e, constituyen definitivamente el vector de estado:

$$\chi_{ee} = \begin{bmatrix} d_e \\ \theta_e \end{bmatrix}$$

La siguiente figura muestra un ejemplo de los errores lateral y de orientación para una configuración concreta del robot.

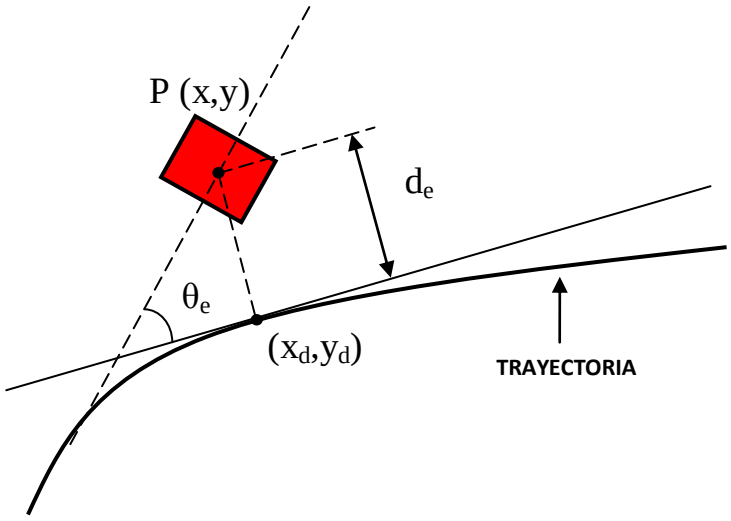


Figura 5.4. Esquema de error lateral y de orientación.

Como se observa, el error lateral d_e se define como la distancia desde la ubicación real del robot hasta el punto más cercano de la trayectoria, lo cual implica que es perpendicular a la tangente a la trayectoria en dicho punto, es decir, a θ_d. Esto, y el hecho de que pueda tomar signo positivo o negativo según se encuentre por encima o por debajo de la trayectoria, hacen que el error lateral, junto con el de

orientación, sea suficiente para caracterizar de forma exacta el error en la ubicación del robot. Proyectando las distancias entre las abscisas y ordenadas sobre la línea que une la posición real y de consigna se obtiene el error lateral:

$$d_e = -(x - x_d) \cdot sen\theta_d + (y - y_d) \cdot \cos\theta_d$$

$$\theta_e = \theta - \theta_d$$

LINEALIZACIÓN DEL SISTEMA. ECUACIÓN DE ESTADOS

La ecuación de estados que define al sistema tendrá la forma general:

$$\begin{bmatrix} \dot{d}_e \\ \dot{\theta}_e \end{bmatrix} = A_e \cdot \begin{bmatrix} d_e \\ \theta_e \end{bmatrix} + B_e \cdot \Omega$$

donde es posible linealizar las ecuaciones diferenciales (debido a que los errores oscilarán en torno a cero) , de la siguiente forma:

$$\dot{d}_e = -\dot{x} \cdot \text{sen}\,\theta_d + \dot{y} \cdot \cos\theta_d = -V \cdot \cos\theta \cdot \text{sen}\,\theta_d + V \cdot \text{sen}\,\theta \cdot \cos\theta_d =$$
$$V \cdot (\text{sen}\,\theta \cdot \cos\theta_d - \cos\theta \cdot \text{sen}\,\theta_d) = V \cdot \text{sen}(\theta - \theta_d) = V \cdot \text{sen}\,\theta_e \approx V_d \cdot \theta_e$$

$$\dot{\theta}_e = \dot{\theta} - \dot{\theta}_d \approx \dot{\theta} = \Omega$$

donde V_d es la consigna de velocidad lineal, que para el diseño de este controlador puede ser considerada como un parámetro, puesto que es controlada por el controlador de velocidad lineal. La linealización será válida siempre que el error de orientación y su seno sean aproximadamente iguales, lo cual se puede considerar que se cumple hasta los 30 grados, tal y como se muestra en la siguiente figura:

Por lo tanto, la ecuación de estados linealizada puede expresarse de la siguiente forma:

$$\begin{bmatrix} \dot{d}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} 0 & V_d \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} d_e \\ \theta_e \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \Omega$$

DISCRETIZACIÓN DEL SISTEMA

La consigna de velocidad angular sólo será actualizada a intervalos fijos de tiempo (periodo de muestreo para Ω = Ts). Por tanto, antes de obtener la ganancia de

realimentación, es necesario discretizar el sistema obtenido anteriormente, y a partir de este punto diseñar el controlador sobre el sistema discretizado.

La ecuación de estado del sistema discretizado será de la forma:

$$\chi(n+1) = G \cdot \chi(n) + H \cdot \Omega(n)$$

donde la relación entre las matrices G y H del sistema discreto con las matrices A_e y B_e del sistema continuo viene dada por ([2]):

$$G = e^{A_e \cdot Ts} \qquad \text{y} \qquad H = \left(\int_0^{Ts} e^{A_e \cdot \lambda} \cdot d\lambda \right) \cdot B_e$$

Aplicando las expresiones anteriores, se obtienen las siguientes matrices para el sistema discretizado :

$$G = \begin{bmatrix} 1 & V_d \cdot Ts \\ 0 & 1 \end{bmatrix} \quad \text{y} \quad H = \begin{bmatrix} V_d \cdot Ts^2 / 2 \\ Ts \end{bmatrix}$$

y por tanto :

$$\begin{bmatrix} d_e(n+1) \\ \theta_e(n+1) \end{bmatrix} = \begin{bmatrix} 1 & V_d \cdot Ts \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} d_e(n) \\ \theta_e(n) \end{bmatrix} + \begin{bmatrix} \frac{V_d \cdot Ts^2}{2} \\ Ts \end{bmatrix} \cdot \Omega(n)$$

CONTROLABILIDAD DEL SISTEMA

Antes de pasar al ajuste de los parámetros del controlador, debe comprobarse que el sistema es controlable. Para ello, se construye la matriz de controlabilidad, y se comprueba que su determinante es no nulo.

$$\left| \begin{bmatrix} H & G \cdot H \end{bmatrix} \right| = \left| \begin{bmatrix} \frac{V_d \cdot Ts^2}{2} & \frac{3 \cdot V_d \cdot Ts^2}{2} \\ Ts & Ts \end{bmatrix} \right| = -V_d \cdot Ts^3$$

Según esto, el sistema será controlable siempre que el robot esté en movimiento y el periodo de muestreo sea no nulo.

5.2.6 Determinación de los parámetros del controlador

Una vez representado el sistema en variables de estado, y comprobada la controlabilidad del mismo, es necesario calcular el valor de la matriz de ganancias K, para obtener la respuesta deseada. El valor de K determina el lugar de los polos del lazo cerrado, del cual dependen características importantes como la estabilidad o la velocidad de respuesta del sistema.

El método escogido para el ajuste de la matriz de realimentación es el conocido como control óptimo. El control óptimo lineal cuadrático (LQR) se basa en la elección de la matriz de realimentación K de tal forma que minimice una función de coste cuadrática. La elección de dicha función de coste, también conocida como índice de comportamiento, depende de las especificaciones del diseño.

Generalmente, suele ser conveniente minimizar un función generalizada de los estados del sistema $\chi(n)$, como:

$$J = \sum_{n=0}^{N} \chi^T(n) \cdot Q \cdot \chi(n)$$

donde $\chi^T(n)$ es la conjugada traspuesta del vector de estado, Q es una matriz real o hermítica definida o semidefinida positiva, y N es el número de periodos de muestreo en los que se ejecuta el sistema, y que puede ser finito o infinito.

Sin embargo, además de la consideración de los errores como medida del comportamiento de sistemas, con frecuencia se tiene presente la energía necesaria para la acción de control. Si el vector de estado se minimiza sin considerar la energía requerida, el diseño puede llevar a un resultado que requiera valores excesivamente elevados de la señal de control. Esto no es deseable, ya que todos los sistemas físicos están sujetos a saturación. Entonces, por razones de orden práctico, se limita también al vector de control u, mediante otro índice de comportamiento cuadrático del mismo tipo, por ejemplo:

$$T = \sum_{n=0}^{N} u^T(n) \cdot R \cdot u(n)$$

De esta forma, el índice de comportamiento completo a minimizar será del tipo:

$$\wp = \sum_{n=0}^{N} \chi^T(n) \cdot Q \cdot \chi(n) + u^T(n) \cdot R \cdot u(n)$$

siendo Q y R conocidas como “matrices de ponderación”, que determinan la importancia relativa entre el estado y la entrada de control. Q debe ser una matriz cuadrada hermítica definida positiva, y R debe ser semidefinida positiva.

En el caso del control de velocidad angular diseñado:

$$\chi(n) = \begin{bmatrix} d_e(n) \\ \theta_e(n) \end{bmatrix} \text{ y } \quad u(n) = \Omega(n)$$

y además Q será una matriz cuadrada hermítica definida positiva de orden 2, y R será un escalar. Por tanto, la función de coste se puede expresar también:

$$\wp = \sum \begin{bmatrix} d_e(n) & \theta_e(n) \end{bmatrix} \cdot \begin{bmatrix} q_{11} & q_{12} \\ q_{12} & q_{22} \end{bmatrix} \cdot \begin{bmatrix} d_e(n) \\ \theta_e(n) \end{bmatrix} + \Omega(n) \cdot R \cdot \Omega(n) =$$
$$\sum q_{11} \cdot d_e^2(n) + q_{22} \cdot \theta_e^2(n) + 2 \cdot q_{12} \cdot \theta_e(n) \cdot d_e(n) + R \cdot \Omega^2(n)$$

Generalmente, suele hacerse $q_{12} = 0$, y de esta forma :

$q_{11} \Rightarrow$ determina la importancia relativa del error lateral.

$q_{22} \Rightarrow$ determina la importancia relativa del error de orientación.

R $\Rightarrow$ determina la importancia relativa de la entrada de control Ω.

Se habla de importancia relativa puesto que lo determinante no es el valor absoluto de estos parámetros, sino la relación que guardan entre ellos. Una orientación para el ajuste de Q y R consiste en hacer que cada uno de los elementos de estas matrices sea la inversa del cuadrado de la desviación máxima deseada de la señal asociada a dicho elemento.

Se puede demostrar que, una vez determinados los valores de Q y de R, el valor de la matriz de realimentación K puede obtenerse de la resolución de la ecuación de Ricatti en régimen estacionario (5):

$$P = Q + G^* \cdot P \cdot \left(I + H \cdot R^{-1} \cdot H^* \cdot P\right)^{-1} \cdot G$$
$$K = \left(R + H^* \cdot P \cdot H\right)^{-1} \cdot H^* \cdot P \cdot G$$

5.2.7 Implementación

Para implementar el sistema de control se ha desarrollado este algoritmo en lenguaje C dentro del programa cliente. El código de la aplicación sigue una estructura modular, de tal forma que el control óptimo se programa de forma independiente, hace uso de funciones auxiliares para realizar los cálculos matriciales necesarios, y repite todo el proceso cada vez que se requiere un nuevo cálculo de la velocidad angular (Ω) a aplicar.

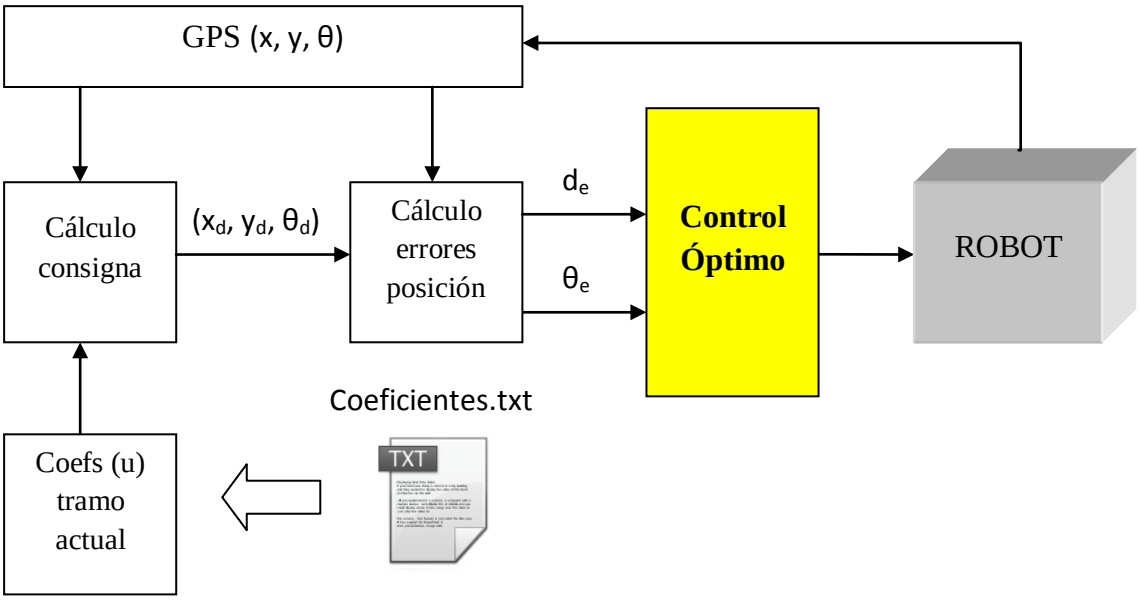


Figura 5.5. Esquema de funcionamiento del Control Óptimo.

El controlador utiliza como parámetros de entrada los errores lateral y de orientación que presenta el robot con respecto al punto de consigna en cada instante.

El control borroso implementado se encarga de formar las matrices A,B,Q y R del control óptimo vistas en el capítulo anterior. Con ayuda de funciones auxiliares que realizan las operaciones elementales con matrices, calcula la matriz K del control óptimo y obtiene la velocidad angular que se enviará al robot.

5.3 Control Borroso

5.3.1 Introducción

Continuando con la misma idea de disponer un controlador que permita al robot desplazarse siguiendo la trayectoria y minimizando el error en posición, se introduce un control borroso que a diferencia del control óptimo que actuaba sobre la velocidad angular, éste lo hará sobre la velocidad lineal para obtener el parámetro de velocidad lineal que el controlador óptimo explicado anteriormente tomará como input.

Para conseguir un comportamiento apropiado del robot al desplazarse a lo largo de la trayectoria determinada, es obvio que debe realizarse a su vez un ajuste de la velocidad lineal en función de determinadas características de la trayectoria y del entorno. Así, por ejemplo, es deseable que la velocidad vaya disminuyendo gradualmente al acercarse al punto objetivo, para evitar una frenada brusca. Del mismo modo, es conveniente reducir la velocidad en los tramos de trayectoria que presenten mayor curvatura, lo cual permitirá ajustarse mucho mejor a la misma, y por último también parece lógico ir más despacio cuando estamos frente a un obstáculo que podría incluso moverse e impactar con nosotros como pudiera ser otro robot o cualquier otro obstáculo no esperado.

En definitiva, la salida del controlador de velocidad lineal es la velocidad de avance del robot, siendo las entradas al mismo determinadas características de la trayectoria y del entorno como las citadas anteriormente. Ahora bien, ¿qué ecuaciones ligan todos estos parámetros? En este caso no se dispone de un modelo de comportamiento como en el diseño del controlador de velocidad angular, pero sí de una idea de cómo ha de actuar el sistema. Así, por ejemplo, si el radio de curvatura es grande la velocidad puede ser grande también. Si la distancia al destino es pequeña, deberá disminuir la velocidad. Este tipo de razonamiento puede ser abordado fácilmente mediante técnicas de control borroso, siendo ésta la opción elegida para el diseño del controlador.

Como sucede para cualquier sistema, en el diseño de un sistema borroso hay que distinguir dos partes: una primera en la que se determinan los parámetros que intervienen en el proceso, y una segunda que consiste en la ejecución del proceso que obtiene las salidas a partir de las entradas mediante la aplicación de una serie de algoritmos que dependen de los parámetros anteriores. A la primera etapa se le conoce como “Base del conocimiento”, y a la segunda “Estructura de procesamiento”. En el Anexo de este trabajo se introducen brevemente los conceptos teóricos sobre Control Borroso que se aplicarán a lo largo de este apartado.

A continuación se explica la aplicación del Control Borroso para el cálculo de la velocidad lineal del robot, de forma similar a como se hacía en (1), abordando el problema en tres etapas:

1. Organización de la Base del Conocimiento. En esta primera etapa deben definirse todos los parámetros de interés del controlador borroso, siendo éstos :

- Variables de entrada y salida que intervienen en el controlador.
- Número de conjuntos borrosos de cada variable de entrada o salida, y para cada uno de estos conjuntos debe definirse su etiqueta lingüística y su función de pertenencia.
- Reglas que regirán el comportamiento del sistema.

En los tres puntos indicados es necesario tomar una serie de decisiones que en general estarán condicionadas por las limitaciones de la plataforma hardware en la que se ejecutará el algoritmo.

2. Organización de la Estructura de Procesamiento. En este apartado deben tomarse a su vez ciertas decisiones sobre la ejecución del algoritmo de control borroso, tales como los tipos de operadores usados para implementar la intersección o unión de variables borrosas, métodos de implicación o de agregación de las salidas, etc. En este caso también se tenderá a adoptar la solución más simple que ofrezca resultados satisfactorios con el fin de reducir al máximo el tiempo de ejecución final del algoritmo de control.

3. Simulación de resultados. Estos han sido ensayados directamente en el simulador real del robot, pudiéndose ver el análisis de los resultados obtenidos en el próximo capítulo.

5.3.2 Organización de la Base del Conocimiento

La organización de la base del conocimiento se va a realizar en los siguientes pasos que se describen a continuación:

a) Elección y Cálculo de las variables de entrada al controlador

Se han escogido como variables de entrada o factores de decisión del controlador borroso un total de tres parámetros, el radio de curvatura de la trayectoria en la posición en la que se encuentre el robot, su distancia al destino y la distancia al obstáculo más cercano. Este último parámetro es calculado fácilmente a partir de la lectura de los sensores del robot. La salida final del controlador es la velocidad lineal.

Por lo tanto, el cálculo de los factores de decisión en cada instante de tiempo se realiza en función de las características de la trayectoria y el entorno, según el siguiente diagrama de bloques:

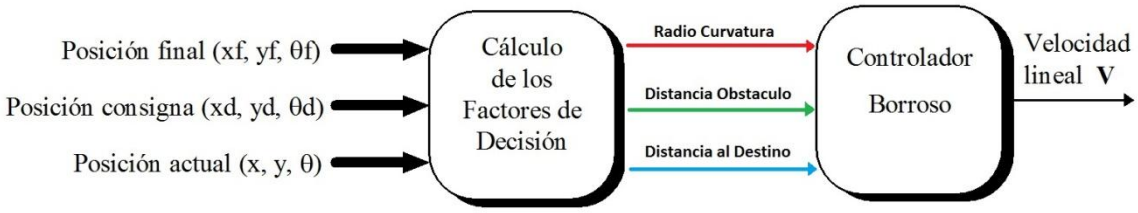


Figura 5.6. Estructura del controlador borroso.

La posición final (destino) es un valor conocido introducido por el usuario. El controlador calculará la posición de consigna (punto más cercano de la trayectoria) que corresponde al robot en cada instante, mientras que la posición actual se obtiene por gps o bien a partir de la información de los encoders.

Los factores de decisión distancia al destino y distancia mínima al obstáculo (variables de entrada al controlador) se calculan a partir de la consigna (xd, yd, θd).

En el caso de la distancia al destino (dis), conocida la posición actual del móvil (x,y) y la posición de destino (xf, yf), puede calcularse mediante la fórmula:

$$dis = \sqrt{(x_f - x)^2 + (y_f - y)^2}$$

En lo que al radio de curvatura (rc) se refiere, puede determinarse según la siguiente expresión:

$$rc = \left. \frac{\sqrt{\left(1 + f'^2\right)^3}}{f''} \right|_{(xd,yd)}$$

siendo f la ecuación de la trayectoria y f' y f'' sus derivadas primera y segunda (al ser la trayectoria una spline, su ecuación es un polinomio de tercer grado, cuyas derivadas son sencillas de calcular).

Por último, el valor de la distancia mínima al obstáculo (dobs) se obtiene al calcular el valor mínimo de las lecturas de los sensores (dsens):

$$d_{obs} = \min [d_{sens}]$$

b) Elección y Caracterización de los Conjuntos Borrosos

Para cada una de las cuatro variables de entrada/salida debe determinarse:

- El rango de valores que constituye su universo de discusión.
- El número de conjuntos borrosos, y la etiqueta lingüística de cada uno de ellos.
- La forma de las funciones de pertenencia de los conjuntos borrosos, así como sus soportes y centros.

Es obvio que las variables pueden tomar valores comprendidos entre 0 e infinito. Sin embargo, se acotará el valor máximo del universo de discusión a un número finito, siendo necesario añadir un saturador a la salida del bloque de cálculo de los factores de decisión. Por defecto, el universo de discusión de la variable rc (radio de curvatura) se ha establecido en el rango (0-9) m, de la variable dis (distancia al destino) en (0-1) m, para la distancia al obstáculo en (0-5) m, y en el caso de la velocidad lineal del robot, se ha considerado apropiado un valor máximo de 0.4 m/s al tratarse de un robot “Pioneer”, por lo que el universo de discusión comprende el rango (0-0.4) m/s.

La siguiente decisión a tomar es el número de conjuntos borrosos para cada variable, y la forma de sus funciones de pertenencia. Dada la eficacia que proporcionaban los conjuntos de forma triangular como se demostró en (1), se ha actuado de forma similar, utilizando para cada variable los tres conjuntos borrosos que vemos a continuación:

RADIO DE CURVATURA: 0 – 2.5 m.

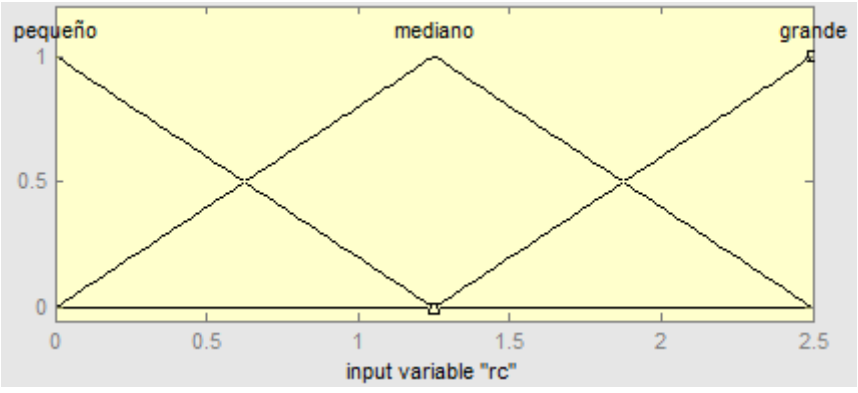


Figura 5.7. Funciones de pertenencia del radio de curvatura (rc).

DISTANCIA AL DESTINO: 0 – 1 m.

