UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

****

FACULTAD DE INGENIERÍA

Optimización de Campos Potenciales para Navegación mediante Algoritmos Genéticos

T E S I S

QUE PARA OBTENER EL TÍTULO DE

INGENIERO ELÉCTRICO ELECTRÓNICO

MÓDULO ELECTRÓNICA

P R E S E N T A

JESÚS CRUZ NAVARRO

DIRIGIDA POR

DR. JESÚS SAVAGE CARMONA

Ciudad Universitaria
Mayo 2013

Agradezco a ...

... a Ari, por su amistad y cariño incondicional en todo momento.

... a Celia, por apoyarme y creer en mi.

... a mi padre, por convertirme en el hombre que soy actualmente.

... a Marco, por su ayuda y consejos.

... a Justina, por permitirme experimentar con ella.

... a PAPIIT IN11761 “Robot de servicio para asistencia a adultos mayores y en sistemas hospitalarios”, por el apoyo recibido durante esta investigación.

Contenido

1 Introducción **1**
- 1.1 Objetivo . . . 4
- 1.2 Descripción del documento . . . 4

2 Marco teórico **7**
- 2.1 Robots móviles autónomos . . . 7
  - 2.1.1 Paradigmas de la robótica . . . 8
  - 2.1.2 Locomoción . . . 13
- 2.2 Campos potenciales artificiales . . . 16
- 2.3 Algoritmos genéticos . . . 18
  - 2.3.1 Creación de la población inicial . . . 20
  - 2.3.2 Evaluación de los cromosomas . . . 20
  - 2.3.3 Selección . . . 21
  - 2.3.4 Recombinación . . . 22
  - 2.3.5 Mutación . . . 23
  - 2.3.6 Convergencia . . . 23

3 Desarrollo del simulador **25**
- 3.1 El sistema de navegación . . . 25
- 3.2 El simulador . . . 29
  - 3.2.1 La interfaz gráfica . . . 31
  - 3.2.2 Los obstáculos . . . 31
  - 3.2.3 El robot . . . 31
  - 3.2.4 El sensor láser . . . 42

4 Planeación de trayectorias **53**
- 4.1 Campos Potenciales Artificiales . . . 54
  - 4.1.1 Fuerza de atracción . . . 54

4.1.2 Fuerza de repulsión 54
4.1.3 Fuerza Resultante 56
4.1.4 Algoritmo de implementación 58
4.2 Control del robot 58
4.3 Planeación de trayectoria 65

5 Implementación del algoritmo genético 69
5.1 Codificación de los cromosomas 70
5.2 Población inicial 71
5.3 Evaluación de los cromosomas 72
5.4 Función de evaluación 73
5.5 Selección de los cromosomas padres 75
5.6 Cruza de los cromosoma padre 76
5.7 Mutación de los hijos 77
5.8 Término del algoritmo 78

6 Optimización de Trayectorias 79
6.1 Metodología 79
6.2 Resultados en simulación 81
6.2.1 Mapa 1 : Pasillo con un obstáculo 81
6.2.2 Mapa 2 : Ambiente Real Complejo 88
6.2.3 Mapa 3 : Oscilaciones en pasillos angostos 95
6.2.4 Mapa 4 : Objetivo final no alcanzable por obstáculos cercanos 97
6.2.5 Mapa 5 : Ambiente con múltiples cuartos 99
6.3 Resultados experimentales 103
6.3.1 Experimento 1 104
6.3.2 Experimento 2 106

7 Conclusiones y trabajo futuro 111
7.1 Conclusiones 111
7.2 Trabajo Futuro 112

Capítulo 1

Introducción

En los últimos años, los robots se han hecho cada vez más comunes en la vida diaria de los seres humanos. Desde que el término *robot* fue utilizado por primera vez en 1920 por el escritor checo Karel Ĉapek para su obra *Robots Universales de Rossum*, estos han evolucionado hasta sustituir a los humanos en actividades que requieren de una alta precisión, largos periodos de trabajo o conllevan un alto riesgo, como la manufacturara de alta tecnología o la exploración espacial.

Al ser claros los beneficios que los robots aportan a los humanos, nace la inquietud de llevar a los robots a los lugares donde las personas se desenvuelven normalmente para asistirlas en sus labores cotidianas. Estos robot son los denominados robots de servicio.

Para atender esta necesidad, en el laboratorio de Bio-robótica, ubicado en la Facultad de Ingeniería de la UNAM, en el año 2011 inicia el proyecto Justina, a cargo del Dr. Jesús Savage Carmona, con el objetivo de construir un robot móvil autónomo de servicio que pueda ayudar en las tareas domesticas. Para cumplir con este objetivo, en este laboratorio se investigan e implementan técnicas de diversas áreas de la computación, electrónica y mecánica que permitan al robot tener las capacidades para ayudar a los humanos en dichas tareas.

Una de las capacidades básicas que debe poseer este robot es la de trasladarse de un punto a otro en ambientes cotidianos. Generalmente, en estos ambientes existe un gran número de obstáculos, como muebles y personas, con los que al trasladarse existe un riesgo de colisión, poniendo en peligro la seguridad de estos y la de sí mismo. Es por esto que se requiere de un sistema de navegación que le permita obtener información sobre el entorno y, con base en esta, planear trayectorias seguras, eficientes y alejada de los obstáculos para alcanzar un destino determinado.

Actualmente, el sistema de navegación del robot utiliza un mapa creado previamente con la información de los obstáculos existentes, y un conjunto de nodos conectados entre sí en el espacio libre que determinan la trayectoria que se debe seguir para llegar de un punto a otro. Con la información de este mapa, cuando el robot debe llegar a un punto busca la ruta más corta entre los nodos y se traslada por esta en linea recta. Aunque esta forma de navegación es segura y confiable en ambientes controlados, depende enteramente de que los obstáculos sean estáticos. Si existe un cambio de posición o de orientación en estos o se presentan nuevos obstáculos, el riesgo de colisión crece enormemente. Lo que se busca en este trabajo es mejorar la forma en que el robot se traslada entre nodos, permitiendo la evasión de obstáculos aun cuando existan cambios en la configuración de los obstáculos o existan nuevos en el entorno.

Diferentes métodos se han desarrollado para permitir a los robots navegar evadiendo obstáculos en entornos cambiantes, sin embargo, este problema sigue siendo objeto de estudio para el cual no se ha encontrado una solución completa.

Uno de los métodos de navegación más utilizados es el denominado “campos potenciales artificiales”, desarrollado en 1985 por O. Khatib, el cual se basa en situar al robot en un campo donde interactúan dos fuerzas: una repulsiva generada por los obstáculos que lo alejan de estos, y una atractiva que lo impulsa hacia el punto al que se desea llegar. Sumando estos efectos, se puede conseguir que el robot navegue hasta un punto dado, evadiendo los obstáculos que se encuentren entre este y el punto final. Con esto, la evasión de obstáculos, el problema de la planeación de trayectorias, generalmente tratado como un problema de planeacion de alto nivel, puede ser un componente efectivo del control de bajo nivel en tiempo real (Khatib, 1985).

Para implementar el modelo de campos potenciales artificiales, es necesario determinar el valor de un conjunto de constantes, las cuales dependen del entorno en el cual va a navegar el robot. Si el valor de las constantes no es el adecuado, el robot podría no llegar al objetivo final, debido a la fuerza de repulsión generada por ciertas configuraciones de obstáculos o experimentar movimiento erráticos, denominados oscilaciones, que hacen que el riesgo de colisión entre el robot y los obstáculos aumente considerablemente.

El método de campos potenciales considera al robot como omnidireccional, es decir, que puede trasladarse en cualquier dirección con cualquier orientación. En el caso del robot Justina, debido a la configuración y tipo de llantas que utiliza, este tipo de movimiento no es posible, por lo que la implementación de este método no es directa y se deben considerar estas restricciones de movimiento al elegir los parámetros de los campos potenciales.

Para elegir un conjunto de parámetros óptimos, se propuso utilizar un algoritmo genético, el cual es un modelo computacional, inspirado en la teoría de la evolución, que se ha utilizado en la resolución de problemas en los que la búsqueda de la solución se realiza en un espacio muy grande. Un uso frecuente de los algoritmos genéticos es como función de optimización en problemas de gran complejidad, en los que no se cuenta con un modelo matemático o no es posible encontrar los valores óptimos de forma analítica. La elección de este método estuvo basada principalmente en dos razones : la primera es que es posible obtener buenas soluciones, aunque no sean la mejor, en relativamente pocas iteraciones, y la segunda es que su implementación es simple, dado que no requiere un modelo matemático de todo el sistema (Sivanandam y Deepa, 2008).

Los algoritmos genéticos basan su función en la asignación de un valor denominado "ajuste" a las soluciones de un problema, en este caso las trayectorias de un robot, el cual representa qué tan buena o mala fue una solución. Creando un cierto número de soluciones y seleccionando y modificando las mejores por medio de los operadores definidos para este método, a través de varias iteraciones, es posible encortar una solución óptima, que satisfaga las características deseadas de un problema dado.

Puesto que el realizar un gran número de trayectorias de prueba en el robot real para obtener sus características y asignarles el valor de ajuste requerido por los algoritmos genéticos para su optimización sería una tarea insegura, lenta e impráctica, se requiere de un programa que simule los sistemas involucrados durante la navegación utilizando el método de campos potenciales y ejecute las trayectorias a ser evaluadas por el algoritmo genético.

La hipótesis de este trabajo es que utilizando un simulador del sistema de navegación del robot Justina es posible ejecutar un algoritmo genético para la obtención un conjunto de valores para las constantes del campo potencial que le permitan describir trayectorias alejada de los obstáculos, en la que exista una gran probabilidad de llegar al punto destino y en el menor tiempo posible, en un ambiente determinado.

Como resultado de los avances que se han obtenido en el desarrollo del robot Justina, este ha participado dos veces en la competencia internacional de róbotica *RoboCup* y participará en la próxima competencia con sede en Eindhoven, Holanda. Dado que las soluciones obtenidos durante la realización del presente trabajo fueron muy exitosas, el sistema desarrollado se utilizará en las pruebas de esta competencia. Esto, junto con un fuerte interés científico en las áreas del conocimiento que conjunta la róbotica, sirvieron como motivación para el desarrollo de esta tesis.

1.1 Objetivo

El objetivo principal del presente trabajo es:

Desarrollar un sistema que permita calcular los parámetros de los campos potenciales para el robot Justina mediante un algoritmo genético para conseguir trayectorias optimizadas que le permita navegar hasta un punto determinado, en un espacio de trabajo definido.

Para cumplir con este objetivo se realizaron las siguientes actividades:

- Desarrollar un sistema que simule los movimientos del robot, basado en su modelo cinemático, y los sensores utilizados durante la navegación que sea escalable y fácil de utilizar.
- Obtener una ecuación para el control de las velocidades de las llantas del robot, tomando en cuanta las restricciones de movimiento de su configuración.
- Desarrollar un algoritmo que permita la evasión de obstáculos durante la navegación, basado en el método de campos potenciales.
- Integrar a este sistema un método que optimice los parámetros de las trayectorias utilizando un algoritmo genético.
- Desarrollar una metodología para el uso del sistema de optimización de trayectorias en un ambiente determinado.
- Probar en simulación un conjunto representativo de trayectorias optimizadas.
- Probar experimentalmente las trayectorias simuladas en el robot Justina.

1.2 Descripción del documento

El presente trabajo es el resultado de la investigación realizada en el Laboratorio de Bio-robótica, en la Facultad de Ingeniería de la UNAM, en el área de navegación autónoma para robots móviles, utilizando como modelo a Justina, un robot de servicio construido en este laboratorio.

En el Capítulo 2 se presentan los conceptos teóricos y las definiciones en los que se basa el presente trabajo, comenzando con una descripción de los robots móviles

autónomos, sus paradigmas y el sistema que permite la navegación. Además se describe el modelo cinemático del sistema de locomoción del robot Justina. Se presenta el concepto y modelo de campos potenciales artificiales que se utilizan en este trabajo y se explica el modelo del algoritmo genético canónico y sus operadores.

El Capítulo 3 comienza con un panorama general de la arquitectura del sistema de navegación del robot Justina y sus características particulares. Con base en esto, se presentan los métodos utilizados para la simulación de los movimientos del robot y de los sensores.

En el Capítulo 4 se presenta la implementación del método de campos potenciales en el robot Justina y el algoritmo desarrollado que hace uso de esta implementación para la planeación de trayectorias. Se presenta también la ecuación utilizada para el control de las velocidades de las llantas.

En el Capítulo 5 se describe la implementación del algoritmo genético en el programa de simulación y la de sus operadores utilizados para la optimización de trayectorias

En el Capítulo 6 se describe la metodología propuesta para la optimización de trayectorias. Además, en este capitulo se presentan los resultados en simulación obtenidos utilizando esta metodología y los resultados experimentales que verifican las simulaciones.

Finalmente, las conclusiones obtenidas a partir de la presente investigación y el trabajo futuro propuesto se presentan en el Capítulo 7.

Capítulo 2

Marco teórico

Este capítulo comienza, en la Sección 2.1, con la definición y conceptos de los robots móviles autónomos y los paradigmas de inteligencia artificial enfocados a estos. Además, se exponen los tipos de locomoción que existen y se analiza el modelo cinemático de los robots móviles en configuración de par diferencial. A continuación, en la Sección 2.2, se explica el concepto de campo potencial artificial y se plantean sus ecuaciones. Para finalizar, en la sección 2.3, se describe el modelo de los algoritmos genéticos simples y de los operadores básicos que lo conforman.

2.1 Robots móviles autónomos

Un robot es un dispositivo mecánico versátil, equipado con sensores y actuadores para realizar movimientos, que es controlado por un sistema de cómputo. Algunos ejemplos de estos son un brazo manipulador, un vehículo con patas o ruedas, o una combinación de estos (Latombe, 1991). Los movimientos que son ejecutados por el robot pueden ser pre-programados, tele-operados o autónomos.

En el caso de los robots pre-programados, el espacio de trabajo en el que ejecutan sus movimientos debe ser un ambiente estático y controlado cuidadosamente, puesto que no seria práctico ni realista programar cada posible interacción que pueda surgir entre estos y los objetos dentro del espacio de trabajo. Este tipo de comportamiento es empleado principalmente en los robots industriales, dado que realizan una tarea repetitiva, por ejemplo, en una línea de ensamblaje.

Los robots tele-operados se utilizan principalmente en lugares donde el acceso a los seres humanos puede ser difícil o conlleva algún tipo de riesgo. Utilizan sensores, como cámaras y sonares, que transmiten las lecturas remotamente a un operador humano para que este, a una distancia segura, pueda conocer el estado del entorno y del robot y, con base a esto, tomar decisiones para ejecutar los movimientos necesarios para completar su tarea. Un ejemplo de estos es el robot Sojourner creado por la NASA para explorar la superficie de Marte en 1997.

Los robots autónomos son robots capaces de llevar a cabo una tarea o realizar una función específica, en ambientes no controlados y sin ningún tipo de operación por parte de un humano por periodos de tiempo prolongados. Considerando estas características, en la actualidad no existen robots que sean totalmente autónomos, puesto que la mayoría de los robots autónomos no pueden funcionar por largos periodos de tiempo en estos ambientes, excepto en cierto tipo de situaciones. Un ejemplo de esto es la competencia internacional RoboCup, en donde diferentes tipos de robots muestran una autonomía completa durante las pruebas en las que se exhiben (Bekey, 2005).

Durante los últimos años, muchas de las investigaciones en el área de la robótica se han enfocado a este último tipo de robots, puesto que se espera sean de gran utilidad para los humanos, debido en gran medida a que estos robots no están limitados a un ambiente controlado, por lo que pueden asistir a los humanos en espacios tan cotidianos como una casa o una oficina. Además, al no requerir de un operador, los recursos humanos pueden ser utilizados en tareas más complejas. El presente trabajo está enfocado a este último tipo de robots, los robots autónomos.

El potencial de los robots autónomos para facilitar la vida de los humanos es muy alto. Es por esto que surge la necesidad de las investigaciones enfocadas en dotar a los robots de una completa autonomía. Se puede obtener una idea de lo que esta autonomía conllevaría en un futuro en lo mencionado por Latombe (1991):

> "Uno de los objetivos finales de la róbotica es el crear robots autónomos. Estos aceptarán una descripción de alto nivel de una tarea y la ejecutarán sin intervención humana. Esta descripción especificará *qué* es lo que requiere el usuario que sea realizado en lugar de especificar *cómo* realizarlo."

2.1.1 Paradigmas de la robótica

Los robots autónomos necesitan un sistema de inteligencia que les permita resolver problemas y desarrollar estrategias para cumplir con un conjunto de tareas asignadas,

mientras operan en ambiente dinámicos. Meystel (1991) menciona que la autonomía está determinada por el nivel de inteligencia de una máquina: un alto nivel de inteligencia conlleva un alto nivel de autonomía.

Las acciones que pueden ser desempeñadas por los robots autónomos se clasifican en tres categorías, según la función que desempeñan dentro del sistema. Estas tres clasificaciones, denominadas *primitivas de la robótica*, son sensar, planear y actuar.

Sensar Se refiere a las funciones que obtienen información de los sensores del robot, los cuales pueden ser de dos tipos dependiendo de la información que proporcionan: propioceptivos y exteroceptivos.

Los sensores *propioceptivos* sirven para medir el estado interno del robot. Uno de los sensores más utilizados de este tipo en los robots con ruedas, son los encoders, con los cuales se puede calcular la posición del robot a partir de la medición de la posición angular de sus llantas.

Los sensores *exteroceptivos* se utilizan para obtener información sobre el ambiente en el que se desempeña. Una de las aplicaciones más utilizadas de estos tipos de sensores es la detección de obstáculos a partir de sonares, sensores de presión o láseres.

Planear Se habla de una función de planeación cuando se generan subtareas o se toman desiciones, llamadas directivas, a partir de la información obtenida de los sensores o de alguna representación del conocimiento generado anteriormente, como pude ser un mapa o un comando por voz.

Actuar Estas funciones envían comandos a los actuadores para ejecutar el movimiento de alguna parte del robot, generalmente basados en una ley de control que permite que el actuador llegue a un valor deseado en un tiempo óptimo.

En la Tabla 2.1 se presentan las primitivas de la robótica junto con la información que cada una utiliza como entrada y los resultados de esta función obtenidos a la salida. Utilizando estas primitivas, se definen tres paradigmas para organizar la inteligencia artificial de los robots autónomos. Estos son modelos de la forma en que los robots ejecutan sus tareas. Dichos modelos se basan en la forma en que interactúan las primitivas entre sí mediante sus entradas y salidas. Los paradigmas planteados por Murphy (2000) se describen a continuación.

Paradigma jerárquico

El primer paradigma utilizado fue el jerárquico, con el que se trató de emular la forma en que el humano piensa desde una perspectiva introspectiva.

Tabla 2.1 Primitivas de la robótica.

Primitivas	Entrada	Salida
SENSAR	Datos del sensor	Información sensada
PLANEAR	Información	Directivas
ACTUAR	Información sensada o directivas	Comandos de actuador

Para realizar una tarea utilizando este modelo, el primer paso es adquirir información sobre el ambiente y sobre el robot durante la etapa de sensado. Con esta información y datos adquiridos anteriormente por un proceso cognoscitivo, se genera una representación del mundo y del robot. A partir de esta representación, en la siguiente etapa, el robot planea todas las acciones y genera las directivas que necesita ejecutar para realizar uno o varios objetivos específicos, con el fin de realizar la tarea encomendada. Por último, las directivas generadas se convierten en comandos que son ejecutados por los actuadores. Cuando se completan los objetivos planeados y se han ejecutado las directivas, este proceso termina y se repite nuevamente, hasta que el robot lleva a cabo la tarea especificada. Estos pasos se pueden observar en la Figura 2.1.

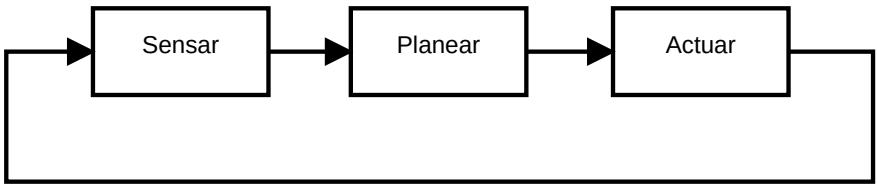


Figura 2.1 Paradigma jerárquico.

Con este, se pueden llevar a cabo tareas muy complejas dado que, el tener un modelo completo del mundo permite una planeación muy eficiente al momento de generar los objetivos y directivas para realizarlas.

Las desventajas de este tipo de paradigma son que, dependiendo de la complejidad del modelo del mundo, la etapa de planeación tiende a ser muy lenta y costosa computacionalmente, por lo que no permite una rápida ejecución de las funciones de actuado. Por tanto, si algún suceso inesperado surge después de la etapa de sensado, la planeación podría no ser óptima, dado que el modelo del mundo estará incompleto y no se generarán directivas para actuar ante dicho suceso.

Paradigma reactivo

Este modelo ésta basado, en gran medida, en observaciones del proceso que llevan a cabo los insectos para moverse. Fue propuesto inicialmente por Brooks (1986). En este, la etapa de planeación es removida dando lugar a que las funciones de sensar y actuar se comuniquen directamente. Brooks afirma que, para un robot móvil autónomo, el único modelo necesario del mundo es la información de los sensores del ambiente en el que se esta moviendo. La información de los sensores es enviada directamente a los actuadores, figura 2.2, con lo que se activan una serie de directivas de bajo nivel simples, concurrentes y asíncronas, para el control de estos. Directivas de alto nivel también pueden ser activadas, dependiendo de la tarea que se quiera realizar. Cabe mencionar que las directivas de bajo nivel no conocen los comportamientos de las directivas de alto nivel, lo que da lugar a comportamientos emergentes que se pueden denominar inteligentes.

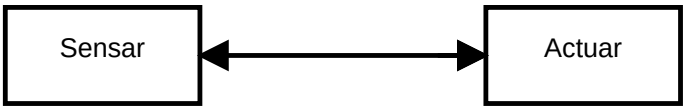


Figura 2.2 Paradigma reactivo.

El paradigma reactivo presenta la ventaja de que, al no existir una fase de planeación, la velocidad del control y las capacidades de reacción del robot se ven incrementadas. Asimismo, dado que no es necesario generar y procesar un modelo global del mundo, los requerimientos computacionales son mucho menores que los del paradigma jerárquico.

A pesar de estas ventajas, los robots con una arquitectura puramente reactiva no son capaces de resolver tareas demasiado complejas (Bekey, 2005), por lo que no es conveniente utilizarla para robots de propósito general.

El modelo utilizado en el presente trabajo esta basado en este paradigma, en donde los obstáculos son detectados para su evasión, sin generar una representación del mundo.

Mataric (1992) define la estrategia de diseño de este tipo de sistemas. Este diseño procede desde abajo hacia arriba (bottom-up) donde se comienza definiendo los reflejos del robot para garantizar su seguridad, por ejemplo, el evadir obstáculos. El proceso es descrito de la siguiente manera:

1 Especificar cualitativamente los comportamientos deseados. Estos comportamientos son los objetivos del sistema

2 Especificar los comportamientos en términos de las acciones en el espacio observable. Esto es, descomponer los comportamientos en acciones observables y disjuntas, que servirán como objetivos secundarios.

3 Especificar las acciones en términos de los actuadores del robot. Esto es, traducir las acciones en comandos para los actuadores. En este sentido, para maximizar la percepción de los sensores, se deben ejecutar movimientos pequeños e incrementales.

Paradigma híbrido

El paradigma híbrido nació en la década de los 90 y actualmente sigue siendo objeto de estudio. Su desarrollo está basado en el paradigma reactivo, con la intención de ejecutar tareas más complejas que con este eran muy complicadas de manejar.

Al igual que en el paradigma reactivo, existe una comunicación directa entre la etapa de sensar y la de actuar, en donde la información detectada puede disparar comportamientos que serán ejecutados inmediatamente por los actuadores. Sin embargo, la información sensada también es procesada por funciones de planeación que genera un modelo global del mundo, como se ve en la Figura 2.3. Éstas realizan tareas de inteligencia artificial relacionadas con la planeación de objetivos globales, resolución de problemas y aprendizaje de maquina. Las funciones de planeacion pueden generar directivas para los actuadores para cumplir con objetivos específicos.

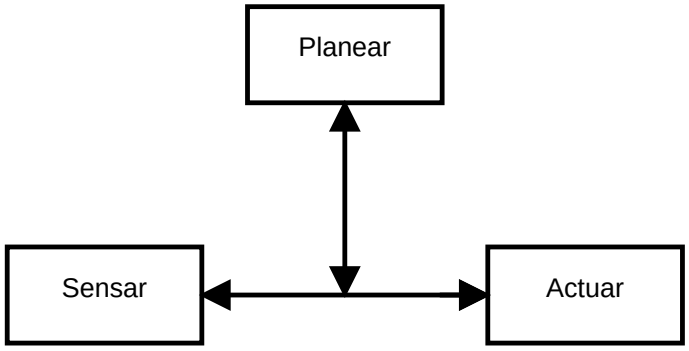


Figura 2.3 Paradigma híbrido.

Con este paradigma se pueden desarrollar robots autónomos de propósito general que reciban y ejecuten tareas encomendadas de los humanos, utilizando las ventajas de los dos paradigmas mencionados anteriormente. Se obtienen funciones cognoscitivas de nivel superior y un modelo global detallado del mundo, al mimo tiempo que una rápida ejecución de comportamientos básicos para adecuarse a las perturbaciones que se pueden presentar en ambientes no controlados.

2.1.2 Locomoción

Si se desea que un robot ejecute tareas en ambientes cotidianos, es necesario que pueda trasladarse de un punto a otro. Los robots que cuentan con esta capacidad se denominan robots móviles.

Para que un robot móvil se desplace autónomamente, además de un sistema de inteligencia artificial que le permita identificar la trayectoria por la que debe moverse, es necesario un sistema mecánico que ejecute los comandos generados por la inteligencia, para que pueda llegar de un punto inicial a un punto final en un tiempo finito.

Existen diferentes formas de clasificar a los robots móviles, algunos toman como base el ambiente en el que se moverán, dividiéndolos en terrestres, aéreos y acuáticos. Otro tipo de clasificación utilizada se basa en la aplicación que el robot va a desarrollar, en donde sus categorías abarcan robots militares, de servicio, espaciales, etc.

Dentro de los robots terrestres, estos se pueden dividir tomando en cuenta el tipo de dispositivos utilizados para ejecutar el movimiento. De acuerdo con esta clasificación, los robots móviles se clasifican en:

- Robots con ruedas
- Robots con patas
- Robots con orugas

Dentro de esta clasificación, los robots con ruedas son los más populares dentro de los robots móviles, debido en gran medida a la facilidad de implementación, su eficiencia y la estabilidad que presentan, aunque generalmente están reservados a terrenos planos en donde el deslizamiento de las ruedas es mínimo. Según el número y tipo de ruedas y su cinemática, estos pueden clasificarse en:

- Ackermann
- Síncrona

- Triciclo
- Diferencial

Siendo esta última configuración la utilizada por el robot Justina. A continuación se mencionan algunas de sus características y el modelo cinemático que la representa.

En la configuración diferencial el robot utiliza dos llantas fijas controladas por motores independientes, una a cada lado con sus ejes de giro alineados. Variando la velocidad de las ruedas, se consigue que el robot trace trayectorias alrededor de un punto fijo común a ambas llantas. Este punto se encuentra sobre el eje de rotación de las llantas y se le denomina "centro instantáneo de rotación". Al variar las velocidades en las llantas, la posición del centro instantáneo de rotación varia a lo largo del eje de las llantas, lo que permite al robot desplazarse en linea, trazar curvas y girar sobre su propio eje. Esta presenta una singularidad, es decir, una restricción al movimiento, por la cual el robot solo puede moverse en una dirección perpendicular al eje de las llantas.

