

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

MAESTRÍA EN INGENIERÍA DE CONTROL Y AUTOMATIZACIÓN



**DISEÑO DE UN SISTEMA DE NAVEGACIÓN AUTÓNOMO PARA ROBOTS
MÓVILES USANDO FUSIÓN DE SENSORES Y CONTROLADORES
NEURO DIFUSOS**


Tesis para optar el Título de Master, que presenta:

Autor: Luis Miguel Enciso Salas

Asesor de Tesis: Dr. Antonio M. Morán Cárdenas

Lima, Mayo del 2015



A large, faint, circular watermark of the Pontificia Universidad Católica del Perú logo is centered in the background. It contains the text "ET LUX IN TENEBRIS LUCE" at the top and "MCMXVII" at the bottom, surrounding a central emblem of a ship and a star.

A mis padres quienes con su cariño, esfuerzi y coraje me inspiraron siempre a superarme y lograr mis metas, y a mis hermanos quienes siempre me han apoyado y guiado.

Agradecimientos.

Agradezco al Dr. Antonio Morán por sus consejos y enorme contribución al desarrollo de esta tesis y al Dr. Sotomayor por su invaluable apoyo y revisión de la misma.



Resumen

El objetivo principal de esta tesis fue el desarrollar un sistema de navegación autónomo para la aplicación en robots móviles, para ello se uso el enfoque reactivo por presentar una aproximación más real al problema del planeamiento.

El sistema de navegación se dividió en dos partes para fines del diseño; en la primera se desarrolló e implementó un controlador para el planeamiento de la trayectoria mediante el uso de controladores neuro-difusos y el uso de cuadrículas de certeza; mientras que en la segunda parte, se afrontó el problema de la localización del robot en un entorno desconocido, para lo cual se hizo uso de la fusión de sensores odométricos, inerciales y de redes inalámbricas.

Para este propósito se diseño e implementó el controlador para diferentes plataformas de simulación, además se implementó el mismo en un robot Pioneer P3-AT, lográndose resultados interesantes que permiten apreciar este algoritmo como una alternativa atractiva en la navegación autónoma de los robots móviles, el método presentado representa un híbrido entre los métodos de navegación por fusión de conductas y los métodos basados en campo de fuerzas por lo cual permite aprovechar las ventajas de ambos métodos.

Para la localización se implementó la fusión de sensores presentes en los celulares modernos y sensores de velocidad o encoders, la fusión de sensores se implementó en dos instancias, la primera para la estimación precisa de la orientación por medio de filtros de Kalman y la segunda en donde se realiza la estimación de las coordenadas de la posición con el uso de filtros de Kalman para sistemas no lineales.

Índice general

1. Estado del Arte de los Sistemas de Navegación de Robots Móviles Tipo Carro	3
1.1. Introducción	3
1.2. Descripción de los Robots Móviles Tipo Carro	6
1.3. Estado del Arte de los Sistemas de Navegación Autónoma para Robots	8
1.3.1. Estado de Arte de la Localización	9
1.3.2. Estado de Arte del Planeamiento de Rutas y Construcción de Mapas	14
1.4. Estado del Arte de los Algoritmos de Fusión de Sensores	19
1.5. Estado del Arte de los Controladores Neuro-difusos Aplicados a Robots Móviles	23
1.6. Objetivos de la Tesis	27
2. Análisis Cinemático de los Robots Móviles Tipo Carro	28
2.1. Introducción	28
2.2. Modelo Cinemático de los Robots Móviles Tipo Carro	28
2.3. Modelo Cinemático del Robot Móvil Tipo Skid-steer	31
2.3.1. Presentación del Robot Móvil Skid-steer	31
2.3.2. Análisis Cinemático del Robot Móvil Skid-steer	32
2.4. Presentación del Robot Móvil Pioneer P3-AT	34
2.5. Estructura de Control Planteada	36
3. Diseño del Sistema de Navegación para Robots Móviles Tipo Carro usando Controladores Neuro-difusos	37
3.1. Introducción	37
3.2. Fundamento de los Sistemas Neuro-difusos	38
3.2.1. Sistemas Neuro-difusos TS	39

3.3. Desarrollo del Sistema de Navegación basado en Controladores Neuro-difusos	41
3.3.1. Estructura del Sistema de Navegación	41
3.3.2. Construcción del Histograma de Cuadrículas de Certeza	42
3.3.3. Diseño del Controlador Neuro-difuso Goal Seeking	44
3.3.3.1. Controlador Goal Seeking con manejo en adelante	44
3.3.3.2. Controlador Goal Seeking con manejo en adelante y retroceso	45
3.3.4. Diseño de Conductas Obstacle Avoidance y Wall Following	47
3.3.4.1. Cuantificación de fuerzas en la cuadrícula	47
3.3.4.2. Controlador neuro difuso de evasión de obstáculos	49
3.3.4.3. Controlador neuro difuso de seguimiento de paredes	50
3.3.5. Diseño del Controlador Neuro-difuso para la Fusión de Conductas	52
3.4. Simulación del Sistema de Navegación para el Robot Móvil Tipo Carro	55
3.4.1. Simulación en MATLAB	55
3.4.2. Interfaz de Usuario en GUIDE	59
3.5. Simulación del Sistema de Navegación para el Robot P3-AT implementado en Stage	60
3.5.1. Introducción a Software Player/Stage	60
3.5.2. Simulación en Player/Stage	61
3.6. Comparación con otros Controladores	66
3.6.1. Comparación con un Controlador por Conductas	66
3.6.2. Comparación con Algoritmos VFF y VFH	70
3.7. Conclusiones	74
4. Diseño del Sistema de Fusión de Sensores	76
4.1. Introducción	76
4.2. Calibración de los Sensores	76
4.2.1. Sensores Inerciales (IMU)	76
4.2.1.1. Acelerómetro	77
4.2.1.2. Giroscopio	79
4.2.2. Magnetómetro	80
4.2.3. Aceleración Lineal	83

4.2.4. Sensores de Red Inalámbrica (WSN)	84
4.3. Fundamento de los Filtros de Kalman	85
4.3.1. Filtro de Kalman Discreto	85
4.3.2. Filtros de Kalman Extendido(EKF)	86
4.4. Diseño del Sistema de Localización usando Fusión de Sensores	88
4.4.1. Caracterización de los Sensores	88
4.4.1.1. Odometría	88
4.4.1.2. IMU	88
4.4.1.3. WSN	88
4.4.2. Estimación de la Orientación	89
4.4.2.1. Estimación de los ángulos de Euler	89
4.4.2.2. Diseño del filtro de Kalman	91
4.4.2.3. Ajuste en línea de la matriz de covarianza R	92
4.4.2.4. Pruebas del estimador de la orientación	93
4.4.3. Estimación de la Ubicación	95
4.4.3.1. Diseño del EKF	95
4.4.3.2. Pruebas del estimador por EKF	96
4.5. Conclusiones	104
5. Pruebas y Resultados del Sistema de Navegación	106
5.1. Introducción	106
5.2. Implementación del Sistema en el Robot P3-AT	106
5.3. Implementación del Código en C++ usando la Librería Aria	109
5.4. Ensayos con el Sistema de Navegación	111
5.5. Conclusiones preliminares	113
Conclusiones Generales	115
Bibliografía	116
A. Programas Usados para la Simulación	126
A.1. Simulación en Matlab	126
A.2. Simulación en Stage	130
A.3. Simulación de la localización de un robot móvil mediante EKF	132

B. Programas Usados en la Implementación	139
B.1. Aplicación en Android	139
B.2. Código para Arduino	144
B.3. Controlador implementado en C++	148



Índice de figuras

1.1. Robot Dante II.	
<i>Fuente:</i> http://www.frc.ri.cmu.edu/robots/robs/photos/1994_Dantell.jpg	4
1.2. Robot teleoperado Sojourner.	
<i>Fuente:</i> http://mars.jpl.nasa.gov/MPF/roverctrlnav/images/fur_corner.jpg	4
1.3. Robot subacuático SeaBotix.	
<i>Fuente:</i> http://www.seabotix.com/	4
1.4. Robot MagneBike.	
<i>Fuente:</i> http://www.asl.ethz.ch/research/asl/alstom	5
1.5. Robots terrestres, aéreos y acuáticos.	6
1.6. Estructura de los robots móviles tipo carro. [1]	8
1.7. Esquema general de localización [1]	9
1.8. Arquitectura de la navegación basada en conducta [1].	10
1.9. Arquitectura de la navegación basada en mapas.	10
1.10. Descomposición celular exacta[1].	11
1.11. Ejemplo de cuadrículas de ocupancia.	11
1.12. Diagrama de un sistema de localización con filtros de Kalman para robots móviles[1].	12
1.13. Diagrama de la localización por el método SLAM[2].	13
1.14. Navegación por marcas en forma de Z [3].	13
1.15. Navegación con balizas de referencia [1].	14
1.16. Gráficos de visibilidad[1].	15
1.17. Diagramas de Voronoi.	15
1.18. Descomposición en células aproximadas, izquierda: Método de células aproximadas, derecha: Método de células-variables aproximadas.	15
1.19. Exploración rápida de árboles aleatorios[1].	16
1.20. Esquema de un navegación por campos potenciales[2].	17
1.21. Conducta de para el seguimiento de paredes (wall-following)[1]	17

1.22. Navegación por histogramas de campos vectoriales[1].	18
1.23. Método de navegación bubble band[1].	18
1.24. Método de navegación de ventanas dinámicas[1].	19
1.25. Crecimiento de la incertidumbre en trayectoria recta[1].	20
1.26. Arquitectura centralizada de la fusión de sensores[4].	21
1.27. Arquitectura descentralizada de la fusión de sensores	22
1.28. Estructura básica de un controlador difuso para robots móviles[2].	23
1.29. Estructura de los neurocontroladores con mecanismo de aprendizaje [2].	24
1.30. Derecha: Diagrama del controlador neuro-difuso propuesto por Li para el control del robot móvil[5], Izquierda: Inclusión de un PID en la estructura de control[6].	25
1.31. Diagrama de bloques de la estructura propuesta por Rusu[7].	26
1.32. Estructura del neuro controlador propuesto por Khatoon [8].	26
1.33. Estructura propuesta por Song, un FKCN se usa en la fusión[9].	26
2.1. Estructura cinemática de un robot móvil tipo carro (adaptado de [2]).	29
2.2. Modelo del robot móvil tipo carro y su simplificación en modelo bicicleta[10].	30
2.3. Mecanismo de movimiento en los robots móviles skid-steer Source: http://www.robotplatform.com/knowledge/Classification_of_Robots/wheel_control_theory.html	31
2.4. Estructura cinemática de un robot móvil tipo skid-steer[11].	33
2.5. Robot móvil Pioneer P3-AT.[12]	35
2.6. Distribución de los sonares en el robot móvil P3-AT.[13]	35
2.7. Estructura del sistema de navegación planteado.	36
3.1. Estructura de una red neuro difusa tipo TS [14]	39
3.2. Mecanismos de inferencia[15].	39
3.3. Sistema neuro difuso con inferencia tipo 3[15].	40
3.4. Estructura global del sistema de navegación propuesto.	42
3.5. Actualización del histograma de cuadrículas mediante medidas de sonares[16].	43
3.6. Ventanas dinámica y semi-dinámica vistas en perspectiva	43
3.7. Funciones de pertenencia de la conducta goal-seeking.	44
3.8. Trayectoria del robot móvil para un ángulo de partida de -40° (la parte trasera corresponde a la marca azul).	45

3.9. Variables lingüísticas para la diferencia en la orientación. 46

3.10.Pruebas del controlador, la cruz azul marca la parte trasera del robot. . 47

3.11.Distribución de los cuadrantes en la cuadrícula de histogramas. 48

3.12.Funciones de pertenencia de las variables de entrada. 49

3.13.Prueba de la conducta obstacle-avoidance. 50

3.14.Funciones de pertenencia de la conductas wall following. 51

3.15.Prueba de la conducta wall-following derecha. 52

3.16.Prueba de la conducta wall-following izquierda. 52

3.17.Funciones de pertenencia de la red neuro difusa de fusión de conductas. 53

3.18.Un caso particular en el movimiento del robot. 54

3.19.Entorno de prueba 1, punto de partida 1. 55

3.20.Entorno de prueba 1, punto de partida 2. 56

3.21.Recorrido en entorno de prueba 2. 56

3.22.Histograma de cuadrículas obtenido tras el ensayo. 57

3.23.Entorno de prueba 3. 57

3.24.Entorno de prueba 4. 58

3.25.Entorno de prueba 5. 58

3.26.Entorno de prueba 6. 59

3.27.Interface de usuario en GUIDE de MATLAB. 59

3.28.Prueba 1 con edición de obstáculos. 60

3.29.Prueba 2 con edición de obstáculos 60

3.30.Eschema del funcionamiento de Player/Stage. 61

3.31.Posición inicial del robot móvil. 64

3.32.Captura 2 del movimiento del robot en su trayectoria hacia la meta. . . 64

3.33.Captura 3 del movimiento del robot en su trayectoria hacia la meta. . . 65

3.34.Llegada a la meta del robot. 65

3.35.Recorrido usando CRBC. 66

3.36.Angulo de timón y factor de velocidad generados usando CRBC. 67

3.37.Tiempo de cómputo por iteración usando CRBC. 67

3.38.Recorrido usando BBFH. 68

3.39.Angulo de timón y factor de velocidad generados cuando se usa BBFH. 68

3.40.Tiempo de respuesta usando BBFH. 69

3.41. Recorrido de CRBC cuando los sonares presentan ruido.	70
3.42. Recorrido de BBFH cuando el recorrido tiene espacios estrechos. . . .	70
3.43. Trayectorias de seguimiento. a) usando VFF[17] sin filtro, b) usando BBFH.	71
3.44. Recorrido cuando se usa VFH.	71
3.45. Angulo de timón y factor de velocidad generados cuando se usa VFH .	72
3.46. Tiempo de cómputo por iteración cuando se usa VFH.	72
3.47. Trayectorias para VFH con dos tamaños de cuadrícula. A. Usando un tamaño de cuadrícula de 25x25; B. Usando un tamaño de cuadrícula de 33x33.	73
3.48. Trayectorias para BBFH con dos tamaños de cuadrícula. A. Usando un tamaño de cuadrícula de 25x25; B. Usando un tamaño de cuadrícula de 33x33.	73
3.49. Trayectorias usando VFH(azul) y BBFH(rojo). A. Cuando el punto de llegada está en la zona media, B. Cuando el punto de llegada esta en una esquina.	74
3.50. Potencias consumidas durante el recorrido con ambos algoritmos. . . .	74
4.1. Representación del acelerómetro.	77
4.2. Ensayo del acelerómetro en reposo.	78
4.3. Esquema de coordenadas en un giroscopio.	79
4.4. Medidas del giroscopio con el equipo en reposo	79
4.5. Integración de las medidas en giroscopio.	80
4.6. Ensayo en reposo del magnetómetro.	81
4.7. Ensayo del magnetómetro con varias rotaciones.	82
4.8. Medidas de la aceleración lineal.	83
4.9. Integración de las medidas en la aceleración.	84
4.10. Esquema de un filtro de Kalman discreto[4]	86
4.11. Esquema del algoritmo EKF[18].	87
4.12. Punto de Acceso TP-Link TL-WR841HP.	89
4.13. Comunicación entre Android y Arduino en funcionamiento.	90
4.14. Sistema de coordenadas NED[19]	90
4.15. Funciones de pertenencia de la red neuronal para el ajuste de covarianza.	93
4.16. Estimación de los ángulos.	93

4.17. Estimación de los ángulos usando filtro de Kalman y filtros complementarios.

94

4.18. Comparación entre las estimaciones.

94

4.19. Recorrido usado para la estimación

97

4.20. Datos en la coordenada X.

97

4.21. Datos en la coordenada y.

98

4.22. Prueba de EKF con ruido gaussiano en la orientación.

98

4.23. Estimación en la coordenada X.

99

4.24. Estimación en la coordenada Y.

99

4.25. Estimación en coordenada X manteniendo fija la posición.

100

4.26. Estimación en coordenada Y manteniendo fija la posición.

100

4.27. Estimación de la trayectoria con condiciones iniciales alejadas y el robot móvil en movimiento.

101

4.28. Estimación de las coordenadas X e Y con condiciones iniciales alejadas y el robot móvil en movimiento.

102

4.29. Estimación de la posición con condiciones iniciales alejadas y el robot móvil estático.

102

4.30. Estimación de las coordenadas X e Y con condiciones iniciales alejadas y el robot móvil estático.

103

4.31. Estimación de la trayectoria con estimación previa y con el robot móvil en movimiento.

103

4.32. Estimación de las coordenadas X e Y con estimación previa y con el robot móvil en movimiento.

104

5.1. Conexiones del robot.

107

5.2. Switch inalámbrico Trendnet para la interconexión del robot con una laptop personal.

108

5.3. Acceso desde Laptop al sistema operativo del computador a bordo. . .

108

5.4. Interconexión del robot con el sistema de localización.

109

5.5. Ciclo de procesamiento en el robot Pioneer P3-AT usando la librería Aria.

109

5.6. Recorrido 1.

112

5.7. Recorrido 2.

112

5.8. Recorrido 3

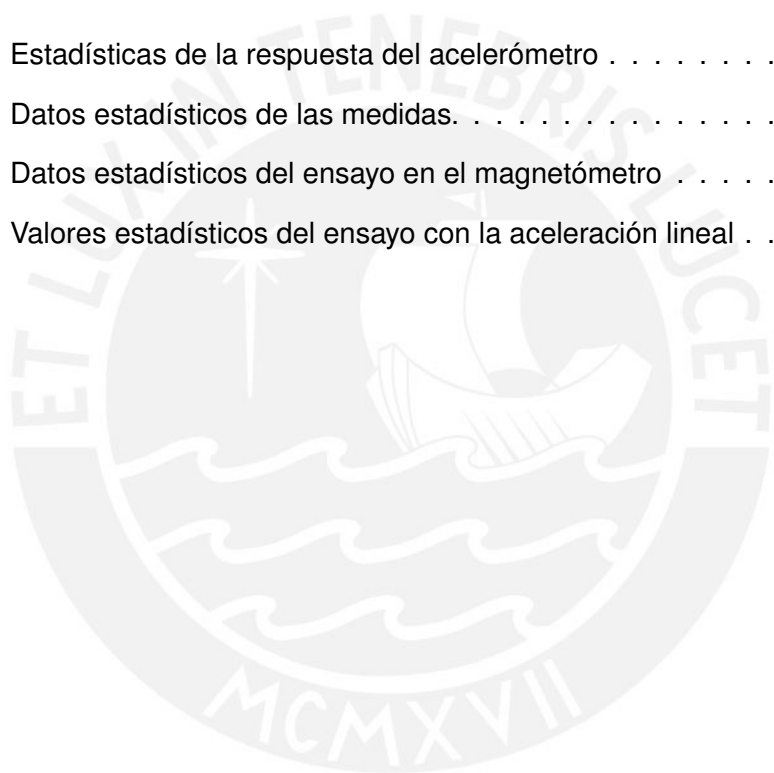
113

5.9. Recorrido 4.

113

Índice de tablas

1.1. Estructuras comunes en robots móviles con ruedas. [1]	7
3.1. Base de reglas para el arbitro de fusión de conductas.	54
4.1. Estadísticas de la respuesta del acelerómetro	78
4.2. Datos estadísticos de las medidas.	80
4.3. Datos estadísticos del ensayo en el magnetómetro	81
4.4. Valores estadísticos del ensayo con la aceleración lineal	83



Introducción

En la actualidad existe cada vez más una notoria influencia de los robots autómatas en casi todas las áreas de la tecnología y la industria. Esta presencia aunada a las grandes ventajas que reporta el trabajo realizado por dichas máquinas, extendiendo los límites del hombre de forma segura, precisa y confiable, hacen de su desarrollo y mejora continua, una materia primordial en el devenir de la tecnologías modernas además de un gran aliciente para la aparición de muchas otras.

El control de la trayectoria seguida por un robot móvil autómata plantea en general un gran reto en los métodos de medición, ya que en la mayoría de casos se requieren mediciones exactas y adecuadamente veloces, debido a las grandes necesidades de precisión en muchos de los sistemas modernos. En este sentido, se requiere no solo de mediciones buenas, sino que además estos sistemas deben ser robustos, confiables y no deben incurrir en altos costos de implementación; por otro lado el sistema de control debe ser capaz de brindar autonomía al robot para poder manejar trayectorias complejas y evadir obstáculos en el camino.

El diseño de algoritmos de control para los robots móviles requiere pues de una medición adecuada a fin de poder alcanzar las especificaciones de control. Para ello y con el fin de emplear eficientemente todos los elementos del sensado en un equipo es necesario emplear técnicas de fusión de sensores de tal manera que se pueda disponer de datos con menor incertidumbre, mayor robustez, supresión del ruido adecuada y cobertura en el tiempo y espacio requeridos.

En el caso de la navegación de robots móviles tipo carro las principales variables a estimar son la posición, dirección y velocidad. Para ello se dispone de una amplia gama de sensores entre los cuales destacan por su bajo costo y gran popularidad los encoders, los cuales sin embargo, se caracterizan por la inclusión de ruido y difícil calibración, para la localización, por otro lado, se siguen diversos esquemas entre los cuales se puede mencionar el uso de radio frecuencia (RF) y los sistemas de posicionamiento global (GPS); los sistemas de radio frecuencia tienen la ventaja de ser menos costosos, sin embargo también presentan problemas de ruido e interferencia.

Por otro lado, para el sistema de navegación en trayectorias complejas se debe observar que el sistema resultante es no-lineal y no-holonómico; la suma de estos dos factores añadido al carácter autónomo que se le desea brindar al robot móvil y las posibles implicaciones del ruido y perturbaciones hace de este un sistema altamente

complejo. Para el diseño del sistema de control se necesita pues del empleo de sistemas con capacidad inteligente tales como las redes neuro difusas, las cuales vienen siendo ampliamente usadas en la actualidad para sistemas inteligentes por su gran desempeño y versatilidad.

La presente tesis tiene como objetivo el desarrollar un sistema de navegación apropiado para robots móviles, el cual pueda ser aplicado usando algoritmos inteligentes como son las redes neuro difusas y utilice para su localización sensores de bajo costo presentes en la mayoría de equipos robóticos actuales.

El documento esta organizado como sigue: El Capítulo 1 presenta en detalle el estado de la investigación actual en los diferentes temas que implica la navegación y la localización en robots móviles, además de realizar una breve introducción a la estructuras de los robots móviles tipo carro. El Capítulo 2 presenta en detalle el análisis cinemático de los robots empleados, es decir el análisis del movimiento sin la consideración dinámica del movimiento, para ambos tipos de robots móviles usados.

El Capítulo 3 presenta el diseño y las simulaciones del controlador neuro difuso para la navegación, además se presentan comparativas con otros tipos de controladores típicos usados en este tipo de navegación y se presentan algunos de los beneficios que este tipo de control puede proporcionar. El Capítulo 4 detalla los sensores y el sistema de fusión de los mismos para la estimación de la posición y orientación. Y por último, en el Capítulo 5 se detallan los aspectos principales para la implementación del sistema de navegación en una plataforma real como es el robot Pioneer P3-AT.

Capítulo 1

Estado del Arte de los Sistemas de Navegación de Robots Móviles Tipo Carro

1.1. Introducción

Pueden encontrarse muchas definiciones del término robot en la bibliografía relativamente antigua, una de las más completas, dada en el Standard Europeo EN775/1992, lo define como sigue: “Un robot manipulador industrial es una máquina manipuladora y multi-propósito controlada automáticamente y reprogramable, con diferentes grados de libertad, el cual puede estar fijado en un lugar o puede ser móvil para el uso en aplicaciones de automatización industrial”.

El desarrollo de robots móviles, sin embargo, es un campo relativamente joven que data apenas de fines de la década de 1970. Con anterioridad el desarrollo se había centrado casi exclusivamente en robots fijos, principalmente de brazos robóticos y manipuladores, los cuales son aún ampliamente usados en la industria representando una gran cuota comercial. Sin embargo, los robots móviles presentan una clara ventaja con respecto a estos, dado que su capacidad de desplazarse libremente flexibiliza su funcionamiento y distribuye sus funciones.

Los robots móviles nacen así principalmente con el fin de posibilitar el trabajo en lugares con ambientes no estructurados y condiciones cambiantes, lugares de difícil acceso para el hombre o donde la suciedad, las sustancias presentes o las condiciones climáticas hacen imposible su presencia[1]. Una muestra de ello es la inmensa cantidad de robots exploradores que se han desarrollado, algunos ejemplos son los robots Dante (1993) y Dante II (1994) mostrados en la Figura 1.1, el robot Sojourner (1997) de la misión Path Finder, que permitió el registro de la superficie marciana (Figura 1.2), o los robots acuáticos Seabotix (Figura 1.3) usados para el registro de superficie marina.



Figura 1.1 – Robot Dante II.

Fuente: http://www.frc.ri.cmu.edu/robots/robs/photos/1994_Dantell.jpg

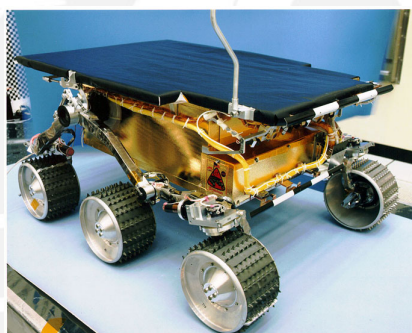


Figura 1.2 – Robot teleoperado Sojourner.

Fuente: http://mars.jpl.nasa.gov/MPF/roverctrlnav/images/fur_corner.jpg

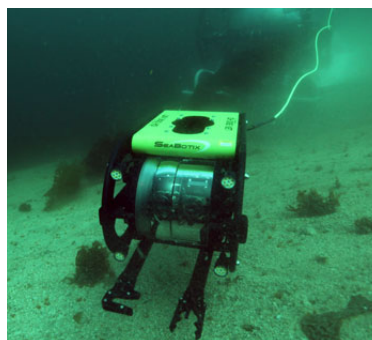


Figura 1.3 – Robot subacuático SeaBotix.

Fuente: <http://www.seabotix.com/>

El desarrollo de los robots móviles se ha extendido además con usos en la industria, donde se han realizado esfuerzos tempranos en el desarrollo de sistemas para el apagado de incendios, el traslado de materiales, la limpieza de edificios, la patrulla y cuidado de áreas de almacenaje y la inspección de maquinarias (figura 1.4) por nombrar algunas.

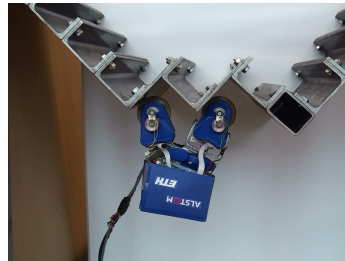


Figura 1.4 – Robot MagneBike.

Fuente: <http://www.asl.ethz.ch/research/asl/alstom>

En la actualidad los sistemas teleoperados gozan de una gran popularidad, sin embargo, la gran dificultad a nivel bajo que representa la locomoción del robot hace necesario que el robot trabaje con cierta autonomía. Es por ello que muchas de las aplicaciones modernas son principalmente manejadas con directivas externas, dejando al robot los complicados cálculos para su desplazamiento.

La autonomía de los robots móviles representa uno de los grandes retos en el desarrollo presente y futuro de la tecnología. El diseño de sistemas de navegación autónoma requiere de una cantidad amplia de conocimientos en diferentes áreas de la investigación que van desde la mecánica y la cinemática, los algoritmos computacionales y la teoría probabilística. Los retos que plantea la navegación dada las condiciones imprevisibles que pueden presentarse en la trayectoria y las limitaciones en el sensado es otro de los puntos en los que aún se trabaja hoy en día.

El interés especial que se ha tenido hacia los robots ha generado una diversidad (figura 1.5) en el desarrollo de los mismos, distinguiéndose estos por el medio por el cual se desplazan como: terrestres, aéreos y acuáticos[20]. No obstante se puede hacer una distinción más profunda en el caso de los robots terrestres según el medio de locomoción usado, distinguiéndose la locomoción con ruedas y la locomoción con piernas[1, 2].

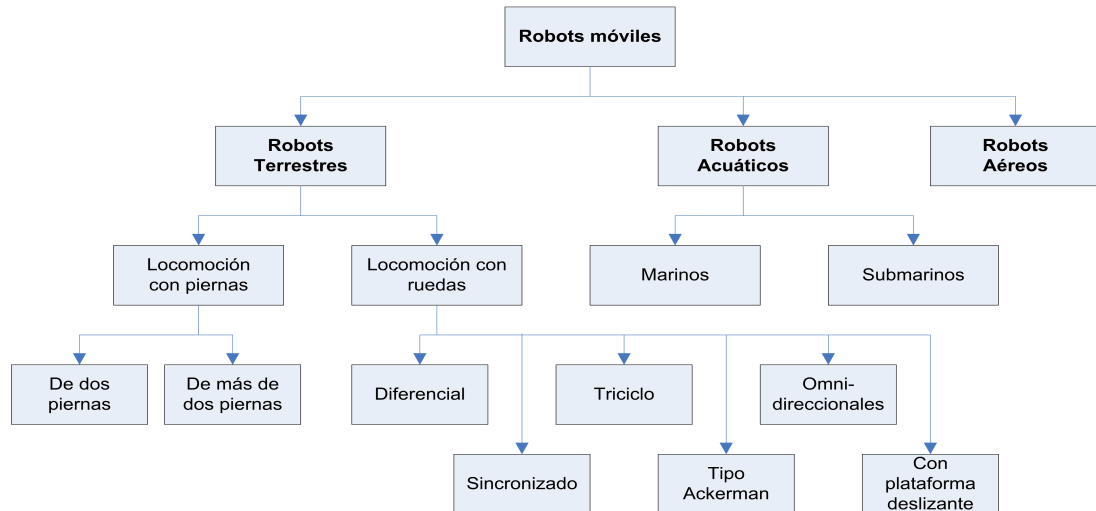



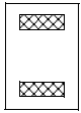
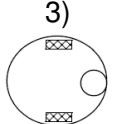
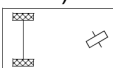
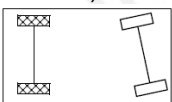
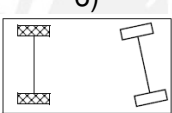


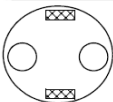
Figura 1.5 – Robots terrestres, aéreos y acuáticos.

El siguiente trabajo se enfoca en sistemas de locomoción con ruedas, para el cual se han planteado diversas estructuras a lo largo de los años, algunas de las cuales pueden verse en [2, 1, 21], la estructura define, entre otras cosas, el propósito y función del vehículo implementado. En la siguiente tesis se trabaja con una estructura de robot móvil tipo carro o de Ackerman, la cual se caracteriza por tener 4 ruedas, tracción en uno o ambos ejes y movimiento libre en el eje sin tracción.

1.2. Descripción de los Robots Móviles Tipo Carro

Dada su gran difusión, existe en la actualidad un gran número de estructuras para los robots móviles con locomoción por ruedas, algunas de las estructuras más populares se muestran en la tabla 1.1.

Tabla 1.1 – Estructuras comunes en robots móviles con ruedas. [1]

# ruedas	Estructura	Características	Ejemplos
2	1) 	Dirección delantera y tracción posterior.	Bicicleta
	2) 	Dos ruedas diferenciables	Cye robot
3	3) 	Dos ruedas centrales diferenciables y una tercera con punto de contacto.	smartRobot
	4) 	Dos ruedas traseras de tracción y una delantera de dirección.	Piaggio robots
	5) 	Dos ruedas traseras libres y una delantera de dirección y tracción	Neptune robot
4	6) 	Dos ruedas traseras de tracción y dos delanteras de dirección	Carros de conducción trasera
	7) 	Dos ruedas traseras de dirección y dos delanteras de tracción	Carros de conducción delantera
	8) 	Cuatro ruedas omnidireccionales	Carnegie Mellon Uranus
	9) 	Dos ruedas diferenciables y 2 ruedas con punto de contacto	Khepera

La estructura de los robots tipo carro, también conocida como de tipo Ackerman (Estructura 6), se basa en la locomoción de los clásicos vehículos de 4 ruedas hechos para la conducción humana, y es por ello la forma más extendida en la actualidad. Su estructura basa su tracción en las ruedas traseras, y la dirección en las ruedas delanteras.

Las características de los robots móviles con ruedas se pueden especificar de acuerdo a tres conceptos generales, los cuales son: La estabilidad, la maniobrabilidad y la controlabilidad[1]. Y en base a estos tres conceptos se puede comparar en cierto grado las diferentes estructuras existentes.

La estabilidad se refiere a la capacidad del vehículo de mantenerse en una posición de equilibrio, y puede ser de dos tipos: estabilidad estática y estabilidad dinámica, siendo la última la más difícil de conseguir y posible solo para vehículos con 3 ruedas o más. La maniobrabilidad por otra parte se refiere a la capacidad del vehículo de moverse en una dirección arbitraria dada cierta orientación inicial del mismo con respecto al eje, en ese sentido la mayor maniobrabilidad es obtenida con los robots omnidireccionales. Por último la controlabilidad se refiere al grado de facilidad con que se puede manipular el robot móvil, y presenta en general una correlación contraria con respecto a la maniobrabilidad, por ejemplo en el caso de los robots omnidireccionales, la configuración de la estructura en si misma hace más complejo su algoritmo[1].

La estructura de los robots móviles tipo carro posee una baja maniobrabilidad debido a que sus ruedas presentan en todo momento un mismo centro de rotación instantáneo (ICR), no obstante tienen una estabilidad media, destacando su estabilidad en giros a velocidad moderada. Mientras que por otro lado, la simpleza en su lógica de manejo hacen que su controlabilidad sea alta.

Su relativa simetría permite además su reducción de su modelo cinemático para efectos del cálculo, de manera que es usual en la mayoría de trabajos científicos trabajar con el modelo bicicleta para su modelado (Estructura 1). Se muestra en la figura 1.6a el robot tipo carro, se tiene en este caso un sólo centro de rotación instantánea (ICR), su reducción en el modelo bicicleta se muestra en la figura 1.6b con una rueda de tracción (w_1) y una rueda de dirección (w_2).

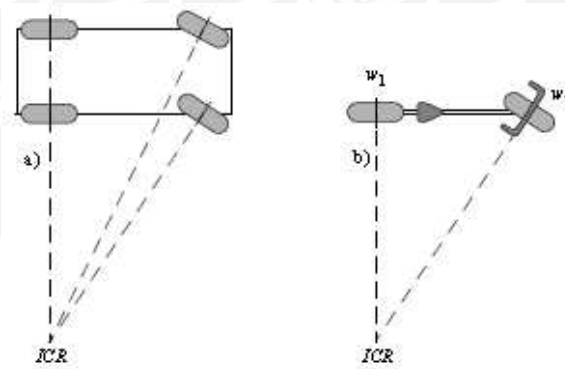


Figura 1.6 – Estructura de los robots móviles tipo carro. [1]

1.3. Estado del Arte de los Sistemas de Navegación Autónoma para Robots

La navegación autónoma puede definirse en general como la habilidad de un móvil de determinar su posición, describir un recorrido deseado evitando obstáculos que puedan interferir con el mismo y alcanzar tras ello una posición final o meta[22, 23, 24, 25].

En general el proceso de la navegación puede dividirse en tres principales problemas los cuales son la localización, el planeamiento del recorrido y la construcción de mapas[26, 27, 2, 23].

La localización implica la determinación de la posición y orientación del robot con respecto a su alrededor, implica además el reconocimiento de los objetos y su distinción en obstáculos o metas[22, 1, 2]. En la figura 1.7 se muestra un esquema del proceso de localización, en general se tiene primero una comparación entre las señales obtenidas por odometría y un modelo interno, con el objetivo de reducir la incertidumbre.

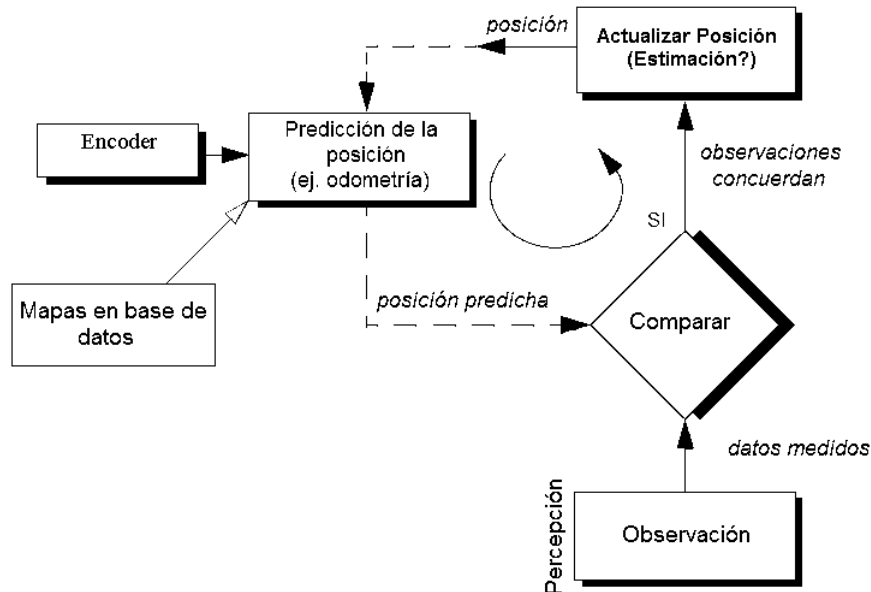


Figura 1.7 – Esquema general de localización [1]

Mientras que el planeamiento de recorrido y la construcción de mapas implican la capacidad del robot de encontrar una ruta libre de colisiones que lleve al robot a alcanzar su objetivo[28, 23, 2], se puede además distinguir al planeamiento como una extensión de la localización ya que depende de que esta última determine la posición actual y la posición de la meta correctamente; mientras que la construcción de mapas, particularmente en sistemas de navegación basados en mapas, es en esencia una notación para describir la posición del robot dentro de un marco de referencia[29].

1.3.1. Estado de Arte de la Localización

Cabe mencionar que no obstante la localización es uno de los puntos más importantes en la navegación, existen sistemas de navegación que no tienen en cuenta este proceso, tales como algunos sistemas de navegación por conducta (behavior-based), cuya estructura se aprecia en la figura 1.8. Este método se basa en la generación de soluciones particulares, cuando estas existen; y tienen como principal ventaja que pueden ser generadas rápidamente para un ambiente específico.

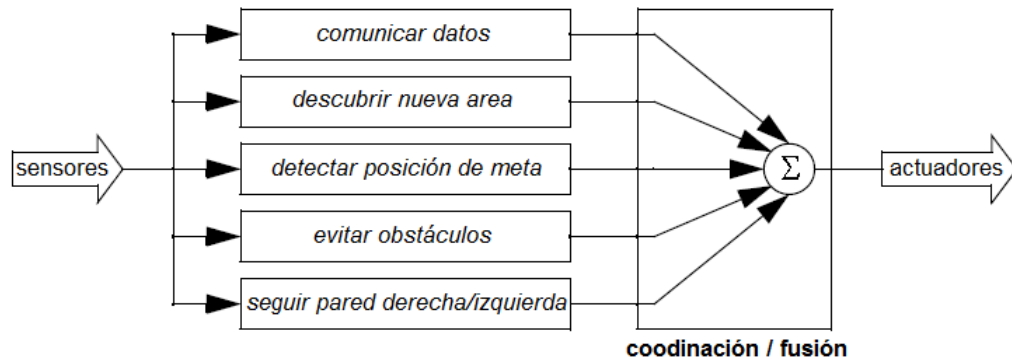


Figura 1.8 – Arquitectura de la navegación basada en conducta [1].

Otro método de navegación que puede evitar la localización es la navegación biomimética, y que no obstante, constituye un tipo de navegación basada en conductas, presenta la particularidad de basar su funcionamiento en conductas de entes biológicos[24]. Entre las desventajas de no considerar la localización, se puede mencionar el hecho de su no escalabilidad, además tienen una dependencia del ambiente en que se implementan y en las características del hardware del sistema, por otra parte el agregar mas conductas en el robot requerirá de una reconfiguración fina a fin de que estas conductas no entren en conflicto[1].

La figura 1.9 muestra, en cambio, el esquema de la navegación basada en mapas, se puede notar que en este caso la localización es uno de los pasos intermedios por el cual se decide la siguiente acción de control. La aproximación basada en mapas implica además una mayor claridad para el usuario ya que los mapas en si representan un medio de comunicación natural entre hombre y robot.

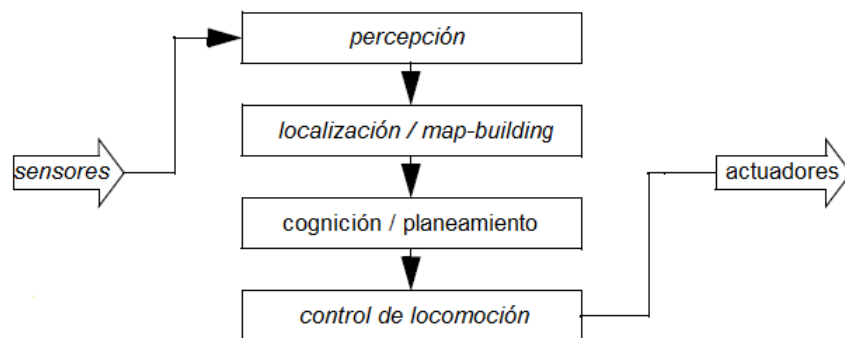


Figura 1.9 – Arquitectura de la navegación basada en mapas.

La manera en que se representa la información del entorno en los métodos de navegación basada en mapas es de suma importancia en su mecanismo. A continuación se detallan algunos de los métodos más comunes para la representación dentro de la navegación basada en mapas.

El método de descomposición por células exactas introducido por Latombe[30] realiza

la descomposición del mapa mediante límites de figuras geométricas simples (figura 1.10), sin embargo esta aproximación no se puede aplicar en todos los casos ya que requiere de información no siempre disponible.

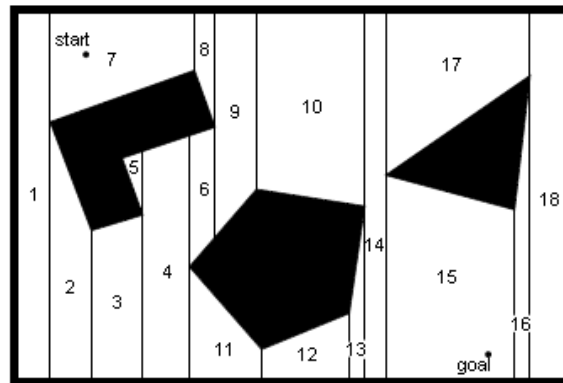


Figura 1.10 – Descomposición celular exacta[1].

La descomposición fija, por otra parte, descompone el mapa en mosaicos pequeños, uno de las versiones más populares de este es el método de cuadrícula de ocupancia[31, 32, 33] (figura 1.11), no obstante estos métodos suponen una geometría del entorno que en muchos casos puede no ser lo más realista. La descomposición topológica por otra parte evita el tratado de la geometría y se centra más en las conexiones entre regiones, el mapa topológico contiene así dos elementos primordiales, los nodos y la conexión entre ellos. Pueden verse ejemplos en [34, 35], sin embargo la descomposición topológica no es clara para el usuario y puede hacer uso intensivo de memoria.

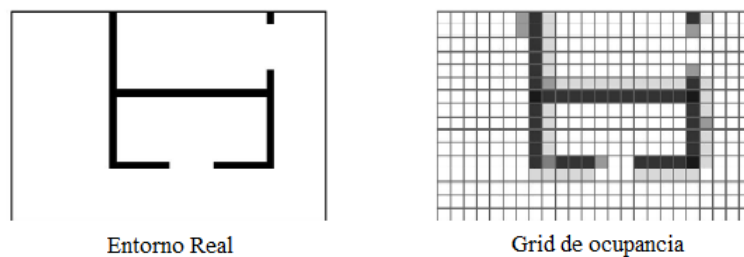


Figura 1.11 – Ejemplo de cuadrículas de ocupancia.

A estos métodos hay que agregar dos enfoques basados en la inclusión del análisis estocástico para sistemas de navegación basado en mapas, el método de localización de Markov[36, 37] y el método de localización por filtros de Kalman[38, 39] (figura 1.12) incluyen la probabilidad para las posiciones y reducen considerablemente la incertidumbre por medio de la regla de Bayes. Además existen también sistemas híbridos que usan ambos enfoques[40].

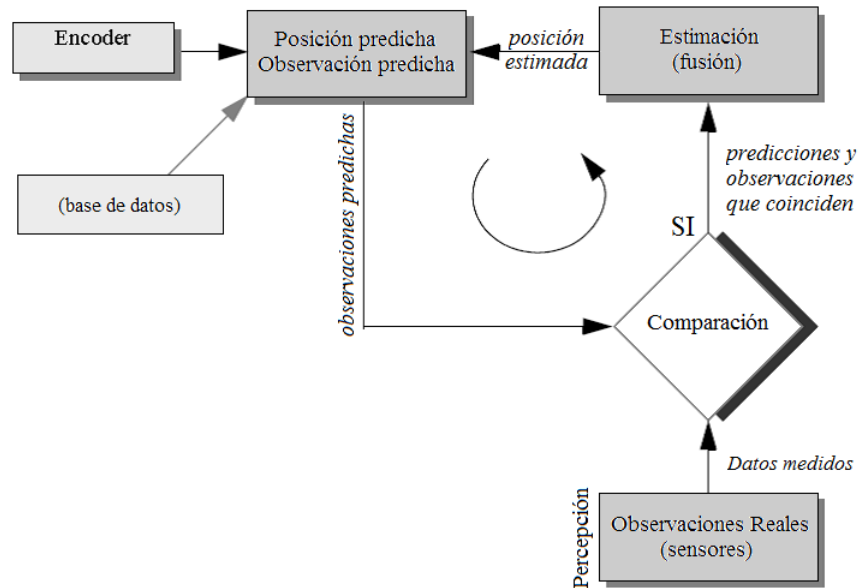


Figura 1.12 – Diagrama de un sistema de localización con filtros de Kalman para robots móviles[1].

Por último están los métodos de construcción automática de mapas (figura 1.13), también conocidas como de localización y mapeo simultáneo (SLAM). En este caso el robot debe ser capaz de reconstruir el mapa del entorno y ubicarse en el a partir de la exploración del mismo[1, 41]. El esquema tradicional está basado en filtros de Kalman[42, 43, 44, 45]. Otro enfoque es el de filtro por partículas SLAM, el cual está basado en el problema de Monte Carlo[46] y en los trabajos de Rao y Blackwell[47, 48], los cuales a su vez se basan en la distribución de la probabilidad en partículas y el uso de funciones no convencionales; además del manejo de filtros de Kalman separados. Este método fue introducido por primera vez en [49], y popularizada por Montemerlo como FastSLAM en[39], el uso de cámaras es frecuente en estos métodos y pueden añadir demasiada complejidad al algoritmo.

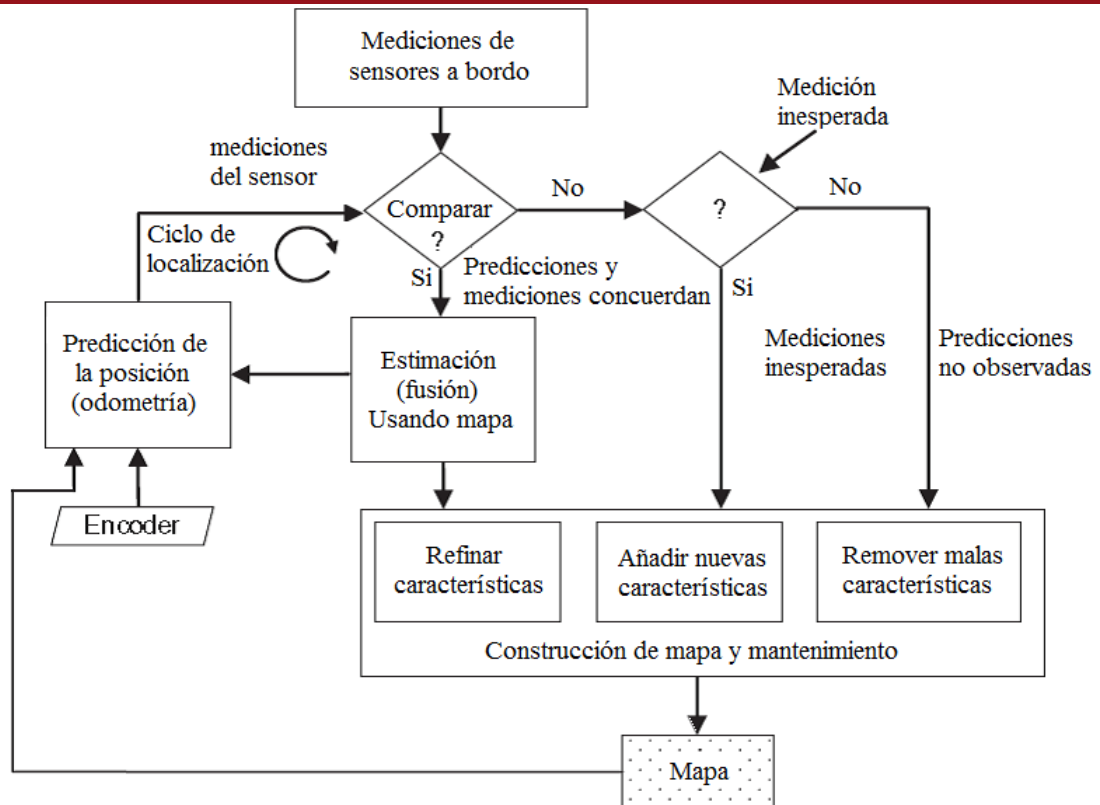


Figura 1.13 – Diagrama de la localización por el método SLAM[2].

Los siguientes son algunos de los métodos para la localización que implican el uso de sensores externos, la localización usando fusión de sensores se analiza y describe en detalle en el punto 1.4.

