

UNA PROPUESTA DE NAVEGACIÓN DELIBERATIVA EN ROBOTS MÓVILES USANDO UN SISTEMA DIFUSO

Cristian David Vargas Valencia

**UNIVERSIDAD DEL VALLE ESCUELA DE INGENIERÍA ELÉCTRICA Y
ELECTRÓNICA INGENIERÍA ELECTRÓNICA GRUPO DE PERCEPCIÓN Y
SISTEMAS INTELIGENTES -PSI-
SANTIAGO DE CALI, JULIO DE 2011**

UNA PROPUESTA DE NAVEGACIÓN DELIBERATIVA EN ROBOTS MÓVILES USANDO UN SISTEMA DIFUSO

Cristian David Vargas Valencia

Tesis para optar por el título de Ingeniero Electrónico

Directores:

Eduardo Caicedo Bravo – Ph.D.

Bayron Andrés Calvache – M.Sc.

**UNIVERSIDAD DEL VALLE ESCUELA DE INGENIERÍA ELÉCTRICA Y
ELECTRÓNICA INGENIERÍA ELECTRÓNICA GRUPO DE PERCEPCIÓN Y
SISTEMAS INTELIGENTES -PSI-
SANTIAGO DE CALI, JULIO DE 2011**

NOTA DE APROBACIÓN

El trabajo de grado titulado “**Una propuesta de navegación deliberativa en robots móviles usando un sistema difuso**”, presentado por el estudiante CRISTIAN DAVID VARGAS VALENCIA, para optar por el título de ingeniero electrónico, fue revisado por el jurado y calificado como:

Ing. Eduardo Caicedo Bravo Ph.D.
Director

Ing. Bayron Calvache M.Sc.
Director

Jurado

Jurado

RESUMEN

En este documento se presenta el diseño e implementación de una arquitectura híbrida deliberativa usando un sistema difuso, basada en la arquitectura automática deliberativa. Este trabajo está dividido en dos bloques funcionales: una capa automática que se encarga de las respuestas reactivas y una capa deliberativa que traza los planes y medidas correctivas para alcanzar un objetivo dado. La arquitectura usada es modular, lo que permite que sea posible incrementar la funcionalidad del sistema en el futuro. Se presenta también una biblioteca para la implementación de sistemas difusos, desarrollada para el proyecto.

Como resultado se entrega una aplicación orientada a la navegación, con capacidades de localización, planeación y seguimiento de trayectorias, la cual al estar construida con base en bloques funcionales configurables, puede ser usada como una herramienta didáctica para la experimentación en robótica móvil.

Palabras clave

- Arquitectura Automática Deliberativa, navegación, sistema difuso, robótica móvil.

ABSTRACT

This document presents the design and implementation of a hybrid deliberative architecture using a fuzzy system. It is based on the automatic deliberative architecture. This system is divided into two main functional blocks: the automatic layer is in charge of reactive actions and the deliberative layer creates the plans and corrective actions in order to reach a given objective. The architecture is modular, what allows future enhancements to the system. The library developed and used to implement the fuzzy systems in this project is presented too.

The final system is a navigation oriented application, with several capabilities like localization and path planning. Most of the blocks in the system are configurable, what allows the application to be used as a didactic tool for experimentation in mobile robotics.

Keywords

- Automatic deliberative architecture, navigation, fuzzy system, mobile robotics.

TABLA DE CONTENIDO

Introducción	1
Objetivos	2
Esquema de solución	2
Estructura del documento	3
Capítulo 1: Navegación reactiva-deliberativa y lógica difusa	
aplicadas a robótica móvil	4
1.1. Navegación reactiva - deliberativa	5
1.1.1. Arquitecturas de control reactivo.....	5
1.1.1.1. Coordinadores competitivos	6
1.1.1.2. Coordinadores cooperativos	6
1.1.2. Arquitecturas de control deliberativo	7
1.1.2.1. Mapeo	7
1.1.2.2. Localización.....	8
1.1.2.3. Planeación de trayectorias	9
1.2. Arquitectura automática deliberativa	9
1.2.1. Nivel automático, habilidades automáticas y acciones reflejas	10
1.2.2. Nivel deliberativo y habilidades deliberativas.....	11
1.2.3. Memoria a corto plazo	11
1.2.4. Memoria a largo plazo	11
1.3. Lógica difusa	11
1.4. Plataforma móvil Pioneer3DX	12
1.4.1. Control de la plataforma	13
1.4.1.1. ARIA.....	14
1.4.1.2. ArNetworking	15
1.4.1.3. ARNL.....	15
1.4.1.4. Cliente para la visualización del estado de la plataforma MobileEyes.....	16
1.5. Laboratorio remoto ROBWEB UV	16
1.6. Estado del arte.....	18
1.7. Conclusiones	20
Capítulo 2: Sistema de navegación reactiva basado en lógica difusa	22
2.1. Estructura del coordinador reactivo.....	23

2.1.1. Clase ArActionDesired	23
2.1.2. Clase ArAction	23
2.1.3. Clase ArRobot.....	24
2.1.4. Clase ArResolver	24
2.1.5. Estructura del coordinador reactivo	25
2.1.5.1. Coordinador por prioridad	26
2.1.5.2. Coordinador difuso.....	27
2.2. Comportamientos reactivos	31
2.2.1. Habilidad evitar obstáculos.....	31
2.2.2. Habilidad velocidad constante	32
2.2.3. Habilidad seguir trayectorias	32
2.3. Biblioteca para la implementación de sistemas difusos.....	34
2.4. Conclusiones	42
Capítulo 3: Sistema de navegación híbrido basado en la	
arquitectura automática-deliberativa.....	43
3.1. Integración de las capas reactiva y deliberativa	44
3.2. Capa deliberativa.....	46
3.2.1. Localización.....	46
3.2.2. Planeación de trayectorias.....	47
3.3. Interfaz para el control remoto de la plataforma y MobileEyes	50
3.4. Conclusiones	53
Capítulo 4 Pruebas y resultados.....	55
4.1. Validación del sistema de navegación	
deliberativa basado en lógica difusa	56
4.1.1. Localización.....	56
4.1.2. Seguimiento de trayectoria	61
4.1.3. Costo computacional	66
4.2. Biblioteca para la implementación de sistemas difusos.....	68
4.3. Conclusiones	70
Capítulo 5: Conclusiones, observaciones generales y aportes futuros	72
5.1. Conclusiones y observaciones.....	72
5.2. Aportes	72
5.3. Trabajos futuros.....	73
Bibliografía.....	74

LISTA DE FIGURAS

Figura 1: Diagrama de bloques del esquema de solución propuesto	3
Figura 2: Clasificación de las arquitecturas de control en robótica móvil.....	5
Figura 3: Esquema de las arquitecturas de control reactivas	5
Figura 4: Coordinador competitivo	6
Figura 5: Coordinador cooperativo.....	7
Figura 6: Ejemplo de celdas de ocupación.....	9
Figura 7: Arquitectura automática deliberativa [Barber00].....	10
Figura 8: Funciones de posición binaria (izquierda) y difusa (derecha)	11
Figura 9: Robot móvil Pioneer3DX.....	13
Figura 10: Estructura de ARIA [ARIA11].	14
Figura 11: Estructura de ArNetworking[ARIA11].	15
Figura 12:Arquitectura de la interfaz de experimentación remota Robweb UV	17
Figura 13: Estructura del coordinador reactivo.....	26
Figura 14: Coordinador por prioridad	26
Figura 15: Coordinador difuso controlando dos variables	27
Figura 16: Conjuntos difusos del coordinador por defecto.	28
Figura 17: Algoritmo para la habilidad de seguir trayectoria.....	33
Figura 18: Seguimiento de trayectoria	34
Figura 19: Ejemplo de conjunto difuso	34
Figura 20: Diagrama de clases de la biblioteca.....	35
Figura 21: Atributos de un rectángulo	39
Figura 22: Metodología para la utilización de la biblioteca	41
Figura 23: Arquitectura automática deliberativa aplicada	44
Figura 24: Planeación de una trayectoria.....	48
Figura 25: Obstáculo virtual	48
Figura 26: Configuración de las habilidades evadir obstáculo.....	49
Figura 27: Robot siguiendo trayectoria	50
Figura 28: Interfaz para el control remoto de la plataforma	50
Figura 29: Resultados arrojados por MobileEyes.....	53
Figura 30: Robot no localizado	57
Figura 31: Robot localizado (una puntuación mayor a 0.6)	57
Figura 32: Posiciones usadas para las pruebas realizadas.....	58
Figura 33: Localización errada con alta puntuación	59
Figura 34: Obstáculos usados en la prueba.....	61
Figura 35: Seguimiento de trayectoria para las pruebas	61
Figura 36: Error trayectoria recorrida – trayectoria planeada para el experimento de ..	63
Figura 37: Aceleración durante el recorrido	64
Figura 38: giro de 180° usando un coordinador por prioridad.....	65
Figura 39: Giro de 180° en pasillo usando coordinador difuso	65
Figura 40: Costo computacional del sistema de navegación difuso	66
Figura 41: Costo computacional del sistema de navegación por prioridad	67
Figura 42: Costo computacional del sistema de navegación cooperativo.....	67
Figura 43: Sistema difuso en el toolbox de lógica difusa de MATLAB.....	69

LISTA DE TABLAS

Tabla 1: Analogía entre las operaciones AND, OR y NOT en lógica binaria y difusa ..	12
Tabla 2: Parámetros de configuración para los algoritmos de localización y	16
Tabla 3: Trabajos realizados en el área en el área de navegación deliberativa.....	20
Tabla 4: Métodos y atributos de la clase Funciones.....	36
Tabla 5: Métodos y atributos de la clase ConjuntoDifuso	37
Tabla 6: Métodos y atributos de la clase Reglas	39
Tabla 7: Métodos y atributos de la clase FuzzyLogic	41
Tabla 8: Resultados pruebas localización	58
Tabla 9: Parámetros de configuración usados para las pruebas.....	62
Tabla 10: Resultados	63
Tabla 11: Validación biblioteca para la implementación de sistemas difusos	69

INTRODUCCIÓN

En robótica móvil siempre se busca que el robot tenga la mejor representación y percepción posible del ambiente, tratando de lograr navegación de una forma autónoma y eficiente. En la percepción del ambiente es indispensable la calidad de los sensores, pues a menores incertidumbres arrojadas por los mismos, mejor podría desenvolverse la plataforma. La navegación realizada por la plataforma depende principalmente de la arquitectura de control utilizada. Estas arquitecturas son generalmente bioinspiradas, intentando imitar desde comportamientos sencillos como el de una hormiga hasta algunos bastante complejos como los de un humano, requiriendo cada vez mayor capacidad computacional.

En el documento se presenta una introducción a las arquitecturas de control en robótica móvil, profundizando en la arquitectura automática deliberativa [Barber00], que busca emular la capacidad de decisión que poseen los seres humanos. Se muestra la implementación de coordinadores que hacen uso de sistemas difusos, su configuración y su adaptación a esta arquitectura.

Se realiza una introducción a las herramientas usadas para la implementación de la arquitectura, tales como la biblioteca para la implementación de sistemas difusos y el software para el control de la plataforma Pioneer3DX utilizada en este proyecto.

OBJETIVOS

Este proyecto propone un esquema de navegación basado en la arquitectura de control automática deliberativa que hace uso de un sistema difuso, buscando que las funcionalidades de la implementación sean fáciles de extender y que a la vez se conserven los niveles de complejidad en niveles razonables.

Se tiene entonces como objetivo general proponer e implementar un sistema de navegación deliberativo para un robot Pioneer3DX basado en habilidades y usando técnicas difusas.

Los objetivos específicos que se pretenden alcanzar son:

- Obtener un estado del arte sobre navegación deliberativa en robótica móvil.
- Apropiar la arquitectura de control usada en el sistema de habilidades del laboratorio remoto de robótica móvil de la Universidad del Valle.
- Diseño e implementación de un coordinador de habilidades reactivas usando técnicas difusas.
- Diseño e implementación de un coordinador difuso para la capa de navegación deliberativa.
- Integrar los coordinadores de las capas reactiva y deliberativa y evaluar su desempeño.

ESQUEMA DE SOLUCIÓN

Se desarrolla el proyecto actual basándose en la arquitectura de control automática deliberativa [Barber00] y en el software desarrollado para el laboratorio remoto RobWeb-UV [Calvache09]. La plataforma utilizada para el desarrollo del proyecto fue el robot Pioneer3DX que hace uso del API ARIA provisto por el fabricante del robot (MobileRobotsinc) [ARIA11].

En la figura 1 se presenta un diagrama del esquema de solución propuesto. La plataforma se controla usando un cliente remoto desde el cual se le puede indicar al robot la tarea a desarrollar. Otro cliente remoto permite monitorear la plataforma.

La capa deliberativa se encarga de trazar los planes que se requieran, de mantener actualizada la posición de la plataforma y de configurar el coordinador difuso. Este coordinador a su vez ejecuta los movimientos requeridos para alcanzar el objetivo, fusionando las diferentes habilidades disponibles.

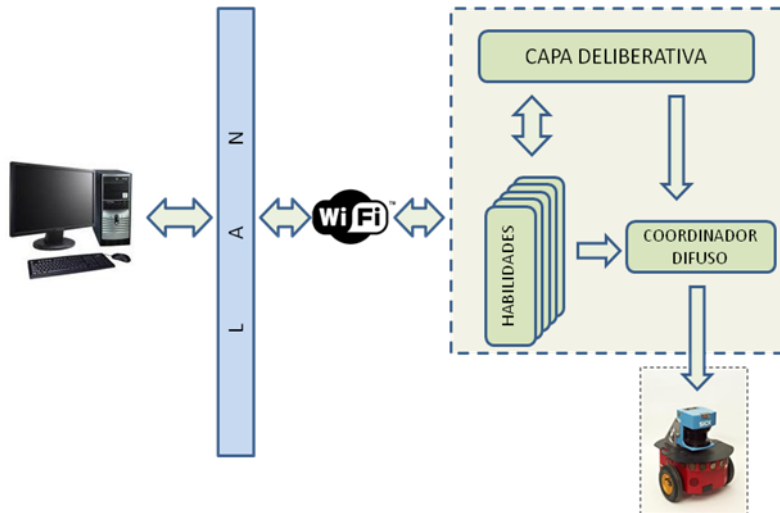


Figura 1: Diagrama de bloques del esquema de solución propuesto

ESTRUCTURA DEL DOCUMENTO

El presente documento se encuentra dividido en 6 capítulos distribuidos de la siguiente manera:

En el primer capítulo se presenta una introducción a las arquitecturas de control en robótica móvil, la API para el control de la plataforma Pioneer3DX, lógica difusa y un estado del arte en el área de navegación deliberativa.

En el segundo capítulo se presenta la implementación del coordinador de habilidades reactivas basado en técnicas difusas y en la arquitectura de control automática deliberativa así como de la biblioteca usada para este fin.

En el tercer capítulo se presenta la implementación de la capa deliberativa y la integración al sistema para conformar la arquitectura de control híbrida deliberativa que se desea.

En el último capítulo se presenta el análisis de resultados obtenidos en este proyecto. Se realizan comparaciones con un esquema que no usa el sistema difuso.

Al final del documento se presentan las conclusiones del proyecto y trabajos futuros.

CAPITULO 1

NAVEGACIÓN REACTIVA – DELIBERATIVA Y LÓGICA DIFUSA APLICADAS A LA ROBÓTICA MÓVIL

*La inteligencia es...la facultad de hacer objetos artificiales,
especialmente herramientas para hacer herramientas*
Henri Bergson

*Intenta hasta el final y nunca dudes,
nada es tan difícil que no pueda averiguarse mediante la búsqueda*
Robert Herrick

En robótica móvil se han propuesto diversas arquitecturas de control. Algunas proponen una navegación sin ningún tipo de planeación, solo actúan de acuerdo con lo que la plataforma percibe por medio de sus sensores en un instante dado. También hay propuestas que plantean realizar una planeación haciendo uso de conocimiento a priori del entorno, como por ejemplo un mapa, para alcanzar algún objetivo. Cada aproximación tiene sus ventajas y desventajas, de donde nacen las arquitecturas

híbridas, que tratan de aprovechar las ventajas de cada una a la vez que se cubren sus deficiencias.

En este capítulo se presenta una descripción general de las arquitecturas reactivas, deliberativas e híbridas, haciendo énfasis en la arquitectura de control automática deliberativa [Barber00], la cual es usada en este proyecto. También se describe la plataforma Pioneer3DX y cada una de las APIs que se usa para controlarla. De la misma forma se presenta la lógica difusa, una herramienta útil para la implementación de coordinadores reactivos.

1.1.NAVEGACION REACTIVA - DELIBERATIVA

Las arquitecturas de control usadas en robótica móvil son diversas y tienen diferentes enfoques, fortalezas y debilidades. A un nivel general las arquitecturas se pueden clasificar en reactivas, deliberativas e híbrida [Arkin98] tal como se muestra en la figura 2.

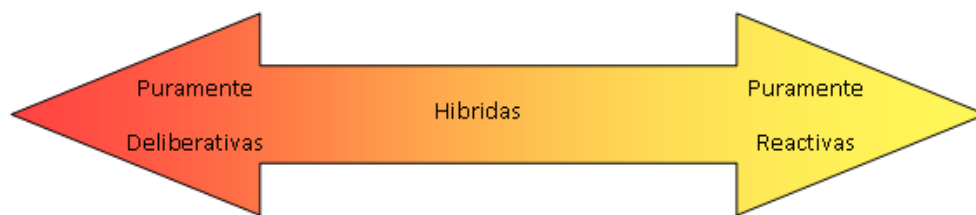


Figura 2: Clasificación de las arquitecturas de control en robótica móvil

1.1.1. Arquitecturas de control reactivo

En los inicios de la robótica se planteó un esquema de control bastante simple, donde la acción que el robot realiza esta directamente influenciada por la lectura actualizada de sus sensores (figura 3).



Figura 3: Esquema de las arquitecturas de control reactivas

Este tipo de arquitectura no presenta una carga computacional importante debido a su simplicidad, lo que hace que su velocidad de respuesta sea alta. Sin embargo las tareas que puede realizar son limitadas en complejidad debido a que no se trazan planes de ningún tipo [Murphy00].

Cuando se usan este tipo de arquitecturas se desea que la plataforma pueda responder adecuadamente dependiendo de la situación, por lo cual en lugar de usar un gran comportamiento para controlarla se usan varios más simples que,

dependiendo de la situación están activos o inactivos, y aportan una salida deseada que un bloque funcional llamado comúnmente coordinador reactivo se encarga de procesar. El coordinador provee una mayor modularidad, permitiendo extender fácilmente la funcionalidad de este tipo de arquitecturas.

Cada una de estas acciones simples que se le entregan al coordinador se conoce como habilidades o comportamientos, dependiendo del contexto.

Dado que estos coordinadores procesan la salida de diversas habilidades se han planteado también diferentes esquemas para hacerlo. Los coordinadores reactivos se pueden clasificar entonces en competitivos y cooperativos [Arkin98].

1.1.1.1. Coordinadores competitivos

Los coordinadores competitivos seleccionan solo una de las habilidades activas dependiendo de algún criterio, normalmente una prioridad, dándole el control total de la plataforma (figura 4).

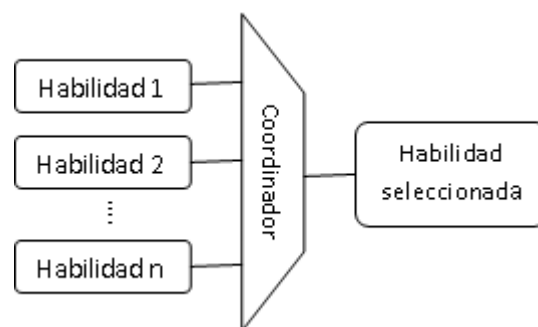


Figura 4: Coordinador competitivo

Estos coordinadores tienen la ventaja de que al darle el control de la plataforma a una habilidad se suprimen las demás, haciéndolos bastante útiles cuando se requiere coordinar habilidades que tratan de preservar la integridad de la plataforma, tales como detenerse o evadir obstáculos que se encuentran demasiado cerca. Otra ventaja de estos coordinadores es su fácil implementación y la poca carga computacional que representan.

La principal desventaja de este tipo de coordinador se presenta cuando se hace la transición de una habilidad a otra, la cual puede ser algo brusca. Físicamente esto trae consigo cambios abruptos de velocidad u orientación, lo cual es indeseado.

1.1.1.2. Coordinadores cooperativos

Cada habilidad activa arroja una salida deseada. Los coordinadores cooperativos procesan cada una de estas salidas y a partir de ellas se calcula una salida total (figura 4). La salida total calculada depende no solo de las habilidades, sino también

de la implementación del coordinador, ya que puede realizarse de diferentes formas (sumas ponderadas, lógica difusa, etc). Generalmente para darle más importancia a ciertas habilidades sobre otras se les asocia un peso.

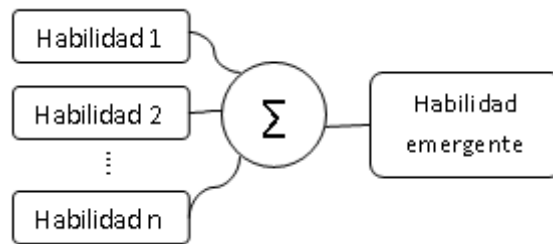


Figura 5: Coordinador cooperativo

Al no haber transiciones bruscas entre comportamientos, en este tipo de coordinador se generan recorridos más suaves que pueden ser mas óptimos comparados con un los obtenidos con un coordinador competitivo. Al ser necesario procesar más de una de las salidas de las habilidades de forma simultánea se incrementa la carga computacional del sistema.

1.1.2. Arquitecturas de control deliberativo

Las arquitecturas de control deliberativo pueden llevar a cabo tareas de más alto nivel que las reactivas, pero requieren modelos del ambiente, información actualizada de la posición del robot y otros datos que sólo se pueden obtener mediante mejores sensores y mayor capacidad de computo, en consecuencia solo se pueden implementar en plataformas robustas.

Este tipo de arquitecturas básicamente traza un plan con base en la información dada a priori, como pueden ser un mapa, una posición inicial y/o una tarea a realizar. Una vez trazado este plan se procede a ejecutarlo. Durante la ejecución del plan se puede presentar un obstáculo que no estaba previsto, evento que hace necesario trazar un nuevo plan, operación que puede tomar un tiempo considerable.

Las arquitecturas deliberativas están generalmente compuestas por bloques como localización, mapeo y planeación de trayectorias que en la mayoría de casos generan una alta carga computacional. Una descripción general de estos bloques se presenta a continuación.

1.1.2.1. Mapeo

Un robot que use una arquitectura deliberativa debe estar en capacidad de procesar mapas. Generalmente se habla de dos tipos de mapas, topológicos y geométricos.

Los mapas topológicos son muy generales, entran muy poco a detallar el entorno y generalmente la escala no es constante en todo el mapa. Este tipo de mapa funciona

al localizar en él un conjunto de puntos de interés o marcas, que sirven como punto de referencia para localizarse. Generalmente no ocupan mucha memoria pero requieren que se introduzcan en el ambiente o algoritmos de visión artificial algo complejos para poder identificar estos puntos.

Los mapas geométricos tratan de generar un modelo detallado del ambiente, usando una escala constante a lo largo del mapa, por lo que su generación y mantenimiento implica un costo mayor en memoria que un mapa topológico. Entre las ventajas de estos mapas está que se puede interpretar más fácilmente por un humano que un mapa topológico, lo que facilita el monitoreo de las tareas en ejecución. Al ser un modelo bastante detallado del entorno permite extraer información para localizar el robot sin necesidad de introducir marcas.

Si se desea realizar un mapeo, se debe seguir también alguna estrategia de exploración, ya que si se navega de forma aleatoria por el ambiente la extracción de toda la información podría tardar demasiado tiempo, en el caso en el que se logre completar [Murphy00].

1.1.2.2. Localización

Una vez un robot posee el mapa de su entorno es necesario que se localice en él y que actualice su posición constantemente. Se puede presentar también el caso en el que se deba realizar mapeo y localización simultáneamente, cuando no hay un mapa del entorno dado a priori.

Es necesario implementar algoritmos de localización debido a que la precisión de la odometría es bastante baja y si se tiene una buena localización es posible trazar y ejecutar de forma más eficiente los planes.