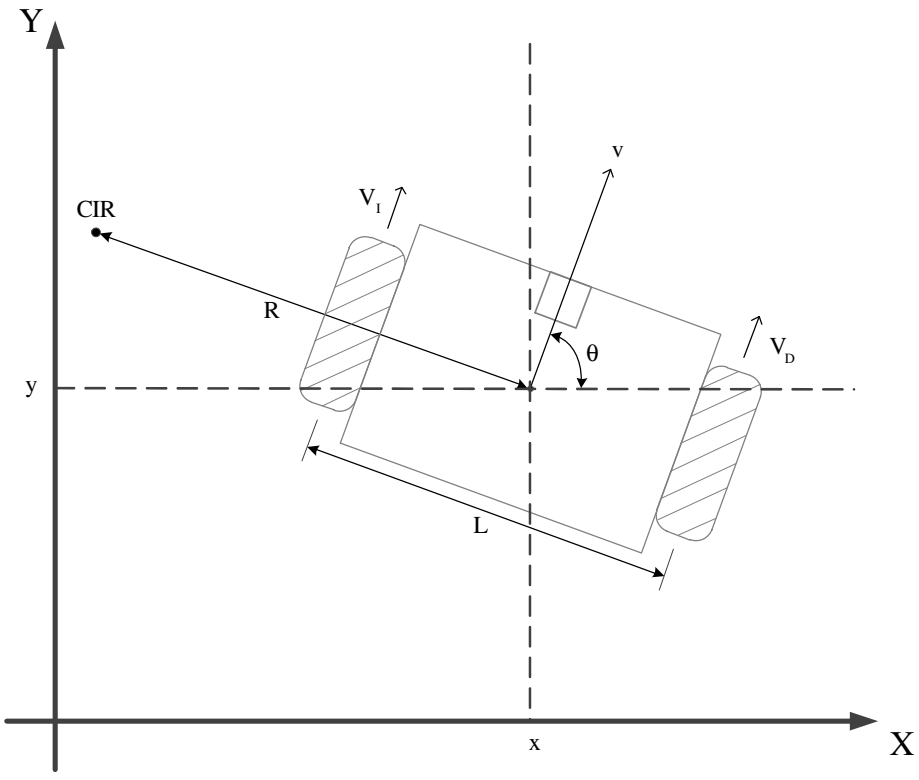


Figura 2.4 Robot con configuración diferencial

Si se considera que el robot se mueve en una superficie plana, sin deslizamiento entre el suelo y las llantas, utilizando la figura 2.4, se tiene que las velocidades lineal v y angular w del robot son (Cook, 2011):

$$v = \frac{V_d + V_i}{2} \tag{2.1}$$

$$\omega = \frac{V_d - V_i}{L} \tag{2.2}$$

en coordenadas absolutas, se obtienen las siguientes ecuaciones:

$$\dot{x} = \frac{v_d + v_i}{2}\cos\theta \tag{2.3}$$

$$\dot{y} = \frac{v_d + v_i}{2}\sin\theta \tag{2.4}$$

$$\dot{\theta} = \frac{v_d - v_i}{L} \tag{2.5}$$

Con estas ecuaciones es fácil observar tres casos de movimiento:

- Si las velocidades en las dos llantas son iguales ($v_d = v_i$) y el sentido de giro es el mismo, no habrá cambio en la orientación del robot ($\dot{\theta} = 0$) y desplazará en linea recta;
- Si las velocidades son de igual magnitud pero el sentido de giro es contrario ($v_d = -v_i$), no existirá desplazamiento en ninguno de los ejes ($\dot{x} = \dot{y} = 0$), pero existirá un cambio en la orientación del robot por lo que el robot girará sobre el punto medio del eje de las llantas;
- Si las velocidades no son iguales ($v_d \neq v_i$), el robot se desplazará en ambos ejes y cambiara su orientación por lo que describirá trayectorias curvas.

Utilizando estas ecuaciones se puede obtener la posición (x, y)y la orientación del robot δ en un espacio coordenado, al establecer un par de velocidades en sus llantas en un tiempo t, conociendo el estado inicial del robot. Estas se utilizaron para simular el movimiento del robot en el capítulo 3 y para determinar la ecuación de control descrita en el capítulo 4.

2.2 Campos potenciales artificiales

Los campos potenciales son un método reactivo de planeación de trayectorias que consiste en modelar el espacio de trabajo como un campo escalar de potenciales en donde el robot se mueve siempre hacia puntos de menor potencial. En este campo, el objetivo final es el punto de potencial más bajo y los obstáculos se representan como puntos de máximo potencial (Vadakkepat, Tan, y Ming-Liang, 2000)

El campo potencial al que esta sometido el robot se obtiene mediante la suma de dos campos: el primero de atracción U_{att}, que lleva al robot hacia el la meta, y otro de repulsión U_{rep}, que lo mantiene lejos de los obstáculos, libre de posibles colisiones. Esto es:

$$U = U_{atr} + U_{rep} \tag{2.6}$$

Calculando el gradiente de la ecuación 2.6, se obtiene la fuerza que actúa sobre el robot dentro del campo, esto es:

$$F(q) = -\nabla U_{atr} + \nabla U_{rep} \tag{2.7}$$

Definiendo las ecuaciones que determinan la fuerza de atracción $F_{atr} = -\nabla$ y la fuerza de repulsión $F_{rep} = \nabla U_{rep}$ se tiene que:

$$F(q) = F_{rep} + F_{rep} \tag{2.8}$$

Utilizando las ecuaciones obtenidas experimentalmente por Arámbula y Padilla (2004) para mejorar el desempeño de las propuestas originalmente por Khatib (1985) tenemos que la fuerza de atracción que tiende a llevar al robot hacia la meta se define como:

$$F_{atr}(q) = -\nabla U_{atr} = -\xi(\mathbf{q} - \mathbf{q}_{atr})\frac{1}{|\mathbf{q} - \mathbf{q}_{atr}|} \tag{2.9}$$

donde:

$\boldsymbol{q}$ = Vector de posición del robot.
$\boldsymbol{q_{atr}}$ = Vector de posición del punto de atracción.
ξ = Constante de atracción ajustable.

Y la fuerza de repulsión que genera un solo obstáculo en el espacio de trabajo está determinada por:

$$F_{rep}(q) = \begin{cases} 0 & \text{si } d > d_{inf} \\ \eta \cdot \sqrt{\frac{1}{d} - \frac{1}{d_0}} \left(\frac{\boldsymbol{q} - \boldsymbol{q}_{obs}}{d^3} \right) & \text{si } d \leq d_{inf} \end{cases} \tag{2.10}$$

donde:

$\boldsymbol{q}$ = Vector de posición del robot.
$\boldsymbol{q_{obs}}$ = Vector de posición de un obstáculo.
d = $|\boldsymbol{q} - \boldsymbol{q_{obs}}|$, distancia entre un obstáculo y el robot.
η = Constante de Repulsion ajustable.
d_{inf} = Constante de Distancia de Influencia ajustable.

Por tanto, si hay un punto de atracción y n obstáculos en el mapa, la fuerza resultante que afectará la posición del robot será:

$$F_{res} = F_{atr}(q) + \sum_{i=1}^{n} F_{rep_n}(q) \tag{2.11}$$

A pesar de ser un método seguro para a las evasión de colisiones y elegante por su sencillez matemática, los campos potenciales tienen las siguientes desventajas (Cen, Wang, y Zhang, 2007) que hacen que se usen en combinación con otros métodos para la planeación de trayectorias:

Mínimo Local Cuando la fuerza de atracción y la de repulsión son iguales en magnitud pero en sentido contrario la magnitud resultante es cero y el robot no alcanzará la meta. Este es el principal problema de los campos potenciales.

Meta inalcanzable Si existe un obstáculo cerca del objetivo final, la fuerza repulsiva aumentará a medida que el robot se acerque a ésta. Si la fuerza repulsiva es mayor que la fuerza de atracción del objetivo final, el robot no llegara al objetivo final, sino que se detendrá en un punto cercano donde la resultante sea cero.

Oscilaciones Para determinadas configuraciones de obstáculos, por ejemplo en pasillos angostos, la fuerza repulsiva alejará al robot de un obstáculo; mientras el robot se aleja esta fuerza disminuye y si existe un obstáculo cercano que empuje

de nuevo al robot hacia el primer obstáculo, el robot se alejará y acercará de estos continuamente por lo que el robot se desplazará como una onda oscilatoria en el plano.

2.3 Algoritmos genéticos

Los algoritmos genéticos son una técnica de búsqueda y optimización de soluciones, desarrollada en 1975 por John H. Holland, los cuales forman parte de la llamada computación evolutiva. Estos son especialmente útiles para resolver problemas en donde la epistasis es grande, es decir, aquéllos en los cuales el resultado final depende en gran medida de la interacción entre las variables independientes, lo cual, en muchos casos no es fácil de modelar matemáticamente, característica requerida por otros modelos.

Recientemente, los algoritmos genéticos han tenido un papel muy importante en la solución de problemas complejos. Pertenecen, en Inteligencia Artificial, al grupo de métodos denominados "débiles". Éstos requieren relativamente poca información acerca del problema. Además, gracias al desarrollo actual de computadoras más rápidas y con gran poder de cálculo, el uso de estos algoritmos permite búsquedas en espacios de soluciones muy grandes en tiempos cortos.

Un algoritmo genético es una búsqueda en un conjunto de soluciones al que se le aplican los operadores básicos definidos por Holland (1992), como selección, recombinación y mutación, para obtener un nuevo espacio de soluciones que, teóricamente, será mejor que el anterior. Mediante estos operadores y evaluando cada generación, se puede hacer al mismo tiempo un muestreo de la eficacia de un subconjunto de soluciones representativo de la población general debido a la característica de paralelismo implícito, una de la más importantes de este método.

Los algoritmos genéticos simples o canónicos basados en el modelo de Holland pueden representarse con el diagrama de flujo de la figura 2.5. Este comienza con la creación de un conjunto de soluciones, denominado población inicial, obtenidas aleatoriamente. Estas soluciones se codifican en cadenas binarias, denominadas cromosomas. Cada cromosoma de este conjunto se evalúa de acuerdo a una función objetivo y se le asigna un valor denominado ajuste, dependiendo de lo bueno o malo que sea el desempeño de ésta para el problema dado. Teniendo el ajuste de cada cromosoma, se selecciona los que presentan un mejor desempeño y, mediante el operador de recombinación, se combinan para generar nuevas soluciones. Algunos cromosomas de este

nuevo grupo de soluciónes son seleccionados al azar y se modifican aleatoriamente con el fin de ampliar el espacio de búsqueda, lo que se conoce como mutación. A cada conjunto de nuevas soluciones se le conoce como "generación". Este proceso se repite hasta que se cumple con una condición de convergencia.

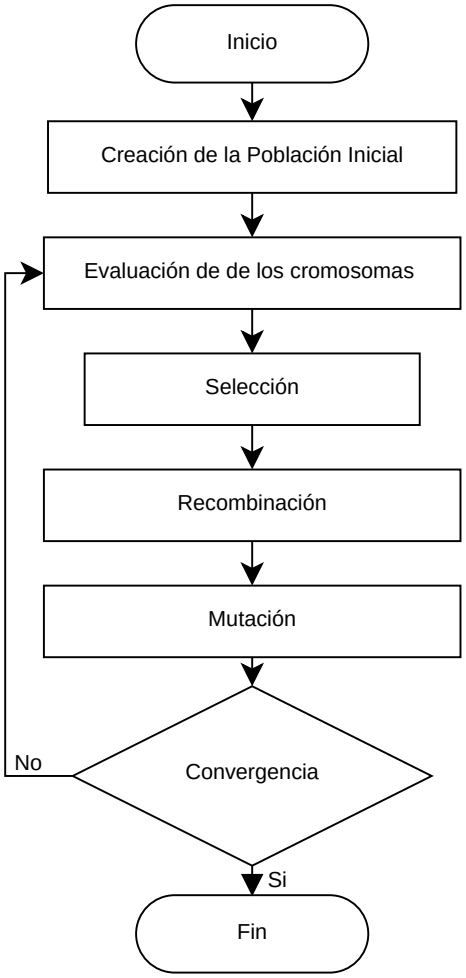


Figura 2.5 Diagrama de flujo de los algoritmos genéticos simples o canónicos.

Existen diferentes formas de implementar los operadores del algoritmo genético, las cuales, dependiendo del problema que se quiera resolver, presentan mejores características que otros, como rapidez de convergencia o la diversidad de las soluciones. A continuación se detallan los operadores básicos.

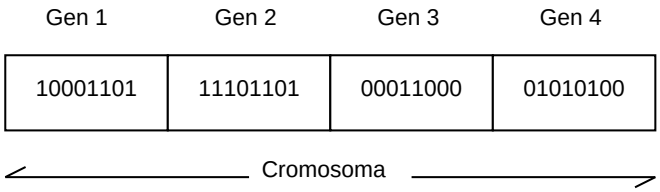


Figura 2.6 Cromosoma conformado por genes

2.3.1 Creación de la población inicial

Generalmente la población inicial es creada al azar. El número de cromosomas se elige tomando en cuenta la diversidad de soluciones y el tiempo de convergencia requerido para el problema en particular. Entre más individuos existan dentro de la población, el tiempo de convergencia será mayor dado que se requieren $O(n\log(n))$ evaluaciones, donde n es el número de individuos de la población, pero se abarca un número mayor de posibilidades en el espacio de soluciones y el riesgo de caer en un óptimo local en lugar del óptimo global se reduce (Sivanandam y Deepa, 2008).

Los cromosomas están formados por genes que representan un parámetro de la solución dentro del problema dado. Éstos se codifican en cadenas, generalmente de valores binarios, y se concatenan para conformar los cromosomas. Los valores que pueden ser asignados a los elementos de la cadena en un gen se llaman alelos. Los alelos de las cadenas binarias son ‘1’ y ‘0’.

2.3.2 Evaluación de los cromosomas

Una vez obtenida una población de cromosomas, se debe obtener un indicador de la eficacia de cada individuo de la población del problema. Para esto, se utiliza la función denominada de evaluación, que se utiliza para calcular un valor escalar denominado ajuste (fitness en inglés), que sirve para controlar la selección y cruce de los mejores individuos dentro del espacio de soluciones. La función de evaluación puede ser un modelo teórico o los resultados de un experimento y debe ser especifica para cada problema.

2.3.3 Selección

Durante el proceso de selección, las mejores soluciones son escogidas para la etapa de recombinación. A estas soluciones se les asigna una mayor probabilidad de reproducirse, esperando con esto que las características del individuo que lo hacen una buena solución puedan ser transferidas a la siguiente generación de cromosomas. Se debe designar un número de cromosomas N_{sel} a seleccionar que serán los padres de la siguiente generación mientras que los demás individuos son reemplazados por nuevos cromosomas generados al azar, lo que se conoce como presión de selección. Con una alta presión de selección, solo un número reducido de los mejores individuos generan descendencia, pero se corre el riesgo de que las soluciones se estandaricen rápidamente y se encuentre un un optimo local. Si la presión de selección es pequeña, el tiempo de convergencia del algoritmo crece.

Existen diferentes métodos de seleccionar a los mejores individuos, algunos de estos son:

De arriba hacia abajo(Top-down)

Los cromosomas se ordenan de mejor a peor basados en su valor de ajuste y se seleccionan N_{sel} número de soluciones, iniciando desde el mejor. Cada solución seleccionada será un padre durante la recombinación. Este método no suele ser muy eficiente pero su implementación es muy sencilla. Si el número de cromosomas a seleccionar es muy baja, la presión de selección sera muy alta.

A cada uno de los individuos de la población se le asigna una parte proporcional a su ajuste de una ruleta, de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Generalmente la población está ordenada con base en el valor de ajuste por lo que las porciones más grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio del intervalo [0..1] y devolver el individuo situado en esa posición de la ruleta. Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido.

Selección por Ruleta

En la selección por ruleta, los individuos tienen una probabilidad de selección proporcional a su valor de ajuste. Una forma de implementar esto es obtener el valor de

la suma total de los ajustes y generar un número aleatorio entre 0 y esta suma total. Los valores de ajuste se van sumando uno por uno hasta que esta suma exceda el valor de la suma total. La solución donde se detuvo la suma anterior será el individuo seleccionado. Este sistema presenta dos inconvenientes: el primero es que si la diferencia entre los valores de ajuste de los mejores y los peores individuos de la generaciones es muy grande, las posibilidades de obtener los mejores se vuelve también muy grande, generando una presión de selección muy alta. El segundo inconveniente es que los peores individuos pueden ser seleccionados varias veces.

Selección por Torneo

En este caso, se seleccionan al azar un número definido de individuos. De estos, el individuo con mejor ajuste es el ganador del torneo y es seleccionado para ser uno de los padres. Utilizando la misma metodología, se selecciona el otro padre, para la creación por recombinación de una nueva solución. El número de individuos que participan en el torneo influye directamente en la presión de selección: entre más individuos, la presión de selección será mayor. El torneo se realiza hasta que se obtiene el número necesarios de pares.

2.3.4 Recombinación

La operación de recombinación se realiza entre dos cromosomas elegidos durante la selección. La idea es combinar los cromosomas esperando con esto que las características que hacen de un individuo una buena solución sean transferidas a los hijos, obteniendo así mejores soluciones. Para hacer esto primero se determina, generalmente al azar, el locus o punto de cruce por donde las cadenas de los cromosomas padres serán divididas. Teniendo los dos cromosomas padres divididos en dos partes, esto es, la parte izquierda y la parte derecha en cada uno, se concatenan la parte izquierda del primer padre con la parte derecha del segundo padre para generar el primer hijo. De la misma manera la parte izquierda del segundo padre se concatena con la parte derecha del primer padre para formar el segundo hijo, como puede observarse en la figura 2.7.

Ya que el número de individuos en la población debe mantenerse constante, los cromosomas padre serán reemplazados por los cromosomas hijo en la población de soluciones.

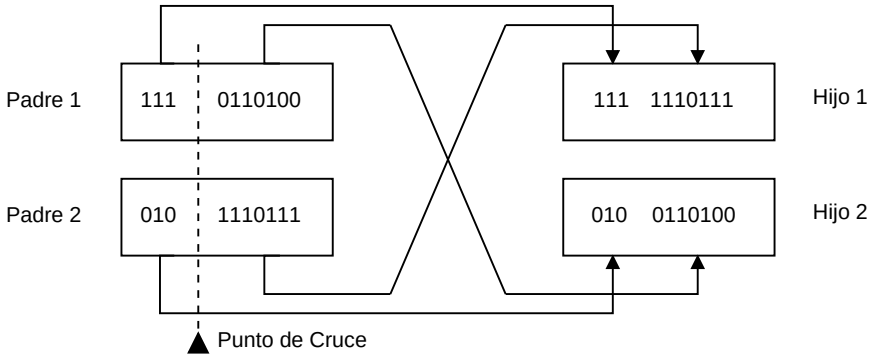


Figura 2.7 Recombinación con un punto de cruza

2.3.5 Mutación

El aplicar el operador mutación sobre un cromosoma provoca que un valor de la cadena cambie de forma aleatoria con una probabilidad, generalmente baja, definida al inicio del algoritmo. El valor de la cadena que se va a cambiar también es seleccionado al azar. Esto permite que, para cualquier solución en el espacio, exista la probabilidad de ser verificada. Con esto, la probabilidad de converger hacia un óptimo local disminuye. En cromosomas con codificación binaria la mutación significa simplemente negar el valor elegido.

2.3.6 Convergencia

Existen diferentes criterios de convergencia para terminar el algoritmo, uno de los más utilizados es definir un número máximo de generaciones para que los cromosomas evolucionen. Llegando a esta, el criterio de convergencia se cumple y el algoritmo se detiene. Cuando se utiliza este criterio, la última generación no es mutada para evitar cambiar las mejores soluciones obtenidas.

Capítulo 3

Desarrollo del simulador

En este capítulo, en la Sección 3.1, se describen las características particulares del sistema de locomoción del robot Justina y la interfaz que presentan los actuadores y sensores que se utilizan durante la navegación.

En la segunda parte, en la Sección 3.2, se presentan los métodos utilizados para la creación del simulador de los movimientos del robot y de los sensores que permiten calcular el estado del robot y del ambiente que lo rodea. Utilizando este simulador, se generaron las trayectorias utilizadas como posibles soluciones para mejorar el comportamiento del robot cuando se mueva de un punto a otro dentro de su espacio de trabajo.

Cabe mencionar que, pensando en la escalabilidad del programa, todos los cálculos que involucran vectores se hacen en el espacio de 3 dimensiones, con el valor de la componente $z = 0$, utilizando una librería que implementa los principales operadores de este espacio.

3.1 El sistema de navegación

La base móvil que integra el sistema de locomoción del robot Justina es de tipo diferencial. Está conformada por dos motores de DC, uno en cada lado, en la parte anterior del robot. En la parte posterior, cuenta con dos ruedas tipo castor para proporcionar estabilidad al robot, como se puede ver en la figura 3.1.

Los motores de DC son alimentados con 18[V] utilizando dos baterías de ácido-plomo: una de 12[V] y una de 6[V] conectadas en serie. Se utiliza la tarjeta RoboClaw

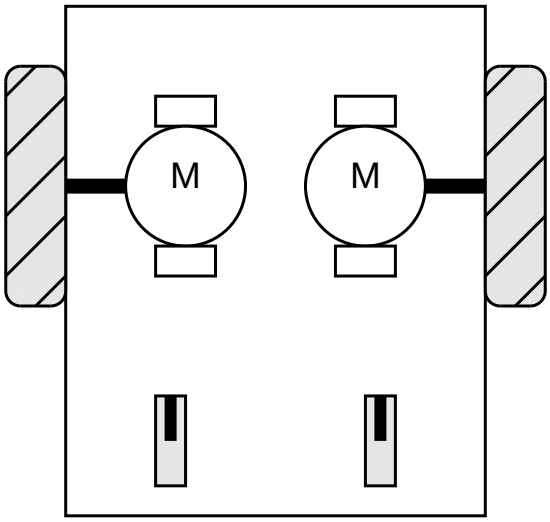


Figura 3.1 Estructura de la base móvil del robot Justina.

(BasicMicro, 2010) para controlar la velocidad de los motores utilizando modulación por ancho de pulsos o PWM. El valor del ancho de pulso se determina mediante comandos a través de un puerto serie RS232.

Para conocer la posición del robot, en cada motor se encuentra conectado un encoder de cuadratura el cual registra la distancia angular recorrida por la rueda y la dirección de giro mediante la generación de pulsos. Con éste, si la cantidad de pulsos se encuentra inicialmente en cero, se puede obtener la distancia lineal recorrida mediante la formula 3.1.

$$d = P\left(\frac{2\pi r}{P_{tot}}\right) \tag{3.1}$$

donde:

d = Distancia lineal recorrida.
P = Pulsos medidos por el encoder.
r = Radio de la llanta.
P_{tot} = Pulsos por vuelta.

La lectura de encoders y la acumulación de los pulsos generados se realiza mediante dos microcontroladores dsPIC30F4011. Éstos tienen implementado por hardware un módulo lector de encoder de cuadratura, con el cual se eliminan las perdidas de conteo de pulsos que podrían generarse cuando el microprocesador esta realizando otras funciones, como puede ser la recepción y el envió de comandos. Además del módulo de lectura de encoders, estos microcontroladores cuentan con dos módulos de comunicación serial, lo que permite un esquema de comunicación maestro-esclavo. Este esquema se escogió con el objetivo de utilizar un solo puerto de comunicaciones en la computadora donde se ejecuta el programa de planeación de trayectorias. El diagrama de comunicación se puede apreciar en la figura 3.2.

El microcontrolador maestro está programado para que, con un comando lea el valor del encoder que esta conectado directamente a él, pida al microcontrolador esclavo el valor del segundo encoder y devuelva ambos valores. Asimismo, la magnitud de la velocidad que se requiere establecer en cada motor puede ser enviado en un solo comando. La magnitud de velocidad que puede ser enviado a las llantas corresponde al valor de un byte, esto es un valor entre 0 y 255, donde el valor de 0 corresponde a la máxima velocidad hacia atrás, un valor de 127 corresponde a una velocidad de 0 y el de 255 a la máxima velocidad hacia el frente, como se puede ver en la la gráfica de la figura 3.3.

Por seguridad, el microcontrolador maestro esta configurado para que, cuando recibe un comando de velocidad, establece los valores enviados en los motores por un tiempo máximo de aproximadamente de 100ms. Si no ha recibido otro comando de velocidad, a los 100 ms establece una velocidad de 0 m/s en los motores. De esta forma, si se presenta un problema de conexión entre la base y la computador, el robot no seguirá una trayectoria sin control, en cambio se detendrá en un tiempo corto. Esto quiere decir que, por ejemplo, si se quiere establecer una velocidad constante por 2 segundos, se deben mandar 20 comandos, uno cada 100 ms, con el valor de la velocidad deseada.

Para obtener información del ambiente, mas específicamente de obstáculos para el robot, el sistema de navegación utiliza un sensor láser Range Finder Hokuyo modelo URG-04LX-UG01 (LTD., 2005). Este sensor escanea un arco de 240° en 683 pasos, es decir toma una lectura de distancia cada 0.36°, para obtener información de los objetos en el ambiente cercano a él. La distancia máxima para la detección de objetos en el ambiente es de 4[m]. Su funcionamiento se basa en el cálculo de la diferencia de fase entre el haz de luz generado y el recibido, permitiendo mediciones estables sin que el color de los objetos altere las lecturas.

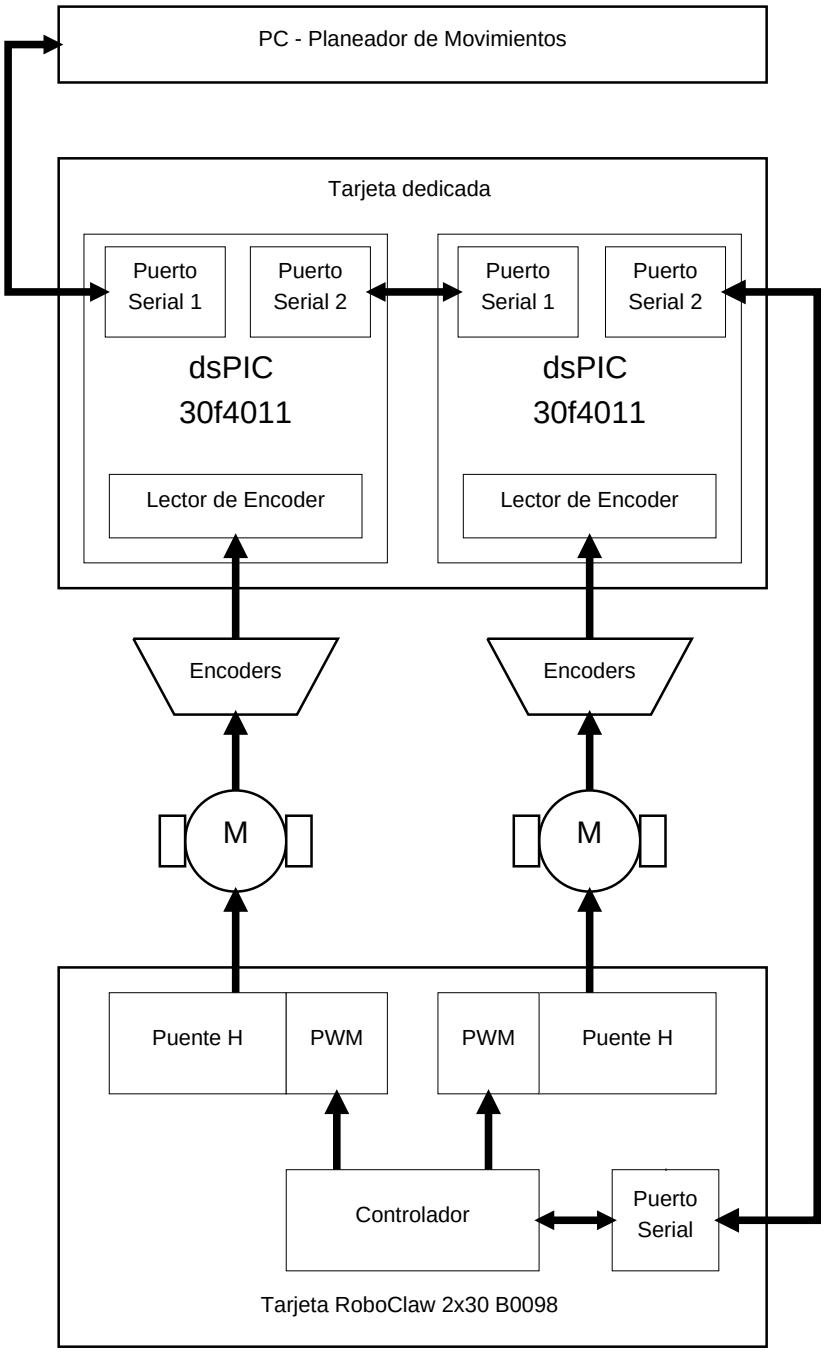


Figura 3.2 Diagrama de comunicacion de la base móvil.

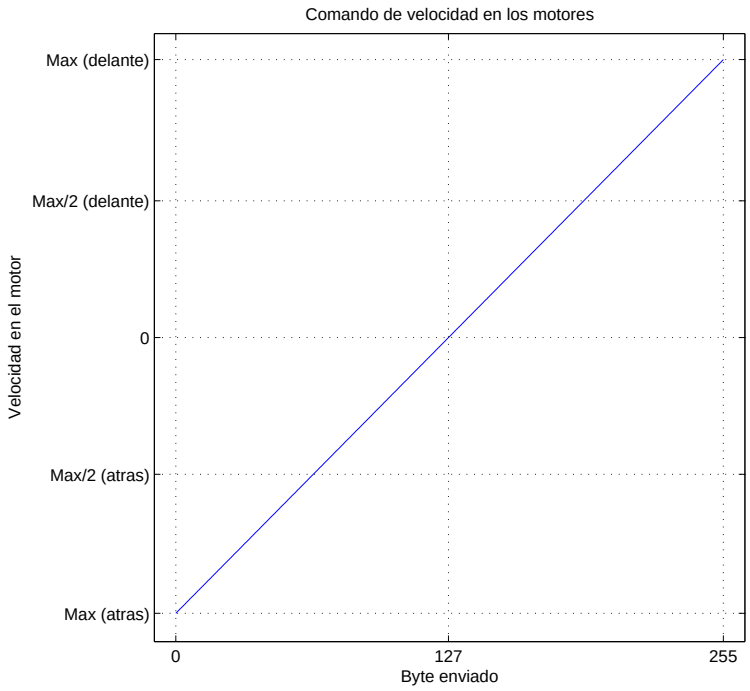


Figura 3.3 Velocidades del motor establecidas por comando.