La navegación basada en puntos de referencia presentada en[3], utiliza elementos externos como marcas para la localización del robot, en la figura 1.14 puede apreciarse un ejemplo de este tipo de navegación, la forma de la marca usada permite una localización precisa, no obstante el recorrido entre marcas debe realizarse mediante estimaciones. La principal desventaja de este método es que requiere de muchas modificaciones en el entorno, las cuales pueden no ser siempre permitidas.

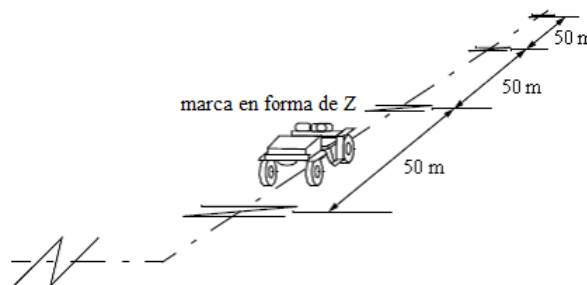


Figura 1.14 – Navegación por marcas en forma de Z [3].

La localización global única se basa en el reconocimiento de cada lugar de forma única, la localización en este caso requiere de un alto grado de información del lugar, lo

cual se logra en general solo mediante visión, pueden verse ejemplos en [50, 51]. Sin embargo, el sistema requiere aún de algunos avances adicionales en la tecnología, además de afrontar un gran reto en la distinción de ambientes ambiguos en los que la distinción entre estos puede no ser clara.

El posicionamiento mediante balizas activas (figura 1.15), es uno de los métodos de localización mas confiables en la actualidad, además es quizás el método más usado en la industria y aplicaciones militares debido a su alta precisión en la localización. Las balizas activas permiten al robot ubicarse en todo momento ya que brindan una referencia fija al mismo, además puede considerarse el mismo esquema para un número mayor de robots sin la modificación de la configuración. No obstante su implementación puede resultar muy costosa y el movimiento del robot a otro entorno requiere una reprogramación y trabajo adicional que puede significar un gasto de tiempo y dinero mayores[1].

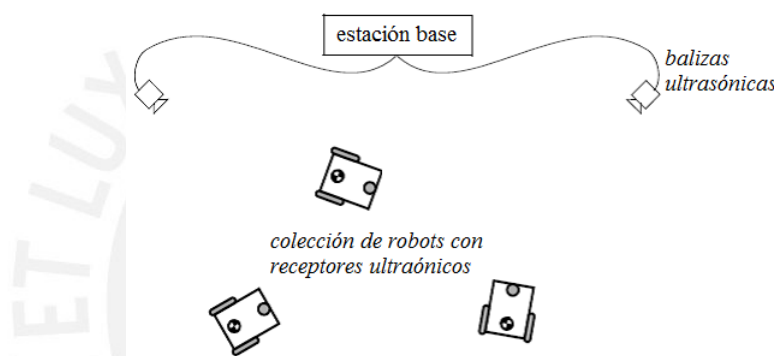


Figura 1.15 – Navegación con balizas de referencia [1].

1.3.2. Estado de Arte del Planeamiento de Rutas y Construcción de Mapas

Como se describió antes, el planeamiento de rutas implica la búsqueda de un camino óptimo y libre de colisiones, que permita al robot moverse desde el inicio hasta la meta. Para alcanzar este objetivo se pueden considerar dos enfoques del problema, el planeamiento considerando obstáculos estáticos y el planeamiento considerando obstáculos dinámicos; no obstante es usual combinar métodos en los sistemas de navegación.

La construcción de gráficos es uno de los primeros retos cuando se enfrenta el problema del planeamiento, para ello se han desarrollado una gran cantidad de enfoques. Las gráficas de visibilidad permiten trazar rutas en base a las formas de los obstáculos[30](figura1.16), no obstante las rutas generadas son en ocasiones peligrosamente cercanas a los vértices.

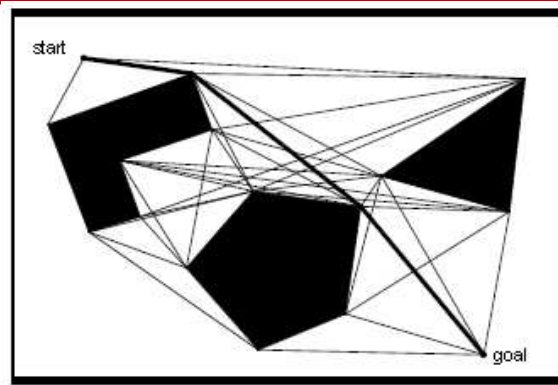


Figura 1.16 – Gráficos de visibilidad[1].

Los diagramas de Voronoi por otra parte generan posibles rutas maximizando la distancia con los obstáculos[30](figura 1.17), este método, sin embargo, puede ser demasiado complejo cuando las estructuras son imperfectas. Un tercer método es la descomposición en células aproximadas, que como ya se mencionó en la sección 1.3.1 divide el mapa en pequeños mosaicos, por su parte la descomposición en células-variables aproximadas mejora un poco el algoritmo al incluir celdas de dimensión variable para encontrar rutas estrechas (figura 1.18). Otro método usado está basado en los gráficos Lattice[52], el cual permite la elección de la ruta en una forma más elaborada.

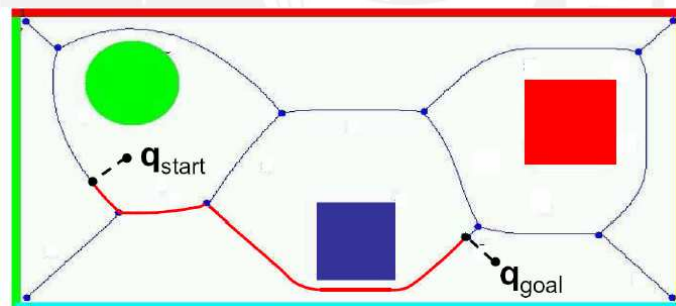


Figura 1.17 – Diagramas de Voronoi.

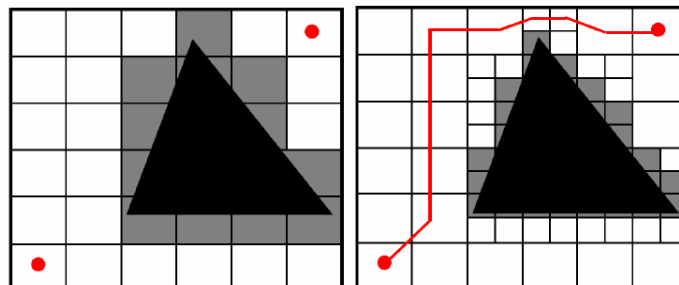


Figura 1.18 – Descomposición en células aproximadas, izquierda: Método de células aproximadas, derecha: Método de células-variables aproximadas.

Otros dos enfoques de gran uso son los métodos determinísticos tradicionales y los métodos probabilísticos modernos.

Los métodos determinísticos tienen como objetivo la obtención de la mejor ruta basados en los mapas construidos antes. Uno de ellos es el algoritmo breath-first search que calcula los saltos entre nodos necesarios para alcanzar la meta, el costo entre cada enlace debe ser el mismo. Por otra parte el algoritmo Dijkstra[53] permite asignar valores a los enlaces entre nodos y mejorar así la selección de la ruta, no obstante se necesita un mayor número de iteraciones. El algoritmo A*[54] que es similar al algoritmo Dijkstra incluye un factor heurístico adicional al costo y puede disminuir el número de recorridos entre nodos comparado al algoritmo Dijkstra.

Los métodos probabilísticos por otra parte excluyen las rutas prohibidas ya sea por posibles colisiones o por limitaciones en la cinemática del robot. El algoritmo probabilístico de hoja de ruta (PRM) genera los nodos aleatoriamente y elimina las rutas prohibidas, necesita no obstante datos pre-procesados. Otro método es el algoritmo de exploración rápida en arboles aleatorios (RRTs), que como su nombre indica construye un árbol de rutas posibles a partir de añadir sucesivamente nodos aleatorios adicionales y basado en las restricciones del robot móvil, además estos nodos deben verificarse libres de colisión; el algoritmo garantiza encontrar una ruta más no que esta sea la más óptima; un ejemplo se puede apreciar en la figura 1.19, las iteraciones crecen rápidamente en cada proceso de propagación.



Figura 1.19 – Exploración rápida de arboles aleatorios[1].

El enfoque reactivo del planeamiento es sin duda uno de los mas estudiados y para el cual se han desarrollado una gran cantidad de métodos. En este caso la respuesta se basa en las mediciones de los sensores y por ende corresponde a un método más práctico, en general no depende de la disponibilidad de mapas; a continuación se muestran algunos de los más populares.

Los vehículos de Braitenberg[55] consideran la conexión directa entre los sensores y el motor, y constituyen la primera solución de tipo reactiva planteada. La interconexión directa permite desarrollar algunas conductas sencillas para la realización exitosa del planeamiento, no obstante las rutas generadas distan mucho de ser las más óptimas [10].

El método de planeamiento por campos potenciales[56, 57, 58] es uno de los más populares. Su funcionamiento, tal como el nombre lo indica, se basa en el concepto

de los campos eléctricos, en este caso un obstáculo se traduce como una fuente de repulsión y la meta como una fuente de atracción (figura 1.20) este enfoque puede fallar no obstante en sendas estrechas y otros casos particulares como los mínimos locales. El método extendido del mismo, desarrollado en[59] permite trayectorias mas suaves, mientras que en [60] Feder desarrolla una modificación para evitar los mínimos locales.

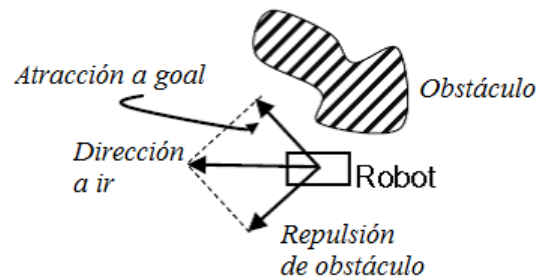


Figura 1.20 – Esquema de un navegación por campos potenciales[2].

En[61] se presenta el algoritmo Bug (figura 1.21) y como se observa su funcionamiento se basa en rodear obstáculos para la evasión, las rutas generadas no son óptimas. Por otro lado, el método de histograma de campo vectorial (VFH) desarrollado por Borenstein y Koren[16] permite realizar una actualización de los mapas mediante la construcción intermedia de histogramas (figura 1.22), el costo en este caso se calcula a partir de 3 parámetros, sin embargo el cálculo necesario para la obtención de las medidas es solo aceptable para algoritmos de control sencillos.

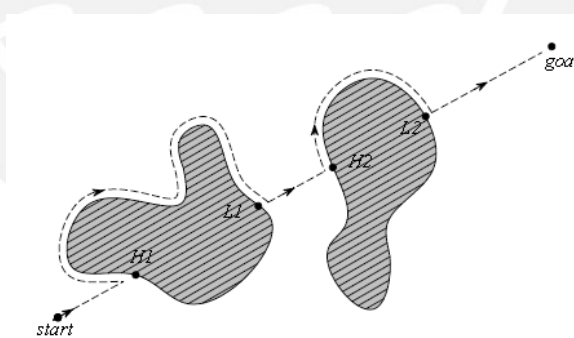


Figura 1.21 – Conducta de para el seguimiento de paredes (wall-following)[1]

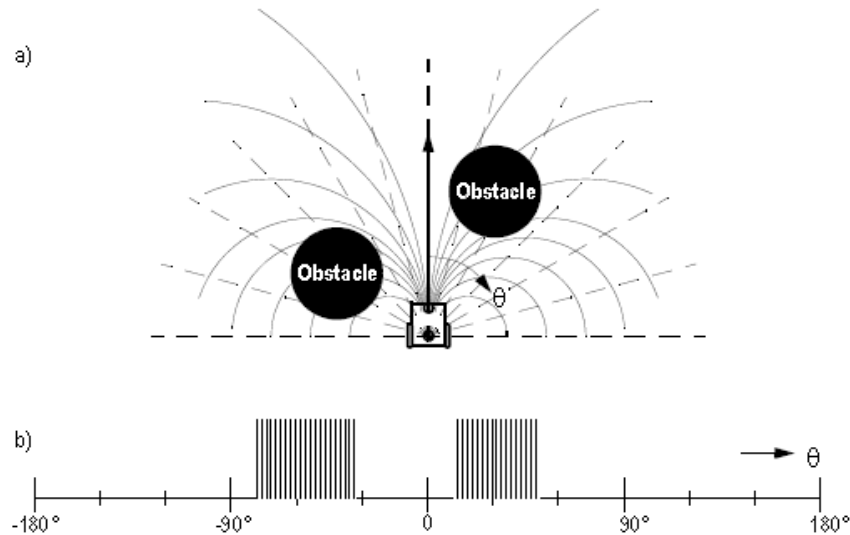


Figura 1.22 – Navegación por histogramas de campos vectoriales[1].

Existen también algunas modificaciones de este método como VFH+[62] y VFH*[63] que toman también en cuenta limitaciones cinemáticas de los robots móviles. Por otra parte el método bubble band[64] desarrollado por Quinlan y Khatib usa burbujas para representar el espacio libre y permite tomar en cuenta las dimensiones del robot (figura 1.23), no obstante necesita de conocimientos previos del entorno.

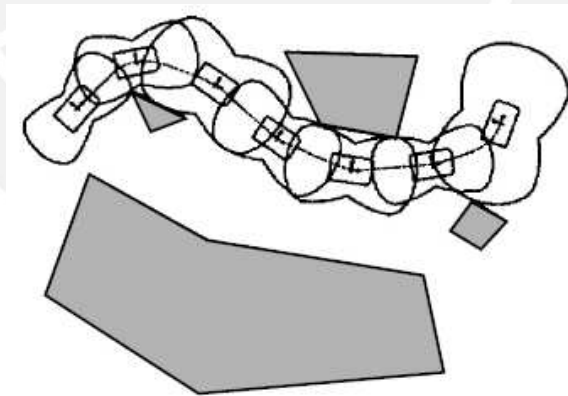


Figura 1.23 – Método de navegación bubble band[1].

El método de velocidad de curvatura (CVM) [65] presentado por Simmons genera rutas basado en las posibles trayectorias curvas del robot y puede en ese sentido incluir datos de la cinemática y algunas restricciones de los robots. Ko y Simmons presentan una mejora del mismo llamado método de curvatura de pista(LCM) en [66] donde proponen una mejora en la elección entre rutas alternativas a las curvas, no

obstante estos métodos necesitan una cuidadosa elección de sus parámetros para su correcto funcionamiento.

Otro enfoque es el de las ventanas dinámicas (1.24) presentado por Fox[67], el algoritmo crea primero una ventana con todos los sets de velocidades posibles y escoge la ruta en base a una función objetiva obtenida a partir de tres parámetros, como es de esperar permite incluir aspectos de la cinemática y restricciones del robot.

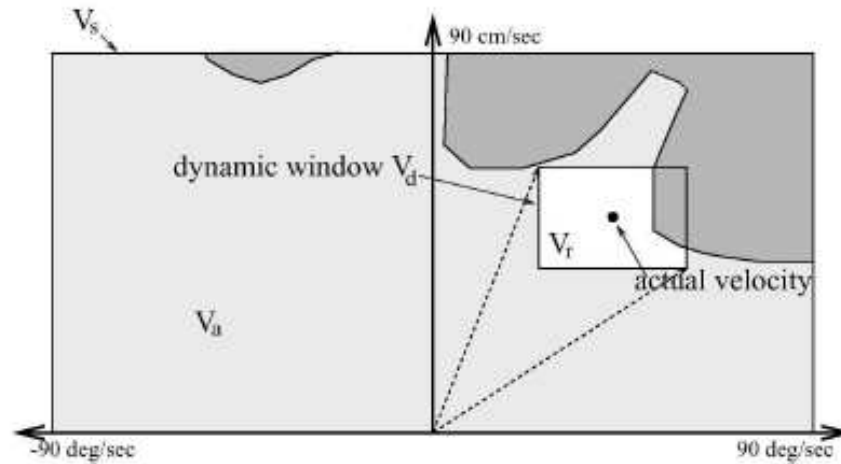


Figura 1.24 – Método de navegación de ventanas dinámicas[1].

El método de Schlegel[68] usa también las posibles trayectorias en arcos, este método está enfocado en el uso de láseres, una desventaja del método es que necesita tablas con cálculos hechos previamente. Otro método popular es el método del gradiente desarrollado por Konolige[69], en el que se realiza un re-cálculo rápido de la dirección del robot tomando en cuenta la cercanía de los obstáculos.

Los enfoques difuso y neuro difuso también corresponden a los métodos reactivos y presentan grandes ventajas por su relativa similitud al razonamiento humano, algunos de los trabajos en este campo se mostraron con más detalle en la sección 1.5, al igual que los enfoques basados en redes neuronales.

1.4. Estado del Arte de los Algoritmos de Fusión de Sensores

Antes de presentar las tendencias en los trabajos actuales de fusión de sensores, se hace necesario explicar algunas de las principales características de los sensores usados en robots móviles. Estos pueden distinguirse en dos grandes grupos, los sensores para la estimación de la posición relativa y los sensores para la estimación de la posición absoluta[21].

Entre las técnicas de sensado de la posición relativa puede mencionarse como principales la odometría y la navegación inercial. La odometría se basa principalmente

en el uso de encoders para medir la velocidad y orientación del robot, la desventaja principal en este caso es el crecimiento en el error si no se provee al sistema de una referencia adecuada y periódica. Por otro lado la navegación inercial trabaja principalmente con giroscopios y acelerómetros, las mediciones que se realizan en este caso son de la velocidad de rotación y la aceleración; y para la obtención del ángulo de giro y la posición se usa por ello hasta 2 integraciones. Sin embargo, este proceso puede generar un error considerable, lo que lo hace inviable en trayectos largos (figura 1.25).

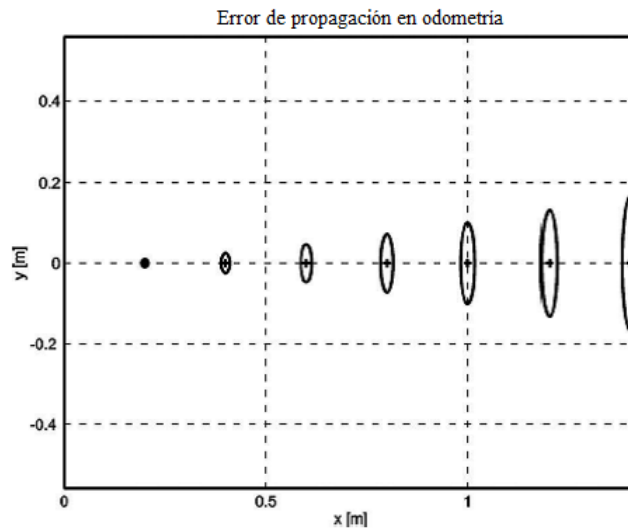


Figura 1.25 – Crecimiento de la incertidumbre en trayectoria recta[1].

Para los sensores de posición absoluta por otra parte se deben usar elementos de referencia externa; los principales métodos en este caso son, las balizas activas, el reconocimiento de puntos de referencia artificial, el reconocimiento de puntos de referencia natural, y el basado en modelos de concordancia.

En el caso del método de balizas activas se pueden usar los láseres y los dispositivos de radio frecuencia. Este método presenta en general un costo entre alto y moderado, y permiten estimar la posición absoluta, para ello la posición de los transmisores debe ser fija. En el reconocimiento de puntos de referencia artificial, en cambio se usan puntos de referencia colocados en distintas ubicaciones, el sistema se puede diseñar de tal manera que se mejora la robustez, sin embargo, requiere de un tiempo de cómputo mayor y de un gran número de referencias.

En el método por reconocimiento de puntos de referencia natural, se hace uso de puntos de referencia externa, los cuales son en general, formas distinguibles en el entorno. Este método sin embargo no logra aún alcanzar una precisión óptima. Por último los modelos de concordancia, se basan en la comparación de las mediciones tomadas del entorno con los mapas o modelos internos que el robot tiene del mismo.

Sumarizando se puede apreciar que no existe una solución única en el caso del sentido, y en general se usa una combinación de técnicas para obtener medidas que se complementen y refuerzen (redundancia). En este contexto la fusión de sensores

es el proceso de integrar los datos obtenidos por diferentes sensores para detectar objetos y para la estimación de parámetros y estados necesarios en la localización, construcción de mapas, planeamiento, cálculo de la trayectoria, planeamiento de la locomoción, y ejecución del movimiento[70] y puede mejorar el desempeño de un sistema en las siguientes maneras: Representación, Certeza, Precisión e Integridad[4].

Existe una gran cantidad de trabajos en fusión de sensores, una clasificación de estos planteada por Kam[70] las divide en dos grandes enfoques la fusión de sensores en nivel bajo y la fusión de sensores en nivel alto, la distinción radica en la forma en que se realiza la estimación. En la fusión en nivel bajo las medidas se integran directamente a fin de obtener las estimaciones, por otro lado en la fusión en nivel alto las medidas se integran indirectamente basados en estructuras jerárquicas.

En la fusión de nivel bajo se pueden mencionar tres tendencias, la primera es el uso de la arquitectura centralizada (figura 1.26) en la cual se tiene el conocimiento de la estocástica del sistema y los datos sensados, y en la que comúnmente se usa filtros de Kalman, y otras modificaciones del mismo como EKF (extended Kalman filter). Los filtros de Kalman son los elementos más usados en la literatura actual, Hong[71] usa los filtros de Kalman para integrar datos difusos y con ruido, mientras que Cox[72] usa filtros de Kalman en la fusión en datos obtenidos por odometría y sensores láser, en [73] Crowley usa filtros de Kalman para corregir los datos sensados por sensores ultrasónicos en base a su correspondencia. Otros ejemplos se pueden apreciar en [74], donde Tarin usa filtros de Kalman y una base de reglas en la fusión mejorando la estimación en la odometría y Lizarralde[75] que usa la fusión con filtros de Kalman para datos obtenidos por odometría y sensores ultrasónicos.

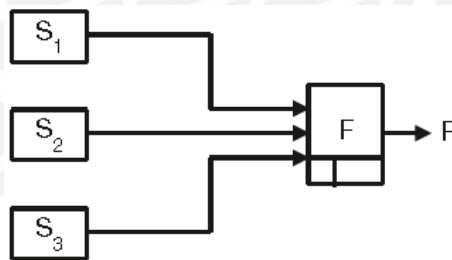


Figura 1.26 – Arquitectura centralizada de la fusión de sensores[4].

Mientras que en la segunda tendencia se usa una estructura descentralizada (figura 1.27), se usa además al igual que antes la estocástica para la fusión de datos, no obstante este enfoque considera el uso de nodos locales de fusión que permiten mejorar entre otras cosas la sensibilidad. En [76] Rao muestra una estructura descentralizada de fusión de datos sensados por cámaras CCD y barreras ópticas, Asadi y Bozorg [77] muestran por otro lado la fusión con una estructura descentralizada de sensores inerciales, encoders y GPS, y su uso en la estimación simultánea de la posición y el mapeo (método SLAM).

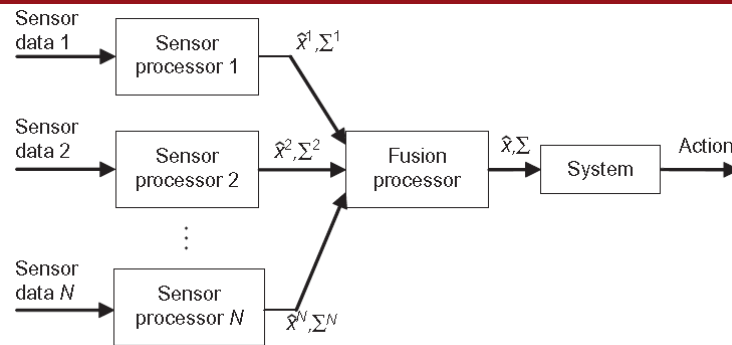


Figura 1.27 – Arquitectura descentralizada de la fusión de sensores

Una tercera tendencia se da cuando la estocástica es desconocida, enfocando el proceso a una fusión de sensores basada en reglas. Flynn muestra en [78] la implementación de una fusión de lecturas para sonares y sensores infrarojos mediante tres reglas heurísticas, Tarin [74] usa en la fusión además de filtros de Kalman, la elección de las medidas entre los encoders y el giroscopio basado en la comparación de los mismos con un umbral, otro ejemplo se muestra en [79], donde Enriquez realiza la fusión de sensores inalámbricos de red (WSN) y de radiofrecuencia ID (RFID), y para la fusión de los mismos en algunas condiciones críticas, en las que emplea reglas aritméticas sencillas.

La fusión de nivel alto en cambio presenta dos líneas principales; una arquitectura basada en conductas, en la cual el control se construye por medio de la unificación de conductas, las cuales muchas veces están constituidas de forma jerárquica. Este es también uno de los enfoques preferidos, y se pueden observar muchos ejemplos del mismo como en [80], donde Arkin utiliza una estructura jerárquica para integrar las conductas de docking y evasión de obstáculos, mientras que en [81], Adams usa una estructura de tres niveles, usando el nivel 0 para evasión de obstáculos con el método de campos potenciales, el nivel 1 para el seguimiento y el nivel 2 para el planeamiento de ruta. En otros casos se han empleado también sistemas de lógica difusa [5], [82], redes neuronales [7] y algoritmos genéticos [83], estos últimos enfoques se analizan con más detenimiento en la siguiente sección.

Por último el desarrollo de marcos unificados [70], principalmente basados en redes neuronales, se usa cuando la estocástica es desconocida o no está disponible. Un ejemplo de este enfoque puede verse en [84], en la misma se usan dos redes neuronales (denominadas por Nagata, de razonamiento y de instinto), la adaptación de las redes neuronales permite así ofrecer un marco unificado a la fusión de sensores, no obstante se debe tener un especial cuidado con los datos y el entrenamiento de las mismas.

1.5. Estado del Arte de los Controladores Neuro-difusos Aplicados a Robots Móviles

La lógica difusa o fuzzy, introducida por Zadeh en la década de 1960[85], basa su funcionamiento en el razonamiento humano experto, estableciendo para ello reglas lingüísticas y condiciones if-then. Ha sido usada desde entonces para la implementación de sistemas de control robustos en sistemas con poca o insuficiente información.

Una estructura clásica para el control en robots móviles se muestra en la figura 1.28, el controlador difuso se compone en general de una unidad de entrada fuzzificadora (IFU), un mecanismo de inferencia difusa (FIM), la unidad de salida defuzzificadora (OFU), y la base de reglas difusa. Se tiene a la salida del mismo la señal de control correspondiente para el movimiento del robot.

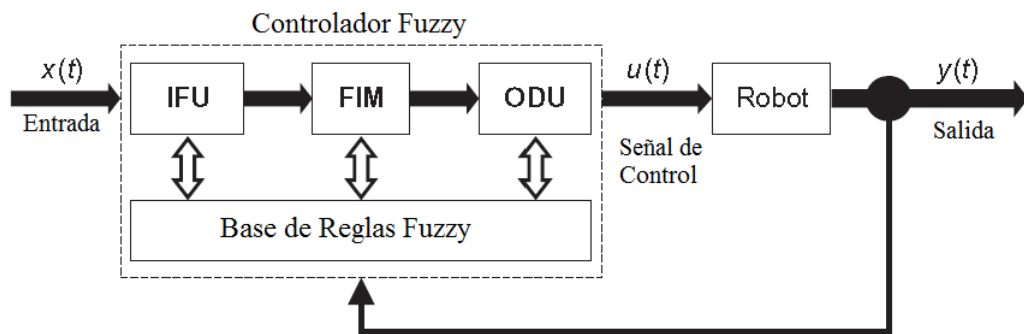


Figura 1.28 – Estructura básica de un controlador difuso para robots móviles[2].

Como ejemplos tempranos de su uso pueden mencionarse los trabajos de Sugeno y su implementación de un control para la trayectoria de un robot en un ambiente limitado por 2 paredes[86]; por otro lado en [87] Takeuchi muestra un controlador difuso para la prevención de colisiones. No obstante su buen desempeño, el controlador difuso presenta grandes desventajas, como son la dificultad en su diseño al carecer de una metodología sistemática, y el hecho de que sólo es viable para implementar algoritmos de navegación simples y con un objetivo único.

Por otra parte, las redes neuronales se componen de varias unidades centrales llamadas 'neuronas artificiales', las cuales dotan a la red de una capacidad de cálculo poderosa. Las redes neuronales tienen además la capacidad de aprender y generalizar. La estructura general de los controladores con aprendizaje supervisado se muestra en la figura 1.29, la red aprende en un periodo mediante el lazo superior y una vez terminado este, el neurocontrolador toma el control y el mecanismo de aprendizaje se desconecta.

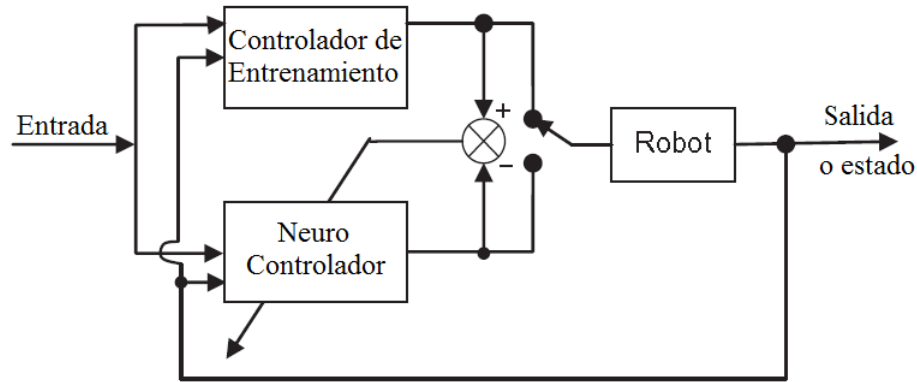


Figura 1.29 – Estructura de los neurocontroladores con mecanismo de aprendizaje [2].

Los dos tipos de redes neuronales más utilizados para el control son el de perceptrón multicapa (MLP) y el de funciones de base radial (RBF)[2]. Aplicaciones de las redes neuronales pueden verse en [88], donde se propone un controlador RBF para el ajuste en la orientación de un robot móvil diferencial, y en [89] donde se propone la integración de la red neuronal y un controlador cinemático a fin de asegurar la robustez. Estructuras más recientes siguen enfocando la unión de redes neuronales con otros controladores; en [90, 91] se proponen estructuras de redes neuronales adaptativas diseñadas en cooperación con controladores backstepping.

Los sistemas neuro-difusos (NFS), propuestos por primera vez por Jang[15] unen la lógica difusa con las grandes ventajas de las redes neuronales dadas por su capacidad de aprendizaje y adaptación. Los controladores neuro-difusos proveen así una herramienta poderosa, robusta y flexible para el procesamiento en una variedad grande de sistemas lineales y no-lineales. Las buenas características de los controladores neuro-difusos han logrado que estos gocen de una gran popularidad en muchos sectores de la tecnología y las ciencias sociales como puede verse en[92].

La aplicación de los controladores neuro-difusos han gozado desde su creación de una gran fama en la aplicación en robots móviles. Un sistema neuro difuso para el control del robot móvil en ambientes desconocidos basado en conducta es presentado por Li en[5], la estructura planteada en este caso se muestra en la figura 1.30, es necesario en este caso entrenar la red convenientemente, y en [6] Li presenta por otro lado la estructura base con la inclusión de un PID en el lazo de control para mejorar la trayectoria (figura 1.30). Otra aproximación temprana se muestra en [93], donde se diseñó un controlador para el seguimiento de paredes y evasión de obstáculos.

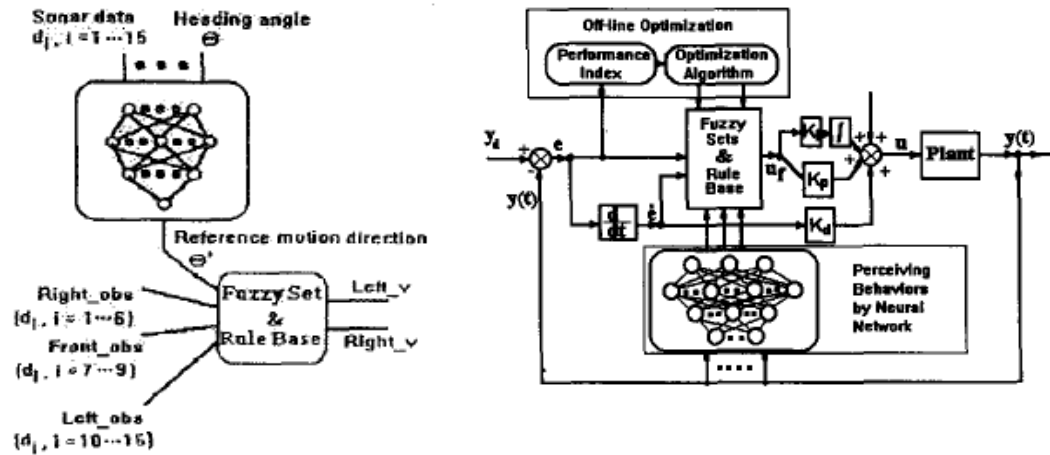


Figura 1.30 – Derecha: Diagrama del controlador neuro-difuso propuesto por Li para el control del robot móvil[5], Izquierda: Inclusión de un PID en la estructura de control[6].

En [7] Rusu plantea la estructura mostrada en la figura 1.31, la idea es la fusión de las conductas implementados en lógica difusa, mientras la conducta Go-Tangent esta implementada mediante un sistema adaptativo neuro difuso (ANFIS), usa además un bloque final para el caso en que dos o más conductas se activen. Khatoon muestra en[8] la fusión de las conductas y Goal-seeking mediante un sistema neuro-difuso (figura 1.32), la estructura permite mejorar los resultados que las conductas obtienen por separado. Por otra parte Song desarrolla en [9] un controlador mediante fusión de tres conductas, evasión de obstáculos, búsqueda de meta, y seguimiento de paredes, implementadas mediante lógica difusa; además usa una estructura FKCN (Fuzzy Kohonen Clustering Method) para calcular los pesos correspondientes.

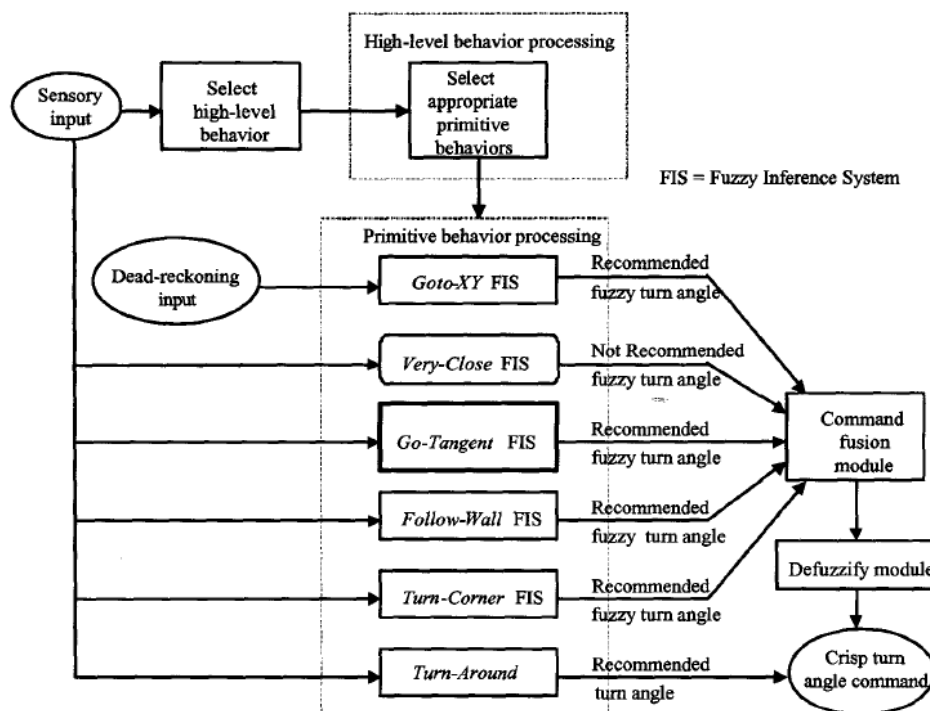


Figura 1.31 – Diagrama de bloques de la estructura propuesta por Rusu[7].

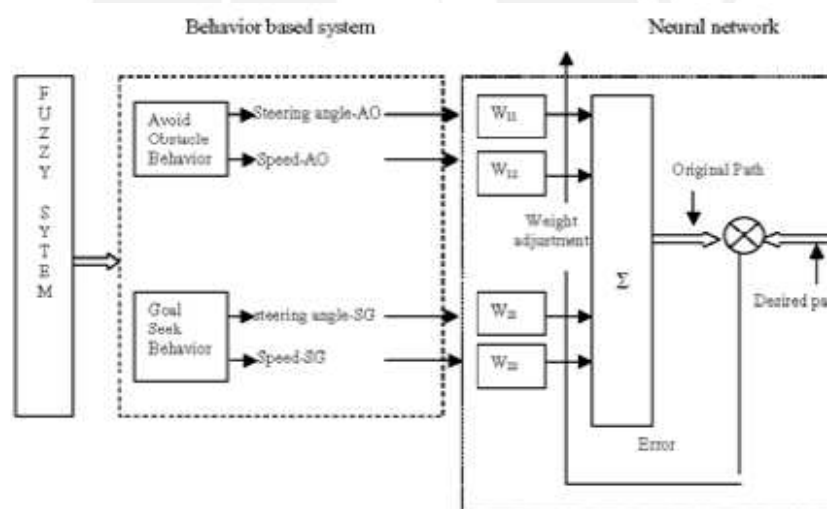


Figura 1.32 – Estructura del neuro controlador propuesto por Khatoon [8].

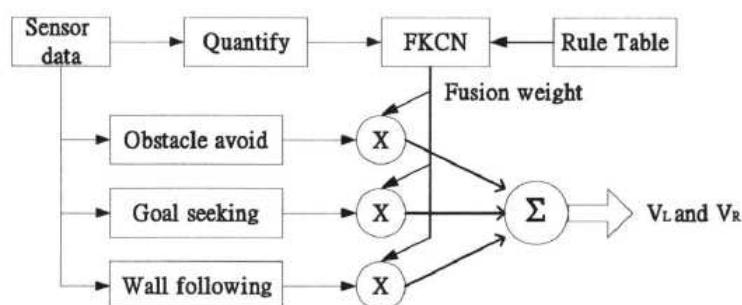


Figura 1.33 – Estructura propuesta por Song, un FKCN se usa en la fusión[9].

Por otra parte Zhang desarrolla en [94] un controlador con dos bloques difuso y una red neuronal basada en RAM (RAM-based a[95]) para mejorar la rapidez en los cálculos. Nurmani, usa igualmente neuronas RAM-based en [96] para integrar 4 conductas, obstacle-avoidance, left-wall following, right-wall following y goal-seeking[97]. Zerafa por otra parte desarrolla en [98] un controlador ANFIS para la evasión de obstáculos, adiciona además una regla para llegar a una meta.

1.6. Objetivos de la Tesis

Considerando la importancia que el desarrollo de máquinas inteligentes, tales como los robots móviles no holonómicos, viene representando para la investigación y la industria; y la aún insuficiente solución que existe para este problema, debido a las limitaciones impuestas por el sensado y la dificultad en el control de la navegación, esta tesis presenta el siguiente objetivo general:

“Desarrollar un sistema de navegación autónomo para robots móviles tipo carro basado en controladores neuro-difusos y fusión de sensores. “

Para poder alcanzar este objetivo es necesario realizar los siguientes objetivos específicos:

1. Obtener y simular los modelos cinemáticos de los robots móviles definidos para la implementación del sistema de navegación.
2. Diseñar el controlador para el planeamiento de la trayectoria en robots móviles mediante el uso de redes neuro difusas.
3. Simular mediante software los sistemas para el control de robots móviles tipo carro, usando para ello el modelo cinemático.
4. Elaborar el algoritmo para la implementación de la fusión de sensores, considerando encoders, sensores de radio frecuencia y otros elementos de sensado adicionales.
5. Implementar los algoritmos propuestos en un robot Pioneer P3-AT equipado con encoders, sensores de radio frecuencia y otros elementos de sensado adicionales.

Capítulo 2

Análisis Cinemático de los Robots Móviles Tipo Carro

2.1. Introducción

El diseño del sistema de control para el planeamiento de las trayectorias requiere de simulaciones que consideren las características dinámicas de los robots móviles en consideración, el método básico para afrontar este problema recibe el nombre de análisis cinemático.

El análisis cinemático en un robot móvil implica la obtención de la relación existente entre los parámetros geométricos representativos del movimiento y las restricciones en el mismo, considerando además la naturaleza no holonómica presente en la mayoría de vehículos.

Este análisis es muy particular para cada estructura del robot móvil a estudiar, ya que depende de la geometría y el medio de locomoción del robot móvil. Por este motivo antes de realizar su modelamiento es necesario la definición exacta de la estructura sobre la cual se ha de trabajar.

En la siguiente sección se deduce el modelo cinemático de los robots móviles tipo carro y tipo skid-steer, además se presentan las características del robot móvil Pioneer P3-AT, y por último se plantea la estructura del sistema de control a implementarse.

2.2. Modelo Cinemático de los Robots Móviles Tipo Carro

Para el análisis cinemático del robot móvil es común establecer presunciones sobre el movimiento del mismo que permiten que el modelado sea posible y que el análisis dinámico sea evitado, estos se pueden sumarizar en los 4 puntos siguientes[99, 100, 14]:

1. El robot móvil funciona a velocidades bajas.
2. El robot móvil se mueve sobre una superficie plana.
3. La fricción debido a la traslación en un punto de contacto entre la rueda y la superficie es suficientemente alta para que no ocurra ningún deslizamiento en el movimiento de traslación.
4. La fricción debido a la rotación en un punto de contacto entre la rueda y la superficie es suficientemente alta para que no ocurra ningún deslizamiento en el movimiento de rotación.

La figura 2.1 muestra la estructura del robot móvil tipo carro dentro de las coordenadas globales $\{O\}$, el estado en el movimiento del robot puede representarse por el vector p :

$$p = [x_Q, y_Q, \theta, \gamma]' \quad (2.1)$$

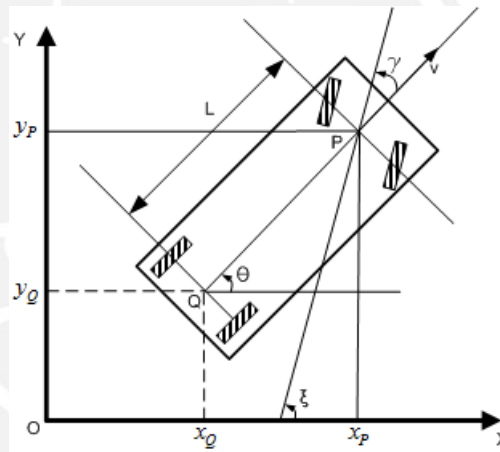


Figura 2.1 – Estructura cinemática de un robot móvil tipo carro (adaptado de [2]).

Donde se tiene que x_Q e y_Q son las coordenadas del robot móvil en el punto medio de su eje trasero, θ es el ángulo del robot con respecto a las coordenadas globales y γ el ángulo del timón y por ende el de las ruedas delanteras.

Las restricciones no holonómicas dadas en los puntos P y Q, se pueden deducir fácilmente de la geometría mostrada antes, y se transcriben respectivamente en las ecuaciones 2.2 y 2.3:

$$\dot{x}_Q \cos \theta + \dot{y}_Q \sin \theta = 0 \quad (2.2)$$

$$\dot{x}_P \cos(\theta + \gamma) + \dot{y}_P \sin(\theta + \gamma) = 0 \quad (2.3)$$

Además dado que existe una relación geométrica entre las posiciones de P y Q, deducible de las dimensiones del carro, se puede reemplazar estas relaciones para poder trabajar sólo con las coordenadas del último punto. Luego se tiene las restricciones

no holonómicas en forma matricial en la ecuación 2.4, nótese que esta matriz es no invertible.

$$\begin{bmatrix} -\sin\theta & \cos\theta & 0 & 0 \\ -\sin(\theta + \gamma) & \cos(\theta + \gamma) & D\cos\gamma & 0 \end{bmatrix} \begin{bmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\theta} \\ \dot{\gamma} \end{bmatrix} = 0 \quad (2.4)$$

Por otra parte, haciendo uso de la simplificación en el modelo bicicleta, se puede reducir el modelo de este robot móvil como en la figura 2.2, el mismo se constituye en la zona central del robot móvil y con base en el punto Q de la figura anterior, se añade además un marco de referencia local que nos permite distinguir la dirección en el movimiento.

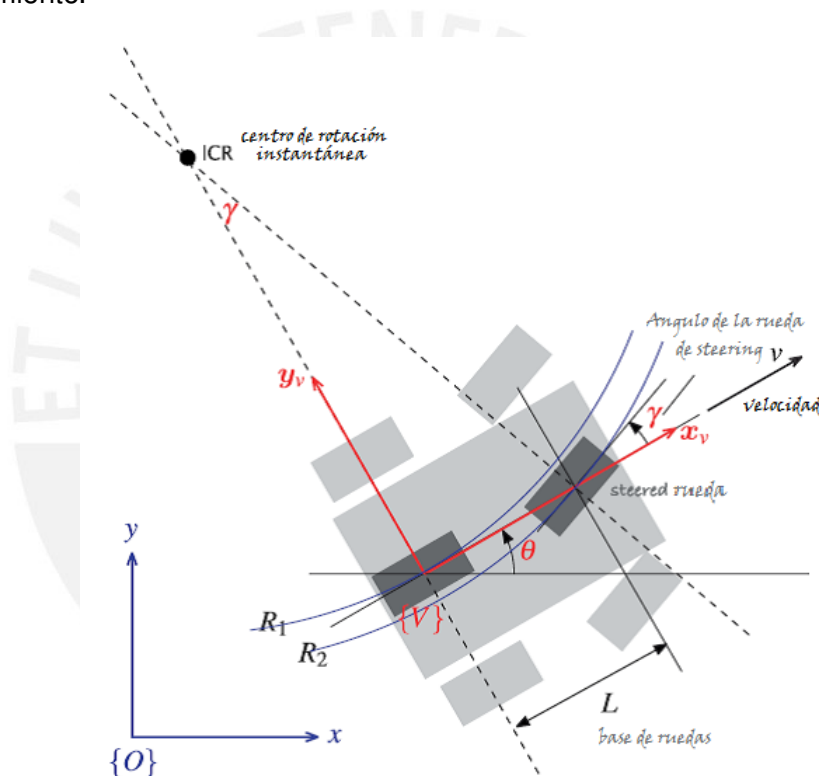


Figura 2.2 – Modelo del robot móvil tipo carro y su simplificación en modelo bicicleta[10].

En el marco $\{V\}$ el robot no puede avanzar en forma lateral ya que no hay deslizamiento, por lo cual su velocidad es enteramente en la dirección x . Por otra parte, se puede apreciar que existe un solo centro de rotación instantánea y dos radios de rotación R_1 y R_2 ; luego el punto de referencia (Q) sigue una trayectoria circular según

$$\dot{\theta} = \frac{v}{R_1} \quad (2.5)$$

De la que se desprende la ecuación 2.6; las ecuaciones 2.7 y 2.8 por otra parte representan las velocidades en x e y en el marco de referencia global. Estas tres representan las ecuaciones de movimiento en tiempo continuo

$$\dot{\phi} = \frac{v}{L} \tan \gamma \quad (2.6)$$

$$\dot{x} = v \cos \theta \quad (2.7)$$

$$\dot{y} = v \sin \theta \quad (2.8)$$

Y se pueden además deducir las siguientes relaciones en tiempo discreto[14]:

$$\Delta x_k = r \cos \theta_k \quad (2.9)$$

$$\Delta y_k = r \sin \theta_k \quad (2.10)$$

$$\Delta \theta_k = \frac{r}{D} \tan \gamma_k \quad (2.11)$$

Estas últimas ecuaciones pueden usarse para la simulación del robot móvil tipo carro que se muestran en los capítulos posteriores.

2.3. Modelo Cinemático del Robot Móvil Tipo Skid-steer

2.3.1. Presentación del Robot Móvil Skid-steer

La locomoción skid-steer es ampliamente usada en muchos vehículos terrestres como cargadores frontales, maquinaria agrícola, de minería y otros[11, 101]. Su mecanismo usa velocidad diferencial para su movimiento, no obstante un requerimiento especial para este es la presencia de deslizamiento cuando el robot gira, en la figura 2.3 se puede observar dos de los casos que se pueden presentar en el mismo.

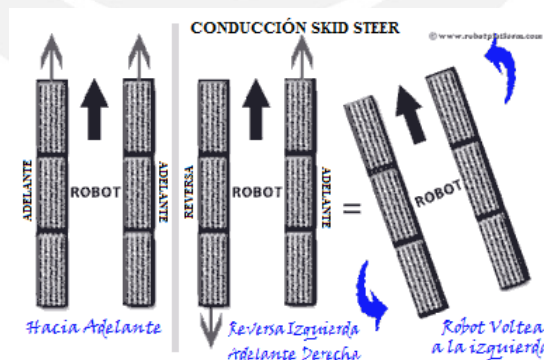


Figura 2.3 – Mecanismo de movimiento en los robots móviles skid-steer

Source: http://www.robotplatform.com/knowledge/Classification_of_Robots/wheel_control_theory.html

El movimiento hacia adelante o en retroceso requiere que ambas ruedas giren en el mismo sentido, mientras que el giro requiere que las ruedas de los lados izquierdo y derecho roten a diferentes velocidades. Esta estructura carece por ende de un

mecanismo de direccionamiento (timón) y provee al robot de una gran maniobrabilidad y controlabilidad, además de robustez y sencillez mecánica[11]. Otra ventaja del mismo es que puede funcionar en diferentes terrenos lo que lo hace bueno para la navegación en exteriores[102].

No obstante, el análisis cinemático del mismo es, por otro lado, bastante complejo debido a las interacciones y restricciones cinemáticas impuestas por este tipo de movimiento. El deslizamiento no permite definir el comportamiento futuro del robot sólo en base a las entradas, además las asunciones principales de no deslizamiento rotacional y translacional no pueden mantenerse.