Un método bastante usado para la localización es el método de Monte Carlo (MCL) [Dellaert99]. Básicamente se le da al robot una posición aleatoria en el mapa y basándose en un modelo de los sensores y el teorema de Bayes, con cada lectura de los sensores se calcula la probabilidad de que esa posición sea correcta. Este procedimiento se realiza para muchas posiciones aleatorias (muestras). Se considera como correcta la posición con la probabilidad más alta y si la probabilidad de alguna es demasiado baja, se reemplaza con una nueva posición aleatoria. Cada movimiento aplicado a la plataforma se debe ver reflejado en cada una de las muestras para mantener una estimación actualizada de la misma.

Para reducir el rango de estas configuraciones espaciales aleatorias se debe presentar en el mapa un área designada como casa o área de inicio, lugar donde se realiza la localización inicial del robot. Esto se hace con el fin de facilitar la tarea de localización, ya que hallar la posición del robot en cualquier lugar de un mapa sin datos a priori es un problema complejo, conocido como robot secuestrado. Una vez se ha realizado la localización inicial, se debe ejecutar el algoritmo para la corrección de los errores de odometría.

1.1.2.3. Planeación de trayectorias

Hay diversos métodos usados para planear trayectorias y se eligen dependiendo de los sensores y capacidad computacional que se posee el robot. Se requiere un mapa sobre el cual generar la trayectoria, así como puntos origen (posición actual del robot) y destino.

Uno de los métodos usados es el de las celdas de ocupación. Básicamente se divide todo el mapa en celdas de igual tamaño, luego para cada celda se revisa la presencia de obstáculos. Si hay obstáculos presentes se subdivide la celda y en caso de no haberlos se marca como navegable. Se itera en cada nueva celda hasta que su tamaño sea considerado lo suficientemente pequeño. En la figura 6 se presenta un esquema de descomposición por celdas aplicado sobre un área con presencia de obstáculos. Los obstáculos deben ser ensanchados antes de ejecutar el algoritmo, para tener en cuenta el radio del robot, evitando así posibles colisiones.

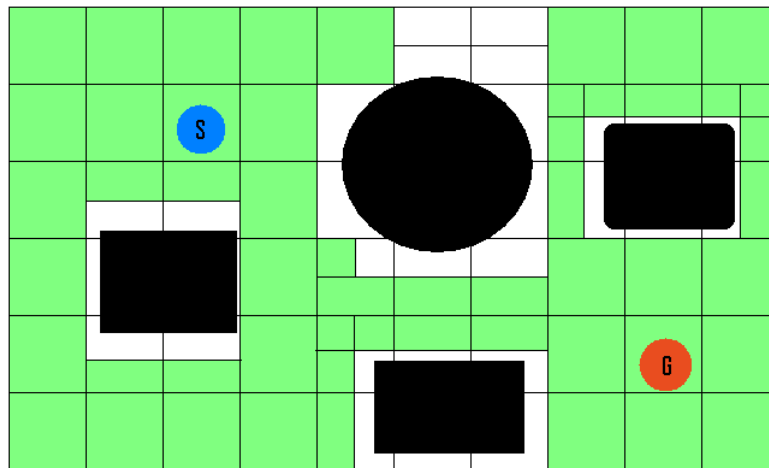


Figura 6: Ejemplo de celdas de ocupación

Una vez se ha dividido el mapa en zonas navegables y no navegables se ubican los puntos origen y destino y se traza una trayectoria dependiendo de los parámetros que se deseen optimizar (distancia, suavidad, etc) [Murphy00].

1.2. ARQUITECTURA DE CONTROL AUTOMATICA DELIBERATIVA

Tratando de aprovechar la velocidad de reacción de las arquitecturas reactivas y la ejecución de tareas complejas de las arquitecturas deliberativas se han planteado las arquitecturas híbridas. Entre estas arquitecturas planteadas se encuentra la automática deliberativa [Barber00], la cual se tomo como base para el desarrollo de este proyecto. Esta arquitectura se estructura a partir de la forma en la que razonan y actúan los seres humanos. El nombre proviene de los dos niveles de razonamiento propuestos: automático y deliberativo (figura 7).

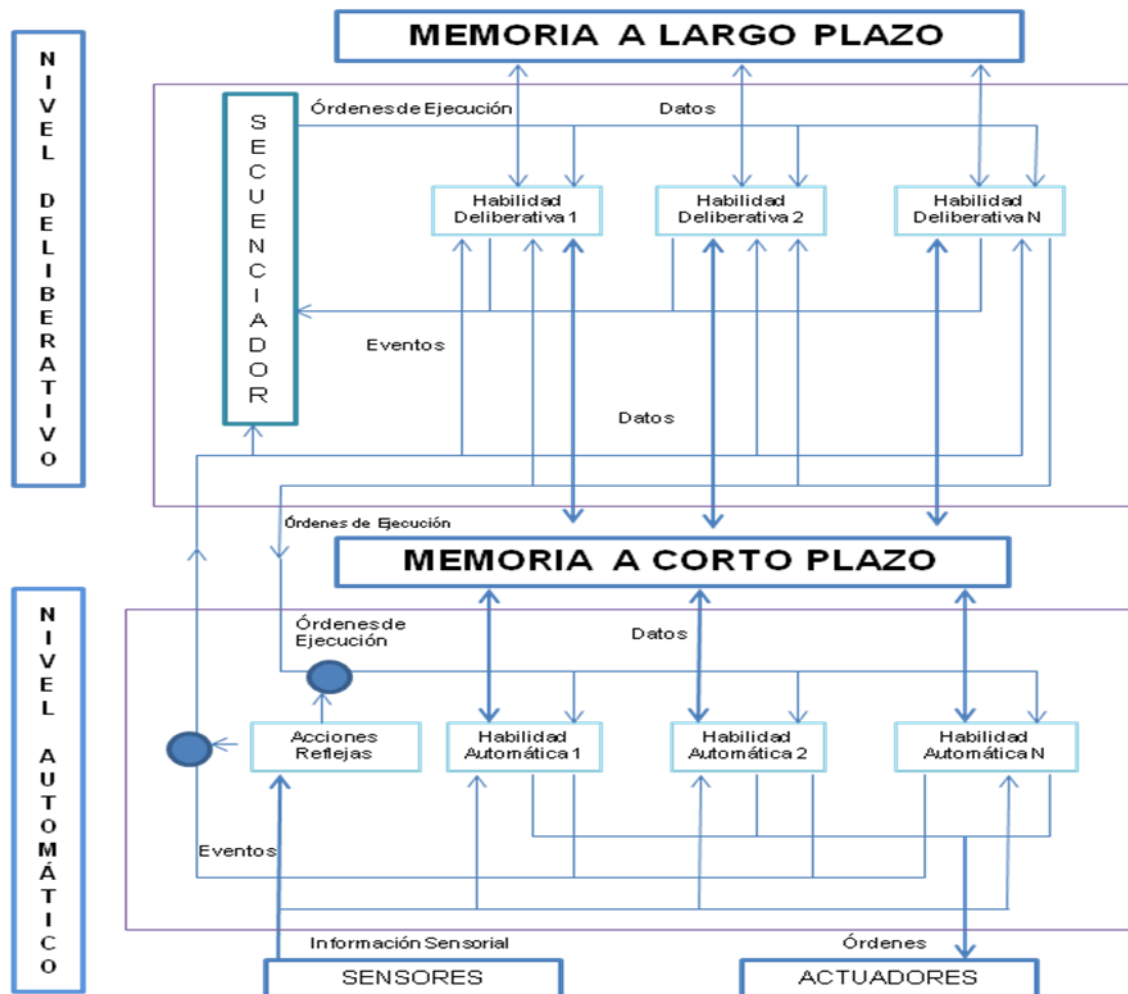


Figura 7: Arquitectura automática deliberativa [Barber00]

1.2.1. Nivel automático, habilidades automáticas y acciones reflejas

Los comportamientos reactivos se dividen en dos tipos: habilidades automáticas y acciones reflejas. Las habilidades automáticas son procesadas normalmente como se hacía en las arquitecturas reactivas, pero las acciones reflejas pueden inhibirlas en caso de que se requiera, como por ejemplo cuando está en riesgo la integridad de la plataforma.

Las habilidades automáticas y las acciones reflejas usadas en esta arquitectura son análogas a los comportamientos reactivos de las arquitecturas reactivas clásicas, con algunas diferencias como por ejemplo el manejo de eventos, uno de los métodos usados para la interacción con el nivel deliberativo. Algunas de estas habilidades están en capacidad de arrojar eventos para notificar cambios importantes durante la ejecución de una tarea como por ejemplo el caso la llegada al final de una trayectoria que se estaba siguiendo, donde la habilidad de seguimiento arroja un evento para notificar que ha concluido su tarea [Barber00].

1.2.2. Nivel deliberativo y habilidades deliberativas

El nivel deliberativo está compuesto por habilidades deliberativas y su respectivo secuenciador. Las habilidades deliberativas hacen referencias a los comportamientos de alto nivel, tales como localización y planeación de trayectorias. Estas habilidades controlan el movimiento de la plataforma indirectamente, modificando el peso y los parámetros de las habilidades automáticas en el coordinador de acuerdo con las acciones que desean realizar. El secuenciador de habilidades deliberativas es dado a priori y su configuración depende de la tarea que se desea realizar. Se encarga de secuenciar las diferentes habilidades deliberativas para llevar a cabo la tarea.

1.2.3. Memoria a corto plazo

En esta memoria se almacena la información relevante a corto plazo. Se almacena por ejemplo la información referente al estado del robot (velocidad, posición, orientación, etc). Por medio de esta memoria se comunican las habilidades deliberativas con las automáticas, pasando argumentos o actualizando el estado.

1.2.4. Memoria a largo plazo

Esta memoria almacena datos que no son muy dinámicos y generalmente contiene información de la tarea y el entorno. Se almacena por ejemplo el mapa del área de trabajo, el cual se requiere para realizar la localización y planeación de trayectorias.

1.3. LOGICA DIFUSA

La lógica difusa es uno de los campos más activos en inteligencia artificial y sistemas inteligentes. En la lógica binaria la respuesta a las proposiciones puede tener solo dos posibles valores: 0 o 1 (falso o verdadero) mientras que en la lógica difusa se puede tomar cualquier valor entre estos [Kaehler11].

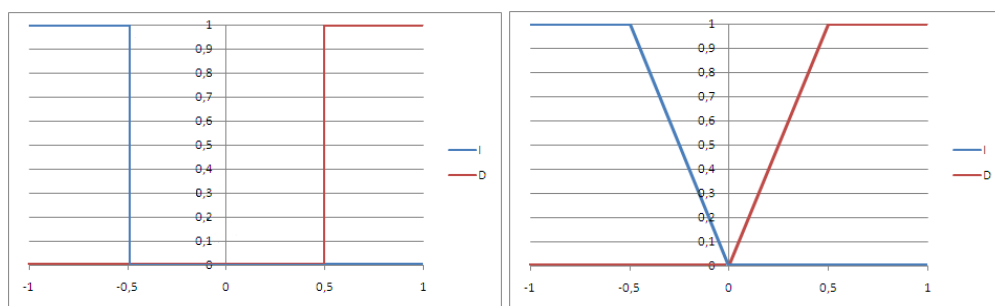


Figura 8: Funciones de posición binaria (izquierda) y difusa (derecha)

En la figura 8 se muestran un par de funciones que definen la posición de un objeto respecto a un punto de referencia. En una aproximación binaria se debe sobrepasar cierto umbral para que la función tome otro valor, mientras que en la aproximación difusa este valor va cambiando gradualmente. Las funciones que definen el grado de pertenencia se conocen como funciones de membresía y son asociadas en un conjunto difuso, los cuales se asignan tanto a las entradas como a las salidas.

Las proposiciones en lógica difusa vienen representadas por reglas lingüísticas, las cuales son proposiciones condicionales de la forma: *IF ConjuntoDifusoA IS FuncionMembresiaX AND/OR ConjuntoDifusoB IS FuncionMembresiaY AND/OR... THEN ConjuntoDifusoSalida is FuncionMembresiaSalidaZ*". Las reglas definen la relación de las funciones de membresía de la entrada con las de la salida. Cuando se asocia más de una función de membresía a la entrada con la salida se hace necesario aplicar operaciones que permitan la unión (AND/OR). Estas operaciones son análogas a las versiones usadas en lógica binaria tal como se puede ver en la tabla 1.

X	Y	$\min(X,Y)$	$\max(X,Y)$	$1 - X$	$1 - Y$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0
		AND	OR	NOT(X)	NOT(Y)

Tabla 1: Analogía entre las operaciones AND, OR y NOT en lógica binaria y difusa

A la salida del sistema se tiene también un conjunto difuso, por lo que se debe aplicar un método de resolución (defuzzyfication) para convertirlo a una salida numérica. Uno de los métodos más usados para esto es el del centroide de masa, que calcula el centroide del sistema a partir del centroide de cada componente del conjunto.

1.4. PLATAFORMA MÓVIL PIONEER3DX

La implementación de la arquitectura de control automática deliberativa se realizó usando las bibliotecas desarrolladas por el fabricante de la plataforma móvil, y que son portables entre robots de la misma familia. Se realiza una breve descripción tanto de la plataforma Pioneer3DX así como de las bibliotecas usadas.

El robot móvil Pioneer3DX es una plataforma de tipo diferencial que posee diversos sensores que posibilitan su navegación [ARIA11]. Una plataforma de este tipo se presenta en la figura 9.



Figura 9: Robot móvil Pioneer3DX

Uno de los sensores de distancia es un anillo de 8 ultrasonidos que permite medir la proximidad a los objetos que se encuentren entre 10cm y 4m. Estos se ubican en la parte frontal de la plataforma cubriendo ángulos desde -90° hasta 90° respecto a su orientación y calculan la distancia a los objetos emitiendo sonido a altas frecuencias y midiendo el tiempo de vuelo. Los sensores de ultrasonido tienen un ángulo de apertura bastante amplio, alrededor de 15° , lo que hace que la ubicación real del objeto tenga un error que se debe tener en cuenta a la hora de navegar con el robot. Las principales ventajas de estos sensores son el bajo costo y la baja carga computacional que representan.

Otros sensores con los que cuenta la plataforma son los encoders, los cuales se ubican en los dos motores y permiten determinar la distancia recorrida por el robot. Las lecturas de los encoders se utilizan para posicionar el robot, sin embargo son una fuente de error considerable. Por ejemplo si una llanta se resbala, está un poco desinflada respecto a la otra o sube por un bache las lecturas arrojadas por los codificadores para actualizar la posición serán incorrectas. No es apropiado usar estos sensores para localizar el robot en trayectorias largas, superficies desiguales o resbalosas. Las lecturas de los encoders pueden ser usadas en conjunto con otros sensores para ayudar a posicionar la plataforma, usando algún algoritmo que permita corregir el error de posición, incrementando así la precisión de la localización.

El sensor con mayor precisión que se tiene es el sensor laser (distancia). Este sensor tiene problemas cuando los obstáculos son transparentes, pero esto puede ser solucionado usándolo en conjunto con los ultrasonidos de forma que un sensor cubra las deficiencias del otro. El láser cubre ángulos desde -90° hasta 90° respecto a la orientación del robot y arroja 180 lecturas cada vez, una por cada grado cubierto. Es posible configurar el sensor para que arroje lecturas cada 0.5° , pero esto implica mayor carga computacional para procesarlas.

Las lecturas de los encoders y el láser son usadas por el algoritmo de localización. Los ultrasonidos y el láser son usados por las habilidades del nivel automático.

1.4.1. Control de la plataforma

La plataforma es controlada por medio de un modelo cliente – servidor. El servidor es el sistema operativo AROS, el cual se encarga del control a bajo nivel de la plataforma. En este nivel se controlan los actuadores y se captura la información de los sensores.

Hay varias APIs (Applications-Programming Interface) usadas para el control de la plataforma, siendo la principal ARIA.

1.4.1.1. ARIA

El cliente es una aplicación que se comunica con AROS y puede compartir información con ella. El cliente recibe los datos de los sensores y actuadores y le envía al servidor los comandos de movimiento de acuerdo a la situación. El cliente usado es la API (Applications-Programming Interface) de ARIA (ActivmediaRobotics Interface forApplication), la cual proporciona una serie de bibliotecas que permiten controlar el robot a alto nivel. ARIA fue desarrollado en C++, por lo que hace uso de las ventajas de la programación orientada a objetos. La clase principal de esta API es ArRobot, la cual proporciona métodos y objetos para la supervisión y control del robot y sincroniza los diferentes subprocesos presentes [Aria11]. La estructura de ARIA se presenta en la figura 10.

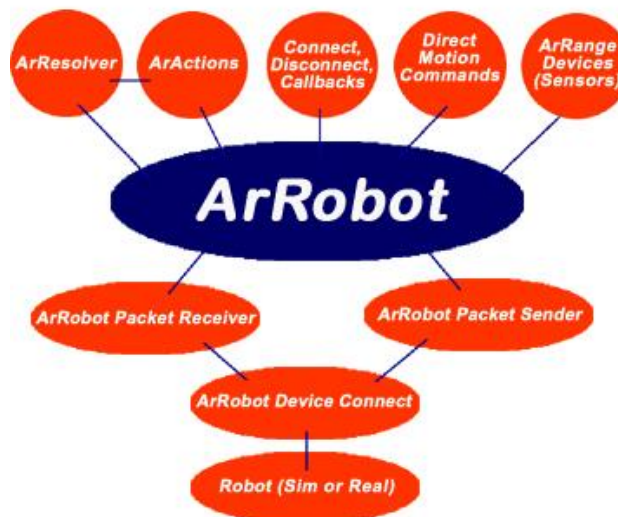


Figura 10: Estructura de ARIA [ARIA11].

ARIA proporciona soporte a Java y Python por medio del uso de envolturas, sin embargo tienen características no soportadas.

Hay principalmente dos formas de dar órdenes a la plataforma con ARIA. La primera de ellas consiste en usar comandos directos o comandos de movimiento. Estos comandos no son adecuados para realizar acciones complejas, ya que se usan para modificar variables a bajo nivel. La segunda forma de controlar la plataforma es por medio de acciones, donde a partir de bloques simples de control de movimiento se puede crear una acción compleja. Las acciones están a un mayor nivel que los comandos de movimiento, permitiendo así esquemas de movimiento más inteligentes. La característica más atractiva de las acciones es que pueden combinarse por medio de un coordinador para generar acciones aún más complejas. Estas acciones son análogas a las habilidades reactivas de las arquitecturas de control reactivas.

1.4.1.2. ArNetworking

Dado que en ocasiones es deseable monitorear o controlar los experimentos en una terminal diferente al robot, se hace uso de ArNetworking[ARIA11]. Esta API permite implementar funciones de red de una forma bastante sencilla. Permite por ejemplo enviar las lecturas de los sensores a un cliente remoto. Para las peticiones al servidor se usan objetos de la clase ArNetPacket, los cuales permiten hacer peticiones de dos tipos: envíos periódicos de información o de solo una vez. También permite implementar sockets de una forma sencilla, usando ArSocket.

En la figura 11 se presenta la estructura de ArNetworking. Básicamente en el servidor se crea un objeto tipo ArServerBase, se habilita para que reciba conexiones entrantes y se le asignan funciones para manejar las peticiones del cliente. Estas funciones pueden ser predeterminadas en la API o personalizadas. En las funciones personalizadas se puede por ejemplo enviar la trayectoria recorrida por el robot, y el cliente la dibujara.

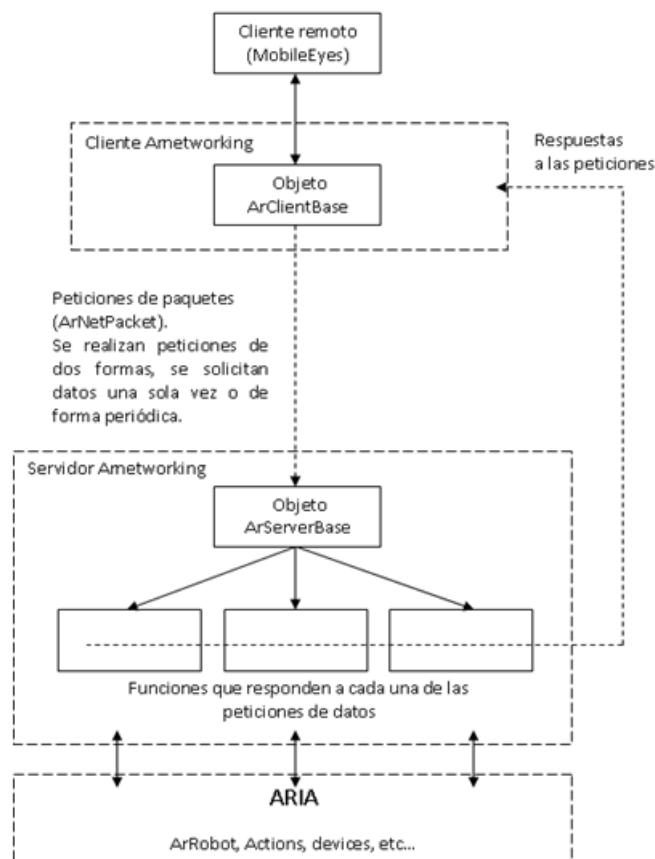


Figura 11: Estructura de ArNetworking[ARIA11].

1.4.1.3. ARNL

ARNL está construido sobre ARIA y su objetivo es realizar navegación inteligente y localización. Hay varias versiones que usan diferentes sensores para realizar la

localización, como laser, sonar y GPS. Para este proyecto se hará uso de la versión que hace uso del laser, la cual localiza al robot usando un método basado en Monte Carlo (MCL) y planea trayectorias basándose en celdas de ocupación. Para el método de localización se genera una puntuación cuyo valor varía entre 0 y 1 asociada con la correlación entre las lecturas del laser y las lecturas esperadas que obtiene a partir del mapa.

Se pueden modificar los parámetros del sistema de localización y planeación de trayectoria por medio de un archivo de configuración (arnl.p). Los principales parámetros configurables se presentan en la tabla 2.

Parámetro	Tipo	Valor por defecto
Mapa a cargar	String	
Numero de muestras para el método de localización (MCL)	Int	2000
Numero de muestras para la localización inicial (MCL)	Int	4000
Ajustar numero de muestras de acuerdo a la situación	Boolean	False
Puntuación mínima para que una localización se considere valida	Double	0,2
Usar fusión sensorial	Boolean	Trae
Tamaño mínimo de la celda de ocupación (mm)	Int	100

Tabla 2: Parámetros de configuración para los algoritmos de localización y planeación de trayectoria

1.4.1.4. Cliente para la visualización del estado de la plataforma MobileEyes

Este cliente permite la visualización gráfica del estado del robot (lecturas de sensores, mapa cargado en el robot, posición, velocidad, entre otros). Es posible agregar nuevos servicios de visualización desde el servidor, definiendo funciones propias, permitiendo así ampliar la funcionalidad del cliente.