3.2 El simulador

Para la creación del simulador, se utilizó el lenguaje de programación orientado a objetos (OOP, por sus siglas en ingles) C#. Este lenguaje, además de ser el más utilizado para la creación de los módulos que controlan al robot Justina, presenta las siguientes características (Watson y cols., 2012) que fueron tomadas en cuenta para su elección final como lenguaje para desarrollo del simulador:

1 Sintaxis simple y fácil de aprender.

2 Es más robusto que otros lenguajes, por ejemplo C++.

3 La fase de depuración es muy sencilla.

4 Puede ser compilado y ejecutado en varios sistemas operativos (Windows, Linux y MacOS).

5 Se puede utilizar una gran cantidad de funciones integradas en la plataforma .NET.

6 Existen librerías para la utilización del hardware que utiliza el robot Justina y para la integración con su sistema principal desarrollados por el equipo del laboratorio de Bioróbotica.

Un diagrama de la estructura del software desarrollado, con detalle en las clases del simulador y sus relaciones puede observase en la figura 3.4.

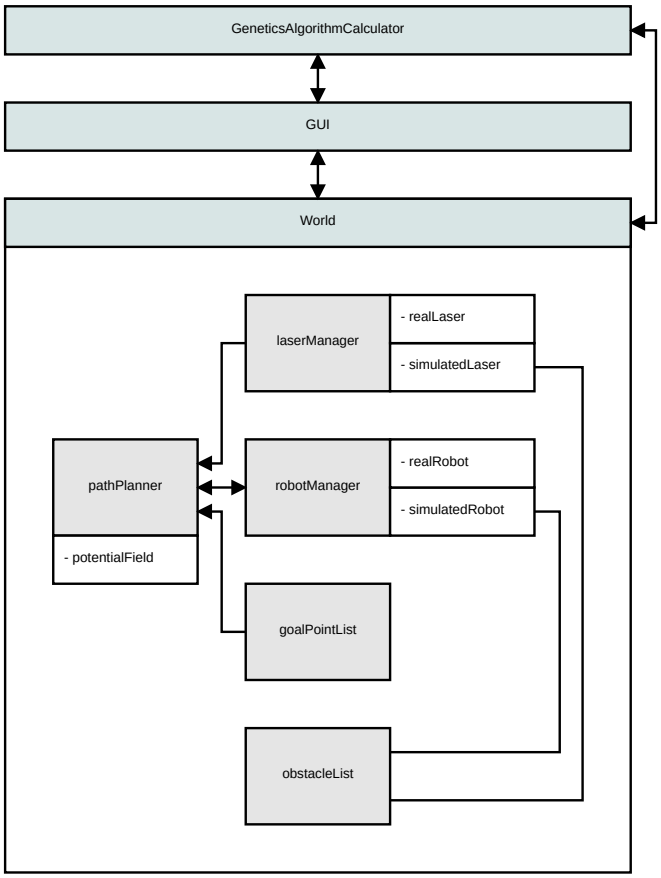


Figura 3.4 Diagrama de la estructura del software.

Cabe aclarar que el programa desarrollado esta escrito en ingles, por lo que todos los nombres de métodos y clases mencionados están en dicho idioma. Esto es porque todos los programas realizados para el proyecto Justina serán compartidos a través de Internet para poder ser verificados y utilizados por cualquier persona de cualquier país.

3.2.1 La interfaz gráfica

Se desarrolló una interfaz gráfica, o GUI (Graphical User Interface, por sus siglas en inglés) con la intención de facilitar el uso de este software. Ésta fue desarrollada utilizando las librerías de la plataforma *.NET*. Dentro de esta interfaz es posible crear una mapa del entorno de trabajo generando obstáculos mediante *drag and drop*, es decir, mediante la acción de presionar el botón del ratón, arrastrarlo para dimensionar el obstáculo y soltar para generarlo. De la misma manera se pueden mover los objetos que existen en el mapa como la posición del robot y los puntos de una trayectoria. Este mapa y las características de los objetos en él pueden ser guardados en un archivo XML para futuras pruebas. La interfaz gráfica se puede observar en la figura 3.5.

3.2.2 Los obstáculos

Los obstáculos son representaciones de los objetos físicos que impedirán el paso del robot durante una trayectoria. Ejemplos de estos son puertas, paredes, muebles y personas. Se creó la clase *Obstacle* para definir sus características y métodos.

En el presente trabajo, los obstáculos son simulados como rectángulos en el mapa. Esta simple representación es suficiente para hacer una aproximación de todos los objetos que pueden estar en la trayectoria por la que se desplaza el robot y permite, en gran medida, simplificar los cálculos para simular las relaciones entre estos y el robot.

Para construir un obstáculo dentro del programa es necesario que el usuario lo dibuje en el mapa. Los rectángulos son creados obteniendo la posición de su centro y las medidas del largo y ancho de sus lados.

Como se verá más adelante, muchos de los cálculos realizados requieren la posición de sus vértices, por lo que, cuando un objeto de la clase *Obstacles* es creado, la posición de sus cuatro vértices es calculada y guardada dentro de un arreglo de cuatro vectores, mediante el método *UpdateVertices*. De igual manera, cuando una de sus propiedades es modificada, se re-calculan los vértices con este método.

3.2.3 El robot

La clase robot define los parámetros que son necesarios para calcular la posición del robot utilizando las lecturas de los encoder, cuando se trata del robot real, o resolviendo la ecuación del movimiento, cuando se simula, para el tipo de robot que se

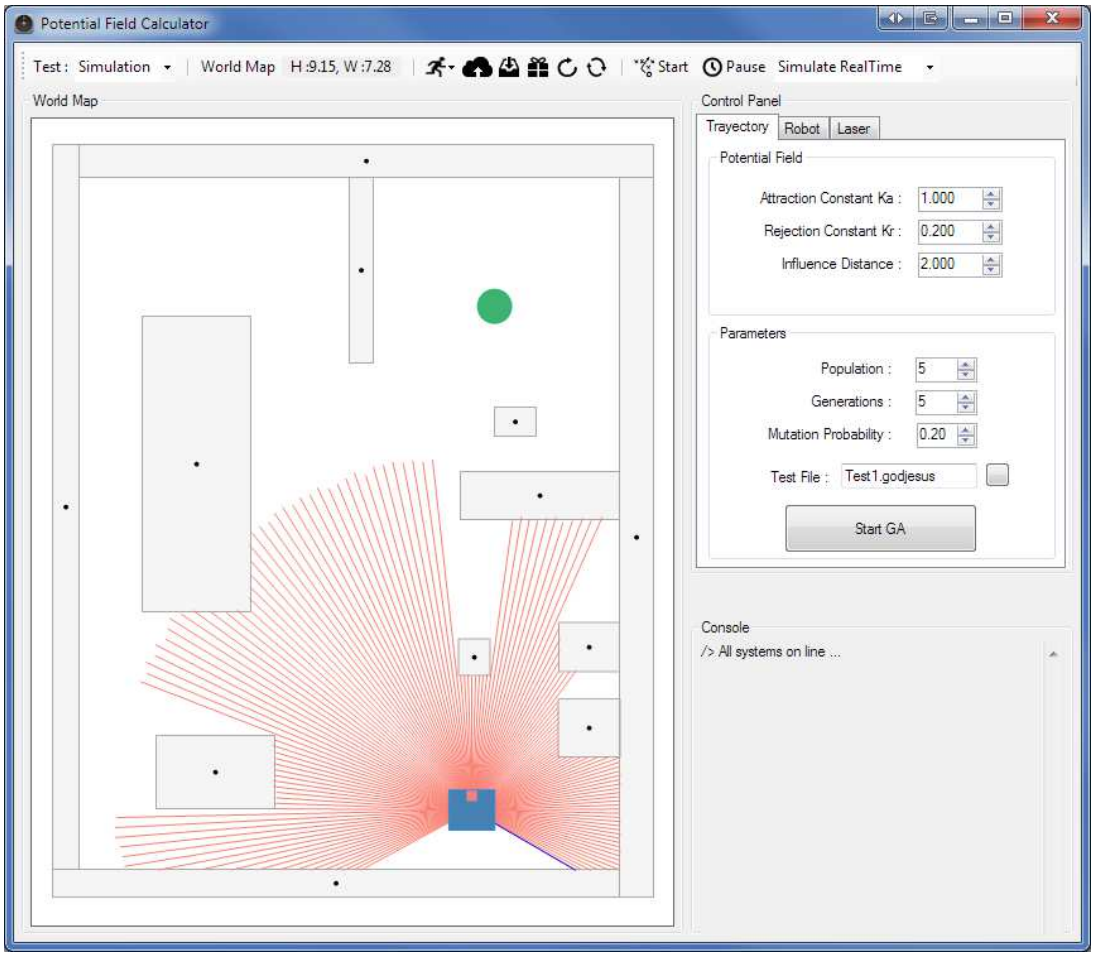


Figura 3.5 Interfaz gráfica del simulador.

utiliza, en este caso, un robot de par diferencial. En ambos casos, el estado del robot esta definido por sus coordenadas en el plano XY y su orientación θ.

Parámetros del robot

Los parámetros requeridos por el programa y los valores específicos del robot Justina son:

- Pulsos de encoder por giro de las ruedas, $PPR = 8000[ppv]$
- Radio de las ruedas, $r = 0.0935[m]$
- Ancho de la base, $a = 0.52[m]$
- Largo de la base, $l = 0.45[m]$

Simulación de la cinemática

La simulación del movimiento de la base se genera resolviendo la ecuación diferencial del modelo cinemático de los robots de tipo par diferencial. Al resolver esta ecuación, utilizando las velocidades enviadas a las llantas como entradas, se puede obtener la posición del robot y su orientación, en un determinado instante de tiempo, esto es:

$$\begin{aligned} \dot{x} &= \frac{v_d + v_i}{2}\cos\theta \\ \dot{y} &= \frac{v_d + v_i}{2}\sin\theta \\ \dot{\theta} &= \frac{v_d - v_i}{L} \end{aligned}$$

Esta ecuación es una ecuación diferencial ordinaria de primer orden, la cual puede ser resuelta utilizando los métodos de Runge-Kutta. Estos, desarrollados alrededor del año 1990, son un conjunto de métodos iterativos para la resolución numérica de ecuaciones diferenciales ordinarias, de la forma:

$$\dot{y} = f(t, y) \quad \text{con} \quad y_0 = f(t_0, y_0) \tag{3.2}$$

De manera que al utilizar este método para resolver la ecuación 3.2, se puede obtener el valor siguiente del sistema conociendo el estado estado actual y las entrada en un periodo de tiempo determinado. Esto es:

$$y_{n+1} = f(t_n, y_n) \tag{3.3}$$

Al resolver el sistema con este método, el valor que determina el estado siguiente del sistema para los robots de par diferencial, tomando en cuenta que el tiempo no es una variable explicita en la dinámica del sistema.

$$\begin{aligned} x_{n+1} &= f(v_d, v_i, \theta_n) \\ y_{n+1} &= f(v_d, v_i, \theta_n) \\ \theta_{n+1} &= f(v_d, v_i, \theta_n) \end{aligned} \tag{3.4}$$

Esto quiere decir que se puede conocer la posición del robot en el plano XY y su orientación θ después de enviarle una determinada velocidad a las llantas, si se conoce su posición y orientación inicial y los valores de velocidad enviados. Usando esto se simularon los movimientos del robot.

De entre el conjunto de métodos de Runge-Kutta, se eligió el desarrollado por Bogacki y Shampine (1996) por ser un método sencillo y eficiente de orden tres. Este establece que si y_n es la solución del sistema en el tiempo t_n y $\Delta t = t_{n+1} - t_n$, entonces se puede obtener y_{n+1} de la siguiente forma:

$$\begin{aligned} k_1 &= f(y_n) \\ k_2 &= f(y_n + \frac{1}{2}\Delta t k_1) \\ k_3 &= f(y_n + \frac{3}{4}\Delta t k_2) \\ y_{n+1} &= y_n + \frac{2}{9}\Delta t k_1 + \frac{1}{3}\Delta t k_2 + \frac{4}{9}\Delta t k_3 \end{aligned} \tag{3.5}$$

Como se puede observar, para utilizar este método, sólo es necesario realizar tres evaluaciones de la ecuación del sistema y sumar sus resultados ponderados. Si se elige un Δt suficientemente pequeño, la aproximación de este método será suficiente para las características del sistema.

Para su implementación, se creo la clase estática *ODEsolver*, la cual resuelve esta ecuación mediante el método mencionado anteriormente. Pensando en la escalabilidad

del simulador, esta puede recibir por parámetros cualquier tipo de modelo siempre que este sea un mapeo de $R^n \longrightarrow R^n$. Esto se logra gracias al uso de delegados de lenguaje C#. Un delegado es una referencia a un método. Cuando se asigna un método a un delegado, se comporta exactamente como el método. Son similares a los punteros a función utilizados en C++, pero poseen seguridad de tipos y pueden ser pasados como parámetro a otra función.

Esto permite que, en determinado momento, si se quiere simular un robot con diferente locomoción para su optimización, como puede ser una base de tipo Ackerman, sólo es necesario crear el método con este nuevo modelo, sin modificar otra parte del programa.

Para simular el movimiento del robot para un par de velocidades establecidas durante 100 ms, acorde al tiempo de respuesta de la base del robot, se utilizó un paso de simulación de 1 ms. Esto quiere decir que se cálcula el estado siguiente del robot al establecer por 1 ms las velocidades de las llantas 100 veces, antes de determinar la posición final.

El algoritmo 1 es el utilizado para calcular la siguiente posición del robot, dadas un par de velocidades, con el que se simulan los movimientos del robot.

Simulación de colisiones

Además de la simulación de los movimientos del robot, es necesario simular la relación del robot con los objetos en su entorno. Para el presente trabajo, dado que se pretende mejorar la evasión de obstáculos, es necesario saber cuando el robot colisionó con alguno de estos.

El número de colisiones en un cada trayectoria es una de las características más importantes que sea desean mejorar en el robot Justina, puesto que, cualquier colisión puede dañar al robot o a un humano si éste estuviera en su camino.

Como se mencionó anteriormente, los obstáculos son representados como rectángulos en el mapa. Igualmente, el robot, dada sus características geométricas, el área que ocupa puede ser representada por un rectángulo de las mismas dimensiones que la base del robot. Esto reduce la detección de colisión entre el robot y los obstáculos al problema de intersección entre polígonos en dos dimensiones.

Para detectar esta intersección entre el rectángulo que representa al robot y los rectángulos que representan los obstáculos en el mapa, se utiliza el teorema del eje separador (Separating axis theorem o SAT, por sus siglas en ingles). El SAT establece que, dados dos objetos convexos, si existe una recta tal que, las proyecciones de los

Algoritmo 1: Cálculo de la posición siguiente del robot

entrada Velocidad de la llanta izquierda V_l
entrada Velocidad de la llanta derecha V_d
entrada Posición del robot P_R
entrada Orientación del robot θ
entrada Diámetro del robot L
salida Vector de estado del robot $\begin{bmatrix} x & y & \theta \end{bmatrix}$

$\Delta t \leftarrow 0.001$
para $i = 0$ **hasta** $i = 100$ **hacer**
$Y_n[1] = \frac{V_l+V_d}{2}\cos(\theta)$
$Y_n[2] = \frac{V_l+V_d}{2}\sin(\theta)$
$Y_n[3] = \frac{V_l-V_d}{L}$
$k_1[1] \leftarrow Y_n[1]$
$k_1[2] \leftarrow Y_n[2]$
$k_1[3] \leftarrow Y_n[3]$
$Y_{n1}[1] = Y_n[1] + \frac{1}{2}\Delta t k_1[1]$
$Y_{n1}[2] = Y_n[2] + \frac{1}{2}\Delta t k_1[2]$
$Y_{n1}[3] = Y_n[3] + \frac{1}{2}\Delta t k_1[3]$
$k_2[1] = \frac{V_l+V_d}{2}\cos(Y_{n1}[3])$
$k_2[2] = \frac{V_l+V_d}{2}\sin(Y_{n1}[3])$
$k_2[3] = \frac{V_l-V_d}{L}$
$Y_{n2}[1] = Y_n[1] + \frac{3}{4}\Delta t k_2[1]$
$Y_{n2}[2] = Y_n[2] + \frac{3}{4}\Delta t k_2[2]$
$Y_{n2}[3] = Y_n[3] + \frac{3}{4}\Delta t k_2[3]$
$k_3[1] = \frac{V_l+V_d}{2}\cos(Y_{n2}[3])$
$k_3[2] = \frac{V_l+V_d}{2}\sin(Y_{n2}[3])$
$k_3[3] = \frac{V_l-V_d}{L}$
$Y_{n+1}[1] = Y_n[1] + \frac{2}{9}\Delta t k_1[1] + \frac{1}{3}\Delta t k_2[1] + \frac{4}{9}\Delta t k_3[1]$
$Y_{n+2}[2] = Y_n[2] + \frac{2}{9}\Delta t k_1[2] + \frac{1}{3}\Delta t k_2[2] + \frac{4}{9}\Delta t k_3[2]$
$Y_{n+3}[3] = Y_n[3] + \frac{2}{9}\Delta t k_1[3] + \frac{1}{3}\Delta t k_2[3] + \frac{4}{9}\Delta t k_3[3]$
$x \leftarrow Y_{n+1}[1]$
$y \leftarrow Y_{n+1}[2]$
$\theta \leftarrow Y_{n+1}[3]$
regresa $\begin{bmatrix} x & y & \theta \end{bmatrix}$

objetos sobre esta recta no se intersectan, los objetos no se intersectan. Esta recta es llamada eje separador y, además, las rectas trasladadas serán también ejes separadores (Schneider y Eberly, 2003).

Si los objetos son rectángulos, como es el caso del robot y de los obstáculos, sólo es necesario probar que el eje separador es una recta normal a alguno de los lados de los objetos. Ésto puede comprobarse fácilmente entendiendo que sólo existen tres formas en que dos polígonos convexos pueden tocarse sin que exista intersección, estos son: un lado con un lado, un lado con un vértice y un vértice con un vértice, como puede verse en la figura 3.6. Siendo así, si los objetos están separados, deberá existir una linea perpendicular a alguno de los lados de alguno de los objetos que los separa y el eje separador sera normal a éste.

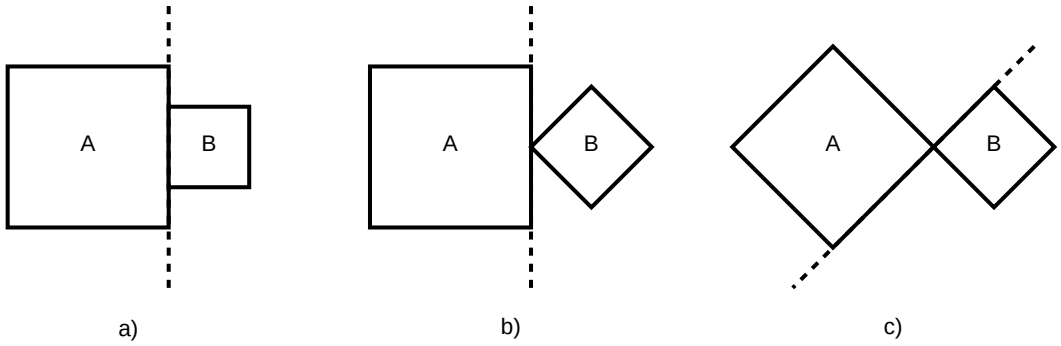


Figura 3.6 Formas en que se pueden tocar dos polígonos sin intersectarse.

En el caso del simulador, dado que, tanto el robot como los obstáculos son polígonos rectangulares en donde existen 2 lados perpendiculares a los otros dos, y dado que cualquier recta trasladada también es un eje separador, sólo es necesario comprobar la existencia del eje separador cuatro veces por obstáculo: para dos lados del robot y para dos lados del obstáculo. La figura 3.7 es un ejemplo de lo mencionado anteriormente, en el cual existen dos ejes separadores, por tanto las proyecciones están separadas.

Los vectores directores de las rectas candidatas a ser el eje separador se obtienen de la siguiente manera: para los obstáculos, dado que siempre son representados por rectángulos paralelos a los ejes coordenados, es decir no orientados, son los vectores unitarios:

$$\hat{\mathbf{O}}_W = (1,0,0) \tag{3.6}$$

$$\hat{\mathbf{O}}_H = (0,1,0) \tag{3.7}$$

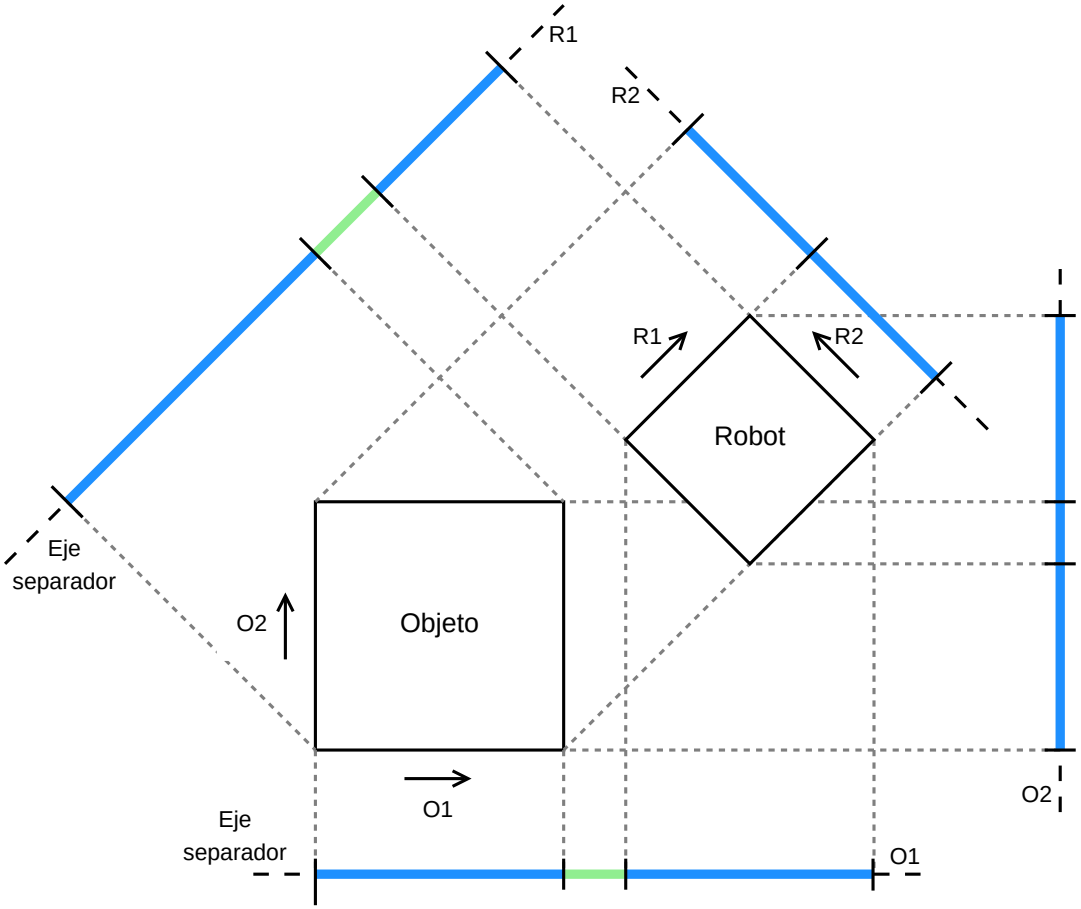


Figura 3.7 Ejemplo de eje separador

Los vectores directores de los lados del robot, si θ_R es la orientación del robot, son;

$$\hat{\mathbf{R}}_W = (\cos\theta_R, \sin\theta_R, 0) \tag{3.8}$$

$$\hat{\mathbf{R}}_H = \left(\cos(\theta_R + \frac{\pi}{2}), \sin(\theta_R + \frac{\pi}{2}, 0\right) \tag{3.9}$$

Una vez que se tienen los vectores directores de los candidatos a eje separador, es necesario analizar si al menos en una de estas cuatro rectas, las proyecciones del robot y del obstáculo no se traslapan. Si esto sucede, se puede afirmar que entre el obstáculo y el robot no existe colisión.

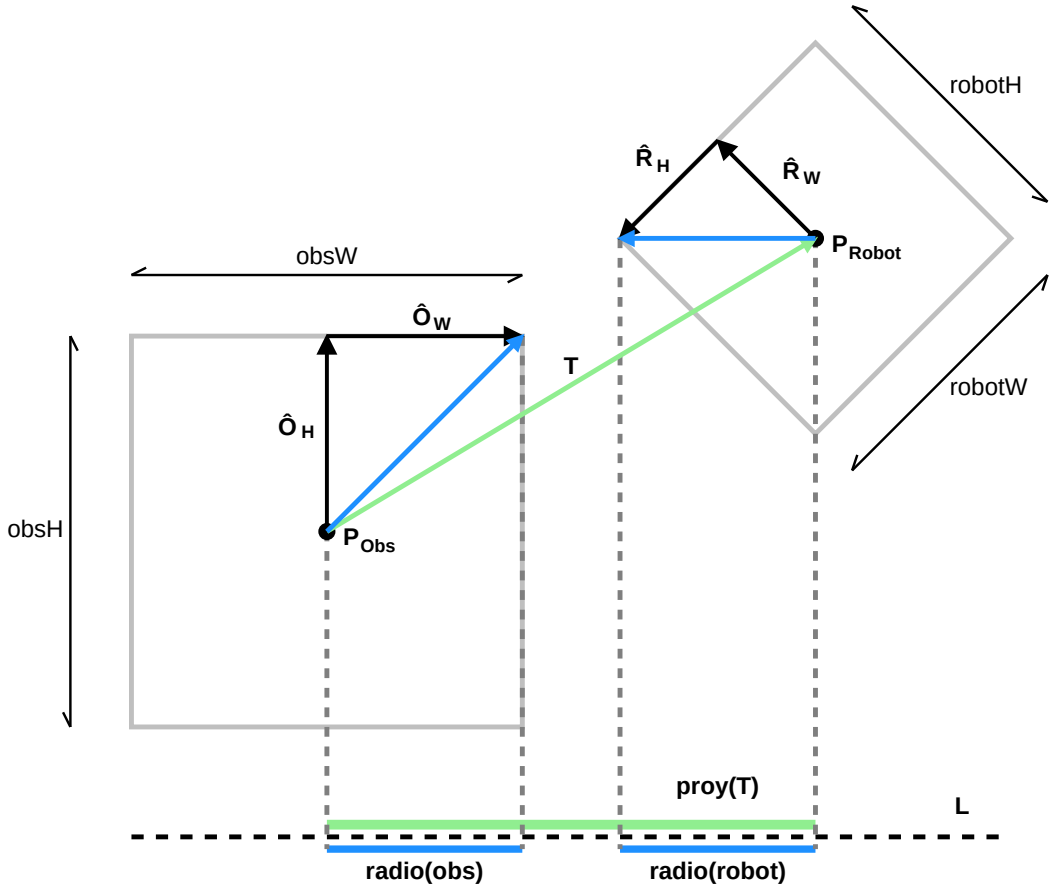


Figura 3.8 Robot con configuración diferencial.