Por estos motivos la navegación basada en odometría es especialmente complicada, hecho que hace necesario la estimación de los deslizamientos mediante sensores adicionales. Los sensores que se consideran para este fin son los inerciales[11, 101].

2.3.2. Análisis Cinemático del Robot Móvil Skid-steer

De la misma manera que en la sección anterior se consideran asunciones sobre el movimiento del robot skid-steer para hacer posible su modelado[102]

1. El centro de masa del robot esta localizado en el centro geométrico del cuerpo de su estructura.
2. Las dos ruedas de cada lado poseen la misma velocidad.
3. El robot realiza su movimiento en un terreno firme y las 4 ruedas están siempre en contacto con el mismo.

El esquema del robot puede verse en la figura 2.4, en este el control se realiza a través de las velocidades v_l y v_r en las ruedas izquierda y derecha respectivamente. Además para trabajar con los desplazamientos lineales en el plano se usa la transformación dada por la ecuación 2.12.

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = f \begin{bmatrix} v_l \\ v_r \end{bmatrix} \quad (2.12)$$

donde v_x y v_y representan las velocidades de traslación en el robot en las direcciones x e y, mientras que ω_z es la velocidad angular del mismo.

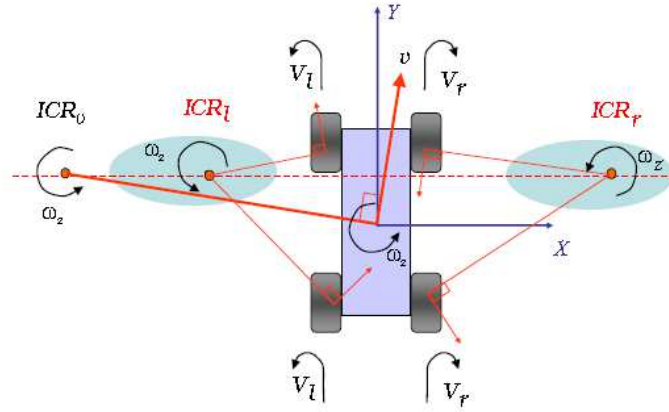


Figura 2.4 – Estructura cinemática de un robot móvil tipo skid-steer[11].

Las relaciones en un marco de referencia global podrían además deducirse sin inconvenientes a partir de la ecuación 2.13

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \quad (2.13)$$

Se debe notar además que se pueden distinguir tres centros de rotación instantánea, los cuales se encuentran además en una misma línea paralela al eje x [103]. Se considera el centro de rotación general ICR_G con coordenadas x_G e y_G , el centro de rotación a la izquierda ICR_l con coordenadas x_l e y_l , y el centro de rotación a la derecha ICR_r con coordenadas x_r e y_r .

Estas coordenadas pueden ser calculadas según las relaciones 2.14 - 2.17, se debe notar además que la coordenada y es la misma en los 3 casos para el marco de referencia local.

$$x_G = \frac{-v_y}{\omega_z} \quad (2.14)$$

$$x_l = \frac{\alpha_l v_l - v_y}{\omega_z} \quad (2.15)$$

$$x_r = \frac{\alpha_r v_r - v_y}{\omega_z} \quad (2.16)$$

$$y_G = y_l = y_r = \frac{v_x}{\omega_z} \quad (2.17)$$

Luego de las ecuaciones 2.12 y 2.14 - 2.17 se puede deducir la matriz de transformación de las velocidades v_l y v_r :

$$J = \frac{1}{x_r - x_l} \begin{bmatrix} -y_G \alpha_l & y_G \alpha_r \\ x_r \alpha_l & -x_l \alpha_r \\ -\alpha_l & \alpha_r \end{bmatrix} \quad (2.18)$$

Es apreciable además que en el caso en que el ICR del robot móvil descansa sobre el eje local de coordenadas X y de forma simétrica con respecto al eje Y , la matriz tendrá la forma siguiente:

$$J = \frac{\alpha}{2x_{ICR}} \begin{bmatrix} 0 & 0 \\ x_{ICR} & x_{ICR} \\ -1 & 1 \end{bmatrix} \quad (2.19)$$

donde se tiene que $x_{ICR} = x_l = -x_r$.

Las velocidades de traslación local pueden entonces encontrarse de la siguiente manera:

$$v_y = \frac{v_l + v_r}{2} \quad (2.20)$$

$$v_x = 0 \quad (2.21)$$

$$\omega_z = \frac{-v_l + v_r}{2x_{ICR}} \quad (2.22)$$

Se definen también dos parámetros fundamentales para una posible identificación del modelo cinemático, los cuales son la excentricidad λ y un término relativo a la eficiencia dado por χ :

$$\lambda = \frac{v_l + v_r}{v_r - v_l} = \frac{x_l + x_r}{x_r - x_l} \quad (2.23)$$

$$\chi = \frac{L}{x_r - x_l} \quad (2.24)$$

2.4. Presentación del Robot Móvil Pioneer P3-AT

El robot Pioneer P3-AT (figura 2.5) es una plataforma de desarrollo e investigación desarrollado por la fundación MobileRobots, el cual goza de una gran popularidad en el ámbito académico y de la investigación[102, 104, 11, 105, 106].



Figura 2.5 – Robot móvil Pioneer P3-AT.[12]

Cada robot P3-AT incluye una armadura de aluminio, el sistema de balanceo el cual incluye las 4 ruedas de tracción, motores DC reversibles, el control de movimiento del motor y su electrónica de potencia, encoders de alta resolución, y las baterías, todo manejado por un microcontrolador embebido e integrado con el software del servidor[13].

El robot P3-AT del laboratorio de control y automatización cuenta además con ocho sonares distribuidos en la zona delantera, como se muestra en la figura (2.6), los cuales facilitan en gran medida la implementación de algoritmos de evasión de obstáculos. El alcance máximo de cada sonar es de aproximadamente 5 metros[13].

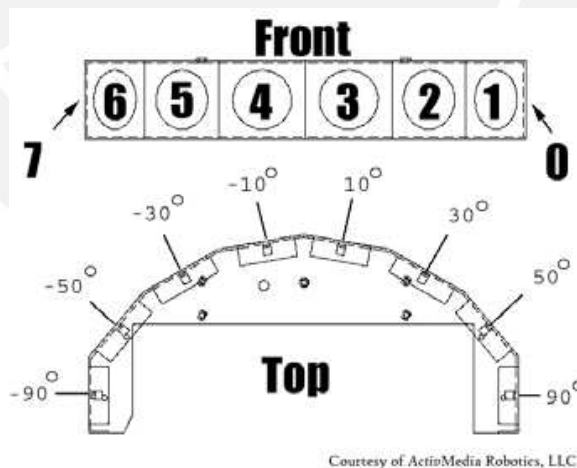


Figura 2.6 – Distribución de los sonares en el robot móvil P3-AT.[13]

Este robot cuenta también con una computadora a bordo, con sistema operativo Linux, y la librería ARIA proveída por MobileRobots para la programación en C y C++. Además la supervisión de la misma puede realizarse externamente desde una laptop cuando se implementa una red LAN, y para ello se cuenta con un antena RF y una red wifi local.

Se añadirán además a esta estructura básica los elementos de sensado en radiofre-

cuencia y elementos inerciales. Para esto se consideró un modelamiento del mismo basado en un microcontrolador Arduino y los sensores presentes en smartphones Android, este sistema se muestra en detalle en el capítulo 4.

2.5. Estructura de Control Planteada

La arquitectura de control planteada en la siguiente tesis se muestra en la figura 2.7. Se tiene en la misma una estructura jerárquica, la entrada estará compuesta por un grupo de sensores, los sonares para la detección y evasión de obstáculos, la odometría compuesta por los encoders y el giroscopio para la estimación de la velocidad y orientación, la unidad de movimiento inercial (IMU) para el reforzamiento de las medidas en la odometría y el sistema de sensores de radiofrecuencia (RF) para completar la localización.

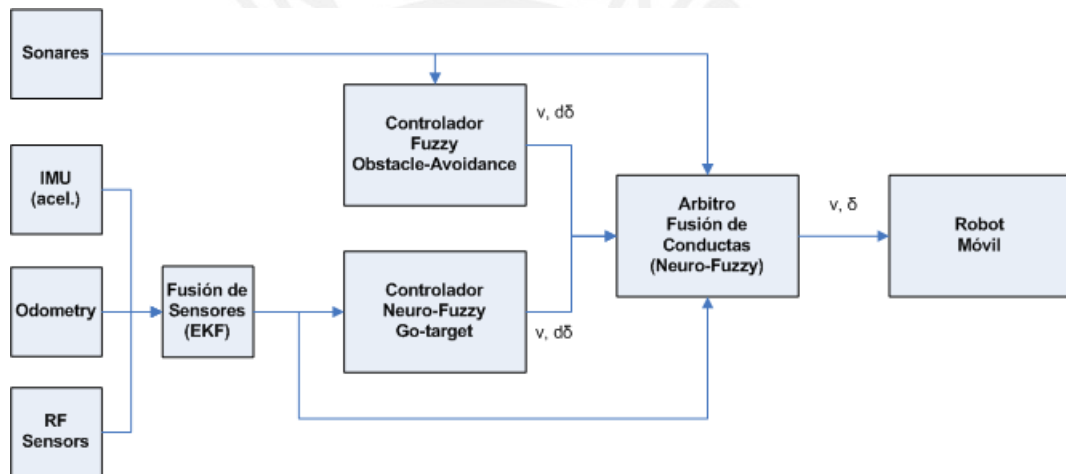


Figura 2.7 – Estructura del sistema de navegación planteado.

Los elementos de la IMU, la odometría y RF se integraran mediante fusión de sensores para disminuir la incertidumbre del sistema y poder proveer de datos confiables al control. En la parte de control se plantea el uso de controladores principales, un controlador difuso para generar la conducta de evasión de obstáculos y el controlador neuro-difuso para la navegación hacia el objetivo.

Se hace necesario además el uso de un arbitro para la definición y estimación de la señal de control final que se envía al robot móvil, el cual también se implementa mediante una estructura neuro difusa.

Capítulo 3

Diseño del Sistema de Navegación para Robots Móviles Tipo Carro usando Controladores Neuro-difusos

3.1. Introducción

Los sistemas de navegación en robots móviles vistos en detalle en la sección 1.3 se han desarrollado bajo diversas perspectivas que puede sumarse en dos grandes grupos, los enfoques reactivo y determinístico.

En el enfoque determinístico el planeamiento de la trayectoria puede llevarse a cabo bajo la presunción del conocimiento del mapa completo del entorno, lo cual no puede ser siempre posible y por ende presenta grandes desventajas. En el enfoque reactivo, por otra parte, el planeamiento se lleva a cabo durante el desplazamiento del robot, y permite afrontar el problema de la navegación en una forma más factible en la mayoría de aplicaciones.

La siguiente tesis afronta el problema de la navegación mediante el enfoque reactivo, basado además en la aplicación de dos conceptos presentados antes, la aplicación de conductas diseñados mediante sistemas difusos y la construcción de mapas basado en las cuadrículas de certeza[31, 107] además se hace uso del concepto de campos de fuerza en la misma forma que es empleado en métodos tales como VFF y VFH[17, 16].

Los sistemas difusos poseen una gran popularidad en los sistemas de navegación por su gran desempeño en situaciones poco convencionales, donde la toma de decisiones y el aprendizaje representan una gran ventaja. Bajo este punto de vista la aplicación de estos sistemas y en particular los sistemas neuro difusos permiten afrontar además las incertidumbres de diversas fuentes de sensado.

Por otro lado la construcción de mapas permite afrontar de una mejor forma la toma de decisiones, el algoritmo VFF permite en ese sentido, expresar los mapas de forma rápida en pequeñas celdas y usar la información contenida a continuación para la

evaluación de la mejor ruta, se presenta en la siguiente tesis una modificación de esta estrategia y su aplicación en forma conjunta con los sistemas neuro difusos.

En la siguiente sección se analiza el diseño de un sistema de navegación propuesto, con aplicación en los robots tipo carro y tipo skid steer, además se muestra las simulaciones del mismo para ambos tipos de robots y los resultados obtenidos para cada caso.

3.2. Fundamento de los Sistemas Neuro-difusos

Los sistemas neuro difusos presentan una gran variedad de estructuras, las diversas variantes de la misma se dan principalmente en dos formas: redes neuronales con capacidad de manejar y procesar información difusa, llamadas redes difusa-neuronales (FNN) y los sistemas difusos integrados con redes neuronales para mejorar algunas de sus características como su flexibilidad, velocidad y adaptabilidad, denominadas redes neuro difusas (NFS)[108][109].

En los sistemas FNN los valores de membresía de los sets difusos se integran ya sea en las entradas, los pesos y/o las salidas de la red neuronal; se incluyen además los sistemas con neuronas difusas por su capacidad de procesar información difusa, ejemplos pueden encontrarse en [110][111][112][113][114].

En los sistemas NFS, en cambio, el sistema se diseña para implementar el razonamiento difuso, siendo los parámetros de las conexiones en la red los que permiten implementar la lógica difusa. Se puede además identificar, en cierto grado, una correspondencia con la estructura de un sistema difuso, distinguiéndose en la misma los nodos con las condiciones de la premisa, la relación y las condiciones de consecuencia, ejemplos pueden apreciarse en [115][116][117].

En la mayoría de sistemas de este segundo tipo se puede distinguir una estructura con tres partes definidas, las cuales se aprecian en la figura 3.1, en la primera parte de la premisa se representan las funciones de membresía, las relaciones AND en forma de productos y la normalización, la segunda parte llamada de relación, en donde se agrupan convenientemente las resultantes normalizadas de la premisa y se conectan a la parte consecuencia; y por último la parte de consecuencia, en donde se calcula la salida en base a un esquema definido, en este caso el de Takagi-Sugeno o tipo 3.

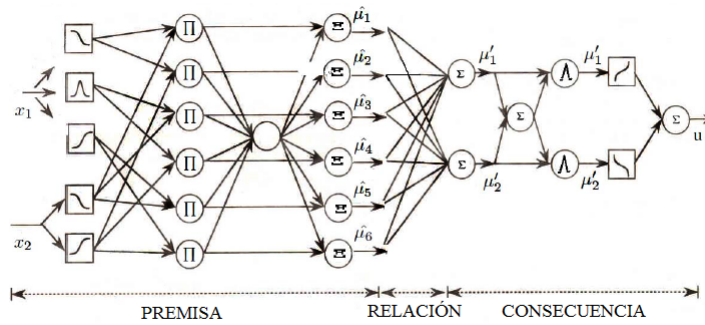


Figura 3.1 – Estructura de una red neuro difusa tipo TS [14]

La siguiente tesis está enfocada principalmente en la estructura Takagi-Sugeno (sistemas neuro difusos TS) que corresponde al tipo 3 en mecanismos de inferencia (ver figura 3.2) donde la salida corresponde a una combinación lineal de los términos inferidos de la entrada, por otra parte en la premisa es común utilizar funciones de membresía gaussianas y sigmoideas.

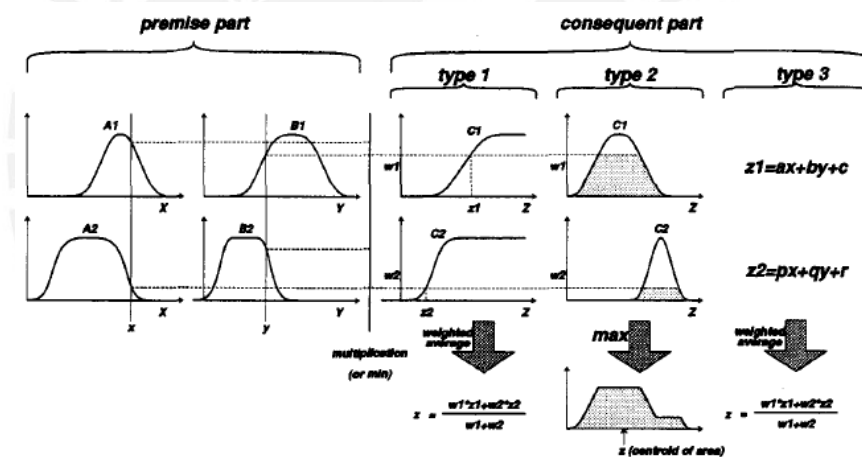


Figura 3.2 – Mecanismos de inferencia[15].

3.2.1. Sistemas Neuro-difusos TS

El modelo neuro difuso TS utiliza una red multicapa para la construcción de la lógica difusa, la estructura de esta red con múltiples entradas y múltiples salidas se puede apreciar en la figura 3.1(estructura ANFIS). La funcionalidad de cada nodo se distingue por su forma, los nodos circulares representan nodos con función fija, mientras que los nodos cuadrados representan elementos que pueden ser variables para la adaptación de la red.

Esta estructura se constituye de 5 capas (figura 3.3), y la función de cada una se detalla a continuación considerando la notación $O_{k,i}$ como la salida i-ésima de la capa k-ésima:

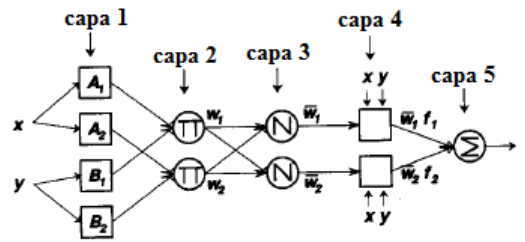


Figura 3.3 – Sistema neuro difuso con inferencia tipo 3[15].

Capa 1: Compuesta por nodos cuadrados, que representan las funciones de pertenencia ($\mu_{Ai}(x)$ o $\mu_{Bi}(x)$) para las variables de entrada. Los parámetros en esta capa se denominan parámetros de premisa y pueden ser modificados en el entrenamiento.

$$O_{1,i} = \mu_{Ai}(x) \quad (3.1)$$

Donde μ_{Ai} es usualmente del tipo gaussiana o sigmoidea:

$$gauss(x) = e^{-\left(\frac{x-c}{a}\right)^2} \quad (3.2)$$

$$sigmoid(x) = \frac{1}{1 + e^{-\left(\frac{x-c}{a}\right)}} \quad (3.3)$$

Capa 2: Esta compuesta por nodos circulares con la función de multiplicación Π y que representan la aplicación de las reglas. En general pueden ser implementadas por cualquier función norma-T que implemente la operación AND.

$$O_{2,i} = \mu_{Ai}(x)\mu_{Bi}(x) \quad (3.4)$$

Capa 3: Esta capa también consta de nodos circulares, en este punto se obtiene el ratio entre la i-ésima señal resultante de las reglas (w_i) con la sumatoria de todos los valores resultantes.

$$O_{3,i} = \frac{w_i}{\sum_j w_j} \quad (3.5)$$

Capa 4: En esta capa se consideran los nodos de consecuencia, en forma general se puede expresar la operación como el producto de los valores normalizados por una función lineal de las entradas, no obstante también se pueden definir pesos fijos como los valores representativos (f_i) de la consecuencia en el mecanismo de inferencia de salida.

$$O_{4,i} = \bar{w}_i f_i \quad (3.6)$$

Capa 5: Por último la capa 5 se constituye de un sumador para realizar la adición de todos los efectos en cada regla procesada por la red

$$O_{5,i} = \sum_j \bar{w}_j f_j \quad (3.7)$$

Los parámetros variables en el entrenamiento son así: 1) Las constantes en los sets difusos de entrada en la capa 1 y 2) Las constantes en el mecanismo de inferencia en la capa 4, los cuales se entrenan mediante el método de Steepest Descent, como sigue:

$$a_i = a_{i-1} - \eta \frac{dJ}{da} \quad (3.8)$$

$$b_i = b_{i-1} - \eta \frac{dJ}{db} \quad (3.9)$$

$$f_i = f_{i-1} - \eta \frac{dJ}{df} \quad (3.10)$$

Donde J representa la función de costo que se desea optimizar y η el factor de aprendizaje, el cual debe estar entre 1 y 0.

3.3. Desarrollo del Sistema de Navegación basado en Controladores Neuro-difusos

3.3.1. Estructura del Sistema de Navegación

La estructura del sistema de navegación diseñado se muestra en la figura 3.4, la actualización de celdas se realiza en similar manera que en el método VFF, haciéndose uso de una ventana dinámica de dimensión $w_s \times w_s$, lo cual permite construir un esquema del entorno y usarlo para la navegación, este bloque será tratado en la sección 3.3.2.

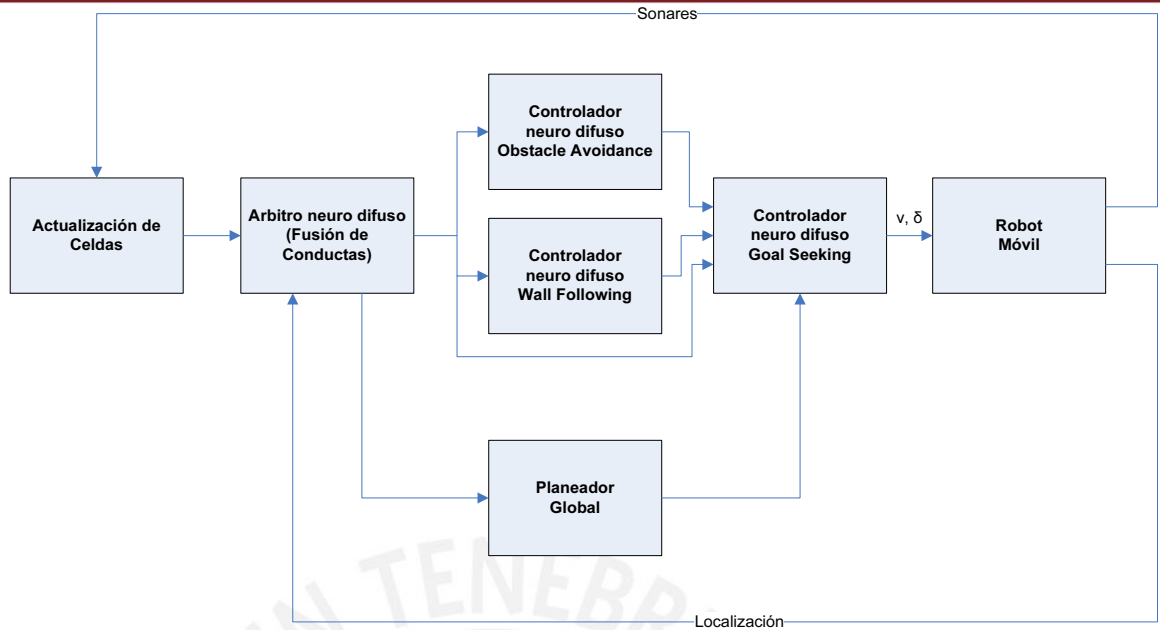


Figura 3.4 – Estructura global del sistema de navegación propuesto.

Para la obtención de la velocidad (v) y el ángulo de timón (δ) del robot, en cada instante, se implementaron tres conductas: búsqueda de meta (goal seeking), evasión de obstáculos (obstacle avoidance) y seguimiento de paredes (wall following); los cuales son tratados en los puntos 3.3.3, 3.3.4.2 y 3.3.4.3 respectivamente. Y para la adecuada convivencia entre estas conductas se elaboró además un controlador para la fusión de conductas (arbitro de conductas), el cual es detallado en el punto 3.3.5.

El sistema desarrollado no tiene como objetivo la búsqueda de una ruta óptima pues para ello es necesario el conocimiento previo del mapa del entorno, el entrenamiento, sin embargo, puede mejorar en gran medida las trayectorias obtenidas en un principio. Adicionalmente debido al proceso se irá construyendo durante la trayectoria un mapa aproximado mediante cuadrículas, el cual puede ser utilizado en última instancia, por ejemplo cuando el robot quede atrapado, para el planeamiento global mediante un método tradicional como A^* .

3.3.2. Construcción del Histograma de Cuadrículas de Certeza

La representación por el método de cuadrículas de certidumbre desarrollado por primera vez por Elfes y Moravec [31, 107], permite representar las medidas obtenidas por medio de elementos con incertidumbre, como los sonares, en una cuadrícula con valores de probabilidad. Este método es simplificado en [17] con el objeto de mejorar la velocidad de respuesta en tiempo real y la obtención del denominado histograma de cuadrículas, mostrado en la figura 3.5.

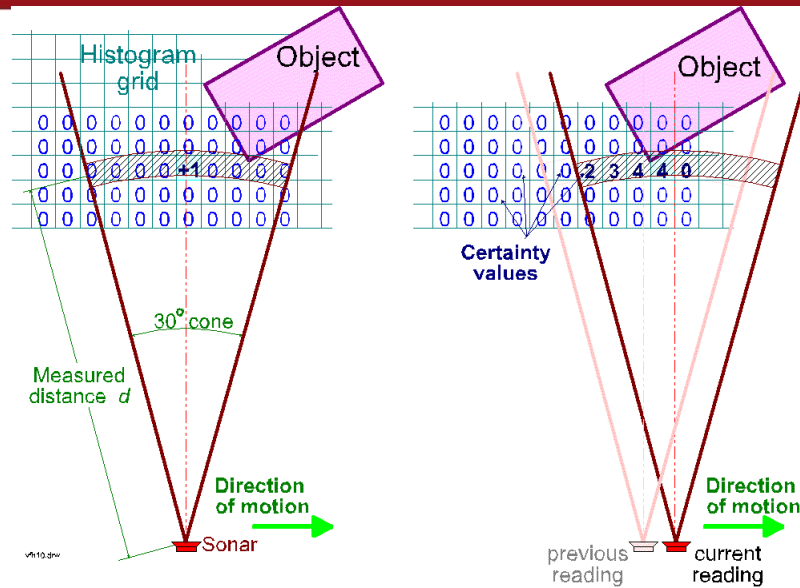


Figura 3.5 – Actualización del histograma de cuadrículas mediante medidas de sonares[16].

El histograma de cuadrículas esta compuesto por celdas las cuales almacenan un número, denominado comúnmente valor de certeza, y que se forma al ir incrementado en una unidad el valor de certeza de la celda cada vez que se registra mediante sensores, típicamente sonares, la presencia de un objeto en la coordenada correspondiente a la celda. Se debe notar además que la actualización se hace sólo en una celda por cada medida y considerando que la celda afectada es la correspondiente con la dirección central del cono del sensor.

Se usa en el procedimiento una ventana dinámica de dimensiones $w_s \times w_s$ que se mueve con el robot, esta ventana tiene por centro las coordenadas del robot y en las celdas el valor actualizado de los valores de certeza; esta ventana dinámica está contenida a su vez en una ventana global de dimensiones $w_E \times w_E$ que contiene las medidas históricas de todas las celdas y que se usa en última instancia para el planeamiento global, se considera el movimiento de la ventana semi-dinámica solo en el caso en que la ventana dinámica traspase el borde de la primera (figura 3.6).

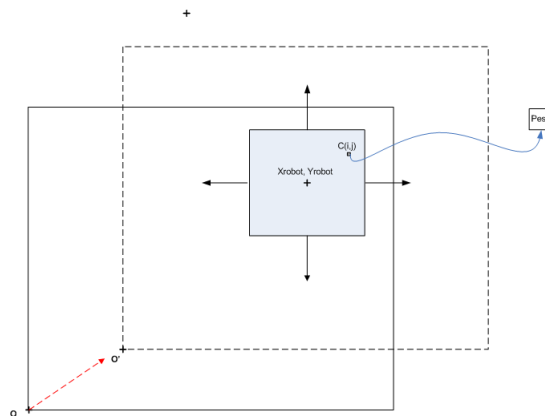


Figura 3.6 – Ventanas dinámica y semi-dinámica vistas en perspectiva

Para la actualización en las celdas se define un límite L en el valor almacenado en las mismas de forma que se uniformiza mejor la información contenida según 3.11. Por otra parte la ubicación a actualizar se define mediante 3.12 y 3.13.

$$\text{Actualización : } C(i, j) = \text{Min}(L, C(i, j) + 1) \quad (3.11)$$

$$i = i_R + \frac{\Delta x \cdot \cos(\phi_R + \phi_S) + \Delta x_S \cdot \cos(\phi_R + \phi_{SR})}{w_g} \quad (3.12)$$

$$j = j_R + \frac{\Delta y \cdot \sin(\phi_R + \phi_S) + \Delta y_S \cdot \sin(\phi_R + \phi_{SR})}{w_g} \quad (3.13)$$

No obstante, este método es una simplificación de las cuadrículas de certidumbre permite también reducir en cierto grado la influencia del error en la medición de los sonares y proporciona una herramienta muy útil para la navegación en medios estáticos.

3.3.3. Diseño del Controlador Neuro-difuso Goal Seeking

3.3.3.1. Controlador Goal Seeking con manejo en adelante

En este controlador neuro difuso se consideró como entrada la diferencia entre la orientación y el ángulo deseado, escogiéndose en este caso 3 variables lingüísticas: PO(diferencia positiva), CE(diferencia cero), NE(diferencia negativa). Las funciones de pertenencia correspondientes se muestran en la figura 3.7.

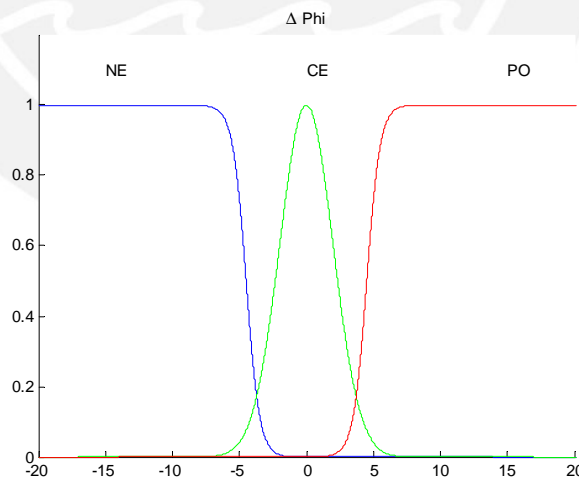


Figura 3.7 – Funciones de pertenencia de la conducta goal-seeking.

La variable de salida en este caso es el ángulo de timón y las reglas difusas se enuncian como sigue:

1. Si la entrada es PO entonces el ángulo del timón es máxima horaria.
2. Si la entrada es CE entonces el ángulo del timón es cero.

3. Si la entrada es NE entonces el ángulo del timón es máximo antihoraria.

Donde el ángulo de timón máximo es definido mecánicamente, y por lo general suele ser de 30° [14].

Para probar el controlador se simuló el comportamiento del robot móvil partiendo desde distintas ubicaciones y se brindó asimismo las coordenadas de la meta, la figura 3.8 muestra una primera prueba, el robot parte desde un ángulo θ de -40° y se mueve hacia la meta de coordenadas (5,5) con una velocidad constante.

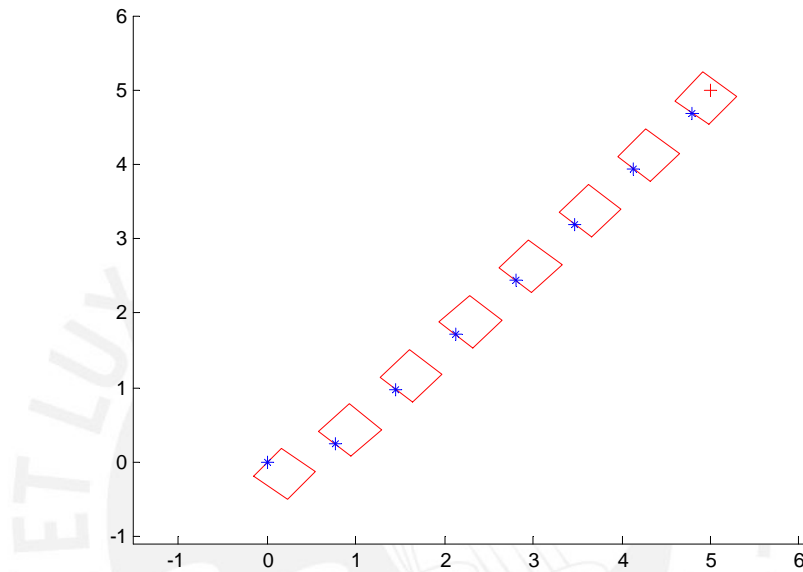


Figura 3.8 – Trayectoria del robot móvil para un ángulo de partida de -40° (la parte trasera corresponde a la marca azul).

3.3.3.2. Controlador Goal Seeking con manejo en adelante y retroceso

Para la extensión de esta conducta a la dirección de atraso es necesario aumentar el número de variables lingüísticas, la entrada es nuevamente la diferencia entre la orientación y el ángulo deseado, y en este caso 7 variables lingüísticas: MAH(mucho antihorario), RAH(regular antihorario), PAH(poco antihorario), CE(cero), PHO(poco horario), RHO(regular horario) y MHO(mucho horario). Las funciones de pertenencia se aprecian en la figura 3.9 junto con los rangos elegidos en un inicio para el funcionamiento de la red neuro difusa.

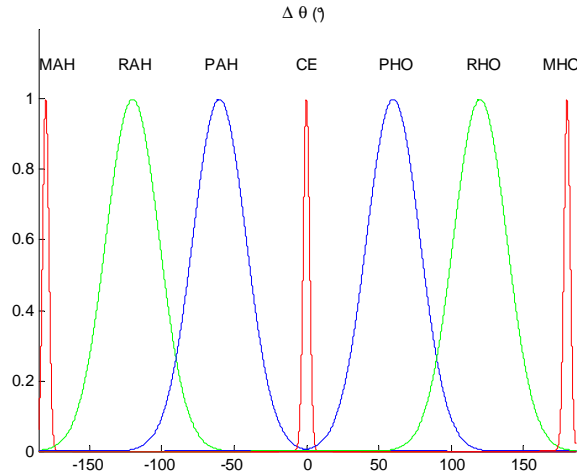


Figura 3.9 – Variables lingüísticas para la diferencia en la orientación.

Para lograr el manejo en las dos direcciones (adelanto y en retroceso), se establece dos centros de convergencia, uno alrededor de los 0° y el otro alrededor de los 180° (o -180°) de diferencia angular ($\Delta\theta$), además es preciso además incluir un factor de corrección para evitar según:

$$factor1 = -fdp(A3 - 180) \quad (3.14)$$

$$factor2 = -fdp(A5 + 180) \quad (3.15)$$

Donde $A3$ y $A5$ representan las funciones de pertenencia para POH y PAH respectivamente, y fdp representa la operación de función de pertenencia, además estos factores de corrección se aplican en forma de suma en las premisas 3 ($A3$) y 5 ($A5$) de la red neuronal.

Para probar el controlador se simuló el comportamiento del robot partiendo desde distintas ubicaciones y se brindó asimismo las coordenadas de la meta, la figura 3.10 muestra la prueba realizada, el robot parte desde un ángulo θ de 0° y se mueve hacia la meta de coordenadas (5,5) con una velocidad constante y en ambos sentidos, adelanto o retroceso, según convenga.

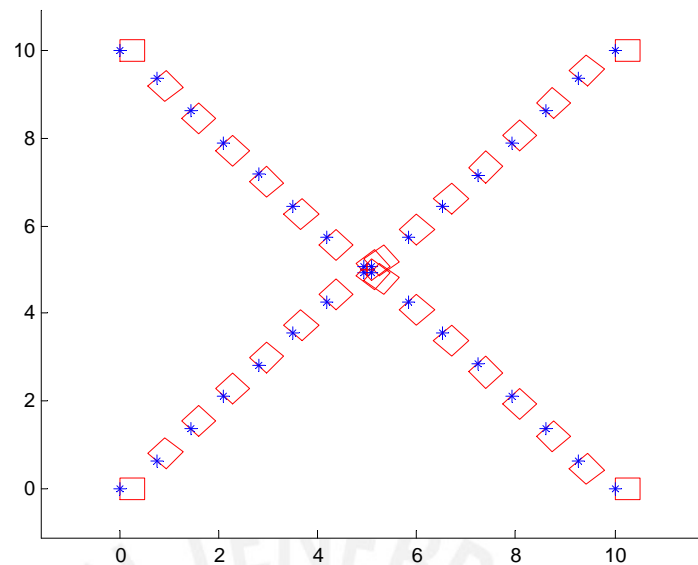


Figura 3.10 – Pruebas del controlador, la cruz azul marca la parte trasera del robot.

Esta conducta sólo es aplicable si el robot móvil posee sensores en las partes delantera y trasera, su implementación por ende estaría limitada por el hardware y no es desarrollada en lo sucesivo.

3.3.4. Diseño de Conductas Obstacle Avoidance y Wall Following

3.3.4.1. Cuantificación de fuerzas en la cuadrícula

Con el objetivo de la utilización de las cuadrículas de histograma por parte de los siguientes controladores, este se divide en 4 sectores o cuadrantes, definidos a partir de un ángulo de referencia, los cuales se muestran en detalle en la figura 3.11 y como puede observarse las denominaciones de cada cuadrante están inspiradas en los puntos cardinales.

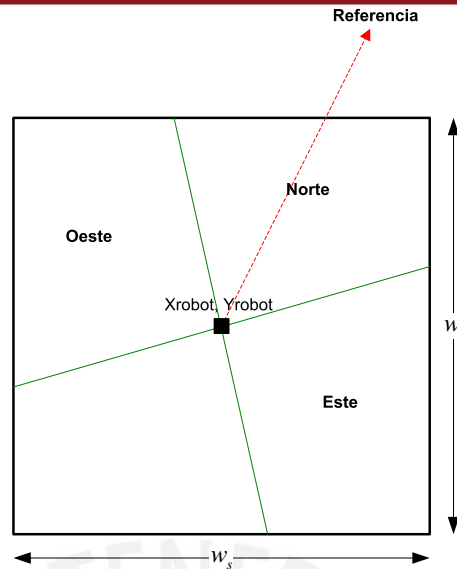


Figura 3.11 – Distribución de los cuadrantes en la cuadrícula de histogramas.

El ángulo de la referencia se establece como la orientación actual del robot, los sectores principales son el Norte, Oeste y Este; no obstante también se calcula las secciones Oeste-Norte, Norte-Oeste, Norte-Este y Este-Norte para mejorar la distinción en las conductas.

Seguidamente para cuantificar la información contenida en cada cuadrante se realiza el cálculo de las fuerzas resultantes por cada uno de ellos, el principio en este concepto está inspirado en los enfoques de navegación con campos de fuerza, como VFF[17]; y se puede trabajar con la siguiente relación:

$$F = K_F \sum_{i=i_0}^{i_F} \sum_{j=j_0}^{j_F} \frac{C(i,j)}{d(i,j)} \left(\frac{x_i - x_R}{d(i,j)} \hat{x} + \frac{y_j - y_R}{d(i,j)} \hat{y} \right), \quad (i,j) \in C_F \quad (3.16)$$

donde (i, j) son las coordenadas de las celdas que se definen en los rangos variables i_0 a i_F y j_0 a j_F y que pertenecen al cuadrante C_F de la cuadrícula; x_R e y_R son las coordenadas de la celda correspondiente a las coordenadas del robot, $C(i, j)$ es el valor contenido en la celda, $d(i, j)$ es la distancia de la celda (i, j) con respecto a la celda del robot y K_F es una constante de fuerza.

Este cálculo se detalla mejor en el algoritmo 3.1 mostrado a continuación, en donde se requiere como parámetro de entrada el ángulo deseado definido como $\alpha_{Deseado}$; y al término del mismo se deben obtener las 3 fuerzas necesarias para el procesamiento siguiente.

Algoritmo 3.1 Obtención de las fuerzas por cuadrantes en la cuadrícula de histogramas.

```

1  aCuadrantes = ObtenerCuadrantes(aDeseado);
2  for i=0 to N-1
3      for j=0 to M-1
4          if C(i,j)>0
5              a = ObtenerAngulo(xi-xR,y-yR)
6              if (a>aCuadrantes(1)) y (a<aCuadrantes(2))
7                  Feste = Feste + Kf*C(i,j)*((xi-xR)/d(i,j)
8                      +(yi-yR)/d(i,j))/d(i,j);
9              else if (a>aCuadrantes(2)) y (a<aCuadrantes(3))
10                 Fnorte = Fnorte + Kf*C(i,j)*((xi-xR)/d(i,j)
11                     +(yi-yR)/d(i,j))/d(i,j);
12             else if (a>aCuadrantes(3)) y (a<aCuadrantes(4))
13                 Foeste = Foeste + Kf*C(i,j)*((xi-xR)/d(i,j)
14                     +(yi-yR)/d(i,j))/d(i,j);
15             endif
16         endif
17     endfor
18 endfor

```

La cuantificación permite obtener tres parámetros identificadores de la presencia o ausencia de obstáculos en los cuadrantes analizados, y por ende puede usarse estos parámetros para el diseño de los controladores neuro difusos mostrados a continuación.

3.3.4.2. Controlador neuro difuso de evasión de obstáculos

En el diseño de este controlador se considera como entrada la fuerza en el cuadrante norte con tres variables lingüísticas según su intensidad: PEN (pequeña fuerza norte), REN (cero fuerza norte) y GRN (gran fuerza norte). Todas estas variables se aprecian en la figura 3.12 junto con los rangos elegidos.

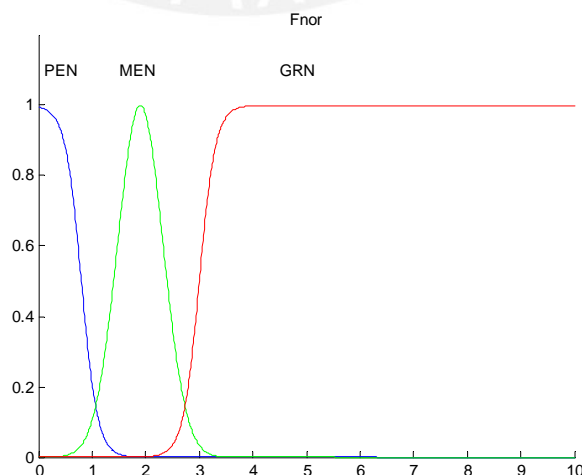


Figura 3.12 – Funciones de pertenencia de las variables de entrada.

Las variables de salida en este caso son, el ángulo de timón y la velocidad del robot y las reglas difusas se enuncian como sigue:

1. Si la entrada es PEN entonces el ángulo del timón es cero y la velocidad es máxima.
2. Si la entrada es REN entonces el ángulo del timón es medio y la velocidad es media.
3. Si la entrada es GRN entonces el ángulo del timón es máximo y la velocidad es lenta.

El ángulo del timón es limitado a 30° , y la velocidad máxima se define por la capacidad de cómputo y el tiempo de muestreo.

Además para la determinación del sentido en el timón (horario o antihorario) se usa como variables auxiliares la fuerza combinada en los cuadrantes

Además para la evaluación de esta conducta se realiza un ensayo con un entorno con diversos obstáculos que se muestra en la figura 3.13, y que permite apreciar la capacidad evasiva de la conducta para diferentes casos.

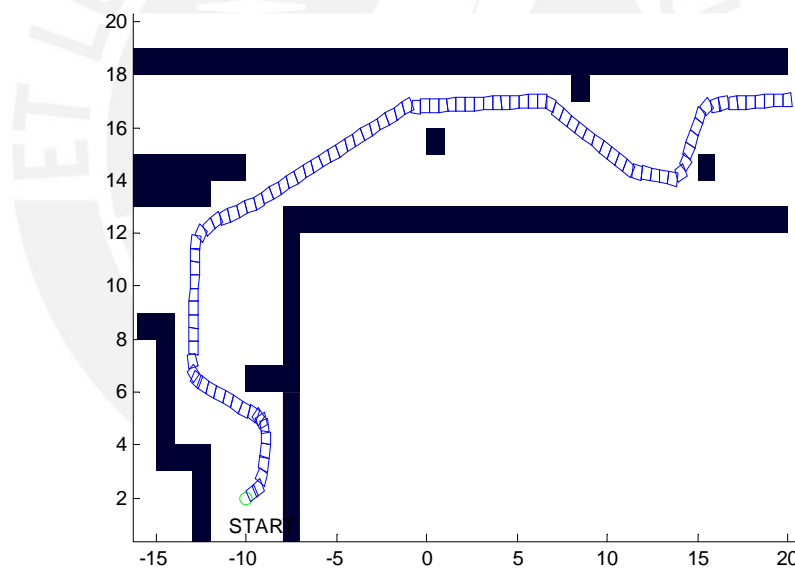


Figura 3.13 – Prueba de la conducta obstacle-avoidance.

3.3.4.3. Controlador neuro difuso de seguimiento de paredes

En el seguimiento de paredes se consideran dos conductas diferentes, dado que se pueden presentar dos casos: el seguimiento de paredes a la derecha y el seguimiento de las paredes a la izquierda. Para ello se diseñó las conductas tomando en cuenta dos magnitudes: la magnitud de la fuerza en adelante y la magnitud de la fuerza al lado, el objetivo de estas conductas es lograr que el robot móvil se alinee con la pared más cercana (al lado) para luego poder seguirla de cerca.

Las funciones de pertenencia se diseñan entonces, en ambos casos, usando la magnitud de la fuerza al lado y la magnitud de la fuerza en adelante; para la primera se tomaron las siguientes variables lingüísticas: MPEFS (fuerza al lado muy pequeña), PEFS (fuerza al lado pequeña), MEFS (fuerza al lado regular), GRFS (fuerza al lado grande), MGRFS (fuerza al lado muy grande); mientras que para la fuerza en adelante se tomaron las siguientes variables lingüísticas: MPEFA (fuerza en adelante muy pequeña), PEFA (fuerza en adelante pequeña), MEFA (fuerza en adelante media), GRFA (fuerza en adelante grande), MGRFA (fuerza en adelante muy grande). Estas funciones de pertenencia se muestran en la figura 3.14.

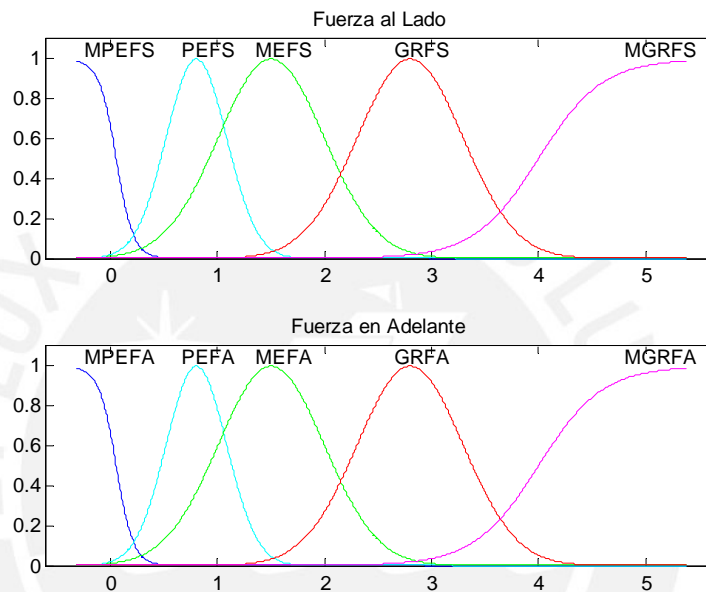


Figura 3.14 – Funciones de pertenencia de la conductas wall following.

En la figura 3.15 se muestra una prueba con el uso de sólo la conducta de seguimiento de paredes a la derecha, mientras que en la figura 3.16 se muestra el caso en el que sólo se usa la conducta de seguimiento de paredes izquierda.

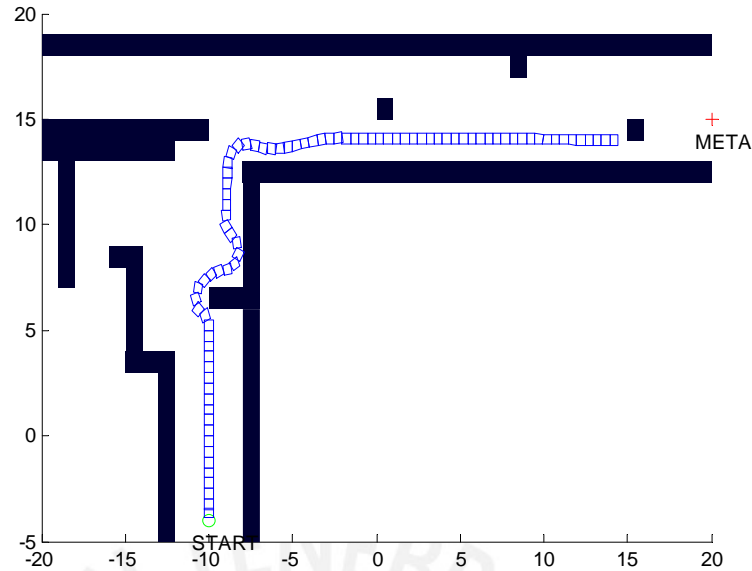


Figura 3.15 – Prueba de la conducta wall-following derecha.

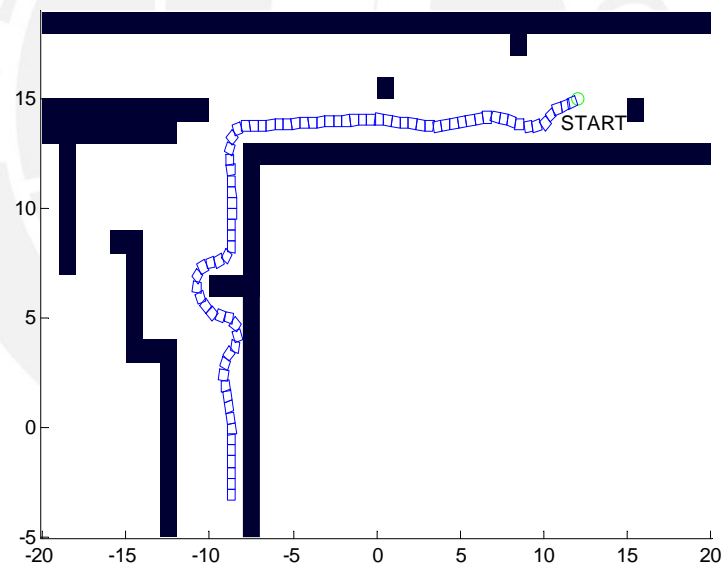


Figura 3.16 – Prueba de la conducta wall-following izquierda.