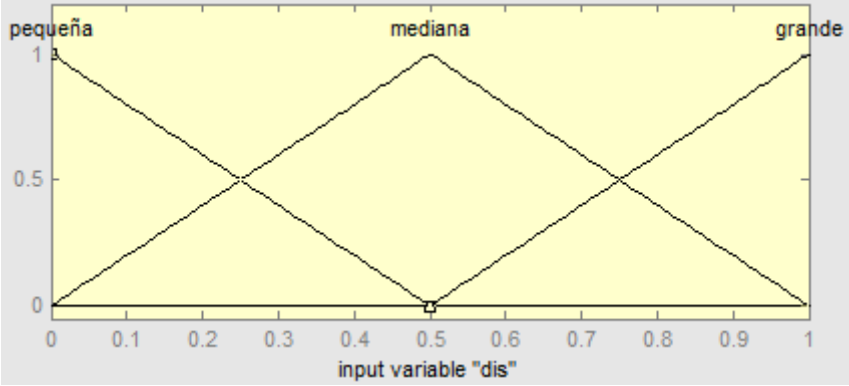


Figura 5.8. Funciones de pertenencia de la distancia al destino (dis).

DISTANCIA AL OBSTÁCULO: 0 – 5 m.

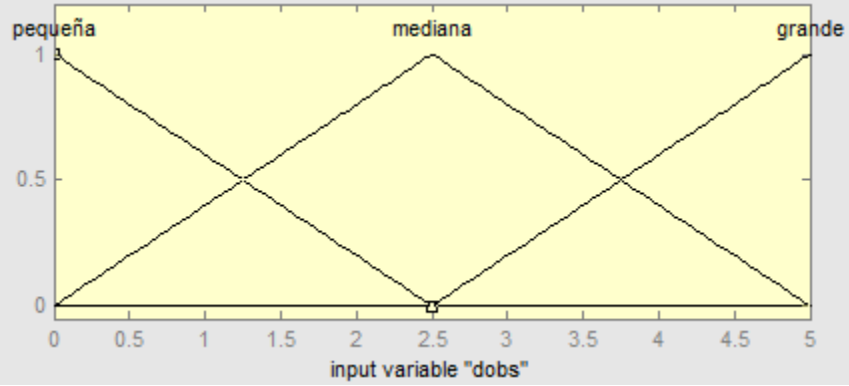


Figura 5.9. Funciones de pertenencia de la distancia al obstáculo (dobs).

VELOCIDAD LINEAL: 0 – 0.9 m/s

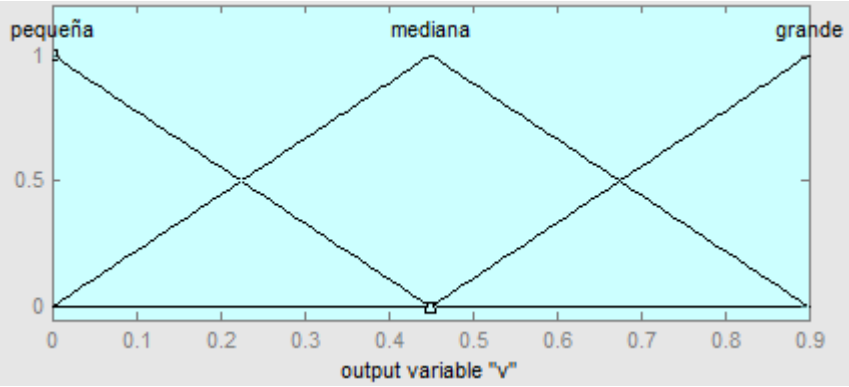


Figura 5.10. Funciones de pertenencia de la velocidad lineal (v).

c) Elección de las Reglas del Sistema Borroso

Al existir 3 conjuntos borrosos en cada una de las 3 variables a considerar, se obtienen un total de 3^3 = 27 reglas diferentes de comportamiento. Tras varias simulaciones, se ha elegido la siguiente base de reglas al comprobar que proporcionan una respuesta adecuada del sistema:

Radio	Distancia Destino	Distancia Obstáculo	Velocidad Lineal
Pequeño	Pequeña	Pequeña	Pequeña
Pequeño	Pequeña	Mediana	Pequeña
Pequeño	Pequeña	Grande	Pequeña
Pequeño	Mediana	Pequeña	Pequeña
Pequeño	Mediana	Mediana	Mediana
Pequeño	Mediana	Grande	Mediana
Pequeño	Grande	Pequeña	Pequeña
Pequeño	Grande	Mediana	Mediana
Pequeño	Grande	Grande	Grande
Mediano	Pequeña	Pequeña	Pequeña
Mediano	Pequeña	Mediana	Pequeña
Mediano	Pequeña	Grande	Pequeña
Mediano	Mediana	Pequeña	Pequeña
Mediano	Mediana	Mediana	Mediana
Mediano	Mediana	Grande	Mediana
Mediano	Grande	Pequeña	Pequeña
Mediano	Grande	Mediana	Mediana
Mediano	Grande	Grande	Grande
Grande	Pequeña	Pequeña	Grande
Grande	Pequeña	Mediana	Pequeña
Grande	Pequeña	Grande	Pequeña
Grande	Mediana	Pequeña	Pequeña
Grande	Mediana	Mediana	Pequeña
Grande	Mediana	Grande	Mediana
Grande	Grande	Pequeña	Mediana
Grande	Grande	Mediana	Pequeña
Grande	Grande	Grande	Mediana

Tabla 5.1. Reglas de comportamiento del controlador borroso.

El conjunto de reglas puede simplificarse, sin embargo se ha preferido mantener este formato para que el usuario del programa pueda modificarlas libremente facilitando su comprensión.

5.3.3 Organización de la Estructura de Procesamiento

Para implementar el control borroso se han seguido las instrucciones propuestas en (1), aplicándose los pasos del siguiente modo:

1. Borrosificación de las entradas. Conocidos los valores instantáneos del radio de curvatura y de la distancia al destino, se determina la pertenencia de estas variables a cada uno de los tres conjuntos borrosos asociados a las mismas ("grande", "mediana" y "pequeña"). Para ello, basta con evaluar las funciones de pertenencia de cada conjunto para el valor de la entrada actual.

2. Cálculo de los antecedentes de las reglas mediante la aplicación del operador AND escogido. Por su simplicidad, se ha escogido como operación AND la función "mínimo". Para cada regla, se obtiene como antecedente un número real comprendido entre 0 y 1.

3. Cálculo de las salidas de las reglas. Para reducir al máximo el tiempo de ejecución del algoritmo, se ha escogido el método Sugeno, que requiere menos cálculos. De este modo, la salida de cada regla (ri) es el producto del valor del antecedente de dicha regla (wi) por una función del conjunto borroso del consecuente. Dicha función se ha escogido como el valor constante del centro de la función de pertenencia de dicho conjunto borroso (ci).

$$r_i = w_i \cdot c_i$$

4. Agregación de todas las salidas. Por tratarse del método Sugeno, la agregación de todas las reglas, y por tanto la salida final del control borroso (velocidad lineal) es un número exacto, no siendo necesario el proceso de desborrosificación.

$$v = \frac{\sum_{i=1}^{9} r_i}{\sum_{i=1}^{9} w_i}$$

6. Implementación en entorno Player/Stage

6.1 Introducción

El entorno Player/Stage (7) dispone de un servidor (Player) al que se conectan los clientes por medio de una conexión TCP, permitiendo interactuar con robots reales o bien realizar simulaciones. Los programas cliente que se conectan a Player no están restringidos al uso de un único lenguaje de programación, puede emplearse cualquiera para el que se hayan desarrollado drivers que permitan la conexión a través de sockets TCP. En este caso se ha elegido el lenguaje C empleando los drivers que el propio Player/Stage ofrece para dar soporte a las comunicaciones a bajo nivel.

Cada programa cliente que se conecta al servidor Player controla un único robot. El proceso de diseño del controlador se ha llevado a cabo en dos etapas; en primer lugar el desarrollo se enfoca al control de un robot y después se incorporan nuevas funcionalidades que permiten la navegación de N clientes en un mismo entorno. En la Figura 6.1 se muestra un esquema de las tareas que componen el sistema:

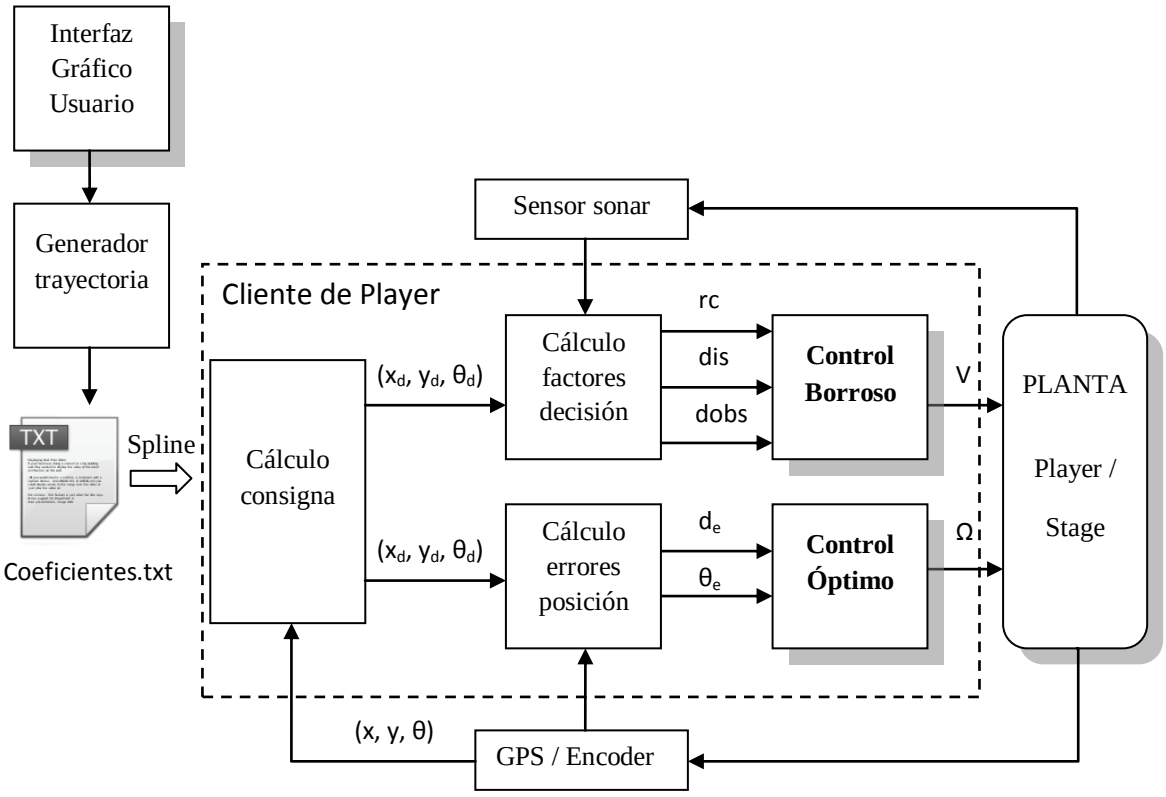


Figura 6.1. Esquema detallado del controlador.

6.2 Implementación

En primer lugar hay que definir los interfaces que se van a utilizar para conectarse al servidor y establecer comunicación con los dispositivos necesarios. La Tabla 6.1 resume las funciones empleadas que intervienen en la comunicación con el servidor Player:

INTERFAZ	FUNCIONES
Position2d	playerc_position2d_create (playerc_client_t *client, int index) playerc_position2d_destroy (playerc_position2d_t *device) playerc_position2d_subscribe (playerc_position2d_t *device, int access) playerc_position2d_unsubscribe (playerc_position2d_t *device) playerc_position2d_set_cmd_vel (playerc_position2d_t *device, double vx, double vy, double va, int state)

Sonar	playerc sonar t *playerc sonar create(playerc client t *client, int index);
	playerc sonar destroy(playerc sonar t *device);
	playerc sonar subscribe(playerc sonar t *device, int access);
	playerc sonar unsubscribe(playerc sonar t *device);
	playerc sonar get geom(playerc sonar t *device);

Graphics2d	playerc_graphics2d_create (playerc_client_t *client, int index) playerc_graphics2d_destroy (playerc_graphics2d_t *device) playerc_graphics2d_subscribe (playerc_graphics2d_t *device, int access) playerc_graphics2d_unsubscribe (playerc_graphics2d_t *device) playerc_graphics2d_setcolor (playerc_graphics2d_t *device, player_color_t col) playerc_graphics2d_draw_points (playerc_graphics2d_t *device, player_point_2d_t pts[], int count) playerc_graphics2d_draw_polyline (playerc_graphics2d_t *device, player_point_2d_t pts[], int count)
Map	playerc_map_create (playerc_client_t *client, int index) playerc_map_destroy (playerc_map_t *device) playerc_map_subscribe (playerc_map_t *device, int access) playerc_map_unsubscribe (playerc_map_t *device) playerc_map_get_map (playerc_map_t *device) #Define PLAYERC_MAP_INDEX(dev, i, j) ((dev->width) * (j) + (i))

Tabla 6.1. Funciones de comunicación empleadas.

Por tanto, el programa cliente comienza con la creación y conexión de los dispositivos que se van a utilizar (position2d, graphics2d, sonar, map y el propio cliente). Una vez se encuentra conectado al servidor con los dispositivos operativos, accede al fichero que contiene el mapa del entorno a través del interfaz “map”. En este punto se almacena en memoria el mapa en forma de matriz de celdas y cada una

de ellas se rellena con un índice en función del nivel de ocupación. Esta matriz permitirá clasificar los obstáculos, diferenciando los estáticos (paredes, muros…) de cualquier otro elemento detectado durante la simulación, como otros robots.

La siguiente operación será inicializar los parámetros de control a los valores deseados y recuperar los coeficientes de las Splines en función de "u" que residen en el fichero creado por el generador de trayectorias. Estos coeficientes caracterizan cada tramo de la spline y, junto a la posición actual del robot que vendrá dada por GPS, se obtienen los parámetros que permiten calcular la velocidad angular y lineal (x_d, y_d, θ_d, d_e, θ_e) a través de los controles óptimo y borroso respectivamente. Como última acción antes de entrar en el bucle de control principal, se dibuja la trayectoria sobre el mapa para visualizar el recorrido que seguirá el robot.

En este momento se dispone de los elementos necesarios para que el robot comience a desplazarse de forma controlada siguiendo la trayectoria. El programa entra en un bucle donde realiza periódicamente una serie de acciones que modifican el comportamiento del robot de acuerdo a los objetivos previstos. El servidor Player proporciona los datos de ubicación del robot a través del interfaz "position2d", y a partir de ellos se calculan los errores respecto a la posición y orientación de consigna, así como el radio de curvatura que presenta la trayectoria en ese punto. Con estos parámetros actualizados, el Control Borroso está en disposición de proporcionar la velocidad lineal (V), y después el Óptimo calculará la angular (Ω).

Todo este proceso se repite a lo largo del recorrido, por tanto es necesario conocer en qué tramo se encuentra el robot para cargar los coeficientes que correspondan a dicho instante. Se ha determinado que un valor de "u" por encima de 0,999 será indicador de cambio de tramo (recordemos que el rango del parámetro "u" va de 0 a 1, y vuelve a tomar el valor "0" cada vez que se inicia un nuevo tramo).

Una vez se dispone de la velocidad lineal y angular, éstas son enviadas al robot para que aplicándolas a través de sus actuadores, sea capaz de seguir la trayectoria calculada. Por último, cuando el robot ha llegado al final de la trayectoria se procede a desconectar los dispositivos empleados a la vez que se informa al usuario.

Durante la ejecución del programa cliente es necesario consultar el mapa para comprobar si los puntos de impacto detectados por los sensores corresponden a una pared del entorno, o por el contrario se trata de otro tipo de obstáculo. Para la correcta interpretación del mapa ha de tenerse en cuenta que el interfaz "map" establece el punto (0,0) de referencia en el extremo inferior izquierdo de la imagen, por tanto debe realizarse una pequeña conversión como se aprecia en la Figura 6.2.

Las dimensiones de la imagen vienen definidas por el número de píxeles horizonatales (X) y verticales (Y). Estos valores se recogen en sendas variables dentro del programa cliente (*timagenx, timageny*) lo que aporta la flexibilidad adecuada para admitir mapas de diferentes tamaños.

Coordx(map) = Coord(x) + X;

Coordy(map) = Coord(y) + Y;

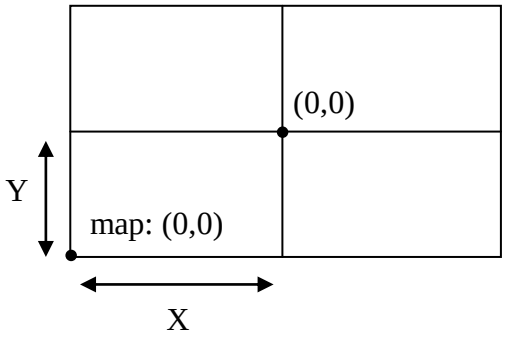
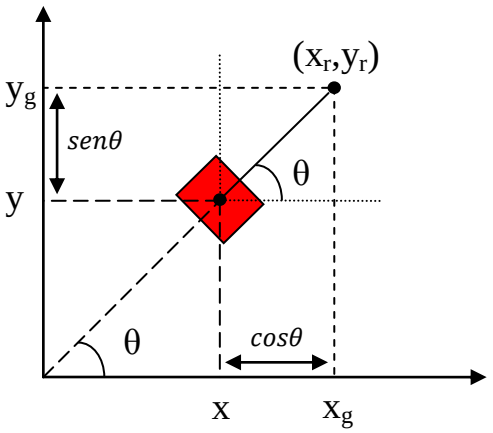


Figura 6.2. Conversión de la imagen.

Por otro lado, cuando se obtiene la distancia de un sensor láser, es necesario conocer la coordenada del sistema global donde se ha detectado el impacto. Para ello se recurre a la función *playerc_sonar_get_geom (sonar),* que proporciona la ubicación de los sensores respecto al centro de coordenadas local del robot. Con este dato y conociendo la posición real del robot a través de odometría, es posible determinar el punto de impacto aplicando un cambio de base de la siguiente forma:



$$\begin{pmatrix} x_g \\ y_g \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{pmatrix} \cdot \begin{pmatrix} x_r \\ y_r \end{pmatrix}$$

Figura 6.3. Matriz de cambio de base.

Donde se tiene que:

(x_g,y_g) son las coordenadas del punto de impacto en el sistema global.

(x,y) son las coordenadas de la posición del robot en el sistema global.

(x_r,y_r) son las coordenadas del punto de impacto respecto al sistema local del robot.

θ es la orientación del robot en el sistema global.

Para realizar el cálculo, en primer lugar se obtienen las coordenadas del impacto respecto al sistema local del robot, a partir de la ubicación y orientación del sensor:

*ximp=sonar->poses[sensor].px+sonar->scan[sensor]*cos(sonar->poses[sensor].pyaw);*

*yimp=sonar->poses[sensor].py+sonar->scan[sensor]*sin(sonar->poses[sensor].pyaw);*

Y después se aplica la conversión mediante las operaciones con matrices para obtener los puntos de impacto respecto al sistema global:

*puntoimpactox=(Posicion.x)+ximp*cos(Posicion.o)-(yimp*sin(Posicion.o));*

*puntoimpactoy=(Posicion.y)+ximp*sin(Posicion.o)+(yimp*cos(Posicion.o));*

En cuanto al tratamiento de los coeficientes de la spline, se ha decidido almacenarlos siguiendo una estructura matricial como muestra la Figura 6.4:

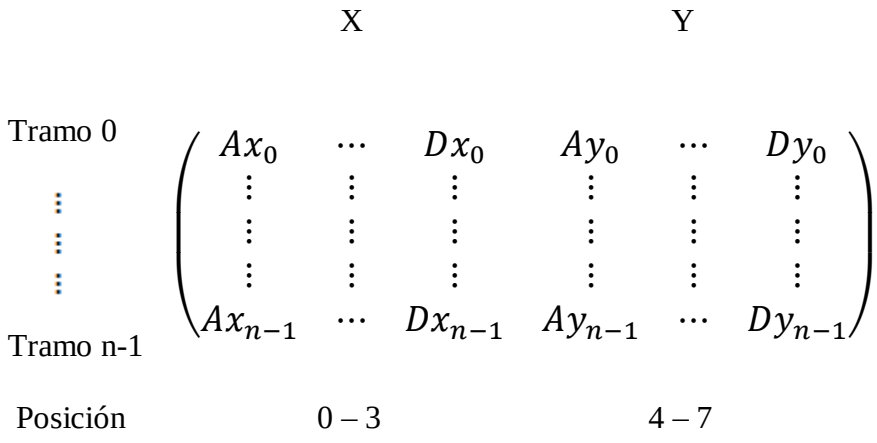


Figura 6.4. Matriz para almacenar los coeficientes de la spline.

A continuación se muestra el esquema de funcionamiento que sigue el programa cliente escrito en lenguaje C, comenzando por la inicialización de dispositivos y recopilación de información:

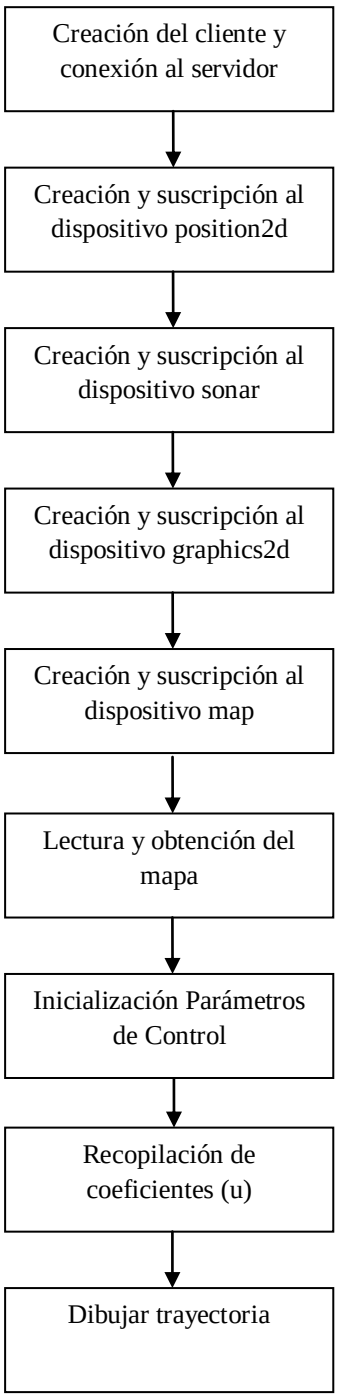


Figura 6.5. Inicialización del programa cliente.