Para verificar el traslape de las proyecciones, se utiliza el método denominado "Prueba rápida de traslape" (Fast Overlap Test), algoritmo presentado por Gottschalk, Lin, y Manocha (1996), el cual reduce el número de operaciones, utilizando sólo los radios de los objetos, es decir, la mitad de las proyecciones de los objetos. Utilizando la figura 3.8, si L es un vector director de los candidatos a eje separador, entonce el radio de un obstáculo y el radio del robot son, respectivamente:

$$radioObs = \left|\frac{obsW}{2}\hat{\mathbf{O}}_W \bullet \mathbf{L}\right| + \left|\frac{obsH}{2}\hat{\mathbf{O}}_H \bullet \mathbf{L}\right|$$

$$radioRobot = \left|\frac{robotW}{2}\hat{\mathbf{R}}_W \bullet \mathbf{L}\right| + \left|\frac{robotH}{2}\hat{\mathbf{R}}_H \bullet \mathbf{L}\right|$$

En este algoritmo, es necesario obtener el valor absoluto de la proyección del vector que va desde el centro del robot hasta el centro del obstáculo en el eje que será verificado como eje separador, esto es, si P_{obs} es la posición del centro del obstáculo y P_{robot} es la posición del centro del robot, entonces:

$$\begin{aligned} \mathbf{T} &= \mathbf{P}_{Robot} - \mathbf{P}_{Obs} \\ proyT_L &= \mathbf{T} \bullet \mathbf{L} \end{aligned} \tag{3.10}$$

Si el valor absoluto de $proyT_L$ es mayor que la suma de los radios de los objetos, las proyecciones no se traslapan y la recta en la que se proyectaron los objetos es un eje separador, por lo que al existir una recta separadora, los objetos no se intersectan. Si no es así, se necesitan verificar los otros tres posibles candidatos a eje separador. Si en los 4 casos esto no se cumple, entonces los objetos se intersectan y el robot habrá colisionado con el obstáculo del mapa. Esto es, si $i = 1,2,3,4$ es cada uno de los vectores directores de las ecuaciones 3.6 y 3.8, entonces no existirá colisión con un obstáculo si la siguiente ecuación se cumple para cualquier valor de i:

$$\left|proyT_{L_i}\right| > radioObs + radioRobot$$

Estas cuatro comprobaciones se hacen por cada obstáculo, por lo que el número de operaciones del método es dependiente del número de obstáculos. El algoritmo 2 es el implementado para la detección de colisiones en el simulador.

Simulación del movimiento del robot

Utilizando los métodos vistos anteriormente, se realiza la simulación completa del robot utilizando el algoritmo 3.

Algoritmo 2: Detección de colisiones

entrada Arreglo de obstáculos $Obstaculos[]$
entrada Dimensiones del robot
entrada Posición del robot P_R
salida **verdadero**si existe colisión, **falso**si no existe colisión
  $r_H \leftarrow$ Mitad del largo del robot
  $r_W \leftarrow$ Mitad del ancho del robot
  $eje[1] \leftarrow \mathbf{R_H} = \left(\cos(\theta_R), \quad \sin(\theta_R), \quad 0\right)$
  $eje[2] \leftarrow \mathbf{R_W} = \left(\cos(\theta_R - \frac{\pi}{2}), \quad \sin(\theta_R - \frac{\pi}{2}), \quad 0\right)$
  $eje[3] \leftarrow \mathbf{O_H} = \left(1, \quad 0, \quad 0\right)$
  $eje[4] \leftarrow \mathbf{O_W} = \left(0, \quad 1, \quad 0\right)$
  para todo Obstáculo O en $Obstaculos[]$ **hacer**
    $ejeSep \leftarrow$ **falso**
    $eje = P_R - P_O$
    $o_W \leftarrow$ Mitad del ancho del obstáculo O
    $o_H \leftarrow$ Mitad del largo del obstáculo O
    para $i = 1$ **hasta** $i = 4$ **hacer**
      $proy_{eje} = eje \bullet eje[i]$
      $radio_O = \left|o_W(\mathbf{O_W} \bullet eje[i])\right| + \left|o_H(\mathbf{O_H} \bullet eje[i])\right|$
      $radio_R = \left|r_W(\mathbf{R_W} \bullet eje[i])\right| + \left|r_H(\mathbf{R_H} \bullet eje[i])\right|$
      si $\left|proy_{eje}\right| < radio_O + radio_R$ **entonces**
        $ejeSep \leftarrow$ **verdadero**
        Probar con siguiente Obstáculo
    si $ejeSep =$ **falso entonces**
      regresa **verdadero**
  regresa **falso**

Algoritmo 3: Simulación del robot

entrada Velocidades de las llantas izquierda V_l y derecha V_d
  Establecer velocidades V_l y V_r
  Calcular estado *siguiente del robot*
  Detectar colisión
  si Colisión Detectada **entonces**
    Mantener estado del robot
  si no
    Establecer estado del robot como estado *siguiente del robot*

3.2.4 El sensor láser

El sensor láser descrito en la sección 3.1 entrega al sistema del robot un arreglo de lecturas que forman un arco con centro en el láser. En este, cada lectura representa un paso angular con la información de la distancia donde el haz de láser rebota. Esta representa parte de un obstáculo detectado en el espacio de trabajo del robot. Aproximadamente cada 100 ms, el sensor láser envía un arreglo de 683 lecturas, en un angulo de detección de 240°. Cada lectura esta espaciada por 0.36°.

Parámetros del Láser

El láser utilizado cuenta con parámetros especificados en su hoja de datos, que servirán para la simulación de las lecturas que entrega. Estos parámetros son:

- Número de Lecturas = 683
- Angulo de detección = 240°
- Resolución angular = 0.36°
- Distancia máxima = 4000 [mm]
- Tiempo de lectura = 100 [ms/lectura]

Las lecturas del láser proporcionan tres datos: la distancia del haz del láser, el ángulo de la lectura y si la lectura fue errónea o no. Como se puede ver en la figura 3.9, para el arreglo de lecturas entregado por el láser, la lectura $L_{N/2}$ se encuentra a 0° y sale directamente hacia el frente del sensor láser. A partir de esta lectura y en sentido contrario de las manecillas del robot se encuentran las lecturas con ángulos positivos hasta llegar a la ultima lectura L_N, la cual tienen un angulo de +120°. En

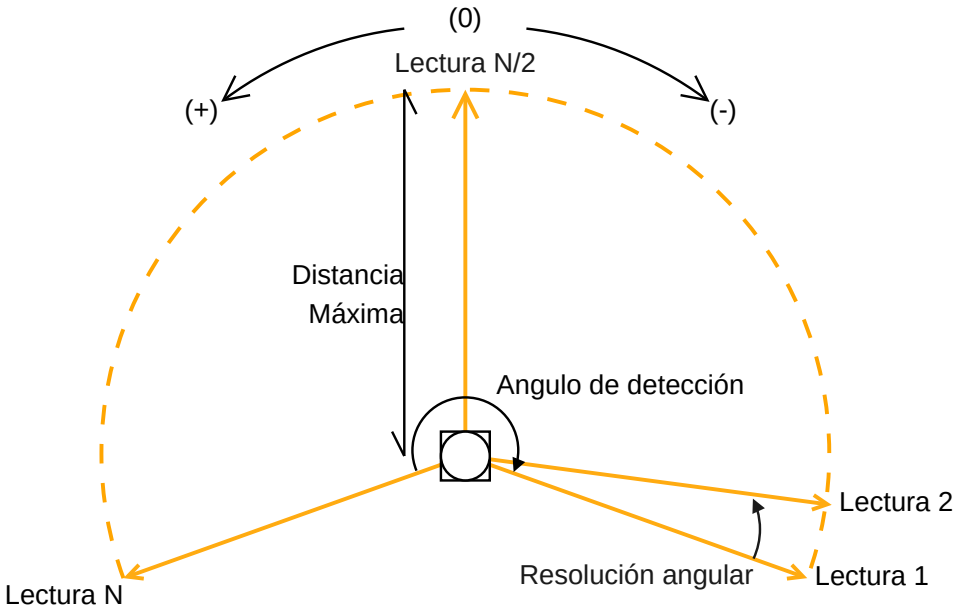


Figura 3.9 Parámetros del láser.

sentido contrario de las manecillas de reloj, a partir de la lectura $L_{N/2}$ se encuentran las lecturas con ángulos negativos hasta llegar a la primera lectura del arreglo L_0, la cual tienen un ángulo de $-120°$.

Además de estos parámetros, la hoja de especificaciones define la precisión de las lecturas, la cual es dependiente de la distancia a la que se encuentran los objetos detectados. Esto es:

- Distancia = [0.02m , 1m] , precisión $\pm 0.03m$
- Distancia = [0.20m , 4m] , precisión $\pm 3\,\%$

Simulación de las lecturas del láser

Como se mencionó anteriormente, el funcionamiento del láser se basa en la generación de haces para la detección de objetos en el ambiente. Dado que estos haces tienen una distancia máxima de detección pueden ser interpretados como segmentos de una recta. Igualmente, los obstáculos, al ser representados como polígonos rectangulares, pueden ser representados como cuatro segmentos de recta, uno representando cada lado. Por tanto, el algoritmo más simple para obtener la intersección entre los obstáculos y los haces del láser, es iterar a través de cada lado del polígono y pro-

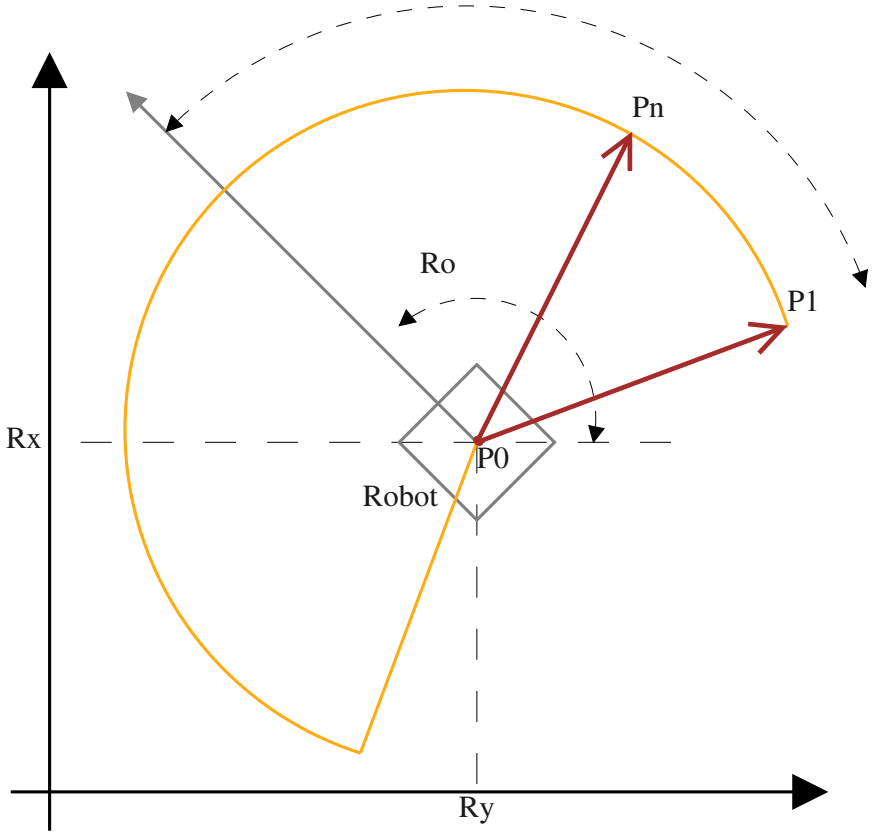


Figura 3.10 Haces del láser como segmentos de recta.

bar la intersección entre este y el segmento de línea que representa el haz del láser (Schneider y Eberly, 2003). Esto se hace para cada haz de láser generado.

El láser como segmentos de recta

Para obtener la ecuación del segmento de recta $\langle P_0, P_n \rangle$ que define un haz del láser en el mapa, se obtienen los puntos inicial y final del segmento. Por simplicidad, el sensor láser se posicionó en el centro del robot, por lo que el punto inicial de todo segmento de rayo del láser será la posición del robot. Entonces el punto inicial será $P_0 = P_R$.

Para el punto final P_n, si θ_r es la orientación del robot, d_{max} la distancia máxima del sensor, θ_{max} es el ángulo de detección, r_a la resolución angular y n el número de lectura, como se puede ver en la figura 3.10, sus componentes serán:

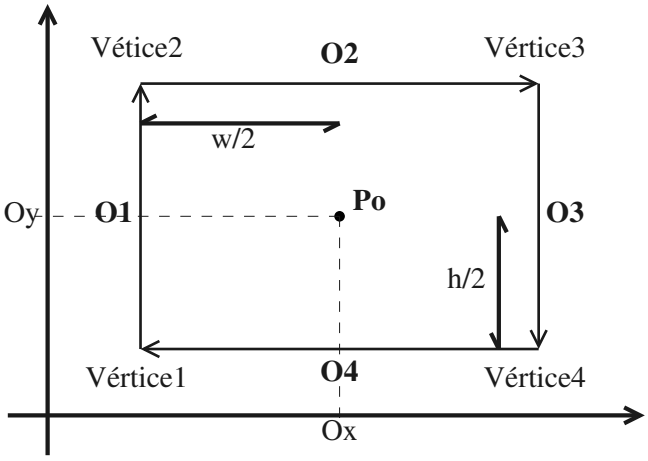


Figura 3.11 Representación de obstáculos utilizando segmentos de recta.

$$
\begin{aligned}
P_{nX} &= d_{max}\cos\left(\theta_R - \frac{\theta_{max}}{2} + r_a(n-1)\right) + P_{rX} \\
P_{nY} &= d_{max}\sin\left(\theta_R - \frac{\theta_{max}}{2} + r_a(n-1)\right) + P_{rY}
\end{aligned}
\tag{3.11}
$$

Con estos puntos, se obtiene la ecuación paramétrica del segmento de linea de los haces del láser. Esta ecuación es de la forma $X(t) = P + t\vec{d}$ donde $t \in [0,1]$ por tratarse de un segmento de recta. En esta ecuación, $\vec{d}$ es el vector director del segmento de recta, obtenido mediante $\vec{d}_P = P_n - P_0$. Entonces, para los rayos del láser, esto queda como:

$$
X(t)_n = P_0 + t\vec{d}_P \quad \text{donde} \quad \vec{d}_P = (P_n - P_0) \tag{3.12}
$$

Segmentos de los obstáculos

Los obstáculos están compuestos por cuatro segmentos de recta. Cada uno de estos segmentos es un lado del rectángulo que representa los obstáculos, para los cuales cada vértice es el punto inicial y final de estos segmentos. Utilizando la figura 3.11, si w es el ancho del obstáculo y h es la altura, se obtienen los vectores de los cuatros vértices de la siguiente manera.

$$\begin{aligned} V_1 &= P_O + \left(-\frac{w}{2}, -\frac{h}{2}, 0\right) \\ V_2 &= P_O + \left(-\frac{w}{2}, \frac{h}{2}, 0\right) \\ V_3 &= P_O + \left(\frac{w}{2}, \frac{h}{2}, 0\right) \\ V_4 &= P_O + \left(\frac{w}{2}, -\frac{h}{2}, 0\right) \end{aligned} \tag{3.13}$$

Teniendo los vértices de los obstáculos, se pueden obtener las ecuaciones paramétricas utilizando el mismo método que para los rayos del láser. Al ser segmentos, se define la misma condición para el parámetro $t \in [0, 1]$.

$$\begin{aligned} O_1(t) &= V_1 + t_1\vec{d}_1 \quad \text{donde} \quad \vec{d}_1 = (V_2 - V_1) \\ O_2(t) &= V_2 + t_2\vec{d}_2 \quad \text{donde} \quad \vec{d}_2 = (V_3 - V_2) \\ O_3(t) &= V_3 + t_3\vec{d}_3 \quad \text{donde} \quad \vec{d}_3 = (V_4 - V_3) \\ O_4(t) &= V_1 + t_4\vec{d}_4 \quad \text{donde} \quad \vec{d}_4 = (V_1 - V_4) \end{aligned} \tag{3.14}$$

Las ecuaciones paramétricas de los lados del obstáculo se calculan desde el momento en que se construye un objeto de tipo *Obstacle* en el programa y se recalculan cuando un obstáculo cambia de posición en el mapa, para reducir el número de operaciones realizadas durante la simulación del láser.

Intersección entre segmentos

Las ecuaciones anteriores se utilizan para simular la información de un solo haz de láser sobre un obstáculo del mapa. Para este haz, se debe comprobar si existen puntos de intersección entre este y cada lado del obstáculo $O_n(t)$ para $n = 1, 2, 3, 4$.

Como se observa en la Figura 3.12, un único segmento de recta de láser, puede tener varios puntos de intersección con uno o varios obstáculos, por lo que es necesario, probar la intersección entre un rayo y cada obstáculo del mapa y escoger el punto de intersección más cercano al punto inicial del haz del láser.

Para saber si existe intersección entre dos segmentos de recta y, si existe, calcular el punto $I(x, y)$ donde éstas se intersectan, se utiliza la ecuación general de la forma

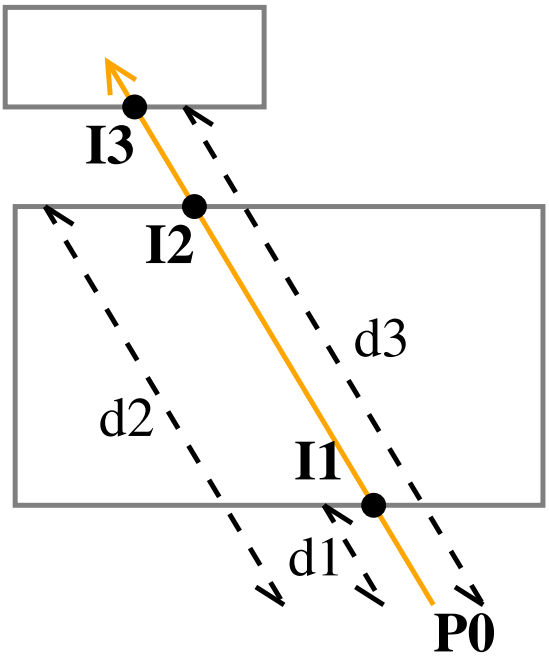


Figura 3.12 Intersección entre un segmento de láser y dos obstáculos.

$ax + by + c = 0$ de las rectas de ambos segmentos. Obteniendo estas ecuaciones, se forma un sistema de dos ecuaciones con dos incógnitas, el cual se resuelve utilizando el método de determinantes.

Para obtener las ecuaciones generales desde la forma paramétrica $X(t) = P + t\vec{d}$ se utiliza la siguiente ecuación:

$$-d_x x + d_y y + (P_x d_y - P_y d_x) = 0 \tag{3.15}$$

donde:

$$\begin{aligned} a &= -d_x \\ b &= d_y \\ c &= P_x d_y - P_y d_x \end{aligned} \tag{3.16}$$

Una vez que se tienen las dos ecuaciones generales de los segmentos de recta, se verifica que el sistema sea compatible determinado, es decir, que exista un único punto de intersección. Para reducir los cálculos, se ignora el caso en donde el sistema

es compatible indeterminado, es decir cuando las lineas se intersectan en infinitos puntos. Este último caso se presenta cuando un rayo paralelo a un lado rebota en un vértice. Este caso es improbable en la vida real por lo que los cálculos no se consideran necesarios.

Si tenemos las ecuaciones generales de un haz de láser $a_P x + b_P y + c_P = 0$ y las ecuaciones de los segmentos de obstáculos $a_{O_n} x + b_{O_n} y + c_{O_n} = 0$, el determinante del sistema sera:

$$det_n = a_P b_{O_n} - b_P a_{O_n} \tag{3.17}$$

Si $det_n \neq 0$ entonces existe una única solución, y las rectas por las que pasan los segmentos se cruzan en un punto $I(x, y)$. Las componentes del punto de intersección son:

$$\begin{aligned} I_x &= \frac{1}{det}(b_P c_O - c_P b_O) \\ I_y &= \frac{1}{det}(c_P a_O + a_P c_O) \end{aligned} \tag{3.18}$$

Si existe un punto de intersección, se debe comprobar que este pertenece a ambos segmentos. Para esto, se utiliza la forma paramétrica de los segmentos obtenidas anteriormente. Para que un punto pertenezca a un segmento de recta, el valor de t para ambos componentes debe de ser un valor entre 0 y 1, es decir $t \in [0, 1]$. Para esto se despeja t de la ecuación paramétrica y se evalúa de la siguiente manera:

$$\begin{aligned} t_x &= \frac{I_x - P_{0x}}{d_x} \\ t_y &= \frac{I_y - P_{0y}}{d_y} \end{aligned} \tag{3.19}$$

Obteniendo estos valores, el punto pertenecerá a un segmento de recta si $0 \leq t_x \leq 1$ y $0 \leq t_y \leq 1$.

Para verificar la intersección entre segmentos de recta, se utilizó el algoritmo 4.

Una vez que se tiene todos los puntos de intersección entre un haz de láser y todos los obstáculos, se selecciona como punto final el punto de intersección más cercano al

Algoritmo 4: Intersección entre segmentos

entrada Segmento 1 con ecuación general $A_1x + B_1y + C_1 = 0$ y paramétrica $X(t_1) = P_{01} + t_2\mathbf{u_1}$

entrada Segmento 2 con ecuación general $A_2x + B_2y + C_2 = 0$ y paramétrica $X(t_2) = P_{02} + t_2\mathbf{u_2}$

salida **falso** si los segmentos no se intercectan; si se intersectan, **I** punto donde se intersectan

$det = A_1B_2 - B_1A_2$

I ← punto de intersección

si $det = 0$ **entonces**

  regresa **falso**

$I_x = \frac{1}{det}(-B_2C_1 + B_1C_2)$

$I_y = \frac{1}{det}(A_2C_1 + A_1C_2)$

$t_{x1} = \frac{I_x - P_{01x}}{u_{1x}}$

$t_{y_1} = \frac{I_y - P_{01y}}{u_{1y}}$

si $(0 \leq t_{x1} \leq 1)$**Y**$(0 \leq t_{y_1} \leq 1)$ **entonces**

  I pertenece al segmento 1

$t_{x2} = \frac{I_x - P_{02x}}{u_{2x}}$

$t_{y_2} = \frac{I_y - P_{02y}}{u_{2y}}$

si $(0 \leq t_{x2} \leq 1)$**Y**$(0 \leq t_{y_2} \leq 1)$ **entonces**

  I pertenece al segmento 2

si **I** pertenece a ambos segmentos **entonces**

  regresa **I**

si no

  regresa **falso**

punto inicial. Esto se hizo comparando las magnitudes de los vector que van del punto inicial del segmento a los puntos de intersección. Con la distancia al punto final, se representa la información de este haz de láser.

Ruido del láser

Con los datos de precisión del sensor láser, se agrega ruido a las lecturas utilizando una distribución uniforme de probabilidad. Para esto, se genera un valor aleatorio $\epsilon \in \mathbb{R}$ donde $-3 \leq \epsilon \leq 3$ el cual representa el porcentaje de error de la distancia D de la lectura.

Simulación del total de lecturas del láser

El láser real entrega un arreglo de objetos de tipo *LaserReadings*. La información que contiene cada objeto de este clase es la distancia de detección *distance* y el ángulo de cada lectura *angle*. Además se indica si un valor de lectura fue errónea.

La simulación de todos los rayos de láser en un mapa con un conjunto de obstaculos se implemento siguiendo el algoritmo 5. En ésta, se puede modificar el angulo de detección, el número de lecturas y la distancia máxima de detección. Así mismo, se puede determinar un porcentaje de lecturas erróneas y la precisión del mismo.

Algoritmo 5: Simulación del sensor láser

entrada Arreglo *Obstáculos[]* de la clase *Obstacles*
entrada Posición del láser $\mathbf{P_L}$
entrada Orientación del láser θ_L
entrada Distancia de detección D
entrada Ángulo de detección α_{max}
entrada Número de lecturas de láser N
entrada Probabilidad de error P_e
salida Arreglo *LecturasLaser[]* de N lecturas
  para $i = 0$ **hasta** N **hacer**
    $error \leftarrow$ Número aleatorio $\in \mathbf{R}$ en el intervalo $[0, P_e]$
    si $error \geq P_e$ **entonces**
      $r_{ang} = \frac{\alpha_{max}}{N}$
      $\alpha = \theta_L - \alpha_{max}$
      $\mathbf{P_f} \leftarrow$ punto final del segmento de láser
      $P_{f_x} = D\cos(\alpha + i * r_{ang}) + P_{Lx}$
      $P_{f_y} = D\sin(\alpha + i * r_{ang}) + P_{Ly}$
      $l \leftarrow$ segmento de linea desde P_L hasta P_f
      $dist_f \leftarrow \infty$
      para todo Obstáculo O **en** *Obstáculos[]* **hacer**
        para todo Segmento S **en** O **hacer**
          Verificar intersección entre S y l
          si Existe intersección en el punto **I entonces**
            $dist = |\mathbf{I} - \mathbf{P_L}|$
            si $dist < dist_f$ **entonces**
              $dist_f \leftarrow dist$
      $ruido \leftarrow$ Número aleatorio $\in \mathbf{R}$ en el intervalo $[-0.97, 1.03]$
      $l_i \leftarrow$ lectura de láser
      $l_{idist} = dist_f * ruido$
      $l_{iang} = \alpha + i * r_{ang}$
      $l_{ierror} \leftarrow$ **falso**
    si no
      $l_i \leftarrow$ lectura de láser
      $l_{idist} = D$
      $l_{iang} = \alpha + i * r_{ang}$
      $l_{ierror} \leftarrow$ **verdadero**
    LecturasLaser[i] $\leftarrow l_i$
  regresa *LecturasLaser[]*

Capítulo 4

Planeación de trayectorias

Utilizando los datos del simulador o los datos del robot real, se tiene la información de los obstáculos cercanos al robot, mediante un arreglo de lecturas de láser, y el estado del robot $\begin{bmatrix} x & y & \theta \end{bmatrix}$ al establecer un par de velocidades a los motores del robot. Con estos datos, se puede utilizar un método reactivo de evasión de obstáculos para fijar una trayectoria deseada, segura y alejada de los obstáculos para el robot. Una vez que se tiene una trayectoria deseada, es necesaria una ley de control que permita reducir el error entre el estado deseado y el actual. Cada uno de estos pasos conlleva fijar parámetros para la obtención de una trayectoria conveniente para las características del robot y del ambiente. En este capítulo se presentan cada una de los operaciones antes mencionadas.

En la Sección 4.1 se desarrolla la implementación de la técnica de evasión de obstáculos utilizada en el presente trabajo denominada Campos Potenciales Artificiales, la cual determina, con cada lectura de láser, una posición deseada.

En la Sección 4.2 se explica la obtención del error entre el robot y la trayectoria deseada y el control utilizado para reducir este error, con el fin de alcanzar el punto establecido por la función de campos potenciales hasta que la posición del robot sea el objetivo.

Con el método de cálculo de campos potenciales y el control de robot, en la sección 4.3, se explica el algoritmo general utilizado para la evasión de obstáculos propuesto y se ilustran algunos ejemplos.

4.1 Campos Potenciales Artificiales

La técnica reactiva de campos potenciales establece tres fuerzas que definen la posición deseada del robot en cada punto del mapa. Estas son la fuerza atractiva F_{atr}, la fuerza de repulsión F_{rep} y la fuerza resultante F_{res} la cual es la suma de estas últimas dos.

En el programa desarrollado se generó la clase *PotentialField* encargada de calcular las tres fuerzas antes mencionadas, utilizando la información del sensor láser, el estado del robot y el punto final al que se desea llegar, denominado *Goal Point*. Además de estos datos, es necesario establecer el valor de las constantes de atracción K_{att}, de repulsión K_{rep} y la distancia de influencia D_{inf}, propiedades de esta clase, para que el campo potencial funcione adecuadamente.

Cabe recordar que la frecuencia de muestreo del sensor láser es de 10 lecturas en cada segundo, por lo que el cálculo de estas fuerzas se realiza cada 100 ms.

4.1.1 Fuerza de atracción

La fuerza de atracción se considera como un vector que se obtiene multiplicando el valor de la constante de atracción, por el vector unitario que va de la posición del robot P_R, hasta la posición del objetivo final, o goal point, P_{GP}.

$$F_{att} = K_{att}\left(\frac{P_{GP} - P_R}{|P_{GP} - P_R|}\right) \tag{4.1}$$

4.1.2 Fuerza de repulsión

La fuerza de repulsión total que influye en el robot, es la suma de todas las fuerzas de repulsión ejercidas por los obstáculos en el mapa sobre el robot. En este sentido, cada lectura del láser se tomó como fuerza repulsiva parcial, si se encuentra dentro de la distancia de influencia D_{inf}.

Como se mencionó anteriormente, la información de una lectura del láser de tipo *LaserReading* es la distancia del haz y el ángulo del mismo, siendo la lectura $L_{N/2}$ el ángulo 0°. Así pues, las componentes x y y de la fuerza de repulsión $F_{rep}(q)$ generada por una sola lectura de láser n que tiene una distancia de lectura $d_n > D_{inf}$ y angulo α_n que afectan al robot con orientación θ_R, están dadas por:

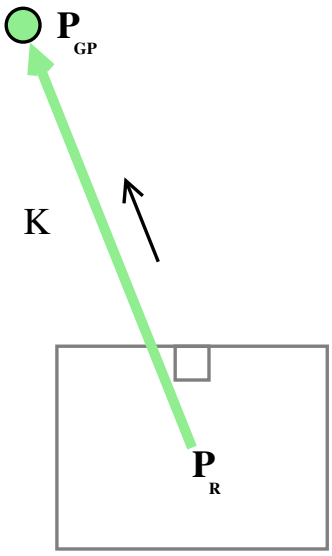


Figura 4.1 Fuerza de atracción generada por el objetivo final.

$$
\begin{aligned}
F_{rep_{nx}} &= \sqrt{\frac{1}{d_n} - \frac{1}{D_{inf}}} \left(\frac{1}{{d_n}^3}\right) (-d_n \cos(\alpha_n + \theta_R)) \\
F_{rep_{ny}} &= \sqrt{\frac{1}{d_n} - \frac{1}{D_{inf}}} \left(\frac{1}{{d_n}^3}\right) (-d_n \sin(\alpha_n + \theta_R))
\end{aligned}
\tag{4.2}
$$

Al sumar las fuerzas de repulsión parciales generadas por cada lectura del láser F_{rep_n} dentro de la distancia de influencia se obtiene la fuerza de repulsión total F_{repT}, como se muestra en la figura 4.2.