1.5. LABORATORIO REMOTO ROBWEB UV

El proyecto del laboratorio remoto abarca la concepción, diseño e implementación de la interfaz *RobWeb-UV* para la experimentación remota y en línea con robots móviles. La interfaz está conformada por tres componentes básicos: la plataforma de experimentación (robot móvil) que actúa como Servidor de Datos, un Servidor de Usuarios que actúa como intermediario y comunica a los clientes con la plataforma y

las aplicaciones cliente mediante las cuales los usuarios interactúan con la plataforma. Cada uno de los componentes se enlaza por medio de una estructura de comunicación que es la que finalmente le permite a los usuarios acceder a la plataforma de experimentación a través de Internet (figura 12). [Calvache09]

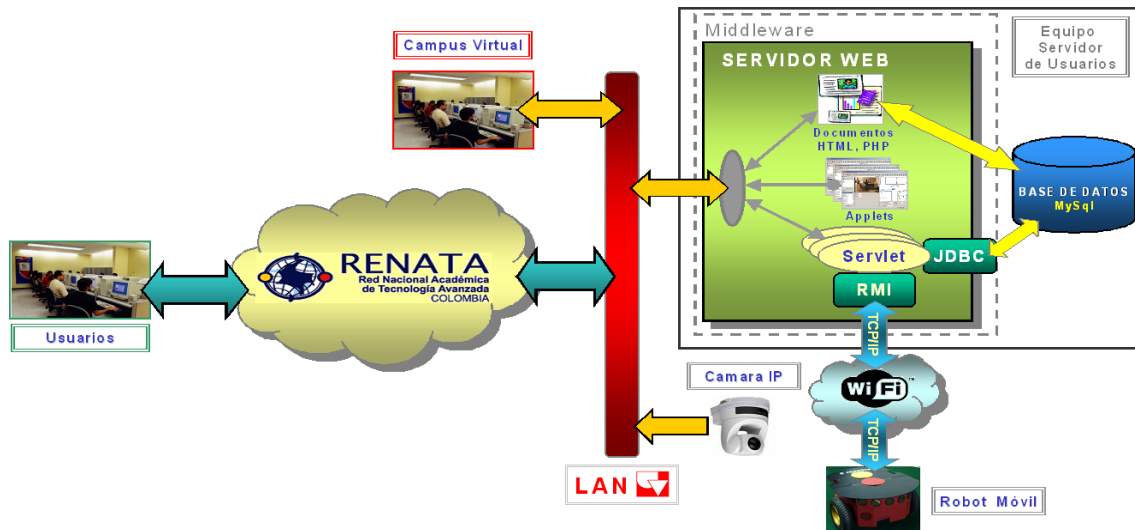


Figura 12: Arquitectura de la interfaz de experimentación remota Robweb UV
[Calvache09]

El componente principal del sistema es el robot móvil de tipo diferencial Pioneer 3DX. El software de control de la plataforma se almacena y ejecuta en un computador embebido en la misma. Los esquemas de control y supervisión del robot están programados según el modelo definido por la arquitectura *Automática Deliberativa AD* [Barber00]. Esta arquitectura se basa en un conjunto de Habilidades Automáticas, actuadores y sensores virtuales, que representan las capacidades sensoras y de actuación del robot. El software de control además de soportar los experimentos desarrollados actúa como un Servidor de Datos que permite a los clientes acceder a la información sensorial real y de estado generado por la plataforma móvil, así como la configuración de los diferentes experimentos soportados. [Calvache09]

El Servidor de Usuarios se enlaza a través de la red inalámbrica al robot móvil y mediante la red LAN de la Universidad a los usuarios finales, tanto externos como internos. Se puede dividir en dos módulos independientes: la interfaz de administración y el módulo *middleware*. La interfaz de administración se encarga del almacenamiento de la información y la administración y gestión de los recursos del sistema (datos, video, etc). El nivel de *middleware* es el que permite que las aplicaciones (Applets) descargadas por los usuarios se puedan comunicar con la plataforma móvil para ejecutar los experimentos.

Los usuarios del sistema de experimentación remota acceden a la plataforma móvil a través de la interfaz de usuario. Esta es una interfaz gráfica creada empleando la tecnología *Applet* de Java. Al ser un *Applet*, la aplicación se descarga desde el Servidor de Usuarios y puede ser ejecutada por cualquier navegador Web con soporte Java (Mozilla, Flock, Internet Explorer, entre otros). La interfaz de usuario se

construye adicionando paneles al *Applet* según el tipo de experimento que se desea realizar. Cada panel tiene una función particular y en conjunto le entregan al usuario la capacidad de supervisar y monitorear la plataforma móvil.

La propuesta del proyecto actual pretende conservar compatibilidad con el laboratorio remoto por lo que el desarrollo se basa en la misma arquitectura de control (automática deliberativa), implementando una aplicación local que trabaja sobre la plataforma móvil sin hacer uso de la arquitectura de comunicación del laboratorio. La contribución de este proyecto al laboratorio remoto son los algoritmos desarrollados para la coordinación de comportamientos en el nivel automático de esta arquitectura que, de ser integrados al laboratorio permitirían realizar experimentos haciendo uso de un coordinador difuso con diversas posibilidades de configuración. Además de esto, las tareas de alto nivel que es posible llevar a cabo apoyándose en el nivel deliberativo implementado, podrían permitir que los usuarios del laboratorio experimenten con algoritmos de localización y seguimiento de trayectoria configurables.

1.6. ESTADO DEL ARTE

Desde los inicios de la robótica, la navegación autónoma de una plataforma ha sido uno de los objetivos principales. Con la evolución del hardware y software que se ha tenido desde ese entonces ha avanzado también la navegación. Los principales retos que se presentan en navegación son la localización, el mapeo y la planeación de trayectorias sobre el entorno donde se pretende navegar. Con sensores más robustos y procesadores capaces de trabajar en tiempo real ambos problemas se han logrado solucionar hasta cierto grado, por ejemplo para el caso de la localización los métodos que hacen uso de visión artificial o algoritmos probabilísticos han adquirido bastante popularidad.

Para el caso del mapeo, generalmente se hace un recorrido guiado del robot por todo el escenario y se almacena la lectura de sus sensores, a partir de las cuales se genera el mapa. Sensores como el láser arrojan mejores resultados que sensores más simples, tales como el sonar o los infrarrojos. A menudo se usan también técnicas de fusión sensorial para incrementar la calidad de las lecturas. Por otra parte la planeación de trayectorias ha sido un área muy activa, se han desarrollado diversos algoritmos para realizar esta tarea. Generalmente se ha tenido la longitud de la trayectoria como el parámetro a optimizar, pero han surgido otras propuestas diferentes que toman en cuenta suavidad de la trayectoria, el mejor camino teniendo en cuenta el tráfico (para vehículos autónomos) [Kristijan08], entre otros. Una habilidad que se encuentra estrechamente relacionada con la planeación de trayectorias es la de seguimiento, para la cual se han propuesto también varias alternativas. Una estrategia de seguimiento bastante usada consiste en considerar los obstáculos no presentes en el mapa (no mapeados) como dinámicos, es decir, que solo estarán presentes un corto periodo de tiempo por lo que no es necesario incluirlos en el mapa ni en el plan y se evaden haciendo uso de la capa reactiva [Valdir06]. También se pueden considerar todos los obstáculos no mapeados como estáticos, incluyéndolos en el mapa y calculando una nueva trayectoria que los tenga en cuenta [Ching08]. Una

última aproximación consiste en la inclusión de un sistema que este en capacidad de determinar el tipo de obstáculo ante el que se encuentra (dinámico o estático) y que determine qué estrategia usar dependiendo de esto.

Debido a la facilidad de implementación, flexibilidad y a la poca carga computacional que genera, la lógica difusa ha probado ser una herramienta válida para diversos bloques de las arquitecturas de control, y se han implementado con éxito sistemas con comportamientos reactivos difusos [Donbing07], coordinadores basados en lógica difusa e incluso algoritmos de mapeo usando técnicas difusas.

A continuación se presenta una tabla comparativa de los diferentes trabajos que se han tomado en cuenta para realizar el estado del arte en navegación deliberativa, realizando una comparación basada en las técnicas y los tipos de sensor utilizados en cada proyecto. Se incluye en la comparación también el proyecto actual, lo que permite obtener una visión general del alcance del proyecto.

Nombre del trabajo	Sensores usados	Técnica
Modular design of fuzzy controller integrating deliberative and reactive strategies[Zhang96]	Infrarrojo	Lógica difusa, diseño modular, navegación reactiva, navegación deliberativa
ISR: An Intelligent Service Robot[Magnus99]	Cámara Ultrasonido Laser	Localización y navegación autónoma. Robótica de servicio
An intelligent semi autonomouspassenger transportation system [Prassler99]	Ultrasonido Infrarrojo Laser	Localización y navegación semiautónoma. Robótica de rehabilitación
Hybrid deliberative/reactive for human-robot interaction [Valdir06]	Cámara omnidireccional Infrarrojo Laser	Localización, navegación semiautónoma e interacción humano-maquina. Robótica de rehabilitación
An improved fuzzy logic based navigation system for mobile robots[Yang03]	Infrarrojo Ultrasonido	Navegación basada en lógica difusa
An architecture for indoor navigation [Wefeng04]	Ultrasonido Laser	Navegación autónoma
A fuzzy logic-based reactive navigation algorithm for mobile robots[Yang05]	Infrarrojo Ultrasonido	Navegación basada en lógica difusa
A fuzzy algorithm for navigation of mobile robots in unknown environments [Tsong-Li05]	Ultrasonido	Navegación basada en lógica difusa
Integration of coordination architecture and behaviour fuzzy learning in quadruped walking robot[Donbing07]	Infrarrojo Cámara	Arquitectura híbrida, lógica difusa
Autonomous navigation of an indoor tour robot guide [Ching08]	Modulo RFID Ultrasonido Laser	Localización, navegación autónoma e interacción humano-maquina. Robótica de servicio

Safe Vehicle Navigation in Dynamic Urban Scenarios [Kristijan08]	DGPS* IMU** Cámaras móviles y estáticas Laser	Localización y navegación autónoma en presencia de otros agentes
A unique multifunctional landmark for autonomous navigation[Wang09]	Cámara	Landmarks y navegación autónoma
Mobile robot navigation using Fuzzy logic controller[Raguraman09]	Infrarrojo Odometro	Navegación basada en lógica difusa
Design of autonomous mobile robot for hospital [Wang09]	Cámara GPS Ultrasonido Infrarrojo INS***	Diseño Localización y navegación autónoma
Una propuesta de navegación deliberativa en robots móviles usando un sistema difuso	Laser Ultrasonido	Navegación reactiva, navegación deliberativa, localización, planeación y seguimiento de trayectorias, lógica difusa

*Differential Global Positioning System

**Inertial Measurement Unit

***InertialNavigationSystem

Tabla 3: Trabajos realizados en el área en el área de navegación deliberativa

1.7. CONCLUSIONES

De acuerdo al estado del arte, la diversidad de los sensores usados en las plataformas móviles ha ido en aumento y claramente predominan los sensores de distancia (ultrasonido, infrarrojo, laser), ya que pueden ser considerados indispensables para cualquier tarea de navegación. Otros sensores bastante usados son las cámaras, que con el aumento del poder de cómputo son una alternativa viable, ya que la visión artificial da mucha información del ambiente.

La lógica difusa es usada en el área de la robótica dado que no requiere de modelos precisos para arrojar resultados válidos. Las propuestas que hacen uso de esta herramienta, generalmente la utilizan para las tareas de navegación, especialmente en el nivel reactivo donde pueden ser útiles para diversas tareas. Pueden diseñarse comportamientos que respondan de forma adecuada, aun ante las incertidumbres de los sensores. También pueden usarse para combinar comportamientos en este nivel, de forma que puedan llevarse a cabo tareas más complejas. Otro uso que se les ha dado es en algoritmos de mapeo. Los tipos de función de membresía que más se encontró en los trabajos consultados fueron triangulares, trapezoidales, gaussianas y sigmoidales.

La arquitectura de control automática deliberativa está basada en la forma en la que razonan y actúan los seres humanos. Dado que los humanos razonan en una forma que puede ser considerada difusa, usar la lógica difusa para implementar parte de esta arquitectura es una propuesta válida.

Se han desarrollado varias bibliotecas para el control de plataformas móviles (como por ejemplo player). En el caso del Pioneer3DX la más usada es ARIA. Para facilitar tareas de comunicación se hace uso de ArNetworking y para facilitar tareas de mapeo, localización y planeación de trayectorias se utiliza ARNL. ARIA fue diseñada usando un paradigma orientado a objetos y de forma modular, lo cual facilita la implementación de diversas arquitecturas de control.

CAPITULO 2

SISTEMA DE NAVEGACIÓN REACTIVO

BASADO EN LÓGICA DIFUSA: NRLD

Unirse es el principio; mantenerse juntos es el proceso; trabajar juntos es el éxito
Henry Ford

Para progresar no basta actuar, hay que saber en qué sentido actuar
Gustavo Le Bon

La arquitectura de control automática deliberativa está compuesta por dos niveles, automático y deliberativo. En este capítulo se presenta el desarrollo e implementación del nivel automático de la arquitectura, al cual se hará referencia como NRLD (Sistema de Navegación Reactiva basado en Lógica Difusa). Se opta por usar esta arquitectura de control por motivos de compatibilidad, dado que fue la utilizada para el desarrollo del laboratorio remoto RobWeb-UV [Calvache09] y se pretende dejar abierta la posibilidad de integrar los desarrollos realizados para el proyecto actual con el laboratorio en algún proyecto futuro.

Los principales componentes del nivel automático son las habilidades automáticas, las acciones reflejas y los coordinadores que se encargaran de procesar las salidas de las habilidades. El esquema propuesto está compuesto por un coordinador cooperativo para las habilidades automáticas y otro competitivo para las acciones reflejas, los cuales se desarrollaron de forma que fueran fáciles de configurar y no causaran excesiva carga computacional en la plataforma sobre la que se ejecutan. Estos

coordinadores fueron implementados en el lenguaje de programación C++, debido a que el soporte que da ARIA a lenguajes como JAVA no es total. Para el coordinador cooperativo se optó por usar lógica difusa para su implementación, dado que los resultados arrojados por esta opción son bastante acertados y es fácilmente configurable, lo que la hace adecuada para una herramienta didáctica. Para el coordinador competitivo se decidió usar una implementación sencilla de un esquema por prioridad.

2.1. COORDINADOR REACTIVO

Para detallar los desarrollos realizados es necesario describir algunas de las clases principales de ARIA [ARIA11], las cuales serán usadas durante esta etapa. Se presenta una introducción a las clases `ArActionDesired`, `ArAction` (para la implementación de comportamientos reactivos), `ArRobot` y `ArResolver` (para coordinadores reactivos).

2.1.1. Clase `ArActionDesired`

Los objetos `ArActionDesired` son muy importantes para esta etapa, ya que son los objetos que usa cada comportamiento para comunicarle al coordinador cuales son los comandos que desea enviarle al robot. Hay seis canales que se pueden modificar:

- Velocidad traslacional: Hace referencia a la velocidad con la que se desea que se traslade el robot.
- Orientación absoluta: Modifica la orientación respecto a un marco de referencia global.
- Orientación relativa: Modifica la orientación respecto a un marco de referencia relativo.
- Velocidad positiva máxima: Limita la velocidad traslacional máxima de avance del robot.
- Velocidad negativa máxima: Limita la velocidad traslacional máxima de retroceso del robot.
- Velocidad rotacional máxima: Limita la velocidad máxima a la que puede rotar el robot.

2.1.2. Clase `ArAction`

Los comportamientos en ARIA son clases heredadas de `ArAction` y deben sobrescribir los métodos `fire()` y `setRobot()`.

El método con la firma `setRobot()` recibe como argumento un puntero a un objeto `ArRobot` y no retorna ningún valor. Este método es usado para que el comportamiento conozca a que robot pertenece y pueda acceder a sus datos sensoriales.

El método `fire()` recibe como argumento un objeto `ArActionDesired` y retorna un puntero a un objeto del mismo tipo. El argumento de entrada es útil cuando se desea conocer cuál es la acción que se desea realizar hasta el momento, posiblemente indicada por otros comportamientos, sin embargo no fue necesario usarlo en ninguno de los comportamientos del proyecto actual. En este método se toman decisiones (generalmente implican movimiento) de acuerdo a las lecturas de los sensores del robot y se modifican los diferentes canales de un objeto `ArActionDesired` para reflejarlas. `ArAction::fire()` es llamado desde el coordinador cuando se desea conocer la salida de un comportamiento dado.

2.1.3. Clase `ArRobot`

Otro objeto de gran importancia en ARIA son los `ArRobot`, ya que representan al robot. De estos objetos se puede extraer información sensorial y modificar parámetros tales como velocidad, orientación, etc. Tanto el coordinador como los comportamientos deben poseer un apuntador al objeto robot en el que están trabajando. Entre los diferentes métodos de la clase `ArRobot` se encuentra `addAction()`. El primer parámetro de este método es un apuntador a un comportamiento mientras que el segundo es una variable de tipo entero cuyo uso depende del coordinador. Este método almacena en un objeto tipo `ActionMap` (definido en la clase `ArResolver`) cada uno de estos pares de argumentos.

Los `ActionMap` se pueden ver como tablas con dos columnas y n filas, donde n es el número total de comportamientos reactivos presentes en el robot (habilidades automáticas más acciones reflejas en este caso).

2.1.4. Clase `ArResolver`

Para la creación de un coordinador usando ARIA es necesario crear una clase que herede de `ArResolver`. Además de esto se debe sobrescribir el método `resolve()`. El método recibe tres parámetros, un puntero a un objeto `ActionMap`, un puntero a un objeto `ArRobot` y un parámetro tipo boolean que indica si se deben incluir datos al log (Esta última variable no fue tomada en cuenta para el desarrollo de los coordinadores en este proyecto). La variable a retornar es un puntero a un objeto `ArActionDesired`. En este método se recuperan los datos de los comportamientos desde el `ActionMap` y se operan dependiendo del coordinador. Hay una metodología a seguir dentro de este método, la cual se presenta a continuación:

- Inicio: se recuperan los comportamientos del objeto `ArResolver::ActionMap`.
- Iteración: se debe iterar por cada uno de los comportamientos y evaluar si existen (si el apuntador no hace referencia a un valor `NULL`) y si están activos. Si el comportamiento evaluado cumple con estas condiciones se procede a llamar su método `ArAction::fire()` y se almacenan los resultados.

- Calcular la salida: Una vez se ha finalizado la iteración, se procede a calcular la salida que será enviada a la plataforma. Esta última etapa es dependiente del coordinador. Por ejemplo un coordinador competitivo por prioridad seleccionará el comportamiento con mayor prioridad y enviara su salida al robot mientras que un coordinador cooperativo tomara en cuenta la salida de cada uno de los comportamientos activos para el cálculo que se requiere realizar.

2.1.5. Estructura del coordinador reactivo

Se hará uso entonces de dos coordinadores diferentes, uno competitivo por prioridad y otro cooperativo basado en lógica difusa. Estos dos coordinadores estarán en una sola clase, dado que según la estructura de ARIA solo es posible asignar un `ArResolver` al robot a la vez. Se implementa el coordinador de esta manera porque es la forma en que se coordinan los comportamientos reactivos en la arquitectura automática deliberativa, diferenciando entre habilidades automáticas y acciones reflejas. Las habilidades automáticas poseen el control de la plataforma siempre y cuando ninguna acción refleja esté activa. Cuando las estrategias de navegación de alto nivel no son suficientes para evitar situaciones que comprometen la integridad de la plataforma, es necesario que se activen las acciones reflejas. Estas acciones, análogas a los reflejos presentes en los animales, toman el control total de la plataforma durante el tiempo necesario para evitar la situación de riesgo.

Como se mencionó anteriormente, para adicionarle comportamientos al robot en ARIA, se hace uso de un método llamado `ArRobot::addAction()`, el cual recibe como parámetros un puntero al comportamiento y un valor entero. Este valor entero es importante, ya que va a ser utilizado como variable para asignar las acciones a un coordinador e indicar su peso o prioridad según sea el caso. El signo de esta variable indica a que coordinador pertenece. Un número negativo indica que la acción es refleja, por lo que pertenece al coordinador por prioridad con una prioridad igual a la magnitud de este número. Un número positivo identificara a la acción como una habilidad automática, cuyo peso es ese mismo número, el cual será procesado por el coordinador difuso. Este comportamiento se logra con un coordinador por prioridad, cuyo algoritmo consiste en identificar si hay acciones reflejas activas y, en caso de que las haya, seleccionar de entre estas la de mayor prioridad y darle control total de la plataforma [Arkin98]. Las acciones que generalmente se incluyen en este coordinador son de evasión de obstáculos, configuradas adecuadamente para responder ante situaciones en las que hay riesgo de colisión.

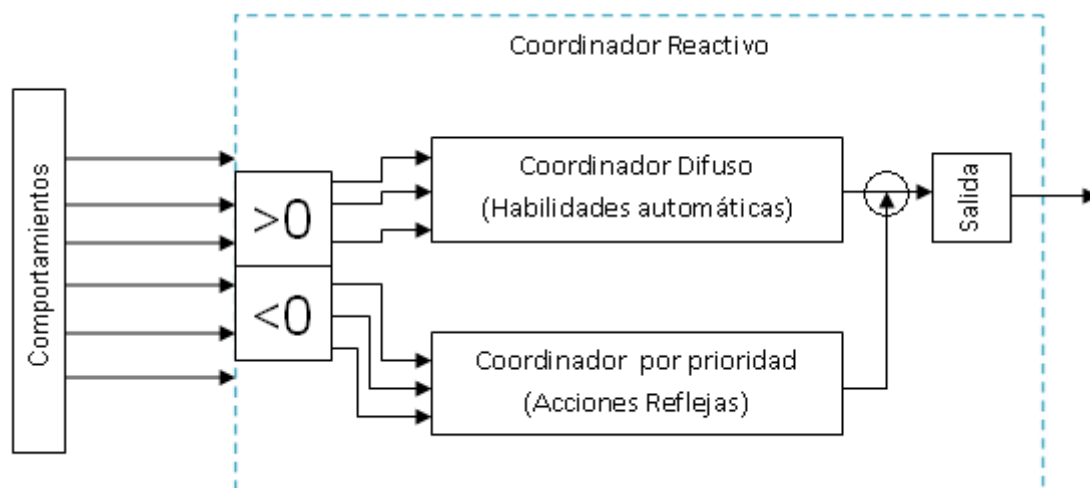


Figura 13: Estructura del coordinador reactivo

Si se activa una acción refleja, se tomara la salida arrojada por el coordinador de prioridad ignorando completamente la salida arrojada por el coordinador difuso. Si ninguna acción refleja está activa se actúa de acuerdo con la salida del coordinador difuso [Barber00].

En la figura 13 se presenta el coordinador reactivo desarrollado, donde se pueden observar ambos coordinadores y como se dividen las habilidades asignadas dependiendo del signo del parámetro usado al adicionarlas.

2.1.5.1. Coordinador por prioridad

El coordinador por prioridad propuesto es muy simple, evalúa si hay acciones reflejas que se encuentren activas y, en caso de ser así, selecciona la de mayor prioridad y envía su salida al robot. En caso de que haya más de una acción refleja con la misma prioridad se toma la que haya sido añadida primero al robot, ignorando las demás. No se toma en cuenta ningún otro parámetro ni se combinan de ninguna forma las acciones en este coordinador.

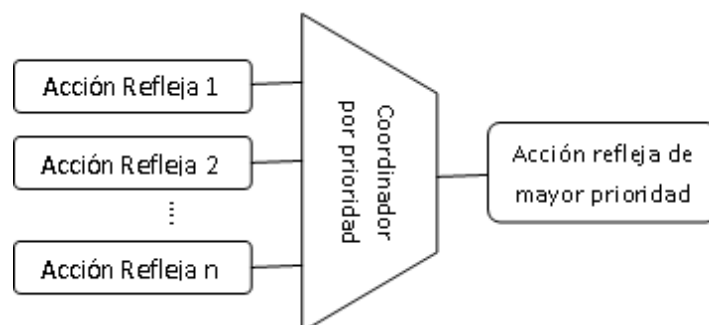


Figura 14: Coordinador por prioridad

En la figura 14 se presenta el esquema del coordinador por prioridad descrito anteriormente. Este coordinador solo se debería activar en casos excepcionales, ya que ajustando correctamente los pesos de las habilidades automáticas del coordinador difuso es posible navegar de forma segura en la mayoría de los casos.

2.1.5.2 Coordinador difuso

El encargado de las habilidades automáticas es un coordinador cooperativo basado en lógica difusa. Este coordinador es más complejo que el anterior y se ha desarrollado de forma tal que pueda ser fácilmente configurable.

Al igual que en cualquier otro coordinador, se determinan que habilidades se encuentran activas, se determina su salida (haciendo uso del método `ArAction::fire()`) y se introducen las variables de interés como entradas al coordinador. Hay un controlador por cada variable, con distintos rangos y funciones de membresía, de forma que sea adecuado para cada variable. Las variables a controlar pueden ser por ejemplo velocidad traslacional, orientación relativa, orientación absoluta, entre otras [ARIA11]. El coordinador por defecto controla la orientación relativa y la velocidad lineal. Si por ejemplo se desea controlar la velocidad lineal, se introducen al coordinador difuso los valores de velocidad lineal arrojados por cada una de las habilidades activas en un objeto tipo `ArActionDesired`. La salida de todos los controladores son entonces unificadas en una acción deseada, que es la acción que se desea el robot realice y ha sido influenciada por la salida de cada una de las habilidades activas.

En la figura 15 se presenta la estructura de un coordinador difuso controlando dos variables. Las variables A y B pueden ser cualquiera de las mencionadas anteriormente. Los conjuntos difusos para cada variable deben ser diferentes, para garantizar que la magnitud de la salida se encuentre en un rango coherente. El conjunto difuso por defecto para la orientación relativa trabaja en un rango de -90 a 90 ya que se hace referencia a grados mientras que el conjunto predeterminado para la velocidad lineal trabaja en el rango de -500 a 500 mm/s. A pesar de que no se usan para este proyecto habilidades que arrojen una velocidad lineal negativa, el coordinador está en capacidad de procesarla si esto se llega a presentar en el futuro.