Bucle de control dentro del programa principal que se repetirá hasta alcanzar el final de la trayectoria:

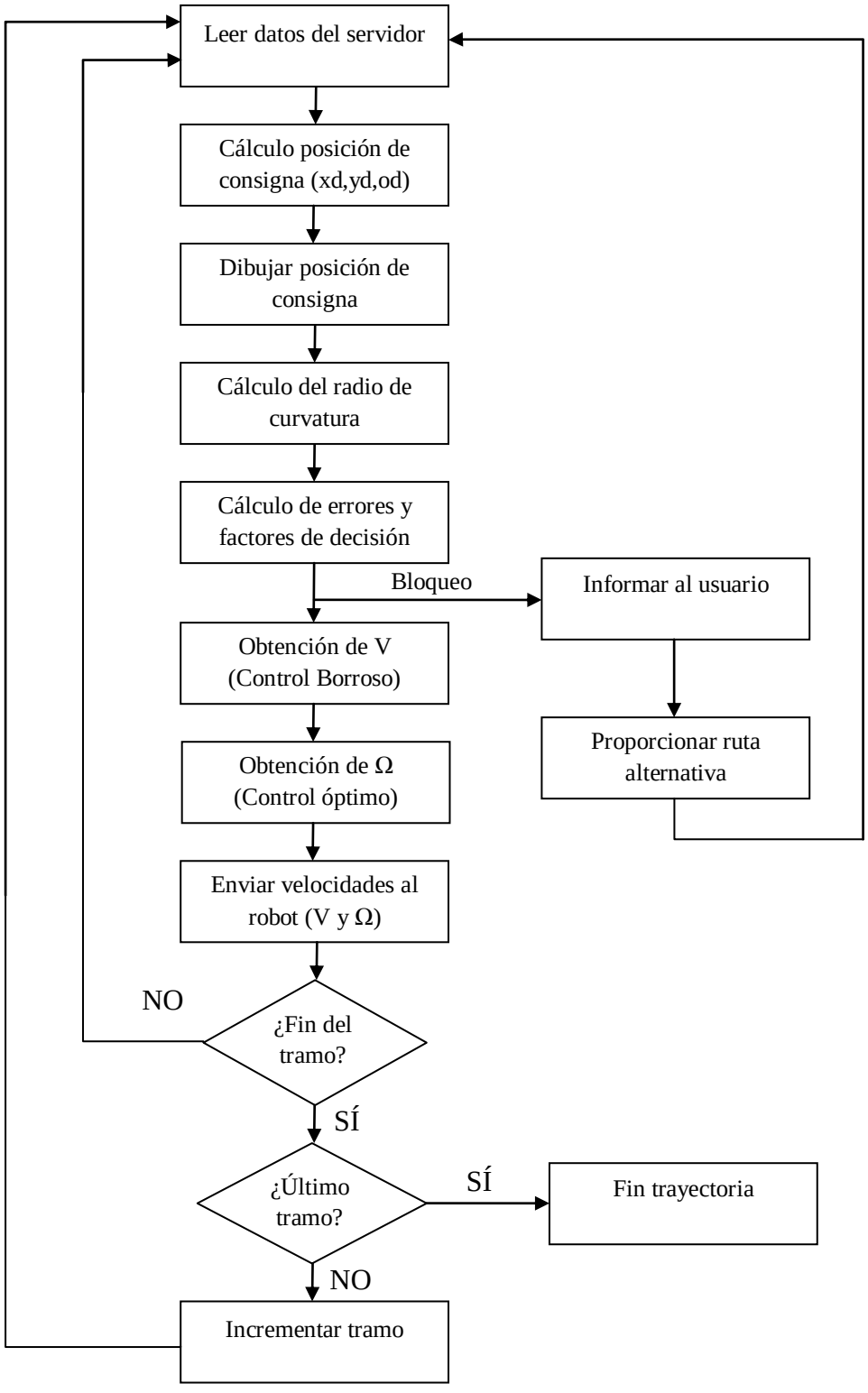


Figura 6.6. Bucle principal de control.

7. Aplicación a un sistema multirrobot.

7.1 Ampliación a N robots

Dado que el entorno Player/Stage permite la conexión simultánea de múltiples clientes, se pretende ampliar el sistema para que varios clientes realicen el seguimiento de su propia trayectoria de forma independiente. Al abordar un entorno con múltiples robots surgen nuevos problemas que se han solucionado implementando un control de velocidad basado en prioridades. De este modo cuando dos o más robots rebasen el umbral de visibilidad que se ha establecido, sabrán cómo ajustar su velocidad para facilitar el paso al de mayor prioridad.

Para llevar a cabo el desarrollo de la nueva funcionalidad, se modifica el sistema responsable del ajuste de la velocidad lineal; el control borroso. Las 27 reglas de comportamiento que resultan al combinar los 3 conjuntos borrosos (pequeño, mediano y grande) presentes en cada una de las 3 variables a considerar (radio de curvatura, distancia al destino y distancia al obstáculo) determinan cómo reacciona el robot ante diferentes escenarios.

Por ejemplo, al introducir un patrón que establezca una velocidad lineal de salida pequeña para un obstáculo lejano, se conseguirá que este robot frene su avance de forma más notable ante la presencia de otros robots. Por otro lado está definido el rango de valores que establece el grado de pertenencia a cada una de las funciones. Por tanto, parece evidente que será necesario definir una nueva configuración para el conjunto borroso que evalúa la distancia al obstáculo.

Dado que el número de robots es un parámetro variable, se ha buscado una regla que permita adjudicar prioridades de mayor a menor comenzando por el primer robot. Así, el robot que comience en primer lugar tendrá un conjunto borroso para la variable distancia al obstáculo similar al que presenta el controlador original. A partir de aquí, los nuevos clientes que aparezcan verán modificado su conjunto borroso de modo que la función de pertenencia “distancia al obstáculo pequeña” irá incrementando su valor máximo y abarcando cada vez más espacio dentro del conjunto, mientras las funciones de pertenencia “mediana” y “grande” serán desplazadas progresivamente a la derecha, siendo cada vez más pequeñas y por tanto tendrán menos peso a medida que la prioridad del robot disminuya.

El conjunto borroso de partida para la variable “Distancia al obstáculo” se muestra en la Figura 7.1:

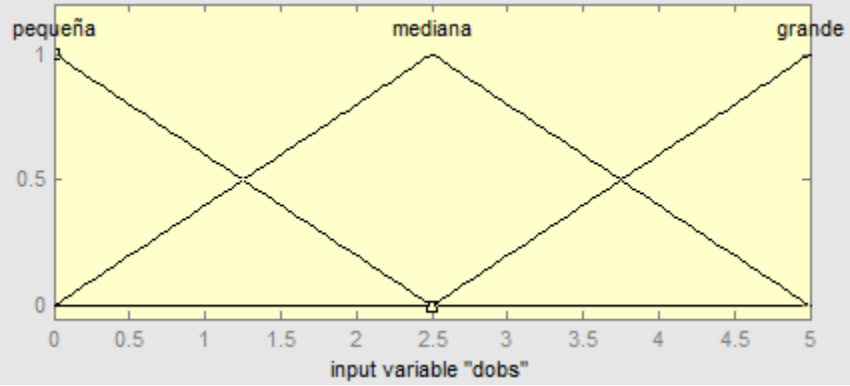


Figura 7.1. Conjunto borroso inicial para la distancia al obstáculo.

Como regla básica para implementar la nueva funcionalidad se establece la siguiente fórmula:

$$\frac{dobs\_max}{2} * (1 + N * Inc)$$

Siendo N el número de robot ligado a la prioridad (mayor o igual a 0), e Inc el incremento o desplazamiento deseado que se aplique a la función de pertenencia (en % expresado de 0 a 1).

Por ejemplo, para N=1 (un robot por delante con mayor prioridad) y un incremento del 10% adicional respecto al robot precedente, se tienen los siguientes conjuntos borrosos:

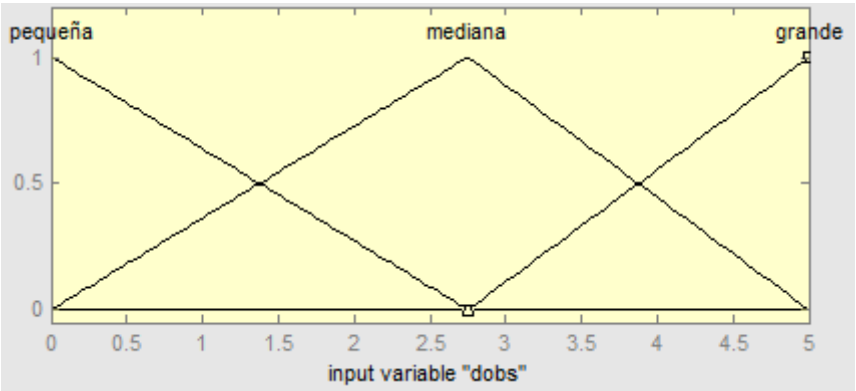


Figura 7.2. Conjunto borroso para N=1, Inc= 0.1 (10%).

El código asociado a esta configuración será:

```
dobs_max=5;
```

Primer conjunto borroso de la variable: dobs

```
borroso.params[3][0][0]=0.0;
borroso.params[3][0][1]=0.0;
borroso.params[3][0][2]=(dobs_max/2)*(1+(1*0.1));
```

Segundo conjunto borroso de la variable: dobs

```
borroso.params[3][1][0]=0.0;
borroso.params[3][1][1]=(dobs_max/2)*(1+(1*0.1));
borroso.params[3][1][2]=dobs_max;
```

Tercer conjunto borroso de la variable: dobs

```
borroso.params[3][2][0]=(dobs_max/2)*(1+(1*0.1));
borroso.params[3][2][1]=dobs_max;
borroso.params[3][2][2]=dobs_max;
```

Después de realizar un ajuste experimental a través de las pruebas correspondientes, se determinó que el controlador presentaba un buen comportamiento para la gestión de múltiples robots cuando las funciones de pertenencia se desplazaban al menos un 5% adicional respecto al robot precedente.

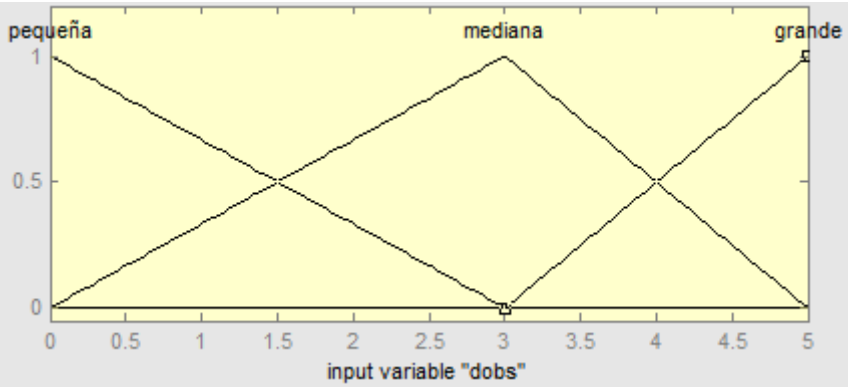


Figura 7.3. Conjunto borroso para N=6 robots, Inc=0,05 (5%).

7.2 Script principal e Interfaz gráfica.

El programa cliente que se ha desarrollado no puede ejecutarse por sí solo; al iniciar solicita la conexión al servidor Player, por lo que es necesario que éste haya sido arrancado primero, se mantenga a la escucha, y admita la conexión en el puerto que ha establecido a tal efecto.

Por otra parte, el servidor Player necesita un fichero inicial de configuración que normalmente tiene extensión ".cfg", donde se definen los drivers e interfaces asociados que se van a emplear. Con estos datos se establece una relación donde se asigna a cada dispositivo físico (o que intervenga en la simulación) el dispositivo lógico de Player que le corresponda.

En el caso de Stage el fichero de configuración ".world" describe el conjunto de robots, sensores y cualquier otro elemento que participe en la simulación. Las propiedades de cada dispositivo, el puerto para conectarse al servidor Player, la posición y orientación iniciales dentro del mapa, y cualquier otra característica será definida dentro del fichero.

La elaboración de estos ficheros puede realizarse a mano para cada ocasión, especificando los elementos que intervienen y la configuración deseada para cada uno de ellos.

Dado que este proyecto pretende ampliar el controlador a un número variable e indefinido de robots, los ficheros de configuración no pueden ser únicos. Por tanto, se hace necesario automatizar la creación de estos ficheros de configuración atendiendo a los parámetros deseados por el usuario.

Una vez se dispone de los ficheros, el proceso de puesta en marcha consta de dos pasos ordenados; en primer lugar se arranca el servidor Player y después se lanza el programa cliente para que pueda establecer la conexión con éxito; de hacerlo al revés nos encontraríamos con un error de conexión. Así, parece necesaria una interacción con el usuario para iniciar el controlador, recibir información o proceder al cierre del programa cuando finaliza. Para dar solución a estos problemas se ha desarrollado un script que automatiza el proceso de configuración y puesta en marcha del sistema. Además, se incorpora una interfaz gráfica de usuario que facilita el manejo del controlador y lo hace más accesible de modo que cualquier persona pueda utilizarlo.

Para realizar el proceso de automatización se ha ideado dividir cada fichero de configuración en dos ficheros; el primero contiene el código base a emplear y el otro la parte variable, que se marca con etiquetas para realizar los cambios oportunos a través del editor *sed* disponible en Ubuntu 9.10 (Figura 7.4), mientras que para el

desarrollo de la interfaz gráfica se emplea el paquete *yad*. En la Figura 7.5 se aprecian las distintas capas que forman la aplicación. El código del script que se emplea para generar los ficheros de configuración, así como el contenido de los mismos puede consultarse en el apartado “Planos” de este trabajo.

Figura 7.4. Generación de ficheros de configuración.

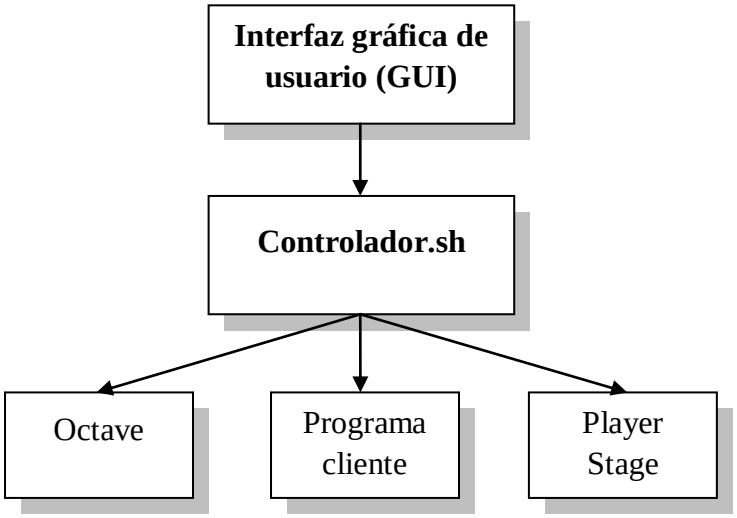


Figura 7.5. Diagrama de capas de la aplicación desarrollada.

8. Resultados

8.1 Introducción

En este apartado se exponen los resultados que se han obtenido en las pruebas realizadas para analizar la respuesta del robot ante la acción del sistema de control. Durante el proceso de desarrollo se han realizado multitud de simulaciones para verificar que el sistema se comportaba como debía, ya que tratándose de un proceso incremental, cada nueva funcionalidad añadida requería en muchas ocasiones un correcto funcionamiento de las anteriores.

En los siguientes puntos se detallan los resultados y el comportamiento que experimenta el robot al aplicar los controles óptimo y borroso con distintas configuraciones.

8.2 Análisis Control Óptimo

Para analizar los resultados obtenidos con el control óptimo implementado se han realizado diferentes simulaciones en las que el robot describía una misma trayectoria. Dado que el ajuste del controlador se consigue al encontrar un compromiso entre los tres parámetros a considerar (q11, q22, r), no teniendo importancia el valor absoluto de cada uno de ellos por separado, el único cambio en las distintas simulaciones ha sido la variación del parámetro q11, que en este caso afecta al error lateral.

Se han realizado tres simulaciones donde los parámetros del controlador tomaban los siguientes valores:

- q11=10.0, q22=1.0, r=1.0
- q11=50.0, q22=1.0, r=1.0
- q11=100.0, q22=1.0, r=1.0

En la Figura 8.1 aparece la trayectoria objetivo propuesta junto a las que describieron los robots en las simulaciones realizadas. A simple vista da la impresión de que las trayectorias son prácticamente idénticas, presentando una buena aproximación al recorrido ideal.

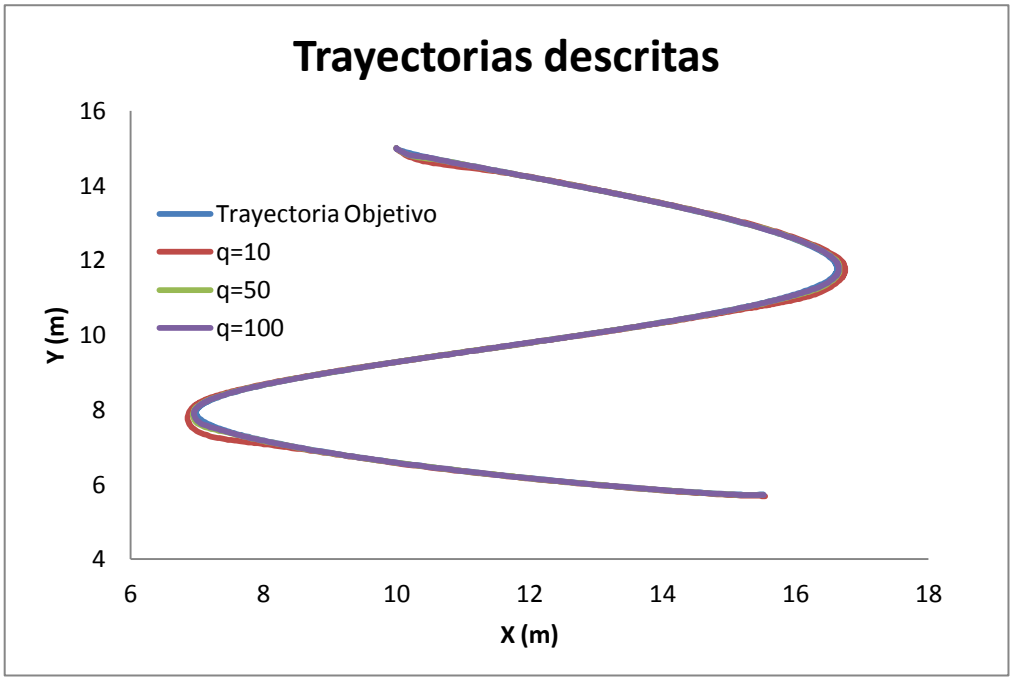


Figura 8.1. Trayectorias descritas en las simulaciones.

Sin embargo, si se analiza con detalle las zonas más críticas de la trayectoria para comparar visualmente el recorrido realizado en cada configuración (Figura 8.2 y Figura 8.3), se aprecian las primeras diferencias.

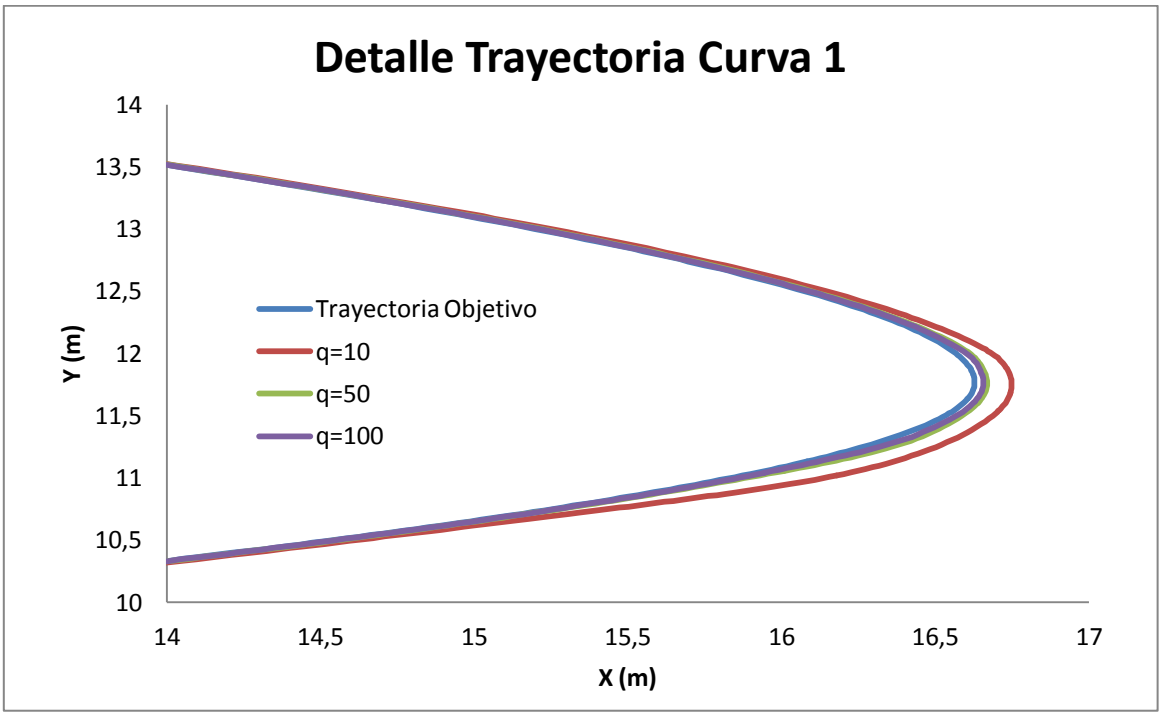


Figura 8.2. Detalle de la curva 1.

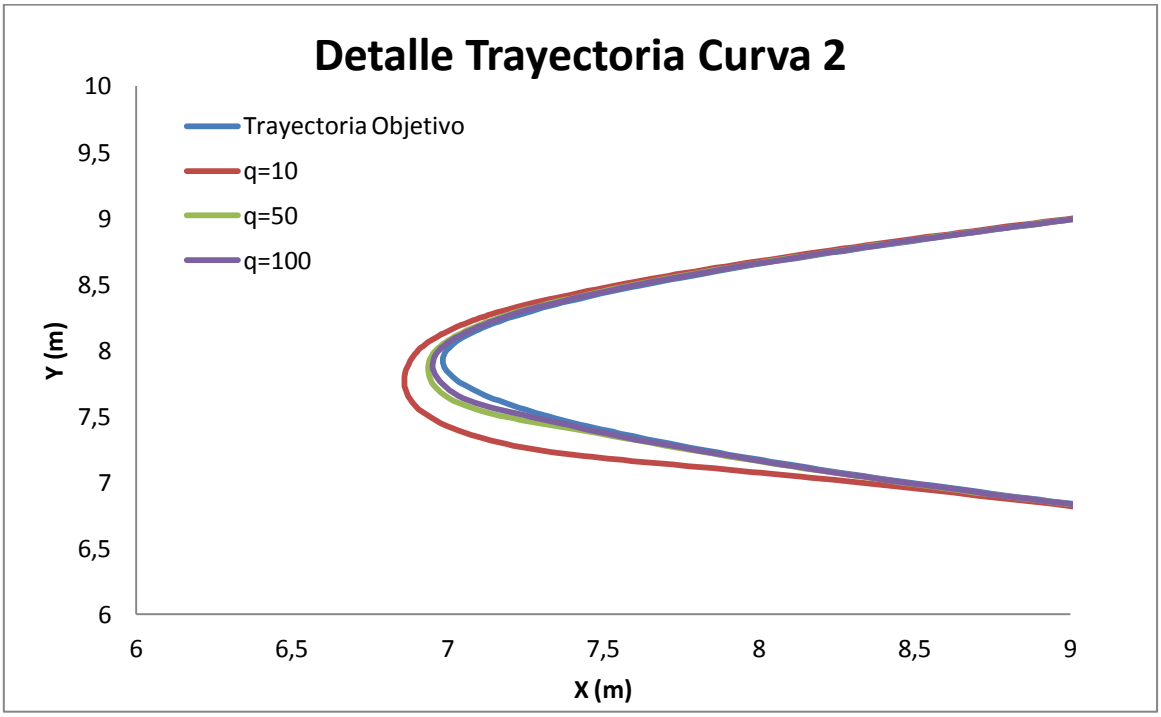


Figura 8.3. Detalle de la curva 2.