La multiplicación por la constante de repulsión K_{rep} se realiza una vez obtenida la suma final, para reducir el número de operaciones en cada iteración.

Además de esto, se propone dividir la fuerza total entre el número de lecturas del láser correctas, con el fin de que el valor de las fuerzas obtenidas para diferentes modelos de sensores láser con diferente número de lecturas totales sean aproximadamente iguales. Esto es, si N es el número de lecturas correctas de láser, la fuerza de repulsión total que afecta al robot será:

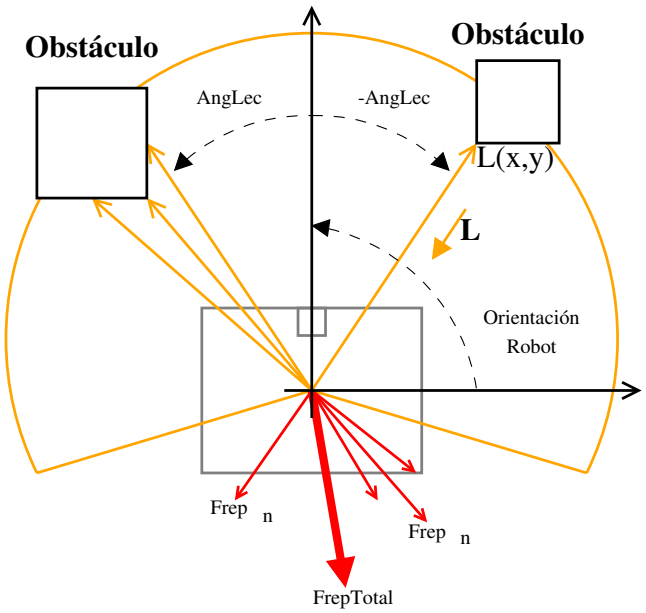


Figura 4.2 Fuerza de repulsión generada por las lecturas del láser.

$$F_{rep_T} = \frac{K_{rep}}{N} \sum_{n=0}^{N} F_{rep_n} \tag{4.3}$$

4.1.3 Fuerza Resultante

Esta es obtenida al sumar ambas fuerzas, la atractiva y la repulsiva, esto es:

$$F_{res} = F_{att} + F_{rep_T} \tag{4.4}$$

Sumando el vector de la fuerza resultante al vector de posición del robot, se genera el vector de posición de la siguiente posición deseada, es decir, el punto al que debe trasladarse el robot en ese instante de tiempo, como puede verse en la figura 4.3.

Por tanto, si P_R es la posición del robot y F_{res} es el vector de la fuerza resultante, la posición P_D del siguiente punto deseado es:

$$P_D = P_R + F_{atr} + F_{rep} \tag{4.5}$$

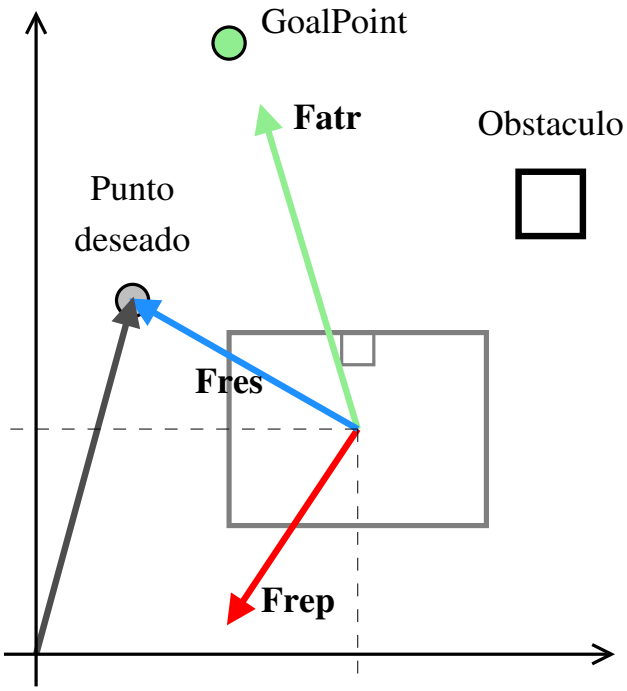


Figura 4.3 Siguiente posición deseada, generada por la fuerza resultante.

Como se verá en la sección 4.2, esta posición deseada determina el error que el robot debe reducir hasta que la posición deseada sea el objetivo final o goal point, esto es, cuando el robot llegue a su destino.

4.1.4 Algoritmo de implementación

Las tres fuerzas mencionadas anteriormente se obtienen mediante el método *CalculatePotentialField* de la clase *PotentialField*. Utiliza la información provista por las lecturas del láser, la cual esta formada por la distancia de lectura, el ángulo de esta y si la lectura fue errónea o no. Este método devuelve un arreglo con las tres fuerzas generadas por el campo potencial. Utiliza los siguientes parámetros de entrada:

- Arreglo de lecturas de láser, simuladas o reales.
- Estado $\begin{bmatrix} x & y & \theta \end{bmatrix}$ del robot, calculada por odometría o simulada.
- Posición del objetivo final o goal point, determinada por el usuario o por otro modulo del robot.

El algoritmo 6 es el utilizado para la implementación del método *CalculatePotentialField* para el cálculo de las fuerzas generadas por el campo potencial.

4.2 Control del robot

Una vez que la técnica reactiva de campos potenciales determina una posición deseada, se necesita generar un par de velocidades en los motores del robot para que ejecute el movimiento hasta el punto deseado, en el cual la velocidad en ambas llantas debe ser cero. Este movimiento conlleva una velocidad de rotación, es decir, que el robot gire hacia donde se encuentra la posición deseada y una velocidad de traslación, la cual acercará al robot hasta la posición deseada.

De las ecuaciones 2.1 y 2.2, se obtienen las velocidades de las llantas derecha (V_d) e izquierda (V_l) estableciendo las velocidades de traslación υ y rotación ω, donde L es la distancia entre las llantas, es decir el diámetro del robot. Estas son:

Algoritmo 6: Cálculo de las fuerzas del campo potencial

entrada Arreglo *lecturasLaser[]* de L lecturas de láser l.
entrada Vector de posición del goal point P_{GP}.
entrada Vector de posición del robot P_R.
entrada Orientación del robot θ_R.
entrada Valores de las constantes K_{atr}, K_{rep}, D_{inf}.
salida Arreglo de tres de vectores *fuerzas[]*.

// Cálculo de la fuerza de atracción
Inicializar vector $F_{atr} = (0,0,0)$
$\mathbf{F_{atr}} \leftarrow K_{atr}\,|\mathbf{P_{GP}} - \mathbf{P_R}|$
fuerzas[0] $\leftarrow \mathbf{F_{atr}}$
// Cálculo de la fuerza repulsión
$N \leftarrow 0$
Inicializar vector $\mathbf{F_{rep}} = (0,0,0)$
para todo *lectura de láser* l en *lecturasLaser[]* **hacer**
  si l es correcta **entonces**
    $d \leftarrow$ distancia de la lectura l
    $\alpha \leftarrow$ ángulo de la lectura l
    $\mathbf{u} \leftarrow$ vector director de la fuerza de repulsión parcial
    $f \leftarrow$ magnitud de la fuerza de repulsión parcial
    si $d > D_{inf}$ **entonces**
      $f = 0$
    si no
      $f = \sqrt{\frac{1}{d} - \frac{1}{d}}\frac{1}{d^3}$
    $u_x = -d\cos(\alpha + \theta_R)$
    $u_y = -d\sin(\alpha + \theta_R)$
    $\mathbf{F_{rep}} = \mathbf{F_{rep}} + f\mathbf{u}$
    $N = N + 1$
$\mathbf{F_{rep}} = \left(\frac{K_{rep}}{N}\right)\mathbf{F_{rep}}$
fuerzas[1] $\leftarrow \mathbf{F_{rep}}$
// Cálculo de la fuerza resultante
$\mathbf{F_{rep}} = \mathbf{F_{atr}} + \mathbf{F_{rep}}$
fuerzas[2] $\leftarrow \mathbf{F_{res}}$
regresa *fuerzas[]*

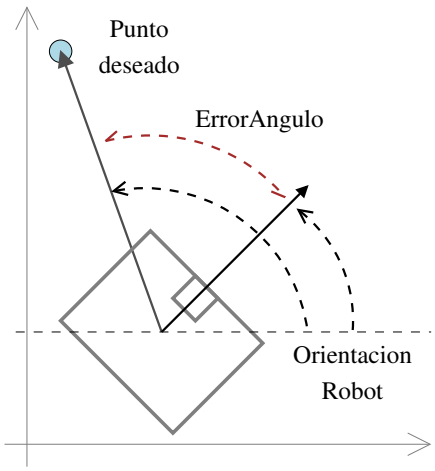


Figura 4.4 Error de ángulo entre la posición del robot y la posición deseada.

$$V_d = v + \frac{L}{2}\omega \tag{4.6}$$

$$V_l = v - \frac{L}{2}\omega \tag{4.7}$$

Para determinar la velocidad de rotación y traslación, se consideró como señal de error e_θ el angulo entre la orientación del robot y el vector de posición de la posición deseada, como se ve en la figura 4.4. Mientras se minimiza este error, el robot debe trasladarse hacia el punto deseado, lo que reducirá la distancia hacia este, hasta alcanzar una velocidad de traslación máxima v_{Max}, cuando $e_\theta = 0$. Una vez que el robot se encuentre a una distancia de tolerancia d_{tol} pequeña se considera que el robot ha llegado al punto final y el control se detiene.

Error de ángulo

El error de ángulo se considera como el ángulo más pequeño entre la orientación del robot y el vector P_{RD} que va desde la posición del robot P_R hasta la posición deseada P_D. El rango de valores para este error de ángulo es $(-\pi, \pi]$. Un ángulo negativo significa que el robot debe dar un giro en dirección contraria a las manecillas del reloj para corregirlo y un ángulo positivo, un giro en sentido de las manecillas del reloj.

El angulo ϕ del vector que va desde la posición del robot hasta el punto deseado $P_{RD} = P_D - P_R$ en un rango de $[0, 2\pi]$ se obtuvo mediante:

$$\phi = \begin{cases} \arccos\left(\frac{P_{RDx}}{|P_{RD}|}\right) & \text{si} \quad P_{RDy} \geq 0 \\ 2\pi - \arccos\left(\frac{P_{RDx}}{|P_{RD}|}\right) & \text{si} \quad P_{RDy} < 0 \end{cases} \tag{4.8}$$

La orientación del robot θ esta en el rango $[-\pi, \pi]$. Para facilitar los cálculos se obtiene θ' con rango $[0, 2\pi]$ de la siguiente manera:

$$\theta' = \begin{cases} \theta & \text{si } \theta \geq 0 \\ 2\pi + \theta & \text{si } \theta < 0 \end{cases} \tag{4.9}$$

Con los dos ángulos obtenidos anteriormente, se obtiene el error $e'_\theta = \phi - \theta'$ con rango $[-2\pi, 2\pi]$. Como se menciono anteriormente, es necesario que el error de ángulo se encuentre en el rango $[-\pi, \pi]$ por lo que el error de ángulo e_θ finalmente será:

$$e_\theta = \begin{cases} e'_\theta & \text{si} \quad -\pi \leq e'_\theta \leq \pi \\ e'_\theta - 2\pi & \text{si} \quad e'_\theta > \pi \\ e'_\theta + 2\pi & \text{si} \quad e'_\theta < -\pi \end{cases} \tag{4.10}$$

Velocidades de las llantas

Con el error de ángulo entre la posición deseada y la orientación del robot, se necesita determinar un par de velocidades en las llantas tales que:

- El robot gire con una velocidad angular proporcional al error de ángulo.
- La velocidad de traslación debe ser inversamente proporcional al error de ángulo.
- La velocidad de traslación no debe ser negativa.

Para obtener estas condiciones, se utilizan las ecuaciones propuestas por Petrilli Barcelo (2007), utilizadas para un robot diferencial que persigue una pelota, solo que en este caso, la "pelota" será la siguiente posición deseada, generada por los campos potenciales.

Velocidad de traslación

Para la velocidad de traslación v se utiliza una función gaussiana de la forma:

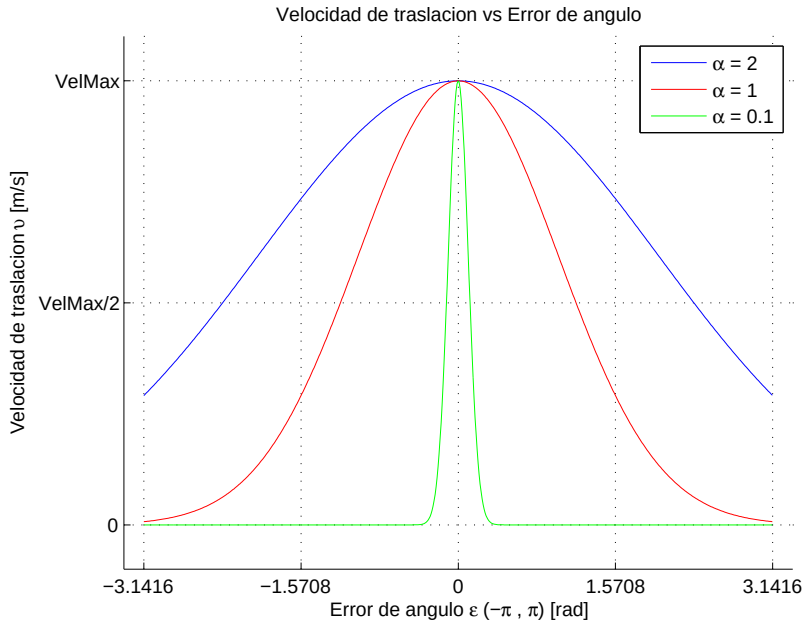


Figura 4.5 Velocidad de traslación en función del error de angulo

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{4.11}$$

Los parámetros a, b, y c, determinan características de la forma de esta función. Se ajustan para que el valor máximo v_{max} se obtenga cuando el error de angulo es cero. Para e_θ, esta queda de la siguiente forma:

$$v = v_{max}e^{-\frac{{e_\theta}^2}{2a^2}} \tag{4.12}$$

Como se puede ver en la gráfica de la figura 4.5, el parámetro α se puede ajustar para determinar el valor del angulo en el cual el robot comenzará a trasladarse. Para $\alpha < 1$, el robot comenzará a moverse después de un cierto valor de error de angulo hasta alcanzar v_{max} cuando el valor es cero. Para $\alpha > 1$, el robot se trasladará con cualquier error de angulo hasta alcanzar v_{max} desde errores de ángulo diferentes de cero.

Velocidad angular

La velocidad de rotación corresponde a una función sigmoide trasladada hacia arriba, la cual crece gradualmente desde un valor mínimo, pasando por cero cuando la variable independiente es cero, hasta un valor máximo. Con esto, el robot tendrá una velocidad angular máxima cuando , es decir, girarán en una dirección con una velocidad máxima, hasta un valor máximo positivo, es decir girarán con una velocidad máxima en la otra dirección. Además si el error de ángulo es cero, el robot no girará. Esta función está dada por:

$$y(x) = a\left(\frac{2}{1+e^{-\frac{x-b}{c}}} - 1\right) \tag{4.13}$$

Los parámetros de la ecuación se ajustan para que los valores máximos y mínimos sean ω_{max} y $-\omega_{max}$. Además, esta se centra en cero. La ecuación adecuada para el ángulo de error e_θ queda como:

$$\omega = \omega_{max}\left(\frac{2}{1+e^{-\frac{e_\theta}{\beta}}} - 1\right) \tag{4.14}$$

El parámetro β define el ángulo de cambio de la velocidad máxima y mínima, es decir, a partir de qué ángulo la velocidad angular comenzará a decrecer, como se puede observar en la gráfica de la figura 4.6.

Velocidad total de las llantas

Con el cálculo de las velocidades de traslación y angular a partir del error de angulo se generan las velocidades de las llantas izquierda y derecha para alcanzar la posicion deseada. Estas se obtienen sustituyendo las ecuaciones (4.12) y (4.14) en (4.6). Haciendo esto, las velocidades de las llantas izquierda y derecha son:

$$V_d(e_\theta) = v_{max}e^{-\frac{{e_\theta}^2}{2\alpha^2}} + \frac{L}{2}\omega_{max}\left(\frac{2}{1+e^{-\frac{e_\theta}{\beta}}} - 1\right) \tag{4.15}$$

$$V_l(e_\theta) = v_{max}e^{-\frac{{e_\theta}^2}{2\alpha^2}} - \frac{L}{2}\omega_{max}\left(\frac{2}{1+e^{-\frac{e_\theta}{\beta}}} - 1\right) \tag{4.16}$$

Las ecuaciones 4.15 y 4.16 proporciona el control de las velocidades de las llantas. La gráfica de la figura 4.7 muestra estas últimas ecuaciones. Variando los parámetros de α y β se cambia la forma en que el robot realizará la trayectoria al punto deseado.

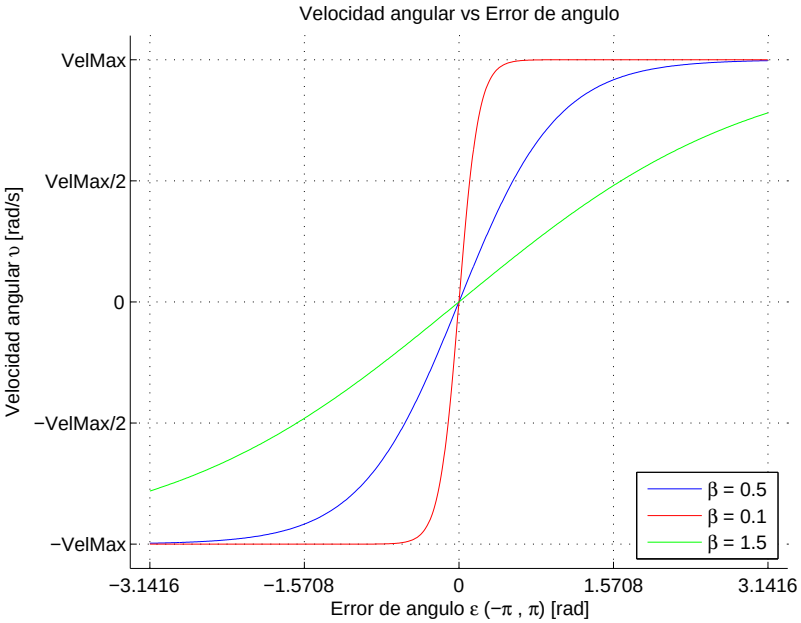


Figura 4.6 Velocidad angular en función del error de ángulo.

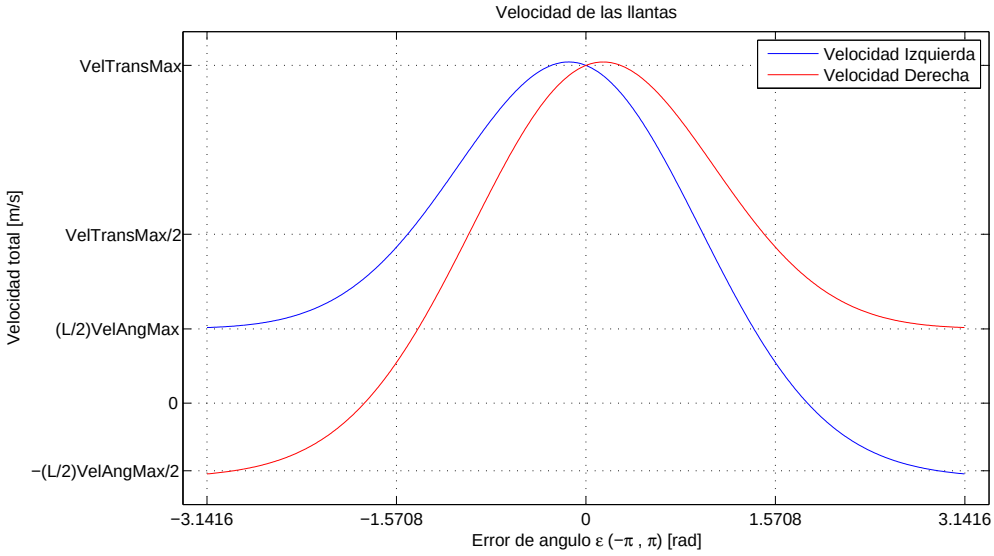


Figura 4.7 Velocidades de las llantas

4.3 Planeación de trayectoria

Una vez obtenido una posición deseada, que es la resultante del campo potencial, el robot se mueve hacia este siguiendo la ley de control mencionada anteriormente. Cuando existe una nueva lecturas del sensor láser, el robot realiza un nuevo cálculo de campo potencial con las lecturas nuevas, definiendo una nueva posición deseada. Esto se repite hasta que alguna de las siguientes condiciones se cumplen:

- El robot llega al goalpoint.
- El robot colisiona con algún obstáculo.
- El numero de iteraciones de excede un numero máximo.

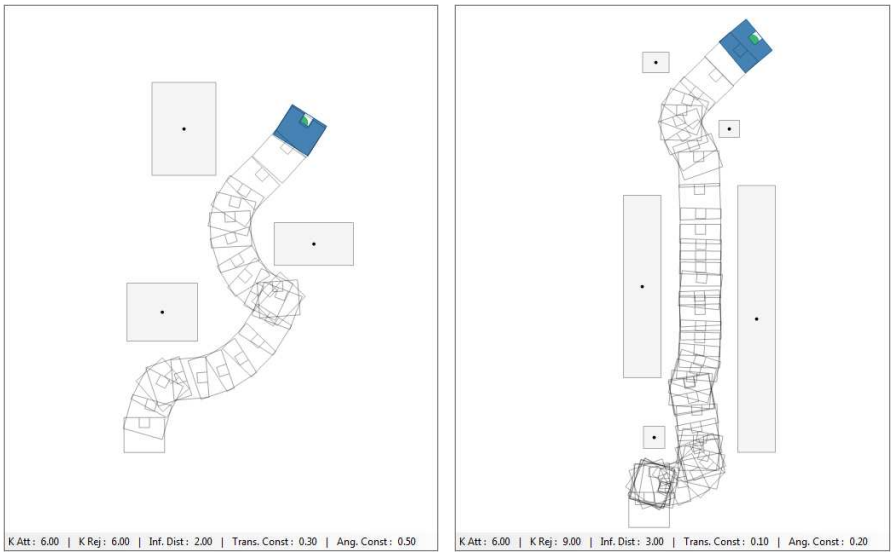
El algoritmo 7 es el algoritmo propuesto para la planeación de trayectorias.

Algoritmo 7: Algoritmo de planeación de trayectorias

entrada Arreglo de lecturas de laser.
entrada Arreglo de objetivos de la trayectoria *GoalPoint[]*
  para todo goalPoint *GP* en *GoalPoint[]* **hacer**
    Calcula distancia al objetivo final D_f
    mientras ($D_f < tolerancia$) **o** ($pasos > pasosMax$) **o** (**negado** colisión) **hacer**
      Calcular campo potencial
      Calcular error de angulo e_θ
      Calcular velocidades de las llantas V_d y V_l
      Establecer velocidades de las llantas
      Detectar colisión
      Calcular distancia al objetivo final D_f
      pasos++
  Establece velocidades de las llantas en cero // Por seguridad
  Crear reporte de trayectoria

Al finalizar cada trayectoria, se genera un reporte con información sobre las condiciones de la trayectoria que servirán más adelante para calcular la función de evaluación utilizada por el algoritmo genético para la optimización. Las características reportadas de cada trayectoria son:

- Distancia al objetivo final.
- Numero de pasos en la trayectoria.



(a) Mapa 1 (b) Mapa 2

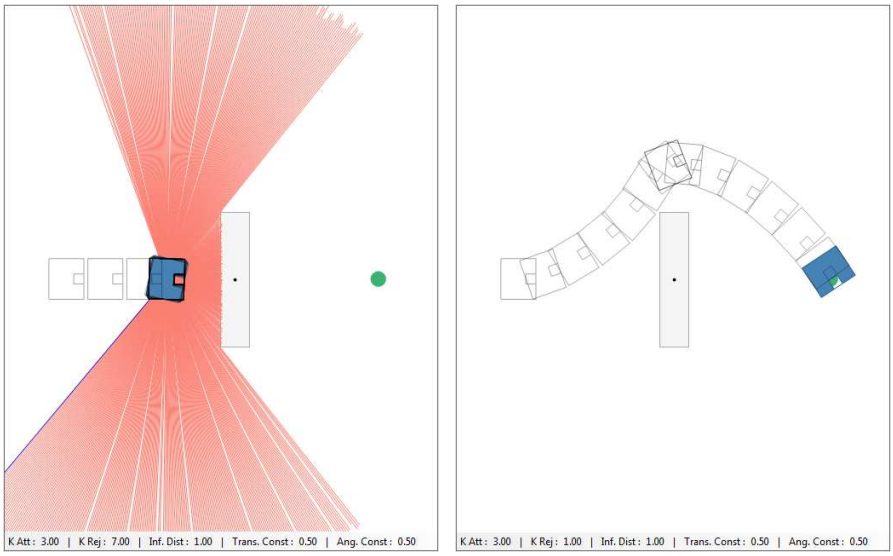
Figura 4.8 Ejemplos de trayectorias en diferentes mapas

- Distancia al obstáculo más cercano.
- Promedio de distancia a los obstáculos.
- Colisión detectada.

En la figura 4.8 se presentan algunos ejemplos trayectorias simuladas en diferentes ambientes utilizando parámetros obtenidos manualmente.

Cabe mencionar que el planeador de trayectorias sigue restringido por el problema de mínimos locales, por lo que si, para un mapa determinado, no se encuentra un conjunto de valores que permitan al robot llegar a su destino, se considera que existe un mínimo local en ese recorrido y se opta por recorrer la trayectoria utilizando un goal point intermedio que lo ayude a sortear el mínimo local y pueda llegar a su destino. Esto se realizó en el ejemplo de la figura 4.9.

Además de esto, se observó otro tipo de problema que impide que el robot llegue al objetivo final, o presente algún tipo de oscilación, dependiendo de los valores de las constantes. Este problema se presenta cuando el robot está muy cerca de un obstáculo y la fuerza resultante determina un giro en dirección contraria del obstáculo. Cuando el robot gira para no colisionar con el obstáculo y, dado que el rango de detección del láser esta limitado a un arco de circulo, el sensor láser dejará de detectar al obstáculo, por lo que la fuerza resultante determinará nuevamente un giro hacia el obstáculo. En

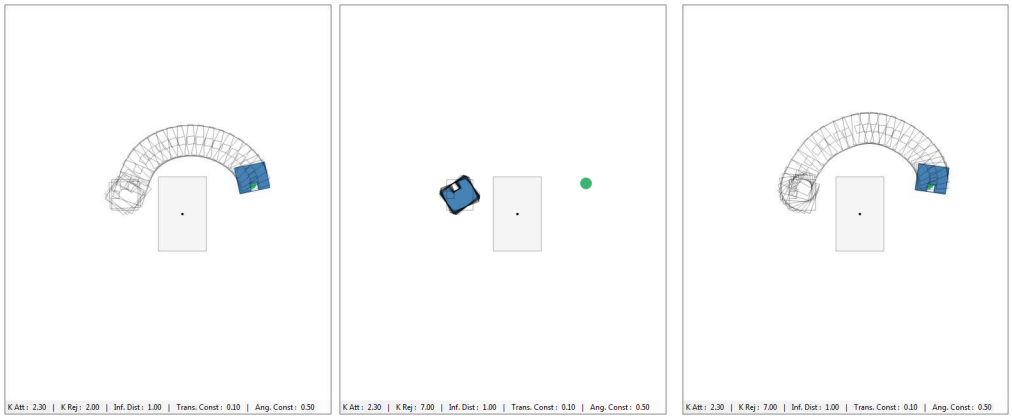


(a) Mínimo local (b) Usando goal point intermedio

Figura 4.9 Ejemplo de mínimo local

este momento, la fuerza repulsiva del obstáculo se presentara nuevamente, haciendo que el robot repita este procedimiento una y otra vez.