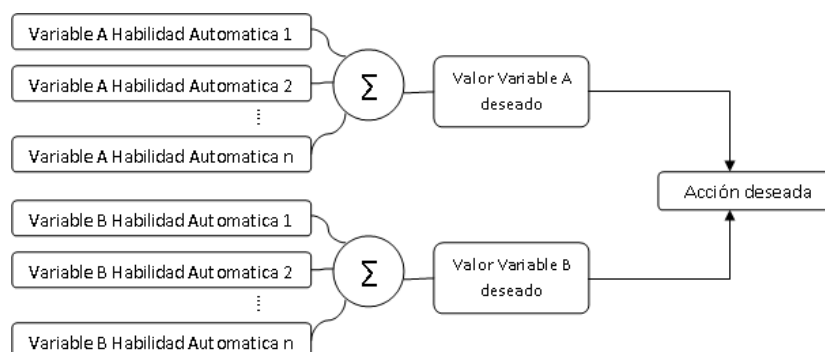


Figura 15: Coordinador difuso controlando dos variables

Los conjuntos difusos usados para controlar la velocidad lineal y la orientación relativa se muestran en la figura 16. Se hace uso de estos conjuntos tanto para la salida de cada una de las habilidades (entradas al sistema) como para la salida del coordinador. Dado que el tiempo del ciclo de trabajo del robot es de 100ms, se debe implementar un sistema lo más ligero posible, razón por la cual se optó por este esquema en lugar de hacer uso de funciones gaussianas, cuyo cálculo requiere mucho más tiempo.

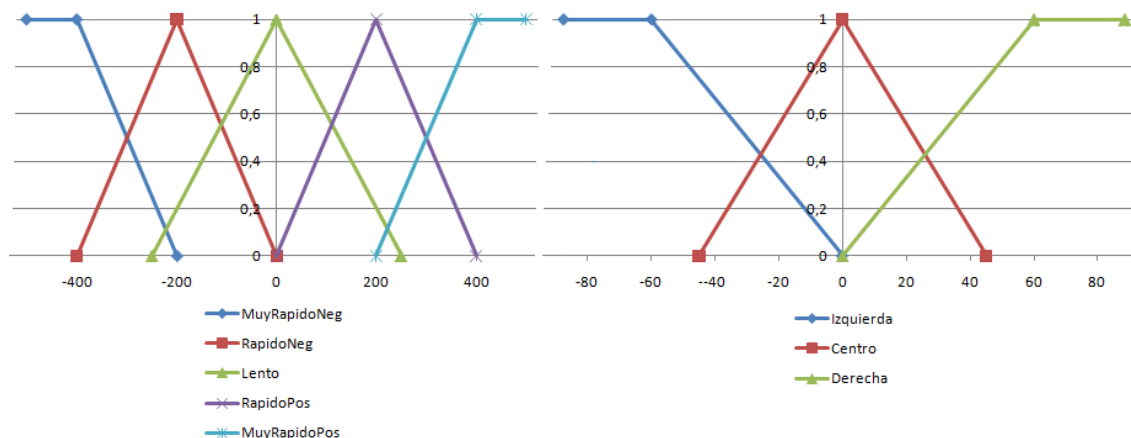


**Figura 16: Conjuntos difusos del coordinador por defecto.
Velocidad lineal (Izquierda) – Orientación relativa (derecha).**

Los sistemas difusos son fáciles de implementar una vez se tiene una herramienta para hacerlo. En el caso de este proyecto fue necesario desarrollar también esta herramienta, una biblioteca para la implementación de sistemas difusos. A continuación se presentan los parámetros usados para los conjuntos difusos de los controladores por defecto (se presenta más detalle acerca del uso de estas funciones propias de la biblioteca en la sección 2.3):

- Velocidad traslacional (-500, 500):
 - FuncionTrap("mrapidoneg", -600, -500, -400, -200);
 - FuncionTriangular("rapidoneg", -400, -200, 0);
 - FuncionTriangular("lento", -250, 0, 250);
 - FuncionTriangular("rapidopos", 0, 200, 400);
 - FuncionTrap("mrapidopos", 200, 400, 500, 600);
- Orientación relativa (-90, 90):
 - FuncionTrap("izquierda", -180, -90, -60, 0);
 - FuncionTriangular* centro=new FuncionTriangular("centro", -45, 0, 45);
 - FuncionTrap* derecha=new FuncionTrap("derecha", 0, 60, 90, 180);

Para cada función de membresía se debe definir un nombre, el mismo con el cual será identificada en las reglas lingüísticas. Además de esto se requiere una serie de parámetros numéricos dependientes del tipo de función que se desea usar, para una función triangular se deben definir los tres vértices del triángulo mientras que para una función trapezoidal se definen cuatro valores: inicio, primer punto máximo, segundo punto máximo y final. Estas funciones individuales se agrupan en conjuntos difusos, que se corresponden a las variables que se desean controlar, los cuales definen los rangos de operación (universo de discurso). Cualquier valor fuera de este rango se considera nulo, por lo que las funciones que estén fuera del rango, se comportarán como si hubieran sido cortadas, tal como se muestra en la figura 15 para el caso de las funciones trapezoidales.

Para describir la relación entre las entradas y la salida, se hace uso de las reglas lingüísticas. Estas reglas son expresiones condicionales de la forma IF “nombre de la habilidad” IS “nombre función de membresía” THEN “nombre salida” IS “nombre función de membresía”. Varias condiciones pueden ser usadas en la misma regla por medio de conectores AND u OR (en el coordinador se usaron OR).

Si por ejemplo se tienen dos habilidades en el coordinador difuso, las reglas a usar para controlar la orientación relativa serían:

- IF Habilidad1 IS izquierda OR Habilidad2 IS izquierda THEN salida IS izquierda
- IF Habilidad1 IS derecha OR Habilidad2 IS derecha THEN salida IS derecha
- IF Habilidad1 IS centro OR Habilidad2 IS centro THEN salida IS centro

Dado que a la salida del sistema se tiene un conjunto difuso, se hace necesario aplicar un método para obtener una salida numérica. Para el proyecto se usa el método del centroide de masa, el cual muy comúnmente usado en el área de lógica difusa, de fácil implementación y que arroja resultados bastante acertados.

A parte del método `ArResolver::resolve()` que se debe sobrescribir para que el coordinador funcione y donde se ha implementado todo lo descrito anteriormente, se han escrito también otras funciones para permitir que se pueda configurar para que use otros conjuntos difusos y controle variables diferentes a las que se determinaron por defecto (velocidad traslacional y orientación relativa).

Para la configuración del coordinador se debe hacer uso del método `configurarCoordinador()`. Este método recibe cuatro parámetros y no devuelve resultados. Los dos primeros argumentos son punteros a conjuntos difusos para la entrada y la salida respectivamente. Estos conjuntos deben ser creados usando la biblioteca de clases desarrollada para el proyecto (sección 2.3) y por lo tanto puede usarse cualquier tipo de función de membresía a la que se le de soporte. Al momento del desarrollo del proyecto, se da soporte a funciones triangulares, trapezoidales y gaussianas.

El tercer argumento es un vector<string> que contiene las diferentes reglas lingüísticas a utilizar. Los vectores en C++ son un tipo de contenedor definido en la STL (Standard Template Library) y pueden verse simplemente como arreglos dinámicos. A diferencia de los arreglos clásicos, un vector conoce su tamaño (lo que evita la necesidad de pasar un argumento más para indicar el tamaño del arreglo) y arroja excepciones

cuando se trata de referenciar elementos que no existen, lo que lo hace más seguro (es posible referenciar posiciones más allá del tamaño máximo de un arreglo clásico, lo que ocasiona que se modifiquen posiciones de memoria de forma irregular, ocasionando problemas en tiempo de ejecución difíciles de depurar). Es posible modificar su tamaño en tiempo de ejecución, y añadir elementos dependiendo de las necesidades, lo que los hace perfecto para almacenar datos cuya longitud no se conoce. Se hace uso extensivo de estos vectores para permitir una configuración dinámica del coordinador.

Las reglas lingüísticas que se usan en este vector están simplificadas, es decir, solo se usan como un medio para asociar funciones de membresía a la entrada con funciones de membresía a la salida. Por ejemplo para obtener el controlador para orientación relativa presentado anteriormente, el vector que se pasa como parámetro debe contener los siguientes strings:

- “if habilidad is izquierda then salida is izquierda”
- “if habilidad is centro then salida is centro”
- “if habilidad is derecha then salida is derecha”

Este método hace posible la independencia entre la entrada de configuración y el número de habilidades. Se pueden definir relaciones más complejas entre la entrada y la salida, solo es necesario mencionar la relación que se desea que tengan. Cuando esta relación implica más de una habilidad, se hace uso de la operación “OR” para realizar la unión.

El cuarto y último argumento es un string que define que variable se desea controlar con el sistema ingresado y que pueden ser cualquiera de los canales de un objeto tipo `ArActionDesired`: “velocidad traslacional”, “orientación absoluta”, “orientación relativa”, “velocidad positiva máxima”, “velocidad negativa máxima” o “velocidad rotacional máxima”.

Para evitar tener que cargar desde el `ArResolver::ActionMap` las habilidades y crear los conjuntos difusos cada vez que se ejecuta el método `ArResolver::resolve()` (operación innecesaria y que crea una carga computacional significativa), se crea el sistema solo una vez al inicio de la aplicación, y no será eliminado a menos que se haga uso de otro método definido para el coordinador pero que solo puede ser usado internamente, el método `destruir()`. Este método es llamado cuando se configura el coordinador, eliminando así los conjuntos difusos por defecto y creando el nuevo controlador para la variable deseada. Para controlar más de una variable se debe llamar de nuevo el método.

Se puede presentar el caso en el que se defina un coordinador incoherente, como sería el caso en el que se tengan por ejemplo solo controladores para velocidad negativa máxima y velocidad positiva máxima. Estas variables no darán por sí solas movimiento a la plataforma, por lo que no tendría sentido hacerlo, sin embargo se ha dejado abierta la posibilidad dado que se presenta la opción de configurar el coordinador al gusto del usuario como una herramienta para la experimentación y el aprendizaje. La plataforma no correrá peligro aun cuando se haga esto, dado que el

coordinador por prioridad a cargo de las acciones reflejas se encarga de garantizar la integridad de la plataforma en todo momento.

En caso de que se añadan nuevas habilidades al robot, es necesario redefinir los sistemas difusos, por lo que son desechados llamando la función `destruir()` y se reconstruyen teniendo en cuenta las nuevas habilidades adicionadas al robot. En caso de que se modifiquen los pesos esto no es necesario, simplemente se modifica la variable asociada al peso que ha cambiado.

2.2. COMPORTAMIENTOS REACTIVOS

En el nivel automático de la arquitectura utilizada se encuentran las acciones reflejas y las habilidades automáticas, las cuales se diferencian por el coordinador que procesara su salida. Las acciones reflejas toman el control de la plataforma si están activas, sino el control pertenece a la salida del coordinador cooperativo que consiste en una combinación de la salida de todas las habilidades automáticas activas. Algunas de estas habilidades fueron desarrolladas en C++ tomando como base las habilidades del laboratorio remoto RobWeb-UV [Calvache09], las cuales hacen uso tanto del sonar como del láser para lograr un mejor desempeño.

A continuación se realiza una descripción de cada uno de los comportamientos usados.

2.2.1 Habilidad evitar obstáculos

Permite evitar obstáculos en un rango dado a cierta velocidad (si se usa como acción refleja esta velocidad debe configurarse para ser nula o muy baja). Se pueden usar diferentes configuraciones de esta misma habilidad para formar un esquema de evasión más elaborado. Hace uso del método con la firma `double ArRobot::checkRangeDevicesCurrentPolar(double angulo1, double angulo2)`, el cual retorna la lectura mínima entre los ángulos establecidos, lectura que puede provenir de cualquiera de los sensores de rango, en este caso sonar o laser. Para su inicialización se requiere llama el constructor que recibe los siguientes parametros:

- nombre: parámetro tipo string usado para identificar internamente la habilidad.
- distMin: parámetro tipo double que determina la distancia mínima a la cual se puede encontrar un objeto para activar la habilidad. Parámetro en milímetros (mm).
- giro: parámetro tipo double que indica cuantos grados se debe girar en caso de que se encuentre un obstáculo.
- velGiro: parámetro tipo double que indica la velocidad a la que se debe girar. Esta velocidad es traslacional, por lo que si la distancia mínima usada es baja, se recomienda que esta velocidad también lo sea. Parámetro en milímetros por segundo (mm/s).
- angulo1 y angulo2: parámetros tipo double que indican la zona que la habilidad va a cubrir. Deben estar en el rango $0^\circ - 90^\circ$, a pesar de que los sensores

cubren desde -90° a 90° . Esto se debe a que la habilidad toma en realidad dos zonas: la zona delimitada por los ángulos ingresados y la zona delimitada por el negativo de estos mismos ángulos. Si por ejemplo se inicializa con 40° y 90° , se cubrirán las zonas desde 40° hasta 90° y desde -90° hasta -40°

La evasión que se realiza es muy simple, se observa si hay obstáculos que se encuentren más cerca de lo que deberían (límite definido por `distMin`), se determina a qué lado se encuentra el más cercano (izquierda o derecha), y se gira en sentido contrario. Una vez hecho esto se configuran dos parámetros de un objeto `ArActionDesired`, la velocidad (`velGiro`) y el ángulo que se debe girar (`giro`, su signo cambia dependiendo de la dirección en la que se desee girar). Esta configuración se hace por medio de los métodos `ArActionDesired::setVel(double)` y `ArActionDesired::setDeltaHeading(double)`;

2.2.2. Habilidad velocidad constante

Ordena a la plataforma avanzar a una velocidad dada. Esta habilidad es útil cuando se desea realizar navegación aleatoria, ya que garantiza que el robot va a estar siempre en movimiento.

Para su inicialización se hace uso del constructor, que recibe como argumentos:

- nombre: parámetro tipo `string` usado para identificar internamente la habilidad.
- velocidad: parámetro tipo `double` que indica la velocidad que la habilidad configurara en su objeto `ArActionDesired`.

En su método `ArAction::fire()` se pasa al objeto `ArActionDesired` la velocidad. No toma en cuenta ninguna información sensorial ni modifica ningún otro canal.

2.2.3. Habilidad seguir trayectoria

Habilidad desarrollada para el proyecto que implementa un seguimiento de trayectoria sencillo. Su constructor recibe los siguientes parámetros:

- `L`: parámetro tipo `double`, usado para indicar la distancia de la posición objetivo a seguir.
- `maxVel`: parámetro tipo `double` que hace referencia a la velocidad de seguimiento.

Además del constructor hay otro método que se usa bastante en esta clase, el método `setTrayectoria()`. Este método recibe una referencia a una lista de objetos tipo `ArPose` y no retorna argumentos.

Las listas en C++ están definidas en la STL (Standard Template Library) y son bastante utilizadas debido a que se pueden recorrer de forma bastante rápida (han sido optimizadas para ello). Requieren del uso de iteradores, objetos de un tipo diseñados para ser usados para recorrer diferentes tipos de contenedores.

Los objetos tipo ArPose representan una posición en un plano (x, y, orientación). Todas las posiciones en ARIA están representadas por objetos de este tipo.

Si se está iniciando el seguimiento, se debe orientar el robot primero, por lo que se detiene y modifica su orientación de forma que se facilite el seguimiento. Una vez hecho esto, se ubica el punto más cercano a una distancia L, configurable por el usuario (300mm por defecto), y trata de alcanzarlo. Esta habilidad arroja siempre una velocidad lineal constante, que se puede configurar al momento de la inicialización (400mm/s por defecto), modificando solo su orientación para alcanzar el objetivo. El punto objetivo es continuamente actualizado, reemplazándolo por el que se encuentre más cerca a la distancia L, haciendo esto hasta el final de la trayectoria. Usado por la habilidad deliberativa “planeación y seguimiento de trayectorias”. Se presenta el diagrama de flujo del algoritmo utilizado en la figura 17.

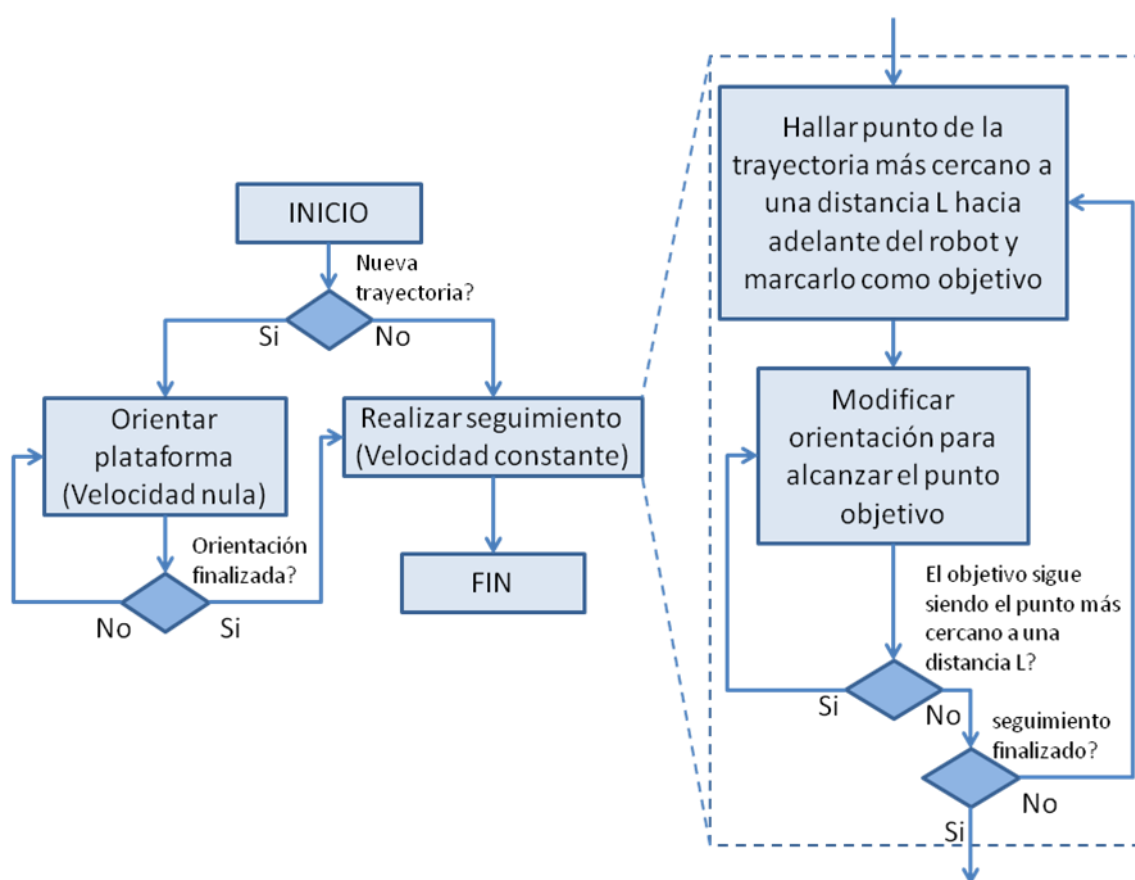


Figura 17: Algoritmo para la habilidad de seguir trayectoria

Como ya se menciono anteriormente, la velocidad lineal es constante y solo se modifica la orientación para seguir la trayectoria. El ángulo por el cual se debe modificar la orientación relativa del robot se puede apreciar en la figura 18.

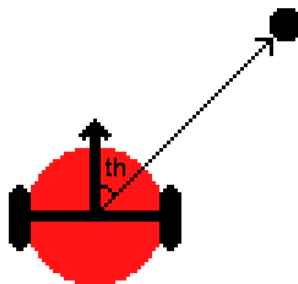


Figura 18: Seguimiento de trayectoria

2.3. BIBLIOTECA PARA LA IMPLEMENTACIÓN DE SISTEMAS DIFUSOS

Para la implementación de los sistemas difusos necesarios para el proyecto, se requiere una biblioteca de funciones que facilite esta tarea. Se pensó en inicialmente usar FuzzyLite [FuzzyLite11], pero a pesar de que provee las herramientas necesarias para crear un sistema difuso, su utilización es compleja. Por esto se optó por desarrollar una biblioteca propia con una estructura más simple e intuitiva, de forma que pudiera ser usada con fines educativos. Otro impedimento a la hora de elegir una librería fue el lenguaje de programación, ya que hay algunas librerías escritas en JAVA que hubieran sido útiles, como jFuzzyLogic [jFuzzyLogic11], pero el coordinador requiere ser escrito en C++ por limitaciones de la envoltura de ARIA. Se desarrolló entonces una biblioteca para la implementación de sistemas difusos. Esta biblioteca permite desarrollar sistemas difusos completos definiendo:

- Conjuntos difusos (para entradas y salida)
- Reglas lingüísticas
- Método para resolver la salida
- Controlador (resuelve la salida usando el método deseado)

La biblioteca desarrollada es utilizada en el proyecto para la implementación del coordinador de habilidades automáticas, parte del sistema de navegación reactivo basado en lógica difusa.

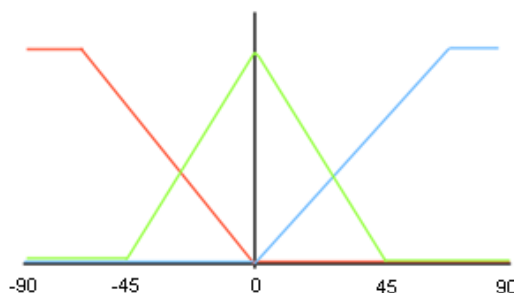


Figura 19: Ejemplo de conjunto difuso

Las funciones de membresía actualmente disponibles son triangular, trapezoidal y gaussiana. Se escogieron estas funciones dado que son muy usadas en el campo de la lógica difusa. En la figura 19 se muestra un conjunto difuso con tres funciones de membresía, dos trapezoidales y una triangular. La adición de una función de membresía a un conjunto difuso depende de la forma de la función. Por ejemplo una función de membresía triangular debe definir tres puntos (inicio, punto máximo y final), pero una trapezoidal debe definir cuatro (inicio, primer punto máximo, segundo punto máximo, final). El caso de las funciones gaussianas es un poco más complejo, dado que son definidas por la ecuación $f(x) = e^{\frac{-(x-b)^2}{2c^2}}$. Para crear una función de membresía gaussiana se deben definir b (centro) y c (ancho).

Se presenta un diagrama de clases simplificado de la biblioteca en la figura 20. Las flechas punteadas indican una relación de composición y las no punteadas indican herencia.

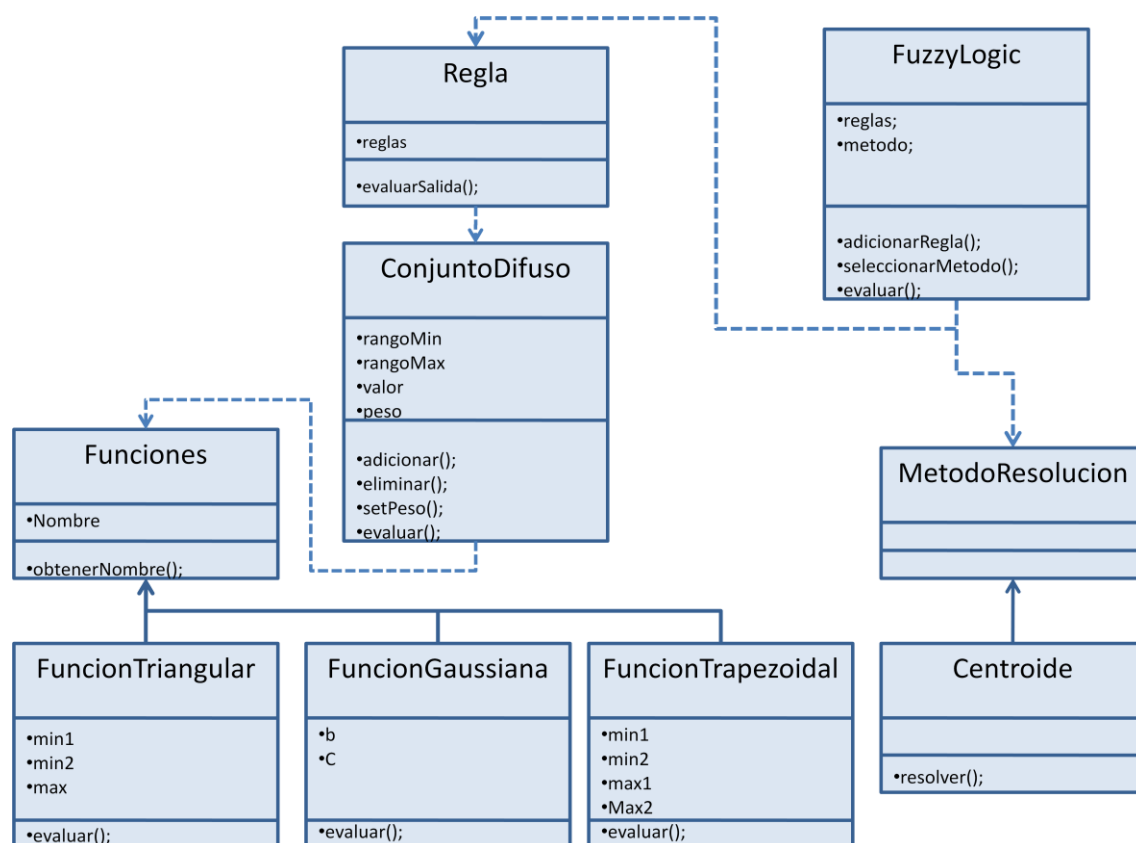


Figura 20: Diagrama de clases de la biblioteca

Las funciones triangular, trapezoidal y gaussiana son heredadas de la clase Funciones y el método del centroide de masa es heredado de MetodoResolucion. Este diseño modular se hizo con el fin de permitir la fácil extensión de la biblioteca en caso de ser requerido, con lo que se pueden incluir nuevas características sin alterar el código existente.