En el detalle puede apreciarse que la trayectoria descrita mejora de forma considerable al aumentar el valor del parámetro q11.

Para comprender mejor la repercusión que tiene la variación del parámetro q11 en las simulaciones se ha graficado la evolución temporal de distintas variables realacionadas con el movimiento del robot:

- Error de Orientación
- Error Lateral
- Velocidad Angular
- Velocidad Lineal

El error de orientación (Figura 8.4) disminuye ligeramente al ir aumentando el parámetro q11, existiendo una diferencia mayor en el cambio de 10 a 50, que al pasar de 50 a 100.

Con repecto al error de lateral (Figura 8.5), las conclusiones anteriores son extrapolables pero en esta ocasión la diferencia es más clara, presentando una sustancial mejora el error lateral al pasar de q11=10 a q11=50.

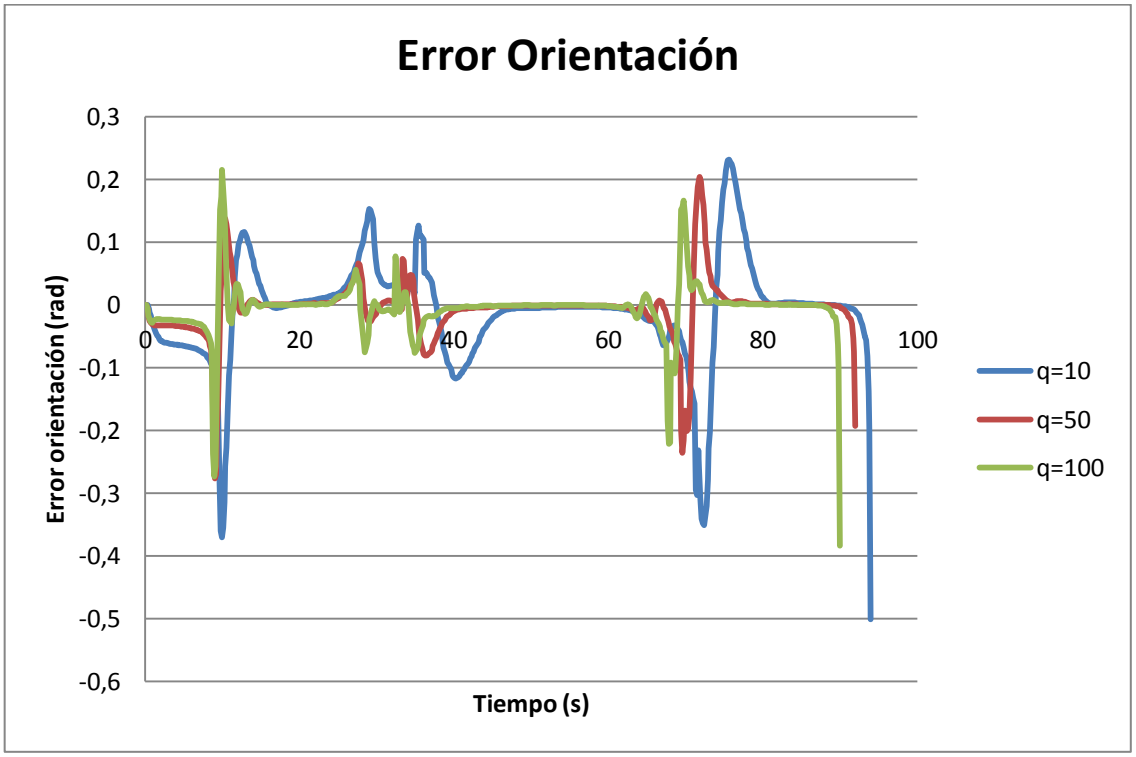


Figura 8.4. Error de orientación.

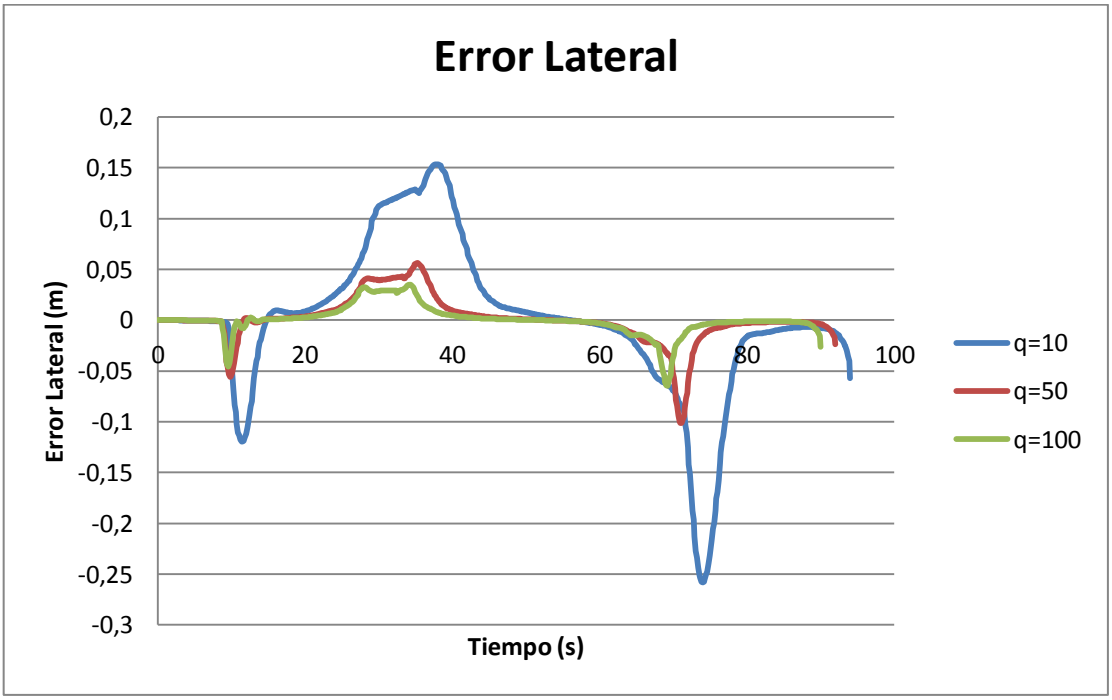


Figura 8.5. Error lateral.

En el caso de la velocidad angular, se obtiene el resultado inverso al que ocurria con los errores, de modo que al aumentar el parámetro q11, el robot reacciona más rápidamente al giro necesario consiguiendo seguir la trayectoria con más precisión. En la Figura 8.6 se observa como la velocidad angular aumenta al hacerlo q11. Aunque es apreciable a simple vista, esta variación no es muy significativa, estando las tres simulaciones en el mismo rango de valores para cada punto del trayecto.

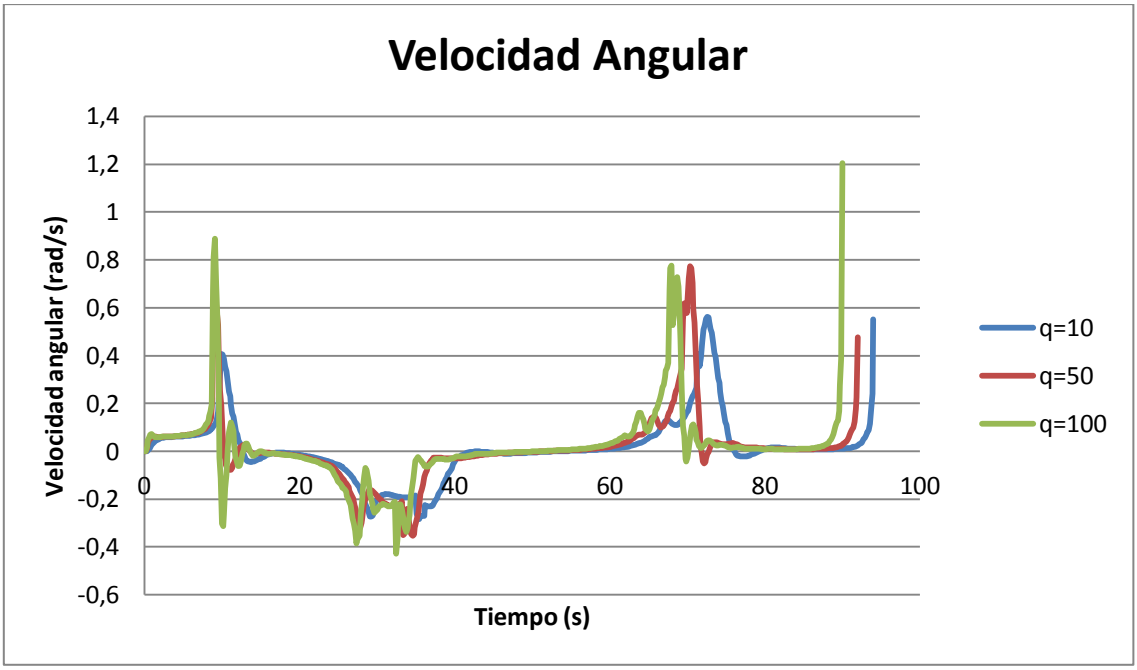


Figura 8.6. Velocidad angular.

La velocidad lineal (Figura 8.7) es un parámetro calculado por el control borroso que será analizado en el apartado siguiente. En este caso puede apreciarse con claridad la saturación de la velocidad del robot en 0,4 m/s, que adquiere en las zonas con menor curvatura.

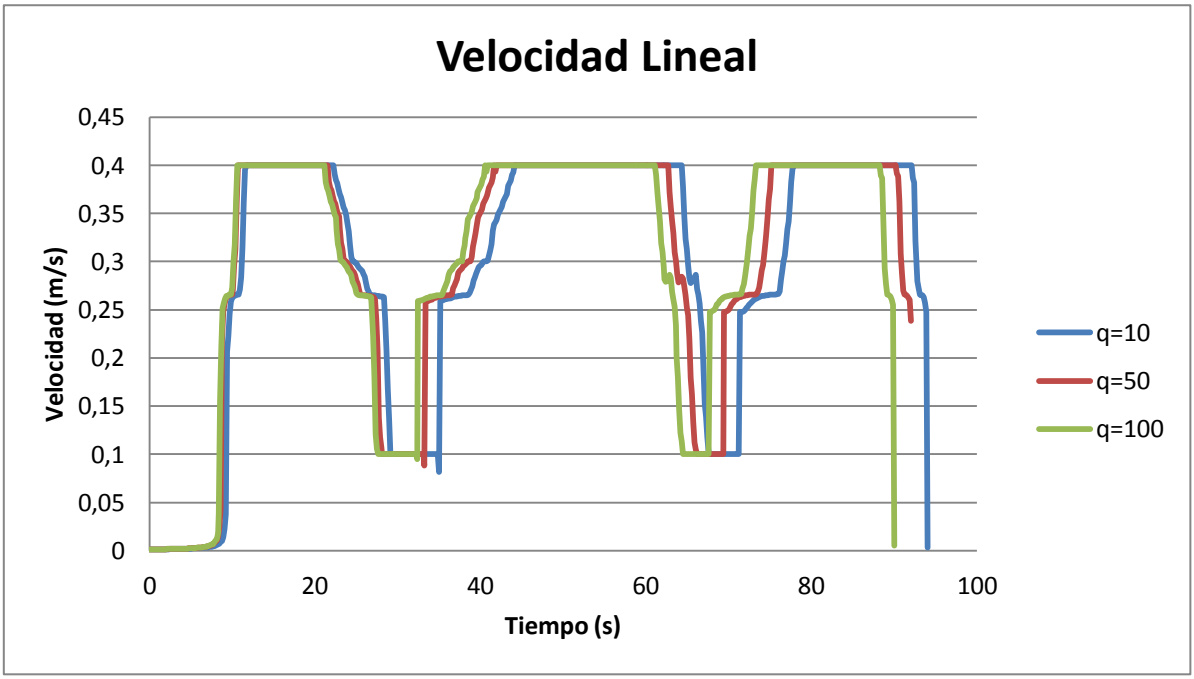


Figura 8.7. Velocidad lineal.

Para contrastar los resultados obtenidos más allá de las impresiones que se obtienen con las historias temporales de las variables, la Tabla 8.1 recoge los valores medios y máximos de cada una de ellas. Para facilitar la comprensión, se han introducido dos columnas adicionales que indican la diferencia (en porcentaje) del valor obtenido con q11=50 y q11=100 respecto a la simulación inicial con q=10.

RESUMEN RESPUESTA CONTROL ÓPTIMO					
q	**10**	**50**	**100**	**Diferencia 50 (%)**	**Diferencia 100 (%)**
Tiempo (s)	94	92	90	-2,1	-4,3
V. lineal media (m/s)	0,2938	0,3001	0,3010	2,1	2,4
V. lineal max (m/s)	0,4	0,4	0,4	0,0	0,0
V. angular media (m/s)	0,0762	0,0771	0,0817	1,1	7,2
V. angular max (rad/s)	0,5626	0,7739	1,2040	37,6	114,0
Error Lat Max (m)	0,2582	0,1013	0,0648	-60,8	-74,9
Error Lat Medio (m)	0,0457	0,0128	0,0082	-72,0	-82,0
Error Ori Max (rad)	0,5003	0,2765	0,3832	-44,7	-23,4
Error Ori Medio (rad)	0,0501	0,0222	0,0178	-55,8	-64,4

Tabla 8.1. Resultados simulación Control Óptimo.

En primer lugar, el incremento de q11 propicia que el robot complete el recorrido en menos tiempo. Partiendo de la simulación inicial con q11=10, cada simulación sucesiva gana 2 segundos respecto a la anterior, lo que equivale a una mejora de más de un 2%.

Como era de esperar, la diferencia de tiempos propicia que la velocidad lineal media también sale beneficiada. Por otro lado la velocidad lineal máxima es en los tres casos 0.4 m/s al estar limitada en ese valor por el controlador del robot.

En cuanto a la velocidad angular media y máxima, las conclusiones son extrapolables a las realizadas con la historia temporal. Son más elevadas al aumentar q11 aunque no se aprecia una gran diferencia dada la magnitud de los valores, en cualquier caso la variación porcentual si es importante.

El error lateral muestra como se veía a simple vista en la historia temporal una clara mejoría al aumentar el parámetro q11. Se observan mejoras en su valor medio de más del 80% en las simulaciones realizadas.

Para finalizar, el valor máximo del error de orientación se corresponde con algún dato atípico que presenta el robot al llegar al final del recorrido, sin embargo su valor medio ofrece una lectura más realista, llegándose a registrar mejoras de casi el 65%.

8.3 Análisis Control Borroso

Este punto se centra en el análisis y comprobación del correcto funcionamiento del control borroso implementado. Tal como se describió en anteriores capítulos, el control borroso regula la velocidad lineal del robot en función de las reglas de comportamiento establecidas para los cuatro parámetros siguientes: radio de curvatura de la trayectoria, distancia al punto objetivo de la trayectoria y distancia a un obstáculo ya sea estático o dinámico.

Debido a la complejidad de evaluar el funcionamiento del control en una situación donde influyen varios factores simultáneamente, se procede a analizar su comportamiento en cada variable por separado. De esta forma se puede observar con más claridad el resultado obtenido al variar cada uno de los parámetros.

8.3.1 Comportamiento en curva

En primer lugar se analiza cómo influye el radio de curvatura en la velocidad lineal obtenida por el control borroso. Para llevar a cabo la prueba se emplea una trayectoria que alterna curvas con tramos rectos en un entorno donde no existe ningún otro obstáculo que podría desvirtuar el resultado. En estas condiciones se consigue aislar al robot para que el control borroso actúe atendiendo únicamente a la variación de este parámetro. Para comprender mejor el análisis que se pretende llevar a cabo, en la Figura 8.8 se observa la trayectoria propuesta:

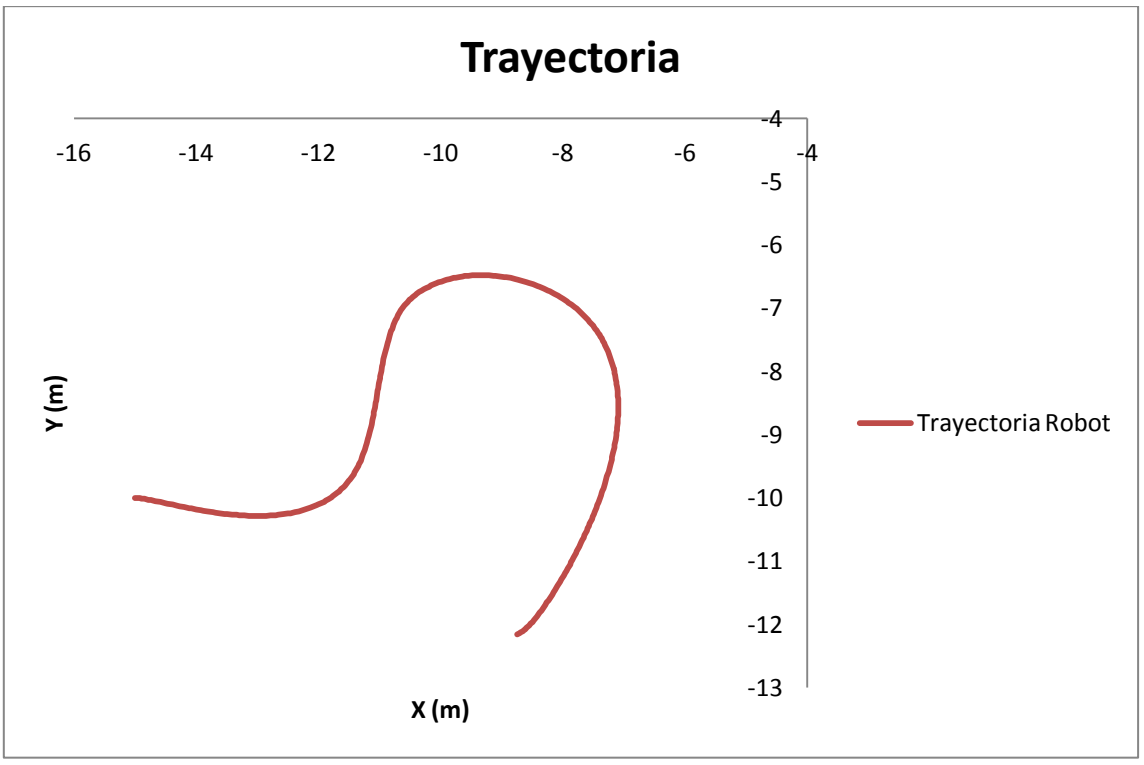


Figura 8.8. Trayectoria de prueba.

A continuación, la Figura 8.9 muestra la evolución de la velocidad lineal a lo largo del recorrido:

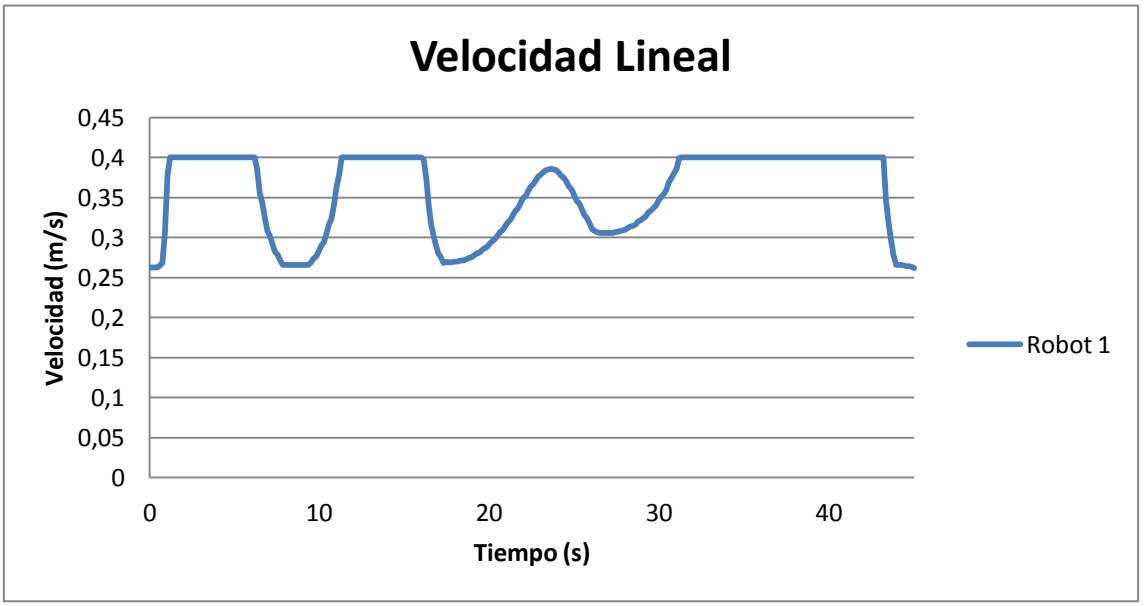


Figura 8.9. Velocidad lineal.

Se observa una clara correspondencia entre los tramos que presentan menor curvatura y la velocidad máxima de saturación, que sufre una repentina bajada cuando el robot llega a los tramos más curvos del recorrido. Del mismo modo, la recuperación es más rápida (mayor aceleración) cuanto más recto es el tramo siguiente a la curva que completa el robot.

8.3.2 Comportamiento en distancia al objetivo

El parámetro distancia al objetivo se ha implementado para garantizar una parada suave del robot cuando llega al final del recorrido. De nuevo se evalúa en las condiciones apropiadas para valorar la influencia real de esta variable en el controlador. Por tanto la trayectoria que se emplea en esta ocasión (Figura 8.10) no presenta ninguna curvatura que haría aminorar la velocidad del robot, ni existen obstáculos en las proximidades.

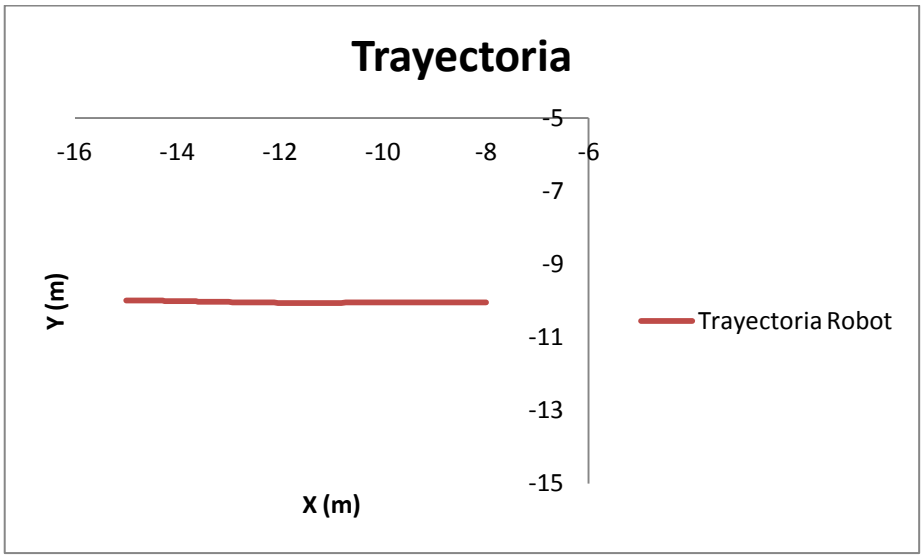


Figura 8.10. Trayectoria para evaluar la distancia al destino.