Para ejemplificar esto, se simularon tres pruebas en el mapa de la figura 4.10. En éste, el robot comienza con una orientación contraria al obstáculo y al goal point. La primera trayectoria, figura 4.10a, se realizó con un constante de repulsión pequeña, $K_{rep} = 2.0$ con la cual, el robot logra evadir el obstáculo. En la segunda, figura 4.10b, la constante se estableció en $K_{rep} = 7.0$, por lo que se presentan las oscilaciones que impiden al robot llegar al goalpoint. En la trayectoria de la figura 4.10c, se establecieron las mismas condiciones que para cuando se presentan las oscilaciones, excepto que el láser fue simulado con un rango de detección de 2π por lo que, al girar, el robot sigue detectando el obstáculo. Esto permite que el robot se aleje y lo evada, llegando finalmente al goal point.



(a) No hay oscilaciones (b) Oscilaciones (c) Laser con rango de 2π

Figura 4.10 Oscilación por rango de deteccion del laser

Capítulo 5

Implementación del algoritmo genético

Para que el robot pueda navegar reactivamente en un ambiente de forma eficiente utilizando el algoritmo de planeación de trayectorias descrito en la Sección 4.3, es necesario establecer las constantes de atracción, repulsión y la distancia de influencia de los campos potenciales. Para ambientes sencillos, es fácil determinar estas constantes mediante algunas pruebas en el simulador. Sin embargo, a medida que un mapa se vuelve más complejo esta tarea se vuelve demasiado impráctica dado que las combinaciones de constantes que permiten al robot trazar una trayectoria exitosa se reducen y el número de pruebas necesarias para encontrarlas aumenta considerablemente.

Con el objetivo de buscar las constantes que determinan una trayectoria satisfactoria para el robot, se implementó el algoritmo genético canónico presentado en el capítulo 2, el cual haciendo uso del simulador, realizará una búsqueda de éstas en el espacio de soluciones para un mapa .

La implementación de cada paso del algoritmo representado en el diagrama de la figura 2.5, se explica en las siguientes secciones.

Cabe mencionar que con el objetivo de hacer escalable el programa para resolver otro tipo de problemas relacionados con la planeación de trayectorias y la navegación mediante algoritmos genéticos, la implementación de éste se hizo lo más general posible, haciendo uso de las herramientas proporcionadas por el lenguaje C#.

5.1 Codificación de los cromosomas

Para la codificación de los cromosomas se eligieron cadenas binarias formadas por un arreglo de genes. Cada gen representa un parámetro $P \in \mathbb{R}$ a ser evaluado en el problema a optimizar.

Se definió la clase *Gen*, la cual contiene la cadena binaria que representa el valor decimal que corresponde al parámetro a evaluar. Para obtener la codificación binaria de un gen a partir de un valor real, es necesario determinar el valor máximo y mínimo que puede tener dicho valor. Por tanto, la información que contiene cada objeto de la la clase *Gen* es:

- Valor decimal v_{dec}
- Valor binario v_{bin}
- Valor decimal máximo v_{max}
- Valor decimal mínimo v_{min}

Los valores v_{max} y v_{min} definen el rango del gen, es decir, el intervalo de valores dentro del cual estará el parámetro representado por este. Al escoger estos dos parámetros se define también la resolución de este gen en particular, es decir, el paso de cuantización. Dado que se eligieron cadenas de 8 bits para la representación binaria de cada gen, la resolución res de cada gen será:

$$res = \frac{v_{max} - v_{min}}{2^8 - 1} \tag{5.1}$$

Una vez obtenida la resolución de un gen, se puede obtener la cadena binaria que lo representa dividendo el valor decimal v_{dec} entre la resolución y redondeando el valor obtenido. El valor redondeado será un entero sin signo en el intervalo 0 a 255. Este entero sin signo val_{10}, representa en base 10 el valor binario en base 2. Esto es:

$$val_{10} = round\left(\frac{v_{dec}}{res}\right) \tag{5.2}$$

Para convertir este valor en base 10 a base 2, se hace uso del método *Convert.ToString()*, implementado por la librería *system* de C#, el cual devuelve un valor de tipo *string* con la cadena de bits de un entero no signado de 8 bits. Esto resulta muy conveniente, dado que, como se verá posteriormente, se utilizaron los métodos de manejo de cadenas del

lenguaje C# para la implementación de los operadores sobre cromosomas del algoritmo genético.

$$v_{bin} = Convert.ToString(val_{10}) \tag{5.3}$$

Una vez que se define un arreglo de la clase *Gen*, representando cada uno de estos un parámetro a evaluar, se puede instanciar un objeto del tipo *Chromosome*. De esta forma, la información de cada gen es accesible a través de dicho cromosoma. Esta clase proporciona además la forma binaria del cromosoma, al concatenar la representación binaria de cada gen en un cadena de caracteres. Si N es el número de genes y v_{binn} es la cadena de texto de caracteres binarios de cada gen, entonces la forma binaria de un cromosoma se obtiene de la siguiente manera:

$$cromosoma = \underbrace{v_{bin1}}_{Gen1} + \underbrace{v_{bin2}}_{Gen2} + \cdots + \underbrace{v_{binN}}_{GenN} \tag{5.4}$$

5.2 Población inicial

El primer paso para la ejecución de un algoritmo genético es la creación de la población inicial. Para esto, se debe definir primero el número cromosomas N que integrarán la población inicial. Los cromosomas iniciales se crean a partir de valores decimales aleatorios para cada gen en cuestión. Además de esto, se definen los parámetros a evaluar para un problema dado. El número de parámetros a evaluar será el número de genes en cada cromosoma. Se deben definir también los valores máximos y mínimos de cada gen, los cuales dependerán del problema a resolver.

Para generar un valor decimal de un gen aleatoriamente, se utiliza la clase *System.Random*. Ésta proporciona un valor aleatorio entre 0 y 1. Puesto que es necesario que el valor decimal se encuentre en un rango entre v_{max} y v_{min}, si r es un valor aleatorio en el intervalo $[0, 1]$, el valor decimal se consigue de la siguiente manera:

$$v_{dec} = v_{min} + r(v_{max} - v_{min}) \tag{5.5}$$

Utilizando este método para generar los valores aleatorios iniciales para cada cromosoma, la población inicial se crea utilizando el algoritmo 8.

Algoritmo 8: Creación de la población inicial

entrada Número de cromosomas N
entrada Número de genes por cromosoma G
salida Arreglo de cromosomas *PoblacionInical[]*
  para i **hasta** N **hacer**
    Crear nuevo cromosoma C_i
    para $j = 0$ **hasta** G **hacer**
      Crear nuevo gen G_j
      $v_{dec\,j} = v_{min\,j} + random(v_{max\,j} - v_{min\,j})$
      $G_j \leftarrow v_{dec\,j}$
      Agregar G_j a cromosoma C_i
    Agregar C_i a *PoblacionInical[]*
  regresa *PoblacionInical[]*

5.3 Evaluación de los cromosomas

El arreglo de cromosomas provenientes de la población inicial o de una nueva generación, se deben evaluar para obtener el valor de ajuste que representa en el problema a determinar.

Para hacer esto se obtienen los valores decimales de cada gen del cromosoma a evaluar, se asignan a los parámetros que los representan en la planeación de trayectorias y se inicia una simulación. Una vez terminada la simulación, ésta devuelve un reporte de la trayectoria, como se explicó en en la sección 4.3.

Utilizando la información contenida en el reporte de trayectorias, se asigna un valor de ajuste a cada trayectoria, utilizando una función de evaluación que mide las características de ésta.

Una vez obtenido el valor de ajuste de cada trayectoria probada, se devuelve un arreglo con la información de ajuste-cromosoma, el cual será utilizado por los siguientes operadores del algoritmo genético.

El algoritmo 9 es el utilizado para la evaluación de los cromosomas.

Algoritmo 9: Evaluación de los cromosomas

entrada Arreglo *Cromosomas[]* de cromosomas
entrada Función de evaluación $f(cromosoma)$
salida Arreglo *ReporteCromosomas[]* de cromosomas-ajuste
  para todo $cromosoma_i$ en *Cromosomas* **hacer**
    Generar un nuevo mapa con parámetros a evaluar p_j
    para todo Gen_j en $cromosoma_i$ **hacer**
      $p_j \leftarrow v_{dec\,j}$
    Realizar simulación del mapa
    Obtener reporte de simulación
    $fit_i \leftarrow f(cromosoma_i)$
    Agregar fit_i y $cromosoma_i$ a *ReporteCromosomas*
  regresa *ReporteCromosomas*

5.4 Función de evaluación

Para que la trayectoria de un robot de servicio sea considerara favorable, además de que el robot llegue a su destino, se deben tomar otros factores en cuenta, como la seguridad de la trayectoria y la distancia más corta (Shi y Cui, 2010). Para el presente trabajo, dadas las tareas que debe realizar habitualmente el robot Justina y los tiempos en que debe realizarlas, los parámetros que se tomaron en cuenta para obtener el valor de ajuste de las trayectorias del robot mediante la función de evaluación son:

- Distancia final al objetivo d_{gp}
- Distancia al obstáculo más cercano d_{obs}
- Número de pasos en la trayectoria $pasos$

Además de estas consideraciones, se agregaron penalizaciones a la función de evaluación, las cuales, durante la búsqueda de una trayectoria óptima, deberán ser eliminadas por el algoritmo genético (Tuncer y Yildirim, 2012). Estas penalizaciones son:

- Por colisión con un obstáculo
- Por no llegar al objetivo final

La función de evaluación se diseñó para que las mejores trayectorias regresen un valor de ajuste menor y las trayectorias no favorables obtengan un valor mayor. La

función objetivo se dividió en 3 subfunciones, multiplicadas por un factor de peso, el cual puede ser adecuado para obtener trayectorias más específicas. Adecuando la función de evaluación de Shi y Cui (2010), ésta quedan de la siguiente manera:

$$fit = w_1 fit_1 + w_2 fit_2 + w_3 fit_3 + w_4 + w_5 \tag{5.6}$$

Las subfunciones fit_1, fit_2 y fit_3, además de las penalizaciones w_4 y w_5 que la integran se explican a continuación.

Distancia final al objetivo

Dado que se requiere que el robot siempre llegue a su destino, se toma como factor primario la distancia al objetivo final. La distancia se encuentra en metros.

$$fit_1 = d_{gp} \tag{5.7}$$

Distancia al obstáculo más cercano

En segundo lugar se toma en cuenta la seguridad de la trayectoria, por lo que una distancia a los obstáculos menor, generara un valor de ajuste mayor. Puesto que el láser, se encuentra en el centro del robot, la distancia mínima a los obstáculos será el radio del robot. La subfunción de la distancia a los obstáculos es:

$$fit_2 = \left(\frac{a}{d_{obs}}\right)^2 \tag{5.8}$$

La constante a se ajusta para que $fit_2 = 1$ cuando la distancia al obstaculo sea el radio del robot, como se muestra en la figura 5.1.

Número de pasos en la trayectoria

El número de pasos de una trayectoria es la cantidad de comandos de movimiento que se envían a los motores. Si entre dos trayectorias que consiguen llegar al mismo punto, una lo hizo usando un menor número de pasos, esta será mejor. Dado que se tiene un número máximo de pasos en una trayectoria, éstos se normalizan, por lo que la subfunción queda como:

$$fit_3 = \frac{pasos}{pasosMax} \tag{5.9}$$

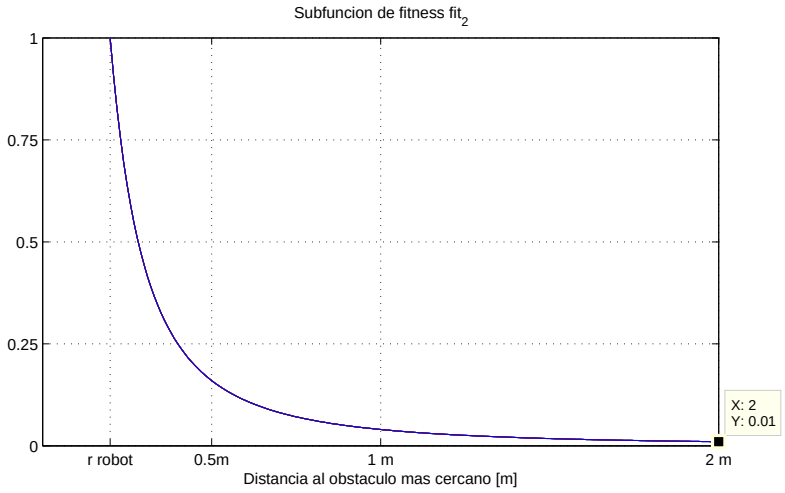


Figura 5.1 Subfuncion de evaluación fit_2

Penalizaciones

Se asignan dos penalizaciones a cada trayectoria, una por colisionar con algún obstáculo w_4 y otra por no llegar al objetivo final w_5, asegurando así, una diferencia más drástica entre una trayectoria en la que el robot llego al objetivo final y otra en la que el robot se quedó muy cerca de este.

5.5 Selección de los cromosomas padres

Una vez que se tiene el conjunto de valores cromosoma-ajuste obtenidos mediante el proceso de evaluación de cromosomas, se deben seleccionar los cromosomas que serán los padres de la siguiente generación mediante el operador de selección. El número de padres seleccionados será igual al número de cromosomas existentes en la población.

Para la selección se eligió el método de torneo determinístico por ser un método eficiente, al no requerir ordenar la lista de cromosomas, y que tiende a soluciones óptimas (Sivanandam y Deepa, 2008). Sin embargo, este tipo de selección puede inducir a que solamente se seleccionen individuos con valores de ajuste muy altos, reduciendo la diversidad de la población y aumentando la posibilidad de que el algoritmo converja en un valor óptimo local, lo que se conoce como presión de selección. Para disminuir

la presión de selección, el número de competidores escogidos para el torneo es dos, lo que permite una mayor diversidad de individuos.

Teniendo un arreglo de cromosomas con sus respectivos valores de ajuste, obtenido mediante el proceso de evaluación visto anteriormente, se seleccionan dos de forma aleatoria con una probabilidad uniforme. Para estos dos, se comparan sus valores de ajuste y se selecciona el que tenga mejor. El cromosoma seleccionado es copiado e ingresado a un nuevo arreglo donde estarán los cromosomas padre. Este proceso se repite N veces, donde N es el número cromosomas en el primer arreglo. El algoritmo 10 es el implementado para este operador.

Algoritmo 10: Selección por torneo

entrada Arreglo $cromosoma[]$ de N cromosomas con sus valores de ajuste $fit[]$
salida Arreglo $padres[]$ de N cromosomas
1: **para** $n = 0$ **hasta** N **hacer**
2: **si** $fit[ran1] <= fit[ran2]$ donde $ran1$ y $ran2$ son enteros aleatorios $\in [0, N]$ **entonces**
3: Agregar $cromosoma[ran1]$ a $padres[]$
4: **si no**
5: Agregar $cromosoma[ran2]$ a $padres[]$
6: $n \leftarrow n + 1$
7: **regresa** $padres[]$

5.6 Cruza de los cromosoma padre

La cruza es el operador mediante el cual, dos cromosomas padre seleccionados dan origen a dos nuevos cromosomas combinando la información que cada uno contiene. Para esto, se dividen las cadenas binarias de cada cromosoma en partes y éstas se intercambian.

Se utilizó una cruza de un solo punto, la cual es utilizada en los algoritmos genéticos tradicionales. En ésta, se dividen las cadenas en un solo punto, denominado punto de cruce. El punto de cruce se determina aleatoriamente con una probabilidad uniforme.

Como se mencionó anteriormente, en la clase *chromosome*, la cadena binaria se genera como un dato de tipo *string*, lo que permite el uso de los métodos para manejo de cadenas del lenguaje C#. Entre estos métodos se encuentra *substring*, el cual regresa

una cadena de caracteres a partir de otra, definiendo el índice a partir del cual se quiere obtener la cadena y el número de caracteres a copiar. Utilizando este método, las cadenas binarias se pueden separar en las partes necesarias, generando la primera parte como la subcadena que va desde el índice cero de la cadena binaria hasta al punto de cruce, y la segunda parte como la subcadena que va después del punto de cruce hasta el final de la cadena.

La implemnetación de la cruza en un solo punto se explica en el algoritmo 11:

Algoritmo 11: Cruza en un solo punto

entrada Arreglo $padres[]$ de N de cadenas binarias padre
salida Arreglo $hijos[]$ de N de cromosomas hijo
  $l \leftarrow$ dimension de las cadenas binarias
  para $i = 0$ **hasta** N-1 **hacer**
    Generar punto de cruce P_c donde P_c es un entero aleatorio en $[0, l]$
    $p_{11} \leftarrow$ subcadena de $padre[i]$ desde 0 hasta P_c
    $p_{12} \leftarrow$ subcadena de $padre[i]$ desde P_c hasta l
    $p_{21} \leftarrow$ subcadena de $padre[i+1]$ desde 0 hasta P_c
    $p_{22} \leftarrow$ subcadena de $padre[i+1]$ desde P_c hasta l
    $hijo[i] \leftarrow$ concatenar p_{11} y p_{22}
    $hijo[i+1] \leftarrow$ concatenar p_{21} y p_{12}
    $i \leftarrow i+2$
  regresa $hijos[]$

5.7 Mutación de los hijos

Para preservar la diversidad en el espacio de búsqueda, comúnmente se utiliza el operador mutación, el cual asegura que la población no se fije en un óptimo local (Michell, 1998). Éste puede modificar, con una probabilidad muy pequeña, los datos de la cadena binaria. Si la probabilidad de mutación es grande, las información de los cromosomas que representan mejores soluciones, se puede corromper.

Se eligió el método de mutación en el cual, cada bit de cada cromosoma, tiene una probabilidad P_m de ser invertido, es decir, si un bit $b = 0$ se intercambia por un uno; de igual manera, si $b = 1$ el valor se intercambia por un cero. La probabilidad P_m se elige al iniciar el algoritmo genético y permanece constante mientras se ejecuta. Comúnmente, esta probabilidad se establece mediante $P_m = 1/l$, donde l es la longitud de la

cadena binaria (Sivanandam y Deepa, 2008). La implementación de este operador se presenta en el algoritmo 12.

Algoritmo 12: Operador de Mutación usando inversión de bit

entrada Arreglo *Cromosomas* de N cadenas binarias bin de longitud l
entrada Probabilidad de mutación P_m
salida Arreglo *mutaciones[]* de N cromosomas mutados
  para $i = 0$ **hasta** $N - 1$ **hacer**
    $bin2 \leftarrow$ cadena vacía de longitud l
    para $j = 0$ **hasta** $l - 1$ **hacer**
      Generar un número aleatorio ran en $I = [0, 1] | ran \in \mathbb{R}$
      si $ran >= P_m$ **entonces**
        $bin2[j] \leftarrow bin[j]$
      si no
        $bin2[j] \leftarrow$ **negado** $bin[j]$
    mutaciones[i] $\leftarrow bin2$
  regresa *mutaciones[i]*

5.8 Término del algoritmo

El arreglo de cromosomas mutados remplazará a la población de la generación anterior y se convertirá en la nueva generación a ser evaluada. Esta nueva generación pasará por todos los métodos descritos anteriormente y generará a su vez, una nueva generación. Este proceso continua hasta llegar a la última generación. El número generaciones es determinado al inicio del algoritmo genético.

Al final, el programa entrega un reporte en formato XML donde se visualiza, por generación, el cromosoma con el mejor valor de ajuste y los valores decimales de los genes que lo integran. Además se guardan, los cromosomas evaluados en de cada generación, el valor de ajuste de éstos y los datos de la trayectoria.

Capítulo 6

Optimización de Trayectorias

En este capítulo, en la sección 6.1 se desarrolla la metodología utilizada para optimizar las trayectorias del robot en un espacio de trabajo determinado, utilizando el algoritmo genético implementado en el capítulo 5. Además, en la sección 6.2, se analizan diferentes trayectorias en diversos espacios de trabajo con los resultados de la optimización mediante algoritmos genéticos. En la sección 6.3 se detallan los resultados de las pruebas experimentales realizados en el robot Justina, utilizando como espacio de trabajo el laboratorio de Bio-robótica y sus alrededores, ubicado en la Facultad de Ingeniería en la UNAM.

6.1 Metodología

Para encontrar una trayectoria óptima, es decir, con un alta probabilidad de alcanzar el punto final y alejada de los obstáculos utilizando un algoritmo genético, para el robot Justina, es necesario definir el espacio de trabajo en el que se desplazará el robot, es decir, el mapa en el cual se realizará una optimización de una trayectoria.

Para crear el mapa de manera sencilla, se utiliza la interfaz gráfica para formar un conjunto de obstáculos definiendo las dimensiones y la posición de éstos. Además de los obstáculos, el mapa debe contener la posición inicial y final del robot en la trayectoria.

Con el mapa creado, se definen los parámetros reales del robot y de los sensores utilizados, como las características del láser y las velocidades máximas del robot. Una vez construido el mapa del espacio de trabajo y elegidos estos parámetros, se guardan

en un archivo de texto. Generar este archivo tiene dos objetivos: guardar los datos para futuras pruebas y que el algoritmo genético defina las características para realizar las simulaciones.

Teniendo las características de las simulaciones, se definen los parámetros del algoritmo genético. Estos parámetros son:

- Número de individuos en la población.
- Número de generaciones.
- Parámetros a evaluar del problema.
- Puntos intermedios en la trayectoria.

Dado que no existen resultados concluyentes sobre qué combinación de parámetros en un algoritmo genético es mejor (Michell, 1998), se optó por iniciar con un número pequeño de individuos y de generaciones, reduciendo el tiempo de ejecución del algoritmo. El tiempo de ejecución del algoritmo también se ve afectado por la complejidad del mapa a evaluar. Entre más obstáculos existan en el mapa, el tiempo de simulación crece. Ésto es porque el simulador del láser y la simulaciones de colisiones del robot se efectúan haciendo cálculos sobre cada obstáculo.

La probabilidad de mutación se fijó para todos los experimentos en $1/L$ donde L es la longitud de la cadena binaria.

Los valores a evaluar en cada trayectoria, es decir, los genes que conforman los cromosomas del algoritmo, para todos los experimentos, son las constantes de atracción K_{atr}, de repulsión K_{rep} y la distancia de influencia D_{inf}. Éstos afectan directamente la siguiente posición deseada en la trayectoria del robot. Además de estos valores, en la mayoría de los experimentos, se evaluaron las constantes de la ecuación de control de las velocidades de las llantas del robot, es decir las constantes α y β de la ecuación 4.15. Estas dos constantes determinan la forma en que el robot se traslada al punto determinado por el campo potencial, por tanto, en conjunto con las constantes del campo potencial, determinan la trayectoria que seguirá el robot.

En algunos mapas, se utilizaron puntos intermedios, siguiendo los experimentos de Arámbula y Padilla (2004) para guiar al robot de una región a otra.

Con los parámetros elegidos, se inicia el algoritmo genético, el cual realizará simulaciones con las características determinadas y, utilizando los operadores de éste, generará un archivo de reporte con los resultados de la evaluación de cada individuo en cada generación.

Del reporte de resultados, se elige al mejor individuo de todas las generaciones y se simula la trayectoria generada por este. Si este individuo genera una trayectoria favorable, se eligen estos parámetros como óptimos. Si la trayectoria no es favorable, se inicia un nuevo algoritmo genético, con un número de población y de individuos mayor y se repite el proceso de optimización.

6.2 Resultados en simulación

A continuación se presentan los resultados obtenidos para diferentes mapas, variando los parámetros del algoritmo genético, con el fin de evaluar las trayectorias de los mejores individuos. Los mapas 1 y 2 representan ambientes reales mientras que los mapas 3, 4 y 5 se obtuvieron a partir de conocidos artículos de investigación sobre navegación de robot móviles.

En los mapas presentados, el robot es identificado por un rectángulo, en color azul, con las dimensiones del robot real. El pequeño rectángulo que se puede observar dentro del robot simboliza el frente del mismo. El círculo verde define el objetivo final de la trayectoria y los círculos amarillos los puntos intermedios, mientras que una línea azul los une en el orden en que deben ser alcanzados. Los obstáculos y las paredes son rectángulos grises, dentro de los cuales, se observa un punto que determina su centro.

6.2.1 Mapa 1 : Pasillo con un obstáculo

Se generó el mapa 1, el cual simula un pasillo de 2.3 metros de ancho con un obstáculo cerca del centro. El robot se encuentra a 4 metros del objetivo final. En linea recta, el obstáculo se encuentra obstruyendo la trayectoria del robot hacia el objetivo final. Este mapa se puede observar en la figura 6.1. Con este, se pretende observar las trayectorias de las mejores soluciones al modificar el número de individuos y el número de generaciones. Idealmente, entre más individuos existan en una población y el número de generaciones sea mayor, el espacio de búsqueda será más grande y se reducirá el riesgo de encontrar una solución óptima local, con lo que se podría encontrar una mejor trayectoria.

Para comparar las trayectorias de las pruebas realizadas, al final de la sección se encuentran las trayectorias descritas durante éstas.

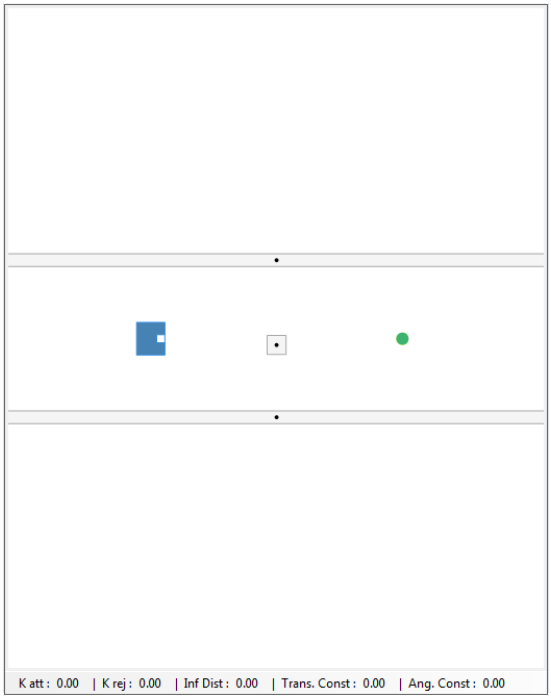


Figura 6.1 Mapa 1

Tabla 6.1 Valores obtenidos del mejor individuo (pob=10, gen=10)

Ajuste	0.8611
K_{atr}	5.8597
K_{rep}	9.6484
D_{inf}	1.2311

Prueba 1

En la primera prueba se ejecutó el algoritmo genético con los siguientes parámetros:

- Número de individuos : 10
- Generaciones : 10
- Parámetros a evaluar : K_{atr}, K_{rep} y D_{inf}
- Puntos intermedios : 0
- Constante de velocidad de traslación $\alpha = 0.1$
- Constante de velocidad de angular $\beta = 0.5$

Dada la simplicidad del mapa se eligió un número pequeño de individuos y de generaciones.

Las constates α y β se eligieron para que, de forma segura, el robot rote hasta corregir el error de ángulo hasta un valor de aproximadamente 0.35 radianes y, a partir de este ángulo, comience a trasladarse. De esta forma se reduce la probabilidad de que el robot colisione mientras describe un arco al rotar y trasladarse al mismo tiempo para alcanzar la siguiente posición deseada.

En la tabla 6.1 se muestra el valor de ajuste del mejor individuo y las constantes obtenidas por este.

La trayectoria del mejor individuo se muestra en la figura 6.2a. En ésta se puede ver que el robot llega a al objetivo final, alejándose del obstáculo usando el campo potencial.

Tabla 6.2 Valores obtenidos del mejor individuo (pob=50, gen=50)

Ajuste	0.6494
K_{atr}	1.2508
K_{rep}	7.2658
D_{inf}	0.7117

Prueba 2

Para el mapa de la figura 6.1, se ejecutó el algoritmo genético variando el número de individuos y el número de generaciones, esperando conseguir mejores resultados. Los parámetros utilizados son:

- Número de individuos : 50
- Generaciones : 50
- Parámetros a evaluar : K_{atr}, K_{rep} y D_{inf}
- Puntos intermedios : 0
- Constante de velocidad de traslación $\alpha = 0.1$
- Constante de velocidad de angular $\beta = 0.5$

Utilizando estos parámetros se consiguió un individuo con mejor valor de ajuste que el obtenido en la prueba 1. El mejor individuo fue encontrado en la generación 42. Los valores obtenidos para el mejor individuo se observan en la tabla 6.2.

Utilizado los valores del mejor individuo, se consiguió una trayectoria más segura, en la cual el robot se mantiene a una distancia mayor del obstáculo en cada paso. Esta trayectoria se puede observar en la figura 6.2b.

Prueba 3

Para las pruebas 1 y 2, se fijaron los valores de las constantes de traslación y angular. Con estas, el robot, se trasladará de forma segura pero se incrementan los pasos de la trayectoria puesto que el robot debe girar antes de comenzar a moverse.

Para la prueba 3, se tomaron como parámetros a evaluar por el algoritmo genético, además de las constantes del campo potencial, las constantes α y β, con el fin de conseguir trayectorias más suaves donde el robot avance al mismo que tiempo que rota.