Para crear un nuevo tipo de función de membresía, se debe crear una nueva clase que herede de Funciones y sobrescribir un método llamado evaluar(), el cual recibe y retorna parámetros de tipo double. Este método sirve para obtener el valor de la nueva función en el punto definido por el parámetro de entrada, por lo que parte de la precisión de la implementación dependerá de este método.

Los métodos y atributos de la clase Funciones se sintetizan en la tabla 4.

Virtual double evaluar(double)=0	Método definido como abstracto, por lo que debe ser sobrescrito por las clases que hereden de "Funciones". Debe retornar el valor de una función en un punto. Tanto el argumento de entrada como de salida son de tipo double.
stringobtenerNombre()	Este método permite obtener el nombre de la función de membresía
Nombre	string que es usado para identificar internamente la función de membresía

Tabla 4: Métodos y atributos de la clase Funciones

Las funciones de membresía deben ser adicionadas a un conjunto difuso, para lo que se hace uso de la función adicionar(). Este método recibe como argumento un puntero a cualquiera de las clases que hereden de funciones y guarda una copia interna de este puntero para operar sobre él según sea necesario. Un conjunto difuso puede contener muchas funciones de membresía, tantas como se requiera. Otras funciones de interés son setValor() y setPeso(), las cuales reciben argumentos tipo double y permiten indicarle al conjunto difuso cuál será su peso y valor respectivamente (por defecto se considera que el peso es 1 y el valor es 0). El valor es la variable de entrada en un sistema difuso.

En la tabla 5 se presentan otras funciones presentes en la clase "ConjuntoDifuso".

ConjuntoDifuso() ConjuntoDifuso(string,double,double)	Constructores de la clase. Se puede inicializar con el nombre, el rango mínimo y el rango máximo. En caso de no hacerse se toma por defecto nombre="", rangMin=0.0 y rangMax=1.0.
void adicionar(Funciones*)	Método usado para adicionar funciones de membresía al conjunto difuso.
vector<Funciones*>obtenerMiembros()	Método que retorna un vector que contiene apunadores a cada una de las funciones de membresía que contiene.
voidcambiarNombre(string)	Usado para cambiar el nombre del conjunto difuso (útil si se tienen nombres duplicados).
stringobtenerNombre()	Retorna el nombre del conjunto difuso
doubleobtenerRangMin() doubleobtenerRangMax()	Métodos que retornan el valor configurado para el rango.
voidsetValor(double)	Usado para cambiar el valor del conjunto (es valor es el parámetro de entrada en un sistema difuso).

void setPeso(double)	Método que se usa para modificar el peso del conjunto difuso.
double obtenerValor() double obtenerPeso()	Estos métodos se usan para recuperar el valor asignado a las variables "valor" y "peso".
double evaluar(string, double)	Recibe como argumentos el nombre de una función de membresía y el punto donde se desea evaluar. Esta función hace uso de la función "Funciones::Evaluar(double)", sin embargo cualquier valor por fuera del rango del conjunto difuso es considerado como cero, independientemente del valor de la función de membresía.
nombre	Atributo string usado para identificar internamente el conjunto difuso
rangMin, rangMax, valor, peso	Atributos tipo double para almacenar el rango mínimo, rango máximo, valor y peso respectivamente.
funciones	vector<Funciones*> que almacena los punteros a cada una de las funciones de membresía adicionadas al conjunto.

Tabla 5: Métodos y atributos de la clase "ConjuntoDifuso"

Otra clase de mucha importancia en la biblioteca es "Regla", ya que es el medio para ingresar las reglas lingüísticas al sistema. Esta es quizá también la clase más compleja, ya que se debe encargarse de convertir las reglas lingüísticas a una forma que se pueda procesar, tomar los valores de cada uno de los conjuntos difusos asociados y generar la salida. Es en esta clase donde se aproxima la salida a una serie de puntos, por lo que aquí se puede modificar el número de muestras que se desea usar para ello. Entre mayor sea el número de puntos (muestras) usado, mejor será la representación de la salida y por lo tanto se tendrán resultados con mayor precisión, pero se requerirá también un mayor número de iteraciones para resolver el sistema y más espacio en memoria, lo que conlleva a un aumento en el uso de recursos.

Para inicializar un nuevo objeto "Regla" se hace uso del constructor que recibe tres parámetros. El primer argumento de entrada es de tipo string y debe contener una regla lingüística como tal, con el nombre de los conjuntos difusos y funciones de membresía a las que haga referencia escritos correctamente. Las operaciones AND u OR son determinadas por la primera operación que se detecte, de izquierda a derecha, es decir, si hay una OR al principio y desde ese punto se usa solo hay AND, se tratará el sistema como si solo hubiera OR. La regla lingüísticas deben entonces estar escritas de la forma:

*IF nombreConjuntoDifuso1 is nombreFuncionMembresia1 AND/OR
nombreConjuntoDifuso2 is nombreFuncionMembresia2 AND/OR ...
nombreConjuntoDifusoN is nombreFuncionMembresiaN THEN
nombreConjuntoDifusoSalida IS nombreFuncionMembresiaSalida.*

El segundo argumento del constructor es un vector que contiene apuntadores a los ConjuntosDifusos implicados en la regla. El tercer y último argumento es un apuntador al conjunto difuso de salida.

Cada objeto de la clase “Regla” se corresponde con una y solo una regla lingüística. La salida evaluada por cada una de las reglas se integrará después en el controlador, un objeto tipo “FuzzyLogic” que es explicado más adelante con detalle en el documento.

En la tabla 6 se presenta una síntesis de todos los métodos y atributos usados para la clase “Reglas”.

Regla(string,vector<ConjuntoDifuso*>const, ConjuntoDifuso*)	Constructor. Recibe como argumentos la regla lingüística y apuntadores a los conjuntos difusos tanto de entrada como de salida implicados en la regla.
void interpretar()	Método usado internamente para interpretar el string que contiene la regla lingüística. Con este método se identifican los conjuntos difusos y funciones de membresía implicados, así como la operación a realizar (AND/OR). Esta función almacena sus resultados en diferentes vectores, tales como “conjuntos” y “miembros”.
vector<string>obtenerConjuntos() vector<string>obtenerMiembros()	Métodos usados para obtener los nombres de los conjuntos y las funciones de membresía implicadas.
doubleobtenerDelta()	Retorna el valor “delta”. Este valor se calcula con base en el rango de la salida y el número de muestras que se desea usar.
vector<double>evaluarSalida()	Método que retorna un vector que contiene los valores de la salida evaluada según la regla. El tamaño de este vector depende del parámetro “numeroMuestras”
doubleobtenerRangSalida()	Retorna el valor del rango mínimo de la salida.
double evaluar()	Este método evalúa cada una de las funciones de membresía implicadas en el punto indicado por el valor del conjunto difuso. Una vez se ha realizado esta operación, se retorna el máximo (OR) o el mínimo (AND) dependiendo de la operación que se debe realizar para la regla.
numeroMuestras	Atributo tipo unsignedint. Se puede modificar para cambiar el número de puntos con el que se representa la salida. Tiene un valor de 50 por defecto.
Delta	Este valor se calcula con base en el rango de la salida y el número de muestras que se desea usar: $\left(\frac{rangMax - rangMin}{numeroMuestras} \right)$.
reglaString	Atributo usado para almacenar la regla lingüística sin procesar. Tipo string.
Salida[2]	Arreglo de tipo string. Contiene dos datos, el nombre del conjunto difuso de la salida y el nombre la función de membresía en ese conjunto al que se hace referencia en la regla.

descomponer conjuntos miembros operaciones	Objetos tipo <code>vector<string></code> . Usados para almacenar diferentes datos que se obtienen a partir de la regla lingüística.
conjuntosDifusos	Vector que contiene los apuntadores a los conjuntos difusos implicados en la regla.
salidaEvaluada	Vector que contiene la salida evaluada para esta regla, representada como una serie de puntos. Su tamaño depende del parámetro "numeroMuestras".

Tabla 6: Métodos y atributos de la clase "Reglas"

La clase `MetodoResolucion` proporciona una interfaz para los métodos de resolución. Para crear un nuevo método basta con heredar de esta clase abstracta y sobrescribir el método `resolver()` que es, de hecho, el único método definido en esta clase. Este tipo de estructura se usa para lograr un comportamiento polimórfico. El primer argumento es la salida, que ha sido aproximada a una serie de puntos y almacenada en un vector (`vector<double>`). El segundo argumento es tipo `double` y representa la distancia entre cada punto de la salida (delta). El último argumento es tipo `double`, que define el rango mínimo del conjunto difuso. Con estos tres parámetros se conoce suficiente información para poder calcular una salida numérica.

Para el caso del centroide, para calcular la salida del sistema se calcula el centroide de cada una de las figuras que la componen y se suman, sin embargo este cálculo no siempre es sencillo dado que en ocasiones las figuras son complejas. Como se mencionó anteriormente, la salida es aproximada a una serie de puntos para simplificar el problema (el número de puntos es configurable) y se aproxima a una serie de rectángulos.

Para calcular el centroide se realiza la operación $c = x * y * z$ a cada rectángulo, donde "x" es el ancho (en este caso definido por el parámetro Delta), "y" es el alto (dependiente del valor de la salida en ese punto) y "z" hace referencia a la posición del rectángulo en el rango del conjunto difuso (el centroide del rectángulo individual), por lo que puede ser positivo o negativo. Se presentan estos datos de forma grafica en la figura 21.

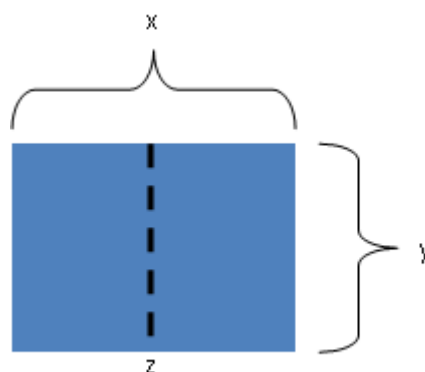


Figura 21: Atributos de un rectangulo

Estos resultados parciales se acumulan y, al final, se dividen por el área total de la salida (sumatoria del área de todos los rectángulos), con lo que se obtiene el centroide. Cabe anotar que a mayor número de puntos usados para representar la salida, menor será el delta y mayor el número de operaciones para calcular el centroide. Esto trae consigo una mayor precisión a cambio de mayor carga computacional.

La clase “FuzzyLogic” se encarga de integrar todos los objetos de las clases descritas anteriormente y de aplicar un método para resolver la salida. Para crear un objeto de este tipo se usa el método “FuzzyLogic(vector<Regla*>)”, que recibe como argumento un vector que contiene apunadores a cada una de las reglas lingüísticas usadas. También puede inicializarse con un constructor vacío (FuzzyLogic()) y adicionarle las reglas vía el método adicionarRegla(Regla*). También es necesario indicarle un método para resolver la salida, lo cual se hace con seleccionarMetodo(MetodoResolucion*). La clase “MetodoResolucion” (conocido en la literatura como método de defuzzyficacion) es una clase abstracta que brinda una interfaz que pueden implementar los diferentes métodos de resolución (por ejemplo el del centroide). La función de los objetos de la clase “FuzzyLogic” es tomar la salida evaluada de cada regla, integrarla en una sola y pasarla al objeto que se encargara de resolverla.

En la tabla 7 se presenta un resumen de los métodos y atributos de esta clase.

FuzzyLogic() FuzzyLogic(vector<Regla*>)	Constructores de la clase. Pueden pasarse las reglas como parámetro durante la inicialización o después, haciendo uso de el método adicionarRegla(Regla*).
voidadicionarRegla(Regla*)	Adiciona una regla al sistema difuso. Toma como parámetro un apuntador a la regla que se desea añadir.
voidseleccionarMetodo(MetodoResolucion*)	Por medio de este método se le indica al sistema que método se desea usar para resolver la salida. Recibe como parámetro un apuntador al objeto tipo MetodoResolucion.
double evaluar()	Método que evalúa el sistema. No toma parámetros, ya que para modificar las entradas al sistema se deben modificar directamente los valores de los conjuntos difusos. Internamente hace uso del método MetodoResolucion::resolver(). Retorna un valor tipo double, correspondiente a la salida del sistema.
Reglas	Vector usado para almacenar las diferentes reglas implicadas en el proceso.
Método	Atributo que contiene un objeto tipo MetodoResolucion, el cual será usado para resolver la salida del sistema.

Tabla 7: Métodos y atributos de la clase "FuzzyLogic"

Como se ha mencionado anteriormente se hace uso de la propiedad de polimorfismo propia de la programación orientada a objetos, de forma que se pueda lograr una buena modularidad. El comportamiento polimórfico, en pocas palabras, hace referencia a la capacidad de tratar cada objeto que ha heredado de una clase padre como si fuera un objeto de la clase padre. Esto trae enormes ventajas ya que cada clase que ha heredado algún método, lo puede implementar de forma diferente. Si se desea evaluar una función triangular o una función trapezoidal en algún punto, en ambos casos se usa la misma llamada a función (evaluar(double)), pero cada clase se evaluará de forma diferente dependiente de su implementación propia.

Como se habrá notado, se hace uso de apuntadores frecuentemente. Cuando se pasan argumentos en lugar de apuntadores, se crea una copia en memoria de estos, lo que hace que se cree un desperdicio de memoria y tiempo (el proceso de copia es lento). Pasar apuntadores en lugar de los objetos directamente permite que se trabaje directamente con el objeto en lugar de trabajar con una copia mejorando al mismo tiempo el consumo de memoria [Stroustrup97].

Otra característica importante presente en la biblioteca es la posibilidad de atenuar las entradas según se desee, dando así una mayor flexibilidad al sistema. Este parámetro de los conjuntos difusos llamado peso puede variar entre 0 y 1 (1 por defecto), con lo cual se puede atenuar la entrada completamente, haciéndola invisible para el sistema.

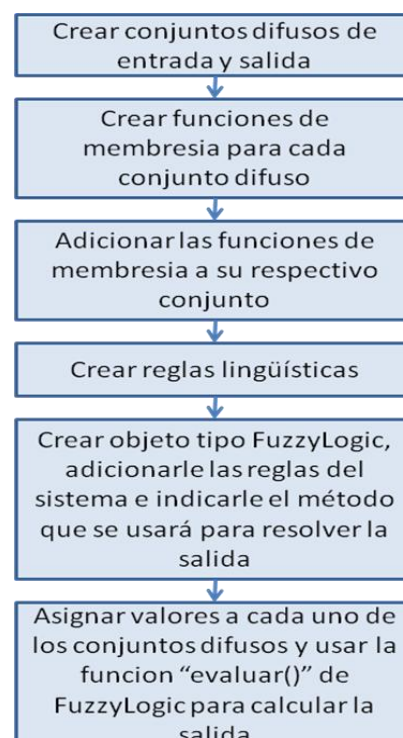


Figura 22: Metodología para la utilización de la biblioteca

La biblioteca permite la implementación de sistemas difusos de una forma bastante intuitiva, ya que solo se requieren conocimientos básicos en lógica difusa para utilizarla. En la figura 22 se presenta la metodología que se debe seguir para implementar un sistema difuso en la biblioteca. Si se desea realizar más de un cálculo se debe repetir solo el último paso, indicando los nuevos valores de los conjuntos difusos.

2.4. CONCLUSIONES

En este capítulo se presentó la implementación de un sistema de navegación reactivo el cual está compuesto por dos elementos: habilidades automáticas y coordinador reactivo basado en lógica difusa.

El coordinador reactivo está dividido en dos partes: una que se encargara de las acciones reflejas y que usa un esquema competitivo por prioridad y otra que esta a cargo de las habilidades automáticas, basado en un esquema cooperativo y un sistema difuso.

Las habilidades implementadas para la capa reactiva deben ser inicializadas con parámetros que van de acuerdo al comportamiento que se desea obtener. Estos parámetros fueron calibrados en forma heurística de acuerdo a los resultados obtenidos en las diferentes pruebas realizadas. Estas pruebas han consistido en navegación aleatoria por diferentes lugares (Para las habilidades de evasión y velocidad constante), lo que permite observar el desempeño de las habilidades en diferentes situaciones y da criterios para modificar sus parámetros. La habilidad de seguimiento de trayectoria se ha parametrizado en base a experimentos usando diferentes configuraciones, tomando los que se considera tuvieron un mejor comportamiento.

La lógica difusa es una buena alternativa para sistemas reactivos, pero se hace necesario el uso de bibliotecas que faciliten su implementación. Las bibliotecas que se investigaron al momento de decidir cual se iba a usar para la implementación de estos sistemas no eran adecuadas para el proyecto dado que algunas eran demasiado complejas[FuzzyLite11], estaban escritas para otros lenguajes diferentes a C++[jFuzzyLogic11], hacían uso de lenguajes de scripting para la definición de sus sistemas o no tenían como objetivo permitir el desarrollo de sistemas difusos embebidos [jFuzzyQt11]. La biblioteca desarrollada fue hecha a medida para el proyecto, por lo que se ajusta a los requerimientos y brinda mucha flexibilidad. A pesar de esto, puede ser usada para definir muchos otros tipos de sistemas difusos, fuera del área para la que fue diseñada.

Dado que en los coordinadores cooperativos se está procesando la salida de todas las habilidades activas, no se van a presentar las transiciones bruscas entre habilidades, las cuales se tenían en los coordinadores por prioridad, esto hace que se generen salidas más suaves.

CAPITULO 3

SISTEMA DE NAVEGACIÓN HIBRIDO BASADO EN LA ARQUITECTURA AUTOMÁTICA DELIBERATIVA: NHAD

Los problemas que enfrentamos actualmente no pueden resolverse con el mismo nivel de pensamiento que teníamos cuando los creamos
Albert Einstein

Actuar es fácil, pensar es difícil; actuar según se piensa es aún más difícil
Johann Wolfgang Goethe

En el capítulo 2 se presento el sistema de navegación reactivo basado en lógica difusa (NRLD). En este capítulo se presenta el sistema hibrido que usa el NRLD como uno de sus bloques para ejecutar tareas de más alto nivel. Se usa como base para la integración la arquitectura de control automática deliberativa [Barber00]. También se presentan las herramientas usadas para el control y monitoreo de las tareas que se le asignan al robot.

3.1. INTEGRACIÓN DE LAS CAPAS REACTIVA Y DELIBERATIVA

El sistema híbrido basado en la arquitectura AutomáticaDeliberativa desarrollado está compuesto de un nivel automático y un nivel deliberativo. En el nivel automático se encuentra el NRLD, el cual fue descrito en el capítulo 2, y que está compuesto por un coordinador reactivo basado en lógica difusa, habilidades automáticas y acciones reflejas. En el nivel deliberativo se encuentran el secuenciador, las diferentes habilidades deliberativas y la memoria a largo plazo. Se puede observar un esquema general de la arquitectura implementada en la figura 22.

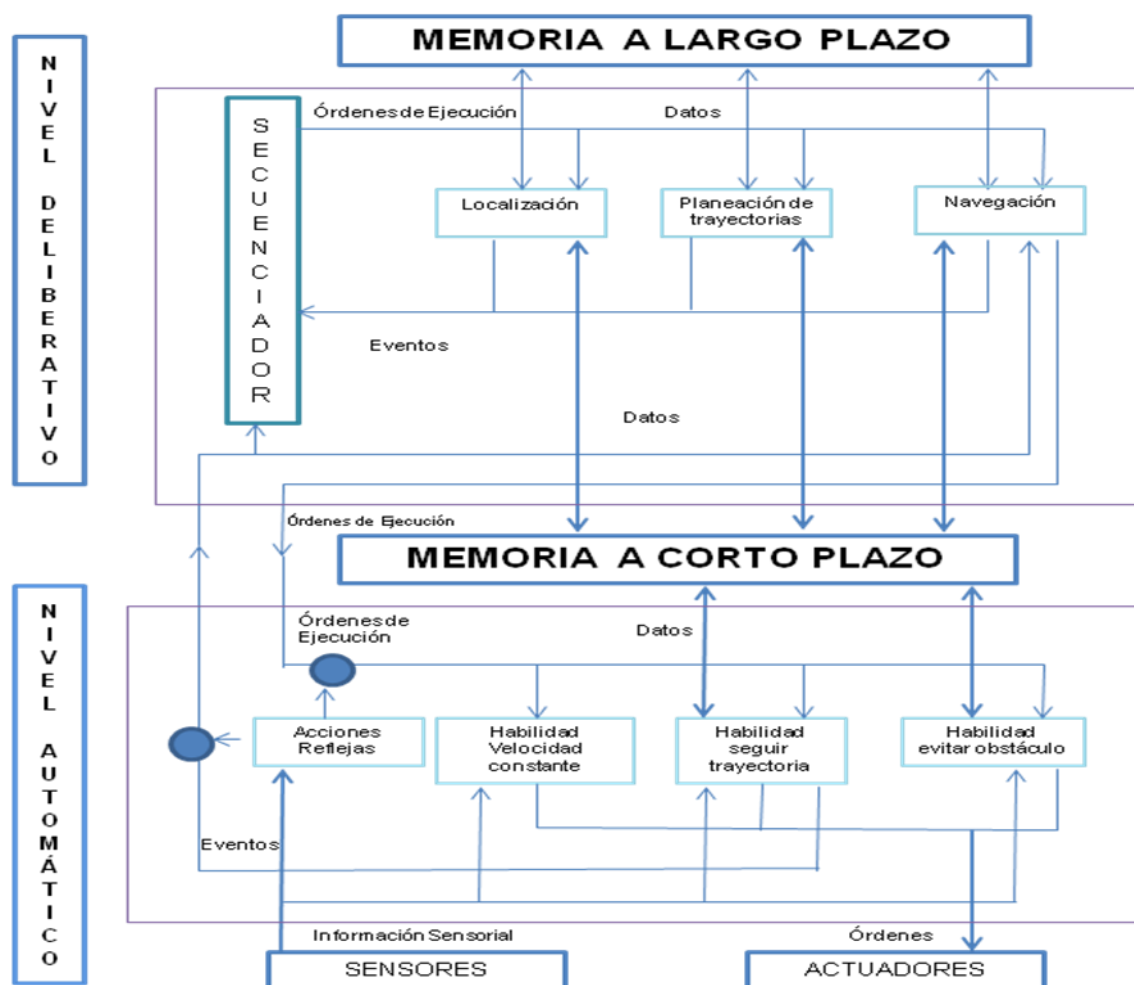


Figura 23: Arquitectura automática deliberativa aplicada

Todas las habilidades deliberativas hacen uso de la memoria a largo plazo, donde está almacenado el mapa, para cumplir con sus tareas. El mapa fue resultado de un proyecto de grado anterior, donde se mapeo el segundo piso de la escuela de ingeniería eléctrica y electrónica de la universidad del Valle. Este mapa fue obtenido por medio de una función que brinda ARNL para la realización de esta tarea. Se realiza un recorrido guiado del robot por todo el entorno a mapear y así se va generando el mapa. Se añadieron posteriormente al mapa obstáculos virtuales para indicar zonas prohibidas fuera del alcance de los sensores, lo cual se hizo por medio

del software de MobileRobots Mapper3. Este programa permite crear mapas geométricos desde cero, cargarlos desde el robot, adicionar zonas objetivo, prohibidas o casas entre otras.

Este mapa es cargado en una zona de memoria que se denomina memoria a largo plazo. En esta memoria se encuentran datos que solo pueden ser accedidos directamente por habilidades deliberativas y que se considera no cambian con frecuencia. Los parámetros en esta memoria generalmente son dados a priori.