El gráfico que muestra la variación de velocidad lineal (Figura 8.11) pone de manifiesto que el robot mantiene la velocidad máxima de saturación a lo largo de la recta y comienza a frenar cuando se aproxima al final del recorrido. Este ejemplo representa el caso de mayor desaceleración, dado que el robot se mueve a toda velocidad cuando rebasa el umbral de proximidad que se ha establecido. Se observa cómo ese primer instante provoca un descenso abrupto de la velocidad, que se va haciendo cada vez más progresivo hasta que el robot se detiene al final del recorrido.

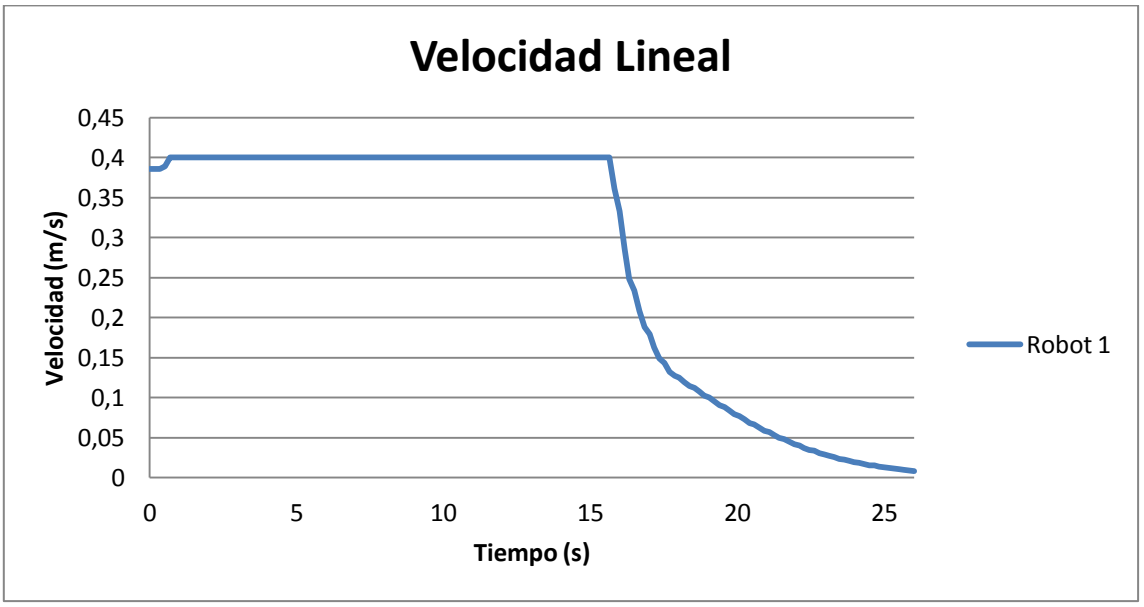


Figura 8.11. Velocidad lineal.

8.3.3 Comportamiento con distancia al obstáculo

El comportamiento del robot en función de la distancia al obstáculo es el caso de control más crítico que se presenta, dado el ajuste minucioso que requieren las diferentes reglas de comportamiento en función del grado de pertenencia que tenga el robot.

Para conseguir que múltiples robots naveguen por el mismo entorno evitando colisiones, se adjudica a cada robot una prioridad que determinará su comportamiento ante la presencia de otros. Con el objetivo de evaluar su funcionamiento se han analizado dos situaciones diferentes en un mismo escenario para las que intervienen dos robots, el primero con un grado de prioridad superior (AZUL), y otro con una prioridad de paso menor (ROJO).

Dada la importancia de este apartado se ha elegido el caso más restrictivo posible, esto es, que la prioridad de ambos robots sea consecutiva (no existen robots con prioridades intermedias) y por tanto la única diferencia entre ambos vendrá dada por un desplazamiento a la derecha del 5% en el conjunto borroso del robot con menor prioridad.

- **CASO 1**

El robot con baja prioridad llega a una intersección junto con el robot un grado de prioridad superior. En la Figura 8.12 podemos ver la sucesión de imágenes que describen la secuencia establecida.

1. Ambos robots se encuentran lejos uno del otro y no se detectan.
2. Los robots se acercan a la intersección y los sensores de ambos comienzan a detectarse.
3. El robot con baja prioridad ha reducido drásticamente su velocidad permitiendo el paso al robot con más prioridad.
4. El robot azul con prioridad más alta es el que primero atraviesa la intersección.
5. El robot azul va desplazándose más rápido y paulatinamente se aleja del robot con baja prioridad.
6. Se observa cómo la distancia entre ambos robots se ha incrementado.
7. Ambos robots han superado la segunda intersección, manteniendo en todo momento una distancia de seguridad considerable.

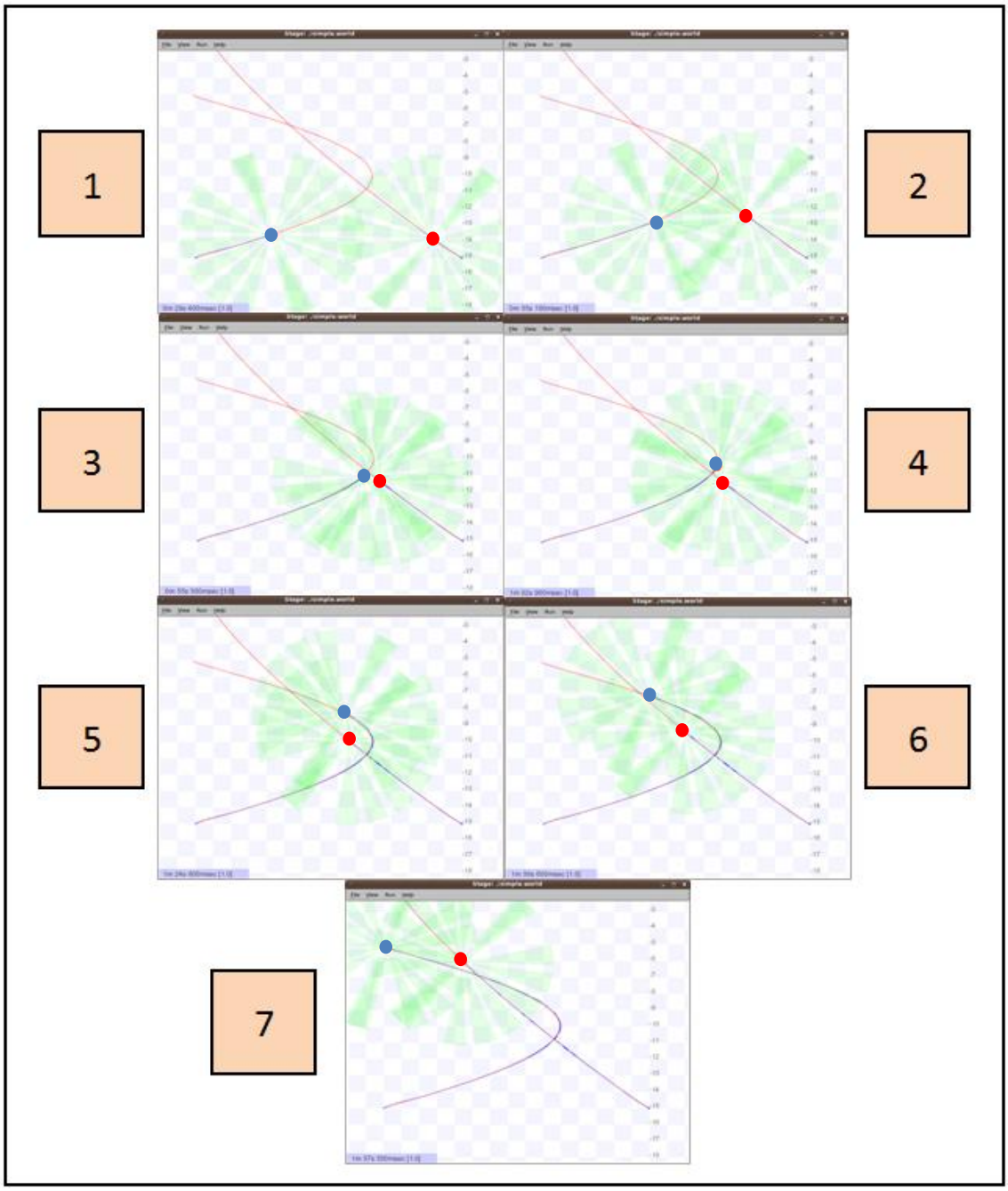


Figura 8.12. Control de prioridad.

A continuación se muestra la historia temporal de la velocidad lineal con la que se desplaza cada uno de los robots manteniendo los colores anteriores (Figura 8.13). Lo primero que llama la atención es comprobar cómo al detectarse entre ellos reducen la velocidad casi simultáneamente, siendo más acusado el descenso en el caso del robot con menor prioridad. Mientras se encuentran lo suficientemente cerca mantienen o reducen la velocidad en función de la distancia, a la vez que se observan pequeños picos debido a la existencia de ángulos muertos en los sensores cuando los robots están girando.

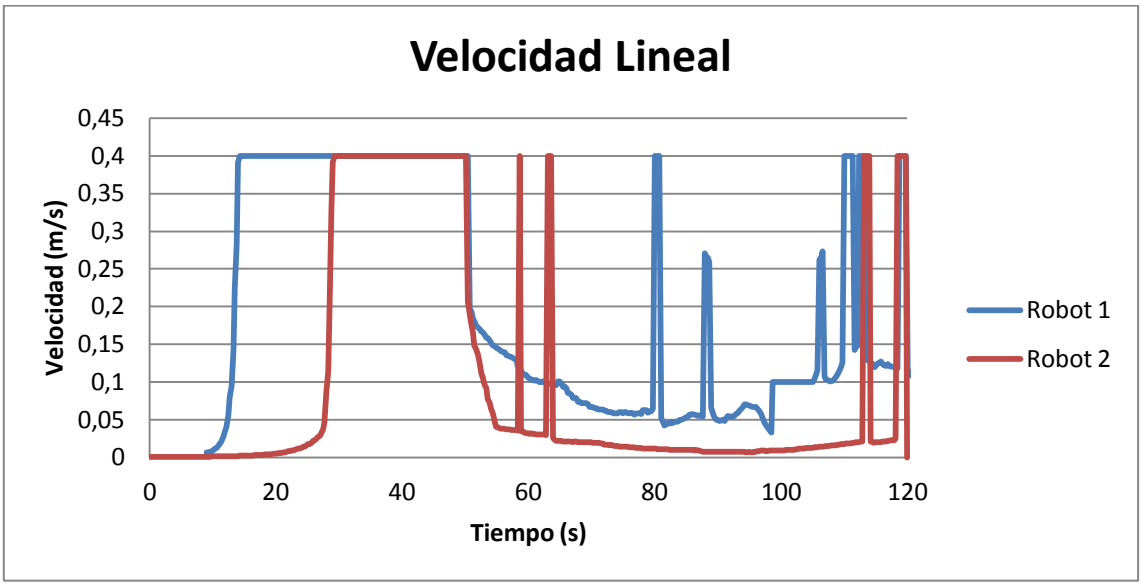


Figura 8.13. Velocidad lineal.

- **CASO 2**

Con el ánimo de buscar los límites del controlador se ha planteado una situación más compleja para analizar el comportamiento. En esta simulación se lanza primero al robot con baja prioridad, dándole tiempo suficiente para que atraviese la intersección en primer lugar. Esto ocurre debido a que el robot de mayor prioridad ha salido con retraso y aún no ha sido detectado. Se mantiene la configuración del nivel de prioridad en el 5%. La sucesión de imágenes que puede verse a continuación se resumiría de la siguiente forma:

1. Robots alejados el uno del otro.
2. Los robots se acercan a la intersección.
3. El robot rojo ha llegado primero a la intersección y evalúa la opción de pasar en función de la distancia al robot azul.
4. El robot rojo cruza la intersección.
5. El robot rojo detecta por detrás al robot azul desplazándose en posiciones muy cercanas a las suya, por lo que baja su velocidad considerablemente.
6. El robot azul cruza la primera intersección.
7. El robot rojo prácticamente no avanza mientras que el azul se desplaza con mayor velocidad a su alrededor.
8. Los robots llegan igualados a la segunda intersección.
9. El robot azul tiene preferencia, por lo que el rojo disminuye aún más su velocidad para ceder el paso manteniendo la distancia mínima de seguridad.
10. Finalmente el robot rojo atraviesa la última intersección.

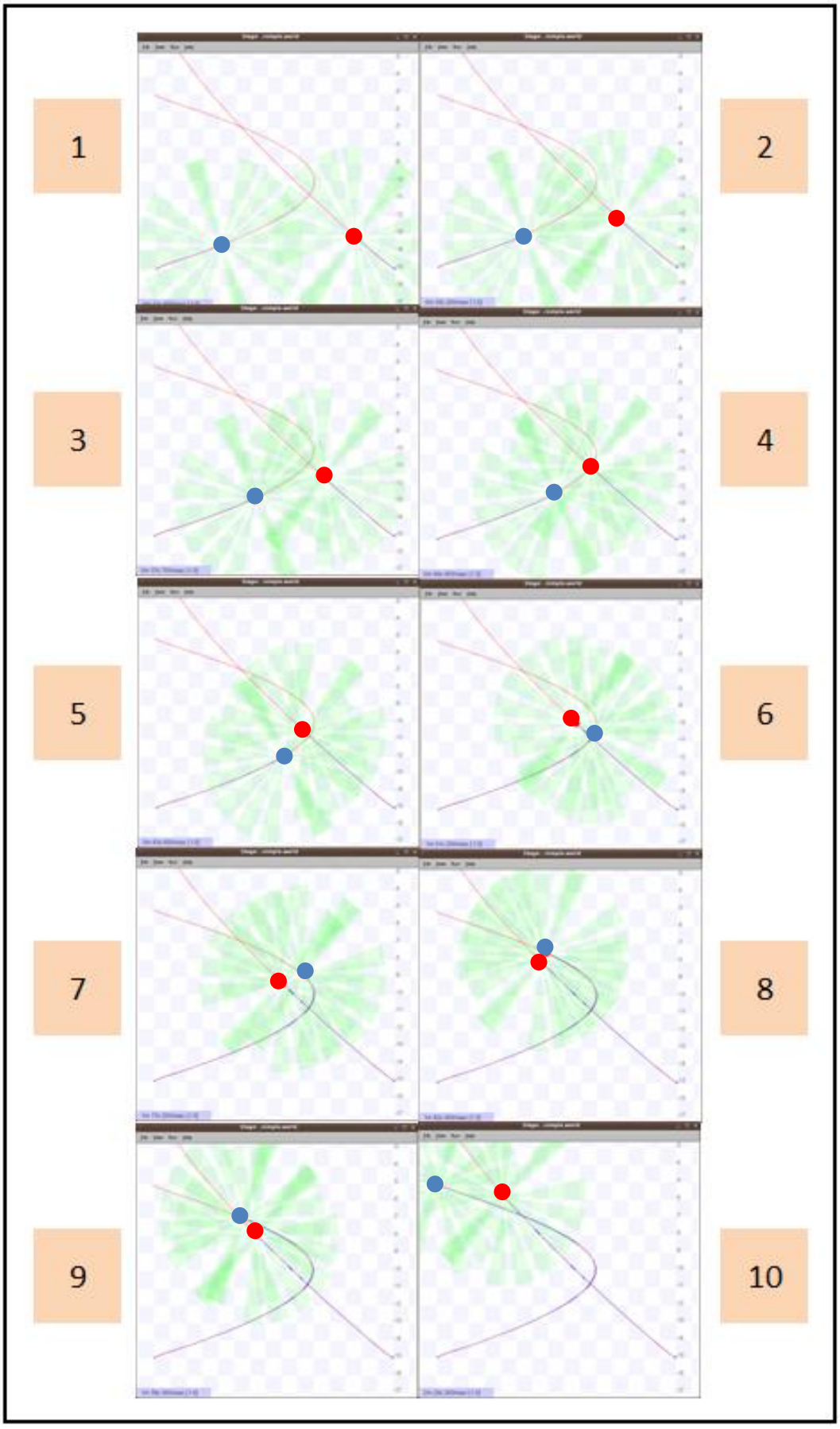


Figura 8.14. Control prioridades (2º caso).

En la Figura 8.15 se muestra de nuevo la evolución de la velocidad lineal en ambos robots. Las conclusiones de la gráfica anterior son extrapolables, presentando algunos picos debidos a la influencia del radio de curvatura, o la intermitente detección del otro robot por parte de los sensores.

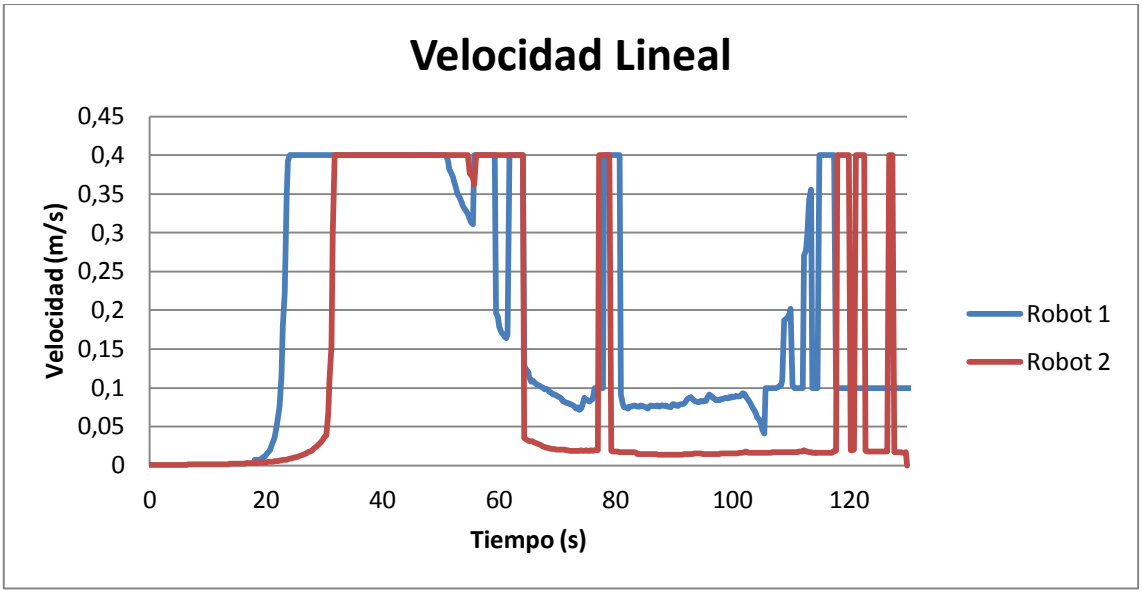


Figura 8.15. Velocidad lineal.

9. Conclusiones

9.1 Conclusiones principales

Dados los resultados obtenidos en las pruebas y simulaciones efectuadas que muestran el correcto funcionamiento del sistema, se puede deducir que el controlador ofrece una respuesta apropiada a los objetivos inicialmente propuestos.

A un nivel más global se ha conseguido cumplir el objetivo principal del proyecto, es decir, la **implementación de un controlador en el entorno Player/Stage** que permita la flexibilidad de trasladar el código generado a robots reales.

Si analizamos más concretamente el trabajo realizado, podemos observar que se ha desarrollado una herramienta de simulación multi robot con las siguientes características:

- **Generación de Trayectorias mediante Splines cúbicas**. A partir de los puntos intermedios suministrados por el usuario se calcula mediante el método de Splines cúbicas la trayectoria a seguir por el robot. Esta técnica garantiza suavidad en los cambios de tramo y proporciona al robot una trayectoria suave sin discontinuidades.

- **Control Óptimo**. La implementación de este controlador permite al robot ajustar la velocidad angular y de este modo seguir sin problemas la trayectoria planificada corrigiendo el error lateral y de orientación que pudiera aparecer en cada instante. Además se ha realizado un ajuste de los parámetros que intervienen en el controlador proporcionando los valores óptimos para su funcionamiento.

- **Control Borroso.** Con este controlador se ha conseguido adaptar la velocidad lineal del robot para cada situación. En todo momento esta viene determinada en función de tres variables:
  - Distancia al punto objetivo.
  - Distancia a un obstáculo móvil/fijo.
  - Radio de curvatura de la trayectoria.

Evaluando el grado de peligrosidad que presenta cada movimiento por medio de las reglas de comportamiento que se han establecido, el robot calculará la velocidad lineal adecuada en cada instante.

- **Controlador Multirobot**. Se ha implementado un controlador multirobot, válido para n robots que confiere a cada robot un comportamiento diferente que evita colisiones entre ellos. Este punto se ha conseguido ajustando la velocidad lineal en función de la prioridad de cada robot al detectarse entre ellos.

- **Información del entorno**. El simulador es capaz de almacenar el mapa de la zona en la que se moverá el robot y determinar los posibles obstáculos estáticos que se puede encontrar.

- **Medidas de seguridad y control añadidas**. Con el fin de aumentar la seguridad del robot y su eficiencia se han implementado funciones adicionales como:

    - Distancia mínima de seguridad: El robot da por finalizado el trayecto al rebasar una distancia de seguridad mínima con un obstáculo que pudiera aparecer en cualquier momento.

    - Ruta alternativa: Si el camino a seguir se encuentra bloqueado por una pared o un muro, el controlador da la posibilidad de introducir una nueva trayectoria que permita al robot continuar su camino hasta el destino previsto.

Interfaz de Usuario. Además de la implementación del controlador descrito anteriormente se ha realizado una interfaz de usuario que permite el manejo del programa para N-robots de forma sencilla. El manual de usuario de la interfaz desarrollada lo podemos encontrar en el capítulo 6.

En último lugar, todas las características implementadas han sido probadas con éxito, por tanto se puede considerar como válida y funcional la aplicación de control desarrollada.

9.2 Trabajos futuros

El control en robótica es un campo ampliamente estudiado desde hace años y que está en continuo crecimiento y evolución. Más adelante se puede continuar ampliando el controlador para dar soporte a nuevas características y adoptar funcionalidades adicionales. Algunos de los pasos propuestos a seguir son:

- **Pruebas con robots físicos**. El objetivo de este proyecto era realizar el software de control en Player/Stage con la idea de trasladar la simulación a un entorno de robots reales el día de mañana.

- **Sistema de Navegación**. La generación de trayectorias mediante Splines podría modificarse de tal modo que los puntos de la trayectoria actualmente introducidos por el usuario se calcularan de forma alternativa por métodos de planificación como *Front-Wave* o similares.