Tabla 6.3 Valores del mejor individuo de la prueba 3

Ajuste	0.6103
K_{atr}	0.5087
K_{rep}	7.2268
D_{inf}	0.7937
α	1.8984
β	0.3680

Los parámetros elegidos para esta prueba fueron:

- Número de individuos : 10
- Generaciones : 10
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 0

En esta prueba, se consiguieron individuos con mejor valor de ajuste, dado que la trayectoria recorrida se realizó en un menor número de pasos. Los valores obtenidos se encuentran en la tabla 6.3.

Se simuló la trayectoria del robot, utilizando los valores de la tabla 6.3. En esta trayectoria, el robot se mantuvo a una distancia segura del obstáculo y se ejecutó en un número menor de pasos, lo que permite al robot trasladarse con mayor velocidad. Esta trayectoria se muestra en la figura 6.2c.

Prueba 4

En la prueba 4 se aumentó el número de individuos y el número de generaciones de la prueba 3. En ésta, al igual que en la prueba 2, se eligió un número de 50 individuos con 50 generaciones. Estos parámetros son:

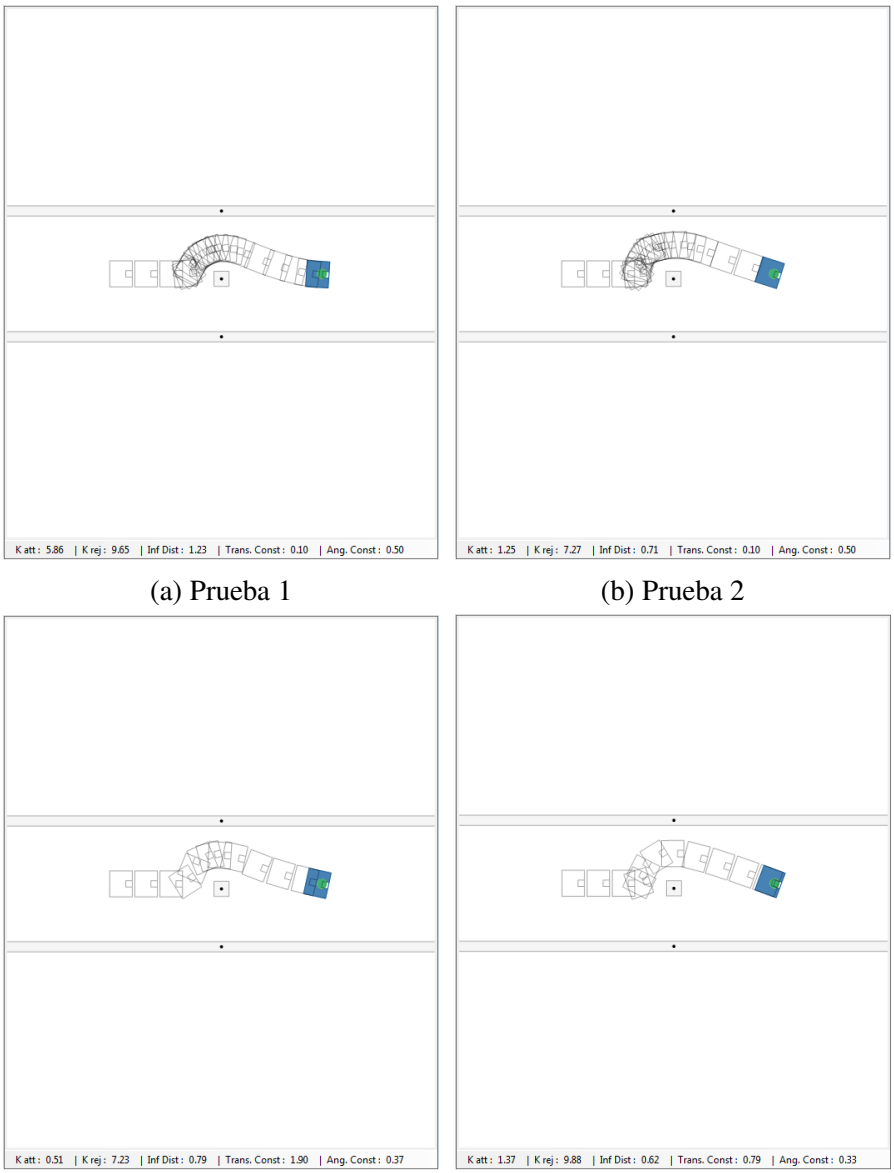
- Número de individuos : 50
- Generaciones : 50
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 0

Tabla 6.4 Valores del mejor individuo de la prueba 4

Ajuste	0.5447
K_{atr}	1.3680
K_{rep}	9.8828
D_{inf}	0.7896
α	0.7896
β	0.3289

Como se esperaba, en esta prueba se obtuvo el individuo con el mejor valor de ajuste de las 4 realizadas. Los parámetros obtenidos por este individuo, encontrado en la generación 47, se resumen en la tabla 6.4.

La trayectoria descrita por el robot con los parámetros del mejor individuo de esta prueba, se ejecutó en un menor número de pasos que en las pruebas 1 y 2 y cercano al número de pasos de la prueba 3. En cuanto a la distancia a los obstáculos, la trayectoria de la prueba 4 mostró un mejor comportamiento, aumentando la distancia a los obstáculos en comparación con la prueba 3. Esta trayectoria se observa en la figura 6.2d.



(a) Prueba 1 (b) Prueba 2

(c) Prueba 3 (d) Prueba 4

Figura 6.2 Trayectorias obtenidas para el mapa 1

6.2.2 Mapa 2 : Ambiente Real Complejo

El mapa de la figura 6.3 es una representación del laboratorio de Biorobótica. Este se usó para pruebas dado que representa una mapa complejo para la navegación debido a la configuración del mobiliario. Además del mobiliario fijo, para las pruebas, se agregaron obstáculos donde comúnmente pueden encontrarse personas trabajando.

Se desea verificar que, utilizando el método de optimización de algoritmos geneticos con un número reducido de individuos y de generaciones, se pueden conseguir valores para que el robot describa trayectorias satisfactorias en un ambiente real complejo.

Dado que se consiguieron mejores resultados evaluando las constantes α y β, los algoritmos genéticos ejecutados para este mapa evalúan dichos valores, además de los del campo potencial.

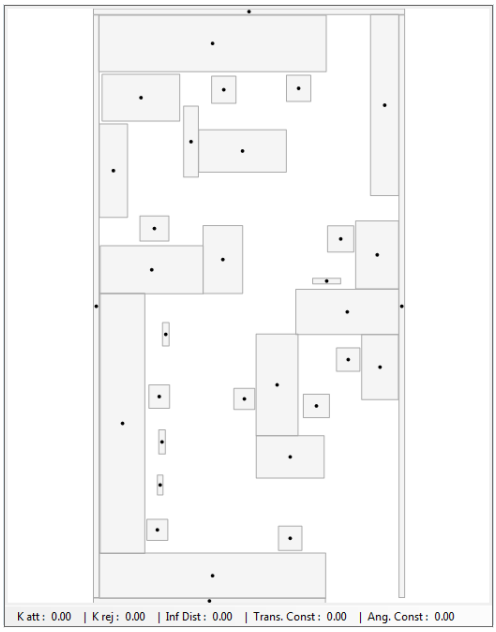


Figura 6.3 Mapa 2 : Laboratorio de Bioróbotica

Prueba 1

Para la prueba 1 se utilizó un punto intermedio, el cual se encuentra en el centro del primer cuarto. Una vez que el robot consigue llegar a este punto, debe pasar a

través de un estrecho pasillo y llegar al objetivo final. El punto intermedio se colocó para evitar la región con forma de "U" que se encuentra enfrente del punto inicial del robot, en dirección al punto final, la cual representa un mínimo local de la trayectoria. Los puntos inicial, final e intermedio se muestran en la figura 6.4.

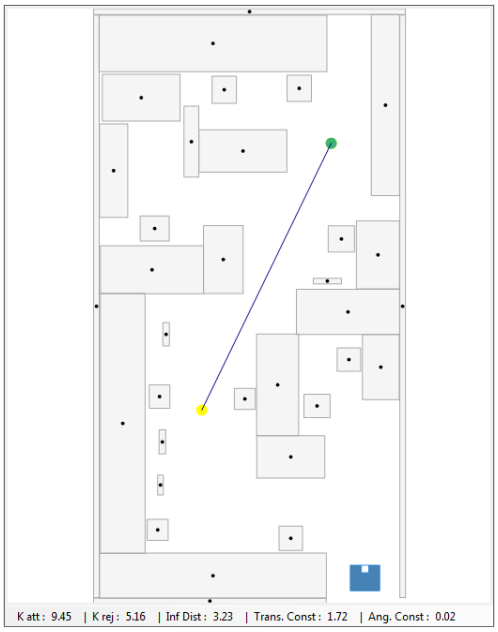


Figura 6.4 Mapa 2 con un punto intermedio

Los parámetros definidos para el algoritmo genético son:

- Número de individuos : 30
- Generaciones : 30
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 1

Con las constantes obtenidas por el algoritmo genético, el robot consiguió llegar al objetivo final. Los valores obtenidos para las constantes de la trayectoria se muestran en la tabla 6.5.

La trayectoria generada por el robot durante la prueba 1, se muestra en la figura 6.5. En ésta se puede observar que un punto crítico de la trayectoria es al pasar por el estrecho pasillo con forma de "L".

Tabla 6.5 Mapa 2: Valores de la prueba 1

Ajuste	0.54474
K_{atr}	9.4531
K_{rep}	5.1567
D_{inf}	3.2266
α	1.7188
β	0.0166

Prueba 2

En la prueba 2 se ejecutó el algoritmo genético utilizando tres puntos intermedios. En ésta el robot debe cruzar el pasillo en forma de "L" tres veces para llegar al objetivo final. Esta prueba se realizó con el objetivo de optimizar los valores de la trayectoria para que se mueva a través del pasillo con forma de "L" independientemente de la dirección en que lo cruce. Con esto se consigue entrenar al robot para trayectorias en ambas direcciones, lo cual la hace más general, que la trayectoria de la prueba 1. Los puntos intermedios utilizados se pueden ver en la figura 6.6.

Los parámetros utilizados en el algoritmo genético son los siguientes:

- Número de individuos : 30
- Generaciones : 30
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 3

El algoritmo genético arrojó los valores de las constantes del campo y de velocidades de la tabla 6.6.

Tabla 6.6 Mapa 2: Valores de la prueba 2

Ajuste	1.3185
K_{atr}	5.9379
K_{rep}	4.8052
D_{inf}	0.9167
α	0.9146
β	0.0010

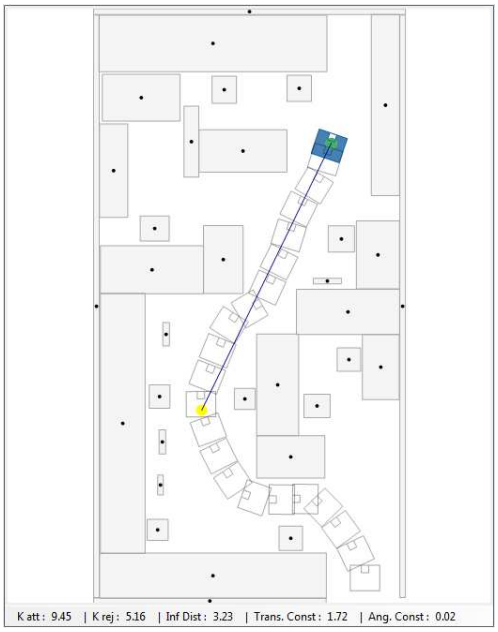


Figura 6.5 Trayectoria de la prueba 1

La trayectoria del robot utilizando los valores de la tabla 6.6, se muestra en la figura 6.7. En esta se puede ver que el robot cruza en ambas direcciones el pasillo en forma de “L” a una distancia apropiada, aunque la distancia entre éste y los otros obstáculos se reduce.

Prueba 3

Por último, se ejecutó una prueba en la que, además de cruzar en ambas direcciones el pasillo en forma de “L”, el robot debe recorrer, en ambas direcciones un corredor estrecho para llegar al punto final. El fin es el mismo que el de la prueba 2, aunque ampliando el tipo de trayectorias que el robot debe recorrer. Los puntos intermedios utilizados en esta trayectoria se pueden observar en la figura 6.8

Los parámetros del algoritmo genético para esta prueba fueron:

- Número de individuos : 30
- Generaciones : 30
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 6

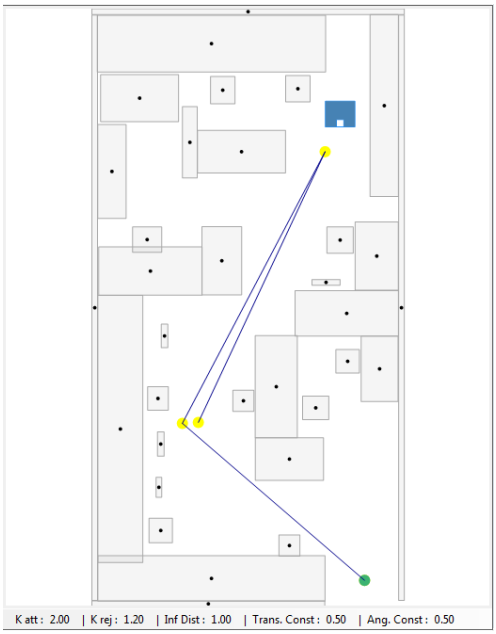


Figura 6.6 Mapa 2 con 3 puntos intermedios.

El individuo con el mejor valor de ajuste, encontrado en la generación 16, arrojó los valores mostrados en la tabla 6.7.

Tabla 6.7 Mapa 2: Valores de la prueba 3

Ajuste	1.4433
K_{atr}	5.8988
K_{rep}	5.9769
D_{inf}	0.8634
α	0.6626
β	0.0259

La trayectoria descrita con los valores del mejor individuo puede observarse en la figura 6.9, en la que el robot mantiene una distancia adecuada al cruzar el pasillo, en ambas direcciones, y además es capaz de navegar por el estrecho pasillo. Al igual que en la prueba 2, el valor de α es menor para esta trayectoria dado que el robot debe ejecutar rotaciones sobre su propio eje sin traslación, para cambiar diametralmente su dirección, por ejemplo, al llegar al final de pasillo.

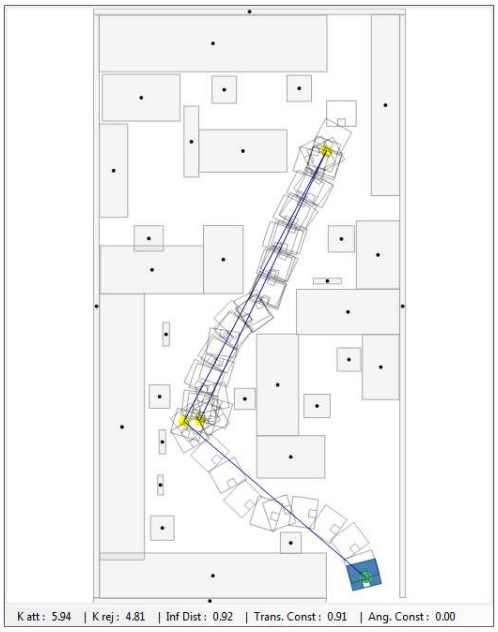


Figura 6.7 Trayectoria de la prueba 2

Con un número pequeño de individuos y de generaciones se consiguió una trayectoria óptima para un ambiente basado en la realidad, complejo y con obstáculos, donde el robot es capaz de navegar, en ambas direcciones, por pasillos estrechos, con un reducido número de puntos intermedios, dada la distribución de los obstáculos.

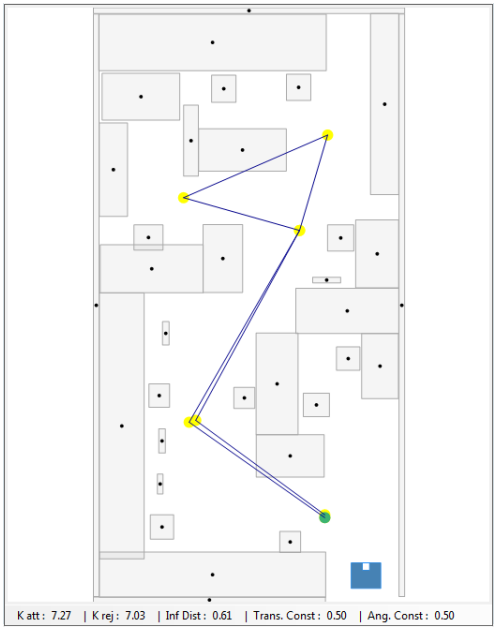


Figura 6.8 Mapa 2 con 6 puntos intermedios

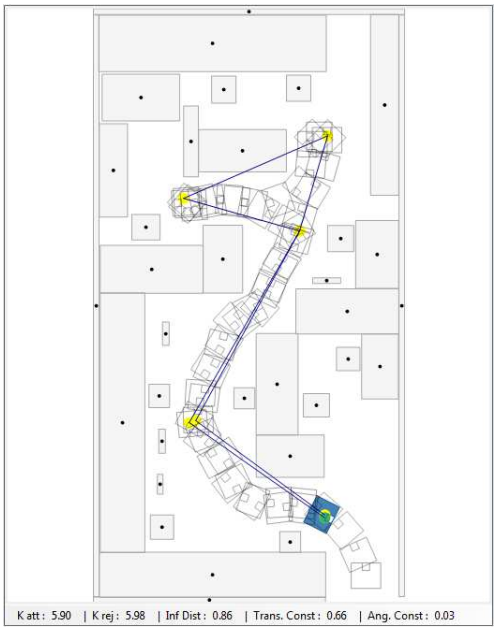


Figura 6.9 Mapa 2 : trayectoria de la prueba 3

6.2.3 Mapa 3 : Oscilaciones en pasillos angostos

Un problema del método de campos potenciales es el que ocurre cuando el robot debe atravesar un pasillo angosto, en el cual, ambas paredes generan fuerzas repulsivas en lados opuestos. Si durante este recorrido, el robot encuentra un cambio en la fuerza repulsiva proveniente de alguno de los lados el pasillo, como pueden ser un obstáculo o un cambio abrupto del ancho del pasillo, la trayectoria descrita por el robot comenzará a oscilar de forma inestable y eventualmente collisionará en alguno de los lados del pasillo.

Para verificar este problema, se generó el mapa de la figura 6.10, el cual es similar al utilizado por Koren y Borenstein (1991), quien identificó este problema como inherente al método de campos potenciales e independiente de su implementación particular. En este mapa, la distancia entre las paredes del pasillo es del doble del ancho del robot.

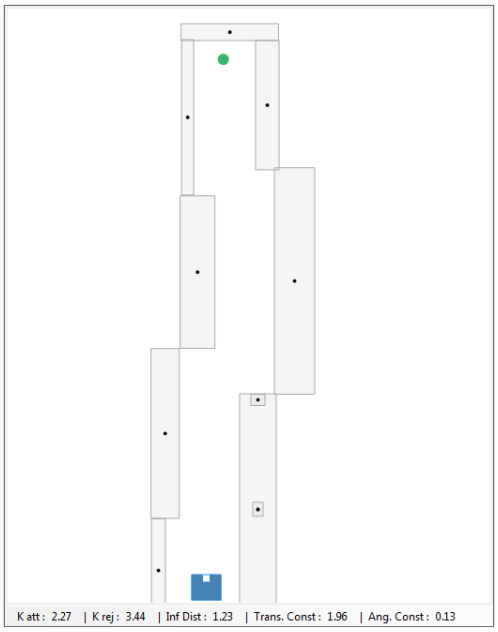


Figura 6.10 Mapa 3 : Pasillo angosto

Dado que el algoritmo genético considera la distancia a los obstáculos, la cual disminuiría con las oscilaciones al acercarse el robot a las paredes del pasillo, y el número de pasos de la trayectoria, los cuales aumentarían cuando el robot no describe una linea recta en un pasillo, se espera que las oscilaciones disminuyan. Los parámetros del algoritmo genético utilizados en esta prueba son:

- Número de individuos : 20
- Generaciones : 20
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 6

Para esta trayectoria se consiguieron, mediante el algoritmo genético, los valores para las constantes del campo potencial y las constates de velocidad expuestas en la tabla 6.8

Tabla 6.8 Mapa 3: Valores de la prueba

Ajuste	0.9535
K_{atr}	2.2663
K_{rep}	3.4381
D_{inf}	1.2311
α	1.9609
β	0.1259

La trayectoria simulada se muestra en la figura 6.11. En esta se puede observar que, aunque siguen presentandose oscilaciones durante la trayectoria, se reducen considerablemente. Durante la trayectoria no existió colisión entre el robot y las paredes del pasillo y puede alcanzar el objetivo final.

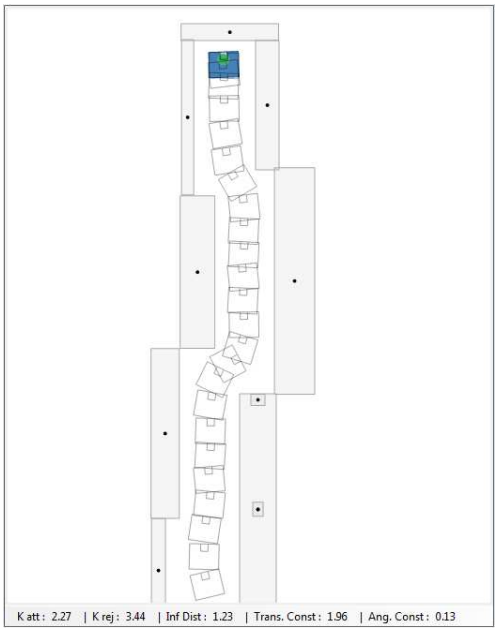


Figura 6.11 Mapa 3 : Trayectoria

6.2.4 Mapa 4 : Objetivo final no alcanzable por obstáculos cercanos

Cuando el objetivo final de la trayectoria se encuentra muy cerca de los obstáculos, para ciertos valores de las constantes del campo potencial, el robot no podrá acercarse a este, debido a que estos obstáculos generar una fuerza repulsiva mayor que la fuerza atractiva.

Este problema fue descrito por Ge y Cui (2000), quienes proponen una nueva función de repulsión la cual también es función de la distancia entre el robot y el objetivo final. El mapa de la figura 6.12 es un mapa similar al utilizado por Ge y Cui para probar la función propuesta. En este, se colocó un obstáculo cerca del objetivo final, a una distancia igual a la parte más ancha del robot, para evitar que no llegue debido a una colisión. Utilizando este mapa se ejecutó un algoritmo genético para observar el comportamiento de las trayectorias ante esta situación. Puesto que el valor de ajuste de los individuos es función de la distancia final del robot al objetivo final y existe penalización si éste no consigue llegar, se espera que los mejores individuos se aproximen más al objetivo final, hasta alcanzarlo.

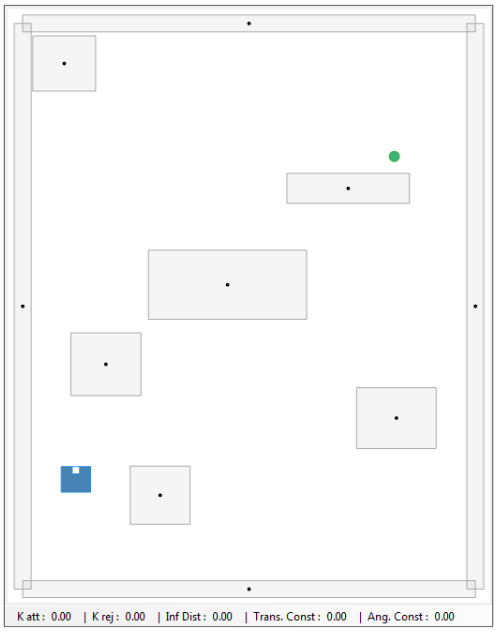


Figura 6.12 Mapa 4 : Obstáculo cerca del punto final

Se ejecutó un algoritmo genético con el cual se pretende encontrar una combinación de valores que permitan al robot viajar seguro y aproximarse a los objetivo final a pesar de su cercanía a los obstáculos. Los parámetros del algoritmo genético utilizado son:

- Número de individuos : 30
- Generaciones : 30
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β
- Puntos intermedios : 0

Los resultados arrojados por el algoritmo genético del mejor individuo se resumen en la tabla 6.9.

Con los valores obtenidos del mejor individuo, se simuló la trayectoria del robot. Ésta se puede observar en la figura 6.13, en la que la posición del robot al final de la trayectoria se encuentra a escasos centímetros del obstáculo. Con esto se observa que, para determinadas trayectorias, es posible encontrar un conjunto de valores que permiten al robot llegar al objetivo final aun cuando este se encuentre muy cerca de los obstáculos.

Tabla 6.9 Mapa 4: Valores de la prueba

Ajuste	0.9535
K_{atr}	2.2663
K_{rep}	3.4381
D_{inf}	1.2311
α	1.9609
β	0.1259

6.2.5 Mapa 5 : Ambiente con múltiples cuartos

El mapa de la figura 6.14 es similar al utilizado por Arámbula y Padilla (2004) para probar un novedoso método denominado campos potenciales adaptables, en el cual, en cada paso de la trayectoria, un algoritmo genético determina el valor de las constantes de las ecuaciones de las fuerzas de atracción y de repulsión. Estas ecuaciones son las utilizadas en el presente trabajo, por lo que se desea verificar si es posible realizar las mismas trayectorias.

Utilizando este método, su robot pudo describir trayectorias satisfactorias que lo llevaron al objetivo final en un ambiente complejo. Este ambiente simula un piso de hospital con 5 cuartos con diferentes propósitos cada uno. Los cuartos son numerados, en sentido contrario al de las manecillas del reloj comenzando con el de la esquina superior izquierda, del 1 al 5. El primero simula un cuarto de almacenamiento, el segundo y el quinto cuarto son habitaciones de hospital con camas colocadas en diferente orientación, el tercer cuarto sería una sala de conferencias y el siguiente, un cuarto vació con obstáculos aleatorios. En este ambiente, simuló trayectorias desde el pasillo de entrada, el cual se encuentra en la esquina superior izquierda, hasta cada cuarto, utilizando dos puntos intermedios para cada trayectoria: uno al final del pasillo de entrada y otro en el umbral de la entrada a cada cuarto.

En esta prueba se desea encontrar un conjunto de valores para las constantes de la trayectoria utilizando el algoritmo genético, con un número pequeño de generaciones y de individuos, que permitan al robot trasladarse de forma similar a cada cuarto desde el pasillo de entrada. Por esta razón, se ejecutó el algoritmo genético 5 veces, uno por cada trayectoria hacia cada cuarto con los siguientes parámetros:

- Número de individuos : 20
- Generaciones : 20
- Parámetros a evaluar : K_{atr}, K_{rep}, D_{inf}, α y β

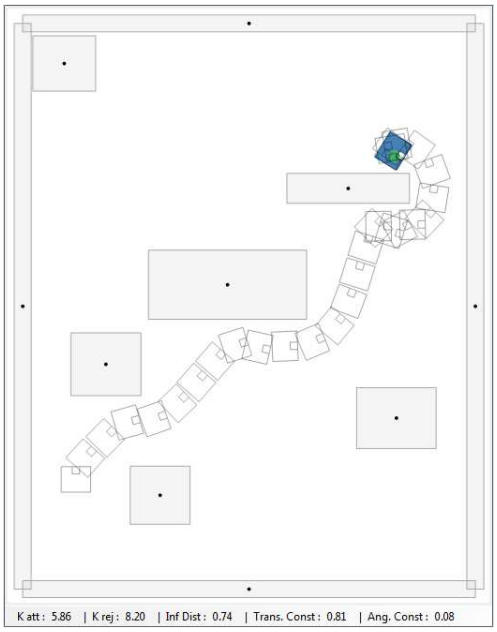


Figura 6.13 Mapa 4 : Trayectoria de la prueba

- Puntos intermedios : 2

Los resultados de los mejores individuos para cada cuarto se presentan en la tabla 6.10.

Las trayectorias simuladas utilizando las valores obtenidos por los algoritmos genéticos, se presentan en la figura 6.15.