El nivel automático se comunica con el nivel deliberativo de dos formas:

- Manejo de eventos: Se envían notificaciones a las habilidades deliberativas de que algún evento ha tenido lugar. Se tiene el ejemplo de la habilidad automática de seguimiento de trayectoria, que notifica a la habilidad deliberativa de navegación que ha terminado su recorrido. Esto se hace por medio de la llamada a una función que modifica una variable que indica si esto ha sucedido y que la habilidad deliberativa chequea frecuentemente. Los eventos generados pueden ser procesados tanto por el secuenciador como por las habilidades deliberativas.
- Memoria a corto plazo: En esta memoria se almacenan los datos dinámicos tales como la velocidad, orientación, posición, etc. Estos parámetros son actualizados automáticamente de forma periódica, pero puede hacerse una actualización cuando se desee, como en el caso de la habilidad deliberativa de localización, la cual modifica estos parámetros para corregir errores de odometría en la posición. La habilidad de planeación de trayectorias requiere conocer también la posición actual del robot en el mapa, por lo que accede a esta memoria para recuperarlos. Las habilidades automáticas pueden también leer estos parámetros e indirectamente modificarlos (los parámetros cambian cuando las habilidades activas ocasionan que la plataforma se mueva). La habilidad automática de seguimiento de trayectoria accede constantemente a esta memoria para medir el progreso de la tarea que está tratando de llevar a cabo. Este medio de comunicación, a diferencia del manejo de eventos, es bidireccional.

Las habilidades deliberativas tienen otra forma de comunicarse con las habilidades automáticas, mediante el uso de comandos de movimiento. En la arquitectura implementada estos comandos de movimiento están representados por diferentes configuraciones del NRLD. Si se desea realizar navegación aleatoria las habilidades y pesos usados en el nivel automático han de ser diferentes a los que se requieren para otra tarea, tal como seguimiento de trayectoria. Las habilidades y los pesos que se deben usar para llevar a cabo una tarea dada vienen dados a priori, calibrados basándose en resultados de pruebas realizadas para cada tarea con parámetros que arrojaban buenos resultados. Actualmente solo se modifican los pesos de las habilidades, con lo que se pueden activar y desactivar según se requiera, pero es posible también cambiar en tiempo de ejecución los parámetros de configuración de cada una de las habilidades. Estos comandos de movimiento son unidireccionales, desde el nivel deliberativo hacia el nivel automático. La única habilidad deliberativa

que puede enviar actualmente comandos de movimiento a las habilidades automáticas es la de navegación, por lo que cuando la plataforma se detiene, sigue una trayectoria o realiza una tarea que implique cambios en la dinámica de la plataforma está actuando esta habilidad.

Se planteó inicialmente el desarrollo de un coordinador difuso para la capa deliberativa, pero la naturaleza de la arquitectura, más específicamente del secuenciador de habilidades deliberativas, el cual realiza las tareas de árbitro en este nivel, impide que sea implementado con alguna técnica cooperativa, como lo es la lógica difusa. Ya que se deseaba continuar con la arquitectura automática deliberativa, para conservar la posibilidad de una futura integración a RobWeb-UV [Calvache09], se optó por usar un secuenciador clásico. El secuenciador implementado coordina la ejecución de las diferentes habilidades deliberativas para llevar a cabo una tarea, donde todas las tareas que han sido implementadas implican navegación. Por ejemplo si se desea ir a algún objetivo en el mapa desde el punto inicial el secuenciador ejecuta secuencialmente los siguientes pasos: detenerse – ubicar objetivo – relocalizar – planear trayectoria – seguir trayectoria hasta llegar a la meta – detenerse, donde cada uno de estos pasos se puede asociar a una habilidad deliberativa.

Sin capa deliberativa no habría planeación, pero sin nivel automático no habría movimiento. Con esto se puede ver la dependencia que tiene la capa deliberativa con la capa reactiva en esta arquitectura. De la misma forma se evidencia que la capa reactiva no necesita del nivel deliberativo para desplazarse de forma segura en el espacio de trabajo, pero es incapaz de alcanzar por si sola un objetivo de forma eficiente.

3.2. CAPA DELIBERATIVA

En la arquitectura automática deliberativa la capa deliberativa está compuesta por el secuenciador, la memoria a largo plazo y las habilidades deliberativas[Barber00].

Las habilidades deliberativas usan la información presente en la memoria a largo plazo (mapa) y a corto plazo (posición actual, orientación, velocidad, etc) para realizar sus tareas. Las habilidades deliberativas con las que se cuenta actualmente son localización, planeación de trayectoria y navegación, implementadas con ayuda de la API ARNL [Aria11], la cual provee un conjunto de funciones para facilitar esta tarea.

3.2.1. Localización

La localización es fundamental en cualquier aplicación que implique navegación, ya que si no se está localizado no es posible realizar cálculos respecto al objetivo al que se quiere llegar.

El algoritmo usado por la biblioteca ARNL es el MCL (Monte Carlo Localization) [Dellaert99]. Lo que hace este algoritmo es una estimación de la posición tomando como base las posiciones conocidas anteriormente. Dado que en el inicio del experimento no se tienen posiciones conocidas válidas, es necesario realizar la

localización cerca de un área inicial conocida como “casa” (área verde en la figura 20), lo cual le da al modelo de localización condiciones iniciales. Esta estimación consiste en crear un vector de posiciones aleatorias alrededor del área inicial. La probabilidad de que el robot se encuentre en una posición dada depende de la lectura del sensor laser. La posición que se tomará como correcta será la que tenga asociada una probabilidad mayor, la cual se calcula con cada lectura del sensor. Una vez se haya realizado esta estimación inicial se realiza una relocalización periódica (ejecutada en un hilo independiente) para corregir errores de odometría.

Los parámetros que se pueden modificar para el algoritmo de localización son (Tabla 2):

- Numero de muestras para el método de localización (2000)
- Numero de muestras para la localización inicial (4000)
- Ajustar numero de muestras (false)
- Mínimo score para que una localización se considere valida (0,2)
- Usar fusión sensorial – Filtro Kalman (true)

Cada vez que se ejecuta el algoritmo de localización, se genera una puntuación de acuerdo con el resultado obtenido que varía entre 0 y 1. De acuerdo a la configuración por defecto de ARNL 0,2 es la puntuación mínima para una localización valida, aunque una puntuación inferior a 0,6 está generalmente relacionada con malas localizaciones según las pruebas realizadas.

La posición corregida es puesta en la memoria a corto plazo modificando los valores que se corresponden con la posición actual del robot. Una vez se tiene el robot localizado es posible realizar tareas de navegación, como ir a puntos clave definidos previamente en el mapa. Cuando se le da la orden de ir a cierto punto, se realiza una planeación de trayectoria y el seguimiento de la misma, lo cual corresponde a otra habilidad deliberativa.

3.2.2. Planeación de trayectorias

En la figura 24 se observa la trayectoria planeada para ir desde la posición inicial (casa) hasta la meta1 (plataforma previamente localizada). Es posible ir de un punto a otro sin ningún tipo de planeación, solo usando técnicas reactivas, pero es poco probable que se llegue al destino y si se llega no se hará en un tiempo óptimo.

La planeación puede ser llevada a cabo en cualquier momento durante el experimento, ya sea mientras el robot está detenido, mientras está realizando navegación aleatoria o siguiendo otra trayectoria.



Figura 24: Planeación de una trayectoria

La planeación de trayectorias calcula sobre el mapa del entorno de trabajo una ruta segura y optima en distancia desde la posición actual hasta un destino dado. En el mapa se tienen obstáculos virtuales, que evitan que el robot planifique rutas por esas zonas. En la figura 25 se puede apreciar un obstáculo virtual insertado en el mapa a priori con el fin de evitar que el robot trate de navegar por unas escaleras que se encuentran en esta zona, las cuales no pueden ser detectadas por los sensores, ya que se encuentran fuera del rango de medición.

El mapa se obtiene de la memoria a largo plazo, memoria que almacena datos que no cambian constantemente y son dados a priori para el experimento. La información acerca de la posición actual del robot se recupera de la memoria a corto plazo, la cual almacena datos que cambian con mucha frecuencia (posición, velocidad, etc).

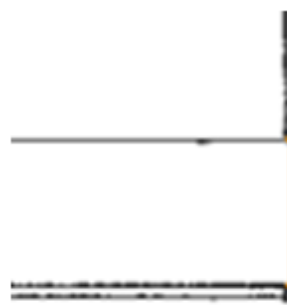


Figura 25: Obstáculo virtual

La planeación se realiza usando un método basado en celdas de ocupación [ARIA11]. Este método consiste en la división del mapa en celdas de un tamaño dado. Una vez se tiene esto, si la celda está completamente vacía se considera navegable. En caso contrario se subdivide la celda y se repite el proceso. Esto se realiza hasta que las subdivisiones han alcanzado un tamaño mínimo predefinido. Una vez se tienen las

zonas navegables, se traza la ruta tomando como parámetro la distancia más corta al objetivo.

En el algoritmo de planeación de trayectorias es posible configurar el tamaño mínimo de la celda de ocupación, 100mm por defecto.

Una vez se tiene la trayectoria planeada se puede recurrir a la habilidad de navegación para realizar el seguimiento de la misma. Esta habilidad deliberativa usa el esquema de evasión de obstáculos de la figura 26 y la habilidad automática “seguimiento de trayectoria” para alcanzar el objetivo, lo que le permite realizar un seguimiento con la capacidad de los evadir obstáculos no mapeados que se encuentren durante el recorrido. Se usan cuatro habilidades de evasión de obstáculos, dos laterales y dos frontales, de las cuales un par pertenece al coordinador difuso y el otro al coordinador por prioridad.

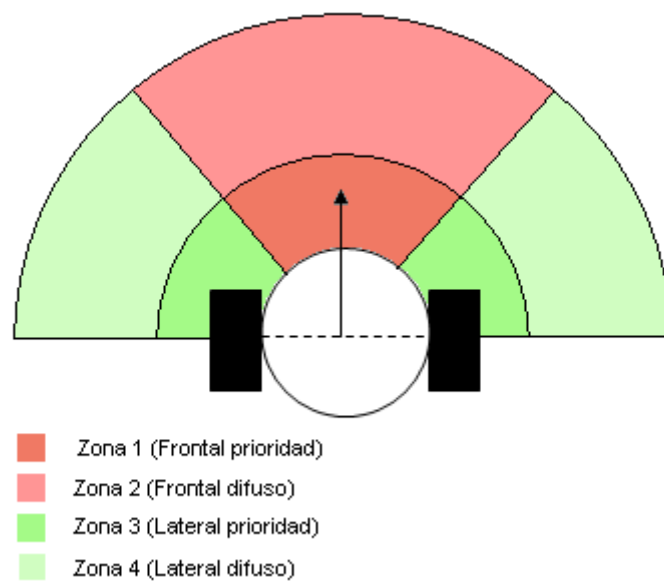


Figura 26: Configuración de las habilidades evadir obstáculo

El giro y la reducción de velocidad en las acciones reflejas en el coordinador por prioridad son mayores respecto a las otras dos, porque son el último recurso para proteger la integridad de la plataforma.

Se prefiere este esquema para la evasión de obstáculos dado que genera trayectorias más suaves. Esto se debe a que las evasiones laterales no detienen la plataforma completamente, solo disminuyen su velocidad, cosa que no sucede en el caso en el que se tiene solo una habilidad de evasión para todo el rango.

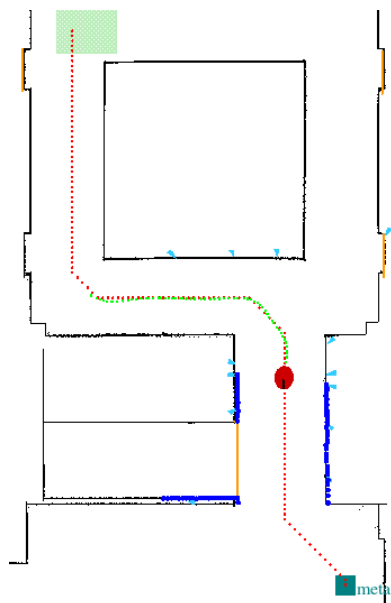


Figura 27: Robot siguiendo trayectoria

En la figura 27 se puede apreciar el robot tratando de seguir una trayectoria (serie de puntos rojos) y la trayectoria real recorrida (serie de puntos verdes) usando este esquema.

3.3. INTERFAZ PARA EL CONTROL REMOTO DE LA PLATAFORMA Y MOBILE EYES

Como mecanismo de interacción con el robot se ha desarrollado una interfaz de usuario sencilla e intuitiva, con la cual es posible darle órdenes. Las ordenes implementadas actualmente son detenerse, wander, localizar y planear y seguir trayectoria. Se implementaron estas tareas porque son bastante comunes en navegación y son útiles para validar la integración realizada. Esta interfaz se puede apreciar en la figura 28.

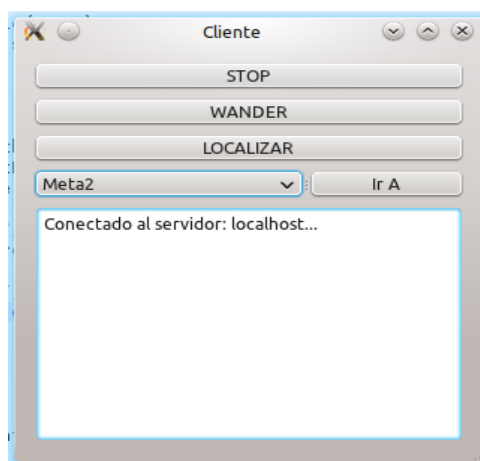


Figura 28: Interfaz para el control remoto de la plataforma

La interfaz se implementa basándose en un modelo cliente – servidor, haciendo uso de los sockets de ARIA (ArSocket) [Aria11] para la comunicación y de la biblioteca Qt [Qt11] para la parte gráfica. Estas bibliotecas usadas para el desarrollo del cliente son multi-plataforma, por lo que pueden ser usadas tanto en entornos Linux como Windows, variando solo su apariencia.

Se tiene un contenedor, que es la ventana donde han sido colocados todos los elementos de la interfaz. Hay 4 botones, cada uno asociado a un callback diferente que envía los comandos al servidor, con datos acerca de la tarea. Estos datos son pasados al secuenciador, el cual se encarga de procesarlos y activar las diferentes habilidades deliberativas según se requiera. Los botones son:

- Stop: Con este botón se envía la petición de parada al robot. No se detiene el servidor ni se apaga el laser, solo se suspende el movimiento de la plataforma.
- Wander: Se envía la petición para realizar navegación aleatoria con la plataforma.
- Localizar: Se trata de localizar la plataforma. Para que esta operación tenga éxito se requiere que el robot este cerca de un área de inicio denominada casa debido al algoritmo de localización usado.
- Ir A: Este botón toma el objetivo que se ha marcado en la lista que está a su lado y envía la orden de planear y seguir trayectoria hacia esta posición. Esta lista contiene todos los objetivos presentes en el mapa.

Hay un área de texto que sirve para indicar al usuario cual fue el último comando que se envió al robot.

Al inicio del experimento se debe establecer la conexión entre el servidor y el cliente. El socket tipo servidor presente en la plataforma bloquea el hilo principal y queda a la espera de conexiones entrantes, por lo que no se podrá continuar con la ejecución del programa sin un cliente. Una vez se ha establecido la conexión, el cliente envía los diferentes comandos hacia el servidor, los cuales representan acciones de alto nivel. El servidor recibe los comandos, los interpreta e inicia la ejecución de la tarea. En caso de que se reciba un nuevo comando, se aborta la tarea actual y se inicia la nueva.

El cliente envía de forma periódica datos al servidor aun cuando no se haya presionado uno de sus botones para indicarle al servidor que no se ha perdido la conexión. El lado servidor aborta la conexión y el experimento cuando no ha recibido ningún dato por cierto tiempo (configurable, 30s por defecto).

Este cliente no provee ningún tipo de retroalimentación acerca del estado actual del experimento, solo se limita a enviar los comandos de movimiento.

Para el monitoreo del experimento se ha de usar MobileEyes, un cliente diseñado por MobileRobots para esta tarea. MobileEyes se conecta al robot y automáticamente descarga el mapa que está usando (si usa uno y el servicio está habilitado), además de que da información acerca de las lecturas de los sensores del robot. Se puede mostrar más o menos información en este cliente dependiendo de los servicios que se añadan al servidor de ArNetworking.

Para usar estos servicios se debe crear un objeto `ArServerBase`, que será pasado como argumento a cada función que se desea añadir. Las funciones usadas en el proyecto son:

- `ArServerInfoRobotserverInfoRobot (&server, &robot)`: Esta función permite visualizar información acerca del estado actual del robot como la posición, velocidad, etc.
- `ArServerInfoSensorsserverInfoSensor (&server, &robot)`: Envía la lectura de los sensores al cliente para que sean dibujadas.
- `ArServerHandlerMapserverMap (&server, &arMap)`: Permite la descarga del mapa por parte del cliente.

Otros servicios adicionales fueron implementados para este proyecto:

- `parthDraw(ArServerClient* client, ArNetPacket* requestPkt)`: envía al cliente que ha sido pasado como parámetro datos acerca de la trayectoria planeada, en caso de que exista. Se usa UDP para el envío de esta información.
- `robotPathDraw(ArServerClient* client, ArNetPacket* requestPkt)`: envía al cliente información acerca de la trayectoria recorrida por el robot durante el experimento. Dado que no es conveniente dibujar la trayectoria que se ha recorrido desde el inicio del experimento porque se tendería a saturar la red por el exceso de información que se trata de enviar, solo se grafican los últimos 60 puntos tomados cada 400ms, mostrando así información acerca de los últimos 24 segundos del experimento. En caso de que se deseen obtener los datos completos acerca del experimento no solo para visualizarlos sino también para realizar otro tipo de análisis, en el servidor se almacena esta información en un archivo llamado "Log.txt". Los datos en este archivo pueden ser directamente importados a programas como MATLAB.

En `MobileEyes` la plataforma es representada por un círculo rojo y una línea negra, donde la línea indica su orientación. Las lecturas de los sensores se indican por medio de puntos azules, los diferentes objetivos son representados por cuadrados, que indican los puntos destino a los que se puede dirigir la plataforma. El área inicial o casa es dibujada como un rectángulo verde de aproximadamente 80cm x 80cm (aunque realmente se toma como casa el centro de este rectángulo, por lo que el tamaño de este no es importante. Lo mismo aplica para los objetivos). Aparte de esto se dibujan también en ocasiones series de puntos rojos y verdes (los puntos rojos solo se dibujan si se ha planeado una trayectoria, de lo contrario solo serán dibujados los puntos verdes), donde los puntos rojos representan la trayectoria planeada entre la posición actual del robot y alguna meta mientras que los puntos verdes hacen referencia a la trayectoria recorrida por la plataforma. En la figura 29 se pueden apreciar estos datos.

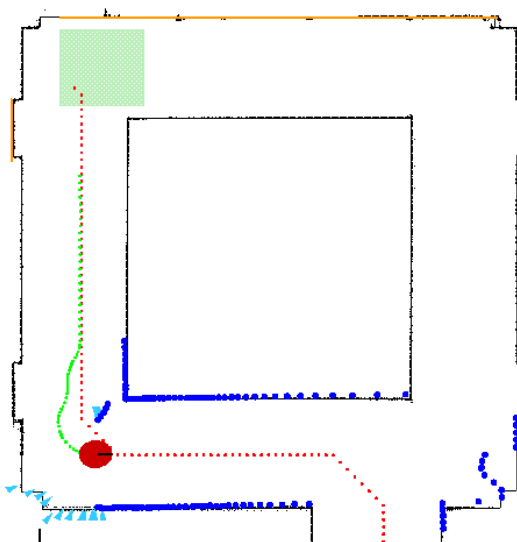


Figura 29: Resultados arrojados por MobileEyes

3.4. CONCLUSIONES

En este capítulo se presentó la implementación de una capa deliberativa para el sistema de navegación así como su integración con el nivel automático, tomando como base el esquema dado por la arquitectura de control automática deliberativa [Barber00]. El nivel deliberativo desarrollado consta esencialmente de tres módulos: habilidades deliberativas (planeación y localización), secuenciador de dichas habilidades y memoria a largo plazo (donde se almacena el mapa).

El esquema de localización utilizado (MonteCarlo Localization) requiere que la plataforma este cerca de un área inicial (casa) al principio del experimento. Los resultados obtenidos con este algoritmo son bastante satisfactorios lo cual se debe en gran parte a la calidad de las lecturas arrojadas por el sensor laser.

En el proyecto se hace uso un mapa geométrico, el cual es almacenado en la memoria a largo plazo (que contiene datos que no cambian frecuentemente y generalmente vienen dados a priori) y fue obtenido mediante un recorrido guiado. A este mapa se le han adicionado obstáculos virtuales para evitar que la plataforma trate de navegar por lugares que representan riesgos, como por ejemplo las escaleras. Sin embargo si la plataforma no se encuentra localizada, los obstáculos pierden su utilidad. Esto pone en evidencia la total dependencia del sistema con la etapa de localización. Una mala localización afecta también la planeación y el seguimiento de trayectorias, dando como resultado experimentos fallidos y riesgosos.

Se planteo inicialmente el desarrollo de un coordinador difuso para la capa deliberativa, pero debido a que el proyecto está basado en la arquitectura de control automática deliberativa, fue necesaria la implementación de un secuenciador clásico, ya que de acuerdo a su definición este bloque no es compatible con técnicas cooperativas como lo es la lógica difusa. Este esquema permite realizar la integración

de las capas reactiva y deliberativa sin alterar la concepción de la arquitectura en la que el proyecto está basado.

La integración del nivel deliberativo (secuenciador, memoria a largo plazo, habilidades de localización, planeación de trayectorias entre otras) y el nivel automático (habilidades automáticas, acciones reflejas, coordinador reactivo basado en técnicas difusas) bajo la arquitectura de control automática deliberativa da lugar a un sistema modular, en el cual la funcionalidad de cada bloque es fácil de extender, incrementando así también la funcionalidad del sistema total.

CAPITULO 4

PRUEBAS Y RESULTADOS

En este capítulo se presentan las diferentes pruebas realizadas. Se presentan inicialmente las pruebas del sistema de localización usado para el proyecto. También se evalúa el sistema de navegación basado en la arquitectura de control automática deliberativa usando técnicas difusas y se compara con un esquema clásico por prioridad, teniendo en cuenta parámetros como suavidad de la trayectoria generada, error durante el recorrido y tiempo requerido para su finalización. Se compara también el uso de recursos en ambos sistemas. Se hace uso del cliente remoto MobileEyes para facilitar la visualización.

Finalmente se realiza la validación de la biblioteca para la implementación de sistemas difusos desarrolla para este proyecto, comparándola con resultados arrojados por el toolbox de lógica difusa de MATLAB.

4.1. VALIDACIÓN DEL SISTEMA DE NAVEGACIÓN DELIBERATIVA BASADO EN LÓGICA DIFUSA

Las pruebas realizadas tuvieron como objetivo evaluar el desempeño del sistema, por lo cual se presentan resultados de cada una de las habilidades deliberativas. Primero se evalúa el desempeño del sistema de localización presentando resultados obtenidos en diferentes situaciones. También se evalúa el sistema de planeación y seguimiento de trayectorias, realizando comparaciones entre el sistema difuso, un sistema por prioridad y un sistema cooperativo. Las aproximaciones que se usaran para realizar la comparación usaran el coordinador por defecto de ARIA (ArPriorityResolver).

Para las pruebas realizadas se ha usado el coordinador difuso por defecto, ya que es el que se desea validar. La casa y las metas en el mapa usado para las pruebas tienen un tamaño por defecto, dado por el software usado para crearlos (Mapper3);

4.1.1. Localización

El objetivo de esta prueba es validar la habilidad deliberativa de localización. Se va a posicionar el robot en diferentes puntos del mapa y se ejecuta el algoritmo de localización. Se realiza la prueba a diferentes distancias relativas al centro de la casa (área verde en la figura 30, usada para el inicio de los experimentos) tomando como parámetro la puntuación arrojada del algoritmo para la localización. Esta puntuación es calculada de acuerdo a como se ajusten las lecturas del sensor a las distancias esperadas y varía entre 0 y 1, siendo 1 la mejor calificación. De acuerdo con los valores por defecto de configuración de ARNL, un valor de 0.2 es el mínimo para que una posición sea considerada válida, sin embargo de acuerdo a las pruebas realizadas ninguna localización con una puntuación cuyo valor se encuentre por debajo de 0.6 es satisfactoria.