- **Mejora de la lógica de control**. Podría resultar de utilidad que el comportamiento del robot estuviera condicionado no únicamente por la situación actual en la que se encuentra sino atendiendo también a la variación ocurrida respecto al instante anterior (introducción de memoria).

PLANOS

En este capítulo se muestran algunas partes de código que intervienen en la configuración y puesta en marcha de la aplicación. El programa cliente no se ha incluído aquí debido a su extensión, siendo posible consultarlo en el soporte digital que se adjunta al proyecto, así como el resto de código desarrollado en este trabajo.

A continuación se detallan los archivos que se han desarrollado para automatizar la configuración inicial de Player/Stage, de forma que el código generado se ajuste a las necesidades del usuario.

1. Base.cfg

```
# Descripción: Archivo base para controlar dispositivos Stage.
# Autor: Juan Alonso
# Carga del plugin Stage
driver
(
  name "stage"
  provides [ "simulation:0" ]
  plugin "stageplugin"

  # Carga del archivo en el simulador
  worldfile "simple.world"
)
```

2. Cfg.ini

```
# Descripción: Archivo variable para configurar el servidor Player.
# Autor: Juan Alonso
driver
```

```
(
  name "stage"
  provides [ "graphics2d:Indice" ]
  model "cave"
)

driver
(
  name "stage"
  provides [ "position2d:Indice" "sonar:Indice"]
  model Nombre
)

driver
(
  name "mapfile"
  provides ["map:Indice"]
  filename "mapa.png"
  resolution 0.1 # 10 cm por pixel
)
```

3. Base.world

```
# Descripción: Archivo base común que describe el entorno utilizado.
# Autor: Juan Alonso
include "pioneer.inc"
include "map.inc"
include "sick.inc"

# Tiempo máximo de simulación
```

```
quit_time 3600 # 1 hora
paused 1
resolution 0.02

# configuración de la interfaz de usuario
window
(
  size [ 800 800 ] # en pixels
  #scale 37.481   # pixels por metro
  scale 50
  center [ 0  0 ]
  rotate [ 0  0 ]
  show_data 1              # 1=on 0=off
)

# carga del archivo que contiene el mapa del entorno
floorplan
(
size [40 40 1]
  name "cave"
  pose [0 0 0 0]
  bitmap "mapa.png"
)
```

4. World.ini

```
# Descripción: Archivo variable que describe los elementos empleados.
# Autor: Juan Alonso

pioneer2dx
```

```
    (
      # Robot Pionner que se va a emplear
      Nombre robot
      pose [ Coordx Coordy 0 Ang ]

      # Configuración del modo de localización y situación en el
mapa
      localization "gps"
      localization_origin [ 0 0 0 0 ]
    )
```

5. Script principal

El código del script desarrollado para automatizar la configuración, interactuar con el usuario y realizar la puesta en marcha del sistema completo se muestra a continuación:

```
#!bin/bash

yad --image "iconos/playerstage.png" --title "CONTROLADOR
MULTITRAYECTORIAS" --timeout=4 --no-buttons --undecorated
A=$(yad --entry --title="CONTROLADOR DE TRAYECTORIAS"
--text="\n\n\nBIENVENIDO AL CONTROLADOR DE TRAYECTORIAS\n\nPARA
PLAYER/STAGE\n\n\n\nPor favor, introduzca el número de robots
que desea emplear:" --image "iconos/exterior.jpg")

cp base.cfg controlador.cfg
cp base.world controlador.world

for i in `seq 0 $[A-1]`
    do
    CAMPOS=0
    while [ "$CAMPOS" -lt "3" ]
```

```
    do

    DATOS=$(yad --title "DETALLE DEL ROBOT "$[i+1]""

    --text="\n\n\n

    Introduzca la posición inicial del robot número
"$[i+1]"\n\n" --form --field "Coordenada X:"

    --field "Coordenada Y:" --field "Orientación (0-360°):"

    --image "iconos/malla.png" --separator=" " --button=gtk-ok:0
&& exit 1)

    Cx=$(echo $DATOS | awk 'BEGIN {FS=" " } { print $1 }')

    Cy=$(echo $DATOS | awk 'BEGIN {FS=" " } { print $2 }')

    Ang=$(echo $DATOS | awk 'BEGIN {FS=" " } { print $3 }')

    NUM=($DATOS)

    CAMPOS=${#NUM[@]}


    if [ "$CAMPOS" -lt "3" ]; then

    yad --title "ATENCIÓN" --text "\n  No ha introducido todos
los datos necesarios. \n\n  Por favor, vuelva a intentarlo." --
button=Aceptar:0 --image "iconos/alerta.png"

    fi

    done

    echo $Cx $Cy > trayectoria$i.txt


    echo "scale=4;(($Ang*3.1416)/180)" | bc >> trayectoria$i.txt


    sed -e 's/trayectoria.txt/trayectoria'"$i"'.txt/g'  tbase.m
> trayectoria.m


    sed -e 's/trayectoria.txt/trayectoria'"$i"'.txt/g'

    -e 's/coeficientesu.txt/coeficientesu'"$i"'.txt/g' Dbase.m >
DosD_spline.m


    yad --title "CONTROLADOR MULTITRAYECTORIA"

    --text "\n   A continuación deberá introducir la trayectoria
deseada para el robot número "$[i+1]"   "
```

```
    --no-buttons --timeout=2 --image "iconos/siguiente.png"

    octave interfaz.m

    sed -e 's/Crear position2d/position2d =
playerc_position2d_create(client,'"$i"');/'

-e 's/Crear sonar/sonar = playerc_sonar_create(client,'"$i"');/'
-e 's/Crear graficos / graficos =
playerc_graphics2d_create(client,'"$i"');/'

-e 's/Crear mapa/map = playerc_map_create(client,'"$i"');/'

-e 's/trayectoria.txt/trayectoria'"$i"'.txt/g'

-e 's/coeficientesu.txt/coeficientesu'"$i"'.txt/g'

-e 's|Indice|(dobs_max/2)*(1+('"$i"'*0.1));|g' velo.c > velo$i.c

sed -e 's/Nombre robot/name "r'"$i"'"/'

-e 's/Coordx/'"$Cx"'/' -e 's/Coordy/'"$Cy"'/'

-e 's/Ang/'"$Ang"'/' world.ini >> controlador.world

sed -e 's/Indice/'"$i"'/g'

-e 's/Nombre/"r'"$i"'"/' cfg.ini >> controlador.cfg


./compila robot$i

    done


yad --title "CONTROLADOR MULTITRAYECTORIA"

--text "\n  Se han generado los ficheros necesarios, ¿desea
comenzar la simulación? "

--image "iconos/ok3.png" --button=Comenzar:0

--button=Salir:1

player Controlador.cfg &


for i in `seq 0 $[A-1]`

do

    ./robot$i &

done

exit
```

PLIEGO DE CONDICIONES

En este apartado se detallan las especificaciones técnicas de los equipos y el material informático que ha sido necesario para la realización de este trabajo.

1. Hardware

Ordenador portátil HP dv6.

Microprocesador: Intel Core i7 1,6 Ghz.

Memoria RAM: 4 GB DDR3

Disco duro: 500 GB

Resolución pantalla: 1366x768 pixels.

Impresora HP 3050.

Modo de impresión	Inyección térmica de tinta
Color:	4.800 x 1.200 ppp.
Negro:	600 x 600 ppp
Conectividad	USB 2.0, Wi-Fi.

2. Software

Sistemas Operativos

Windows 8 Pro.

Linux Ubuntu 9.10

Herramientas Software

Player 3.0.1

Stage 3.2.2

Matlab 7.6

Octave 3.2.2

Microsoft Office 2010

Oracle Virtual Box 4.2.4.

PLANIFICACIÓN Y PRESUPUESTO

P.1 Planificación

Se muestra a continuación las actividades necesarias para el desarrollo del proyecto, así como su duración estimada.

Número	Actividad	Duración (horas)	Precedencias
1	Dirección del proyecto	100	---
2	Planificación	20	1
3	Formación	170	2
4	Programación Herramienta	100	3
5	Análisis resultados	180	4
6	Puesta a punto Herramienta	100	4
7	Redacción del proyecto	180	6
	TOTAL	850	

Tabla P.1. Actividades necesarias para la realización del proyecto.

Se ha considerado en la planificación del proyecto una duración del mismo de aproximadamente 12 meses con las siguientes fechas de comienzo y entrega:

Fecha de comienzo: 15 de Septiembre de 2012.

Fecha de entrega: 31 de Agosto de 2013.

P.2 Presupuesto

En este punto se detalla la aproximación realizada del coste total que supondría la elaboración del proyecto descrito. El conjunto de gastos se ha dividido en tres apartados en función del tipo de importe a tratar:

- Ejecución material.
- Gastos generales y beneficio industrial.
- Honorarios de dirección y redacción.

En cada uno de los apartados se desglosarán los gastos ocasionados presentando finalmente un presupuesto total para la realización del proyecto.

P.2.1 Ejecución Material

El presupuesto en material necesario se puede diferenciar en tres apartados básicos:

A. Costes de equipos

EQUIPO	PRECIO
Ordenador Portátil HP	699 €
Impresora HP	70 €
Total	769 €

Tabla P.2. Costes de equipos.

B. Coste del material empleado

MATERIAL	PRECIO
Ubuntu 9.10	0 €
Microsoft Windows 8	135 €

Microsoft Office 2010	**200 €**
Player-2.0.4	**0 €**
Stage-2.0.3	**0 €**
Material de oficina	**100 €**
Total	**435 €**

Tabla P.3. Costes material empleado.

C. Coste mano de obra

FUNCIÓN	Nº HORAS	€/HORA	TOTAL
Ingeniería	**850**	**50 €**	**42.500 €**
Mecanografiado	**200**	**12 €**	**2.400 €**
		Total	**44.900 €**

Tabla P.4. Costes mano de obra.

Costes Totales de Ejecución Material

Sumando los tres conceptos descritos anteriormente obtendríamos:

PARTIDA	PRECIO
Coste de equipos	**769 €**
Coste Material	**435 €**
Coste por mano de obra	**44.900 €**
Total	**46.104 €**

Tabla P.5. Costes totales ejecución material.

P.2.2 Gastos generales y beneficio industrial

Este apartado abarca los gastos generados en las instalaciones que debieran ser utilizadas para la realización del proyecto más el beneficio industrial del mismo. El cálculo total de ambas cosas se estima aproximadamente como el 20% del presupuesto de ejecución material del proyecto.

Por lo tanto, el valor de los gastos generales y beneficio industrial es de $0.2 \cdot 46.104 = 9.220$ €

P.2.3 Honorarios de dirección y redacción

Los gastos de honorarios de dirección y la redacción de documentos han sido calculados como el 7% de los gastos del presupuesto de ejecución material, quedando finalmente:

Presupuesto de ejecución material	**46.104 €**
Honorarios por dirección	**4.014 €**
Honorarios por redacción	**4.014 €**
Total	**54.132 €**

Tabla P.6. Costes honorarios de dirección y redacción.

P.2.4 Costes totales

PRESUPUESTO	**TRABAJO FIN DE GRADO**					**U.A.H.**	
Nº de orden	PRESUPUESTO EJECUCIÓN POR CONTRATA						
	CONCEPTOS	Nº UNIDADES		Precio unitario		Importe	
1	Ejecución Material	1	Unid	57.344	€/Unid	46.104	€
2	Gastos generales y beneficio industrial	1	Unid	11.468	€/Unid	9.220	€
3	Honorarios de redacción y redacción	1	Unid	8.028	€/Unid	8.028	€
		SUMA				**63.352**	**€**
		IVA (21%)				**13.303**	**€**
		PRESUPUESTO TOTAL				**76.655**	**€**

Tabla P.7. Presupuesto final.

MANUAL DE USUARIO

M.1. Introducción.

El desarrollo del controlador abarca diversos programas que trabajan de forma independiente y coordinada, por tanto es posible preparar unos ficheros *ad-hoc* con la configuración deseada y ejecutar cada parte por separado. Sin embargo, se ha realizado un programa encargado de elaborar los ficheros necesarios, adaptarlos a las necesidades del usuario y ejecutar los programas en el orden apropiado. Para facilitar la interacción con el usuario se introduce una interfaz gráfica que recopila los datos, informa de las acciones que se llevan a cabo y avisa ante cualquier imprevisto si es necesario.

M.2. Manejo del programa

M.2.1. Puesta en marcha

Una vez nos encontramos dentro de Ubuntu, para arrancar el programa basta con hacer doble click sobre su icono y seleccionando la opción “Ejecutar”, o bien a través de la línea de comandos del terminal con la orden “./Controlador.sh”, asegurándonos de estar posicionados en el directorio donde se encuentre.

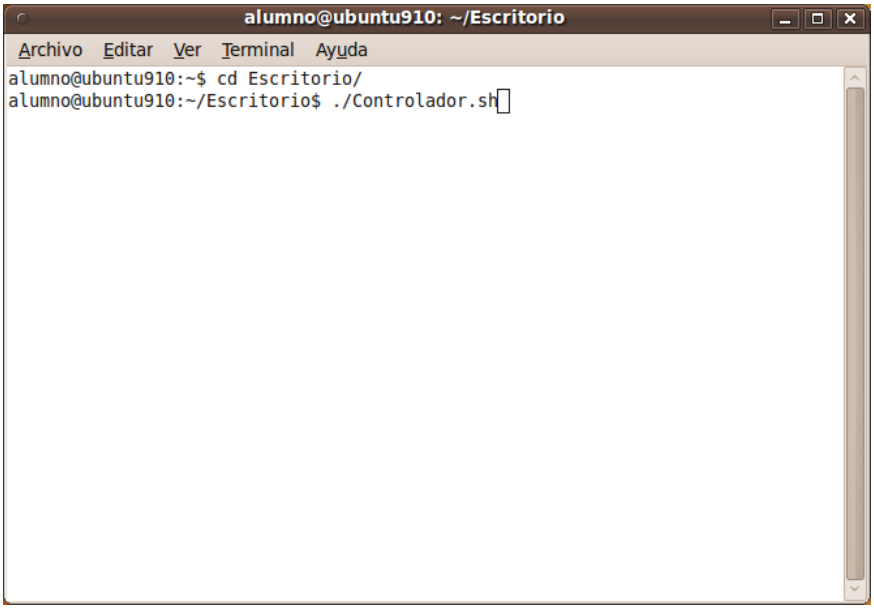


Figura M.1. Puesta en marcha del controlador.

En primer lugar aparece la ventana de presentación mostrada en la Figura M.2. En esta primera pantalla podemos ver el título de la aplicación, nombre del autor y titulación.

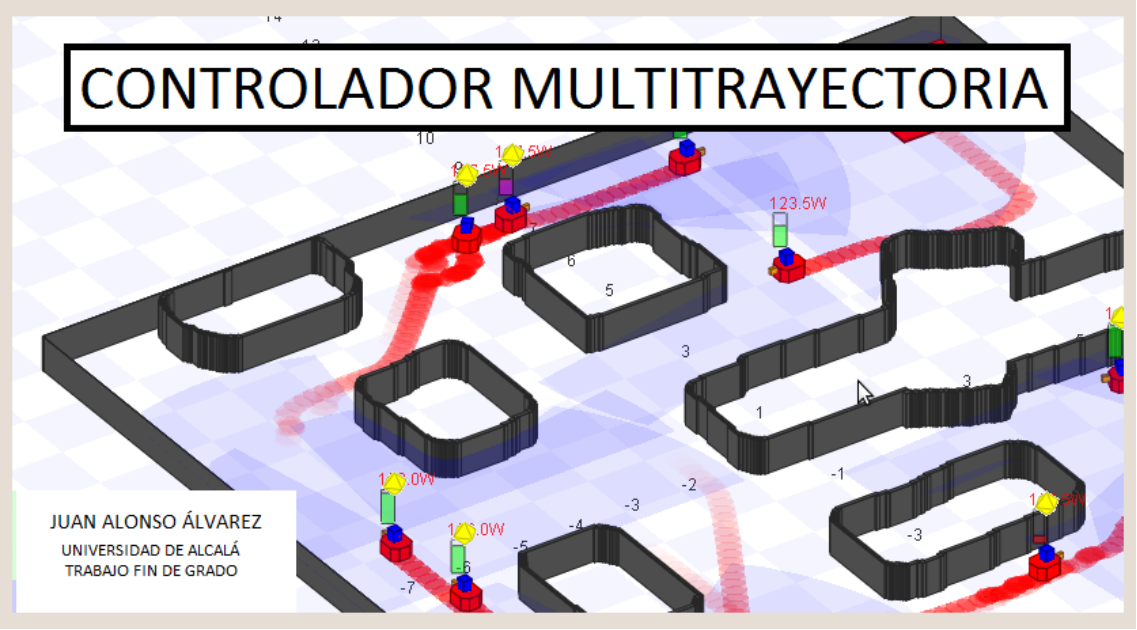


Figura M.2. Pantalla de bienvenida.

M.2.2. Desarrollo del programa

- **Introducción de datos iniciales**

A continuación, en la siguiente ventana (Figura M.3) debemos indicar el número de robots a emplear en la simulación. Es posible introducir cualquier valor a partir de cero, si bien un número demasiado elevado implicará una mayor carga computacional y, dependiendo del hardware que se disponga la velocidad de ejecución puede verse afectada.

Figura M.3. Selección del número de robots a emplear.

El siguiente paso será introducir la posición y orientación iniciales para cada robot. Como se aprecia en el cuadro de diálogo (Figura M.4), en la parte izquierda de la ventana se visualiza el eje de coordenadas sobre el mapa del entorno para conocer las dimensiones globales y localizar rápidamente los obstáculos estáticos que presenta. Los valores "X" y "Y" pueden ser números reales y deben estar contenidos en el rango que ofrezca el eje de coordenadas para que los robots no aparezcan situados fuera del mapa. La orientación por su parte será un ángulo que se introduce en grados, también puede tratarse de un número real, y admite valores negativos.

Si algún dato es incorrecto o no se han completados todos los campos que caracterizan la configuración inicial del robot, aparecerá un aviso (Figura M.5).

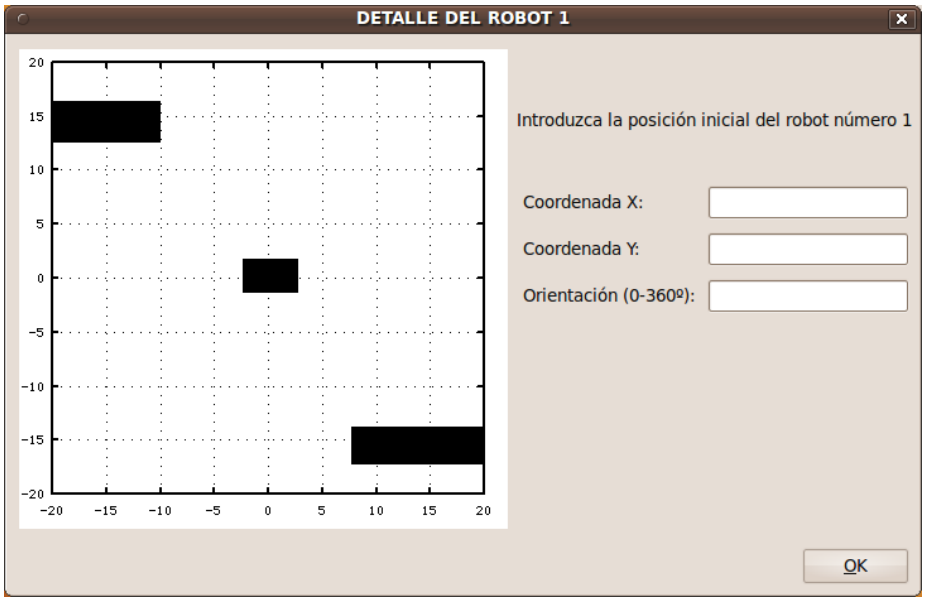


Figura M.4. Posición y orientación iniciales del robot.

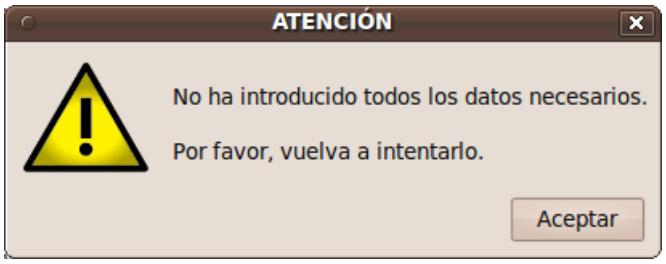


Figura M.5. Mensaje de error al validar los datos.

- **Introducción de trayectorias**

Cuando el usuario confirma la configuración inicial deseada, aparece un mensaje que invita a introducir la trayectoria (Figura M.6). Seguidamente se presenta una pantalla (Figura M.7) donde volvemos a ver el mapa del entorno, incluyendo esta vez un dibujo del robot con la posición y orientación que le corresponde en función de los datos que hayamos introducido en el paso previo. El indicador de orientación es un guión que sale del centro del robot e indica hacia donde mira en su posición de reposo.

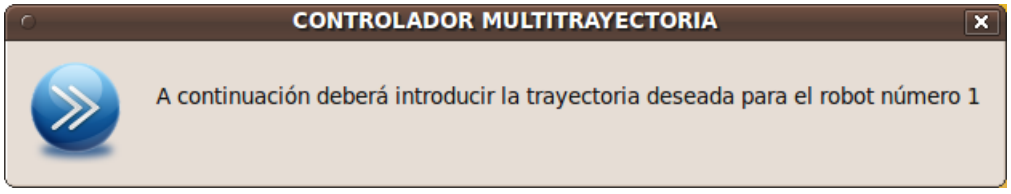


Figura M.6. Información sobre la siguiente acción a realizar.

Para definir la trayectoria deseada, el usuario deberá introducir puntos intermedios por los que desea que pase el robot, teniendo en cuenta que el último punto no pertenecerá a la trayectoria, sino que servirá para definir la orientación final que deseamos una vez el robot ha completado el recorrido. Cuando se hayan introducido todos los puntos, el usuario debe pulsar "ENTER" para completar el proceso y acto seguido aparecerá en pantalla la trayectoria calculada además de las posiciones inicial y final del robot (Figura M.8).