3.3.5. Diseño del Controlador Neuro-difuso para la Fusión de Conductas

El diseño del controlador para la fusión de conductas es la parte final de este sistema de navegación y sirve para regular los modos de operación del sistema de control, de forma que las conductas implementadas no interfieran entre ellas y su selección se realice coordinadamente.

En el diseño de este controlador se consideraron 4 parámetros para las funciones de pertenencia, en primer lugar la fuerza calculada en el cuadrante del norte, para lo cual se eligieron tres variables lingüísticas: PEN (pequeña fuerza norte), REN (regular fuerza norte) y GRN (gran fuerza norte); la segunda variable corresponde a la fuerza en el cuadrante oeste con las siguientes tres variables lingüísticas: PEO (pequeña fuerza oeste), CEO (cero fuerza oeste) y GRO (gran fuerza oeste), la tercera variable es la fuerza en el cuadrante este con otras tres variables lingüísticas: PEE (pequeña fuerza este), CEE (cero fuerza te) y GRE (gran fuerza este); y por último la cuarta variable es la diferencia entre el ángulo actual y el ángulo deseado el cual también posee tres variables lingüísticas: DAH (diferencia antihoraria), DCE (diferencia cero) y DHO (diferencia horaria). Las funciones de pertenencia correspondientes se muestran en la figura 3.17.

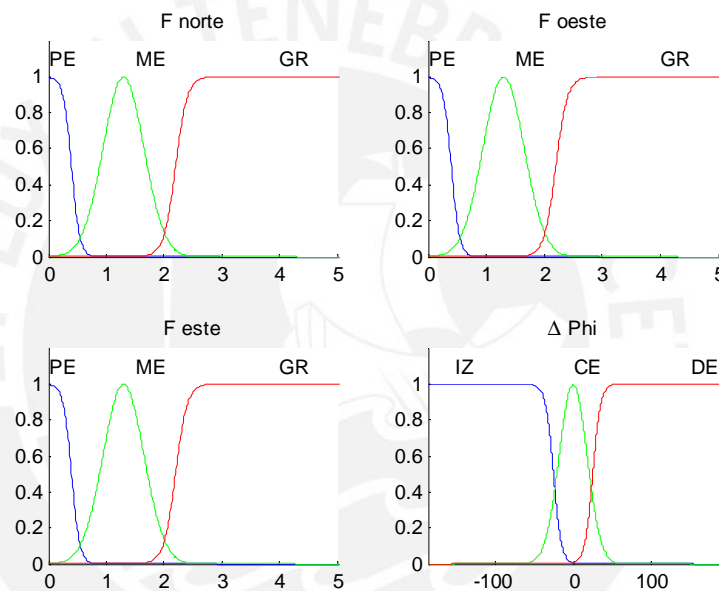


Figura 3.17 – Funciones de pertenencia de la red neuro difusa de fusión de conductas.

La base de reglas se constituye de 81 reglas y su elección debe realizarse cuidadosamente, sin embargo, el diseño de la misma puede llevarse a cabo según muchos criterios por lo cual se puede obtener más de una respuesta correcta. Para el diseño de la base de reglas mostrada en la Tabla 3.1, el criterio considerado de mayor importancia fue el de evadir un obstáculo, lo cual permite mejorar la seguridad en el desplazamiento del robot.

La notación usada en la tabla es la siguiente:

GS: Búsqueda de meta.

OA: Evasión de obstáculos.

WF: Seguimiento de paredes.

Tabla 3.1 – Base de reglas para el arbitro de fusión de conductas.

		DAH			DCE			DHO		
		PEN	REN	GRN	PEN	REN	GRN	PEN	REN	GRN
PEO	PEE	GS	GS	OA	GS	GS	OA	GS	GS	OA
	REE	GS	GS	OA	GS	OA	OA	WF	WF	OA
	GRE	GS	GS	OA	GS	WF	OA	WF	WF	OA
REO	PEE	WF	WF	OA	GS	OA	OA	GS	GS	OA
	REE	WF	WF	OA	GS	OA	OA	WF	WF	OA
	GRE	WF	WF	OA	GS	WF	OA	WF ⁶⁰	WF	OA
GRO	PEE	WF	WF	OA	GS	WF	OA	GS	GS	OA
	REE	WF	WF	OA	GS	WF	OA	WF	WF	OA
	GRE	WF	OA	OA	GS	OA	OA	WF	OA	OA

Para ejemplificar el proceso de la elección de base de reglas se toma el caso 60 de la Tabla 3.1, correspondiente a la columna 7 y la fila 6; el estado del robot en este caso puede visualizarse en la figura 3.18, el robot avanza en dirección vertical positiva y tiene un obstáculo claramente apreciable a su derecha, otro obstáculo a relativa distancia a su izquierda y una casi nula o nula presencia de obstáculos en adelante, además la meta se encuentra establecida por el $\phi_{deseado}$.

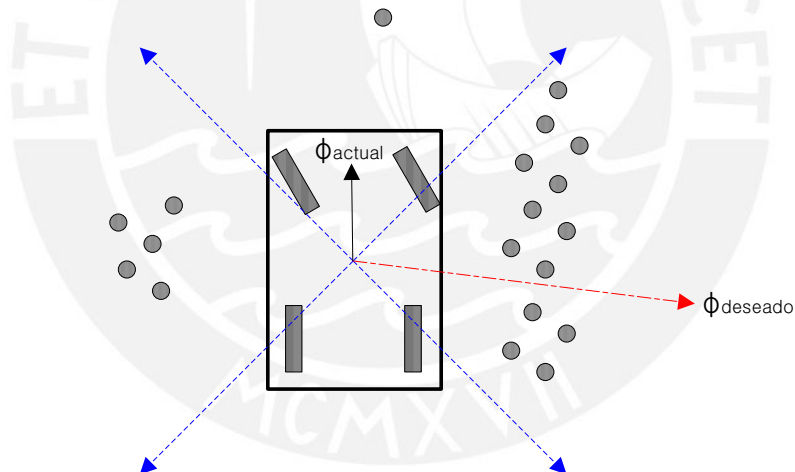


Figura 3.18 – Un caso particular en el movimiento del robot.

Dado este caso dos asumpciones pueden establecerse: 1. el robot no puede virar hacia la meta en esta posición y 2. el robot no se encuentra en una situación de evasión de obstáculo pues el obstáculo en adelante es nulo; por ende basándonos en estas observaciones se puede concluir que es necesario emplear una conducta de seguimiento de paredes tal como aparece en la Tabla 3.1.

3.4. Simulación del Sistema de Navegación para el Robot Móvil Tipo Carro

3.4.1. Simulación en MATLAB

El sistema de navegación fue implementado en MATLAB para observar su respuesta, además se modelaron sonares y diferentes entornos para la realización de las pruebas en diferentes escenarios. El algoritmo se compone de: construcción del grid, obtención de las fuerzas por cuadrantes y uso de los controladores de planeamiento y locomoción según la figura 3.1.

Para las simulaciones que se muestran a continuación se consideraron las siguientes asunciones y condiciones:

- El robot sólo se mueve hacia adelante.
- El ángulo máximo del timón del robot es de $\pm 30^\circ$.
- El robot posee un total de 16 sonares distribuidos en toda la periferia del robot y en los mismos ángulos encontrados en un robot real (Ver Manual P3-AT[13]).
- Sólo se consideran obstáculos estáticos.
- No existe ningún conocimiento previo sobre el entorno.

Las figuras 3.19 y 3.20 muestran un entorno con obstáculos en forma de paredes paralelas, se puede apreciar que el robot es capaz de maniobrar desde cualquier posición inicial en este caso.

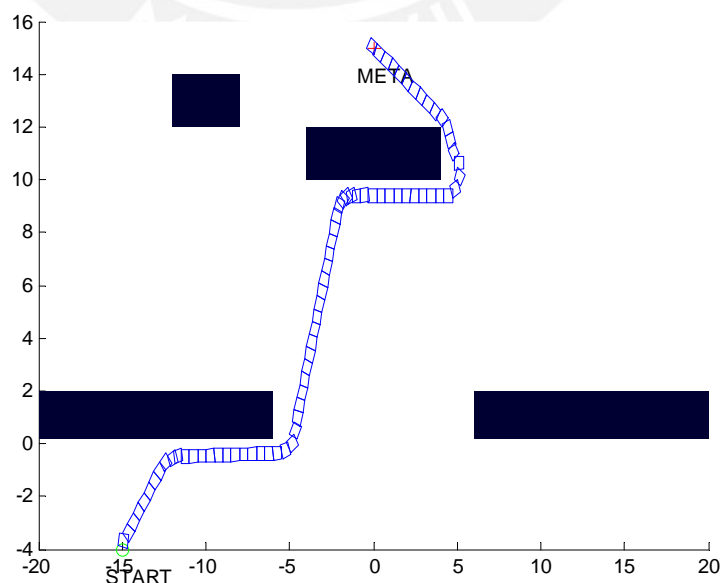


Figura 3.19 – Entorno de prueba 1, punto de partida 1.

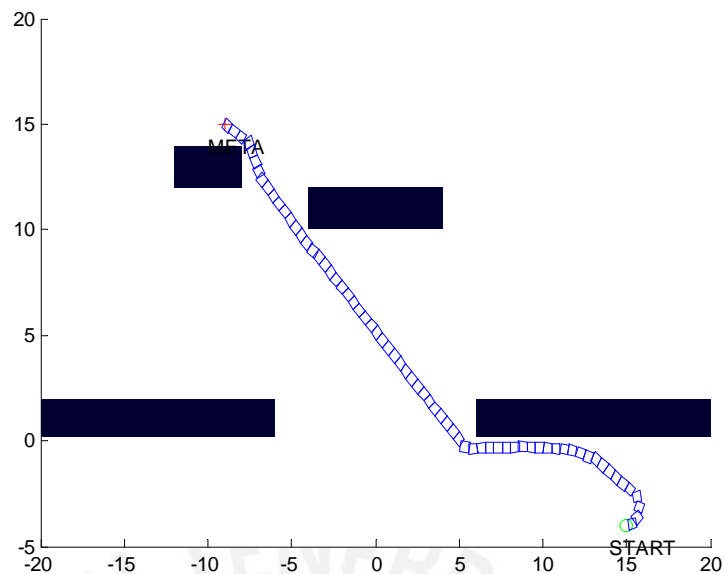


Figura 3.20 – Entorno de prueba 1, punto de partida 2.

La figura 3.21 muestra el ensayo en el entorno empleado en anteriores puntos, en este caso se incluye ruido gaussiano con media $\mu = 1$ y desviación estándar $\sigma = 0,5$ en los sonares para evaluar el desempeño del controlador. La figura 3.22 muestra el histograma de cuadrículas obtenido tras el proceso.

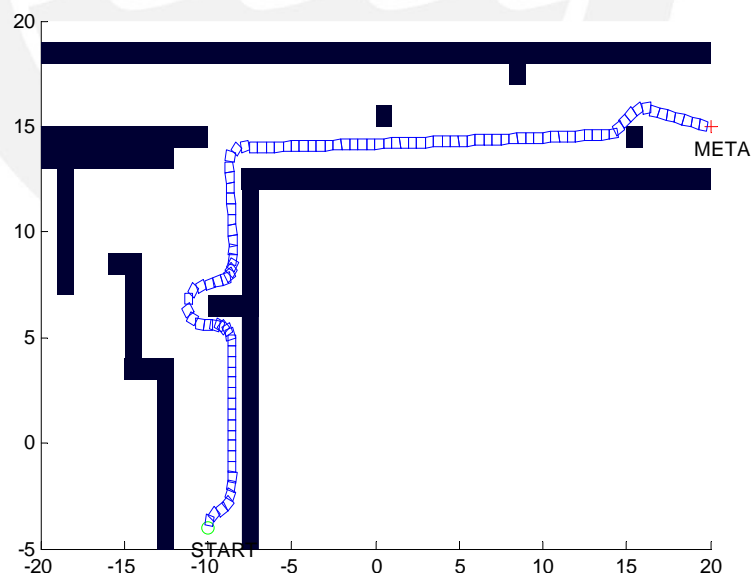


Figura 3.21 – Recorrido en entorno de prueba 2.

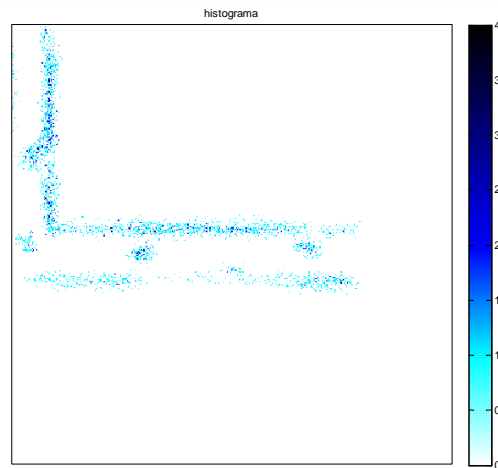


Figura 3.22 – Histograma de cuadrículas obtenido tras el ensayo.

En la figura 3.23 se muestra la trayectoria del robot cuando se tiene otro entorno típico en interiores con espacios cuadrados a manera de salones, se puede observar en este caso que el robot es capaz de maniobrar en espacios estrechos como las aperturas en los salones.

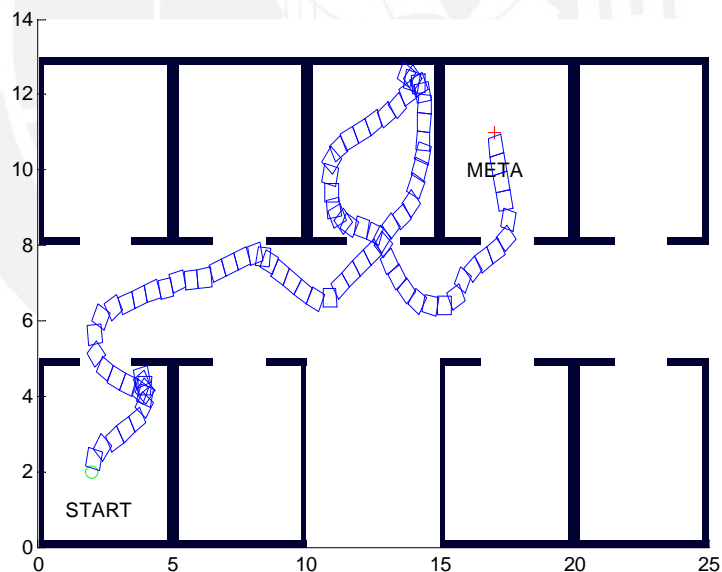


Figura 3.23 – Entorno de prueba 3.

Por último en la figura 3.24 se muestra el desempeño del robot cuando se tienen obstáculos en forma de trampas, se muestra en este caso un espacio grande en 'U' que retiene al robot ante el avance. En este caso se requiere de la activación del algoritmo de escape para poder seguir el recorrido, que como se observa en esta figura puede después llevarse a cabo sin problemas.

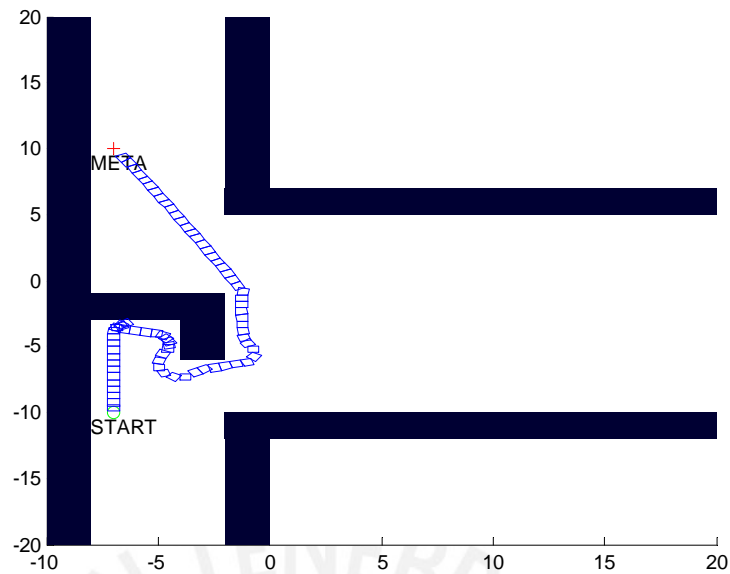


Figura 3.24 – Entorno de prueba 4.

En la figura 3.25 se observa un entorno tomado de [118], nuevamente la trayectoria del robot permite llegar a la meta sin problemas.

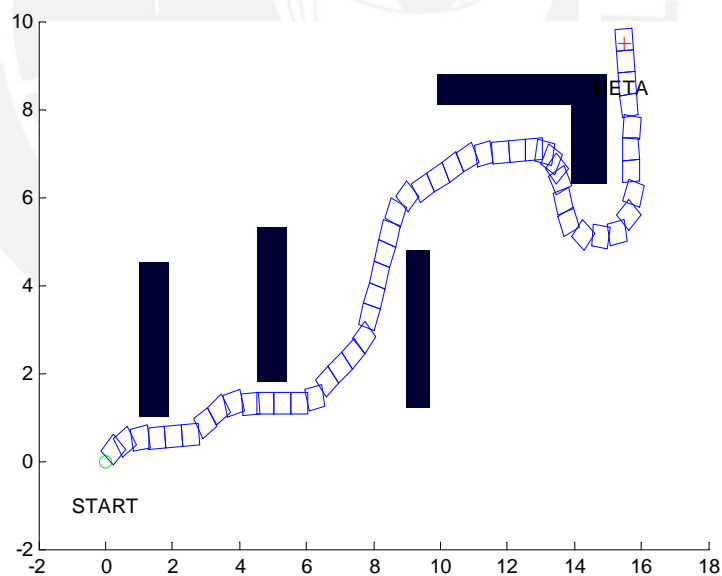


Figura 3.25 – Entorno de prueba 5.

En la figura 3.26 se observa la trayectoria en un entorno con dos paredes paralelas, adaptado de [8], donde se puede observar que el robot puede evitar obstáculos en ambos sentidos.

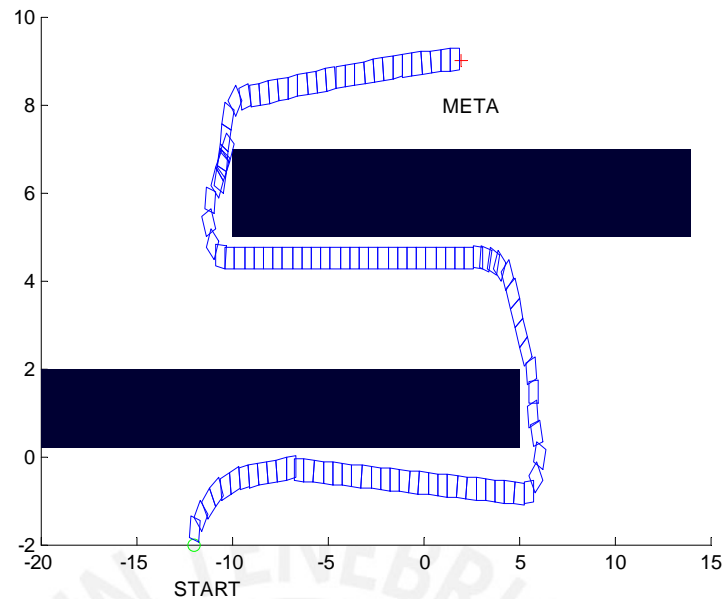


Figura 3.26 – Entorno de prueba 6.

3.4.2. Interfaz de Usuario en GUIDE

Las simulaciones en MATLAB se complementaron además con la creación del interfaz de usuario mostrado en la figura 3.27, donde se permite la creación por el usuario de nuevos obstáculos adicionales a los mostrados en la sección 3.4.1; y la inserción de las condiciones iniciales y las deseadas para la meta.

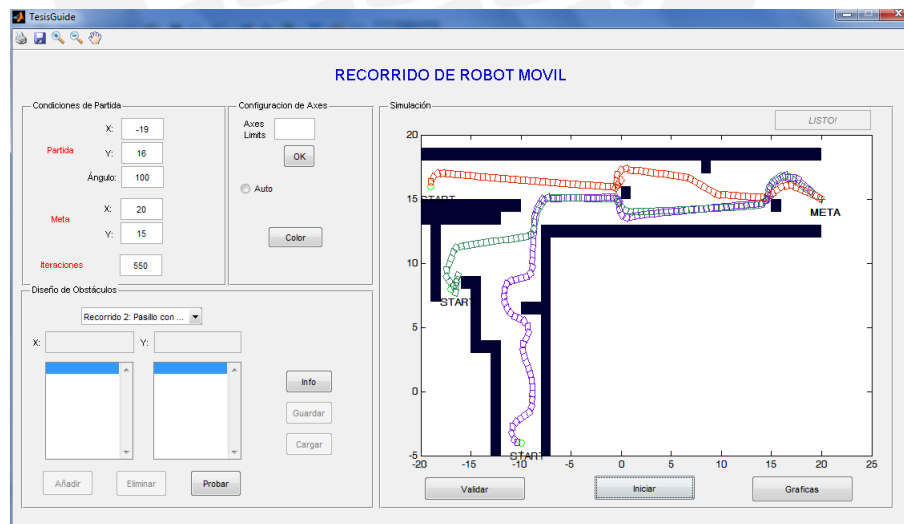


Figura 3.27 – Interfaz de usuario en GUIDE de MATLAB.

Se tiene además la posibilidad de visualizar la simulación del robot simulado en línea dentro de la figura del entorno, en la que se permite además la maniobra con elementos de acercamiento/alejamiento, mover y fijación de los rangos en las coordenadas.

En las figuras 3.28 y 3.29 se muestran algunas de las simulaciones que pueden obtenerse mediante la interfaz desarrollada y que representa una mejora para la comprensión del sistema de navegación por parte del usuario.

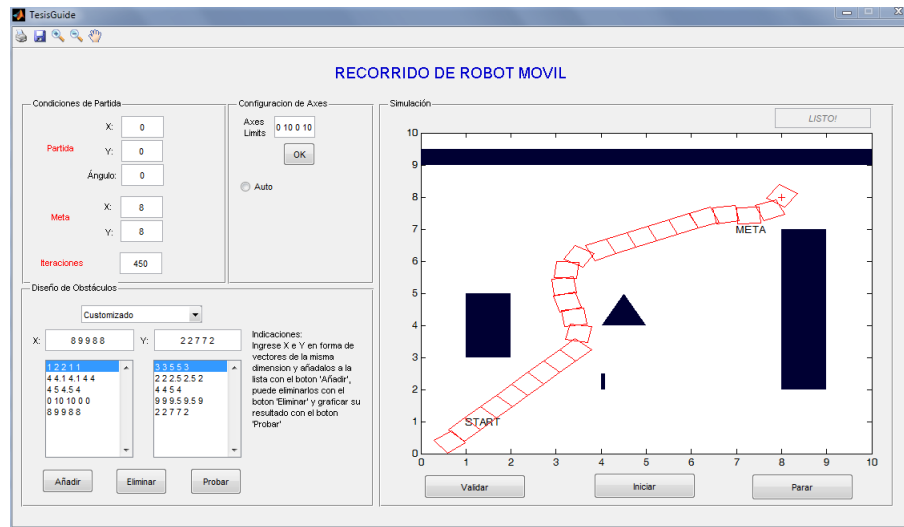


Figura 3.28 – Prueba 1 con edición de obstáculos.

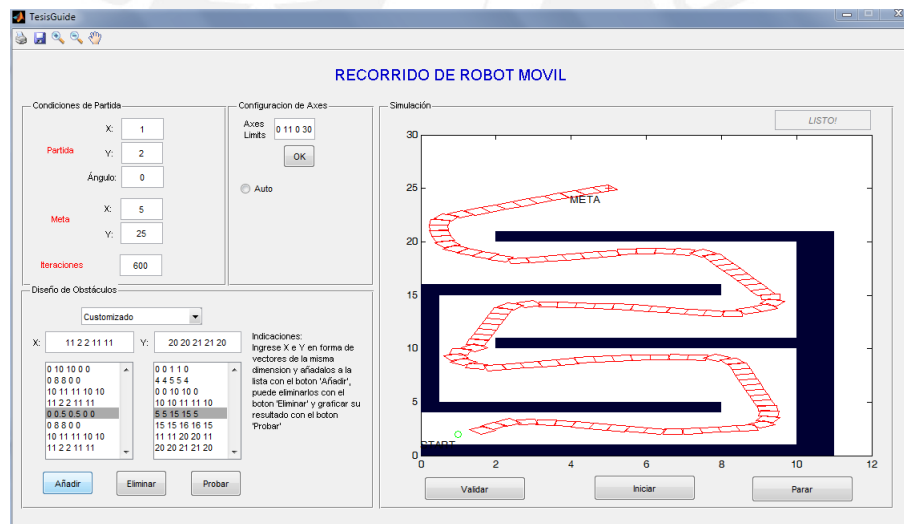


Figura 3.29 – Prueba 2 con edición de obstáculos

3.5. Simulación del Sistema de Navegación para el Robot P3-AT implementado en Stage

3.5.1. Introducción a Software Player/Stage

Player es un software libre desarrollado por el proyecto Player, el cual se usa en la investigación y desarrollo de robots y sistemas de sensado[119]. El proyecto ha implementado también los software complementarios Stage y Gazebo para su uso con Player.

Player implementa las interfaces a los componentes de hardware del robot y sensores de una gran variedad robots móviles y accesorios. Su estructura está basada en el modelo cliente/servidor que permite la ejecución de los programas de control del robot en cualquier computadora con acceso a la red del robot, siendo también posible la conexión de clientes en forma concurrente.

Stage es un simulador que soporta la implementación de múltiples robots móviles en forma conjunta, que muestra su movimiento en un plano de dos dimensiones. Stage, al igual que Player, tiene una gran variedad de modelos de sensores implementados, incluyendo entre estos los sonares, sensores láser y cámaras pan-tilt-zoom[120].

Gazebo, por su parte, permite simular un conjunto de robots móviles al igual que Stage, pero en un entorno 3D.

Un esquema de las conexiones entre estos software se muestra en la figura 3.30, Player permite la conexión con el hardware físico, no obstante para la simulación se puede hacer uso de Stage, el cual implementa una interface estándar para Player. Player realiza la conexión con cualquiera de las tres opciones presentadas y brinda soporte para las conexiones de clientes remotos.

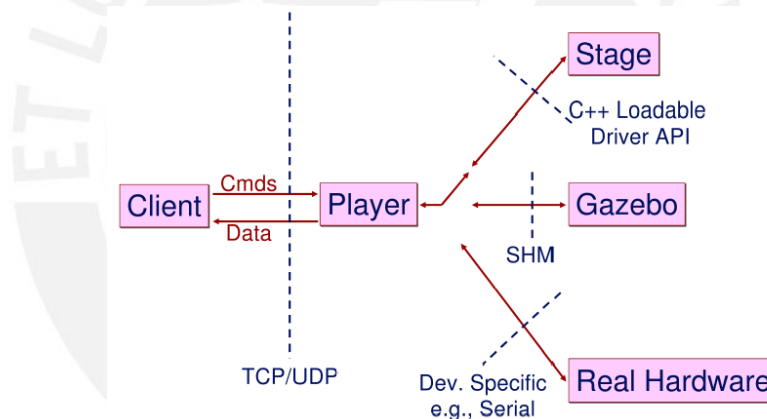


Figura 3.30 – Esquema del funcionamiento de Player/Stage.

Es de resaltar que la conexión entre Player y Stage se realiza de la misma forma que la que se daría con el hardware, y por ende, se requieren mínimos o nulos cambios cuando se traslada la implementación del simulador al hardware.

3.5.2. Simulación en Player/Stage

Para las simulaciones en Stage se configura el entorno mediante el archivo .world (Algoritmo 3.2) en donde se establecen el tamaño y otros datos de la ventana, el mapa a usar y su ubicación dentro de la ventana, así como el nombre y la posición inicial del robot a usar, en este caso del modelo Pioneer P3-AT, el cual hereda sus características del archivo pioneer.cfg presente en la librería de Stage.

Algoritmo 3.2 Configuración del archivo world.

```
# Configuración de la interfaz de usuario
window (
    size [ 635.000 666.000 ] # en pixels
    scale 37.481 # pixels por metro
    center [ -0.019 -0.282 ]
    rotate [ 0 0 ]
    show_data 1 # 1=on
)

# Seleccionando el entorno
size [16.000 16.000 0.800]
pose [0 0 0 0]
bitmap "/usr/src/Stage-3.2.2-Source/worlds/bitmaps/
        autolab.png" # imagen
)
pioneer3at (
# nombre y posición inicial
    name "r0"
    pose [ -5 -5 0 45 ]
    ctrl "programa2"
# reportes libres de error en posición global
    localization "gps"
    localization_origin [ 0 0 0 0 ]
)

```

Seguidamente se escribe el archivo de configuración .cfg (Algoritmo 3.3), en donde se especifican todos los drivers para la simulación, el driver de simulación es importante porque permite modelar la comunicación con el robot por software, se especifica también el archivo del entorno .world a usar. En el último párrafo se especifican los drivers de las interfaces a los componentes deseados, se incluye en esta parte la posición y los sonares.

Algoritmo 3.3 Configuración del archivo world.

```
# cargando el driver del modo de simulación
driver
(
    name "stage"
    provides [ "simulation:0" ]
    plugin "stageplugin"
    # cargando el archivo world correspondiente
    worldfile "prueba2.world"
)
# Creando los drivers adicionales (posición, sonares y gráficos)
# para el robot modelo "r0"
driver
(
    name "stage"
    provides [ "position2d:0" "sonar:0" "graphics2d:0"
              "graphics3d:0" ]
    model "r0"
)
```

La simulación se realizó fijando como punto de partida el ya establecido en el archivo .world (posición $x = -5$, $y = -5$) y especificando como meta en el programa principal del controlador en c++ el punto con coordenadas $x = 5$ e $y = 5$.

La secuencia obtenida en la simulación para este caso se muestra en las figuras 3.31, 3.32, 3.33 y 3.34 en las cuales se aprecia el recorrido seguido por el robot para alcanzar la meta fijada.

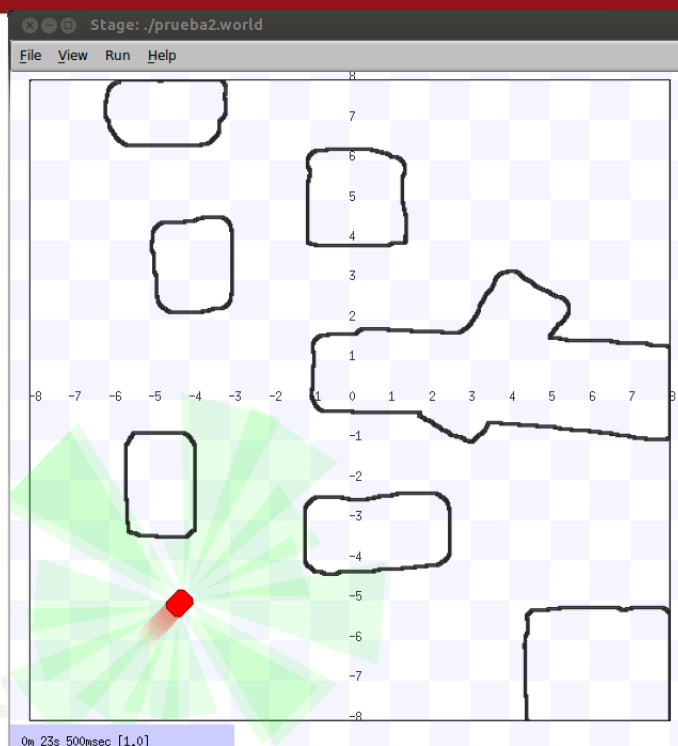


Figura 3.31 – Posición inicial del robot móvil.

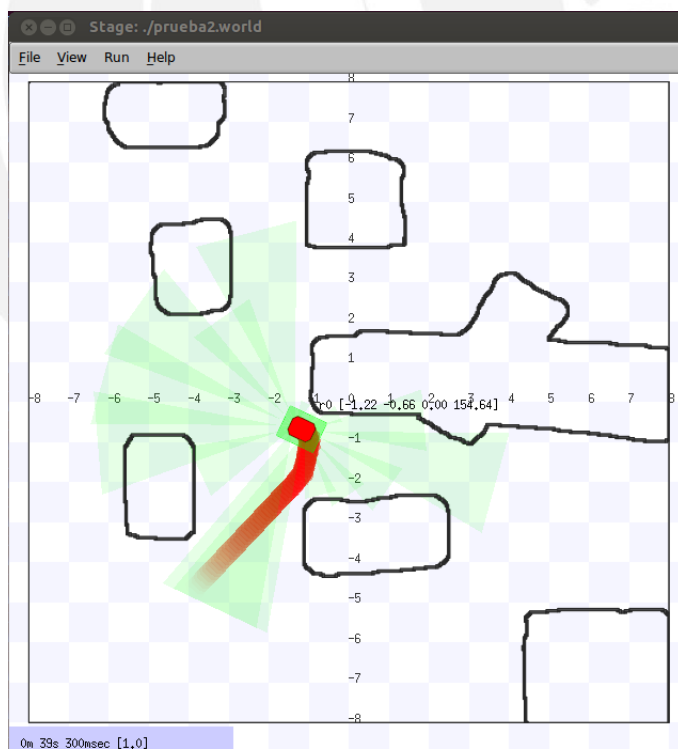


Figura 3.32 – Captura 2 del movimiento del robot en su trayectoria hacia la meta.

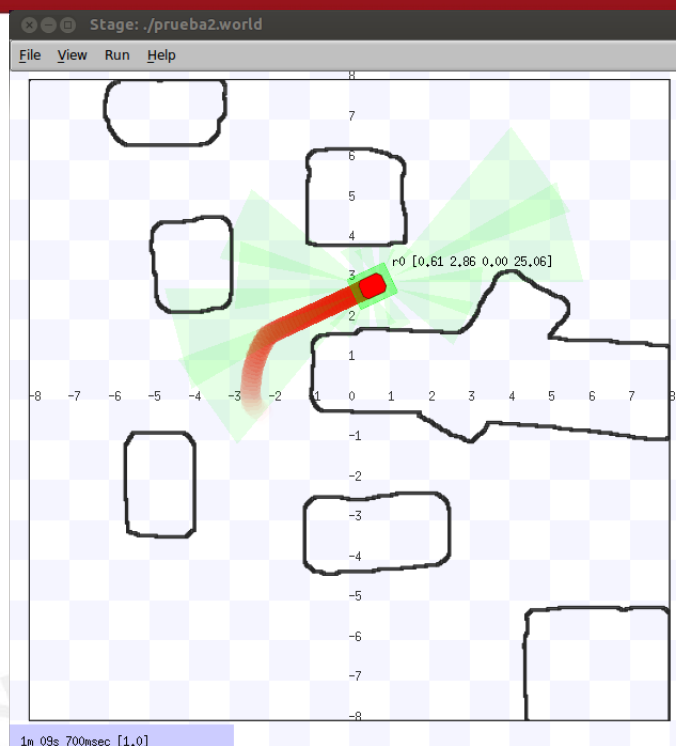


Figura 3.33 – Captura 3 del movimiento del robot en su trayectoria hacia la meta.

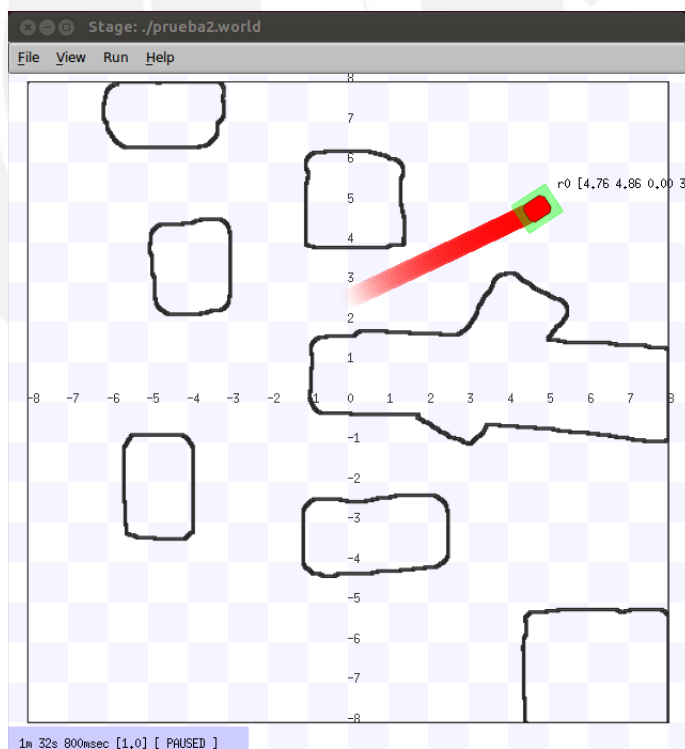


Figura 3.34 – Llegada a la meta del robot.

3.6. Comparación con otros Controladores

En esta sección se evalúa el desempeño del algoritmo propuesto, al cual se le denominará en adelante BBFH (Controlador basado en conductas e histogramas de fuerza), en comparación con otros métodos del mismo tipo empleados comúnmente en la navegación de robots móviles.

3.6.1. Comparación con un Controlador por Conductas

Para evaluar el desempeño del algoritmo comparado a un algoritmo clásico basado en conductas, se elaboró un nuevo controlador con las mismas conductas y arbitro usados antes pero modificando la manera en que se calculan las magnitudes a usar.

La diferencia radica en que las decisiones se toman sólo en base a las mediciones obtenidas de los sonares, en la práctica estos pueden ser reemplazados con sensores láser, sin embargo, estos tienen un precio muy superior. Se denominará a este controlador en lo sucesivo como controlador reactivo basado en conductas (CRBC).

La figura 3.35 muestra el recorrido usando CRBC, el recorrido obtenido es aceptable, no obstante podría ser mejorado usando algunas de las modificaciones presentadas en [6], [7], [9] o [98] que implican elementos de control adicionales y en general entrenamiento adicional. Además se presenta en la figura 3.36 el ángulo del timón generado en cada iteración y el factor de velocidad (donde 1 representa el 100%); y en la figura 3.37 el tiempo de cómputo por cada iteración.

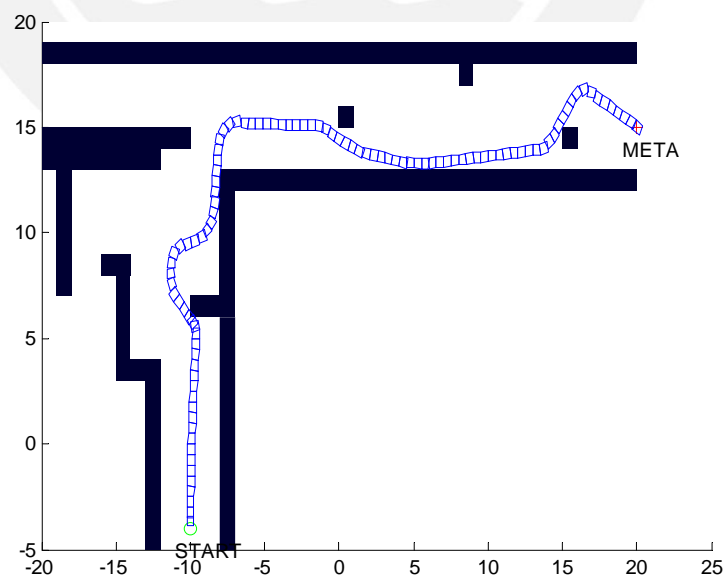


Figura 3.35 – Recorrido usando CRBC.

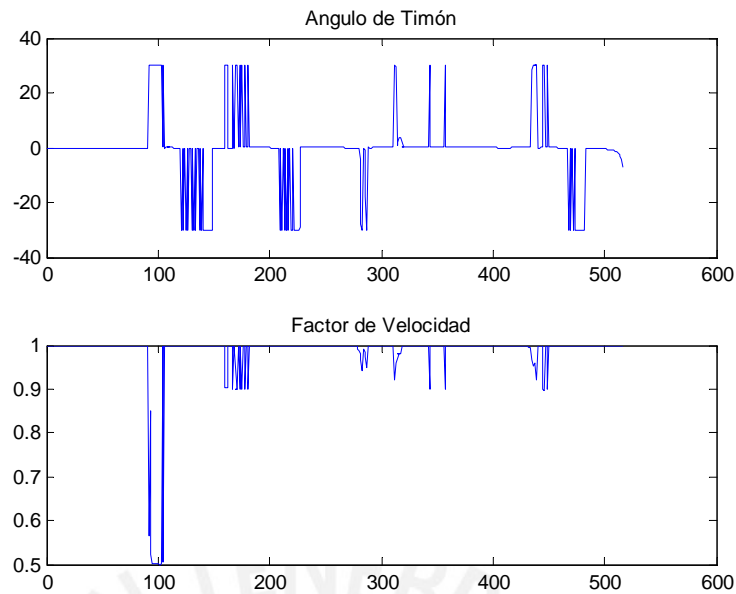


Figura 3.36 – Angulo de timón y factor de velocidad generados usando CRBC.

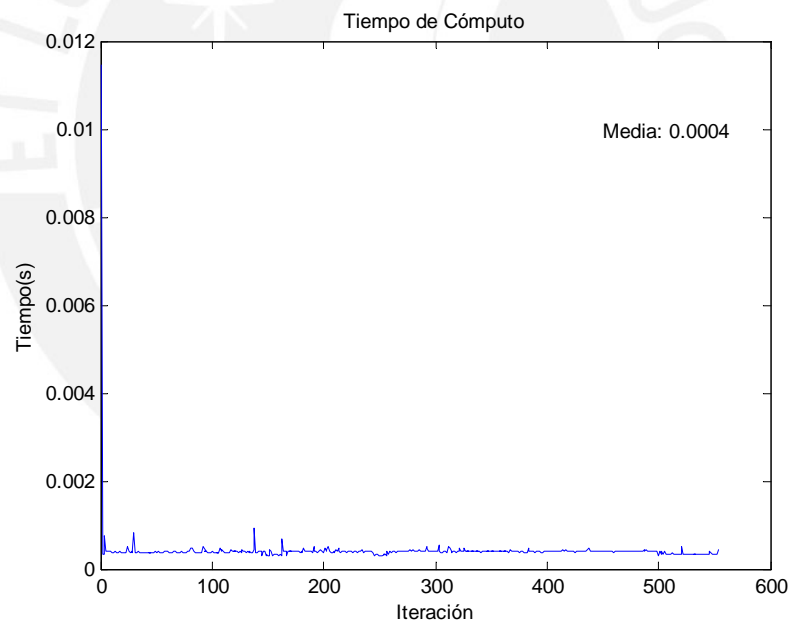


Figura 3.37 – Tiempo de cómputo por iteración usando CRBC.

Por otro lado la respuesta para BBFH se muestra en la figura 3.38, el ángulo del timón y el factor de velocidad se muestran en la figura 3.39 y el tiempo de cómputo por iteración en la figura 3.40.

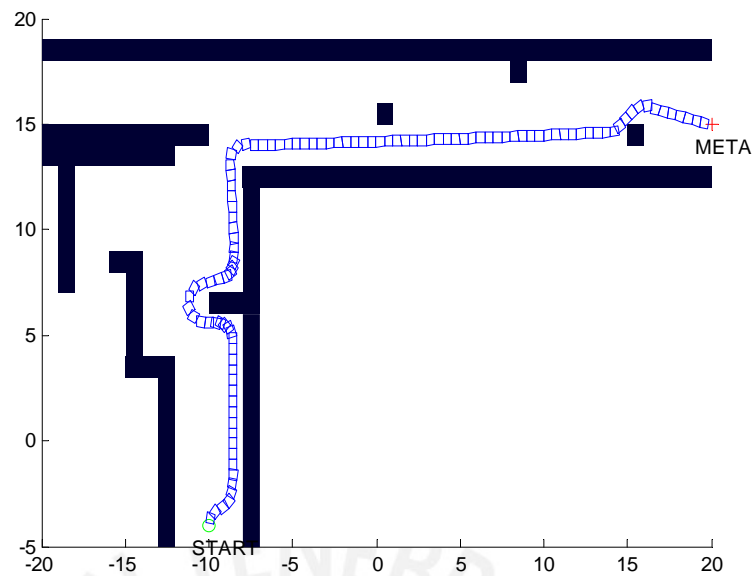


Figura 3.38 – Recorrido usando BBFH.

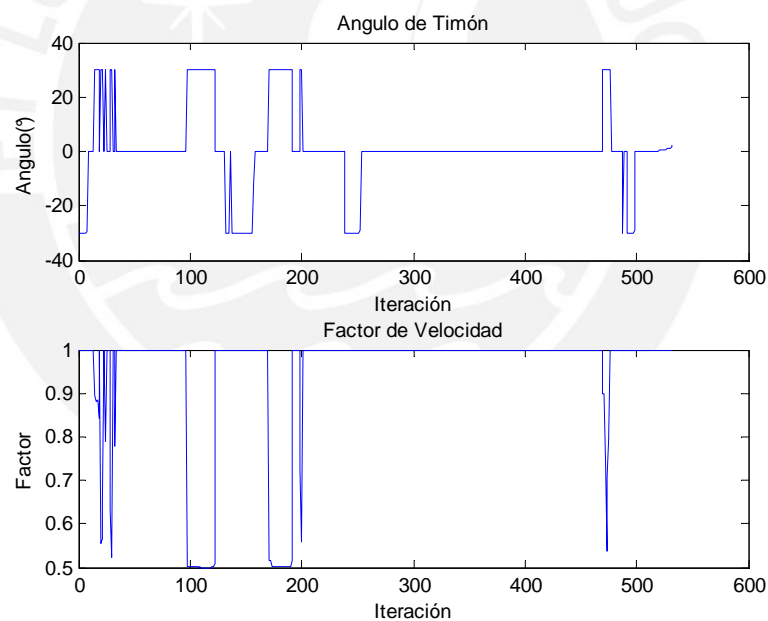


Figura 3.39 – Angulo de timón y factor de velocidad generados cuando se usa BBFH.

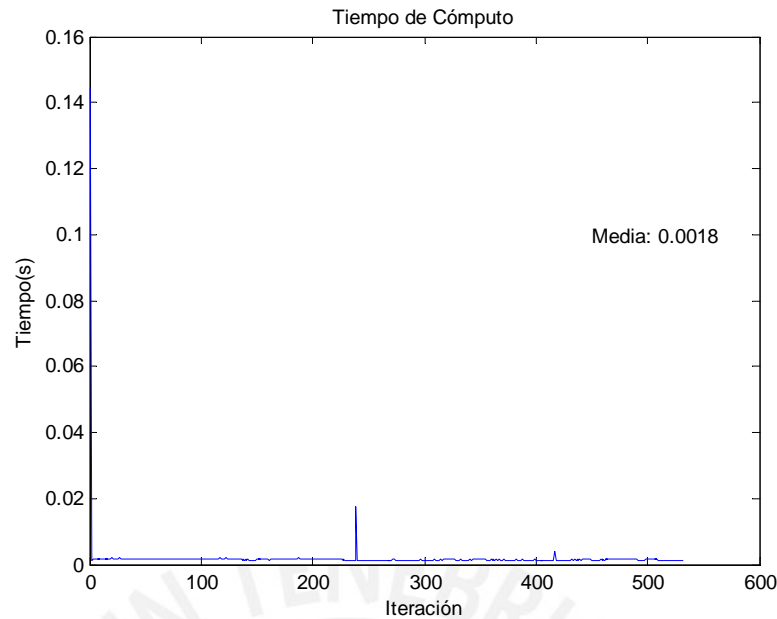


Figura 3.40 – Tiempo de respuesta usando BBFH.

Se puede apreciar que el recorrido mejora, lo cual sucede por dos motivos: 1. La sintonización en un controlador CRBC se hace complicada si los sonares no están posicionados en una geometría correcta para el algoritmo y 2. El cambio en las distancias sensadas puede afectar significativamente el comportamiento del controlador, ninguno de estos problemas se presentan en el controlador diseñado en este trabajo.

Se puede además apreciar de las gráficas 3.36 y 3.38 que el ángulo en el timón puede presentar una mayor variación en un controlador CRBC debido en gran parte a su mayor sensibilidad a las variaciones en las distancias medidas.

Por otro lado, se puede observar de las figura 3.37 y 3.40 que el tiempo de cómputo requerido para CRBC es menor debido al menor número de operaciones necesarias para su operación.

Por último se probó el controlador CRBC con sonares más ruido ruido gaussiano en la medición con media de $\mu = 1$ y desviación estándar $\sigma = 0,5$, la respuesta para BBFH se puede apreciar en las figuras 3.21 y 3.22, y las obtenidas para el controlador CRBC en la figura 3.41 donde aumentan los choques del móvil.

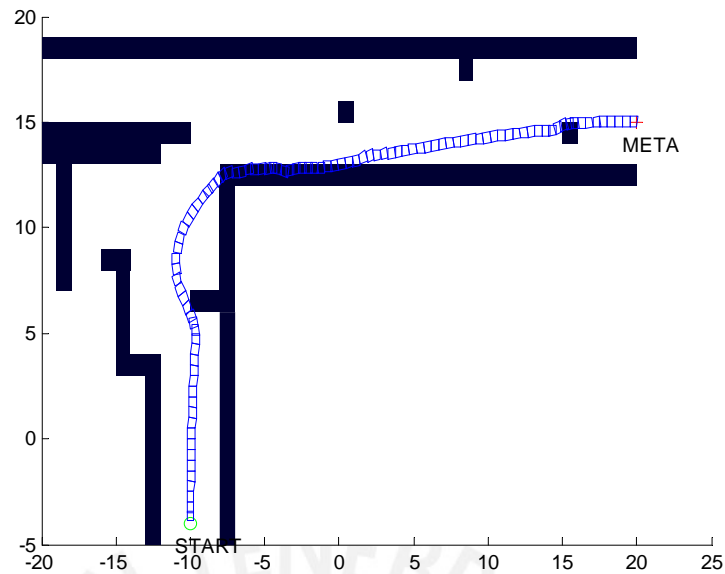


Figura 3.41 – Recorrido de CRBC cuando los sonares presentan ruido.