Como se puede observar, los resultados de las trayectorias simuladas son similares a los obtenidos por Cosio y Castañda (2004). En ninguna trayectoria el robot colisionó

Tabla 6.10 Mapa 5: Valores de las pruebas

	Cuarto 1	Cuarto 2	Cuarto 3	Cuarto 4	Cuarto 5
Ajuste	1.0048	1.3252	1.6619	1.8724	1.8565
K_{atr}	1.4071	2.9694	7.0315	2.2663	4.7270
K_{rep}	8.3595	9.6484	9.1016	8.2423	8.3595
D_{inf}	0.4793	0.4520	2.1742	0.7664	0.8074
α	0.5554	0.3211	0.4773	0.1805	0.40704
β	0.0400	0.3758	0.0790	0.0088	0.4538

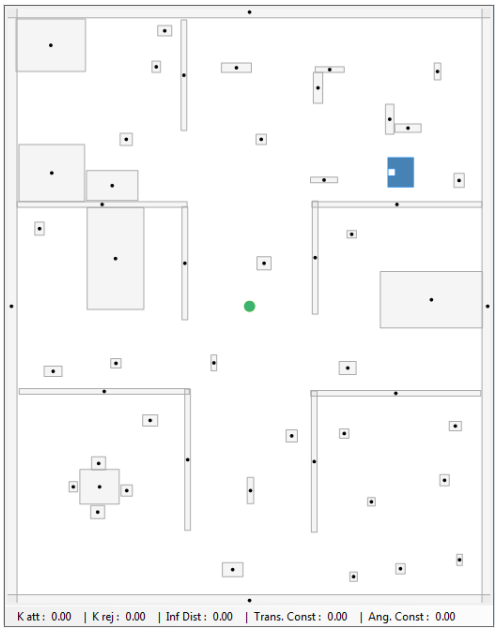
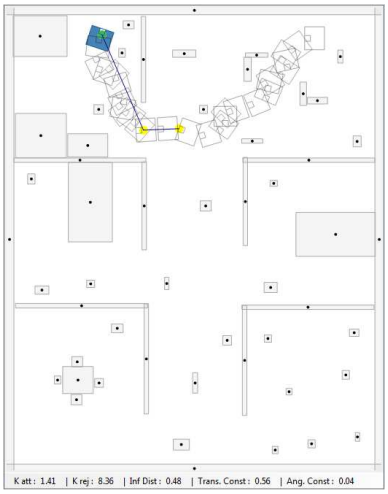
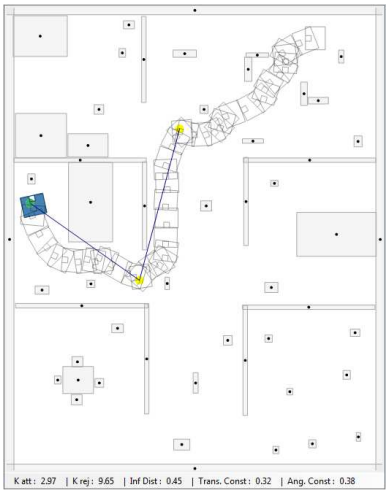


Figura 6.14 Mapa 5 : Piso de hospital

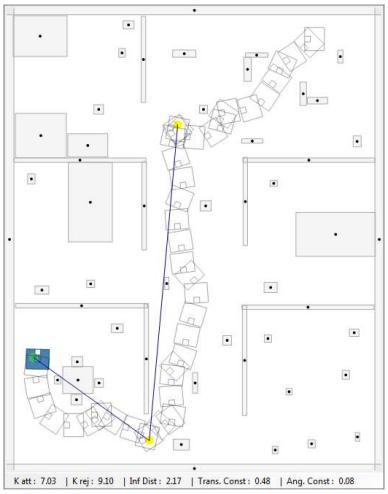
con los obstáculos, aunque en la trayectoria del cuarto 2, el robot pasó muy cerca de los obstáculos, debido en parte a que, el objetivo final se encontraba muy cerca de la pared.



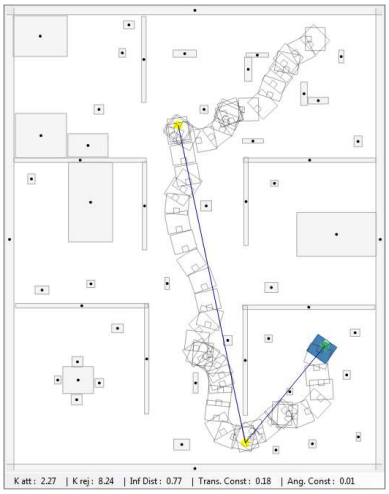
(a) Cuarto 1



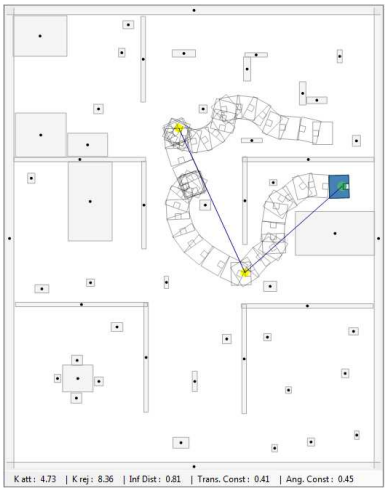
(b) Cuarto 2



(c) Cuarto 3



(d) Cuarto 4



(e) Cuarto 5

Figura 6.15 Mapa 5 : Trayectorias hacia cada cuarto

6.3 Resultados experimentales

Los valores obtenidos por el algoritmo genético para los mapas 1 y 2 de la sección 6.2 se probaron utilizando el robot Justina, dado que estos mapas son sus espacios comunes de trabajo y son accesibles dada la cercanía al robot.

Durante las pruebas experimentales surgió el problema de los errores de odometría inherentes al sistema, los cuales añaden un error a la posición estimada del robot . Estos errores surgen por pequeñas diferencias entre las medidas del robot real y el simulado. Además, un factor importante que causa errores odométricos son los deslizamientos de las llantas durante la trayectoria.

Estos errores afectan el comportamiento de los campos potenciales al modificar la fuerza de atracción del robot, dado que esta depende de la posición del mismo, por tanto, la fuerza resultante se ve ligeramente modificada. Además, los errores odométricos provocan errores en la posición final del robot, dado que el robot estima que la posición errónea concuerda con la posición del objetivo final y detiene la trayectoria. Dado que el sensor láser se mueve con el robot, la posición de los obstáculos es relativa a este y la fuerza de repulsión no se ve alterada, por lo que la evasión de obstáculos es realizable aun con el error antes mencionado.

El sensor láser generó lecturas erróneas cuando los obstáculos se encuentran muy alejados de este, las cuales son ignoradas por el cálculo del campo potencial, como se mencionó en el capítulo 4 . Para su identificación, estas lecturas se dibujan en color negro en las figuras presentadas.

Cada experimento se realizó tres veces. Durante éstos se guardó la posición del robot, la cual fue utilizada para presentar las trayectorias descritas a continuación. Las trayectorias son mostradas junto con el mapa con el cual se ejecutó el algoritmo genético aunque durante la pruebas reales no se ejecuta ninguna operación sobre los obstáculos del mapa. En éstos, la posición de los obstáculos durante el recorrido difieren sutilmente. Esto es causado en parte por los errores de medición al crear el mapa. Aunado a esto, dependiendo de la altura del láser, el robot puede ver ciertos obstáculos que al crear el mapa fueron tomados en cuenta. Además el error de odometría que experimenta el robot hacen que la posición de este en el mapa no sea precisa. A pesar de esto, en todos los experimentos el robot pudo realizar las trayectorias, evadiendo los obstáculos que se encontraban a su paso hasta llegar al destino satisfactoriamente.

6.3.1 Experimento 1

El mapa de la figura 6.1, en el cual se realizó el experimento 1, está basado en el pasillo que se encuentra al exterior del laboratorio de biorobótica. Este se puede ver en la figura 6.16.

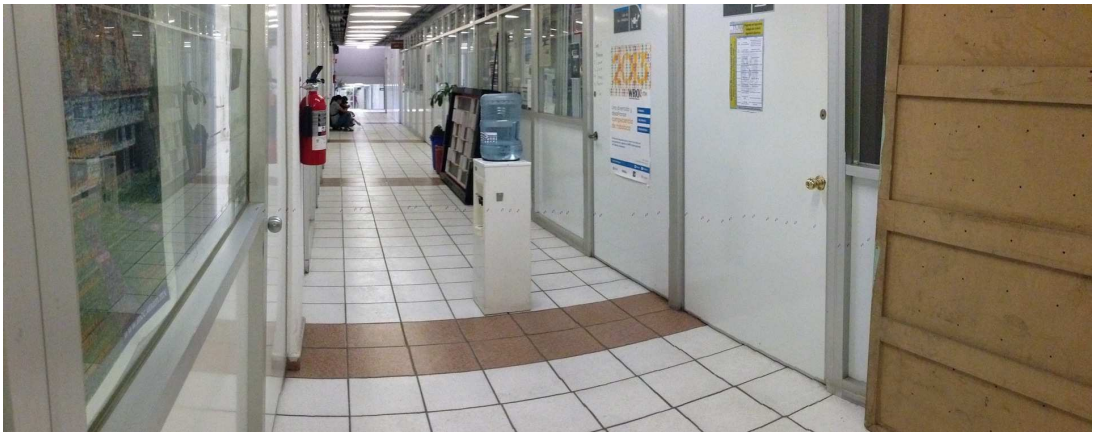


Figura 6.16 Pasillo utilizado para el experimento 1

Como obstáculo se utilizo un dispensador de agua posicionado cerca del centro del pasillo. El robot se colocó en el centro del pasillo, a 2 metros del obstáculo. El robot debía navegar hacia el obstáculo, evadirlo y llegar al objetivo final que se encuentra igualmente a 2 metros del pasillo cruzando el obstáculo.

En la figura 6.17 se observa la información del sensor láser al inicio de la prueba. Como se observa en la imagen, el mapa generado y el ambiente real difieren ligeramente.

Para este experimento se utilizaron los valores de la tabla 6.4. Las trayectorias descritas por el robot durante la prueba se pueden observan en la figura 6.18.

Como se puede observar, las trayectorias son similares a la descrita durante la prueba simulada. Las diferencias entra las posiciones finales de las tres pruebas son debidas a los errores de odometría. El robot satisfactoriamente evadió el obstáculo y llegó al objetivo final.

La imagen 6.19 es una foto del robot Justina evadiendo el obstáculo durante una de las pruebas del experimento 1.

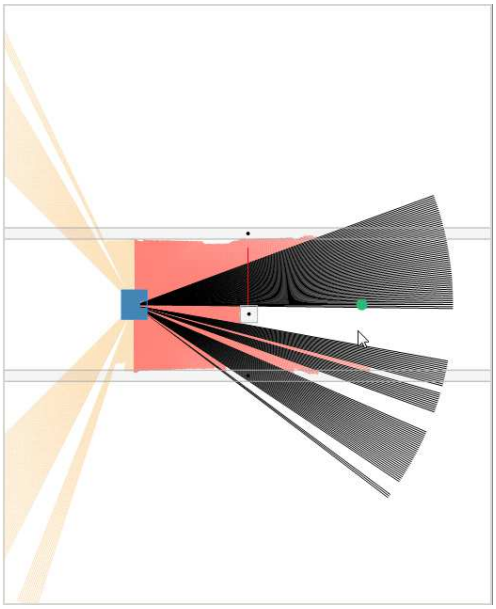
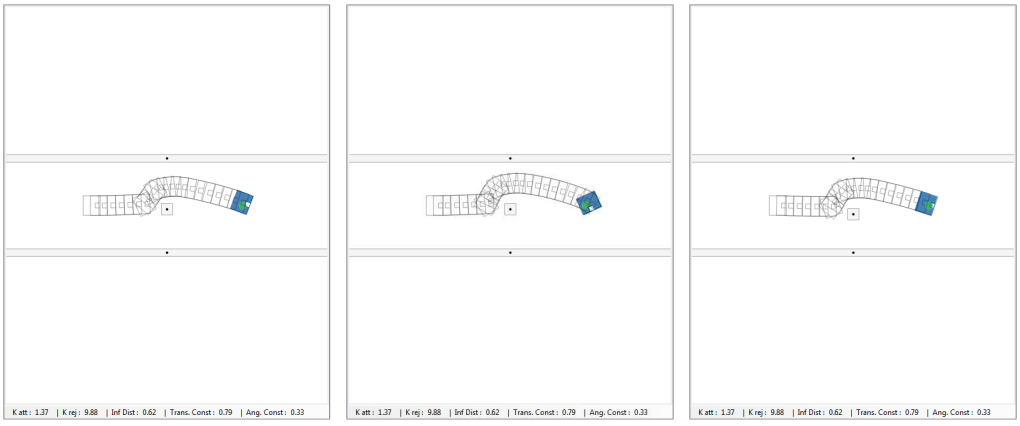


Figura 6.17 Lecturas del láser durante el experimento 1



(a) Prueba 1 (b) Prueba 2 (c) Prueba 3

Figura 6.18 Trayectorias obtenidas durante el experimento 1)

Figura 6.19 Robot Justina durante el experimento 1.

6.3.2 Experimento 2

Este experimento se realizó en el laboratorio en el que esta basado el mapa de la figura 6.3. En este laboratorio se desarrolló el robot Justina. La posición del mobiliario crea un ambiente complejo para navegación autónoma. En la figura 6.20 se puede observar el laboratorio en donde se ejecutó el experimento 2.

Los valores de las constantes utilizados para este experimento fueron los de la tabla 6.7, con los cuales se obtuvo la mejor trayectoria durante las simulaciones del mapa 2, en el cual se basó esta prueba.

Para este experimento se utilizaron los puntos intermedios de la figura 6.21 de la misma manera que lo realizado por Savage y Hernández (2008) . Estos sirvieron para guiar al robot fuera del punto de mínimo local, justo al inicio de la trayectoria, y a través del pasillo. Como se puede observar, la posición de estos puntos difiere de los utilizados durante la optimización mediante el algoritmo genético, sin embargo, los valores de las constantes, al estar optimizados para esta configuración de obstáculos, por lo que deben ser capaces de evadirlos de un modo similar al de las simulaciones.

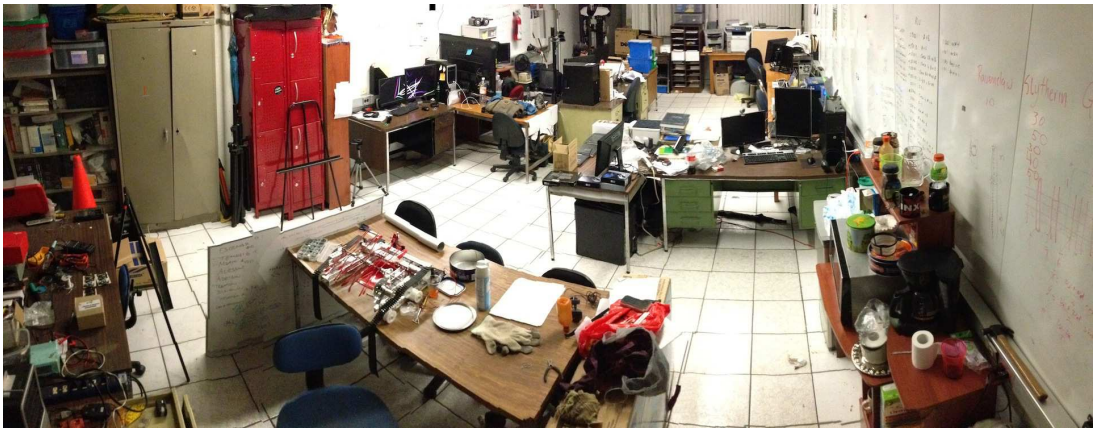


Figura 6.20 Laboratorio utilizado para el experimento 2.

De igual manera, la configuración de los obstáculos no fijos es distinta al de las pruebas del mapa 2, al existir un menor número de personas trabajando durante este experimento. El robot debe ser capaz de detectar la nueva configuración mediante el láser y, reactivamente, evadir los obstáculos.

En este experimento se posicionó el sensor láser a la altura de los bordes de las mesas, sin embargo, en dos de estas se colocaron objetos para poder ser detectadas puesto que la altura era diferente a la de la mayoría.

En la figura 6.22 se puede observar la información del sensor láser durante la trayectoria del robot, donde se verifican las discrepancias existentes entre los obstáculos en el mapa y los obstáculos reales detectados por el sensor láser, con lo que se puede deducir que no es necesaria una información exacta del mapa en el cual se ejecuta el algoritmo.

Las posiciones del robot guardadas durante las trayectorias se muestran en la figura 6.23. En estas se puede observar cómo el robot alcanza todos los puntos intermedios y finaliza en el objetivo final. Las trayectorias difieren entre sí ligeramente, dependiendo de los obstáculos detectados por el sensor láser y el ruido que este genera. Igualmente, los errores odométricos generan ligeros cambios en la posición del robot en cada paso diferentes para cada trayectoria. Esto hace que la distancia del robot a los obstáculos en un mismo paso entre trayectorias sea diferente, modificando las fuerzas generadas por los campos potenciales.

Dado que se habla de un paradigma reactivo, las diferencias entre la posición de los obstáculos del mapa generado y la ubicación real de estos generó diferencias despreciables en cuanto a la forma de las trayectorias esperadas.

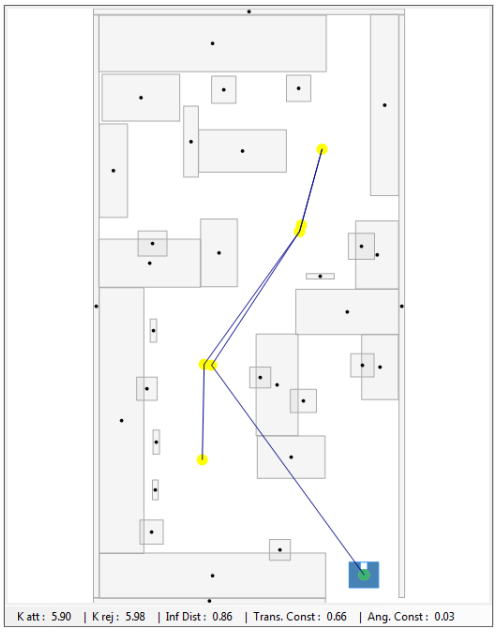


Figura 6.21 Puntos intermedios del experimento 2

En ninguna de las trayectorias, existió colisión entre el robot y los obstáculos, por lo que la evasión de obstáculos estáticos fue realizada satisfactoriamente.

En los tres experimentos, el robot se desplazó de manera continua de un punto intermedio a otro, a una velocidad segura para el robot, hasta alcanzar el objetivo final.

La imagen 6.24 es una foto del robot Justina evadiendo el obstáculo durante una de las pruebas del experimento 2.

(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figura 6.22 Información del sensor durante el experimento 2

(a) Prueba 1 (b) Prueba 2 (c) Prueba 3

Figura 6.23 Trayectorias obtenidas durante el experimento 2

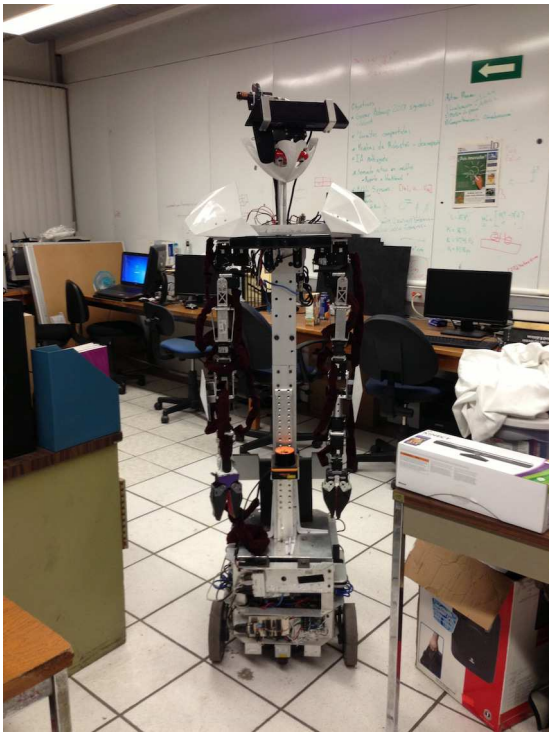


Figura 6.24 Robot Justina durante el experimento 2.

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo se resumen los resultados obtenidos durante la realización de esta tesis y se prestan algunas consideraciones que se deben tomar al implementar el sistema desarrollado.

Así mismo, se plantean algunas propuestas que se encuentran fuera del alcance de este trabajo para continuar con la investigación iniciada con este proyecto.

7.1 Conclusiones

Utilizando el modelo cinemático de un robot móvil de tipo par diferencial y las especificaciones de un sensor láser, se simuló el sistema de navegación del robot Justina. Como se pudo observar en el capitulo 6 las trayectorias simuladas y las realizadas durante los experimentos con el robot son muy similares, por lo que los modelos de simulación utilizados son suficientemente aproximados.

Se comprobó la eficacia del algoritmo de navegación implementado haciendo uso del método de campos potenciales para la evasión de obstáculos, tanto en simulación como en pruebas reales en el robot Justina. Sin embargo, el problema de quedar atrapado en una punto de mínimo local sigue sin resolverse. Para ésto, es posible utilizar puntos intermedios que guíen al robot entre regiones, escapando de estos puntos.

Se verificó la efectividad de los algoritmos genéticos con un número reducido de iteraciones para encontrar un conjunto de parámetros que permiten mejorar la trayectoria del robot al utilizar el método de campos potenciales. Se probó también que este conjunto de parámetros se puede extender a las constantes de la ecuación de control

de las velocidades de las llantas con lo que se obtendrán trayectorias más continuas y mejores en cuanto a velocidad de ejecución.

Aunque al utilizar en el algoritmo genético un número mayor de individuos y de generaciones se encuentran mejores trayectorias, un número de aproximadamente 20 individuos y 20 generaciones fue suficiente, en la mayoría de las pruebas, para generar trayectorias suficientemente buenas que permiten al robot mantenerse un distancia segura de los obstáculos y llegar al objetivo final.

Durante las simulaciones se encontró que un láser con un ángulo de detección mayor genera mejores trayectorias, al tener más información sobre los obstáculos alrededor del robot, sin embargo el algoritmo genético puede adecuar los parámetros para sensores con rangos menores.

Un problema encontrado durante las pruebas efectuadas utilizando el robot real fue la altura de los obstáculos. Estos deben de tener una altura mínima para poder ser detectados por el sensor láser y actuar en consecuencia. Si la altura de estos es menor que la altura del sensor láser, no se obtendrá información sobre su posición y, si estos se encuentran en la trayectoria del robot, se producirá una colisión. Los obstáculos identificado como peligrosos en este sentido son las sillas, puesto que láser solo ve el respaldo de estas y las mesas con una altura menor a la de la mayoría.

A pesar de los errores de odometría que presentó el robot, la evasión de obstáculos se realizó de manera satisfactoria. El principal efecto de estos es un error de distancia entre la posición final del robot y la posición del objetivo final.

7.2 Trabajo Futuro

En el presente trabajo se utilizó el modelo cinemático de los robots de par diferencial. Éste no toma en cuenta algunos factores que modifican el movimiento del robot real, como lo haría su ecuación dinámica. Factores como las fricciones, momentos de inercias, masas y características eléctricas de los motores deben ser tomados en cuenta para mejorar el simulador de los movimientos del robot.

En la ecuación del control de las llantas, se utiliza solamente el error de ángulo entre la orientación del robot y la posiciona deseada por lo que, cuando el robot llega al destino, las velocidades se establecen repentinamente en cero. Esto origina que el robot se detenga abruptamente al llegar al objetivo final. Con las velocidades máximas utilizadas durante la prueba real, este cambio no generó problemas, sin embargo, si se desean utilizar velocidades mayores, esto puede originar dificultades debido a la

inercia del robot como oscilaciones respecto a su vertical o colisiones. Para reducir esto, se debe obtener un ley de control que tome en cuenta el error de distancia y las velocidades actuales de las llantas para determinar las velocidades siguiente.

Los obstáculos se representaron como rectángulos orientados a los ejes coordenados con lo que las simulaciones del sensor láser y de las colisiones requieren muy poco poder de cómputo. La representación de los obstáculos se puede extender a otras formas geométricas como círculos y polígonos irregulares para representar con mayor fidelidad los obstáculos en un ambiente.

Como se mencionó anteriormente, el láser presenta dos limitantes: una es el ángulo de detección del láser y otra es la altura a la que debe posicionarse el láser, con la esperanza de que se pueda detectar la mayoría de los obstáculos. Esto se debe a que el láser proporciona información sobre los obstáculos que se encuentran en un plano paralelo al piso. Actualmente existen sensores que proporcionan información sobre el ambiente en 3D, con lo que se podrían evadir obstáculos sin importar su altura o forma. Lamentablemente la simulación de este tipo de sensores requerería un gran número de cálculos, por lo que los tiempos de simulación aumentarían demasiado. Un enfoque que resolvería este problema sería la utilización de técnicas de cómputo paralelo para simular sensores más complejos.

Además de los operadores del algoritmo genético implementados, se han desarrollado otros con los cuales se podrían obtener mejores soluciones en un menor número de iteraciones. Para comprobar esto, haría falta un riguroso análisis sobre los resultados modificando estos operadores y variando los parámetros del algoritmo genético en diferentes mapas.

Referencias

Arámbula, F. C., y Padilla, M. C. (2004). Autonomous robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, *40*(9 - 10), 1141 - 1156. doi: 10.1016/j.mcm.2004.05.001

BasicMicro. (2010). *B0098 – roboclaw 2 channel 30a motor controller.* (Datasheet)

Bekey, G. (2005). *Autonomous robots: From biological inspiration to implementation and control*. Mit Press.

Bogacki, P., y Shampine, L. (1996). An efficient runge-kutta (4, 5) pair. *Computers & Mathematics with Applications*, *32*(6), 15–28.

Brooks, R. (1986, mar). A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, *2*(1), 14 - 23. doi: 10.1109/JRA.1986.1087032

Cen, Y., Wang, L., y Zhang, H. (2007, oct.). Real-time obstacle avoidance strategy for mobile robot based on improved coordinating potential field with genetic algorithm. En *Control applications, 2007. cca 2007. ieee international conference on* (p. 415 -419). doi: 10.1109/CCA.2007.4389266

Cook, G. (2011). *Mobile robots: Navigation, control and remote sensing*. Wiley.

Cosio, F. A., y Castañda, M. P. (2004). Autonomous robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, *40*(9 - 10), 1141 - 1156. doi: 10.1016/j.mcm.2004.05.001

Ge, S., y Cui, Y. (2000). New potential functions for mobile robot path planning. *Robotics and Automation, IEEE Transactions on*, *16*(5), 615-620. doi: 10.1109/70.880813

Gottschalk, S., Lin, M. C., y Manocha, D. (1996). Obbtree: a hierarchical structure for rapid interference detection. En *Proceedings of the 23rd annual conference on computer graphics and interactive techniques* (pp. 171–180). New York, NY, USA: ACM. doi: 10.1145/237170.237244

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press.

Khatib, O. (1985, mar). Real-time obstacle avoidance for manipulators and mobile robots. En *Robotics and automation. proceedings. 1985 ieee international conference on* (Vol. 2, p. 500 - 505). doi: 10.1109/ROBOT.1985.1087247

Koren, Y., y Borenstein, J. (1991, apr). Potential field methods and their inherent limitations for mobile robot navigation. En *Robotics and automation, 1991. proceedings., 1991 ieee international conference on* (p. 1398 -1404 vol.2). doi: 10.1109/ROBOT.1991.131810

Latombe, J. (1991). *Robot motion planning*. Kluwer Academic.

LTD., H. A. C. (2005, octubre). *Scanning laser range finder urg-04lx.* (Specifications)

Mataric, M. J. (1992). Behavior–based control: Main properties and implications. En *Proceedings, ieee international conference on robotics and automation, workshop on architectures for intelligent control systems* (pp. 46–54).

Meystel, A. (1991). *Autonomous mobile robots: Vehicles with cognitive control*. World Scientific.

Michell, M. (1998). *An introduction to genetic algorithms*. Mit Press.

Murphy, R. (2000). *Introduction to ai robotics*. Mit Press.

Petrilli Barcelo, A. E. (2007). *Control visual de un robot móvil khepera ii*. Tesis de Master no publicada, Instituto Politécnico Nacional.

Savage, J., y Hernández, A. (2008). Local autonomous robot navigation using potential fields. *Mobile Robots motion planning*, 1.

Schneider, P., y Eberly, D. (2003). *Geometric tools for computer graphics*. Boston.

Shi, P., y Cui, Y. (2010). Dynamic path planning for mobile robot based on genetic algorithm in unknown environment. En *Control and decision conference (ccdc), 2010 chinese* (p. 4325-4329). doi: 10.1109/CCDC.2010.5498349

Sivanandam, S., y Deepa, S. (2008). *Introduction to genetic algorithms*. Springer London, Limited.

Tuncer, A., y Yildirim, M. (2012). Dynamic path planning of mobile robots with improved genetic algorithm. *Computers and Electrical Engineering*, *38*(6), 1564 - 1572. doi: 10.1016/j.compeleceng.2012.06.016

Vadakkepat, P., Tan, K. C., y Ming-Liang, W. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. En *Evolutionary computation, 2000. proceedings of the 2000 congress on* (Vol. 1, p. 256 -263 vol.1). doi: 10.1109/CEC.2000.870304

Watson, K., Hammer, J., Reid, J., Skinner, M., Kemper, D., y Nagel, C. (2012). *Beginning visual c# 2012 programming*. Wiley.