Para cada una de las posiciones se toman diez medidas, de las cuales se calcula la media y la desviación estándar, que permiten medir la repetitividad del algoritmo.

Al inicio del experimento se tiene un robot no localizado, tal como se ve en la figura 30, donde las lecturas del laser no concuerdan con la posición donde el robot cree que está.

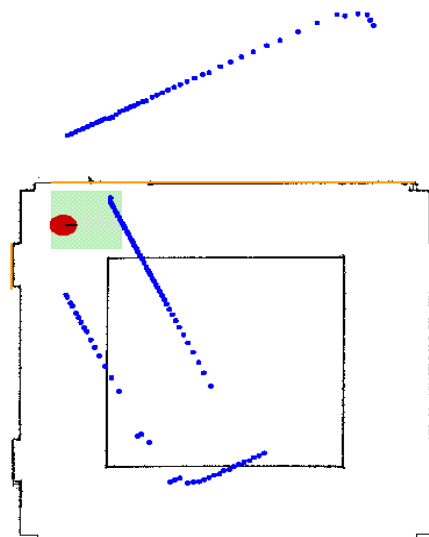


Figura 30: Robot no localizado

Tras aplicar el algoritmo de localización se corrige la posición del robot, obteniendo así el resultado mostrado en la figura 30. Este caso presenta una situación donde la plataforma está cerca del área inicial requerida para la localización.

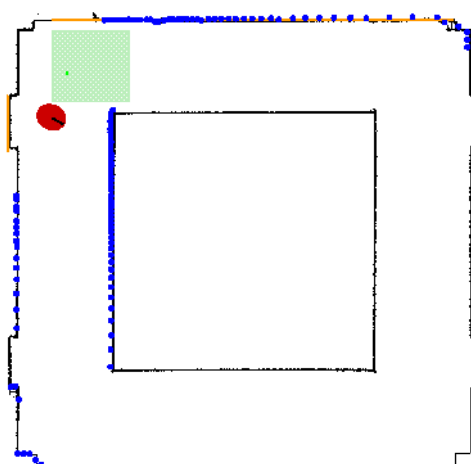


Figura 31: Robot localizado (una puntuación mayor a 0.6)

Aun cuando el robot no se encuentre dentro del área casa es posible que logre localizarse con éxito, sin embargo la probabilidad de que lo logre disminuye con la distancia a esta área.

Se presentan en la figura 32 las diferentes posiciones en el mapa usadas para probar la localización. Se tiene en cuenta para cada posición la distancia respecto al centro de la casa.

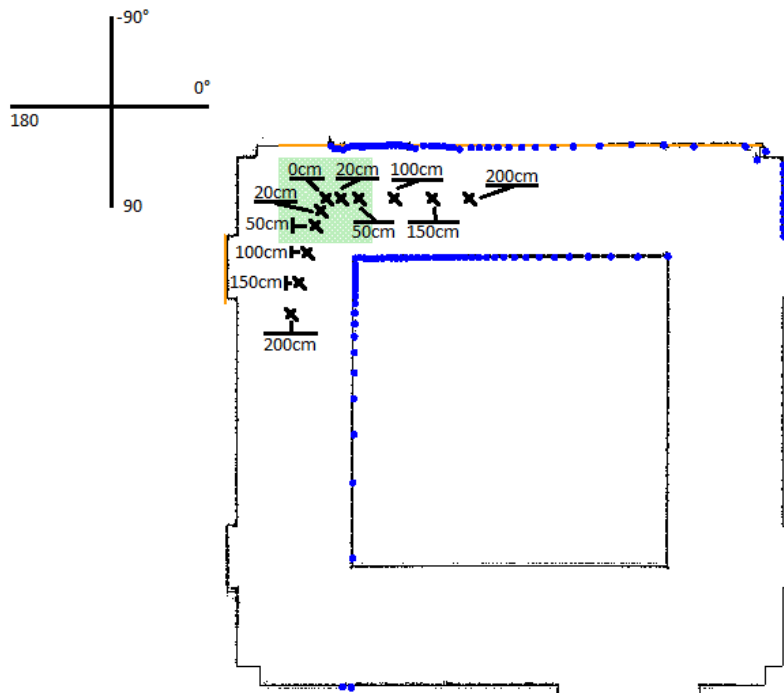


Figura 32: Posiciones usadas para las pruebas realizadas

Para cada una de las posiciones se realizaron 4 pruebas, modificando la orientación del robot (0, 90, -90, 180), realizando así 44 pruebas, tomando 10 resultados en cada una (puntuaciones). La síntesis de las pruebas se presenta en la tabla 8.

Experimento	0°		90°		-90°		180°	
	Media	SD	Media	SD	Media	SD	Media	SD
Centro de la casa	0.7317	0.04206	0.5917	0.20512	0.7574	0.08768	0.4839	0.0606
20cm prueba 1	0.7448	0.4262	0.63	0.20319	0.8155	0.13058	0.537	0.10063
20cm prueba 2	0.7829	0.06172	0.7262	0.23912	0.7447	0.08532	0.4414	0.02584
50cm prueba 1	0.7328	0.04786	0.5543	0.2214	0.7525	0.07213	0.4725	0.1745
50cm prueba 2	0.7585	0.06284	0.6309	0.17777	0.8078	0.03861	0.4956	0.01098
100cm prueba 1	0.794	0.0852	0.6408	0.1006	0.707	0.0446	0.4836	0.04262
100cm prueba 2	0.7575	0.0481	0.6327	0.13087	0.7675	0.0768	0.6435	0.01661
150cm prueba 1°	0.7442	0.08178	0.7214	0.1124	0.6583	0.0643	0.5433	0.1264
150cm prueba 2	0.7482	0.06367	0.691	0.09764	0.7664	0.06973	0.70928	0.05708
200cm prueba 1	0.7101	0.0327	0.691	0.08843	0.7543	0.07323	0.4416	0.1825
200cm prueba 2	0.6889	0.0291	0.7487	0.12119	0.7785	0.07134	0.7563	0.02881

Tabla 8: Resultados pruebas localización (SD: Desviación estándar)

Según las pruebas realizadas, se puede ver que el sistema de localización depende fuertemente del área denominada casa. El algoritmo trata de encontrar un lugar en el mapa cercano al área denominada “casa” que se ajuste a las lecturas del sensor laser, por lo que a medida que se aleja de la misma el número de posibles lugares que pueda confundir aumenta. En la figura 33 se presenta uno de estos casos, donde el algoritmo encuentra una posición que corresponde bastante bien con la lectura del laser, pero el robot no se encuentra realmente allí (físicamente se encuentra en el lugar marcado por la “X”). A pesar de esto se puede apreciar que arroja una puntuación alta (0.709), por lo que se puede concluir que una puntuación alta no garantiza una buena localización, y esto se puede ver más claramente a medida que la plataforma se aleja del área de inicio. Estos resultados se deben en parte al área de experimentación.

Muchas de las medias en la tabla 8 son superiores a 0.6, pero en algunas se tiene una desviación estándar alta, lo que indica que en las pruebas el algoritmo oscilaba entre varias posibles localizaciones.

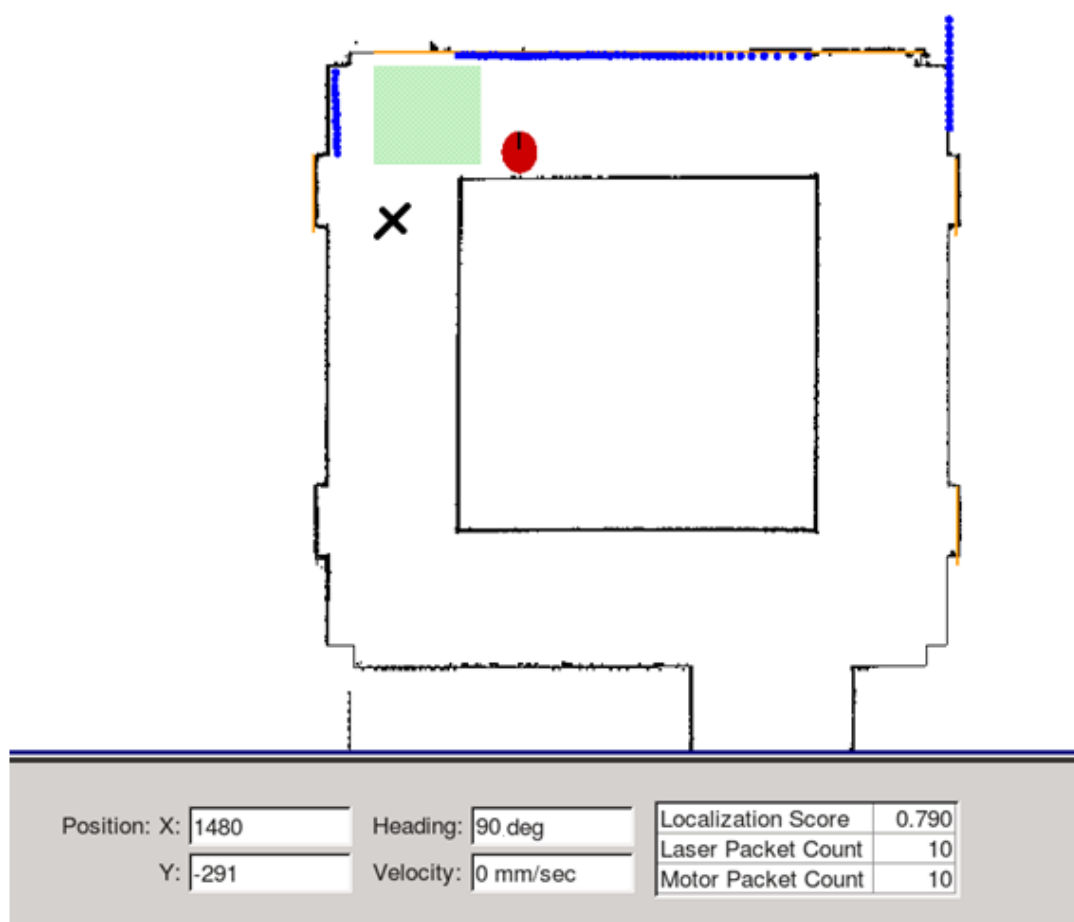


Figura 33: Localización errada con alta puntuación

A pesar de este problema que se presenta con el algoritmo, se puede concluir que si se encuentra cerca de la casa la localización es generalmente bastante buena. La localización inicial es muy importante y, una vez realizada correctamente es bastante probable que la plataforma no se pierda durante el resto del experimento. Esto se logra con la creación de un hilo alternativo al programa principal, el cual realiza

periódicamente una relocalización para corregir errores que se puedan generar durante los desplazamientos del robot.

El algoritmo de localización inicial tarda entre 1 y 2 segundos aproximadamente en arrojar el resultado, tiempo tolerable teniendo en cuenta la tarea que está desempeñando y que es una operación que se realiza al inicio del experimento.

Según las pruebas realizadas solo hay una condición que se debe cumplir para obtener una localización exitosa: que el robot se encuentre a una distancia menor a 150cm respecto al centro del área denominada como casa, distancia a la que a pesar de fallar en ocasiones, es posible que se localice. Esta medida puede variar cambiando el entorno de experimentación, pero se debe tener en cuenta que a medida que se aleja la plataforma de la casa aumenta la probabilidad de una localización errónea.

4.1.2. Seguimiento de trayectoria

El objetivo de la prueba de seguimiento de trayectoria es validar la propuesta de navegación deliberativa basada en lógica difusa a la vez que se realiza una comparación con un sistema por prioridad y un sistema cooperativo.

La prueba para evaluar el seguimiento de trayectoria consiste en añadir un par de obstáculos que no están en el mapa (por lo que la trayectoria planeada no los tuvo en cuenta) a la trayectoria y evaluar su respuesta ante estos eventos fuera del plan. También se realizan pruebas sin obstáculos. Además de esto se realizó una prueba para evaluar la respuesta de los sistemas en un pasillo.

Para las pruebas se hacen algunas consideraciones:

- El entorno de trabajo se considera correctamente mapeado.
- Los obstáculos que no aparecen en él se consideran dinámicos (obstáculos móviles que no hacen parte del entorno, como por ejemplo personas), por lo que no se realiza un mapeo de los mismos. La trayectoria es planeada considerando que estos obstáculos no existen, y son evadidos a medida que se presentan en el camino.

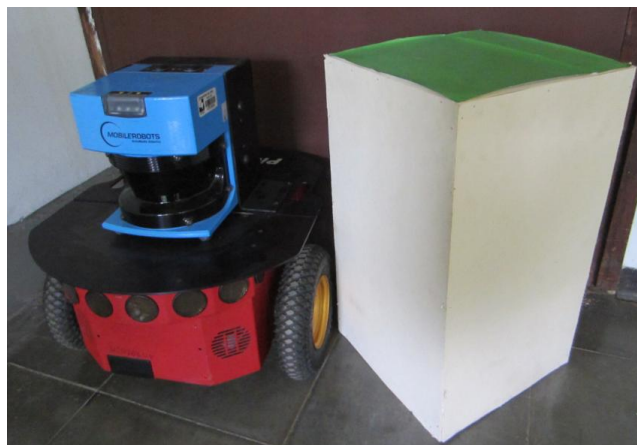


Figura 34: Obstáculos usados en la prueba

Los obstáculos usados en las pruebas son detectados por ambos sensores de distancia (sonares y laser). En la figura 34 se presenta uno de los obstáculos al lado del robot, donde es posible verificar que puede ser detectado por los dos sensores. También se encuentran en el área de pruebas obstáculos con forma cilíndrica.

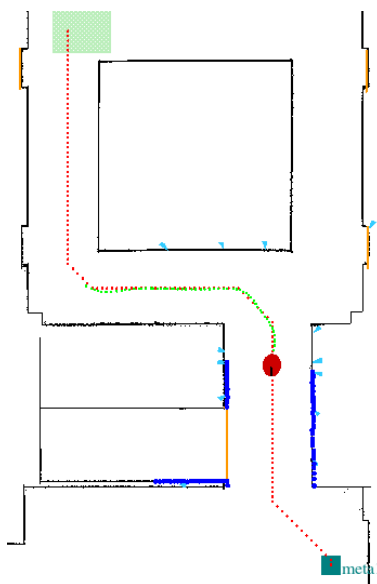


Figura 35: Seguimiento de trayectoria para las pruebas

Se realizaron diversas pruebas para validar la implementación realizada. Para lograr este propósito se configuraron dos sistemas más, un sistema por prioridad y un sistema cooperativo, ambos con parámetros similares a los usados por el sistema difuso desarrollado.

El sistema por prioridad y el sistema cooperativo fueron implementados configurando el coordinador por defecto de ARIA, `ArPriorityResolver`. Este coordinador toma el segundo parámetro usado para añadir un comportamiento al robot (`ArRobot::AddAction(ArAction, int)`) como la prioridad. En caso de que haya más de un

comportamiento con la misma prioridad, se calcula un promedio de sus salidas dependiendo de un parámetro llamado Strength, diferente para cada ArAction y cada canal (velocidad traslacional, orientación relativa, etc) [ARIA11]. El valor de Strength varía entre 0 y 1, por defecto es 1.

Para el coordinador por prioridad no es importante modificar ningún parámetro “Strength” en los comportamientos, ya que basta con usar prioridades diferentes para cada uno. Para el coordinador cooperativo este parámetro (Strength) han de ser tenido en cuenta, ya que modifica la forma en que se promedia la salida de los comportamientos con la misma prioridad (entre más alto sea este valor más influenciara la salida).

Se presentan en la tabla 9 los diferentes parámetros usados para la configuración de cada una de las habilidades en todos los sistemas (difuso, por prioridad y cooperativo). Estos parámetros fueron calibrados de forma heurística en una serie de pruebas que se realizó con este fin. Se decidió usarlos porque responden adecuadamente en cualquiera de los tres sistemas, permitiendo así que el experimento se enfoque en los cambios que implica en la salida el uso de un coordinador diferente.

Habilidad	Distancia activación	Angulo de evasión	Velocidad de evasión	Rango de evasión	Peso (sistema difuso)	Prioridad (sistema por prioridad)	Prioridad – Strength (sistema cooperativo)
Evasión frontal 1	15cm	30°	0mm/s	0° - 40°	-100	100	100 – 0.7
Evasión frontal 2	70cm	30°	100mm/s	0° - 40°	100	99	98 – 0.7
Evasión lateral 1	10cm	12°	150mm/s	40° - 90°	-99	98	99 – 0.7
Evasión lateral 2	30cm	12°	200mm/s	40° - 90°	80	97	98 – 0.7
		L	Velocidad seguimiento				
Seguimiento de trayectorias		30cm	300mm/s		70	80	98 – 0.3

Tabla 9: Parámetros de configuración usados para las pruebas

La primera prueba consistió en un seguimiento de trayectoria sin obstáculos (figura 34) con ambos sistemas, de donde se calculó el error entre la trayectoria planeada y la recorrida. Los resultados de esta comparación se presentan en la figura 35.

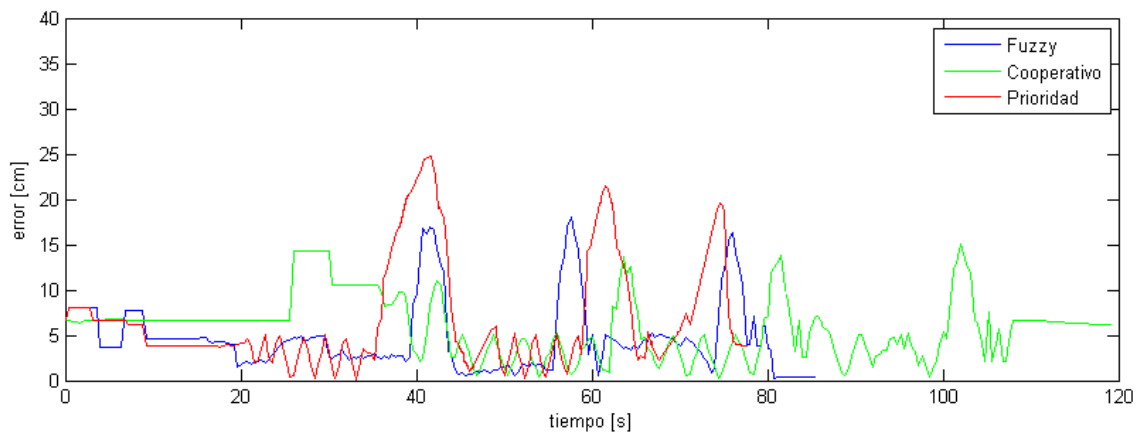


Figura 36: Error trayectoria recorrida – trayectoria planeada para el experimento de Seguimiento de trayectoria sin obstáculos

El error máximo encontrado en la prueba realizada para el seguimiento de trayectorias sin obstáculos, fue de 18cm para el sistema difuso y de 27cm para el sistema por prioridad. Este error se presenta mientras durante el seguimiento de una curva en la trayectoria.

Las pruebas arrojaron los resultados esperados, mostrando las ventajas de la aproximación difusa. Se realizaron varias pruebas con cada uno de los coordinadores, de las cuales se calcularon valores promedio de su error máximo, error promedio, desviación estándar y tiempo. Los resultados de estas pruebas se pueden observar en la tabla 10.

EXPERIMENTO	Error Máximo [mm]	Error promedio [mm]	Desviación Estándar [mm]	Tiempo [s]
Difuso Sin Obstáculos	179.895	46.428	38.887	86
Prioridad Sin Obstáculos	268.815	70.491	69.607	76.8
Cooperativo Sin Obstáculos	150.01	58.82	33.96	119.6
Difuso con Obstáculos	258.871	66.263	60.059	90.133
Prioridad con Obstáculos	399.333	100.885	92.058	93.2
Cooperativo con Obstáculos	377.093	78.879	68.71	138.433

Tabla 10: Resultados

Los experimentos realizados permiten comparar el desempeño del sistema difuso con los de un sistema por prioridad y un sistema cooperativo. Los errores máximos, promedio y la desviación estándar son significativamente menores en la aproximación difusa respecto al sistema por prioridad, lo que significa que el seguimiento de trayectoria es mejor que en el coordinador por prioridad, aun con obstáculos durante el recorrido. Respecto al sistema cooperativo, el error máximo y el error promedio es menor en el sistema difuso (cuando hay presentes obstáculos), sin embargo la desviación estándar en ambos sistemas es similar, debido a que la forma en que combinan los comportamientos da lugar a salidas más suaves.

El error máximo en el coordinador difuso es de aproximadamente 40% menos que en el coordinador por prioridad si hay presentes obstáculos y 33% menor si no los hay. Respecto a el coordinador cooperativo el error máximo es 33% más si hay obstáculos en la trayectoria y 16.2% mayor si no los hay. Sin embargo el error promedio es en ambos casos inferior.

El tiempo para realizar el recorrido es aproximadamente el mismo en el sistema difuso y por prioridad si hay obstáculos presentes, pero es significativamente menor en el sistema por prioridad si no los hay. El tiempo de recorrido del sistema por prioridad es mucho mayor a las otras dos aproximaciones en ambos casos. Si se configura la habilidad de seguimiento de trayectoria para que realice su tarea a mayor velocidad de forma que se compense la disminución que sufre en los coordinadores cooperativos podría mejorarse esto, sin embargo el objetivo de la prueba es validar el funcionamiento del coordinador y compararlo con otros sistemas usando los mismos comportamientos.

Los datos sirven también para verificar la suavidad del recorrido: Una desviación estándar alta corresponde a un seguimiento bastante oscilatorio (la desviación estándar es una medida de dispersión respecto a la media). El sistema difuso arrojó en todos los casos una trayectoria suave (véase tabla 10 para resultados numéricos de las pruebas realizadas), lo cual se evidencia en la figura 37 donde se presenta la aceleración durante el recorrido de ambas aproximaciones. En el peor de los casos el número de oscilaciones en el coordinador difuso es igual que en el coordinador por prioridad, sin embargo la magnitud de estas es mucho menor. En el caso del sistema cooperativo implementado, se tienen oscilaciones de menor magnitud, pero también un seguimiento mucho más lento.

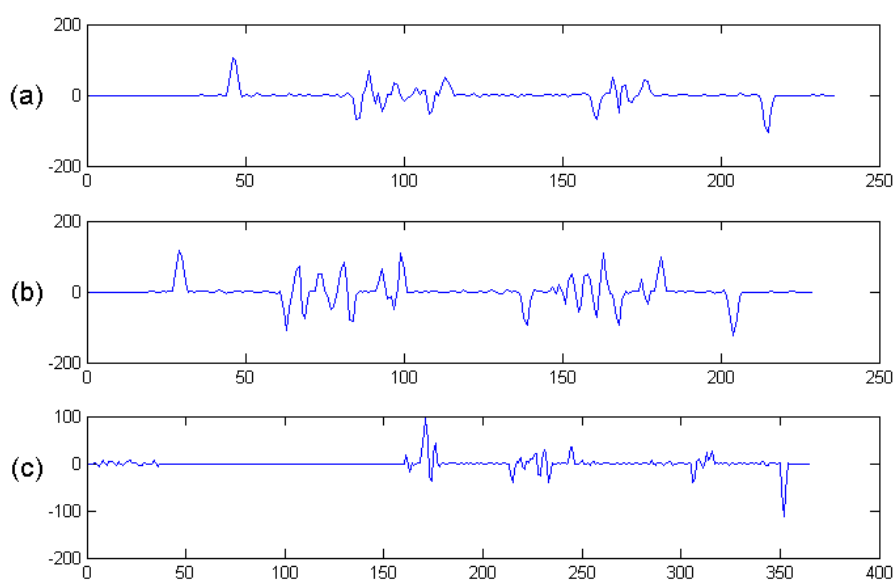


Figura 37: Aceleración (mm/s^2) durante el recorrido
Coordinador difuso (a) – Coordinador por prioridad (b) – Coordinador cooperativo (c)

Se realizó una prueba para evaluar la respuesta de los sistemas en un pasillo, donde se evidencia una debilidad de los coordinadores por prioridad. Si el ancho del pasillo es menor que el rango de sus habilidades de evasión, podría ser incapaz de dar un giro de 180 grados (para las pruebas la evasión lateral tuvo un rango máximo de 30cm y la frontal de 70cm). Esto se evidencia en la figura 36, donde el robot se aleja continuamente del objetivo a alcanzar (área verde) describiendo oscilaciones en su trayectoria, las cuales se dan en los momentos en los que se intenta hacer un giro sin éxito hacia el objetivo.