3.6.2. Comparación con Algoritmos VFF y VFH

Una de las principales desventajas en el método VFF es su mal desempeño al maniobrar por caminos estrechos[17], el controlador BBFH reduce en gran medida estos problemas como puede apreciarse en la figura 3.42, donde se estrecharon los espacios libres.

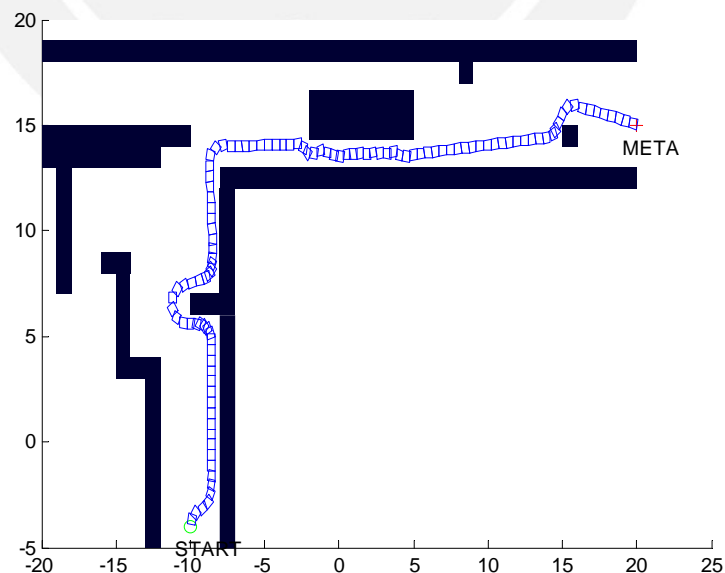


Figura 3.42 – Recorrido de BBFH cuando el recorrido tiene espacios estrechos.

Además es conocida la necesidad de incluir un filtro para suavizar la trayectoria en el

seguimiento de paredes como se observa en la figura 3.43a, mientras que en el caso de BBFH, figura 3.43b este no es necesario.

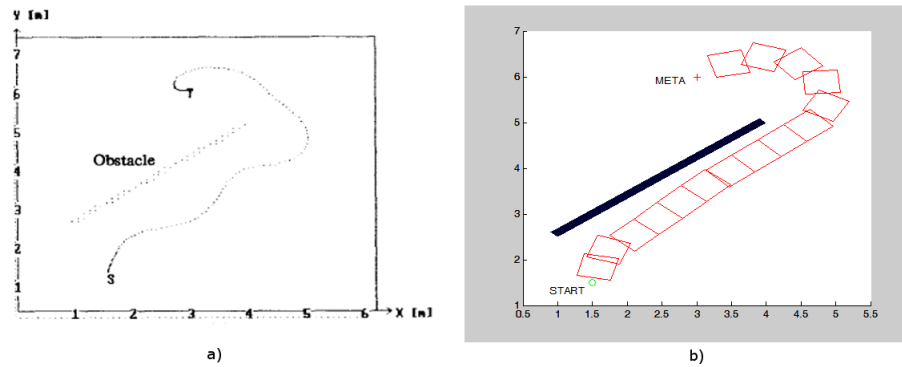


Figura 3.43 – Trayectorias de seguimiento. a) usando VFF[17] sin filtro, b) usando BBFH.

Por otro lado, para evaluar el desempeño de BBFH frente a VFH, se implementó este último con los siguientes parámetros: $\alpha = 5^\circ$, $l = 5$, $umbral = 49$, $s_{max} = 16$ y $h_m = 49$ definidos de acuerdo a[16].

La trayectoria obtenida usando VFH se muestra en la figura 3.44, el ángulo del timón y la velocidad generados en la trayectoria se muestran en la figura 3.45 y el tiempo de cómputo en la figura 3.46.

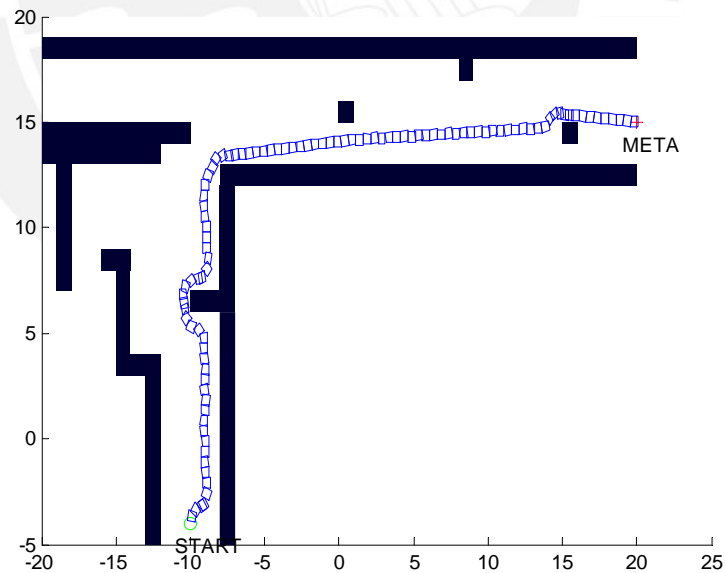


Figura 3.44 – Recorrido cuando se usa VFH.

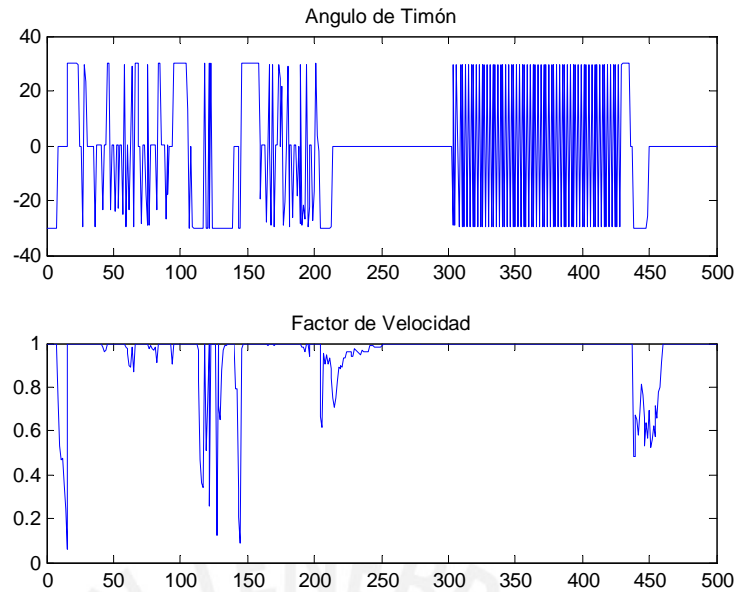


Figura 3.45 – Angulo de timón y factor de velocidad generados cuando se usa VFH

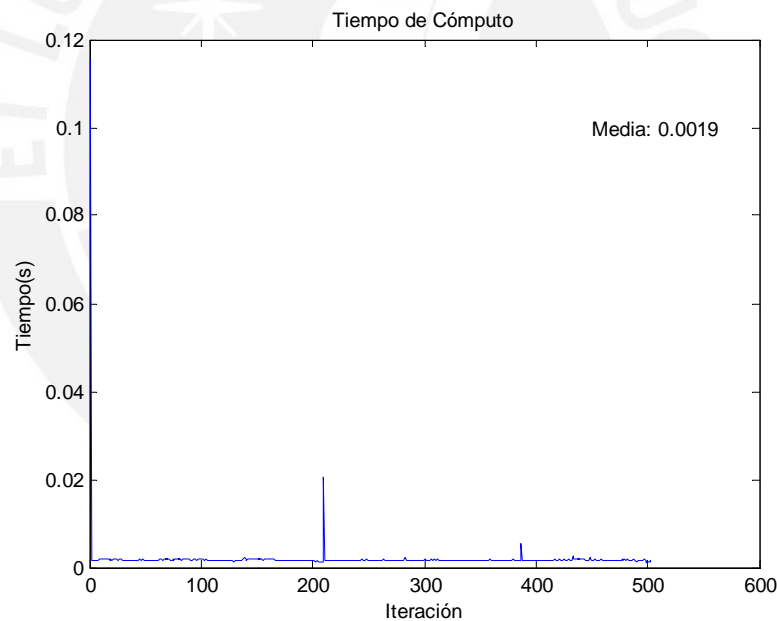


Figura 3.46 – Tiempo de cómputo por iteración cuando se usa VFH.

Los resultados con BBFH para este mismo recorrido se pueden apreciar en las figuras 3.38, 3.39 y 3.40, se tiene en este caso una trayectoria aceptable con ambos algoritmos, por otra parte el tiempo de cómputo en cada iteración es aproximadamente 3 veces superior que en un CRBC para ambos casos, no obstante los recorridos generados son superiores.

Se puede apreciar además que el recorrido de VFH se acerca mucho a los obstáculos, haciendo de su regulación requiera de más trabajo, una clara muestra de esto es que al disminuir el tamaño de la cuadrícula para VFH este puede generar choques más

frecuentes, en la figura 3.47 se muestra a la izquierda el recorrido obtenido al disminuir el tamaño de la cuadrícula a 25x25, y a la izquierda este mismo recorrido usando un tamaño de ventana de 33x33, en la figura 3.48 se muestra los recorridos obtenidos con BBFH para tamaños de cuadrícula de 25x25(izquierda) y 33x33(derecha), los cuales son muy similares.

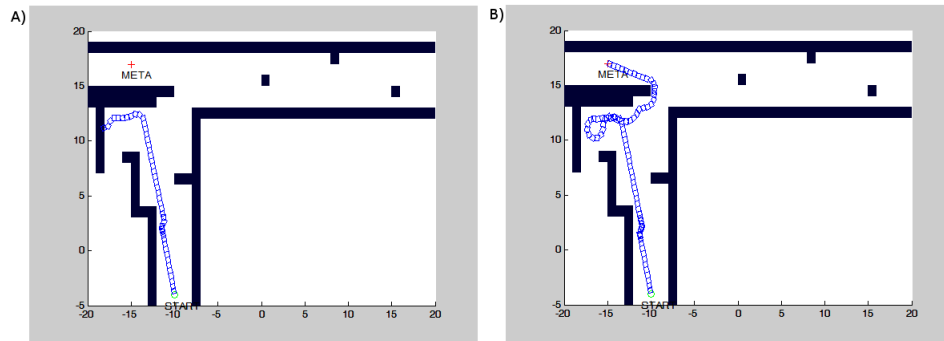


Figura 3.47 – Trayectorias para VFH con dos tamaños de cuadrícula. A. Usando un tamaño de cuadrícula de 25x25; B. Usando un tamaño de cuadrícula de 33x33.

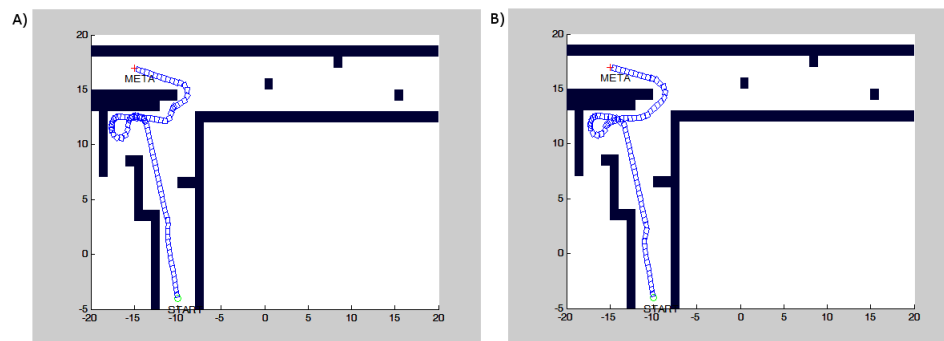


Figura 3.48 – Trayectorias para BBFH con dos tamaños de cuadrícula. A. Usando un tamaño de cuadrícula de 25x25; B. Usando un tamaño de cuadrícula de 33x33.

Es de apreciar además, en estos casos, que el algoritmo VFH genera choques cuando la elección de la orientación deseada contraviene el recorrido permitido por la geometría del robot móvil, un caso adicional se muestra en la figura 3.49 donde se presenta los recorridos generados ante un entorno en forma de laberinto, VFH puede generar choques si la geometría del entorno presenta algunas particularidades.

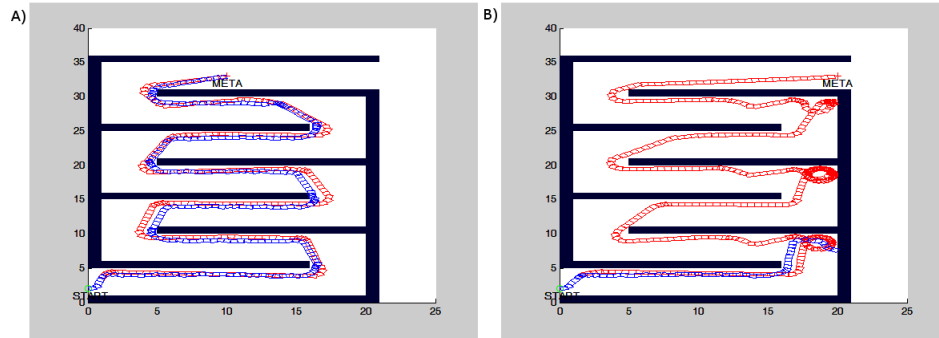


Figura 3.49 – Trayectorias usando VFH(azul) y BBFH(rojo). A. Cuando el punto de llegada está en la zona media, B. Cuando el punto de llegada esta en una esquina.

En suma, VFH requiere adicionalmente una integración con un algoritmo para la evasión de obstáculos o la consideración de código que permita considerar la geometría del robot (algoritmo VFH+), todo lo cual no es necesario en BBFH.

Otro factor a apreciar es que la acción de control en el ángulo de timón que genera VFH contiene una mayor cantidad de variación que BBFH, esto a su vez incrementa la potencia consumida; si se considera que la ley de control es proporcional al voltaje la potencia consumida acumulada en ambos recorridos se muestra en la figura 3.50, se puede apreciar que la potencia necesaria es menor cuando se usa BBFH.

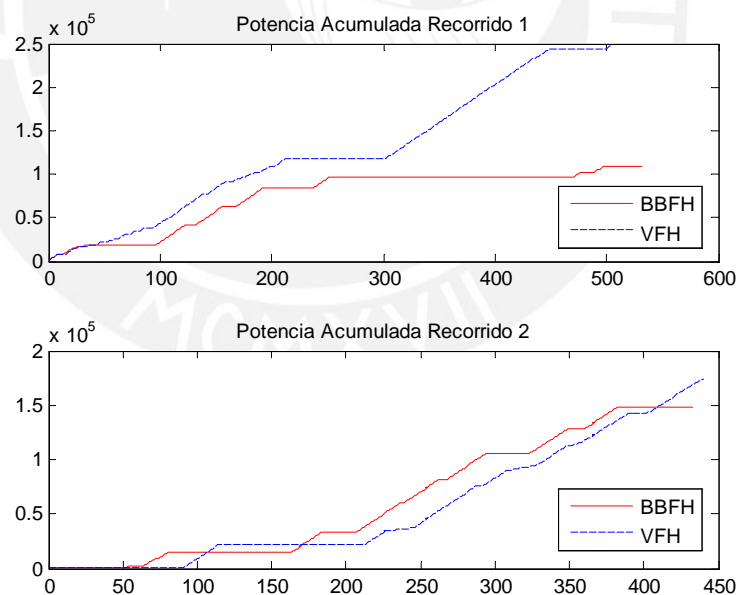


Figura 3.50 – Potencias consumidas durante el recorrido con ambos algoritmos.

3.7. Conclusiones

- Se implementó el sistema de cuadrículas de certeza simplificado para almacenar la información recibida por los sensores y usarla en posteriores operaciones.

- Se diseñaron e implementaron las conductas principales para la navegación mediante redes neuro difusas, las cuales son, la conducta goal seeking, la conducta obstacle avoidance y las conductas wall following.
- Se desarrolló un controlador neuro difuso para realizar la fusión de las conductas antes mencionadas, lo cual permite la navegación del robot de forma continua en ambientes desconocidos.
- Se mostró que el controlador fusiona elementos del control por conductas con la navegación mediante construcción de mapas, lo cual permite entre otras cosas, disminuir el efecto de la incertidumbre del sensado, mejorar la respuesta ante situaciones críticas y reducir la incertidumbre en la señal de control.
- Se demostró la superioridad de BBFH frente a otros métodos de navegación tradicionales, ya que puede trabajar con sensores con incertidumbre, mejora el manejo al presentar menos variabilidad en su ley de control y puede tomar mejores decisiones ante obstáculos complicados.
- Se demostró la factibilidad del controlador diseñado en aplicaciones reales mediante la implementación en simuladores de uso extendido como Matlab y Player/Stage.

Capítulo 4

Diseño del Sistema de Fusión de Sensores

4.1. Introducción

La fusión de sensores es un elemento importante en la robótica moderna, debido a la necesidad de usar una gran variedad de sensores en la navegación y a la naturaleza inherentemente incierta de la medición. La fusión permite en ese sentido la reducción de la incertidumbre y la integración de diferentes fuentes de información de manera coherente.

Los avances en los sensores han implicado en general una mejora en la precisión de los mismos, no obstante la incertidumbre en la medida es casi imposible de eliminar debido a diversos factores, como por ejemplo la acumulación de errores en los encoders o el efecto de deriva debido a las operaciones de integración en los acelerómetros.

En el siguiente capítulo se mostrará el diseño e implementación de un sistema de localización con fusión de sensores para poder obtener los datos de posición y orientación del robot, el algoritmo está basado en la medición odométrica, los sensores inerciales y en datos obtenidos mediante sensores de redes inalámbricas (WSN).

4.2. Calibración de los Sensores

En esta sección se hace una comparación entre los distintos sensores existentes en un dispositivo inteligente, para realizar este estudio se utilizó un smartphone Motorola Moto G, el cual se eligió por poseer una gran cantidad de sensores, entre los cuales se pueden mencionar los siguientes: acelerómetro, giroscopio, magnetómetro, receptor wifi y gps; estos son tratados en detalle en los siguientes puntos.

4.2.1. Sensores Inerciales (IMU)

Aunque en un inicio estos sensores eran considerados caros, la reducción en su tamaño y costo, gracias a su implementación por medio de MEMS (Microelectrome-

chancal Systems); ha mejorado su uso como una alternativa en aplicaciones de localización y navegación.

El uso de los sensores inerciales en sistemas embebidos ha alcanzado una gran difusión en la actualidad gracias a su disposición en la gran mayoría de equipos inteligentes modernos (smartphones y tablets).

4.2.1.1. Acelerómetro

El acelerómetro es un instrumento de la medición de la aceleración, por lo general están constituidos para la medición en 3 ejes. Un esquema de su funcionamiento se aprecia en la figura 4.1, en reposo la medición es de 1 g (unidad de gravedad= $9.80665m/s^2$) y 0 g cuando el móvil está en caída libre.

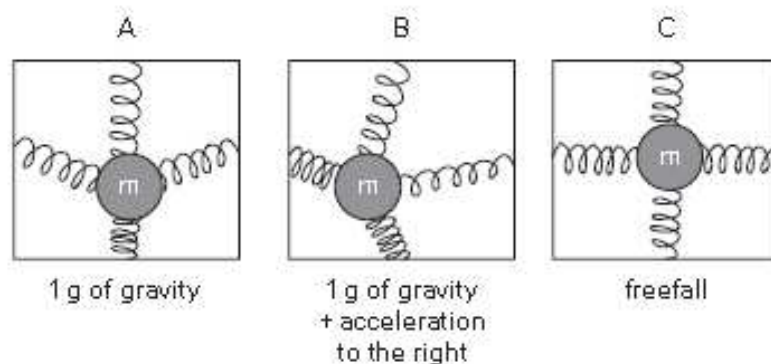
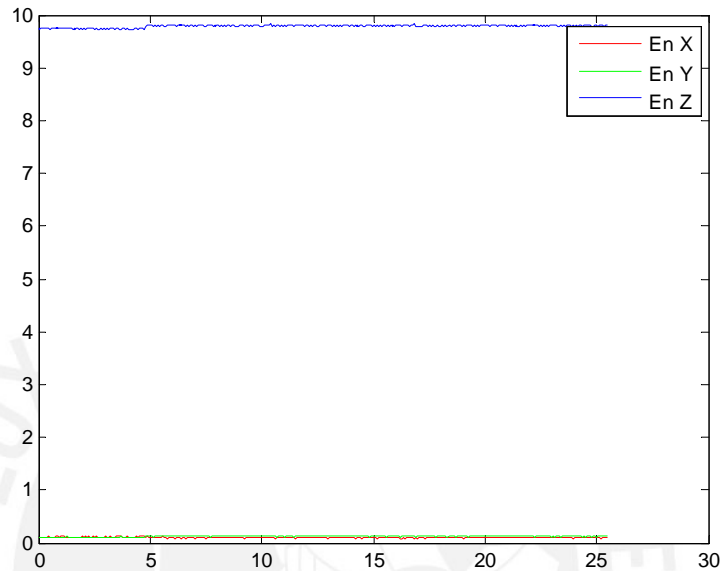


Figura 4.1 – Representación del acelerómetro.

En la figura 4.2 se muestra un ensayo con el acelerómetro en reposo, nótese que la magnitud más influyente es en la dirección Z y es de aproximadamente 1g, y una magnitud de aproximadamente cero en los ejes X e Y.

Tabla 4.1 – Estadísticas de la respuesta del acelerómetro

Eje	Promedio	Max	Min	Varianza
a_X	0.1009	0.1314	0.0802	0.0092
a_Y	0.1230	0.1451	0.0925	0.0098
a_Z	9.7882	9.83	9.7191	0.0258
Total $a(m/s^2)$	9.7895	9.8314	9.7204	0.0258

**Figura 4.2** – Ensayo del acelerómetro en reposo.

Un acelerómetro puede ser caracterizado mediante la siguiente ecuación[121]:

$$\ddot{z}_a = \ddot{z} + g + \varepsilon_a + S_a g + v \quad (4.1)$$

donde

\ddot{z}_a representa la medición del acelerómetro.

\ddot{z} es la aceleración real.

g es la magnitud de la gravedad.

ε_a es el término de bias.

S_a es un factor de escalamiento

v es el ruido en la medición.

Para calcular el factor de escalamiento se utilizó la siguiente ecuación[121]:

$$S_a = \frac{\ddot{z}_{down} - \ddot{z}_{up} - 2g}{2g} \quad (4.2)$$

Ensayando con el acelerómetro se obtuvo que $\ddot{z}_{down} = 9,7882$ y $\ddot{z}_{up} = -9.864$ con lo cual $S_a = 0,002$. Mientras que para el bias se usan los valores en la tabla 4.1.

4.2.1.2. Giroscopio

El giroscopio mide la tasa de cambio en el ángulo del equipo para los 3 ejes de medición de Euler: yaw, roll y pitch (figura 4.3). Los giroscopios poseen además una buena respuesta en espacios cortos de tiempo por lo cual son usados para la estimación a corto plazo.

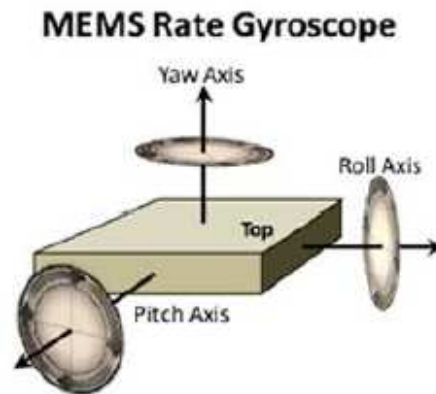


Figura 4.3 – Esquema de coordenadas en un giroscopio.

En la figura 4.4 se aprecia un ensayo con el giroscopio del smartphone en estado de reposo, nótese que aún en este estado existen pequeñas fluctuaciones rápidas las cuales pueden generar un efecto de deriva, las estadísticas de este ensayo se presentan en el tabla 4.2.

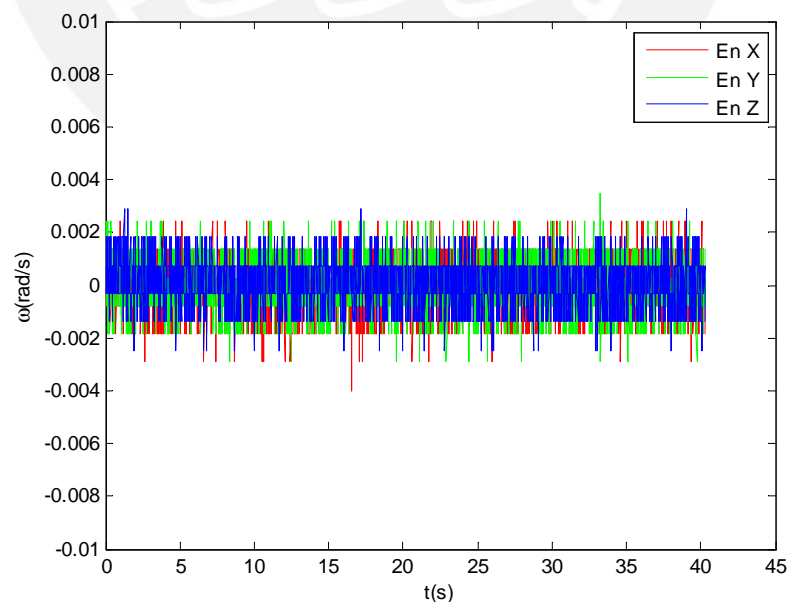


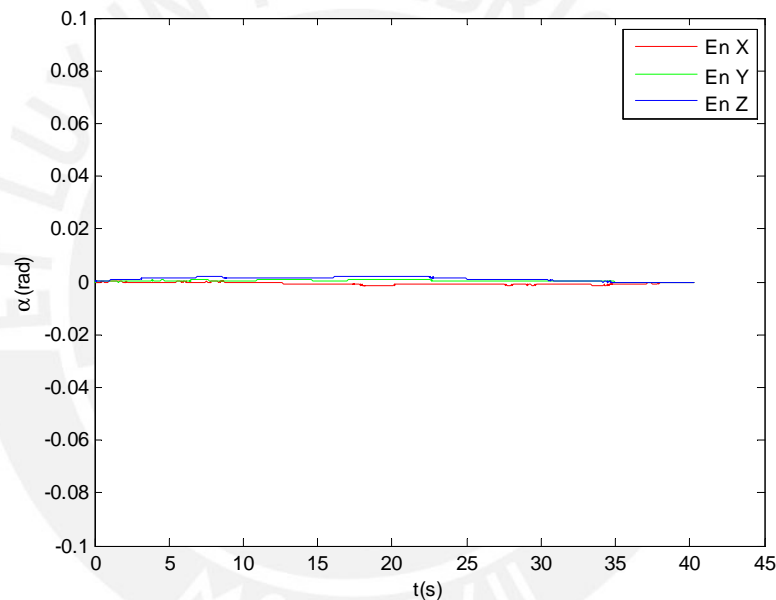
Figura 4.4 – Medidas del giroscopio con el equipo en reposo

Tabla 4.2 – Datos estadísticos de las medidas.

Eje	Promedio	Max	Min	Varianza
g_X	$-6,5243 \times 10^{-5}$	0.0024	-0.004	$9,2076 \times 10^{-4}$
g_Y	$7,1205 \times 10^{-5}$	0.0035	-0.0029	$9,0116 \times 10^{-4}$
g_Z	$5,3434 \times 10^{-5}$	0.0029	-0.0025	$8,2469 \times 10^{-4}$

El giroscopio en Android posee una corrección del bias y por ende el efecto de deriva producida por la integración, expresión 4.3, se minimiza como se observa en la figura 4.5.

$$\alpha = \sum_{i=0}^N \omega dt \quad (4.3)$$

**Figura 4.5** – Integración de las medidas en giroscopio.

4.2.2. Magnetómetro

El magnetómetro permite medir la intensidad del campo magnético en un espacio, su finalidad es usar como referencia el polo norte magnético de la tierra, no obstante las mediciones pueden verse afectadas por otros campos artificiales creados por materiales ferromagnéticos en las cercanías del sensor, por lo que es necesario una calibración del instrumento.

En la figura 4.6 se aprecia un ensayo en reposo del magnetómetro, para ello se posiciona el equipo con el eje X orientado hacia el norte y con la pantalla hacia arriba, en el cuadro 4.3 se muestra además las características de las señales.

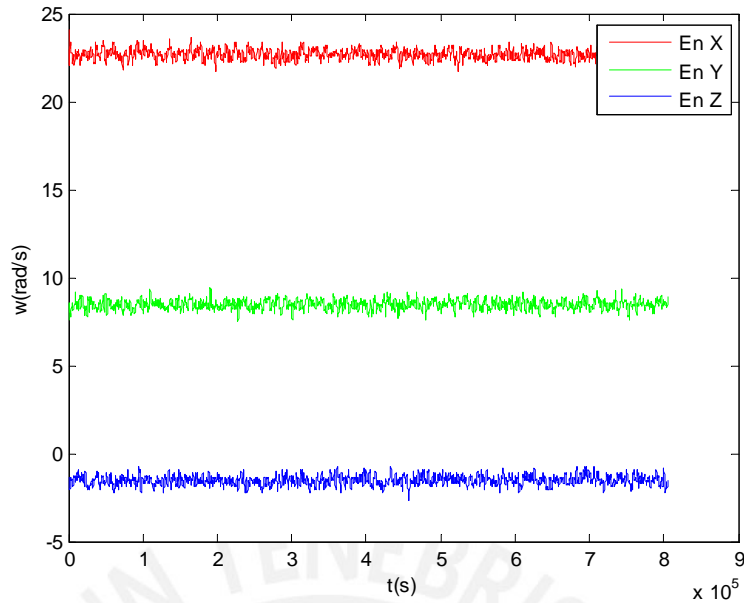


Figura 4.6 – Ensayo en reposo del magnetómetro.

Tabla 4.3 – Datos estadísticos del ensayo en el magnetómetro

Eje	Promedio	Max	Min	Varianza
m_X	22.65	24.12	21.72	0.28
m_Y	8.4543	9.4193	7.45	0.2632
m_Z	-1.5255	-0.7187	-1.5255	0.26

Como se mencionó antes, la principal desventaja del magnetómetro es su sensibilidad a un gran número de interferencias magnéticas en su entorno, los cuales pueden sumarse en las denominadas interferencias de hierro fuerte y suave (hard-iron y soft-iron interference [19, 122]), .

Con el objetivo de estimar la intensidad de ese efecto, se realiza un ensayo para conseguir un rango amplio de mediciones, el cual se muestra en la figura 4.8, se seleccionaron luego algunos puntos del mismo para la calibración del magnetómetro por el método de la optimización de errores mediante cuatro parámetros[122].

En el método usado se parte de la expresión para llegar a la expresión 4.4 de errores residuales en 4.5, se definen además las matrices X , Y y β como se muestra en las ecuaciones 4.6, 4.7 y 4.8.

$$(B_p - V)^T (B_p - V) = B^2 \quad (4.4)$$

$$r = B_x^2 + B_y^2 + B_z^2 + B_x V_x + B_y V_y + B_z V_z + V_x^2 + V_y^2 + V_z^2 - B^2 \quad (4.5)$$

$$X = \begin{bmatrix} B_x[0] & B_y[0] & B_z[0] & 1 \\ B_x[1] & B_y[1] & B_z[1] & 1 \\ \vdots & \vdots & \vdots & \vdots \\ B_x[N] & B_y[N] & B_z[N] & 1 \end{bmatrix} \quad (4.6)$$

$$Y = \begin{bmatrix} B_{x[0]}^2 + B_{y[0]}^2 + B_{z[0]}^2 \\ B_{x[1]}^2 + B_{y[1]}^2 + B_{z[1]}^2 \\ \vdots \\ B_{x[N]}^2 + B_{y[N]}^2 + B_{z[N]}^2 \end{bmatrix} \quad (4.7)$$

$$\beta = \begin{bmatrix} 2V_x \\ 2V_y \\ 2V_z \\ B^2 - V_x^2 - V_y^2 - V_z^2 \end{bmatrix} \quad (4.8)$$

La optimización se obtiene al minimizar la ecuación 4.9 mediante los cuatro parámetros independientes; V_x , V_y , V_z y B , de donde se obtiene la expresión 4.10.

$$\beta = (X^T X)^{-1} X^T Y \quad (4.9)$$

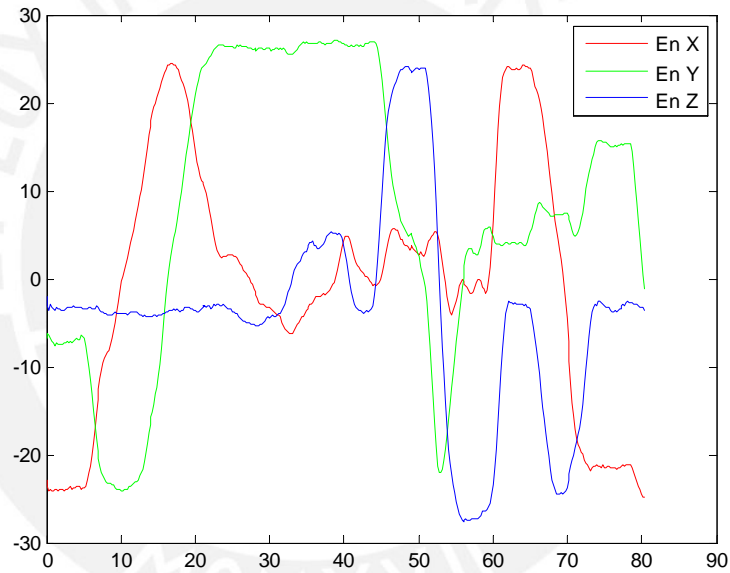


Figura 4.7 – Ensayo del magnetómetro con varias rotaciones.

La calibración se realizó con 11 valores elegidos en forma distribuida de cada 40 muestras en la figura 4.7, tras lo cual se obtuvieron los siguientes valores para los parámetros:

$$V_x = 0,0981 \quad (4.10)$$

$$V_y = -1,5143 \quad (4.11)$$

$$V_z = 1,1018 \quad (4.12)$$

$$B = 25,1797 \quad (4.13)$$

Se puede constatar además el valor obtenido para el campo geomagnético con el

provisto por el Centro de Data Mundial para el Geomagnetismo (<http://wdc.kugi.kyoto-u.ac.jp/igrf/point/index.html>) para la locación de Lima.

Una calibración para tomar en cuenta las interferencias de hierro suave requiere de ajustes en línea de las medidas, algunos métodos para realizar este procedimiento pueden verse en [123, 124].

4.2.3. Aceleración Lineal

Para el cálculo de la aceleración lineal Android excluye el efecto gravitatorio y provee un vector con la magnitud de la aceleración en cada eje. El proceso respeta la relación mostrada en 4.14(<http://developer.android.com/develop/index.html>).

$$A_{total} = A_{gravedad} + A_{lineal}$$

(4.14)

La figura 4.8 muestra el ensayo de la aceleración lineal durante un tiempo de 1200 segundos, cuando el equipo se mantiene en reposo y en la tabla 4.4 los valores estadísticos de este ensayo.

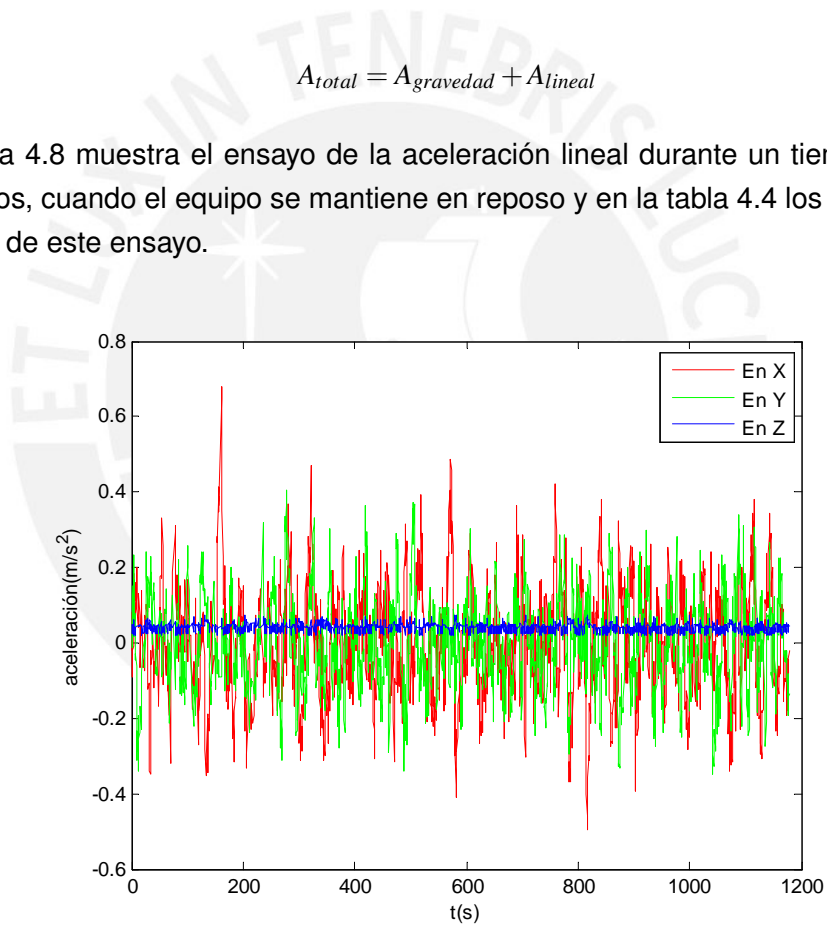


Figura 4.8 – Medidas de la aceleración lineal.

Tabla 4.4 – Valores estadísticos del ensayo con la aceleración lineal

Eje	Promedio	Max	Min	Varianza
I_X	0.0016	0.6806	-0.494	0.1394
I_Y	0.00085	0.4055	-0.3503	0.1178
I_Z	0.0431	0.075	0.0192	0.0085

Extrayendo el offset de las medidas e integrando se obtuvo la gráfica 4.9, donde se muestra que el error en la estimación de la posición puede llegar a ser de un máximo de 4 metros, demostrando que la doble integración es muy desfavorable en este caso.

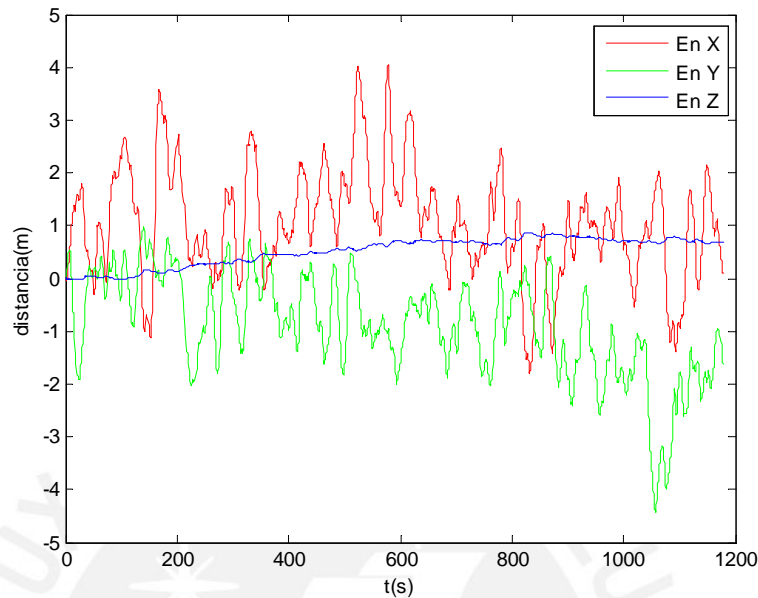


Figura 4.9 – Integración de las medidas en la aceleración.

4.2.4. Sensores de Red Inalámbrica (WSN)

La localización basada en WSN es un área clave en los trabajos de investigación actuales, la aplicación de las mismas a través del aprovechamiento de las capacidades de comunicación inalámbrica permite afrontar la estimación de la distancia basándose principalmente en uno de estos dos factores: la medición de la potencia en la señal recibida (RSSI) y el tiempo de vuelo (TOF).

El siguiente trabajo usa el primer método, para lo cual se mide el RSSI de cuatro puntos de acceso con ubicación conocida, la relación teórica entre la distancia y la caída en potencia de las señales inalámbricas se expresa en la ecuación 4.15[125][126], donde P_r representa la intensidad de la señal recibida, P_o la intensidad de la señal recibida en un punto de referencia fijo d_o , γ es el exponente de pérdida por el camino que suele tener un rango de 1.6-3.5 en interiores[127], d la distancia entre receptor y transmisor y v^r el ruido en la medición.

$$P_r(d) = P_o - 10\gamma \log_{10}(d/d_o) + v^r_{(k)} \quad (4.15)$$

4.3. Fundamento de los Filtros de Kalman

Un filtro de Kalman es constituido por un conjunto de ecuaciones que provee de un eficiente mecanismo computacional para la estimación de las variables de estado de un proceso, de forma tal que minimiza la media del error cuadrático[18]. Los filtros de Kalman han sido un elemento fundamental en el desarrollo de los sistemas autónomos desde su inclusión por Kalman en 1960 [128].

El objetivo del filtro de Kalman es resolver el problema de la estimación el vector de estados $x \in \mathfrak{R}^n$ de un proceso de control en tiempo discreto el cual es gobernado por la siguiente ecuación estocástica lineal:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \quad (4.16)$$

Y con la ecuación de medición:

$$z_k = Hx_k + v_k \quad (4.17)$$

Donde w_k y v_k representan el ruido del proceso y medición respectivamente, y son considerados como ruido blanco gaussiano, no correlacionados y con las siguientes distribuciones de probabilidad:

$$p(w) \sim N(0, Q) \quad (4.18)$$

$$p(v) \sim N(0, R) \quad (4.19)$$

Donde Q es la covarianza en el error del proceso y R es la covarianza en el error de la medición.

4.3.1. Filtro de Kalman Discreto

Definiendo además la covarianza a priori y a posteriori del error en 4.20 y 4.21 respectivamente:

$$P_k^- = E[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T] \quad (4.20)$$

$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \quad (4.21)$$

La estructura del algoritmo de Kalman se puede resumir en dos etapas, como se muestra en la figura 4.10, en la primera etapa denominada de Predicción se realiza el cálculo para la obtención del estado estimado a priori \hat{x}_k^- , y seguidamente el cálculo de la covarianza del error estimado a posteriori P_k^- .

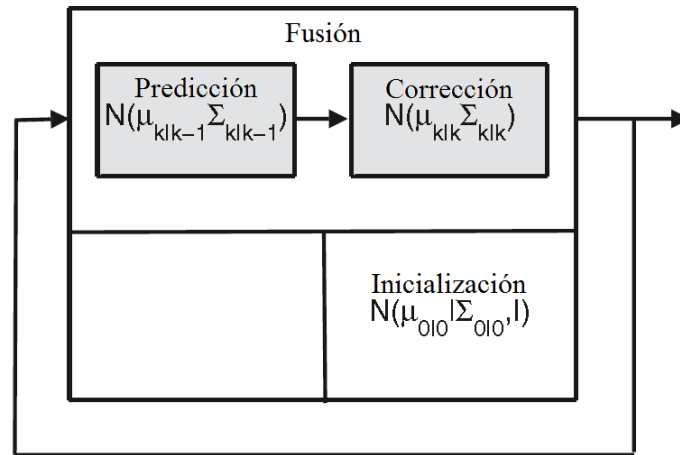


Figura 4.10 – Esquema de un filtro de Kalman discreto[4]

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \quad (4.22)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (4.23)$$

En la segunda etapa denominada de Corrección se realiza el cálculo de la ganancia K mediante la ecuación 4.24, luego para obtener del vector de estados estimados a posteriori \hat{x}_k se corrige la estimada a priori \hat{x}_k^- con el resultado del producto de la ganancia y la diferencia entre la medición y la medición estimada $H\hat{x}_k^-$, diferencia que también es llamada innovación. Por último la varianza del error es actualizada mediante la ecuación 4.26.

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (4.24)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (4.25)$$

$$P_k = (1 - K_k H) P_k^- \quad (4.26)$$

4.3.2. Filtros de Kalman Extendido(EKF)

EKF es una extensión del filtro de Kalman para caso no lineales, para ello se consideran las ecuaciones generales de un sistema estocástico no lineal:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \quad (4.27)$$

$$z_k = h(x_k, v) \quad (4.28)$$

Las cuales se pueden escribir de la siguiente forma linealizada:

$$x_k = \hat{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + W w_{k-1} \quad (4.29)$$

$$z_k = \hat{z}_k + H(x_k - \hat{x}_k) + V v_k \quad (4.30)$$

Donde además se definen las matrices A , W , H y V según:

$$A = \frac{\partial f[i]}{\partial x[j]}(\hat{x}_{k-1}, u_{k-1}, 0) \quad (4.31)$$

$$W = \frac{\partial f[i]}{\partial w[j]}(\hat{x}_{k-1}, u_{k-1}, 0) \quad (4.32)$$

$$H = \frac{\partial h[i]}{\partial x[j]}(\hat{x}_{k-1}, 0) \quad (4.33)$$

$$V = \frac{\partial h[i]}{\partial x[j]}(\hat{x}_{k-1}, 0) \quad (4.34)$$

Luego, el algoritmo del EKF puede enunciarse de la siguiente manera; en la primera parte (Predicción), se calcula el vector de estados estimados a priori, usando para ello la ecuación 4.27 asumiendo error en el proceso igual a cero (ecuación 4.35), y seguidamente se calcula la covarianza a priori del error, P_k^- , empleando para ello la nueva covarianza variable del proceso, Q_{k-1} , (ecuación 4.36)

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \quad (4.35)$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \quad (4.36)$$

Y por otra parte en la etapa de Corrección, se introduce la nueva covarianza de la medición para el cálculo de la ganancia K_k mediante la ecuación 4.37, luego para obtener del vector de estados estimados a posteriori \hat{x}_k se corrige la estimada a priori \hat{x}_k^- con la innovación, esta vez calculada en base a la ecuación . Por último la varianza del error es actualizada mediante la ecuación 4.39.

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \quad (4.37)$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0)) \quad (4.38)$$

$$P_k = (I - K_k H_k) P_k^- \quad (4.39)$$

En resumen el algoritmo de un EKF se presenta en la figura 4.11.

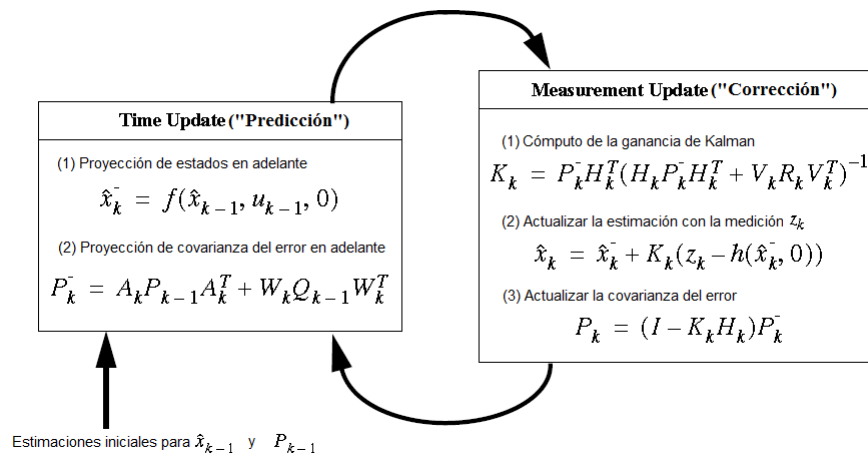


Figura 4.11 – Esquema del algoritmo EKF[18].

4.4. Diseño del Sistema de Localización usando Fusión de Sensores

4.4.1. Caracterización de los Sensores

4.4.1.1. Odometría

En la odometría se puede definir el siguiente vector $x = [v_x \ v_y \ \phi]^T$, donde v_x es la velocidad de traslación en la dirección x local, v_y es la velocidad de traslación en la dirección y local y ϕ la diferencia angular del robot con respecto a la orientación inicial, además se pueden definir las ecuaciones de estado 4.40, 4.41 y 4.42 para estimar la posición respecto a las coordenadas locales.

$$p_{x(k)}^o = p_{x(k-1)}^o + v_x T_s \quad (4.40)$$

$$p_{y(k)}^o = p_{y(k-1)}^o + v_y T_s \quad (4.41)$$

$$\phi_{(k)}^o = \phi_{(k-1)}^o + \omega_{(k-1)}^o T_s \quad (4.42)$$

Donde p_x^o y p_y^o son las posiciones x e y estimadas por medio de los encoders, además una relación entre la medida de los encoders y las velocidades de traslación se puede enunciar según las ecuaciones 2.20 y 2.21 presentadas en el capítulo 2.

4.4.1.2. IMU

Los sensores inerciales son esenciales en la estimación de la orientación del robot, además permiten incluir en la estimación el efecto de los deslizamientos en las ruedas del robot.

Con este objetivo se plantean las ecuaciones para la estimación de la velocidad en 4.43 y 4.44; y para la estimación del ángulo con respecto al norte magnético 4.45.

$$v_{x(k)}^a = v_{x(k-1)}^a + a_{x(k-1)}^a T_s \quad (4.43)$$

$$v_{y(k)}^a = v_{y(k-1)}^a + a_{y(k-1)}^a T_s \quad (4.44)$$

$$\phi_{(k)}^g = \phi_{(k-1)}^g + \omega_{(k-1)}^g T_s \quad (4.45)$$

Además de ello se realizó la implementación de un e-compass para la estimación en el ángulo yaw.

4.4.1.3. WSN

La ecuación ideal 4.15 puede discretizarse obteniéndose la ecuación 4.46, esta última ecuación representa la medición en cada uno de los 4 puntos de acceso usados.

$$y_{i(k)} = P_o - 10\gamma \log_{10}(\sqrt{(p_{x(k)}^r - p_x^i)^2 + (p_{y(k)}^r - p_y^i)^2} + v_{(k)}^r) \quad (4.46)$$

La estimación de la posición a partir de la potencia de equipos WSN es un campo de estudio emergente y de gran interés. No obstante, la estimación en base a este medio es altamente compleja y posiblemente los mejores resultados se pueden obtener al usar filtros de Kalman[125].