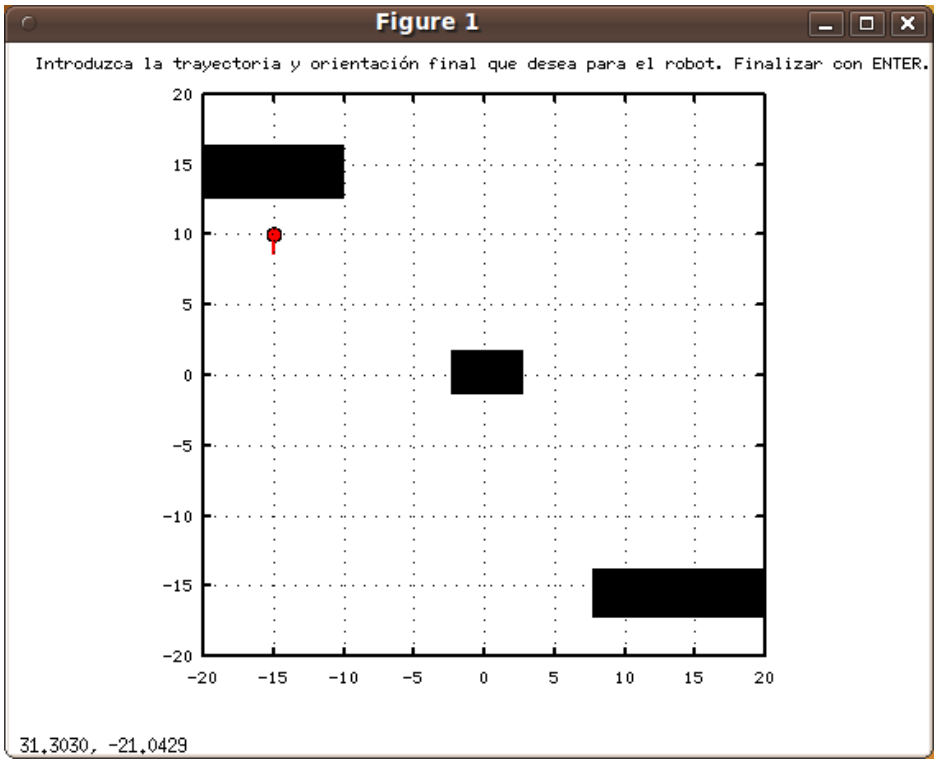


Figura M.7. Introducción de puntos.

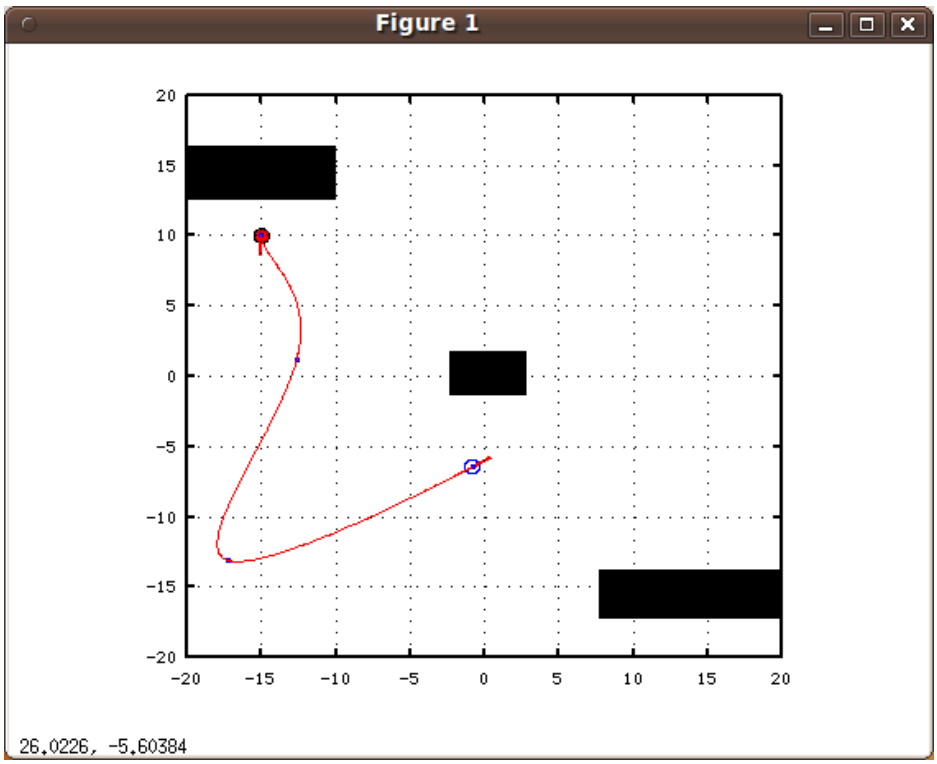


Figura M.8. Trayectoria generada.

Este proceso (Figura M.4 a Figura M.8) se repetirá secuencialmente tantas veces como número de robots sean requeridos para la simulación.

- **Comienzo de la Simulación**

Con todos los datos recopilados, el programa está listo para empezar y aparece el siguiente cuadro de diálogo que dará paso a la simulación cuando el usuario estime oportuno (Figura M.9).

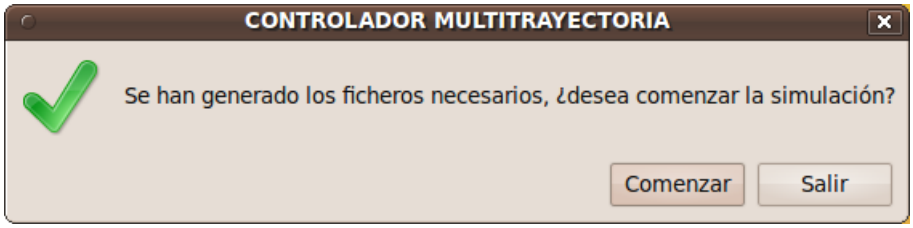


Figura M.9. Controlador preparado para la simulación.

La siguiente imagen que veremos será el mapa completo con todos los robots dispuestos en su posición (Figura M.10). Inmediatamente después podemos apreciar que se produce la conexión con el servidor al aparecer el dibujo de la trayectoria sobre el mapa y el rango de alcance de los dispositivos sónar, momento en el que los robots empiezan a moverse (Figura M.11 y Figura M.12). Para tener una referencia acerca del recorrido completado, el avance sobre la trayectoria aparece dibujado en color azul.

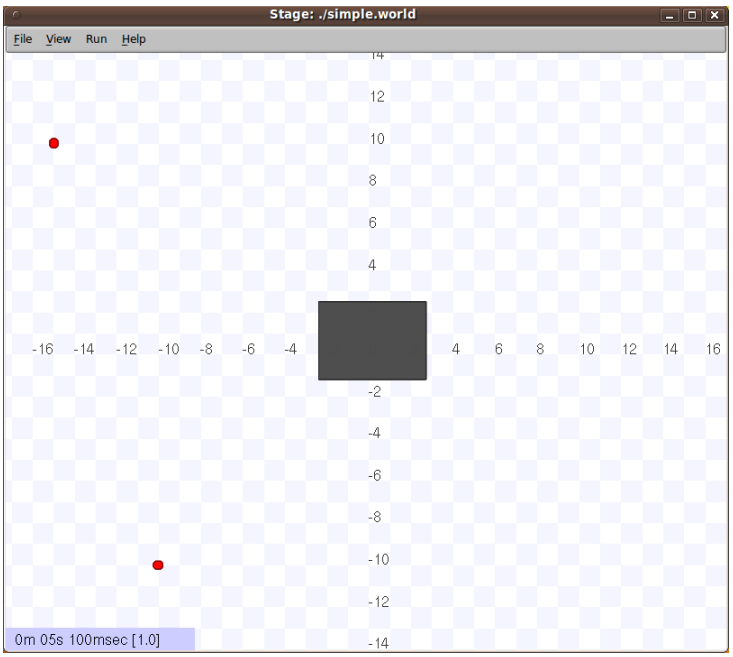


Figura M.10. Inicio de la simulación.

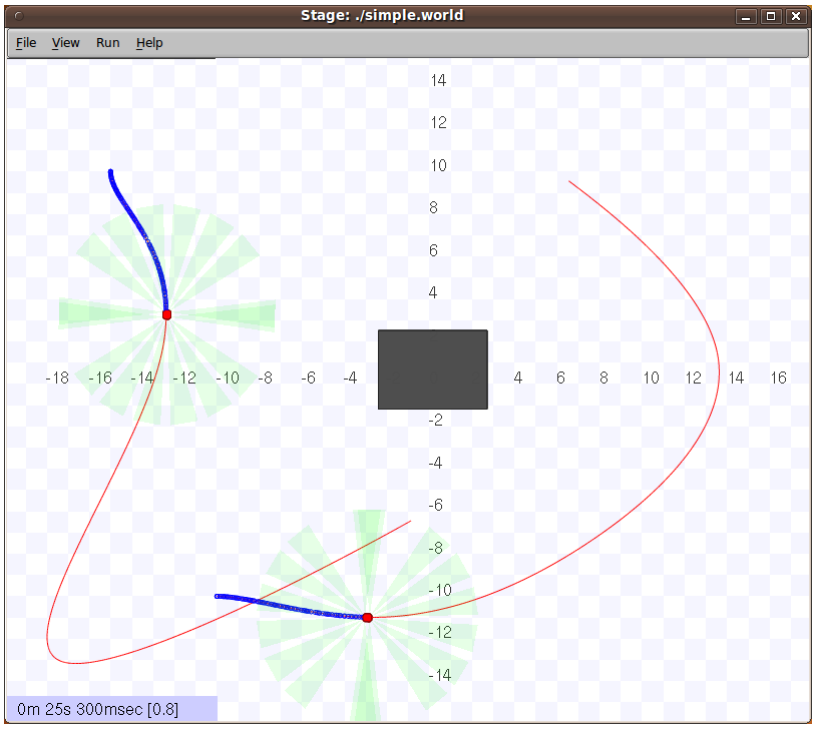


Figura M.11. Arranque de los robots.

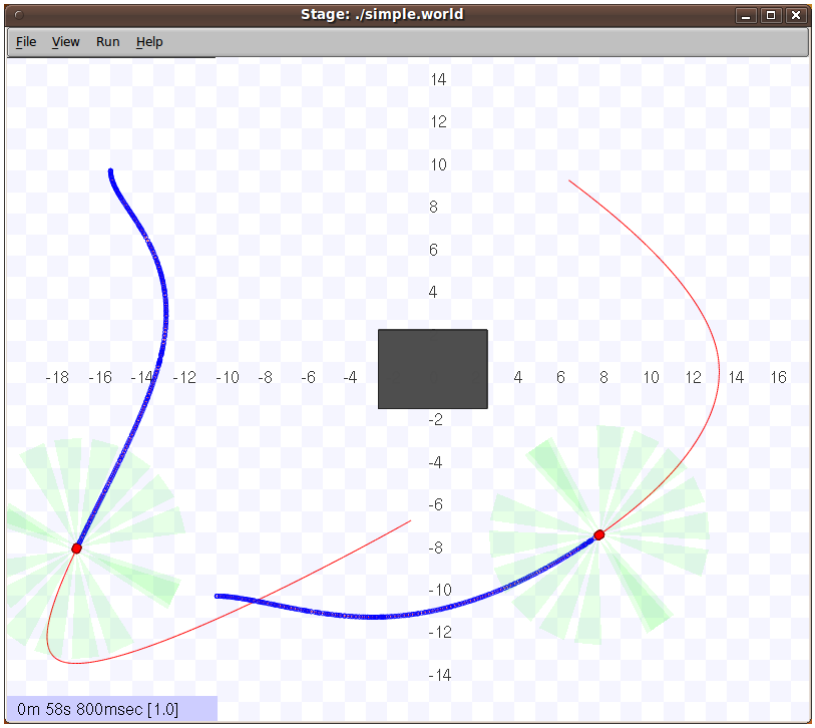


Figura M.12. Avance de los robots.

A continuación podemos ver un ejemplo del desarrollo de la simulación con múltiples robots siguiendo su trayectoria, donde se puede apreciar el avance de todos ellos en función del tiempo (Figura M.13 y Figura M.14):

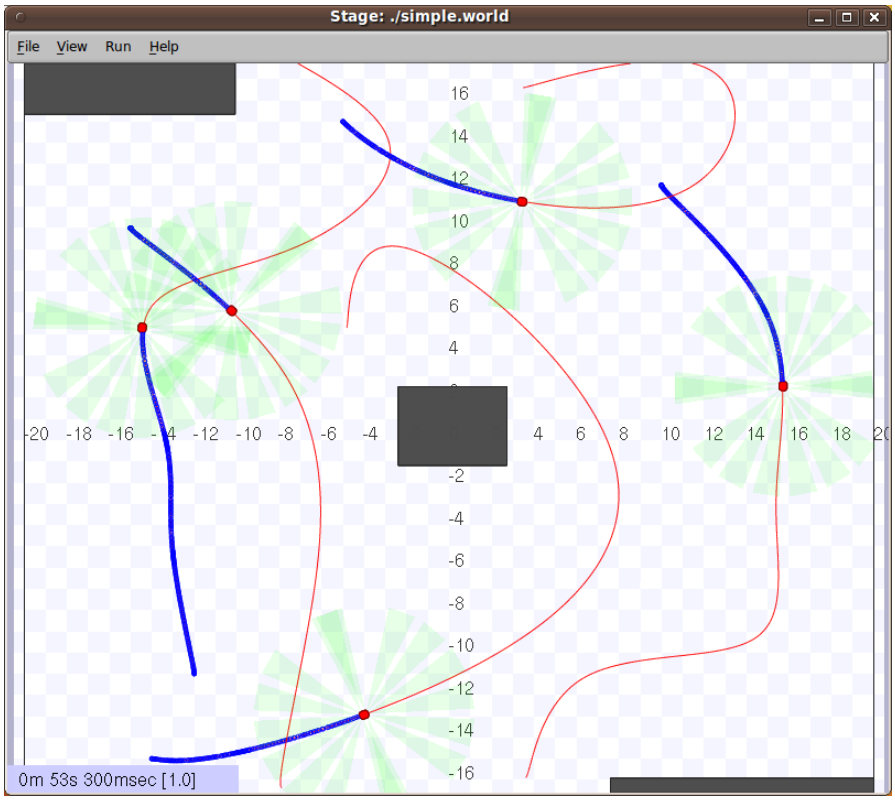


Figura M.13. Ejemplo de simulación con varios robots.

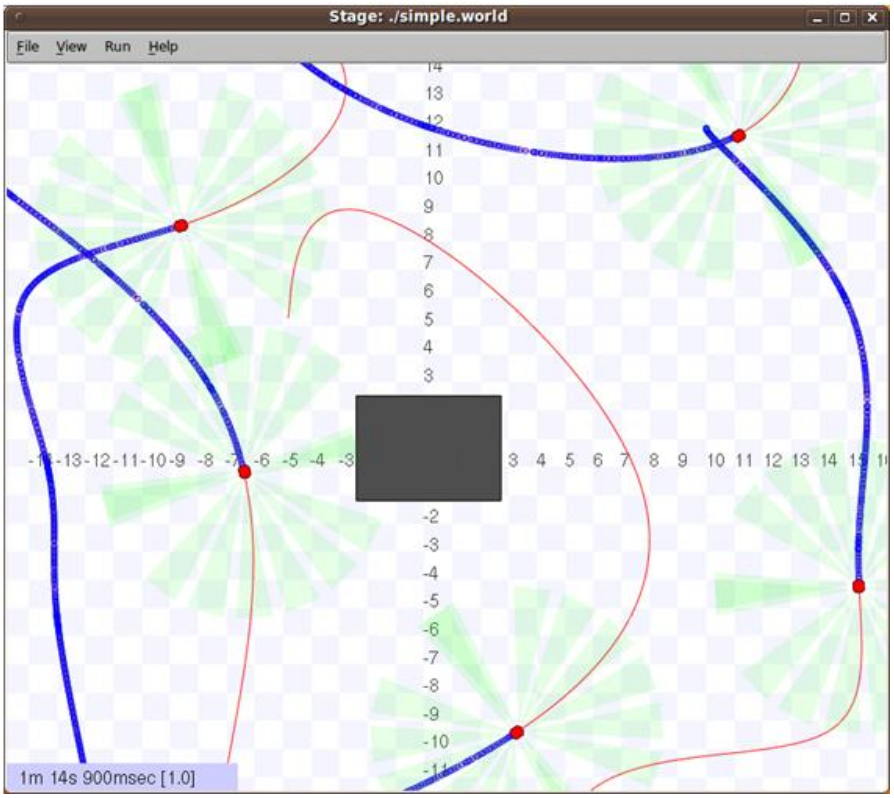


Figura M.14. Avance de los robots en función del tiempo.

Si, durante el recorrido de la trayectoria surgiera algún problema, como podría ser el hecho de prever una colisión, o bien que la trayectoria generada atraviese un muro, el programa informa de la situación y ofrece trazar una ruta alternativa en su caso (Figura M.15 y Figura M.16).

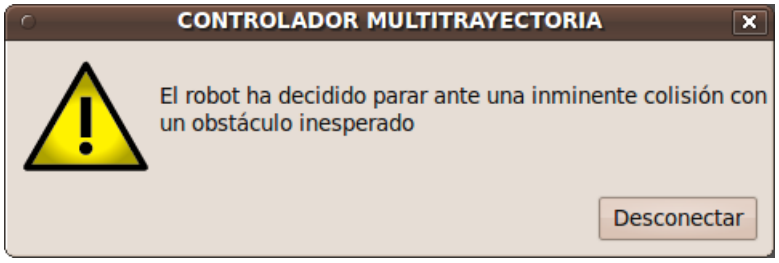


Figura M.15. Parada de emergencia.

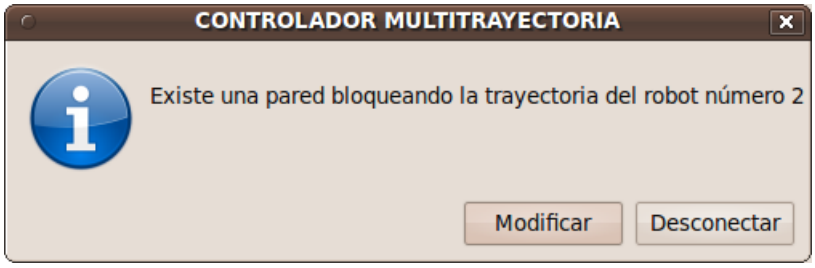


Figura M.16. Mensaje informativo.

Cuando el robot ha llegado a la posición de destino vemos un mensaje informativo y tendremos posibilidad de desconectar el robot que ha finalizado (Figura M.17).

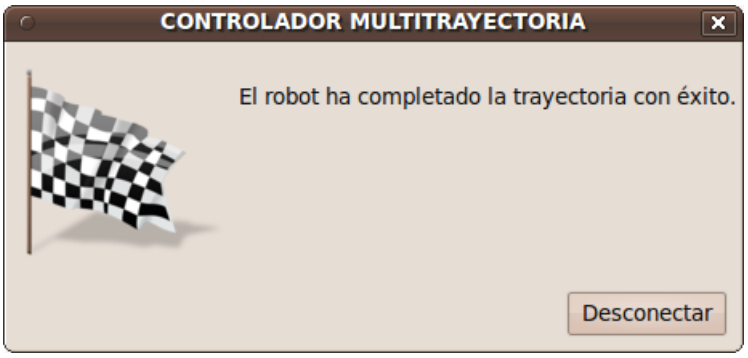


Figura M.17. Fin del recorrido.

BIBLIOGRAFÍA

(1) López Guillén, Elena (1999). *Aplicación de controladores óptimo y Borroso al seguimiento de Trayectorias de una silla de rueda*. Trabajo Fin de Carrera. Ingeniería en Electrónica UAH.

(2) Sánchez Benavente, Antonio (2002). *Generador de trayectorias para sistemas multirobot mediante splines cúbicas*. Trabajo Fin de Carrera. Ingeniería Técnica en Telecomunicación UAH.

(3) Heredia Matesanz, Ismael (2010). *Implementación de métodos de localización y mapeado para robots Pioneer en entorno Player/Stage*. Trabajo Fin de Carrera. Ingeniería Técnica Industrial UAH.

(4) Alonso Sanz, Inés (2008). *Generación de trayectorias con Player para la navegación de un robot móvil en espacios inteligentes*. Trabajo Fin de Carrera. Ingeniería Técnica Industrial UAH.

(5) "Discrete Time Control Systems". K. Ogata.

(6) Ocaña Miguel, Manuel. López Guillén, Elena. *Teoría asignatura* "*Percepción y Control"*. Grado Ingeniería de Computadores. UAH.

(7) Geoff Biggs, Toby Collet, Brian Gerkey, Andrew Howard, Nate Koenig, Jordi Polo, Radu Rusu and Richard Vaughan. "The Player Proyect", Manual User. http://playerstage.sourceforge.net/player

(8) Li-Xin Wang (University of California al Berkeley). *"Adaptive fuzzy systems and control. Design and Stability Analysis". Prentice Hall.*

(9) Ridao Carlini, Miguel Ángel (1995). *Generación automática de trayectorias libres de colisiones para múltiples robots manipuladores*. Tesis Doctoral. Escuela Técnica Superior de Ingeniería Industrial. Universidad de Sevilla.

(10) D. Gallardo, O. Colomina y F. Florez (2008) *Generación de trayectorias robustas mediante computación evolutiva.*. Departamento de Tecnología Informática y Computación. Universidad de Alicante.

ANEXO

A.1. Fundamentos de lógica borrosa.

En este anexo de introducen los fundamentos teóricos que han sido aplicados al desarrollar el control borroso que obtiene la velocidad lineal del robot. En el quinto capítulo de la memoria se exponen los detalles de la implementación.

A.1.1. Introducción.

La teoría de Conjuntos Borrosos se basa en el reconocimiento de que determinados conjuntos poseen unos límites imprecisos. Estos conjuntos están constituidos por colecciones de objetos para los cuales la transición de pertenecer o no a los mismos no es abrupta, sino gradual.

Supóngase, por ejemplo, que se desea evaluar la temperatura de una habitación. El universo de discusión lo constituyen todas las posibles temperaturas a las que puede encontrarse la misma. Si se pretende clasificar dichas temperaturas en varios conjuntos, como “muy alta”, “alta”, “media”, “baja” y “muy baja”, pueden seguirse dos alternativas:

- Utilizar Conjuntos Clásicos, de forma que los límites de dichos conjuntos están perfectamente definidos (por ejemplo, la temperatura es “media” si es mayor que 20 grados y menor que 25). Los límites de estos conjuntos están claramente delimitados y todos los elementos de un conjunto pertenecen al mismo por igual. La pertenencia de una temperatura a un conjunto se evalúa de forma binaria : o pertenece (1) o no pertenece (0).
- Utilizar Conjuntos Borrosos, de forma que una determinada temperatura pueda pertenecer a diferentes conjuntos en grados distintos. De esta forma, la transición de un conjunto a otro no es abrupta, sino gradual. La pertenencia de una temperatura a un conjunto no es “todo o nada”, sino que viene determinada por un número real en el intervalo (0,1).

A.1.2. Funciones de Pertenencia

Sea X una colección de objetos que constituyen el universo de entrada o de discusión de un sistema borroso. Cada conjunto borroso A de X queda caracterizado por una Función de Pertenencia $\mu A(x)$ ($0<\mu A(x)<1$) que representa el grado de pertenencia de un elemento $x \in X$ al conjunto A. En un conjunto clásico, las funciones de pertenencia sólo toman los valores 0 y 1 mientras que en los conjuntos borrosos pueden tomar cualquier valor comprendido en ese rango.

La Figura A.1 muestra las funciones de pertenencia de tres conjuntos borrosos asociados a las variables lingüísticas "lenta", "media" y "rápida" para caracterizar la velocidad de un coche. En este ejemplo, el universo de discusión son todas las posibles velocidades del coche, es decir, X=[0, vmáx], donde vmáx es la velocidad máxima del coche. El elemento x=80 Km/h, por ejemplo, tiene una pertenencia de 0.5 al conjunto "lenta" (μlenta(80)=0.5), de 0.5 al conjunto "media " (μmedia(80)=0.5), y de 0 al conjunto "rápida" (μrápida(80)=0).