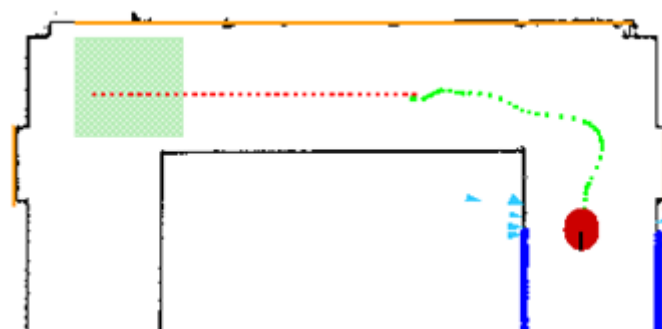


Figura 38: giro de 180° usando un coordinador por prioridad

Este mismo caso especial se aborda usando un coordinador difuso (El coordinador cooperativo también es capaz de superar este obstáculo). En esta ocasión la plataforma es capaz de realizar el giro aun cuando el rango de sus habilidades de evasión es mayor que el ancho del pasillo. Esto se puede observar en la figura 38.

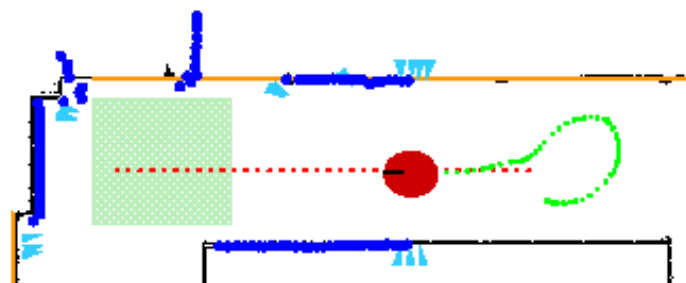


Figura 39: Giro de 180° en pasillo usando coordinador difuso

Según las pruebas realizadas, el sistema difuso respondió de forma adecuada en todos los casos. El tiempo que le toma a este sistema el recorrer una trayectoria es mayor que el requerido por el sistema por prioridad en el caso en el que no se tengan obstáculos no mapeados, sin embargo el error de seguimiento es significativamente menor. Comparado al sistema cooperativo los errores de seguimiento son significativamente menores y la suavidad de la trayectoria que genera el sistema difuso no es inferior.

4.1.3. Costo computacional

Se realizaron pruebas sobre el computador embebido en la plataforma móvil con el fin de comparar el costo computacional generado por el sistema de navegación difuso con el generado por el sistema de navegación basado en prioridad y el sistema de navegación basado en un coordinador cooperativo.

La herramienta usada para este análisis es el gnome-system-monitor, monitor del sistema por defecto del gnome (Entorno de escritorio Linux). Cuando el monitor está en funcionamiento el uso de CPU aumenta entre un 5-10%. Además de esto no permite realizar acercamientos a las graficas que presenta.

En la figura 40 se presenta el costo computacional del sistema difuso. Se observa al inicio el uso de CPU por parte del algoritmo de localización (57.3%). El uso de memoria es bastante bajo, permanece prácticamente constante tanto antes como después del inicio de la aplicación. El uso de CPU es bastante alto cuando se están llevando a cabo movimientos (85% - 100%), ya que se está procesando una gran cantidad de información (relocalización, coordinación de habilidades automáticas, tareas de sincronización, entre otras).

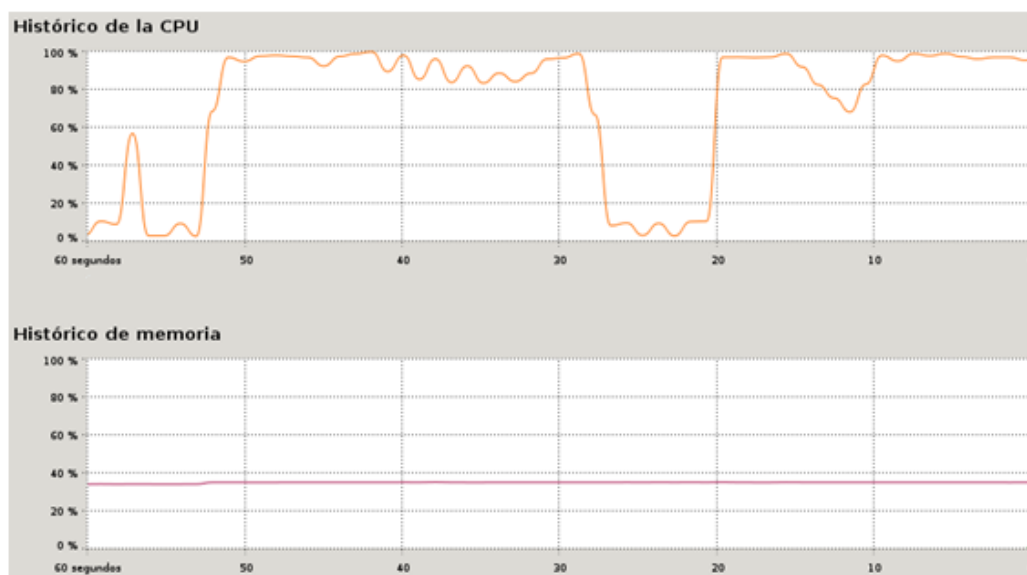


Figura 40: Costo computacional del sistema de navegación difuso

En la figura 41 se muestra la carga computacional de un experimento con el sistema de navegación por prioridad, donde se puede observar que el consumo de memoria es también bastante bajo. El consumo de CPU es bastante alto conservándose casi al 100% cuando esta activo el coordinador por prioridad.

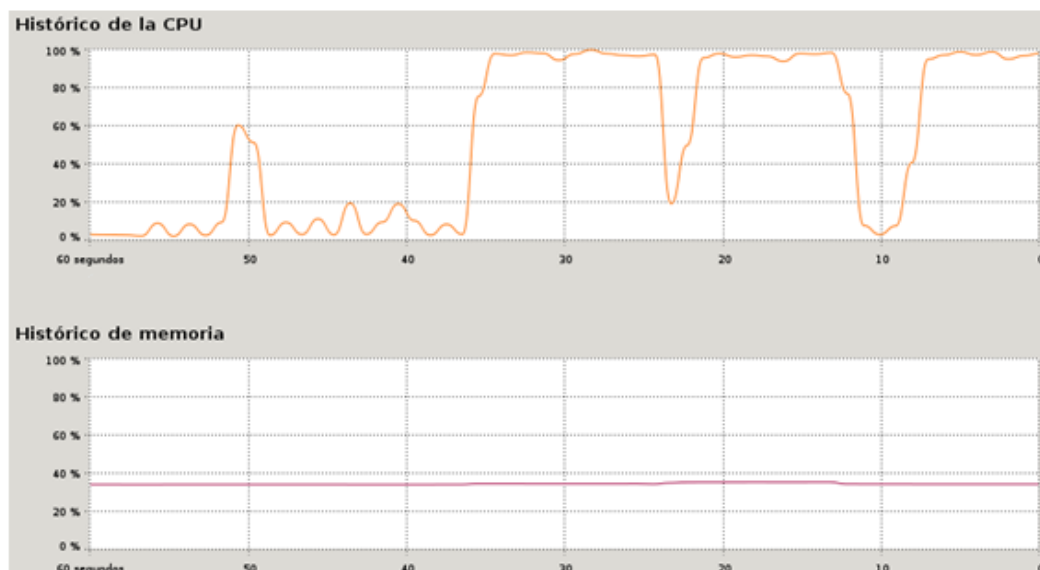


Figura 41: Costo computacional del sistema de navegación por prioridad

Por último se presenta en la figura 42 el grafico correspondiente al costo computacional del sistema de navegación basado en un coordinador cooperativo. Al igual que en los casos anteriores, cuando se están ejecutando tareas el consumo de CPU esta al máximo.



Figura 42: Costo computacional del sistema de navegación cooperativo

A pesar de que el consumo de memoria no se puede apreciar en detalle, se puede observar que permanece prácticamente constante. La memoria presente en la plataforma es de 512MB, bastante pequeña, por lo que es importante evitar fugas de memoria para no degradar el desempeño del sistema. El uso de memoria en todos los

sistemas esta alrededor del 38%, esto incluye también la memoria que requiere el sistema operativo de la plataforma (Debian).

Al comparar el costo computacional de todos los sistemas es posible observar que los recursos utilizados por cada uno de ellos son bastante similares. Estos resultados muestran que el esquema basado en lógica difusa es una propuesta de navegación viable respecto a otros métodos de navegación, ya que no utiliza más recursos y genera resultados satisfactorios.

4.2. BIBLIOTECA PARA LA IMPLEMENTACION DE SISTEMAS DIFUSOS

Con el objetivo de validar el correcto funcionamiento y los índices de error de la biblioteca para la implementación de sistemas difusos desarrollada, se realiza una comparación entre dicha biblioteca y el toolbox de lógica difusa de MATLAB. Es importante conocer estos índices de error para tener una idea más clara acerca de la calidad de los sistemas difusos generados usando esta biblioteca: entre menores índices de error se tengan, mejor se considera su desempeño.

La prueba consiste en la implementación en ambas herramientas de un mismo sistema difuso. El sistema usado para la validación es un ejemplo clásico en lógica difusa [Matlab11], el problema de la propina, la cual se calcula dependiendo de la calidad de la comida y el servicio. Se usa este ejemplo porque permite mostrar que se puede usar la biblioteca para implementar sistemas fuera del área de la robótica.

Se realizan diferentes pruebas, con los mismos valores de entrada al sistema pero aumentando el número de muestras usados para definir la salida en la biblioteca. La salida es representada por una serie de puntos (muestras), con los que se puede aproximar a una serie de rectángulos de los cuales es posible calcular el centroide del sistema, método de resolución usado. A mayor número de puntos, se espera una salida con menor índice de error.

En la figura 43 se presentan las diferentes funciones de membresía del sistema, tanto para la entrada como para la salida. Las entradas del sistema son servicio (malo, regular y bueno) y comida (mala y deliciosa), y la salida es la propina (poca, media y alta).

Las reglas lingüísticas del sistema son:

- Si el servicio es malo o (OR) la comida es mala entonces la propina es poca.
- Si el servicio es regular entonces la comida es media.
- Si el servicio es bueno o (OR) la comida es deliciosa entonces la propina es alta.

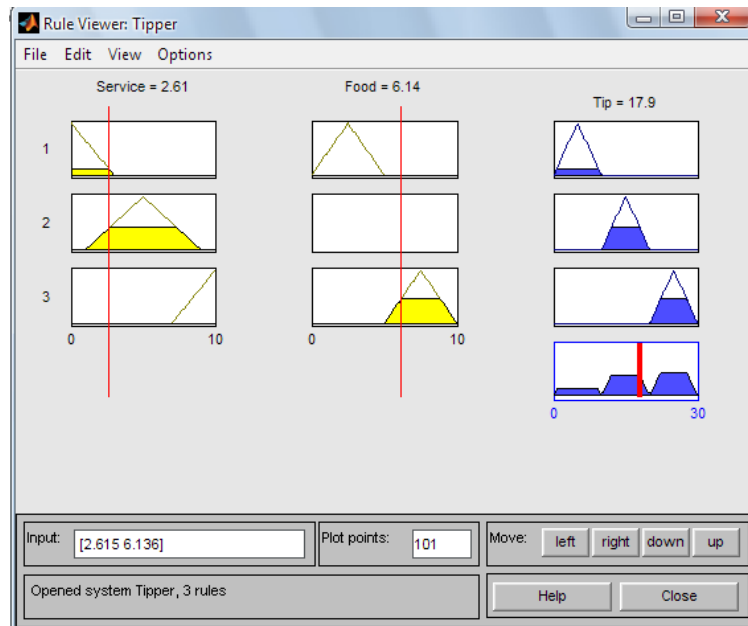


Figura 43: Sistema difuso en el toolbox de lógica difusa de MATLAB [Matlab11].

La implementación usando ambas herramientas arrojó resultados bastante similares, acordes con lo que se esperaba. En la tabla 11 se presentan los resultados de las diferentes pruebas en las cuales se incrementó gradualmente el número de muestras usadas por la biblioteca para la definición del conjunto difuso de salida. El error máximo (considerando como referencia los resultados arrojados por MATLAB) fue de 0.78%. El margen de error es tolerable teniendo en cuenta el tipo de aplicaciones a las que se enfoca la lógica difusa.

Prueba	Servicio	Comida	Propina MATLAB	Propina Biblioteca para la implementación de sistemas difusos					
				50 muestras	Error (%)	100 Muestras	Error (%)	200 muestras	Error (%)
1	2,615	6,136	17,9096	17,852	0,32161522	17,896	0,07593693	17,9089	0,003908518
2	1,055	9,318	12,0507	11,956	0,78584647	12,0274	0,19334976	12,0474	0,027384301
3	4,908	5,136	15,9554	15,8951	0,37792848	15,9349	0,12848315	15,9511	0,026950123
4	9,312	8,864	24,9981	24,9552	0,17161304	24,9889	0,0368028	24,997	0,004400334
5	0,6881	0,773	5,0019	5,0086	0,1339491	5,0018	0,00199924	5,0018	0,00199924

Tabla 11: Validación biblioteca para la implementación de sistemas difusos

Se optó por usar solo 50 muestras para el desarrollo del sistema difuso utilizado en este proyecto dado que aumentar demasiado el número de muestras para obtener mejor precisión traerá consigo también una mayor carga computacional, lo que afectará el rendimiento de todo el sistema.

Usando los resultados de MATLAB como referencia, las pruebas fueron satisfactorias, con un error máximo inferior al 1%. La biblioteca desarrollada es válida como herramienta para el desarrollo de sistemas difusos en C++, permitiendo diseños flexibles y modulares.

4.3. CONCLUSIONES

Se evaluó el sistema de localización usado para el proyecto, con resultados satisfactorios. Su limitación más importante es que requiere que la plataforma se encuentre en un área específica al empezar el experimento para realizar la localización inicial, pero se puede concluir basándose en los resultados de las pruebas realizadas que la plataforma se localiza correctamente en la mayoría de los casos si se encuentra dentro de esta área de inicio (casa). Una vez se ha realizado dicha localización es posible navegar sin ningún problema, ya que los errores de odometría serán corregidos automáticamente por este sistema.

El seguimiento de trayectoria realizado por el coordinador difuso fue superior en velocidad comparado con las otras dos aproximaciones (prioridad y competitivo) cuando se introdujeron obstáculos en el camino. Cuando se realizó el seguimiento sin obstáculos el sistema por prioridad fue más rápido. El error de seguimiento fue menor en todos los casos en la aproximación difusa. La desviación estándar, asociada a la suavidad de la trayectoria, fue similar en los sistemas cooperativo y difuso, pero mucho mayor en el sistema por prioridad (trayectoria más brusca).

El costo computacional asociado a todos los sistemas (difuso, por prioridad y cooperativo) presentó índices de uso de recursos similares. Estos aspectos muestran que la propuesta de navegación deliberativa basada en lógica difusa presentada es válida, eficiente y confiable para el control de una plataforma móvil. Con este desarrollo se amplían las opciones a la hora de decidir que esquema de navegación utilizar para futuras aplicaciones.

La biblioteca para la implementación de sistemas difusos muestra resultados satisfactorios con un error máximo de 0.78% respecto al toolbox de lógica difusa de MATLAB. Estos resultados permiten concluir que la biblioteca implementada presenta un correcto funcionamiento y es válida para la implementación de sistemas difusos no solo en el área de la robótica.

CAPITULO 5

CONCLUSIONES, OBSERVACIONES GENERALES Y APORTES FUTUROS

5.1. CONCLUSIONES Y OBSERVACIONES

Mediante el uso de coordinadores se pueden obtener esquemas de movimiento complejos. Estos coordinadores permiten usar más de una habilidad para reaccionar de forma más inteligente dependiendo de la situación. Una técnica bastante usada para su implementación es la lógica difusa, con bajos costos computacionales y que no requiere de modelos precisos para arrojar resultados válidos.

Dado que la arquitectura de control automática deliberativa se basa en la forma en la que razonan y actúan los seres humanos, seres cuyo razonamiento puede considerarse difuso, es natural considerar el uso de la lógica difusa para la implementación de esta arquitectura.

La biblioteca desarrollada para implementar los sistemas difusos se verificó por medio de una comparación de sus resultados con los arrojados por el toolbox de lógica difusa de MATLAB, obteniendo errores bastante bajos. Con esto es posible concluir que la biblioteca desarrollada es una herramienta válida para el desarrollo de sistemas difusos.

Los coordinadores desarrollados con técnicas difusas (cooperativos) son suficientemente eficientes como para ser considerados una propuesta válida para un sistema de coordinación reactivo, además las salidas que generan son generalmente más suaves que las arrojadas por coordinadores competitivos. Esta suavidad conlleva a menos cambios bruscos de velocidad y trayectorias más naturales como se pudo observar con las pruebas realizadas. Esto se logró con un uso de CPU y memoria similar en los tres sistemas (difuso, por prioridad y cooperativo).

De acuerdo a los resultados obtenidos en el proyecto se puede concluir que las técnicas de navegación basadas en lógica difusa son una alternativa válida para la coordinación de tareas de navegación en robótica móvil. Desarrollos como este amplían las opciones disponibles a la hora de elegir un esquema de navegación a utilizar.

De acuerdo con la concepción de la arquitectura automática deliberativa, el secuenciador de habilidades deliberativas, como su nombre lo indica, ordena la ejecución secuencial de habilidades deliberativas según lo requiera una tarea, en un orden establecido a priori. De acuerdo a esta definición no es posible realizar la implementación de un coordinador basado en técnicas difusas para la capa deliberativa sin realizar modificaciones a la arquitectura misma. Se prefirió entonces continuar con los parámetros dados por la arquitectura en lugar de modificarla, por lo cual se optó por la implementación de un secuenciador clásico.

El algoritmo de localización usado para el proyecto (MCL) debe encontrarse en el área casa o cerca de ella al momento de ejecutarse por primera vez, sin embargo se puede concluir, según las pruebas realizadas, que una vez se ha hecho esto la plataforma puede navegar por todo el entorno sin perder su localización.

Las habilidades deliberativas usadas en el proyecto se conciben de forma tal que modifiquen los pesos y parámetros de las habilidades automáticas en el coordinador reactivo basado en lógica difusa, modificando así la respuesta dinámica del sistema dependiendo de la tarea a realizar.

La estructura bajo la cual se realizó esta integración da lugar a un sistema modular, en el cual es fácil extender la funcionalidad de cada bloque, incrementando así también la funcionalidad del sistema total y abriendo las posibilidades a una integración futura con diferentes proyectos, como con el laboratorio de experimentación remota RobWeb-UV.

5.2. APORTES

El Proyecto deja varios aportes importante, no solo en el área de la robótica móvil. El principal aporte del proyecto es la implementación completa de una arquitectura de control basada en la arquitectura automática deliberativa y en técnicas difusas, diseñada de forma modular.

Algunos aportes particulares a destacar son:

Un cliente que se comunica con la plataforma usando un modelo cliente/servidor para controlar las tareas que se desea ejecutar. La biblioteca para la implementación de sistemas difusos y la implementación de un coordinador difuso basado en dicha biblioteca. Este coordinador es fácilmente configurable en caso de que no se desee usar el sistema por defecto.

Se aporta también una habilidad automática para seguimiento de trayectorias que fue usada durante todo el proyecto. La generación de trayectorias y la localización fueron realizadas con base en las herramientas provistas por ARNL, el código desarrollado puede servir de ejemplo o base si se desea usar estas capacidades en proyectos futuros.

5.3. TRABAJOS FUTUROS

Como trabajos futuros se proponen:

- Mejora de la biblioteca para la implementación de sistemas difusos. Es posible añadir nuevas funciones y métodos de resolución para incrementar su alcance.
- Añadir nuevas habilidades tanto a la capa reactiva como deliberativa, permitiendo así que se pueda llevar a cabo un mayor número de tareas.
- Incluir técnicas de mapeo dinámico.
- Integración funcional a ROBWEB Univalle, permitiendo incluir nuevas interfaces de experimentación.
- Desarrollo de una interfaz grafica para la configuración de las habilidades y el coordinador por parte del usuario.

BIBLIOGRAFIA

[ARIA11]Mobile Robots, Advanced Robotics Interface for Applications (ARIA).API documentation.<http://robots.mobilerobots.com> (2011)

[Arkin98]Arkin, Ronald. **Behaviour based robotics**.MIT Press, 1998.

[Barber00]Barber Castaño, Ramón. **Desarrollo de una Arquitectura para robots Móviles Autónomos. Aplicación a un Sistema de Navegación topológica**. Tesis Doctoral Universidad Carlos III de Madrid, 2000.

[Blanchette08]BlanchetteJasmin; Summerfield Mark.**C++ GUI Programming with Qt 4, Second Edition**. Prentice Hall, 2008.

[Calvache09]Calvache Vallejo, BayronAndres. **Diseño e implementación de una interfaz remota para la experimentación en robotica móvil**. Universidad del Valle – Cali, Colombia, 2009.

[Ching08] Ching-Chih Tsai, Shu-Min Shish, Hsu-Chih Huang, Min-Yu Wang, Chih Chung Lee.**Autonomous navigation of an indoor tour robot guide**. (2008).

[Dellaert99]Dellaert, Frank. Fox, Dieter. Burgard, Wolfram. Thrun, Sebastian. **Monte Carlo Localization for mobile robots**.Computer science department, Carnegie Mellon University, 1999.

[Donbing07] DonbingGu, Huosheng Hu. **Integration of coordination architecture and behaviour fuzzy learning in quadruped walking robots**. (2007).

[FuzzyLite]FuzzyLite. Página principal del proyecto consultada en el 2011 <http://code.google.com/p/fuzzy-lite/> (2011)

[jFuzzyLogic]jFuzzyLogic. Página principal del proyecto consultada en el 2011 <http://jfuzzylogic.sourceforge.net/html/index.html> (2011)

[jFuzzyQt]jFuzzyQt. Página principal del proyecto consultada en el 2011 <http://sourceforge.net/projects/jfuzzyqt/>

[Kaehler11]Kaehler, Steven. **Fuzzy logic tutorial**.<http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html> , consultada en el 2011.

[Kristijan08] KristijanMacek, Dizan Vasquez, Thierry Fraichard, Roland Siegwart. **Safe Vehicle Navigation in Dynamic Urban Scenarios**. (2008).

[Magnus99]Magnus Andersson, Anders Orebäck, Matthias Lindström, Henrik I. Christensen.**ISR: An Intelligent Service Robot**. (1999).

[Matlab11] MATLAB. Documentación del toolbox de lógica difusa. <http://www.mathworks.com/help/toolbox/fuzzy/>, consultada en el 2011.

[Murphy00] Murphy, Robin. **Introduction to AI robotics**.MIT Press, 2000.

[Qt11]Nokia, Cross plataform application and UI framework.API documentation.<http://qt.nokia.com> (2011)

[Raguraman09]Raguraman, S.M; Tamilselvi, D; Shivakumar, N.**Mobile robot navigation using Fuzzy logic controller**.Thiagarajar College of Eng., Madurai, India. (2009).

[ROS11]Robot Operating System.Página principal delproyecto. <http://www.ros.org/wiki/>, consultada en el 2011.

[Stroustrup97] Stroustrup, Bjarne. The C++ programming language – Special Edition.3rd Edition. (1997)

[Tsong-Li05]Tsong-Li Lee; Li-Chun Lai; Chia-Ju Wu.**A fuzzy algorithm for navigation of mobile robots in unknown environments**.Department of Automation Engineering, Nan Kai Coll., Nantou, Taiwan. (2005).

[Valdir06] ValdirGrassi, Sarangi P. Pahrik, Jun Okamoto.**Hybrid deliberative/reactive for human-robot interaction**.(2006).

[Wang09] Wang Jia, Wei Bo-yu, Ye Chen .**Design of an autonomous mobile robot for hospital**. (2009).

[Wefeng04] Wefeng Li, Henrik I. Christensen, Anders Orebäck. **An architecture for indoor navigation**. (2004).

[Yang03]Yang, X; Moallem, M; Patel, R.V.**An improved fuzzy logic based navigation system for mobile robots**.Department of Electricaland Computer Engineering.Western Ontario University. (2003).

[Yang05]Yang, X; Moallem, M; Patel, R.V.**A fuzzy logic-based reactive navigation algorithm for mobile robots**.Department of Electricaland Computer Engineering.Western Ontario University. (2005).

[Yunwei09] Dan Wang, YunweiJia, Zuoliang Cao.**A unique multifunctional landmark for autonomous navigation**.(2009).

[Zhang96]Zhang Jianwei, Wille Frank, Knoll Alois. **Modular design of fuzzy controller integrating deliberative and reactive strategies**.(1996).