El equipo usado en este proyecto se muestra en la figura 4.12, el cual es un punto de acceso de la marca TP-Link y de modelo TL-WR841HP, su potencia máxima puede alcanzar los 10dBm.



Figura 4.12 – Punto de Acceso TP-Link TL-WR841HP.

4.4.2. Estimación de la Orientación

4.4.2.1. Estimación de los ángulos de Euler

Para la implementación de la estimación de la orientación[19], y su posterior integración al robot, se desarrollo la interconexión entre el smartphone y Arduino. Para ello se desarrollo una aplicación en Android (Anexo B.1) que reporta al Arduino (Anexo B.2) las medidas del acelerómetro, giroscopio y magnetómetro cada 20ms (figura 4.13).



Figura 4.13 – Comunicación entre Android y Arduino en funcionamiento.

Es necesario para este fin la implementación de un compass con compensación de inclinación, el sistema de coordenadas elegido para el desarrollo del mismo fue el sistema NED (North-East-Down), el cual se muestra en la figura 4.14.

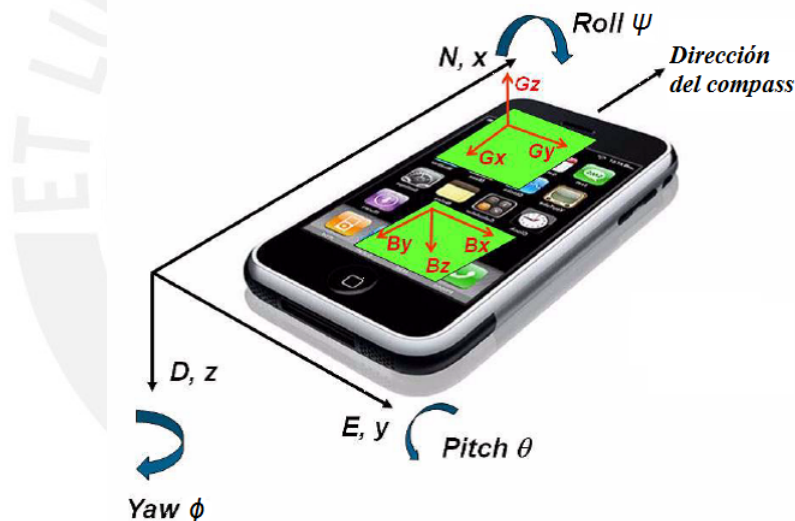


Figura 4.14 – Sistema de coordenadas NED[19]

Las medidas de los sensores son recibidas y procesadas en el arduino para la implementación del compass, el cual se desarrolló como sigue:

1. Se caracterizan las medidas del acelerómetro (g_x , g_y y g_z) en base a una rotación de la medición ideal, esto es considerando solo g en z :

$$\begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = R_{(x,\psi)} R_{(y,\theta)} R_{(z,\phi)} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \quad (4.47)$$

Donde las matrices de rotación extraídas a partir del sistema propuesto en la figura 4.14 son:

$$R_{(x,\psi)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \quad (4.48)$$

$$R_{(y,\theta)} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \quad (4.49)$$

$$R_{(z,\phi)} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.50)$$

2. Se caracterizan las medidas del magnetómetro (m_x , m_y y m_z) en base a una rotación de la medición ideal, esto es la magnitud del campo magnético según la longitud y latitud de la ubicación en z :

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = R_{(x,\psi)} R_{(y,\theta)} R_{(z,\phi)} \begin{bmatrix} B\cos\delta \\ 0 \\ B\sin\delta \end{bmatrix} \quad (4.51)$$

3. Se despejan los ángulos en base a las relaciones establecidas en 4.47 y 4.51, obteniéndose 3 relaciones las cuales se calculan en el siguiente orden:

$$\psi = \text{atan}\left(\frac{g_y}{g_z}\right) \quad (4.52)$$

$$\theta = \text{atan}\left(\frac{-g_x}{g_y \sin\psi + g_z \cos\psi}\right) \quad (4.53)$$

$$\phi = \text{atan}\left(\frac{m_z \sin\psi - m_y \cos\psi}{m_x \cos\theta + m_y \sin\theta \sin\psi + m_z \sin\theta \cos\psi}\right) \quad (4.54)$$

El e-compass trabaja con las medidas proporcionadas por el acelerómetro y el magnetómetro, no obstante una mejor respuesta puede obtenerse si se integran las medidas del giroscopio, para ello se implementó la fusión de estos sensores mediante un filtro de Kalman.

4.4.2.2. Diseño del filtro de Kalman

Por cada uno de los ejes se define el siguiente vector de estados $x = \begin{bmatrix} \alpha & \omega_b^\alpha \end{bmatrix}^T$, donde α es el ángulo a estimar y ω_b^α el bias en la velocidad angular ω^α , luego se puede definir la siguiente ecuación discreta del proceso:

$$x_{(k)} = \begin{bmatrix} 1 & -T_s \\ 0 & 1 \end{bmatrix} x_{(k-1)} + \begin{bmatrix} T_s \\ 0 \end{bmatrix} \omega_{(k-1)}^\alpha + w_{(k-1)} \quad (4.55)$$

Donde $w_{(k-1)}$ representa el error en el proceso.

Mientras que la ecuación de la medición se define de la siguiente manera:

$$z_{(k)} = \begin{bmatrix} 1 & 0 \end{bmatrix} x_{(k)} + v_{(k)} \quad (4.56)$$

Donde $v_{(k)}$ representa el error en la medición.

Además se definen las matrices $Q = \text{Diag}(Q_\alpha, Q_{\omega_b})$, con $Q_\alpha = 0,01$, $Q_{\omega_b} = 0,02$, mientras que la covarianza R es variable y la covarianza inicial del error, P , se inicializa como una matriz de ceros de dimensión 2×2 .

El filtro usa las ecuaciones 4.22, 4.23, 4.24, 4.25 y 4.26; para la estimación de los estados a priori se utiliza en la entrada ω^α , la medida correspondiente al giroscopio; mientras que en la medición se utiliza el ángulo calculado por el compass.

4.4.2.3. Ajuste en línea de la matriz de covarianza R

Para mejorar la estimación del filtro de Kalman se incorporó un bloque de ajuste usando una red neuro difusa[129], para el cual se uso como entrada la diferencia entre la covarianza teórica y real en la innovación.

Para ello se realiza el cálculo de la covarianza real de la innovación $r_i = x_i - h(x_i^-, 0)$, dada por la ecuación 4.57; además se tiene la expresión 4.58 la cual expresa la covarianza teórica[129].

$$\hat{C}_i = \frac{1}{N} \sum_{j=j_0}^i r_j r_j^T \quad (4.57)$$

$$S_i = H_i P_i^- H_i^T + R_i \quad (4.58)$$

La diferencia entre ambas $D_i = S_i - \hat{C}_i$ es utilizada como entrada para la red neuro difusa, para lo cual se definieron tres variables lingüísticas: diferencia positiva (PO), diferencia cero (CE) y diferencia negativa (NE); y cuyas funciones de pertenencia se muestran en la figura 4.15.

Así mismo se definieron las siguientes reglas:

1. Si la entrada es NE entonces se realiza un decremento en la covarianza.
2. Si la entrada es CE entonces el mantiene la covarianza.
3. Si la entrada es PO entonces se realiza un incremento en la covarianza.

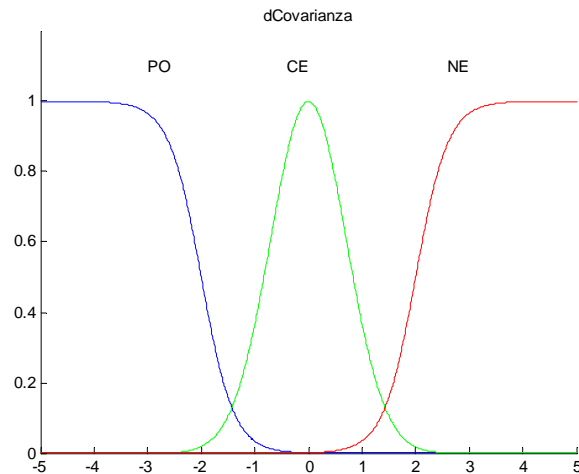


Figura 4.15 – Funciones de pertenencia de la red neuronal para el ajuste de covarianza.

La salida de la red neuro difusa, ΔR_i , es luego sumada en línea a la covarianza R con lo cual se mejora la estimación como se muestra en el siguiente punto.

4.4.2.4. Pruebas del estimador de la orientación

Para realizar las pruebas del estimador se implementó una aplicación en Guide de Matlab la cual recibe los datos desde el Arduino, y en donde además de mostrarse las estimaciones de los tres ángulos de Euler mediante el filtro de Kalman, se presentan además las estimaciones al usar solo los datos medidos por el giroscopio y por la fusión implementada en un filtro complementario.

En las figuras 4.16 y 4.17 se muestran algunas pruebas realizadas con la aplicación.

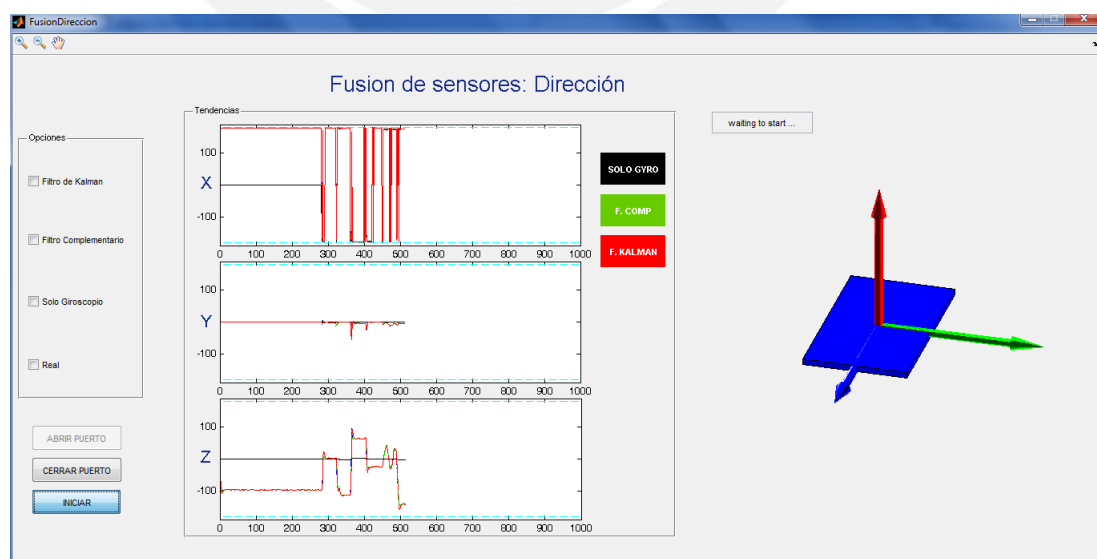


Figura 4.16 – Estimación de los ángulos.

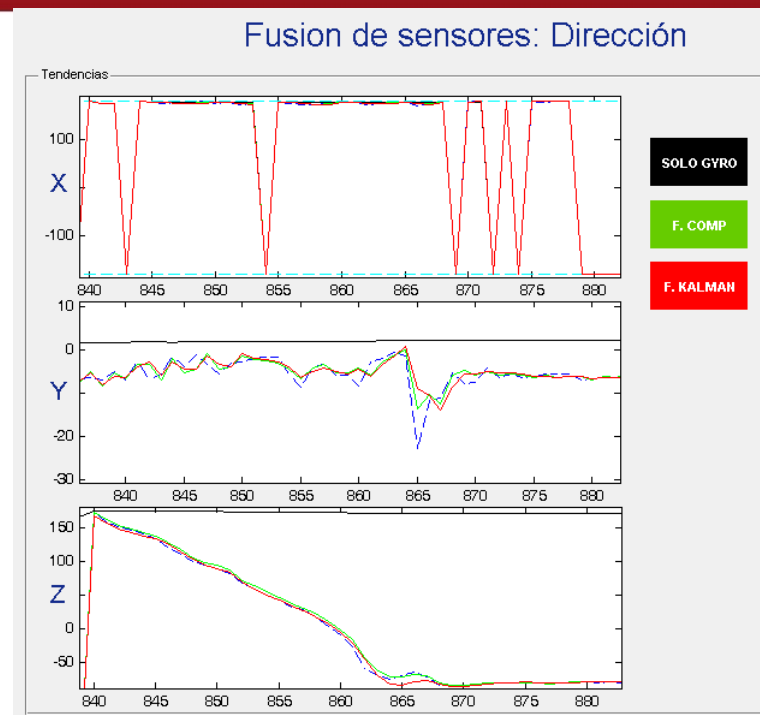


Figura 4.17 – Estimación de los ángulos usando filtro de Kalman y filtros complementarios.

Además con el fin de obtener la variabilidad se ensayó con el celular en una posición estática como se muestra en la figura 4.18, se puede apreciar el ensayo de sólo el filtro de Kalman con dos valores de R fijos y el resultado cuando se usa el filtro de Kalman con ajuste por red neuro difusa, en la figura se aprecian sólo las 100 primeras muestras.

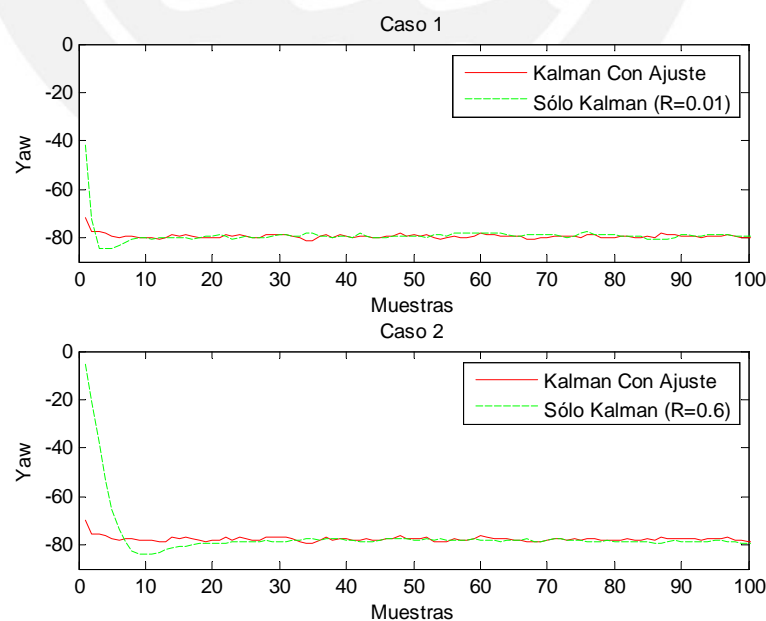


Figura 4.18 – Comparación entre las estimaciones.

En el primer caso, cuando se usa $R = 0.01$ la desviación estandar fue de 1.464 y en

el segundo caso cuando $R = 0.6$ la desviación estandar fue de 3.392, mientras que en ambos caso el filtro de Kalman con ajuste la desviación fue de 0.7128. Por otro lado, si se obvian las primeras muestras y se analiza la desviación estandar en las muestras restantes, se tiene que en el primer caso la desviación fue de 0.7624, en el segundo caso fue 0.5632 y por último cuando se utiliza el ajuste es de 0.6573.

Se puede ver por ende que el estimador de Kalman con ajuste por una red neuro difusa permite mejorar la velocidad de estimación, disminuye el sobreimpulso y estima convenientemente la orientación del ángulo yaw.

4.4.3. Estimación de la Ubicación

4.4.3.1. Diseño del EKF

Para la fusión de sensores se establece el siguiente vector de estados $[p_x \ p_y \ v_x \ v_y \ v_{bx} \ v_{by}]^T$, donde p_x es la posición global en x, p_y es la posición global en y, v_x la velocidad en x, v_y la velocidad en y, v_{bx} el bias en la velocidad de x, y v_{by} el bias en la velocidad de y.

Se definen además las ecuaciones del proceso según las expresiones 4.40, 4.41, 4.43, 4.44:

$$p_{x(k)}^o = p_{x(k-1)}^o + v_{x(k-1)} T_s \quad (4.59)$$

$$p_{y(k)}^o = p_{y(k-1)}^o + v_{y(k-1)} T_s \quad (4.60)$$

$$v_{x(k)}^a = v_{x(k-1)}^a + a_{x(k-1)}^a T_s \quad (4.61)$$

$$v_{y(k)}^a = v_{y(k-1)}^a + a_{y(k-1)}^a T_s \quad (4.62)$$

$$v_{bx(k)} = v_{bx(k-1)} \quad (4.63)$$

$$v_{by(k)} = v_{by(k-1)} \quad (4.64)$$

Las cuales pueden sumarse en la ecuación 4.65:

$$x(k) = \begin{bmatrix} 1 & 0 & T_s & 0 & 0 & 0 \\ 0 & 1 & 0 & T_s & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x(k-1) + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ T_s & 0 \\ 0 & T_s \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u(k-1) \quad (4.65)$$

Mientras que para las ecuaciones de la medición se definieron las siguientes ecuaciones:

$$v_{x(k)}^R = -v_{x(k)} \sin \psi_{(k)} + v_{y(k)} \cos \psi_{(k)} + v_{bx(k)} \quad (4.66)$$

$$v_{y(k)}^R = v_{x(k)} \cos \psi_{(k)} + v_{y(k)} \sin \psi_{(k)} + v_{by(k)} \quad (4.67)$$

Estas dos primeras ecuaciones pueden sumarse en 4.68:

$$z_{1-2}(k) = \begin{bmatrix} 0 & 0 & -\sin\psi_{(k)} & \cos\psi_{(k)} & 1 & 0 \\ 0 & 0 & \cos\psi_{(k)} & \sin\psi_{(k)} & 0 & 1 \end{bmatrix} x_{(k)} + v_{1-2}(k) \quad (4.68)$$

Y las ecuaciones de medición se completan con las medidas obtenidas de los cuatro puntos de acceso usados:

$$z_{3(k)} = P_{o1} - 10\gamma_1 \log_{10}(\sqrt{(p_{x(k)}^r - p_x^1)^2 + (p_{y(k)}^r - p_y^1)^2}) + v_{3(k)} \quad (4.69)$$

$$z_{4(k)} = P_{o2} - 10\gamma_2 \log_{10}(\sqrt{(p_{x(k)}^r - p_x^2)^2 + (p_{y(k)}^r - p_y^2)^2}) + v_{4(k)} \quad (4.70)$$

$$z_{5(k)} = P_{o3} - 10\gamma_3 \log_{10}(\sqrt{(p_{x(k)}^r - p_x^3)^2 + (p_{y(k)}^r - p_y^3)^2}) + v_{5(k)} \quad (4.71)$$

$$z_{6(k)} = P_{o4} - 10\gamma_4 \log_{10}(\sqrt{(p_{x(k)}^r - p_x^3)^2 + (p_{y(k)}^r - p_y^3)^2}) + v_{6(k)} \quad (4.72)$$

La posición es luego estimada por un filtro de Kalman extendido según las ecuaciones 4.73, 4.74, 4.75, 4.76 y 4.77.

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \quad (4.73)$$

$$P_k^- = A_k P_{k-1} A_k^T + Q_{k-1} \quad (4.74)$$

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (4.75)$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0)) \quad (4.76)$$

$$P_k = (1 - K_k H_k) P_k^- \quad (4.77)$$

La matriz A es obtenida de la ecuación 4.65, y la matriz H mediante el Jacobiano de h (ecuación 4.33) evaluado en \hat{x}_{k-1} .

4.4.3.2. Pruebas del estimador por EKF

Para realizar las pruebas con el filtro se simuló en Matlab la velocidad en un robot móvil del tipo skid-steer y las potencias transmitidas por los puntos de acceso, la velocidad medida se obtiene de los encoders de cada par de ruedas (derecha e izquierda) y se transforma en velocidades de traslación en coordenadas x e y locales.

Además se incluyo ruido para cada medición, en los encoders se incluye el ruido propio de la cuantización y un ruido gaussiano de media 0 y desviación estándar 0.1, mientras que en los puntos de acceso se incluye ruido de media cero y desviación estándar 3.

Los resultados se muestran en las figuras 4.19, recorrido usado, 4.20, coordenadas x, y 4.21, recorrido en y, para los datos reales y la estimación por medio de EKF para las señales con ruido.

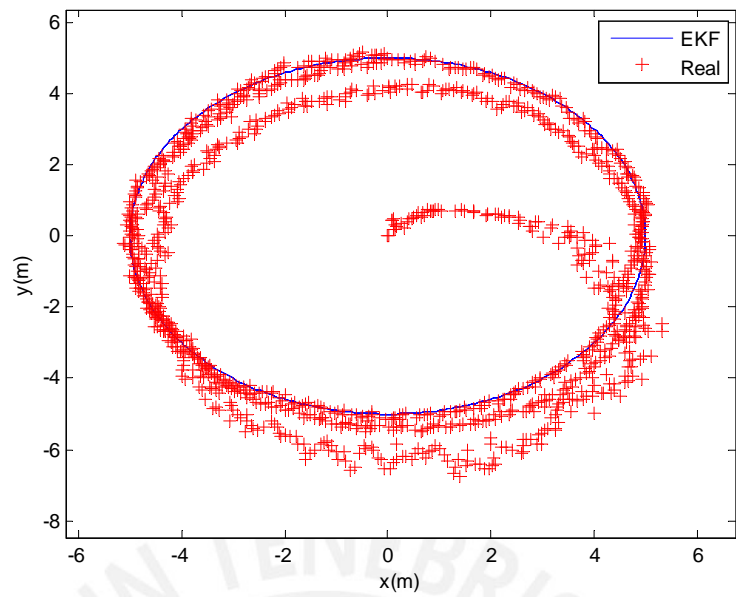


Figura 4.19 – Recorrido usado para la estimación

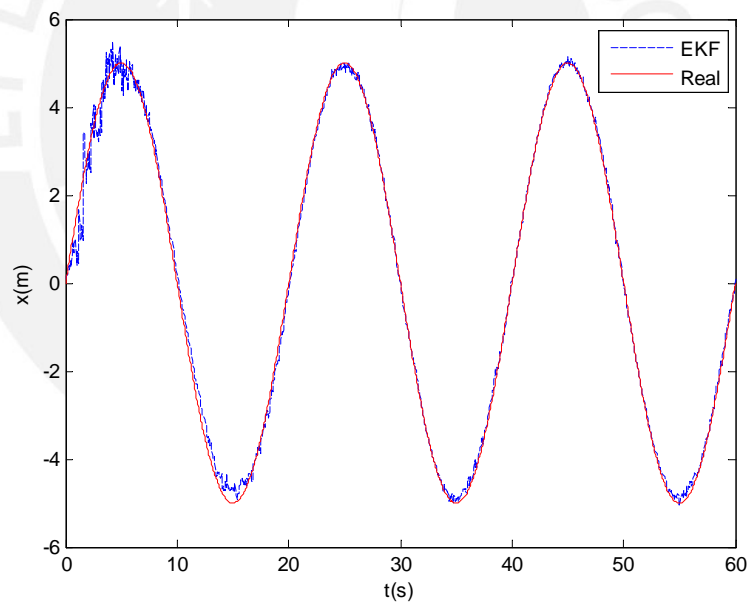


Figura 4.20 – Datos en la coordenada X.

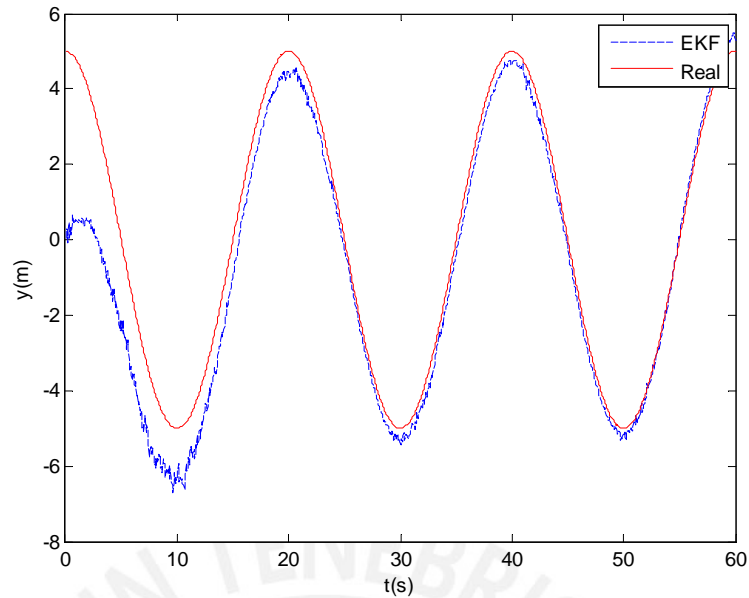


Figura 4.21 – Datos en la coordenada y.

La estimación para la coordenada X tuvo en este caso un RMSE de 0.4491 y MAE de 0.2017; mientras que en la coordenada Y un RMSE de 0.8929 y MAE de 0.7973, la estimación en Y tarda más debido a que las condiciones iniciales están más alejadas en esta coordenada.

Para evaluar el desempeño de la estimación dado un error en la orientación se evaluó nuevamente el EKF con la inclusión de un error en la orientación de media $\mu = 0$ y desviación estándar de $\sigma = 0,5$, además se mantuvieron los errores anteriores, los resultados en este caso se muestran en figuras 4.22, 4.23 y 4.24.

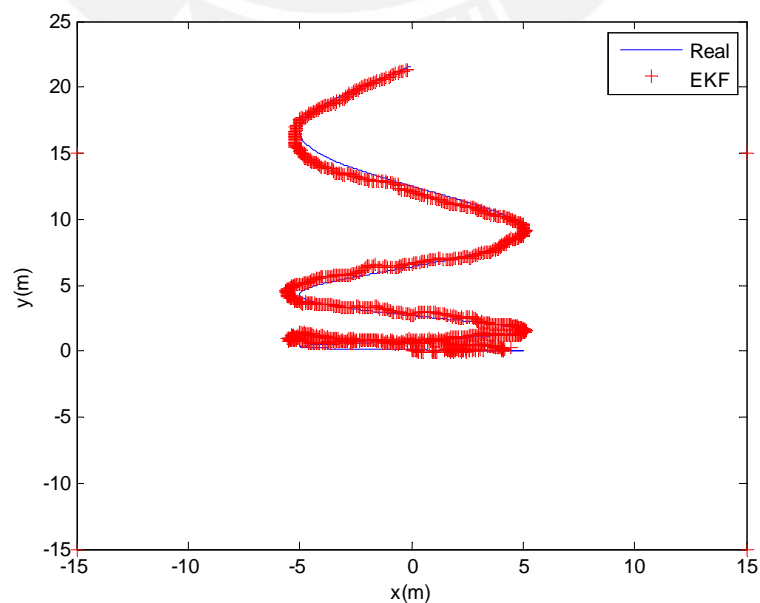


Figura 4.22 – Prueba de EKF con ruido gaussiano en la orientación.

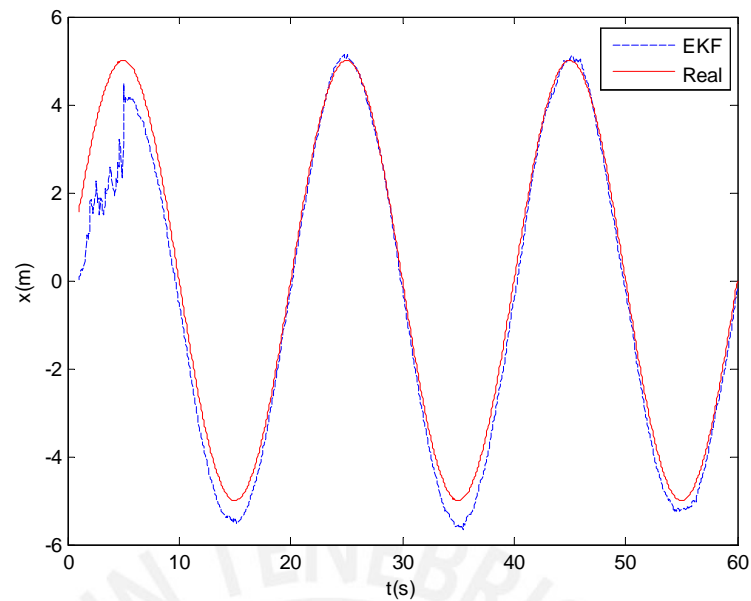


Figura 4.23 – Estimación en la coordenada X.

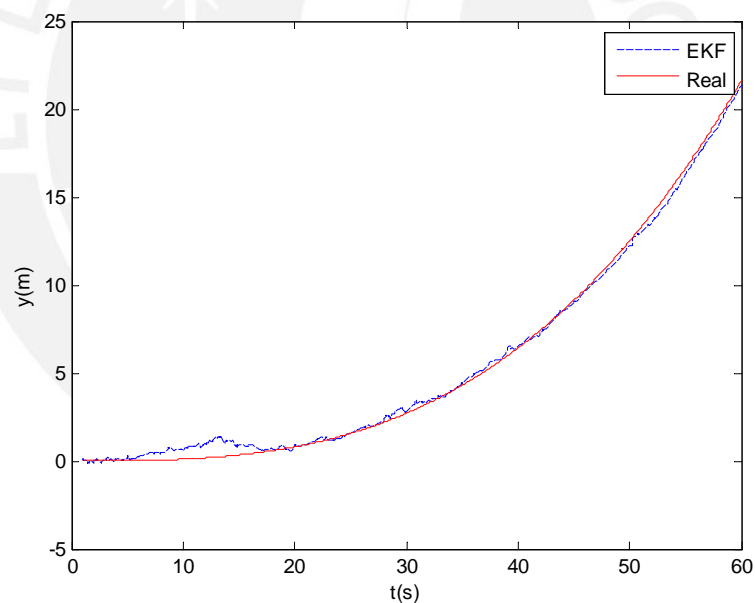


Figura 4.24 – Estimación en la coordenada Y.

La estimación para la coordenada X tuvo en este caso un RMSE de 0.4451 y MAE de 0.6672; mientras que en la coordenada Y un RMSE de 0.2643 y MAE de 0.5141.

En las figuras 4.25 y 4.26 se presenta otro caso en la estimación en las coordenadas X e Y, en donde se mantuvo fija la posición en (5,5), manteniendo además los ruidos modelados antes en cada una de las mediciones.

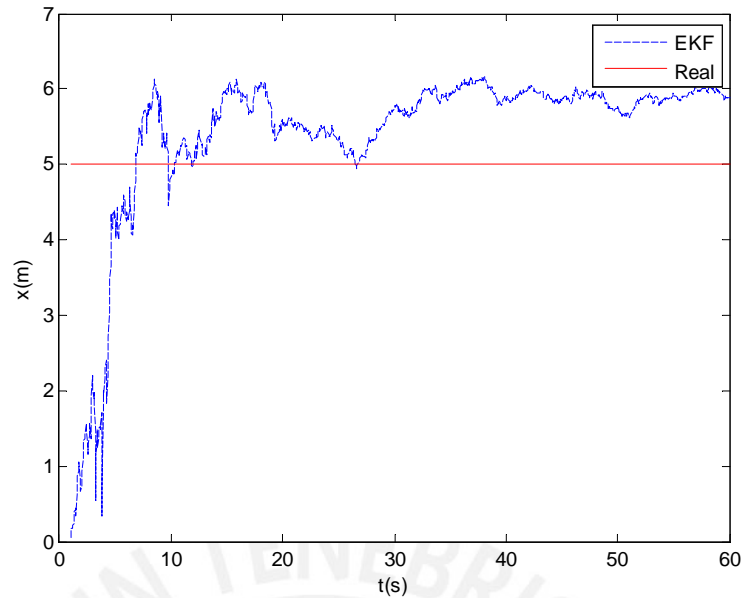


Figura 4.25 – Estimación en coordenada X manteniendo fija la posición.

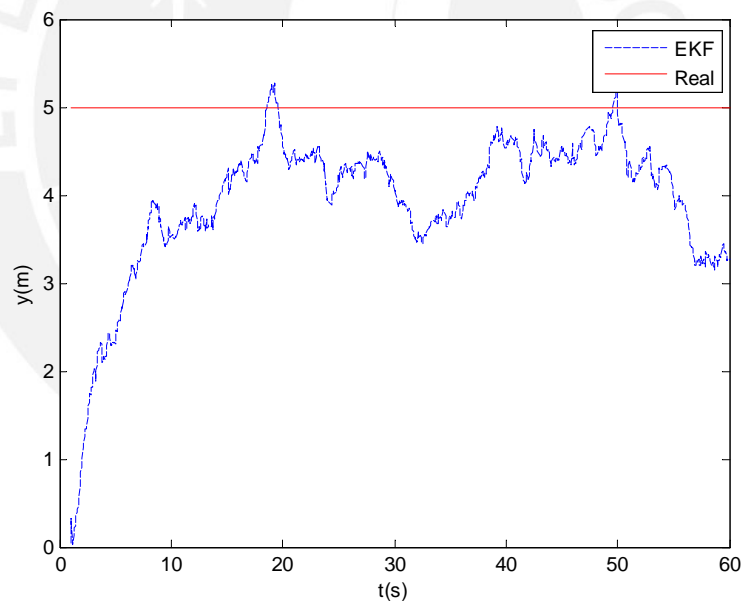


Figura 4.26 – Estimación en coordenada Y manteniendo fija la posición.

La estimación para la coordenada X tuvo en este caso un RMSE de 0.9019 y MAE de 0.9497; mientras que en la coordenada Y un RMSE de 1.0794 y MAE de 1.0389.

Puede verse entonces que el movimiento mejora la estimación debido a que incluye la medida de la velocidad la cual presenta en general una menor incertidumbre, esto nos muestra además que se puede seguir un procedimiento definido para la localización, en la cual se estime en primera instancia la posición del vehículo en un punto fijo antes de realizar el movimiento, seguidamente se inicia el movimiento y da-

do que la estimación estará mejor aproximada debido al primer paso, la estimación en movimiento será más rápida y más precisa.

Para demostrar la viabilidad de este último procedimiento se realizó nuevamente el ensayo de la trayectoria circular, esta vez con las coordenadas x e y iniciales muy lejanas a las reales y con errores en todas las mediciones, las figuras 4.27 y 4.28 muestran la trayectoria y posiciones reales y estimadas si se realizara la fusión únicamente durante el movimiento del robot móvil, la estimación para la coordenada X en este caso tiene un RMSE de 1.0314 y MAE de 1.0156; mientras que la coordenada Y tiene un RMSE de 1.3717 y MAE de 1.1712; además se puede observar que una buena estimación de la posición es alcanzada recién a partir de los 20 segundos.

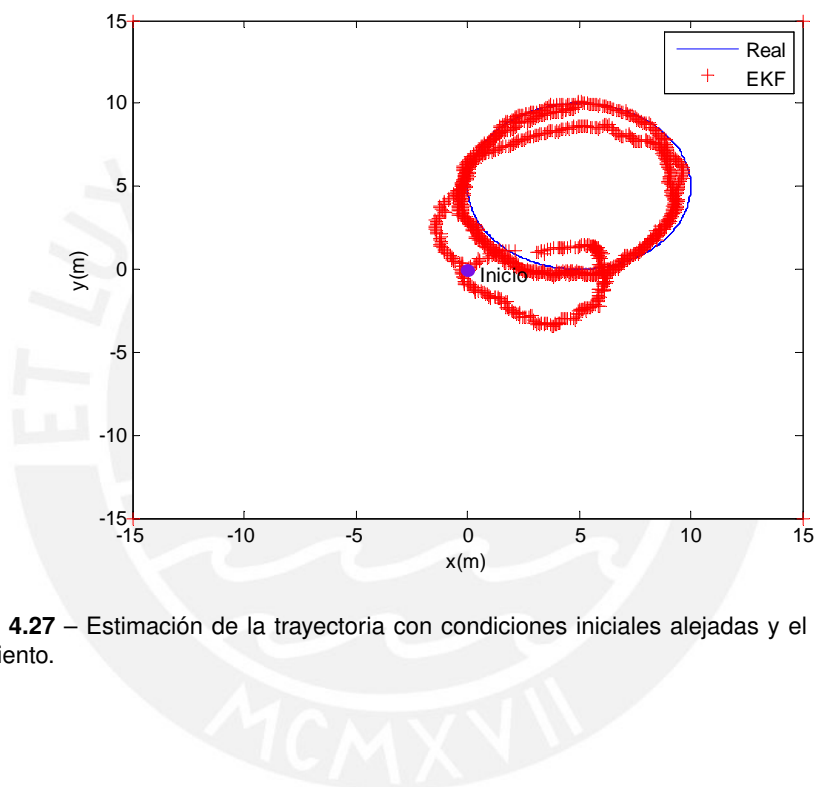


Figura 4.27 – Estimación de la trayectoria con condiciones iniciales alejadas y el robot móvil en movimiento.

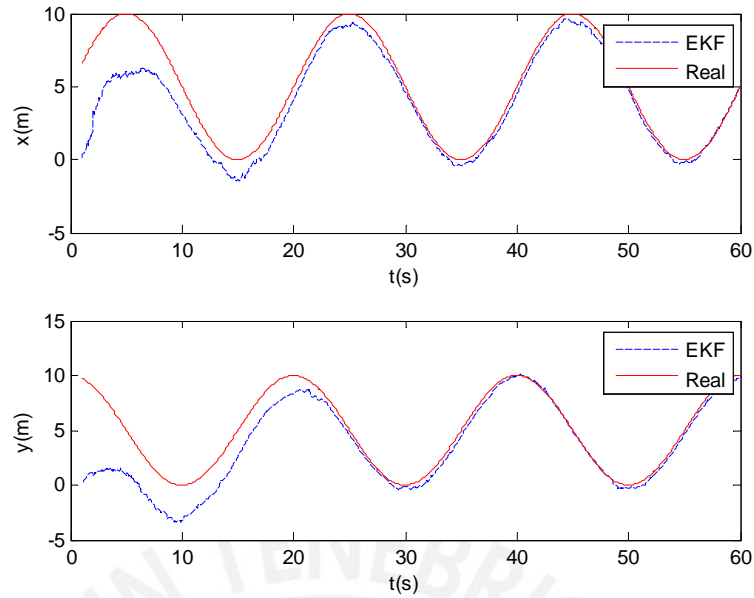


Figura 4.28 – Estimación de las coordenadas X e Y con condiciones iniciales alejadas y el robot móvil en movimiento.

Seguidamente se ensayó la fusión, esta vez con una estimación previa con el robot móvil estático, en las figuras 4.29 y 4.30 se muestra la trayectoria y estimación de la posición del robot estático durante 10 segundos, la posición estimada en esta instancia es $x = 4,8593$ e $y = 7,0954$; mientras que en las figuras 4.31 y 4.32 se muestran la trayectoria y posiciones reales y estimadas durante la fusión con el robot móvil en movimiento, la estimación para la coordenada X en este caso tiene un RMSE de 0.4737 y MAE de 0.2244; mientras que la coordenada Y tiene un RMSE de 0.6171 y MAE de 0.3808; y la estimación es buena a partir de los 5 segundos.

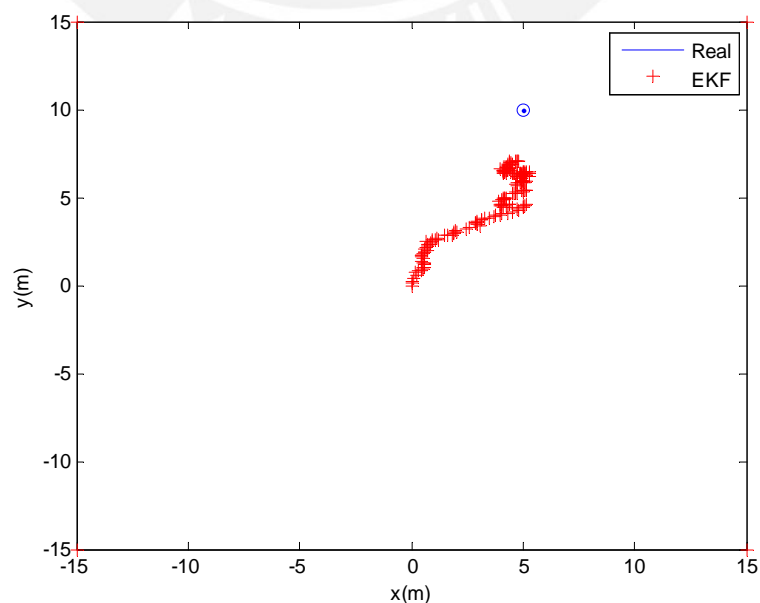


Figura 4.29 – Estimación de la posición con condiciones iniciales alejadas y el robot móvil estático.

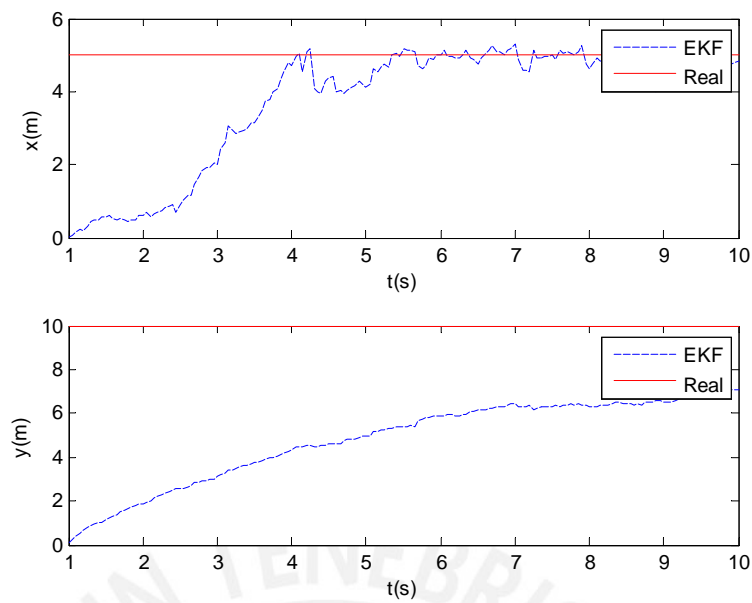


Figura 4.30 – Estimación de las coordenadas X e Y con condiciones iniciales alejadas y el robot móvil estático.

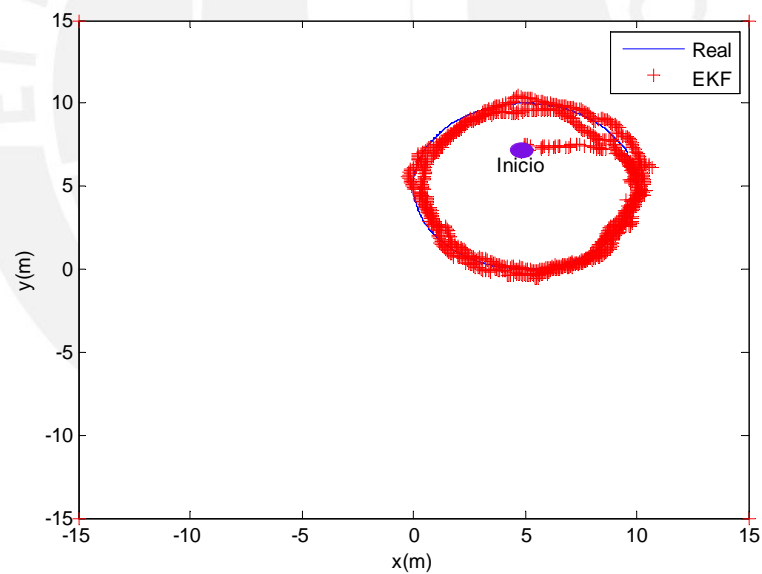


Figura 4.31 – Estimación de la trayectoria con estimación previa y con el robot móvil en movimiento.

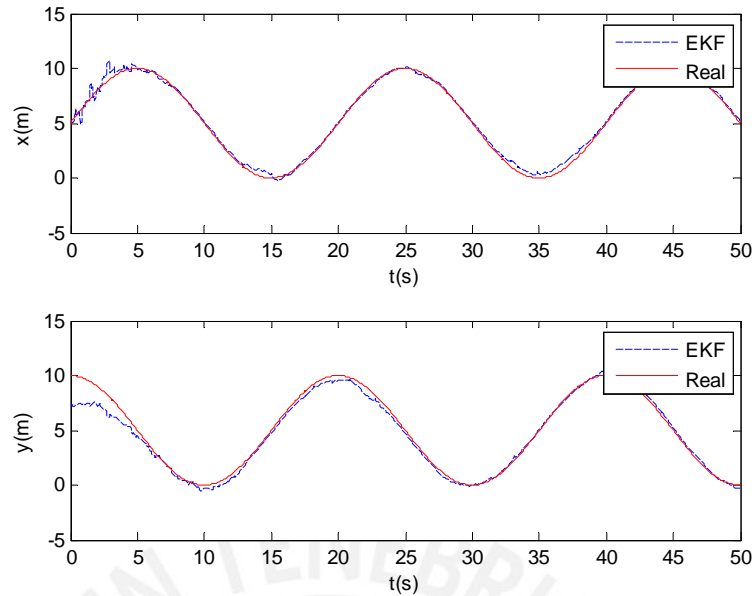


Figura 4.32 – Estimación de las coordenadas X e Y con estimación previa y con el robot móvil en movimiento.

Estos últimos resultados muestran que el procedimiento de inicialización en la estimación de forma estática mejoraría en último término la estimación del robot en todo momento posterior, esta inicialización se daría por única vez y podría mejorarse si se considera un mayor tiempo para el mismo.

4.5. Conclusiones

- Se comprobaron y ajustaron los valores de los sensores del celular inteligente Moto G para la aplicación en las subsiguientes estimaciones.
- Se implementó un e-compass para la estimación de los ángulos de Euler, y se comprobó un error en la estimación de la orientación (yaw) de $\pm 2.5^\circ$.
- Se desarrollo un sistema de fusión de sensores usando un filtro de Kalman para la estimación de la orientación, este estimador requiere del uso de los tres sensores inerciales presentes en el celular inteligente Moto G.
- Se enunciaron las ecuaciones de estado del sistema de sensores y se dedujeron las relaciones necesarias para la estimación de la posición en un sistema de 2 coordenadas.
- Se desarrollo un sistema de fusión de sensores para la estimación de la posición en las coordenadas X e Y, las simulaciones demostraron que se puede obtener una estimación con aproximadamente 1 metro de error cuando las mediciones presentan ruido.

- Se propuso un procedimiento para mejorar la estimación de la posición del robot, el cual permitiría evitar choques aún en condiciones iniciales desfavorables para el recorrido del robot.



Capítulo 5

Pruebas y Resultados del Sistema de Navegación

5.1. Introducción

En el siguiente capítulo se presentan los resultados obtenidos en la implementación del sistema de navegación diseñado en los capítulos 3 y 4, aplicados en un robot P3-AT, y con el uso de sensores odométricos, inerciales y de redes inalámbricas.

Este capítulo se divide en tres partes, en la sección 5.2 se detalla la implementación del controlador y del sistema de localización en el robot móvil P3-AT, en la sección 5.3 se detallan las partes del algoritmo elaborado en C++; y seguidamente en la sección 5.3 muestran algunas pruebas realizadas con el sistema para la navegación de este robot en particular.

5.2. Implementación del Sistema en el Robot P3-AT

El esquema de las interconexiones entre el robot y los componentes del sistema de control se aprecian en la figura 5.1, la computadora a bordo es el ente inteligente y brinda las acciones de control necesarias basado en el algoritmo desarrollado; por otra parte la fusión de sensores se encuentra descentralizada y es realizada por el microcontrolador Arduino.

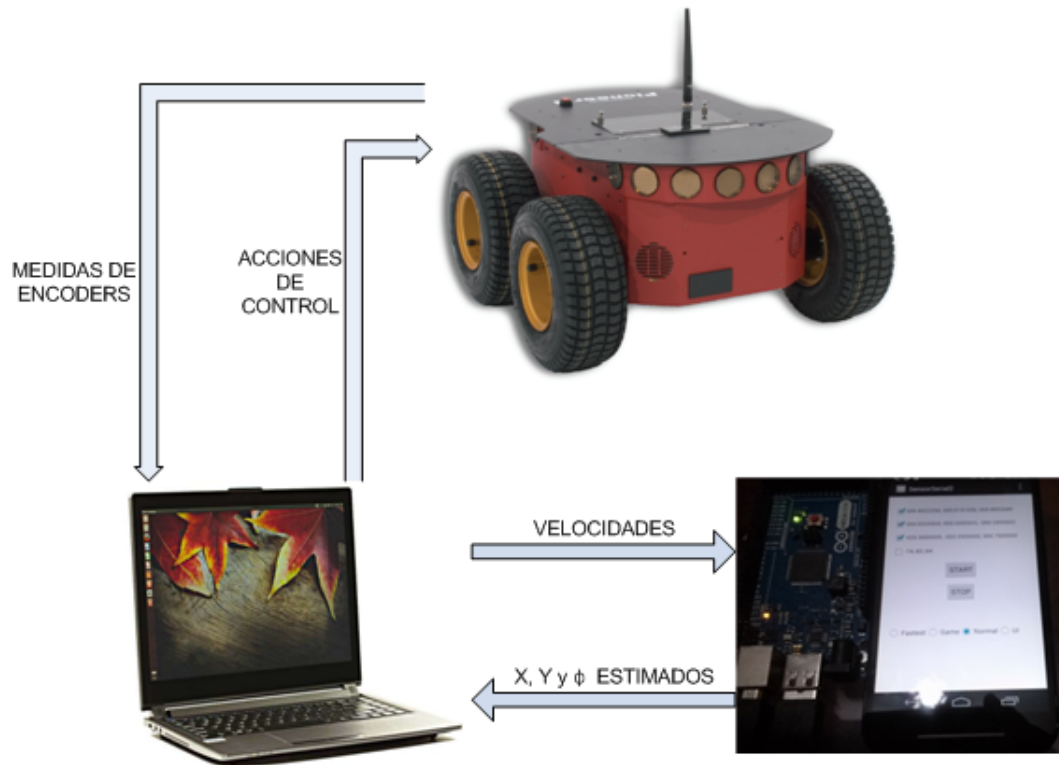


Figura 5.1 – Conexiones del robot.

El microcontrolador se conecta con la computadora mediante conexión serial de dos vías, lo cual permite tanto el envío como la recepción de datos en el Arduino. Los datos del acelerómetro, giroscopio, magnetómetro y el sensor WSN son recibidos desde un celular inteligente Android y las medidas de la velocidad desde la computadora a bordo.