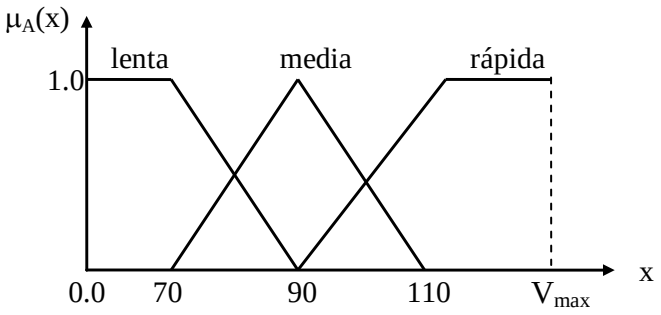


Figura A.1. Ejemplo de conjuntos borrosos y funciones de pertenencia.

La única condición que debe satisfacer una función de pertenencia es que varíe entre 0 y 1. Por lo demás, la función puede tener cualquier forma, siendo las más usadas las funciones triangulares, trapezoidales, gaussianas y sigmoidales [4]. Existe, además, una nomenclatura específica para referirse a ciertas características de las funciones de pertenencia

- *Soporte de una función de pertenencia*: es el conjunto de elementos $x \in X$ para los cuales $\mu_A(x)>0$.
- *Centro de una función de pertenencia:* es el valor x para el cual la función de pertenencia alcanza su valor máximo, generalmente 1.

A.1.3. Operaciones con Conjuntos Borrosos

En la Teoría de Conjuntos Borrosos, al igual que en la Teoría Clásica de Conjuntos, se definen con precisión operaciones entre conjuntos. La Teoría de Conjuntos Borrosos fue formulada inicialmente en base a las siguientes operaciones (8):

- Complementación : $\overline{\mu}_A(x) = 1 - \mu_A(x)$
- Unión: $\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- Intersección : $\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$

Las operación complementaria, unión e intersección definidas de esta forma constituyen una generalización de sus correspondientes operaciones en la Teoría Clásica de Conjuntos. Con ellas se obtienen los mismos resultados que con las operaciones clásicas cuando las pertenencias toman únicamente los valores 0 o 1. Sin embargo, las operaciones máximo y mínimo no son las únicas opciones para implementar la unión e intersección respectivamente de dos conjuntos borrosos. Para ambas operaciones se han definido diferentes tipos de funciones con la restricción de verificar una serie de propiedades.

Así, una determinada función u debe satisfacer al menos los siguientes axiomas para que se considere unión entre dos conjuntos borrosos:

- Condiciones en los límites : $u(0,0) = 0$, $u(0,1) = u(1,0) = u(1,1) = 1$
- Conmutatividad : $u(a,b) = u(b,a)$
- Asociatividad : $u[u(a,b),c] = u[a,u(b,c)]$
- Monotonicidad : Si $a \le a'$ y $b \le b'$, entonces $u(a,b) \le u(a',b')$

De igual forma, para que una función i pueda usarse como intersección de conjuntos borrosos debe verificar los tres últimos axiomas anteriores, y la siguiente condición en los límites:

$$i(1,1) = 1$$
$$i(1,0) = i(0,1) = 0$$

Las funciones u o i que verifiquen estos axiomas se definen en la literatura como Conorma Triangular (T-Conorma) o Norma Triangular (T-Norma) respectivamente. Las Conormas y Normas se hallan relacionadas entre sí a través del operador complementario (Ley de Morgan). Algunas de las más utilizadas se muestran en la Tabla A.1:

Conormas	**Normas**
max(a,b)	min(a,b)
(a+b-ab)	(ab)
min(1,a+b)	max(0,a+b-1)
Resultado :	*Resultado :*
a si b=0	*a* si b=1
b si a=0	b si a=1
1 si resto	0 si resto

Tabla A.1. Ejemplos de Normas y Conormas.

La función clásica 'max' es la más restrictiva de todas las funciones que pueden representar la operación unión entre conjuntos borrosos. Lo mismo sucede con la función clásica 'min'.

$$u(a,b) \geq \max(a,b)$$

$$i(a,b) \leq \min(a,b)$$

Teniendo en cuenta esta última consideración se pueden seleccionar unas u otras operaciones en función de la aplicación que se desee implementar. La Teoría de Conjuntos Borrosos basada en los operadores originales se denomina usualmente Teoría de la Posibilidad.

A.1.4. Lógica Borrosa

La Lógica estudia los métodos y principios de razonamiento en todas las formas posibles. La Lógica Clásica maneja proposiciones que tienen que ser verdaderas o falsas. La Lógica Borrosa maneja proposiciones en las que los antecedentes y los consecuentes son conjuntos borrosos.

Hasta ahora se ha visto el concepto de conjunto borroso, y las operaciones que soporta. La base de la lógica borrosa es la construcción de proposiciones (sentencias condicionales) que relacionen conjuntos borrosos mediante operaciones lógicas. Estas proposiciones son conocidas como “reglas if-then”.

Supóngase una variable de entrada x cuyo universo de discusión es X, clasificado según los conjuntos borrosos Ax, Bx y Cx. Y una variable de salida y cuyo universo es Y, clasificada en los conjuntos borrosos Ay, By y Cy. Una regla establece una relación entre la variable de entrada y la de salida a través de operaciones entre conjuntos borrosos. Así, por ejemplo, una regla sencilla es :

if x is Ax then y is By

donde la parte “if x is Ax” se conoce como el antecedente de la regla y “then y is By” es el consecuente, conclusión o salida de la misma. Nótese que el antecedente en definitiva es la evaluación de la función de pertenencia del conjunto borroso Ax para el valor x, siendo por tanto un valor comprendido entre 0 y 1, mientras que la consecuencia de la regla es la asignación a la variable y de un conjunto borroso completo, en este caso By, afectado de alguna forma por el valor del antecedente, como se verá posteriormente.

El antecedente de una regla puede constar de múltiples partes relacionadas mediante operadores lógicos :

if x is Ax and z is Cz then y is By

En este caso, cada una de las partes del antecedente se calcula por separado, y a continuación se resuelven las operaciones lógicas entre dichas partes, obteniéndose de nuevo un único número comprendido entre 0 y 1.

En definitiva, la interpretación de una regla borrosa se realiza en varias etapas: en primer lugar ha de evaluarse el antecedente (lo cual implica la borrosificación de las entradas - evaluación de su pertenencia al conjunto indicado - y la aplicación de los operadores borrosos que relacionan las diferentes partes del antecedente) y en segundo lugar se aplica el resultado del antecedente al consecuente (proceso conocido como implicación). Existen varios métodos de implicación. Por ejemplo, puede

escalarse el conjunto borroso del consecuente en función del resultado del antecedente, o bien puede producirse un truncamiento del mismo.

A.1.5. Sistemas Borrosos

Un sistema borroso es aquel que relaciona una serie de entradas con una serie de salidas mediante la aplicación de lógica borrosa. Este tipo de sistemas han sido aplicados con éxito en campos tan diversos como el control automático, clasificación de datos, toma de decisiones, sistemas expertos, y visión artificial.

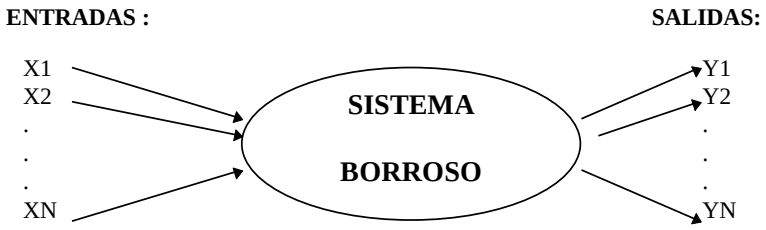


Figura A.2. Concepto de sistema borroso.

Como sucede para cualquier sistema, en el diseño de un sistema borroso hay que distinguir dos partes: una primera en la que se determinan los parámetros que intervienen en el proceso, y una segunda que consiste en la ejecución del proceso que obtiene las salidas a partir de las entradas mediante la aplicación de una serie de algoritmos que dependen de los parámetros anteriores. A la primera etapa se le conoce como "Conocimiento del sistema", y a la segunda "Estructura de procesamiento".

A.1.6. Organización del conocimiento del sistema

En esta primera etapa, el diseñador debe determinar:

- Cuales son las variables de entrada y de salida del sistema, así como su rango de variación. En definitiva, establece qué variables intervienen y cuál es su universo de discusión.
- El número de conjuntos borrosos necesarios para clasificar cada una de las variables anteriores dentro de su universo de discusión. A cada conjunto borroso debe

asignársele una variable lingüística que lo identifique y una función de pertenencia (definiendo en este último caso el soporte y centro de la misma).

- La base de reglas que regirán el proceso. Estas reglas se basan en el conocimiento del diseñador.

Supóngase, como ejemplo, que se dispone de la siguiente base de conocimiento sobre un sistema borroso:

- El sistema tiene dos entradas x y z, y una única salida y.
- Para cada variable se definen cuatro conjuntos borrosos con funciones de pertenencia triangulares, tal y como se muestra en la siguiente figura.

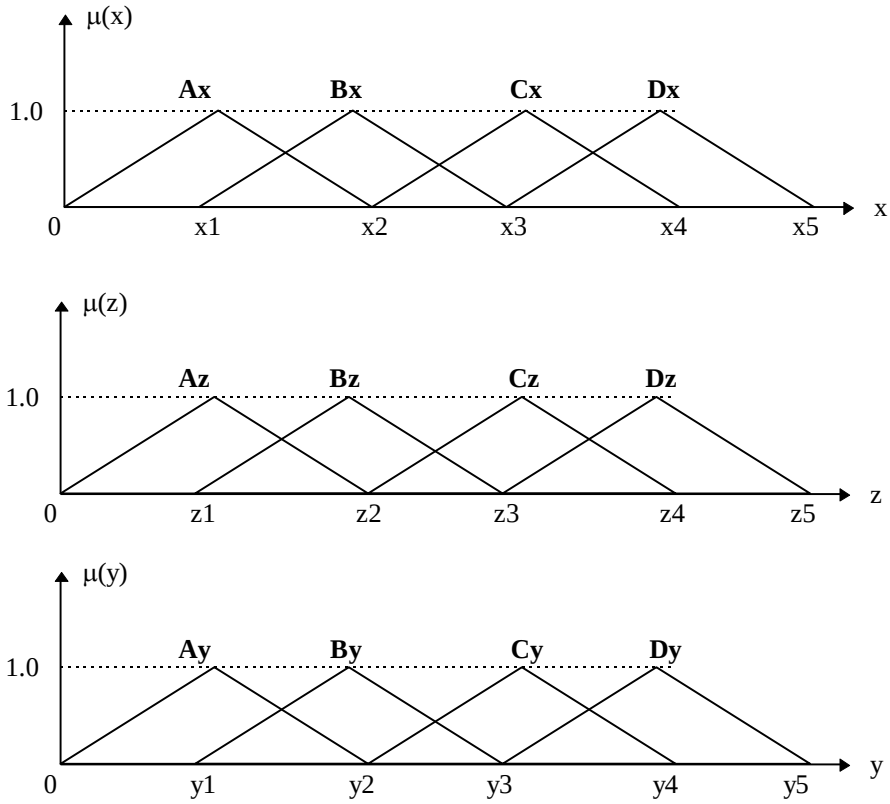


Figura A.3. Funciones de pertenencia del ejemplo.

- Por último, el sistema borroso queda definido por las dos siguientes reglas :

If x is Bx and z is Az then y is Dy

If x is Cx or z is Bz then y is Cy

A.1.7. Estructura de procesamiento

Definida la base del conocimiento, los pasos en los que se ejecuta un sistema borroso son los siguientes:

1. *Borrosificación de las entradas*. Consiste en la evaluación de la pertenencia de cada una de las variables de entrada a los diferentes conjuntos asociados a su universo de discusión. Este primer paso, por tanto, se reduce a la evaluación de la función de pertenencia de cada conjunto borroso para el valor que presenta la entrada en ese momento.

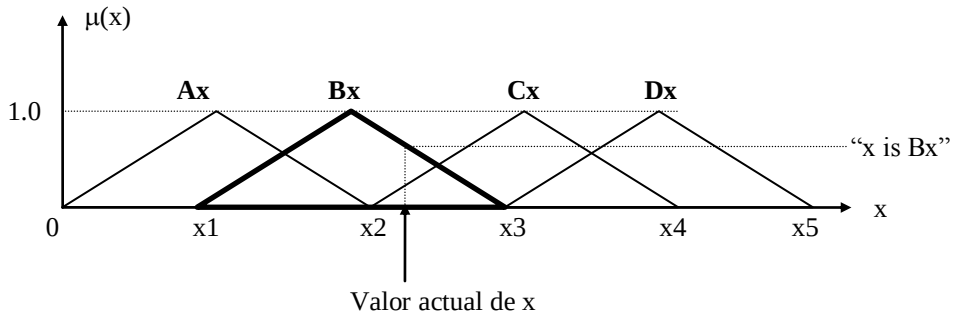


Figura A.4. Ejemplo de borrosificación: evaluación de "x is Bx".

2. *Cálculo de los antecedentes de las reglas* mediante la aplicación de los operadores borrosos que aparezcan en las mismas. Para cada regla, debe obtenerse como antecedente un número real comprendido entre 0 y 1.

   Así, por ejemplo para la primera regla, el antecedente es : *If x is Bx and z is Az,* por lo que debe hacerse la operación and (mínimo u otra T_Norma) de los valores "x is Bx" y "z is Az" obtenidos en el apartado anterior. En el caso de la segunda regla, debe realizarse una operación or (máximo u otra T_Conorma).

3. *Cálculo de las salidas de las reglas*. En este punto existen diferentes métodos. Dependiendo de cuales se apliquen, puede hablarse de dos tipos de sistemas :

   - SISTEMAS TIPO MAMDANI. La salida de cada regla es un conjunto borroso, afectado de alguna manera por el valor del antecedente (esto se conoce como método de implicación, y los más usuales son el truncamiento o escalado de su función de pertenencia). Así, para el ejemplo planteado, y utilizando como método de implicación el "Sup-min" (truncamiento), se obtiene para cada regla :

Regla 1 : *If x is Bx and z is Az then y is Dy*

Regla 2 : *If x is Cx or z is Bz then y is Cy*

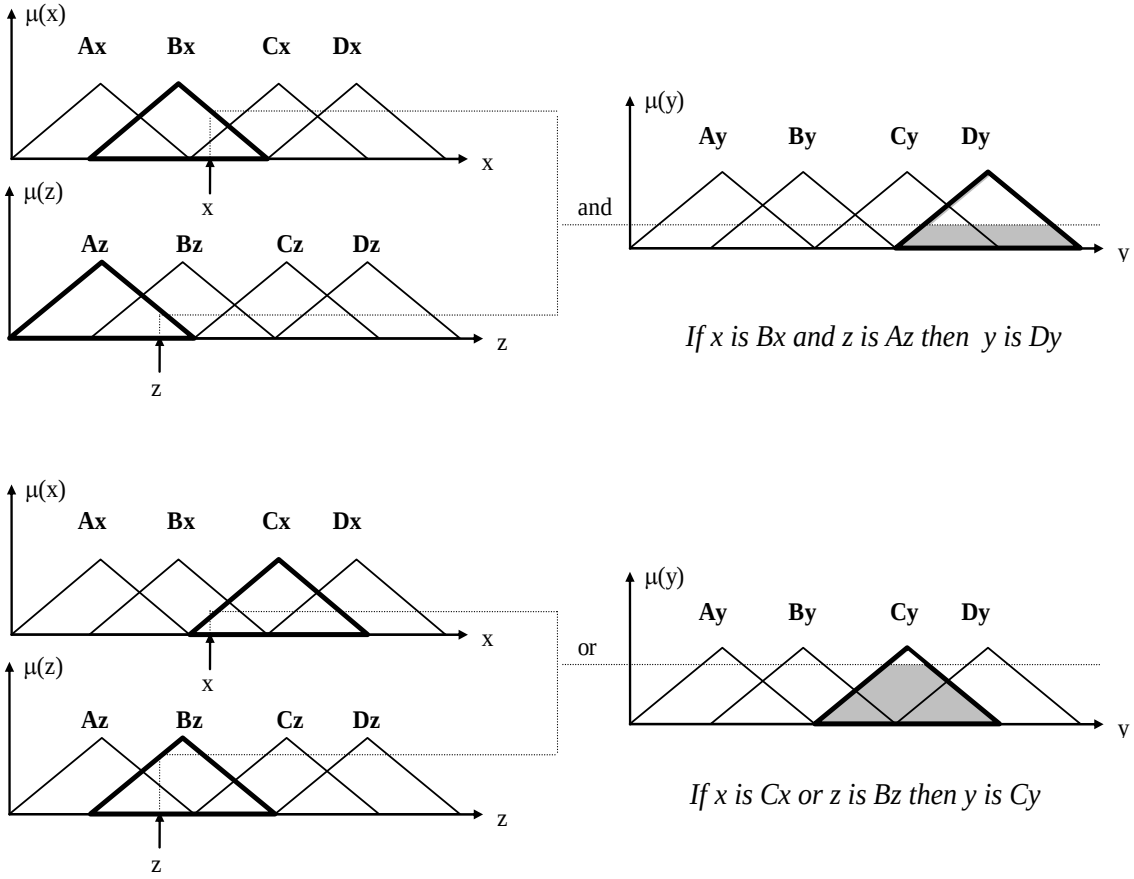
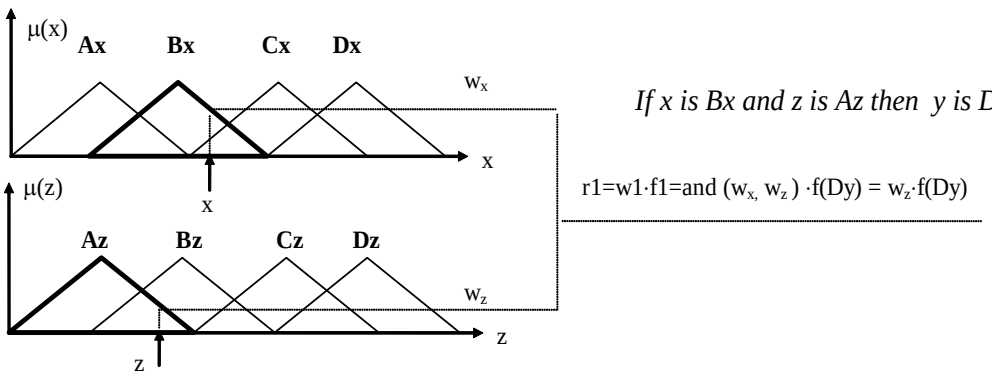


Figura A.5. Evaluación de las reglas según el método Mamdani.

- SISTEMAS TIPO SUGENO. El resultado r_i de cada regla es un valor numérico exacto, resultado de multiplicar el antecedente w_i por una función del consecuente.



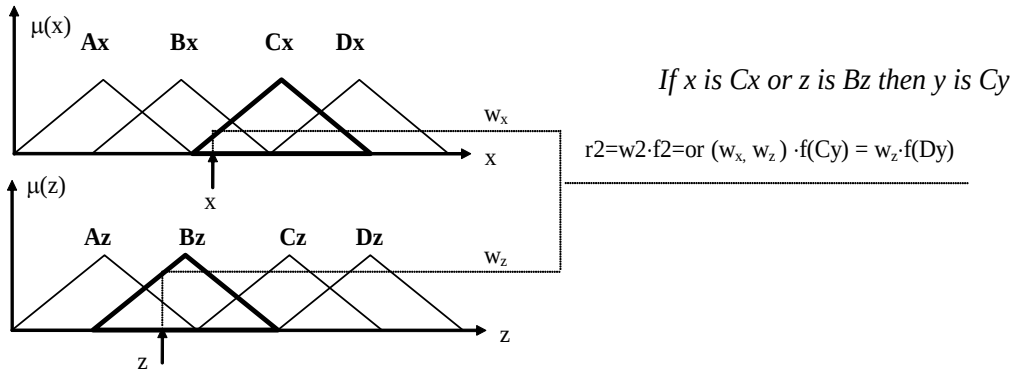


Figura A.6. Evaluación de las reglas según el método Sugeno.

Esta función puede ser, en el caso más sencillo, una constante que se suele hacer coincidir con el centro de la función de pertenencia del conjunto borroso que aparece en el consecuente de la regla a evaluar.

4. *Agregación de todas las salidas.* La agregación de las salidas de todas las reglas también difiere según se utilice el método Mamdani o el Sugeno.

- EN SISTEMAS TIPO MAMDANI se agregan todos los conjuntos borrosos resultado de cada regla en uno sólo mediante alguna operación de agregación (por lo general una T-norma). Por tanto, el resultado de la agregación es un nuevo conjunto borroso.

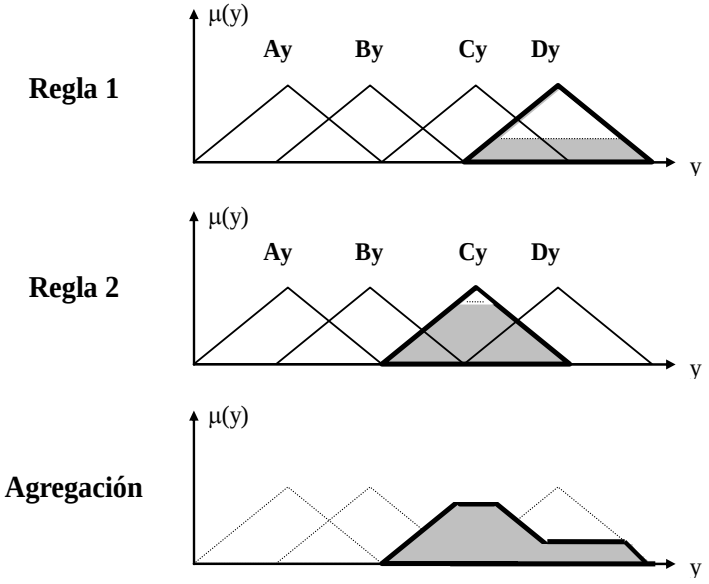


Figura A.7. Agregación de salidas para sistemas tipo Mamdani.

- EN SISTEMAS TIPO SUGENO se hace una suma ponderada de las salidas de cada regla, del siguiente modo:

$$y = \frac{\sum_{i=1}^{n} r_i}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n} w_i \cdot f_i}{\sum_{i=1}^{n} w_i}$$

siendo y el resultado de la agregación de todas las reglas (valor numérico), r_i el resultado de cada regla, w_i el valor del antecedente de la regla i, n el número de reglas y f_i la función escogida para el consecuente de cada regla.

5. *Desborrosificación de las salidas.* Sólo es necesario realizar este paso en sistemas tipo Mamdani, en los que aún queda obtener una salida numérica a partir del conjunto borroso obtenido como resultado de la agregación. Existen diversos métodos, pero los más usados son el cálculo del centro de gravedad del conjunto borroso, el cálculo del bisector de su área, o el cálculo de la media de sus valores máximos.