Por su parte la computadora a bordo obtiene las medidas de los encoders desde el robot y las envía al Arduino para su procesamiento; en el siguiente ciclo la estimación de la posición es recogida por el controlador para aplicar su algoritmo y definir la acción de control a usar.

Un punto de acceso inalámbrico (figura 5.2) es empleado para la interconexión remota de una laptop personal con el robot (figura 5.3), la red lan implementada permite programar el robot remotamente y también puede ser empleada en otras aplicaciones como en la interconexión con una aplicación Android.



Figura 5.2 – Switch inalámbrico Trendnet para la interconexión del robot con una laptop personal.

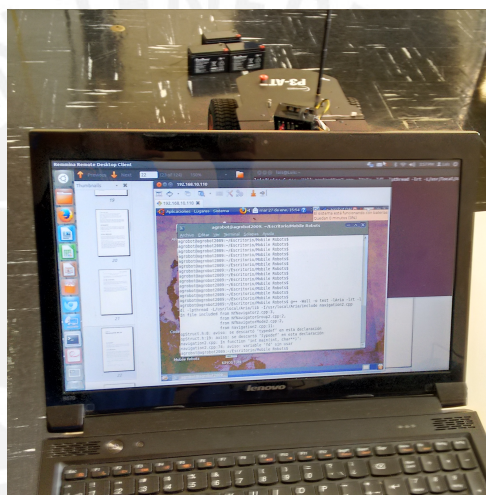


Figura 5.3 – Acceso desde Laptop al sistema operativo del computador a bordo.

El sistema de localización se conecta al robot usando una conexión serial, el celular inteligente es ubicado de forma perpendicular al robot, posición en la cual se reporta la orientación de 0 grados como se aprecia en la figura 5.4, el celular inteligente a su vez esta conectado con el Arduino; ambos elementos se fijan en la carcasa superior del robot.

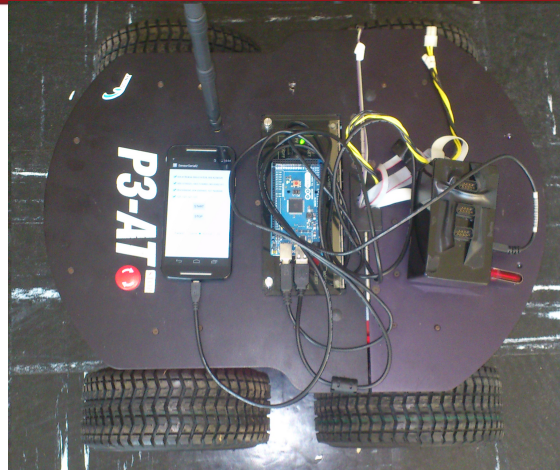


Figura 5.4 – Interconexión del robot con el sistema de localización.

5.3. Implementación del Código en C++ usando la Librería Aria

La librería Aria está implementada con la finalidad de brindar la interconectividad con los robots móviles de MobileRobots, la librería está orientada a objetos y esta especialmente desarrollada para el lenguaje C++ lo que permite acceder al equipo a nivel alto o bajo de la programación, incluye además algunas utilidades para la programación general del robot y soporte para plataformas Windows y Linux.

La computadora a bordo tiene instalada esta librería lo cual le permite realizar una conexión servidor-cliente con el robot P3-AT; además, como se muestra en la figura 5.5, el uso de los objetos ArRobot de la librería Aria permite la ejecución cíclica de las rutinas programadas.

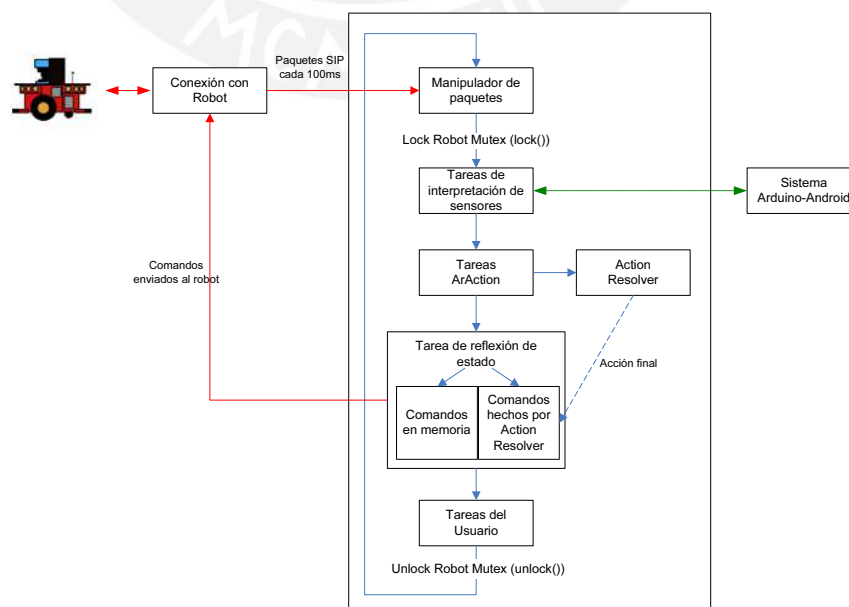


Figura 5.5 – Ciclo de procesamiento en el robot Pioneer P3-AT usando la librería Aria.

El ciclo comprende las siguientes tareas; la manipulación de los paquetes de información de servidor (SIP) que son reportados cada 100 ms por el robot, estos cargan consigo datos actuales de los sensores tales como la posición estimada, las velocidades, las lecturas de los sonares y el voltaje en las baterías entre otros datos.

La siguiente tarea del ciclo está diseñada para la atención a las tareas de interpretación de sensores, se añadió en este punto una tarea con fines de interconexión serial con el microcontrolador Arduino la cual se verá en detalle más adelante.

El siguiente paso consiste en la ejecución de las tareas del tipo `ArAction`, el cual es un tipo de objeto que permite la implementación de algoritmos complejos para su ejecución haciendo una llamada a la función `ArAction::fire` mediante un concepto conocido como `overloading`, además dado que se pueden implementar varias acciones actuando en paralelo se hace necesario el uso de un `ActionResolver` para definir la acción final.

La subclase `ArAction` implementada puede observarse en el Anexo B.3, para la coexistencia con otras `ArAction` se elaboró además un `ArGroup` que coordina todas las acciones en uso, un último paso considerado fue también la elaboración de una subclase `ArMode` que permite llamar a este grupo de acciones cuando se presiona una tecla del teclado, esto posibilita en última instancia el cambio de modo en cualquier instante según el deseo del usuario.

Otro aspecto importante en la elaboración de este sistema es la implementación de la comunicación serial en C++, para ello se hizo uso de la librería `termios.h`, la cual permite manejar las interfase de entrada y salida en Linux.

La parte principal en la comunicación serial es la apertura y configuración del puerto serial a usarse, para ello se deben configurar convenientemente algunas banderas o flags en la estructura `termios` que luego se copia al puerto, el algoritmo 5.1 muestra la configuración necesaria para la conexión con el microcontrolador Arduino.

La línea 5 abre la comunicación en el puerto indicado y define que este se usará para leer y escribir, así como también que no se asumirá el control del puerto ni se esperará la llegada de la señal DCD. En la línea 14 por otro lado se configura la velocidad, el número de bits y la paridad a usarse; otra línea importante es la 15 donde se indica no tomar en cuenta los caracteres CR (retorno de carro); una vez configurados estos parámetros en la estructura `termios`, se debe hacer una pausa en la línea 35, seguido de un reset del puerto en la línea 36, por último se copia la estructura a la configuración del puerto.

Algoritmo 5.1 Rutina de inicialización de la comunicación serial.

```

1  int startSerial() {
2      struct termios oldtio, newtio;
3      int fd;
4      // Abre y configura el puerto serial 1
5      fd = open("/dev/ttyACM1", O_RDWR | O_NOCTTY | O_NDELAY);
6      if (fd == -1)
7      {
8          // Reportar si no se pudo abrir el puerto
9          perror("open_port: Unable to open /dev/ttyACM1");
10     }
11     tcgetattr(fd, &oldtio); // guardar atributos antiguos
12     bzero(&newtio, sizeof(newtio)); // poner a cero todas las banderas
13     // configura velocidad, numero de bits, paridad, modo local y de receptor
14     newtio.c_cflag = B9600 | CRTSCTS | CS8 | CLOCAL | CREAD;
15     newtio.c_iflag = IGNPAR | IGNCR | ICRNL; // ignorar CR en la entrada
16     newtio.c_oflag = 0;
17     newtio.c_lflag = ICANON; // entrada canónica
18     newtio.c_cc[VINTR] = 0; /* Ctrl-c */
19     newtio.c_cc[VQUIT] = 0; /* Ctrl-\ */
20     newtio.c_cc[VERASE] = 0; /* del */
21     newtio.c_cc[VKILL] = 0; /* @ */
22     newtio.c_cc[VEOF] = 4; /* 4 intentos */
23     newtio.c_cc[VTIME] = 0; /* inter-character timer unused */
24     newtio.c_cc[VMIN] = 1; /* bloquear hasta el arribo de 1 dato */
25     newtio.c_cc[VSWTC] = 0; /* '\0' */
26     newtio.c_cc[VSTART] = 0; /* Ctrl-q */
27     newtio.c_cc[VSTOP] = 0; /* Ctrl-s */
28     newtio.c_cc[VSUSP] = 0; /* Ctrl-z */
29     newtio.c_cc[VEOL] = 0; /* '\0' */
30     newtio.c_cc[VREPRINT] = 0; /* Ctrl-r */
31     newtio.c_cc[VDISCARD] = 0; /* Ctrl-u */
32     newtio.c_cc[WERASE] = 0; /* Ctrl-w */
33     newtio.c_cc[VLNEXT] = 0; /* Ctrl-v */
34     newtio.c_cc[VEOL2] = 0; /* '\0' */
35     ArUtil::sleep(200); // dormir 200 ms
36     tcflush(fd, TCIFLUSH); // limpiar el puerto
37     tcsetattr(fd, TCSANOW, &newtio); // fijar los nuevos valores
38     return fd; // retornar el id de la comunicación
39 }

```

Una vez iniciado el puerto, la entrada y salida de datos puede ser realizada de manera similar a una lectura y escritura de datos en un archivo, y para ello se utilizan los comandos `read()` y `write()` respectivamente.

5.4. Ensayos con el Sistema de Navegación

Los recorridos ensayados se muestran en las figuras 5.6, 5.7, 5.8 y 5.9; en todos los casos las trayectorias permiten llegar a la meta establecida y demuestran que el controlador es funcional.

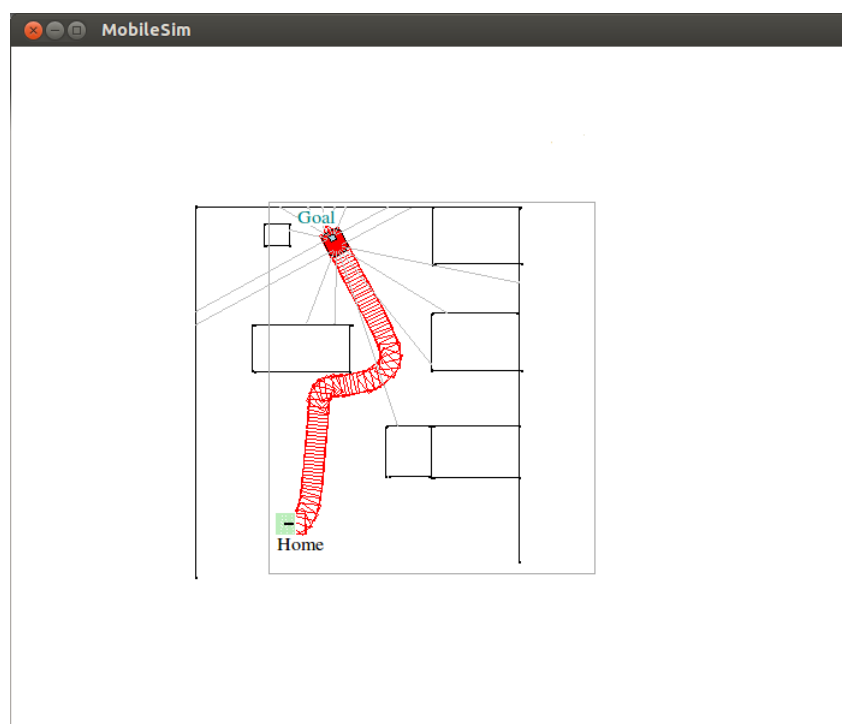


Figura 5.6 – Recorrido 1.

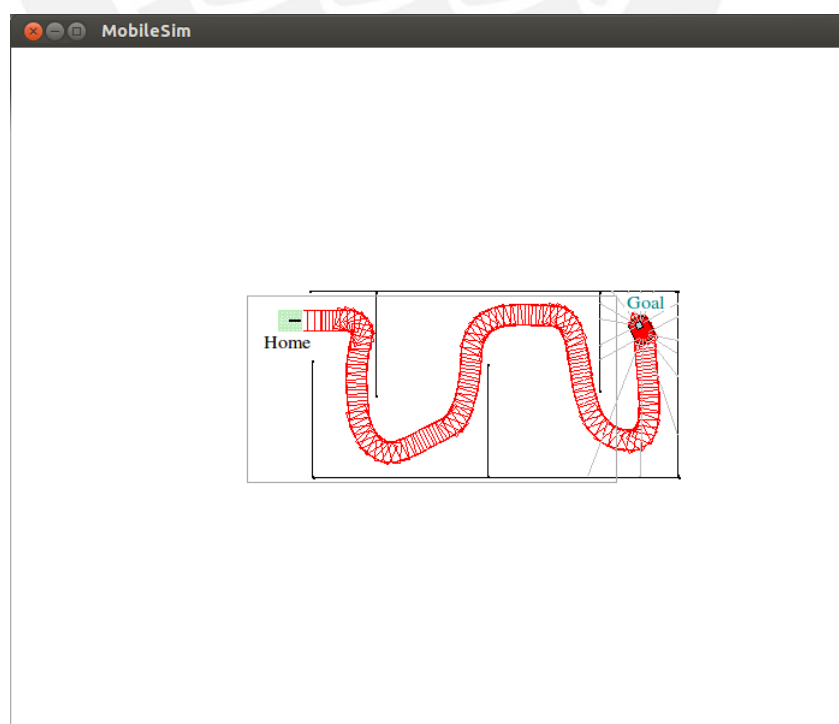


Figura 5.7 – Recorrido 2.

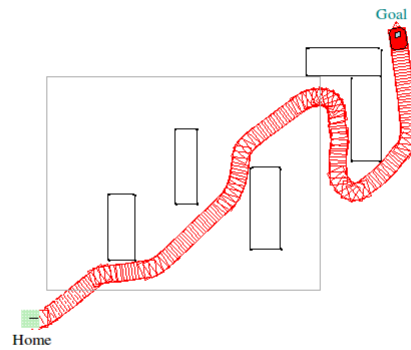


Figura 5.8 – Recorrido 3

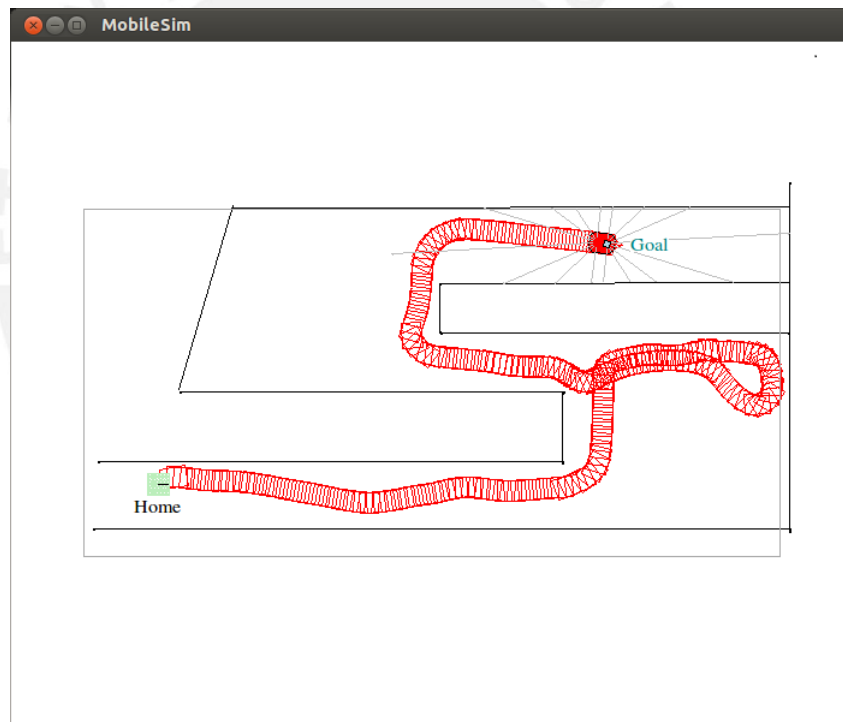


Figura 5.9 – Recorrido 4.

5.5. Conclusiones preliminares

- Se enunciaron los principales conceptos para la implementación del sistema de navegación, así como de las interconexiones necesarias.
- Se comprobó la fácil implementación del sistema en un robot móvil P3-AT y se mostraron las conexiones necesarias para una supervisión remota.

- Se presentaron las principales pautas para la elaboración del código en C++ donde se consideró además la construcción de un subclase ArMode para integrar el control autónomo con el manual.
- Se presentaron las principales pautas para la implementación de la comunicación serial entre el robot y el sistema de localización en el microcontrolador Arduino.
- Se presentaron algunas de las pruebas llevadas a cabo con el robot, mostrándose la manera en que el robot puede conseguir llevar el recorrido en forma exitosa.



Conclusiones Generales

Se fundamentó la importancia del desarrollo de sistemas de navegación para la robótica moderna, y la importancia de esta última en muchos campos de la tecnología actual. Se señaló también que la autonomía en los robots móviles es un campo de amplio interés ya que permitiría flexibilizar aún más las tareas encargadas.

Se desarrolló un sistema de navegación para robots móviles mediante controladores neuro difusos y la construcción de mapas mediante cuadrículas de certeza simplificados, esto permite aprovechar las ventajas de ambas técnicas y muestra que este enfoque puede ser una alternativa en futuros desarrollos de sistemas de navegación autónomos avanzados.

Se diseñó un sistema de localización que utiliza sensores de diversas fuentes, haciendo de especial interés el uso de los presentes en celulares inteligentes, los cuales tienen un uso considerablemente expandido en los últimos años, así como de los sensores odométricos de un robot.

Se implementó un filtro de Kalman para la estimación precisa de la orientación con respecto al norte magnético de la tierra, mientras que para encontrar la posición se implementó un filtro EKF, se mostró como la fusión de varias fuentes de sensado pueden ser integradas coherentemente alcanzando una precisión superior en la estimación.

Se simuló la navegación autónoma usando el algoritmo de control desarrollado y la localización mediante fusión de sensores para el modelo de un robot móvil de tipo carro comprobándose en última instancia el buen desempeño del sistema.

Los resultados obtenidos durante el desarrollo de las investigaciones muestran que el sistema de navegación diseñado es perfectamente aplicable en situaciones reales y puede adaptarse a diversos escenarios.

Bibliografía

- [1] Roland Siegwart, Illah R. Nourbakhsh, y Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. Massachusetts Institute of Technology, second edition, 2011.
- [2] Spyros G. Tzafestas. *Introduction to Mobile Robot Control*. Elsevier, Oxford, 2014.
- [3] A. Lazanas y J. C. Latombe. Landmark robot navigation. *Proceedings of the Tenth National Conference on AI*, 1992.
- [4] H. B. Mitchell. *Multi-Sensor Data Fusion: An Introduction*. Springer, 2007.
- [5] Wei Li. A hybrid neuro-fuzzy system for sensor based robot navigation in unknown environments. *Proceedings of the American Control Conference*, June 1995.
- [6] Wei Li, Chenyu Ma, y F. M. Wahl. A neuro-fuzzy system architecture for behavior-based control of a mobile robot in unknown environments. *Fuzzy Sets and Systems*, 87:133–140, 1997.
- [7] Petru Rusu, Emil M. Petru, Thom E. Whalen, Auel Cornell, y Hans J. W. Spoelder. Behavior-based neuro-fuzzy concontrol for mobile robot navigation. *IEEE Instrumentation and Measurement Tecnology Conf.*, pages 1617–1622, May 2002.
- [8] Shakida Khatoon y Sofía Khatoon. Behavior coordination of autonomous mobile robot navigation by neuro-fuzzy system. *Proccedings of the IEEE 31st Annual Northeast Bioengineering Conference*, pages 56–60, April 2005.
- [9] Kai-Tai Song y Jean-Yuan Lin. Behavior fusion of robot navigation using a fuzzy neural network. *2006 IEEE International Confrence on Systems, Man, and Cybernetics*, pages 4910–4915, Oct 2006.
- [10] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer, 2011.
- [11] Anthony Mandow, Jorge L. Martinez, Jesús Morales, y José L. Blanco. Experimental kinematic for wheeled skid-steer mobile robots. *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1222–1227, 2007.

- [12] mobilerobots. *Pioneer P3-AT Datasheet*, 09366-p3at rev. a edition, 2011.
- [13] MobileRobots. *Pioneer 3 Operations Manual with MobileRobots Exclusive Advanced Robot Control & Operations Software*, version 3 edition, January 2006.
- [14] Antonio Moran Cardenas, Javier G. Razuri, David Sundgren, y Rahim Rahmani. Autonomous motion of mobile robot using fuzzy-neural networks. *12th Mexican International Conference on Artificial Intelligence*, 2013.
- [15] J.-S.R Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems Man and Cybernetics*, 23:665–685, 1993.
- [16] J. Borenstein y Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7:278–288, 1991.
- [17] Johann Borenstein y Yorem Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems Man and Cybernetics*, 19:1179–1187, 1989.
- [18] Greg Welch y Gary Bishop. An introduction to kalman filter. Technical report, University of North Carolina at Chapell Hill, 2006.
- [19] Talat Ozyagcilar. Implementing a tilt-compensated ecompass using accelerometer and magnetometer sensors. Technical report, Freescale Semiconductor, 2012.
- [20] A. Ollero-Baturone. *Robótica Manipuladores y Robots Móviles*. Alfaomega, 2007.
- [21] J. Borenstein, H.R. Everett, y L. Feng. *Where am I? Sensors and Methods for Mobile Robot Positioning*. University of Michigan, 1996.
- [22] T.S. Levitt y D.T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence*, 44:305–360, 1990.
- [23] M.M. Rashid, Nor Saidahbt Mohd Zain, y Faridatunbinti Mohammad Zain. Development of omni directional mobile robot navigation system using rfid for multiple object. *ASSRI Procedia*, 3:474–480, 2012.
- [24] Matthias O. Franz y Hanspeter A. Mallot. Biomimetic robot navigation. *Robotics and Autonomous Systems*, 30:133–153, 2000.
- [25] C.R. Gallistel. *The Organization of Learning*, MIT Press. MIT Press, 1993.
- [26] David Filliat y J.-A. Meyer. Map-based navigation in mobile robots: I. a review of localization strategies. *Cognitive Systems Research*, 4:243–282, 2003.
- [27] David Filliat y J.-A. Meyer. Map-based navigation in mobile robots: li. a review of map-learning and path-planning strategies. *Cognitive Systems Research*, 4:283–317, 2003.

- [28] Mohammad Abdel Kareem Jaradat, Mohammad Al-Rousan, y Lara Quadan. Reinforcement based mobile robot navigation in dynamic environment. *Robotics and Computer-Integrated Manufacturing*, 27:135–149, 2011.
- [29] Wang Yaonan, Yang Yimin, Yuan Xiaofang, Zuo Yi, Zhou Yuanli, y Tan Lei Yin Feng. Autonomous mobile robot navigation system designed in dynamic environment based on transferable belief model. *Measurement*, 44:1389–1405, 2011.
- [30] J. C. Latombe. *Robot Motion Planning*. Kluwer Academics, 1991.
- [31] H. P. Moravec. High resolution maps from wide angle sonar. *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, 3:116–121, March 1985.
- [32] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulz, y D. Schulz. Minerva: A second generation mobile guide robot. *Proceedings of the IEEE international conference on robotics and automation (ICRA 1999)*, 1999.
- [33] B. Yamauchi, A. Shultz, y W. Adams. Integrating exploration and localization for mobile robots. *Adaptive Behavior*, 7:217–230, 1999.
- [34] S. Simhon y G. Dudek. A global topological map formed by local metric maps. *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 1998.
- [35] B. J. Kuipers y Y. T. Byun. A robot exploration and mapping strategy based on semantic hierarchy of spatial representations. *Robotics and Autonomous Systems*, 8:47–63, 1991.
- [36] W. Burgard, A. Derr, y D. FoxD. Fox. Cremers. Intelligent global position estimation and position tracking for mobile robots. *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 98)*, 1998.
- [37] Dieter Fox, Wolfram Burgard, y Sebastian Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.
- [38] K. O. Arras, J. A. Castellanos, y R. Siegwart. Feature-based multi-hypothesis localization and tracking for mobile robots using geometric constraints. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2002)*, 2002.
- [39] M. Montemerlo, S. Thrun. Koeller, y B. Wegbreit. Fastslam: A factores solution to the simultaneous localization and mapping problem. *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.
- [40] A. Papoulis. *Probability, Random Variables, and Sthocastic Processes*. McGraw-Hill, 4th edition edition, 2011.

- [41] S. Thrun, W. Burgard, y D. Fox. *Probabilistic Robotics*. Cambridge, 2005.
- [42] P. Cheeseman y P. Smith. On the representation and estimation of spatial uncertainty. *International Journal of Robotics*, 5:56–68, 1986.
- [43] P. Smith y P. Cheeseman. On the representation and estimation of spacial uncertainty. *International Journal of Robotics Research*, 4:56–68, 1986.
- [44] P. Moutarlier y R. Chatila. An experimental system for incremental environment modmodel by an autonomous mobile robot. *1st International Symposium on Experimental Robotics*, 1989.
- [45] P. Moutarlier y R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modeling. *5th International Symposium on Robotics Research*, 1989.
- [46] N. Metropolis y S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.
- [47] C. R. Rao. Information and accuracy obtainable in estimation of statistical parameters. *Bulletin of Calcutta Mathematical Society*, 37:81–91, 1945.
- [48] D. Blackwell. Conditional expectation and unbiased sequential estimation. *Annals of Mathematical Statistics*, 18:105–110, 1947.
- [49] K. Murphy y S. Rusell. Rao-blackwellized particle filtering for dynamic bayesian networks. In A. Doucet, N. Freitas, y N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pages 499–516. Springer, 2001.
- [50] M. Cummins y P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Reseach*, 27:647–665, 2008.
- [51] M. Cummins y P. Newman. Highly scalable appearance-only slam-fab-map 2.0. *Robotics Sciences and Systems*, June 2009.
- [52] M. Pivtoraiko, R. Knepper, y A. Kelly. Differentially constrained mobile robot. *Journal of Field Robotics*, 26:308–333, 2009.
- [53] E. W. Dijkstra. A note on two pproblem in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [54] M. Likachev, D. Ferguson, y G. Gordon. Ara*: Anytime a* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*, 2003.
- [55] Valentino Braitenberg. *Vehicle: Experiments in Synthetic Psychology*. MIT Press, 1984.
- [56] Osussama Khatib. Potential field approach and operational space formulation in robot control. pages 367–377, 1986.

- [57] Y. Koren y J. Borenstein. High-speed obstacle avoidance for mobile robotics. *Proceedings of the IEEE Symposium on Intelligent Control*, pages 382–384, August 1998.
- [58] Y. Wang y G. S. Chirikjian. A new potential field method for robot path planning. *Proceeding of 2000 IEEE conference robotics and automation*, pages 977–982, April 2000.
- [59] M. Khatib y R. Chatila. An extended potential field approach for mobile robot sensor motion. *Proceedings of the Intelligent Autonomous Systems IAS-4*, pages 490–496, March 1995.
- [60] H. J. S. Feder y J-J. E. Slotine. Real-time path planning using harmonic potential in dynamic environments. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1:874–881, April 1997.
- [61] V. Lumelsky y T. Skewis. Incorporating range sensing in the robot navigation. *IEEE Transactions on Systems, Man and Cybernetics*, 20:1058–1068, 1990.
- [62] I. Ulrich y J. Borenstein. Vfh+: reliable obstacle avoidance for fast mobile robots. *Proceedings of the IEEE international conference on robotics and automation*, 2:1572–1577, May 1998.
- [63] I. Ulrich y J. Borenstein. Vfh*: Local obstacle avoidance with look-ahead verification. *Proceedings of the IEEE International Conference on Robotics and Automation*, 3:2505–2511, 2000.
- [64] S. Quinlan y O. Khatib. Elastic bands: connecting path planning and control. *Proceedings of 2003 IEEE conference robotics and automation*, 2:802–807, 1993.
- [65] R. Simmons. The curvature velocity method for local avoidance. *Proceeding of the 1996 IEEE conference robotics and automation*, 4:3375–3382, 1996.
- [66] N. Y. Ko y R. G. Simmons. The lane-curvature method for local obstacle avoidance. *Proceedings of the 1998 IEEE international conference on robotics and automation*, 3:1615–1621, Oct 1998.
- [67] D. Fox, W. Burgard, y S. Thrun. The dynamic windows approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4:23–33, 1997.
- [68] C. Schlegel. Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot. *Proceeding of 1998 IEEE/RSJ International conference on Intelligent robotics and automation*, 1:594–599, 1998.
- [69] K. Konolige. A gradient method for realtime robot control. *Proceeding of the 2000 IEEE/RSJ International Conference on Intelligent Robotics and Systems (IROS 2000)*, 1:639–646, 2000.

- [70] Moshe Kam, Xiaoxun Zhu, y Paul Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85:108–119, Jan 1997.
- [71] Lang Hong y Gwo-Jieh Wang. Integrating multisensor noisy and fuzzy data. *Proceedings of the 1994 IEEE International Conference on Multisensor and Integration for Intelligent Systems (MFI '94)*, pages 199–206, Oct 1994.
- [72] Ingemar J. Cox. Blanche- an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7:193–204, 1997.
- [73] James L. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, 2:674–680, 1989.
- [74] C. Tarin Sauer, H Brugger, E. P. Hofer, y B. Tibken. Odometry error correction by sensor fusion for autonomous mobile robot navigation. *Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference*, 3:1654–1658, 2001.
- [75] Fernando Lizarralde, Eduardo V. L. Nunes, y Liu Hsu. Mobile robot navigation using sensor fusion. *Proceedings of the 2003 IEEE international conference on robotics and automation*, pages 458–463, 2003.
- [76] B. S. Y. Rao, H. F. Durrant-Whyte, y J. A. Sheen. A fully decentralized multi-sensor system for tracking and surveillance. *International Journal of Robotics Research*, 12:20–44, 1993.
- [77] Ehsan Asadi y Mohammad Bozorg. A decentralized architecture for simultaneous localization and mapping. *IEEE/ASME Transactions on Mechatronics*, 14:64–71, 2009.
- [78] A. M. Flynn. Combinig sonar and infrared sensors for mobile robot navigation. *International Journal of Robotics Research*, 7:54–64, 1988.
- [79] Guillermo Enriquez, Sunhong Park, y Shuji Hashimoto. Wireless sensor network and rfid sensor fusion for mobile robots navigation. *Proceedings of the 2010 IEEE International Conference on Robotics and Biomedics*, pages 1752–1756, 2010.
- [80] Ronald C. Arkin y Robin R. Murphy. Autonomous navigation in a manufacturing environment. *IEEE Transactions on Robotics and Automation*, 6:445–454, 1990.
- [81] M. D. Adams, Huosheng Hu, y P. J. Probert. Towards a real-time architecture for obstacle avoidance and path planning in mobile robots. *Proceeding of 1990 IEEE International conference on Intelligent robotics and automation*, 1:584–589, 1990.

- [82] Kai-Tai Song y Liang-Hwang Sheen. Heuristic fuzzy-neuro network and its application to reactive navigation of a mobile robot. *Sets, Fuzzy and Systems*, 110:331–340, 2000.
- [83] J. W. Hauser y C. N. Purdy. Sensor data processing using genetic algorithms. *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems*, 3:1112–1115, 2000.
- [84] S. Nagata, M. Sekiguchi, y K. Asakawa. Mobile robot control by a structured hierarchical neural network. *IEEE Control Systems Magazine*, 10:69–76, 1990.
- [85] L. A. Zadeh. Fuzzy algorithms. *Information and Control*, 12:94–102, 1968.
- [86] M. Sugeno y M. Nishida. Fuzzy control of model car. *Fuzzy Sets and Systems*, 16:103–113, 1985.
- [87] Tomoyoshi Takeuchi y Yutaka Nagai. Fuzzy control of a mobile robot for obstacle avoidance. *Information Sciences*, 45:231–248, 1988.
- [88] G. Bayar, E. Ilhan Konukseven, y A. Bugra Koku. Control of a differentially driven mobile robot using radial basis function based networks. *Mechanical Engineering Department*, 3:1002–1013, 2008.
- [89] R. Fierro y F. L. Lewis. Control of a nonholonomic mobile robot using neural network. *IEEE Transactions Neural Networks*, 9:589–600, 1998.
- [90] Omid Mohareri, Rached Dhaouadi, y Ahmad B. Rad. Indirect adaptive tracking control of a mobile robot via neural networks. *Neurocomputing*, 88:54–66, 2012.
- [91] J. Ye. Tracking control for nonholonomic mobile robots: Integrating the analog neural network into a backstepping technique. *Neurocomputing*, 71:3373–3378, 2007.
- [92] Samarjit Kar, Sujit Das, y Pijush Kanti Ghosh. Applications of neuro fuzzy system: A brief review and future. *Applied Soft Computing*, 15:243–259, 2014.
- [93] Jelena Godjevac y Nigel Steele. Neuro-fuzzy control of a mobile robot. *Neurocomputing*, 28:127–143, 1999.
- [94] Nian Zhang, Daryl Beetner, Donald C. Wunsch II, Brian Hemmelman, y Abul Hasan. An embedded real-time neuro-fuzzy control for mobile robot navigation. *The 2005 IEEE International Conference on Fuzzy Systems*, pages 319–324, 2005.
- [95] J. Austin. A review of ram based neural networks. *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 26–28, 1994.

- [96] Siti Nurmani, Siti Zaiton, y Dayang Norhayati. An embedded interval type-2 neuro-fuzzy controller for mobile robot navigation. *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4315–4321, 2009.
- [97] Maulin M. Joshi y Mukesh A. Zaveri. Neuro-fuzzy based autonomous mobile robot navigation system. *11th Inf. Conf. Control, Automation, Robotics and Vision*, 2010.
- [98] H. Zerfa y W. Nouibat. Fuzzy reactive navigation for autonomous mobile robot with an offline adaptive neuro fuzzy system. *Proceedings of the 3rd International Conference on Systems*, pages 950–955, Oct 2013.
- [99] P. F. Muir y C. P. Neuman. *KineRobot modeling of wheeled mobile robots*. Carnegie-Mellon University, 1986.
- [100] J. Alexander y J. Maddocks. On the kinematics of mobile wheeled robots. In *Autonomous Robot Vehicles*. Springer New York, 1990.
- [101] Jingyang Ji, Hongpeng Wang, Junjie Zhang, Dezhen Song, Suhada Jayasuriya, y Jingtai Liu. Kinematic modeling and analysis of skid-steered mobile robots with applications to low-cost inertial-measurement-unit-based motion estimation. *IEEE Transactions on Robotics*, 25:1087–1097, 2009.
- [102] Yao Wu, Tianmiao Wang, Jianhong Liang, qiteng Zhao, Xingbang Yang, y Chenhao Han. Experimental kinematic modeling estimation for wheeled skid-steering mobile robots. *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 268–273, 2013.
- [103] J. L. Martinez, A. Mandow, J. Morales, S. Pedraza, y A. García-Cerezo. Approximating kinematics for tracked mobile robots. *The International Journal of Robotics Research*, 24:867–878, 2005.
- [104] Daniel Oliva Sales, Daniel Feitosa, Fernando Santos Osório, y Denis Fernando Wolf. Multi-agent autonomous patrolling system using ann and fsm control. *2012 Second Brazilian Conference on Critical Embedded Systems*, pages 48–53, 2012.
- [105] Rigoberto Lopez-Padilla, Victor Ayala Ramirez, y Raul E. Sanchez-Yanez. Some experiments on reactive obstacle avoidance for a mobile robot. *18th International Conference on Electronics, Communications and Computers*, pages 193–196, 2008.
- [106] Diogo Santos Ortiz Correa, Diego Fernando Sciotti, Marcos Gomes Prado, Daniel Oliva Sales, Denis Fernando Wolf, y Fernando Santos Osório. Mobile robots navigation in indoor environments using kinect sensor. *2012 Second Brazilian Conference on Critical Embedded Systems*, pages 36–41, 2012.

- [107] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3:249–265, 1987.
- [108] Sushmita Mitra y Yoichi Hayashi. Neuro-fuzzy generation: Survey in soft computing framework. *IEEE Transactions on Neural Networks*, 11:748–768, 2000.
- [109] Ajoy K. Palit y Dobrivoje Popovic. *Computational Intelligence in Time Series Forecasting*. Springer-Verlag London, 2005.
- [110] James M. Keller y Douglas J. Hunt. Incorporating fuzzy membership functions into the perceptron algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 7:693–699, 1985.
- [111] Sankar K. Pal y Sushmita Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions Neural Networks*, 3:683–697, 1992.
- [112] Hisao Ishibuchi y Hideo Tanaka. Interpolation of fuzzy if-then rules by neural networks. *International Journal of Approximate Reasoning*, 10:3–27, 1994.
- [113] B. Chen y L. L. Hoberock. A fuzzy neural network architecture for fuzzy control and classification. *IEEE International Conference on Neural Networks*, 2:1168–1173, 1996.
- [114] Robert Fullér. *Neural Fuzzy Systems*. Abo Akademy University, 1995.
- [115] James M. Keller y Hossein Tahani. Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks. *International Journal of Approximate Reasoning*, 6:221–240, 1992.
- [116] Hideyuki Takagi, Noriyuki Suzuki, y Yoshihiro Kojima. Neural networks designed on approximate reasoning architecture and their applications. *IEEE Transactions Neural Networks*, 3:752–760, 1992.
- [117] E. Tsao y J. C. Benzdek. Fuzzy kohonen clustering networks. *IEEE International Conference on Fuzzy Systems*, pages 1035–1043, 1992.
- [118] Veerachai Malyavej, Kumkeaw Warapon, y Manop Aorpimai. Indoor robot localization by rssi/imu sensor fusion. *10th Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 1–6, 2013.
- [119] The player project. internet:<http://playerstage.sourceforge.net/>. Revisado en línea en Setiembre del 2014.
- [120] Brian P. Gerkey, Richard T. Vaughan, Kasper Stoy, Andrew Howard, Gaurav S. Sukhatme, y Maja J. Mataric. Most valuable player: A robot device server for distributed control. *Proceeding of the 2001 IEEE International Conference on Intelligent Robots and Systems*, pages 1226–1231, 2001.

- [121] A. Benini, A. Mancini, A. Marinelli, y S. Longhi. A biased extended kalman filter for indoor localization of a mobile agent using low-cost imu and uwb wireless sensor network. *10th IFAC Symposium on Robot Control*, 10:735–740, 2012.
- [122] Talat Ozyagcilar. Calibrating an ecompass in the presence of hard and soft-iron interference. Technical report, Freescale Semiconductor, 2013.
- [123] Yan Xia Liu, Xi Sheng Li, Xiao Juan Zhang, y Yi Bo Feng. Novel calibration algorithm for a three-axis strapdown magnetometer. *Sensors* 2014, 14:8485–8504, 2014.
- [124] Rui Zhang, Amir Bannoura, y Fabian Höflinger. Indoor localization using a smart phone. *2013 IEEE Sensors Applications Symposium*, pages 38–42, 2013.
- [125] Anis Koubaa, Maissa Ben Jamaa, y Amjaad AlHaqbani. An empirical analysis of the impact of rss to distance mapping on localization in wsns. *Third Conference on Communications and Networking*, pages 1–7, 2012.
- [126] Yasmine Kheira y Karim Abed-Meraim. Divided difference kalman filter for indoor mobile localization. *2013 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–8, 2013.
- [127] R. Zen, G. Wang, y H. Li. Modelling and simulation of rayleigh fading, path loss, and shadowing fading for wireless mobile networks. *Simulation Modelling Practice and Theory*, 19:626–637, 2011.
- [128] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of BasicEngineering*, pages 33–45, 1960.
- [129] Thi Thanh Van Nguyen, Manh Duong Phung, Thuan Hoang Tran, y Quang Vinh Tran. Mobile robot localization using fuzzy neural network based extended kalman filter. *2012 IEEE International Conference on Control System, Computing and Engineering*, pages 416–421, 2012.

Apéndice A

Programas Usados para la Simulación

A.1. Simulación en Matlab

Listing 1 – Controlador en Matlab para la navegación.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controlador para la navegación autónoma
% de un robot móvil tipo carro.
% Autor : Luis Enciso
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xx yy PP count hist dd vv fn fe fo accumModo] = nfNavigation2(obj)
handle1 = guidata(obj);
Xenviron = handle1.user.Xobstacle;
Yenviron = handle1.user.Yobstacle;
rectNum = 1;
for i = 1:handle1.user.nc
    for j = 1:length(handle1.user.Xobstacle{i,1})-1
        rectas(rectNum,:) = [handle1.user.Xobstacle{i,1}(j+1), ...
                             handle1.user.Xobstacle{i,1}(j), ...
                             handle1.user.Yobstacle{i,1}(j+1), ...
                             handle1.user.Yobstacle{i,1}(j)];
        rectNum = rectNum+1;
    end
end
% Input values
xini = handle1.user.xini;
yini = handle1.user.yini;
Pini = handle1.user.Pini;
xodeseado = handle1.user.xdes;
yodeseado = handle1.user.ydes;
countmax = handle1.user.count;
text(xini-1,yini-1,'START','Color','black');
text(xodeseado-1,yodeseado-1,'META','Color','black');
zp = xodeseado;
wp = yodeseado;
plot(handle1.AX_Recorrido,zp,wp,'+r');
zp = xini;
wp = yini;
plot(handle1.AX_Recorrido,zp,wp,'og');
controllerGS = nfGoalSeeking();
controllerOA = nfObstacleAvoidance();
controllerWFD = nfWallFollowRight();
controllerWFL = nfWallFollowLeft();

```



```

controllerWF = nfWallFollowing();
controllerFB = nfFusionBehavior();
x = xini; P = Pini; y = yini;
r = 0.02; L = 0.268; dcenter = 0.134; dradio = 0.25;
container = zeros(401,401);
hist = frame(container,201,201);
setWindow(hist,33,33,201,201);
sumdeltax = 0; sumdeltay = 0;
xant = xini; yant = yini;
xresto = 0; yresto = 0; cond = 0;
activarCuenta = 0; Paccum = 0; esVuelta = 0;
dmin = sqrt((xodeseado-xini)^2+(yodeseado-yini)^2);
countPunto = 1; punto = [xini yini];
Pangle = 0; modo_aux = 0; K = 3;
countRetro = 10; activarRetro = 0;
for count = 1:countmax
    % Sonars (assuming 16 in total)
    asonar = [ 90 50 30 10 -10 -30 -50 -90 -130 -150 -170 170 150 130 ];
    for isonar = 1:14
        xsonar(isonar,1) = x + dcenter*cos(P*pi/180) +
            dradio*cos((P-asonar(isonar))*pi/180);
        ysonar(isonar,1) = y + dcenter*sin(P*pi/180) +
            dradio*sin((P-asonar(isonar))*pi/180);
        mes(isonar,1) = sonar(xsonar(isonar,1),ysonar(isonar,1),pi*P/180,
            asonar(isonar)*pi/180,rectas);
    end
    mmes(count,:) = mes';
    % Histogram update
    for isonar=1:14
        dmes = mes(isonar);
        if dmes <= 3.3
            angle = asonar(isonar);
            col = (dmes*cos((P-angle)*pi/180)+xsonar(isonar,1)-x+xresto)/0.1;
            if col>=0
                col = floor(col);
            else
                col = ceil(col);
            end
            row = (dmes*sin((P-angle)*pi/180)+ysonar(isonar,1)-y+yresto)/0.1;
            if row>=0
                row = floor(row);
            else
                row = ceil(row);
            end
            row = hist.ycont + row;
            col = hist.xcont + col;
            if (row<402) && (row>0) && (col<402) && (col>0)
                hist.contenedor(row,col) = hist.contenedor(row,col)+1;
                if hist.contenedor(row,col)>4
                    hist.contenedor(row,col) = 4;
                end
            end
        end
        setWindow(hist,33,33,hist.xcont,hist.ycont);
    end
end
end
xdeseado = xodeseado;
ydeseado = yodeseado;
deltax = xodeseado-x;
deltay = yodeseado-y;
Pdeseado = 180*atan2(deltay,deltax)/pi;
[fnor1 foes1 fest1 fnorO fnorE foesN festN activarRetro] =

```

```

        getForceQuadrant(hist,P, activarRetro);
fn(count,1) = fnor1; fe(count,1) = fest1; fo(count,1) = foest1;
if (activarRetro==1)&&(countRetro>10);
    countRetro = 1;
end
% Seleccionando el modo a usar
if (modo == 2)
    dPhi1 = Pdeseado-P;
elseif (modo == 1)
    dPhi1 = getValue(fest1+fnorE, controllerWFI);
elseif (modo == -1)
    dPhi1 = getValue(foes1+fnorO, controllerWFD);
elseif (modo == 0)
    if (abs(foesN+fnorO-festN-fnorE) > 0.5) || (abs(dPhi1)<15)
        [ dPhi1 dvel ] = getValue(P,Pdeseado,fnor1,foesN+fnorO,
            festN+fnorE, controllerOA);
    end
end
if countRetro < 10
    dvel = -dvel;
    dPhi1 = -dPhi1;
    countRetro = countRetro + 1;
else
    activarRetro = 0;
end
accumModo(count,1) = modo;
if dPhi1 > 180
    dPhi1 = dPhi1-360;
end
if dPhi1 < -180
    dPhi1 = dPhi1+360;
end
fnorE = 0;
fnorO = 0;
xdeseado = x + 2*(1)*cos((P+dPhi1)*pi/180);
ydeseado = y + 2*(1)*sin((P+dPhi1)*pi/180);
fest_ant = fest1+fnorE; foes_ant = foes1+fnorO;
DDF(count,1) = dPhi1; % Definition of the Goal for the go-to controller
xxdeseado(count,1) = xdeseado;
yydeseado(count,1) = ydeseado;
deltax = xdeseado-x;
deltay = ydeseado-y;
Pdeseado = 180*atan2(deltay, deltax)/pi;
xnuevo = x - xdeseado;
Pnuevo = P - Pdeseado;
if( Pnuevo > 180 )
    Pnuevo = Pnuevo - 360;
end
if( Pnuevo < -180 )
    Pnuevo = Pnuevo + 360;
end
PPdeseado(count,1) = Pdeseado;
PPnuevo(count,1) = Pnuevo;
if ~modo_aux
    modo_ant = modo;
end
% Go-to controller
DxG = getValue(Pnuevo, controllerGS);
if( DxG > 30 )
    DxG = 30;
end

```

```

if ( DxG < -30 )
    DxG = -30;
end
DxV = 1*dvel;
xx(count,1) = x;
yy(count,1) = y;
PP(count,1) = P;
dd(count,1) = DxG;
vv(count,1) = DxV;
P_ant = P;
r = 0.1*DxV;
P = P + r/L*tan(DxG*pi/180) * 180/pi;      %Acumulador de diferencias en orientación
Paccum = Paccum + (P - P_ant);
PPaccum(count,1) = Paccum;      %Ajustando ángulo a rango
if ( P > 360 )
    P = P - 360;
end
if ( P < -360 )
    P = P + 360;
end
x = x + r * cos( (pi/180)*P);
y = y + r * sin( (pi/180)*P);
if ( abs(y - yodeseado) < 0.1 && abs(x - xodeseado)<0.1 )
    break;
end
% Ajuste de celdas histograma
xdelta = x - xant + xresto;
ydelta = y - yant + yresto;
if xdelta>=0
    kxdelta = floor(xdelta/0.1);
    xresto = mod(xdelta,0.1);
else
    kxdelta = ceil(xdelta/0.1);
    xresto = rem(xdelta,0.1);
end
if ydelta>=0
    kydelta = floor(ydelta/0.1);
    yresto = mod(ydelta,0.1);
else
    kydelta = ceil(ydelta/0.1);
    yresto = rem(ydelta,0.1);
end
end
hist.xcont = hist.xcont + kxdelta;
hist.ycont = hist.ycont + kydelta;
setWindow(hist,33,33,hist.xcont,hist.ycont);
sumdeltax = sumdeltax + kxdelta;
sumdeltay = sumdeltay + kydelta;
if (abs(sumdeltax)>=173)|| (abs(sumdeltay)>=173)
    traslacion(hist,hist.ycont,hist.xcont);
    setWindow(hist,33,33,201,201);
    sumdeltax = 0;
    sumdeltay = 0;
    hist.xcont = 201;
    hist.ycont = 201;
end
xant = x;
yant = y;
L = 0.5;
A = 0.5;
if (count==1)|| (mod(count,5)==0)
    xz = x;

```

```

yz = y;
Pz = (pi/180) * P;
x1 = xz + (A/2)*sin(Pz);
y1 = yz - (A/2)*cos(Pz);
x2 = xz - (A/2)*sin(Pz);
y2 = yz + (A/2)*cos(Pz);
xF = xz + (L)*cos(Pz);
yF = yz + (L)*sin(Pz);
x3 = xF - (A/2)*sin(Pz);
y3 = yF + (A/2)*cos(Pz);
x4 = xF + (A/2)*sin(Pz);
y4 = yF - (A/2)*cos(Pz);
xc = [ x1 x2 x3 x4 x1 ]';
yc = [ y1 y2 y3 y4 y1 ]';
plot(handle1.AX_Recorrido,xc,yc, 'Color', handle1.user.colorUI);

end
guidata(obj, handle1);
drawnow;
end

```

A.2. Simulación en Stage

```

#define USAGE \
    "USAGE: _sonarobstacleavoid_[-h_<host>]_[-p_<port>]_[-m]\n" \
    "        _h_<host>:_connect_to_Player_on_this_host\n" \
    "        _p_<port>:_connect_to_Player_on_this_TCP_port\n" \
    "        _m_:turn_on_motors_(be_CAREFUL!)"
#define PI 3.14159265
typedef struct {
    double opDphi;
    double opDv;
} OPcontroller;
typedef struct {
    float fnor;
    float fnorE;
    float fnorO;
    float foes;
    float foesN;
    float fest;
    float festN;
} Forces;
#include <libplayerc++/playerc++.h>
#include <iostream>
#include <math.h>
#include "nfFusionBehavior.h"
#include "nfWallFollowLeft.h"
#include "nfWallFollowRight.h"
#include "nfGoalSeeking.h"
#include "nfObstacleAvoidance.h"
#include "args.h"
#include "frame.h"
using namespace std;
static float dsonar[] = { 0.20026, 0.22674, 0.24035, 0.24648, 0.24648, 0.24035, 0.22674, 0.20026,
0.19807, 0.22334, 0.23658, 0.24251, 0.24251, 0.23658, 0.22334, 0.19807 };
static float asonar[] = { 47.53, 35.17, 21.32, 6.99, -6.99, -21.32, -35.17, -47.53,
-131.82, -144.23, -158.33, -172.9, 172.9, 158.33, 144.23, 131.82 };
static float sang[] = {90,50,30,10,-10,-30,-50,-90,-90,-130,-150,-170,170,150,130,90};

```

```
int main(int argc, char **argv) {
    int numSonar=16;
    float xact, yact, angle, xant=-5, yant=-5;
    // COORDENADAS DESEADAS
    float xdes = 5, ydes = 5;
    float Pact, Pdes, Pn, xn, yn, fn, fe, fo;
    float xresto=0, yresto=0, kxdelta, kydelta;
    float modo, modo_ant=1;
    float dphi, dvel, opFin;
    OPcontroller op;
    Forces groupForces;
    char avoid;
    nfObstacleAvoidance controllerOA;
    nfWallFollowLeft controllerWFL;
    nfWallFollowRight controllerWFD;
    nfGoalSeeking controllerGS;
    nfFusionBehavior controllerFB;
    frame gridRobot;
    gridRobot.setXYcont(201,201);
    // setting initial position in grid
    parse_args(argc,argv);
    // we throw exceptions on creation if we fail
    try {
        using namespace PlayerCc;
        PlayerClient robot (gHostname, gPort);
        Position2dProxy pp (&robot, gIndex);
        SonarProxy sp (&robot, gIndex);
        pp.SetMotorEnable (true);
        // go into read-think-act loop
        double newspeed = 0.0f, newturnrate = 0.0f;
        for (;;)
        {
            /* this blocks until new data comes; 10Hz by default */
            robot.Read();
            avoid = 1;
            newspeed = 0.200;
            xact = pp.GetXPos();
            yact = pp.GetYPos();
            Pact = pp.GetYaw()*180/PI;
            // adjusting position in grid
            float xdelta = (xact - xant + xresto);
            kxdelta = trunc(xdelta/0.1);
            xresto = xdelta - kxdelta*0.1;
            float ydelta = (yact - yant + yresto);
            kydelta = trunc(ydelta/0.1);
            yresto = ydelta - kydelta*0.1;
            gridRobot.xcont = gridRobot.xcont + kxdelta;
            gridRobot.ycont = gridRobot.ycont + kydelta;
            xant = xact;
            yant = yact;
            for (int i = 0; i < numSonar; i++)
                //Loop through sonar
            {
                if ((sp[i])<4.0) {
                    angle = sang[i];
                    int col = gridRobot.xcont+round((sp[i]*cos((Pact+angle)*PI/180)+
                        dsonar[i]*cos((Pact+asonar[i])*PI/180)+xresto)*10);
                    int row = gridRobot.ycont+round((sp[i]*sin((Pact+angle)*PI/180)+
                        dsonar[i]*sin((Pact+asonar[i])*PI/180)+yresto)*10);
                    if ((row<401) && (row>=0) && (col<401) && (col>=0)){
                        gridRobot.Cont[row*401+col] += 1;
                    }
                }
            }
        }
    }
}
```

```

        if (gridRobot.Cont[row*401+col]>4)
            gridRobot.Cont[row*401+col] = 4;
    }
}

// Obtaining forces
groupForces = gridRobot.getForceQuadrant(Pact);
Pdes = atan2(ydes-yact,xdes-xact); //P deseado entre <-180, 180>
Pn = Pact - Pdes*180/PI;
xn = xact - xdes;
yn = yact - ydes;
if (Pn > 180)
    Pn = Pn-360;
if (Pn < -180)
    Pn = Pn+360;
modo = controllerFB.getValue(groupForces.fnor ,
    groupForces.foes ,groupForces.fest ,Pn);
modo = round(modo); //modo = -1;
if (modo == 1){
    dphi = -1*controllerWFI.getValue(groupForces.foes+groupForces.fnorO);
    dvel = 1;
}
else if (modo == -1) {
    dphi = 1*controllerWFD.getValue(groupForces.fest+groupForces.fnorE);
    dvel = 1;
}
else if (modo == 0){
    op = controllerOA.getValue(groupForces.fnor ,groupForces.foesN+
        groupForces.fnorO ,groupForces.festN+groupForces.fnorE);
    dphi = -1*op.opDphi;
    dvel = (op.opDv+1)/2;
}
else if (modo == 2){
    dphi = -1*Pn;
    dvel = 1;
}
modo_ant = modo;
newspeed = 0.1*dvel;
newturnrate = dtor(dphi);
// write commands to robot
pp.SetSpeed(2 * newspeed, newturnrate);
if (abs(xn*xn+yn*yn)<0.1){
    break;
}
}
} catch (PlayerCc::PlayerError & e) {
    std::cerr << e << std::endl;
    return -1;
}
}
}

```

A.3. Simulación de la localización de un robot móvil mediante EKF

```

%Localization using EKF and UKF
%Fusion de encoder, acelerómetro y rssi signals
clear; clc; close all;
%Generando angulo y velocidad reales

```

```

ti = 0; dt = 0.05; tf = 50;
t = ti:dt:tf; t = t'; nt = length(t);
fre = 0.05;
xo = 5*sin(2*pi*fre*t)+5;
yo = 5*cos(2*pi*fre*t)+5;
% xo = 5*sin(2*pi*fre*t);
% yo = (t.^3)/10000;
% xo = 5*ones(nt,1);
% yo = 10*ones(nt,1);
% Errores en la velocidad
w1 = randn(nt,1);
w1 = 0.05*(w1-mean(w1))/std(w1);
% error de velocidad en x
w2 = randn(nt,1);
w2 = 0.05*(w2-mean(w2))/std(w2);
% error de velocidad en y
% Medidas de velocidad locales
xvel = 10*pi*fre*cos(2*pi*fre*t) + w1;
yvel = -10*pi*fre*sin(2*pi*fre*t) + w2;
% xvel = 10*pi*fre*cos(2*pi*fre*t) + w1;
% yvel = 0.0003*t.^2 + w2;
% xvel = 0 + w1;
% yvel = 0 + w2;
% Errores en aceleración
w3 = randn(nt,1);
w3 = 0.05*(w3-mean(w3))/std(w3);
% error del acelerómetro
w4 = randn(nt,1);
w4 = 0.05*(w4-mean(w4))/std(w4);
% error del acelerómetro
% Medidas de acelerómetro
xaccel = -20*pi*pi*fre*fre*sin(2*pi*fre*t)+w3;
yaccel = -20*pi*pi*fre*fre*cos(2*pi*fre*t)+w4;
% xaccel = -20*pi*pi*fre*fre*sin(2*pi*fre*t)+w3;
% yaccel = 0.0006*t+w4;
% xaccel = 0+w3;
% yaccel = 0+w4;
xaccelruido = xaccel + 1*2*randn(nt,1);
%%
% Encoder
pulsos = 2000;
% 2000 pulsos por vuelta
dang = 2*pi/pulsos;
% Determinando angulo medido por encoder
k = 1; x_old = 0;
encoderLeft = encoder(0,2000);
encoderRight = encoder(0,2000);
% Access Points
aPoint1 = accessPoint(-25, 2, -15, -15);
aPoint2 = accessPoint(-25, 2, -15, 15);
aPoint3 = accessPoint(-25, 2, 15, -15);
aPoint4 = accessPoint(-25, 2, 15, 15);
nr1 = randn(nt,1);
nr1 = 3*(nr1-mean(nr1))/std(nr1);
nr2 = randn(nt,1);
nr2 = 3*(nr2-mean(nr2))/std(nr2);
nr3 = randn(nt,1);
nr3 = 3*(nr3-mean(nr3))/std(nr3);
nr4 = randn(nt,1);
nr5 = 3*(nr4-mean(nr4))/std(nr4);
wThe = randn(nt,1);

```



```
wThe = 0.05*(wThe-mean(wThe))/std(wThe);
% error de la orientacion
1plot(xo,yo); hold on;
plot(-15,-15,'+r') plot(-15,15,'+r') plot(15,-15,'+r') plot(15,15,'+r')
axis auto;
%Parameter inicialization for ekf and ukf
Q = [ 1e-3    0    0    0    0    0
      0 1e-3    0    0    0    0
      0    0 1e-4    0    0    0
      0    0    0 1e-4    0    0
      0    0    0    0 1e9    0
      0    0    0    0    0 1e-9 ];
R = [ 1e-4    0    0    0    0    0
      0 1e-4    0    0    0    0
      0    0 40    0    0    0
      0    0    0 40    0    0
      0    0    0    0 40    0
      0    0    0    0    0 40 ];
P_ekf = eye(6,6); P_ukf = eye(6,6);
x = zeros(6,1); u = zeros(2,1);
%Parámetros para access points
l1 = 2; x1 = -15; y1 = -15; l2 = 2; x2 = -15; y2 = 15;
l3 = 2; x3 = 15; y3 = -15; l4 = 2; x4 = 15; y4 = 15;
%ekf stimate states
x_ekf(:,1) = zeros(6,1);
% initial conditions
x_ekf(1) = 4.8593;
% position in X
x_ekf(2) = 7.0954;
% position in Y
%ukf stimate
states x_ukf(:,1) = zeros(6,1); % initial conditions
x_ukf(1) = 4.8593;
% position in X
x_ukf(2) = 7.0954;
% position in Y
%Proceso for tt = ti:dt:tf
xk = xo(k,1);
yk = yo(k,1);
% Medidas
xvel_enc(k,1) = encoderMeasure(encoderLeft,xk,dt);
yvel_enc(k,1) = encoderMeasure(encoderRight,yk,dt);
Pdb1(k,1) = apMeasure(aPoint1,xk,yk)+nr1(k);
Pdb2(k,1) = apMeasure(aPoint2,xk,yk)+nr2(k);
Pdb3(k,1) = apMeasure(aPoint3,xk,yk)+nr3(k);
Pdb4(k,1) = apMeasure(aPoint4,xk,yk)+nr4(k);
% Medida de ángulo
if k>1
    the(k,1) = atan2(yk-yo(k-1,1),xk-xo(k-1,1))+wThe(k);
else
    the(k,1) = wThe(k);
end
% Medida de velocidad global
xvelLoc(k,1) = -xvel(k,1)*sin(the(k,1)) + yvel(k,1)*cos(the(k,1));
yvelLoc(k,1) = xvel(k,1)*cos(the(k,1)) + yvel(k,1)*sin(the(k,1));
% Measurement
z = [ xvelLoc(k,1);
      yvelLoc(k,1);
      Pdb1(k,1);
      Pdb2(k,1);
      Pdb3(k,1);
```

```

Pdb4(k,1) ];
u = [ xacel(k) yacel(k) ]';
% Defining handles functions
% System nonlinear dynamic
fstate = @(x)[x(1)+dt*x(3);
              x(2)+dt*x(4);
              x(3)+dt*u(1);
              x(4)+dt*u(2);
              x(5);
              x(6)];
% measure equations
hmeas = @(x)[-x(3)*sin(the(k,1))+x(4)*cos(the(k,1))+x(5);
              x(3)*cos(the(k,1))+x(4)*sin(the(k,1))+x(6);
              -25-10*I1*log10(sqrt((x(1)-x1)^2+(x(2)-y1)^2));
              -25-10*I2*log10(sqrt((x(1)-x2)^2+(x(2)-y2)^2));
              -25-10*I3*log10(sqrt((x(1)-x3)^2+(x(2)-y3)^2));
              -25-10*I4*log10(sqrt((x(1)-x4)^2+(x(2)-y4)^2))];
% Jacobiano de ecuaciones del sistema
F = @(x)([ 1 0 dt 0 0 0
            0 1 0 dt 0 0
            0 0 1 0 0 0
            0 0 0 1 0 0
            0 0 0 0 1 0
            0 0 0 0 0 1]);
% derivada de las ecuaciones de medición
H = @(x)([ 0 , -sin(the(k,1)) , cos(the(k,1)) , 1 , 0;
            0 , cos(the(k,1)) , sin(the(k,1)) , 0 , 1;
            (-10*I1*(x(1)-x1)/(log(10)*((x(1)-x1)^2+(x(2)-y1)^2))) , ...
            (-10*I1*(x(2)-y1)/(log(10)*((x(1)-x1)^2+(x(2)-y1)^2))) , ...
            0 , 0 , 0 , 0;
            (-10*I1*(x(1)-x2)/(log(10)*((x(1)-x2)^2+(x(2)-y2)^2))) , ...
            (-10*I1*(x(2)-y2)/(log(10)*((x(1)-x2)^2+(x(2)-y2)^2))) , ...
            0 , 0 , 0 , 0;
            (-10*I1*(x(1)-x3)/(log(10)*((x(1)-x3)^2+(x(2)-y3)^2))) , ...
            (-10*I1*(x(2)-y3)/(log(10)*((x(1)-x3)^2+(x(2)-y3)^2))) , ...
            0 , 0 , 0 , 0;
            (-10*I1*(x(1)-x4)/(log(10)*((x(1)-x4)^2+(x(2)-y4)^2))) , ...
            (-10*I1*(x(2)-y4)/(log(10)*((x(1)-x4)^2+(x(2)-y4)^2))) , ...
            0 , 0 , 0 , 0]);
% Using EKF
tic;
[x_ekf(:,k+1) P_ekf] = ekf(fstate,x_ekf(:,k),P_ekf,hmeas,z,Q,R,F,H);
dt_ekf(1,k) = toc;
% Using UKF
tic;
[x_ukf(:,k+1) P_ukf] = ukf(fstate,x_ukf(:,k),P_ukf,hmeas,z,Q,R);
dt_ukf(1,k) = toc;
% Plotting estimated points %
plot(x_ekf(1,k),x_ekf(2,k),'+r');
% x_ukf(1,k),x_ukf(2,k),'og');
% hold on;
% drawnow();
k= k + 1;
end
% Gráfica online de la estimación (disable)
figure(1);
% plot(xo,yo,'ob');
xlabel('x(m)'); ylabel('y(m)');
legend('Real','EKF'); % 'UKF');
% Medidas de los rssi
figure(2)

```

```

subplot(4,1,1)
plot(Pdb1)
subplot(4,1,2)
plot(Pdb2)
subplot(4,1,3)
plot(Pdb3)
subplot(4,1,4)
plot(Pdb4) %
% figure (3)
% subplot(4,1,1)
% plot(d1)
figure(4)
plot(the) %%
% Coordenadas y sus estimadas
figure(6)
subplot(2,1,1);
plot(t,x_ekf(1,2:nt+1),'-b',t,xo,'-r')
xlabel('t(s)'); ylabel('x(m)');
legend('EKF','Real');
subplot(2,1,2)
plot(t,x_ekf(2,2:nt+1),'-b',t,yo,'-r')
xlabel('t(s)'); ylabel('y(m)');
legend('EKF','Real');
figure(7)
subplot(2,1,1); plot(t,x_ukf(1,2:nt+1),'-b',t,xo,'-r')
xlabel('t(s)'); ylabel('x(m)'); legend('UKF','Real');
subplot(2,1,2) plot(t,x_ukf(2,2:nt+1),'-b',t,yo,'-r')
xlabel('t(s)'); ylabel('y(m)'); legend('UKF','Real');
% x MAE
xmae_ekf = 1/length(xo)*sum(abs(xo-x_ekf(1,2:nt+1)'));
% x RMSE
xrmse_ekf = sqrt(1/length(xo)*sum(abs(xo-x_ekf(1,2:nt+1)')));
% y MAE
ymae_ekf = 1/length(yo)*sum(abs(yo-x_ekf(2,2:nt+1)'));
% y RMSE
yrmse_ekf = sqrt(1/length(yo)*sum(abs(yo-x_ekf(2,2:nt+1)')));
% Impresión de estadísticas disp('Statistics for EKF filter: ');
disp(sprintf('xmae=%.4f',xmae_ekf));
disp(sprintf('ymae=%.4f',ymae_ekf));
disp(sprintf('xrmse=%.4f',xrmse_ekf));
disp(sprintf('yrmse=%.4f',yrmse_ekf));
% x MAE
xmae_ukf = 1/length(xo)*sum(abs(xo-x_ukf(1,2:nt+1)'));
% x RMSE xrmse_ukf = sqrt(1/length(xo)*sum(abs(xo-x_ukf(1,2:nt+1)')));
% y MAE
ymae_ukf = 1/length(yo)*sum(abs(yo-x_ukf(2,2:nt+1)'));
% y RMSE
yrmse_ukf = sqrt(1/length(yo)*sum(abs(yo-x_ukf(2,2:nt+1)')));
disp('Statistics for UKF filter: ');
disp(sprintf('xmae=%.4f',xmae_ukf));
disp(sprintf('ymae=%.4f',ymae_ukf));
disp(sprintf('xrmse=%.4f',xrmse_ukf));
disp(sprintf('yrmse=%.4f',yrmse_ukf));
%% Generating the animation
% Del tiempo se toma total y delta
It = length(tt); dt = dt;
% Se define escala para el robot
sc00 = 1;
% Se define distancias
lp1 = 0.15;
% distancia vertical 1 al centro del robot

```

```

lp2 = 0.25;
% distancia vertical 2 al centro del robot
yg1 = 0.15;
% distancia horizontal 1 al eje del robot
yg2 = 0.25;
% distancia horizontal 2 al eje del robot %Se definen funciones para dibujar el robot
Rfx00 = @(lp,yg,psi) (lp*cos(psi)+yg*sin(psi));
Rfy00 = @(lp,yg,psi) (lp*sin(psi)-yg*cos(psi));
f00 = @(x,y,psi) ([x+sc00*Rfx00(lp1,-yg2,psi)
y+sc00*Rfy00(lp1,-yg2,psi)]);
f01 = @(x,y,psi) ([x+sc00*Rfx00(lp2,-yg1,psi)
y+sc00*Rfy00(lp2,-yg1,psi)]);
f02 = @(x,y,psi) ([x+sc00*Rfx00(lp2,yg1,psi)
y+sc00*Rfy00(lp2,yg1,psi)]);
f03 = @(x,y,psi) ([x+sc00*Rfx00(lp1,yg2,psi)
y+sc00*Rfy00(lp1,yg2,psi)]);
f04 = @(x,y,psi) ([x+sc00*Rfx00(-lp1,yg2,psi)
y+sc00*Rfy00(-lp1,yg2,psi)]);
f05 = @(x,y,psi) ([x+sc00*Rfx00(-lp2,yg1,psi) y+sc00*Rfy00(-lp2,yg1,psi)]);
f06 = @(x,y,psi) ([x+sc00*Rfx00(-lp2,-yg1,psi) y+sc00*Rfy00(-lp2,-yg1,psi)]);
f07 = @(x,y,psi) ([x+sc00*Rfx00(-lp1,-yg2,psi) y+sc00*Rfy00(-lp1,-yg2,psi)]);
%Se copian los puntos a vectores
x00 = xo; y00 = yo;
psi00 = the; t00 = ti:dt:tf;
%Se generan los puntos de la animación
v00 = f00(x00,y00,psi00);
v01 = f01(x00,y00,psi00);
v02 = f02(x00,y00,psi00);
v03 = f03(x00,y00,psi00);
v04 = f04(x00,y00,psi00);
v05 = f05(x00,y00,psi00);
v06 = f06(x00,y00,psi00);
v07 = f07(x00,y00,psi00);
%Se indica la posición máxima alcanzable en la simulación
posmax = max(max(abs([v00 v01 v02 v03 v04])));
%Se definen frames por segundo y factor de velocidad de animacion
fps = 10; velsim = 10;
%Se define el tiempo discreto a utilizar en la simulacion
T = round(1:velsim/(fps*dt):lt);
%definiendo el objeto avi
animacion = avifile('localizacion2.avi');
%Se setea el fps
animacion = set(animacion, 'Fps', fps);
%%Se abre el archivo %
open(animacion);
%Se comienza a tomar el tiempo que demora
tic
%Se genera la animación sin mostrar imagenes
hFigure = figure('visible','off');
%Para cada muestra que corresponda
for Ti=1:length(xo)
    %Se dibujan las lineas del robot
    plot([v00(Ti,1) v01(Ti,1)],[v00(Ti,2) v01(Ti,2)], 'r', ...
        [v01(Ti,1) v02(Ti,1)],[v01(Ti,2) v02(Ti,2)], 'r', ...
        [v02(Ti,1) v03(Ti,1)],[v02(Ti,2) v03(Ti,2)], 'r', ...
        [v03(Ti,1) v04(Ti,1)],[v03(Ti,2) v04(Ti,2)], 'r', ...
        [v04(Ti,1) v05(Ti,1)],[v04(Ti,2) v05(Ti,2)], 'r', ...
        [v05(Ti,1) v06(Ti,1)],[v05(Ti,2) v06(Ti,2)], 'r', ...
        [v06(Ti,1) v07(Ti,1)],[v06(Ti,2) v07(Ti,2)], 'r', ...
        [v07(Ti,1) v00(Ti,1)],[v07(Ti,2) v00(Ti,2)], 'r', ...
        x_ekf(1,Ti+1), x_ekf(2,Ti+1), 'b', ...

```

```

        xo,yo,'-b' ...    );
xlabel('y'); ylabel('x');
title(strcat('Animación_de_Robot_x',num2str(velsim,'%d')));
axis([-posmax posmax -posmax posmax])
axis square;
grid on;
set(hFigure, 'PaperPositionMode', 'auto');
cdata = hardcopy(hFigure, '-Dzbuffer', '-r0');
F = getframe(hFigure);    animacion = addframe(animacion,F);
end
%Se cierra el archivo
animacion = close(animacion);
clear animacion;
%Se cierra la figura
close(hFigure);
%Se muestra cuanto tiempo tomo en correr la renderización
toc

```



Apéndice B

Programas Usados en la Implementación

B.1. Aplicación en Android

Listing 2 – Aplicación Android para el envío de datos del sensorado.

```
package com.tesis.sensorserial2;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.net.wifi.ScanResult;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.TextView;
import edu.rosehulman.me435.AccessoryActivity;
public class Serial2Activity extends AccessoryActivity implements SensorEventListener{
    TextView tvAccelerometer;    TextView tvGyro;
    TextView tvMagnetic;    TextView tvRssiWifi;
    CheckBox cbAccelerometer;    CheckBox cbGyro;
    CheckBox cbMagnetic;    CheckBox cbRssiWifi;
    RadioButton rbFastest;    RadioButton rbGame;
    RadioButton rbNormal;    RadioButton rbUI;
    DecimalFormatSymbols dfs = new DecimalFormatSymbols(Locale.GERMAN);
    DecimalFormat decimal_format = new DecimalFormat("000.0000000");
    String accelerometer_cvs_line;
```



```
String gyroscope_cvs_line;
String magnetic_field_cvs_line;
String rssi_cvs_line;
String enc_cvs_line;
Timer timer1;    TimerTask timerTask1;
Timer timer2;    TimerTask timerTask2;
WifiManager mainWifi;    WifiReceiver receiverWifi;
List<ScanResult> wifiList;
StringBuilder sb = new StringBuilder(); // for wifi string
private SensorManager sensorManager;
Random r = new Random();
int startSend = 0;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tvAccelerometer = (TextView) findViewById(R.id.textview1);
    tvGyro = (TextView) findViewById(R.id.textview2);
    tvMagnetic = (TextView) findViewById(R.id.textview3);
    tvRssiWifi = (TextView) findViewById(R.id.textview4);
    cbAccelerometer = (CheckBox) findViewById(R.id.checkAccelerometer);
    cbGyro = (CheckBox) findViewById(R.id.checkGyro);
    cbMagnetic = (CheckBox) findViewById(R.id.checkMagnetic);
    cbRssiWifi = (CheckBox) findViewById(R.id.checkRssiWifi);
    rbFastest = (RadioButton) findViewById(R.id.radioButtonFastest);
    rbGame = (RadioButton) findViewById(R.id.radioButtonGame);
    rbNormal = (RadioButton) findViewById(R.id.radioButtonNormal);
    rbUI = (RadioButton) findViewById(R.id.radioButtonUI);
    dfs.setDecimalSeparator('.');
    decimal_format.setDecimalFormatSymbols(dfs);
    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    mainWifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
    receiverWifi = new WifiReceiver();
    registerReceiver(receiverWifi,
        new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
    tvRssiWifi.setText("\n\nStarting_Scan...\n\n");
    Button start = (Button) findViewById(R.id.start1);
    start.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            // TODO Auto-generated method stub
            startSend = 1;
            startTimerRssi();
            RegisterSensorListener();
            sendCommand("START");
            sendCommand(accelerometer_cvs_line+', '+gyroscope_cvs_line+', '+
                magnetic_field_cvs_line+', '+rssi_cvs_line);
            //For first time
            startTimerSerial();
        }
    });
    Button stop = (Button) findViewById(R.id.stop1);
    stop.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO Auto-generated method stub
            startSend = 0;
            UnregisterSensorListener();

            unregisterReceiver(receiverWifi);
            /*sensorManager.registerListener(LEDActivity.this,
                sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
```



```

        SceneManager.SENSOR_DELAY_NORMAL); */
        stopTimerSerial();
        //stopTimerRssi();
    }
});
}

public void startTimerRssi() {
    //set a new Timer
    timer2 = new Timer();
    //initialize the TimerTask
    initializeRssiTask();
    //schedule the timer, after the first 1000ms
    //the TimerTask will run every 2000ms
    timer2.schedule(timerTask2, 0, 500);
}

public void stopTimerRssi() {
    if (timer2 != null) {
        timer2.cancel();
        timer2 = null;
    }
}

public void startTimerSerial() {
    //set a new Timer
    timer1 = new Timer();
    //initialize the TimerTask
    initializeTimerTask();
    //schedule the timer, after the first 1000ms
    //the TimerTask will run every 2000ms
    timer1.schedule(timerTask1, 1000, 20);
}

public void stopTimerSerial() {
    if (timer1 != null) {
        timer1.cancel();
        timer1 = null;
    }
}

public void initializeTimerTask() {
    timerTask1 = new TimerTask() {
        public void run() {
            sendCommand(accelerometer_cvs_line+', '+gyroscope_cvs_line+', '+
                magnetic_field_cvs_line+', '+rssi_cvs_line);
        }
    };
}

public void initializeRssiTask() {
    timerTask2 = new TimerTask() {
        public void run() {
            mainWifi.startScan();
        }
    };
}

public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

```

```

    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}

public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
        getMagneticField(event);
    }
    else if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        getAccelerometer(event);
    }
    else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
        getGyroscope(event);
    }
}

private void getAccelerometer(SensorEvent event) {
    Date date = new Date();
    if (cbAccelerometer.isChecked()){
        accelerometer_cvs_line = decimal_format.format(event.values[0]) + ",_,"
        + decimal_format.format(event.values[1]) + ",_,"
        + decimal_format.format(event.values[2]);
        if (startSend==1) {
            sendCommand(accelerometer_cvs_line);
            tvAccelerometer.setText(accelerometer_cvs_line);
        }
    }
}

private void getGyroscope(SensorEvent event) {
    Date date = new Date();
    if (cbGyro.isChecked()){
        gyroscope_cvs_line = decimal_format.format(event.values[0]) + ",_,"
        + decimal_format.format(event.values[1]) + ",_,"
        + decimal_format.format(event.values[2]);
        if (startSend==1) {
            sendCommand(gyroscope_cvs_line);
            tvGyro.setText(gyroscope_cvs_line);
        }
    }
}

private void getMagneticField(SensorEvent event) {
    Date date = new Date();
    if (cbMagnetic.isChecked()){
        magnetic_field_cvs_line = decimal_format.format(event.values[0]) + ",_,"
        + decimal_format.format(event.values[1]) + ",_,"
        + decimal_format.format(event.values[2]);
        if (startSend==1) {
            sendCommand(magnetic_field_cvs_line);
            tvMagnetic.setText(magnetic_field_cvs_line);
        }
    }
}

public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // TODO Auto-generated method stub
}

protected void onResume() {
    super.onResume();
    // register this class as a listener for the orientation and
    // accelerometer sensors
}

```

```

        RegisterSensorListener();
        registerReceiver(receiverWifi,
            new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
    }
    protected void onPause() {
        // unregister listener
        super.onPause();
        sensorManager.unregisterListener(this);
        unregisterReceiver(receiverWifi);
    }
    void RegisterSensorListener() {
        try {
            List<Sensor> accelerometer_sensors = sensorManager
                .getSensorList(Sensor.TYPE_ACCELEROMETER);
            sensorManager.registerListener(this, accelerometer_sensors.get(0),
                GetSensorDelay());
        } catch (IndexOutOfBoundsException e) {
            this.cbAccelerometer.setChecked(false);
            this.cbAccelerometer.setClickable(false);
            this.tvAccelerometer.setText("No_accelerometer");
        }
        try {
            List<Sensor> magnetic_field_sensors = sensorManager
                .getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
            sensorManager.registerListener(this, magnetic_field_sensors.get(0),
                GetSensorDelay());
        } catch (IndexOutOfBoundsException e) {
            this.cbMagnetic.setChecked(false);
            this.cbMagnetic.setClickable(false);
            this.tvMagnetic.setText("No_magnetic_field_sensor");
        }
        try {
            List<Sensor> orientation_sensors = sensorManager
                .getSensorList(Sensor.TYPE_GYROSCOPE);
            sensorManager.registerListener(this, orientation_sensors.get(0),
                GetSensorDelay());
        } catch (IndexOutOfBoundsException e) {
            this.cbGyro.setChecked(false);
            this.cbGyro.setClickable(false);
            this.tvGyro.setText("No_gyroscope_sensor");
        }
    }
    int GetSensorDelay() {
        int sensor_delay;
        if (rbFastest.isChecked()) {
            sensor_delay = SensorManager.SENSOR_DELAY_FASTEST;
        } else if (rbGame.isChecked()) {
            sensor_delay = SensorManager.SENSOR_DELAY_GAME;
        } else if (rbNormal.isChecked()) {
            sensor_delay = SensorManager.SENSOR_DELAY_NORMAL;
        } else if (rbUI.isChecked()) {
            sensor_delay = SensorManager.SENSOR_DELAY_UI;
        } else {
            sensor_delay = SensorManager.SENSOR_DELAY_UI;
        }
        return sensor_delay;
    }
    class WifiReceiver extends BroadcastReceiver {
        public void onReceive(Context c, Intent intent) {
            int rssi1=-127,rssi2=-127,rssi3=-127;
            String mac1 = "00:1d:0f:fa:11:ce";

```

```

String mac2 = "6e:8e:2c:99:72:87";
String mac3 = "20:73:55:27:ce:60";
int velR = r.nextInt(100);
    int velL = r.nextInt(100);
wifiList = mainWifi.getScanResults();
for(int i = 0; i < wifiList.size(); i++){
    if (wifiList.get(i).BSSID.compareTo(mac1)==0) {
        rssi1 = (wifiList.get(i)).level;
    }
    if (wifiList.get(i).BSSID.compareTo(mac2)==0) {
        rssi2 = (wifiList.get(i)).level;
    }
    if (wifiList.get(i).BSSID.compareTo(mac3)==0) {
        rssi3 = (wifiList.get(i)).level;
    }
}

rssi_cvs_line = String.valueOf(rssi1) + ',' + String.valueOf(rssi2)
                + ',' + String.valueOf(rssi3)+ ',' + String.valueOf(rssi3);
// simulation
enc_cvs_line = String.valueOf(velR) + ',' + String.valueOf(velL);
tvRssiWifi.setText(rssi_cvs_line);
}
}
private void UnregisterSensorListener() {
    sensorManager.unregisterListener(this);
}
}

```

B.2. Código para Arduino

Listing 3 – Programa para la fusión de sensores en Arduino.

```

#include <MatrixMath.h>
#include <Max3421e.h>
#include <Usb.h>
#include <AndroidAccessory.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "Kalman.h"
#include "EKF.h"
#define RESTRICT_PITCH
Kalman kalmanX, kalmanY, kalmanZ; // Create the Kalman for orientation
EKF ekfFilter(2,2,2,2,-15,-15,-15,15,15,-15,15,15); // Create the EKF for localization
/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
double magX, magY, magZ;
/* APoints and Encoder Data */
double measures[6]={0,0,0,0,0,0};
/* Estimated Angles */
double roll, pitch, yaw; // Roll and pitch
double kalAngleX, kalAngleY, kalAngleZ; // Calculated angle using a Kalman filter
double EKfposX, EKfposY; // Calculated angle using a EKF
double velEKF[2], accEKF[2]; // Velocity and Acceleration measures
int cycles = 0; int nData = 0;
char manufacturer[] = "ArduinoA";

```

```

char model[] = "Moto";
char versionStr[] = "1.0";
String inputString = "";
String inputSerial = "";
String message = "";
boolean stringComplete = false;
AndroidAccessory acc("ArduinoA",
                    "Moto",
                    "SensorSerial2",
                    "1.0",
                    "http://anythyn.blogspot.com/",
                    "12345");

char rxBuf[255];
byte txBuf[255];
int indexFin;
int wait = 1;
uint32_t timer;
double magGain[3];
void setup() {
    delay(1500); // Wait for android to be ready
    acc.powerOn();
    Serial.begin(9600);
    delay(1000); // Wait for sensors to stabilize
}
void loop() {
    if (acc.isConnected() && (wait == 1)) {
        int len = acc.read(rxBuf, sizeof(rxBuf), 1); // Wait until START is received
        if (len > 0) {
            rxBuf[len - 1] = '\0'; // Convert terminator to null character.
            String inputString = String(rxBuf);
            if (inputString.equalsIgnoreCase("START")) {
                wait = 0;
                delay(1000);
                initialize();
                timer = micros();
                Serial.println("START");
            }
        }
    }
    // Execute the process
    else if (wait == 0){
        // Get the readings from Android
        updateAndroid();
        double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
        timer = micros();
        /* Roll and pitch estimation */
        updatePitchRoll();
        double gyroXrate = gyroX; // Convert to deg/s
        double gyroYrate = gyroY; // Convert to deg/s
        // Fix the transition problem
        if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
            kalmanX.setAngle(roll);
            kalAngleX = roll;
        }
        else
            kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using the Kalman filter
        if (abs(kalAngleX) > 90)
            gyroYrate = -gyroYrate; // Invert rate, so it fits the restricted accelerometer reading
        kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
        /* Yaw estimation */
        updateYaw();
    }
}

```

```

double gyroZrate = gyroZ ;
    // Fix the transition problem
    kalAngleZ > 90) || (yaw > 90 && kalAngleZ < -90)) {
    kalmanZ.setAngle(yaw);
    kalAngleZ = yaw;
}

else
    kalAngleZ = kalmanZ.getAngle(yaw, gyroZrate, dt); // Calculate the angle using a Kalman filter
ekfFilter.setParameters(kalAngleZ*3.1416/180,measures,accX,accY,dt);
EKfposX = ekfFilter.getPosX();
EKfposY = ekfFilter.getPosY();
cycles += 1;      /* Imprimir data solo cada 0.1s*/
if (cycles == 2) {
    Serial.println(EKfposX); //Serial.print("\t");
    Serial.println(EKfposY); //Serial.print("\t");
    Serial.println(kalAngleZ); //Serial.print("\t");
}
}
}

void initialize() {
    updateAndroid();
    updatePitchRoll();
    updateYaw();
    /* Set Kalman and gyro starting angle */
    calibrateMag();
    kalmanX.setAngle(roll); // Cálculo de roll
    kalmanY.setAngle(pitch); // Cálculo del pitch
    kalmanZ.setAngle(yaw); // Calculo de yaw mediante Kalman
    timer = micros(); // inicialización del timer
}

void updateAndroid() {
    if (acc.isConnected()) {
        int len = acc.read(rxBuf, sizeof(rxBuf), 1); // Get length of data
        if (len > 0) {
            rxBuf[len - 1] = '\0'; // Convert terminator to null character.
            String inputString = String(rxBuf);
            message = inputString;
            indexFin = inputString.indexOf(','); // Medidas del acelerómetro
            accY = atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');
            accX = atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');
            accZ = -atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');
            // Medidas del giroscopio
            gyroX = atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');
            gyroY = atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');
            gyroZ = atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');
            // Medidas del magnetómetro
            magY = atof(inputString.substring(0,indexFin).c_str());
            inputString = inputString.substring(indexFin+1,inputString.length());
            indexFin = inputString.indexOf(',');

```



```

magX = atof(inputString.substring(0,indexFin).c_str());
inputString = inputString.substring(indexFin+1,inputString.length());
indexFin = inputString.indexOf(',');
magZ = atof(inputString.substring(0,indexFin).c_str());
inputString = inputString.substring(indexFin+1,inputString.length());
indexFin = inputString.indexOf(',');
measures[2] = atof(inputString.substring(0,indexFin).c_str());
inputString = inputString.substring(indexFin+1,inputString.length());
indexFin = inputString.indexOf(',');
measures[3] = atof(inputString.substring(0,indexFin).c_str());
inputString = inputString.substring(indexFin+1,inputString.length());
indexFin = inputString.indexOf(',');
measures[4] = atof(inputString.substring(0,indexFin).c_str());
inputString = inputString.substring(indexFin+1,inputString.length());
indexFin = inputString.indexOf(',');
measures[5] = atof(inputString.substring(0,indexFin).c_str());
}
}
}
void updatePitchRoll() {
    roll = atan2(accY, accZ) * RAD_TO_DEG;
    pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
}
void updateYaw() {
    //magX *= -1; // Invert axis - this it done here, as it should be done after the calibration
    //magY *= -1;
    //magZ *= -1;
    magX *= magGain[0];
    magY *= magGain[1];
    magZ *= magGain[2];
    //magX -= magOffset[0];
    //magY -= magOffset[1];
    //magZ -= magOffset[2];
    double rollAngle = kalAngleX*DEG_TO_RAD;
    double pitchAngle = kalAngleY*DEG_TO_RAD;
    double Bfy = magZ*sin(rollAngle) - magY*cos(rollAngle);
    double Bfx = magX*cos(pitchAngle) + magY*sin(pitchAngle)*sin(rollAngle)
        + magZ*sin(pitchAngle)*cos(rollAngle);
    yaw = atan2(-Bfy, Bfx) * RAD_TO_DEG;
    yaw *= -1;
}
void calibrateMag() {
    magGain[0] = 1;
    magGain[1] = 1;
    magGain[2] = 1;
    magOffset[0] = 0;
    magOffset[1] = 0;
    magOffset[2] = 0;
}
void serialEvent(){
    while (Serial.available()) {
        char inChar = (char)Serial.read();
        if (inChar == '\n') {
            if (nData==0){
                measures[0] = atof(inputSerial.c_str());
                inputSerial = "";
                nData = 1;
            }
        }
        else{
            measures[1] = atof(inputSerial.c_str());
            inputSerial = "";
        }
    }
}

```



```

        nData = 0;
    }
}
else {
    inputSerial += inChar;
}
}
}
}

```

B.3. Controlador implementado en C++

Listing 4 – Controlador implementado como una subclase ArAction.

```

#include "Aria.h"
#include <iostream>
#include "opStruct.h"
#include "ArExport.h"
#include "ariaOSDef.h"
#include "ArResolver.h"
#include "ArRobot.h"
#include "ArAction.h"
#include "frame.h"
#include "nfFusionBehavior.h"
#include "nfWallFollowing.h"
#include "nfWallFollowLeft.h"
#include "nfWallFollowRight.h"
#include "nfGoalSeeking.h"
#include "nfObstacleAvoidance.h"
// This class moves a robot toward the goal indicated by xini and yini
class NfNavigator2 : public ArAction {
public:
enum State {
    NO_TARGET,
    TARGET,
};

    //Target in view
    NfNavigator2(float, float, float, float); //Constructor
    ~NfNavigator2(void); //Destructor
    ArActionDesired *fire(ArActionDesired currentDesired);
    //Return state of action protected:
    ArActionDesired myDesired;
    nfObstacleAvoidance controllerOA;
    nfWallFollowLeft controllerWFR;
    nfWallFollowLeft controllerWFL;
    nfWallFollowRight controllerWFD;
    nfGoalSeeking controllerGS;
    nfFusionBehavior controllerFB;
    frame gridRobot;
    State myState;
    int myChannel;
    float xo, yo, xf, yf, xant, yant, xresto, yresto;
};

// Constructor
NfNavigator2::NfNavigator2(float xini, float yini, float xgoal, float ygoal)
: ArAction("controllerNF", "Moves_from_a_given_point_towards_a_goal.") {
    xo = xini;
    yo = yini;

```

```

        xf = xgoal;
        yf = ygoal;
        xant = xini;
        yant = yini;
        xresto = 0;
        yresto = 0;
        myChannel = 1;
        myState = NO_TARGET;
        gridRobot.setXYcont(101,101); // setting initial position in grid
    }
    //Destructor
    NfNavigator2::~NfNavigator2(void) {}
    // The fire method
    ArActionDesired *NfNavigator2::fire(ArActionDesired currentDesired) {
        static float dsonar[] = { 200.26, 226.74, 240.35, 246.48, 246.48, 240.35, 226.74, 200.26,
        198.07, 223.34, 236.58, 242.51, 242.51, 236.58, 223.34, 198.07 };
        static float asonar[] = { 47.53, 35.17, 21.32, 6.99, -6.99, -21.32, -35.17, -47.53,
        -131.82, -144.23, -158.33, -172.9, 172.9, 158.33, 144.23, 131.82 };
        float xact, yact;
        float Pact, Pdes, Pn, xn, yn;
        float modo;
        float dphi, dvel;
        OPcontroller op;
        Forces groupForces;
        float spos[16], sang[16];
        ArSensorReading* sonarReading;
        ArPose posOdometry;
    int numSonar = 8;
        double angle;
        float kxdelta, kydelta;
        // obtaining the position data
        // Option 1: Relative position from odometry
        posOdometry = myRobot->getPose();
        xact = posOdometry.getX()/1000;
        yact = posOdometry.getY()/1000;
        Pact = posOdometry.getTh();
        // adjusting position in grid
        float xdelta = (xact - xant + xresto);
        kxdelta = trunc(xdelta/0.1);
        xresto = xdelta - kxdelta*0.1;
        float ydelta = (yact - yant + yresto);
        kydelta = trunc(ydelta/0.1);
        yresto = ydelta - kydelta*0.1;
        gridRobot.xcont = gridRobot.xcont + kxdelta;
        gridRobot.ycont = gridRobot.ycont + kydelta;
        xant = xact;
        yant = yact;
    #if 0
        sumdeltax = sumdeltax + kxdelta;
        sumdeltay = sumdeltay + kydelta;
        if (abs(sumdeltax)>=23)||abs(sumdeltay)>=23)
            traslacion(hist, hist.ycont, hist.xcont);
            setWindow(hist,33,33,51,51);
            sumdeltax = 0;
            sumdeltay = 0;
            hist.xcont = 51;
            hist.ycont = 51;
        end
    #endif
        for (int i = 0; i < numSonar; i++)
            //Loop through sonar

```

```

{
    sonarReading = myRobot->getSonarReading(i);
    //Get each sonar reading
    sang[i] = sonarReading->getSensorTh();
    // Obtaining relative position in mm, and updating the grid
    spos[i] = (sonarReading->getRange());
    if ((spos[i]/1000)<3.3) {
        angle = sang[i];
        int col= gridRobot.xcont+round((spos[i]*cos((Pact+angle)*PI/180)+
        dsonar[i]*cos((Pact+asonar[i])*PI/180)-xact+xresto)/100);
        int row = gridRobot.ycont+round((spos[i]*sin((Pact+angle)*PI/180)+
        dsonar[i]*sin((Pact+asonar[i])*PI/180)-yact+yresto)/100);
        if ((row<401) && (row>=0) && (col<401) && (col>=0)){
            gridRobot.Cont[row*401+col] += 1;
            if (gridRobot.Cont[row*401+col]>4)
                gridRobot.Cont[row*401+col] = 4;
        }
    }
}

}

# if 0
std::cout<<"sonar[1]_="<<spos[1]<<"sonar[2]_="<<spos[2]<<"sonar[3]_="
<<spos[3]<<"sonar[4]_="<<spos[4]<<"sonar[5]_="<<spos[5]<<"sonar[6]_="
<<spos[6]<<std::endl;
# endif

// Obtaining forces
groupForces = gridRobot.getForceQuadrant(Pact);
std::cout<<"fnor_="<<groupForces.fnor<<"foes_="<<groupForces.foes
<<"fest_="<<groupForces.fest<<std::endl;
Pdes = atan2(yf-yact,xf-xact); //P deseado entre <-180, 180>
Pn = Pact - Pdes*180/PI;
xn = xact - xf;
yn = yact - yf;
// Correct the value
if (Pn > 180)
    Pn = Pn-360;
if (Pn < -180)
    Pn = Pn+360;
// Selecting the behavior
modo = controllerFB.getValue(groupForces.fnor,groupForces.foes,groupForces.fest,Pn);
modo = round(modo);
if (modo == 1){
    dphi = -1*controllerWFI.getValue(groupForces.fest+groupForces.fnorE);
    dvel = 1;
}
else if (modo == -1) {
    dphi = 1*controllerWFD.getValue(groupForces.foes+groupForces.fnorO);
    dvel = 1;
}
else if (modo == 0){
    op = controllerOA.getValue(groupForces.fnor,groupForces.foesN+
    groupForces.fnorO,groupForces.festN+groupForces.fnorE);
    dphi = -1*op.opDphi;
    dvel = (op.opDv+1)/2;
}
else if (modo == 2){
    dphi = -1*Pn;
    dvel = 1;
}
if (dphi > 180)
    dphi = dphi-360;
if (dphi < -180)

```

```

dphi = dphi+360;
// Reset desired action
myDesired.reset();
// Near limit to the goal
if (fabs(xn*xn+yn*yn)<0.1) {
    myDesired.setVel(0);
    return &myDesired;
}
std::cout<<"La_posicion_es_["<<xact<<" "<<yact<<
    "]"_Modo_["<<modo<<"_turn_["<<dphi<<std::endl;
// Setting the desired commands
myDesired.setDeltaHeading(dphi);
myDesired.setVel(dvel*300);
return &myDesired;
}

```

Listing 5 – Subclase ArGroup implementado para el controlador.

```

#include "Aria.h" #include "NfNavigator.cpp"
// Implements nfNavigator actionn group
class NfNavigatorGroup : public ArActionGroup
{
public:
    NfNavigatorGroup(ArRobot *robot, float xini, float yini,
        float xgoal, float ygoal, int fd, const char *modo);
    ~NfNavigatorGroup(void);
    // Declaring the actions that will be added
protected:
    NfNavigator* nfnavigator;
    ArActionLimiterForwards* limiter;
    ArActionLimiterForwards* limiterFar;
    ArActionLimiterBackwards* backwardsLimiter;
    ArActionConstantVelocity* stop;
    ArActionConstantVelocity* backup;
};
// Constructor
NfNavigatorGroup::NfNavigatorGroup(ArRobot *robot, float xini, float yini,
    float xgoal, float ygoal, int fd, const char *modo): ArActionGroup(robot)
{
    // Instantiate and add the actions to the group
    nfnavigator = new NfNavigator(xini,yini,xgoal,ygoal,fd,modo);
    //limiter = new ArActionLimiterForwards("speed limiter near", 300, 600,250);
    //backwardsLimiter = new ArActionLimiterBackwards;
    //stop = new ArActionConstantVelocity("stop", 0);
    //backup = new ArActionConstantVelocity("backup", -200);
    addAction(nfnavigator, 100);
    //addAction(limiter, 100);
    //addAction(backwardsLimiter, 98);
    //addAction(backup, 50);
    //addAction(stop, 30);
}
// Destructor
NfNavigatorGroup::~NfNavigatorGroup(void) {}

```

Listing 6 – Subclase ArMode del controlador implementado.

```

#include "Aria.h" #include "NfNavigatorGroup.cpp"
class NfNavigatorMode : public ArMode {
public:
    NfNavigatorMode(ArRobot *robot, const char *name, char key, char key2,

```

```

    float xini, float yini, float xgoal, float ygoal, int fd, const char *modo);
~NfNavigatorMode(void);
void activate();
//Activate mode
void deactivate();
//Deactivate mode
// Declaring the group associated with this mode protected:
NfNavigatorGroup myGroup;
float xgoal, ygoal;
};
// Constructor
NfNavigatorMode::NfNavigatorMode(ArRobot *robot, const char *name,
    char key, char key2, float xini, float yini, float xgoal, float ygoal,
    int fd, const char *modo)
    : ArMode(robot, name, key, key2), myGroup(robot, xini, yini, xgoal, ygoal, fd, modo)
{
    myGroup.deactivate();
    //Deactivate group
}
//(only run when key pressed)
// Destructor
NfNavigatorMode::~NfNavigatorMode(void){}
void NfNavigatorMode::activate() {
    if (!baseActivate())
    {
        return;
    }
    myGroup.activateExclusive();
}
// Deactivation method
void NfNavigatorMode::deactivate() {
    if (!baseDeactivate())
    {
        return;
    }
    myGroup.deactivate();
}

```