UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA MECÁNICA

PLANIFICACIÓN DE TRAYECTORIAS PARA ROBOTS MÓVILES EN ENTORNOS NO DINÁMICOS EMPLEANDO EL ALGORITMO RRT

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:
INGENIERO MECATRÓNICO

JORGE SALVADOR PAREDES MERINO

PROMOCIÓN **2010-II**

LIMA-PERÚ

2 012

Dedicatoria

Dedico el presente trabajo a mi familia por su apoyo incondicional, lo cual me permitió culminar mis estudios así como también un agradecimiento especial a los docentes de mi facultad por la orientación brindada a lo largo de mis 5 años de estudio. Finalmente agradecer a mis compañeros por las gratas experiencias compartidas en esta importante etapa de mi vida.

Agradecimientos

Quería agradecer de manera especial al Ing. José F, Oliden Martínez por la ayuda y asesoramiento brindado para la obtención de los robots móviles Moway, los cuales permitieron implementar los algoritmos desarrollados físicamente en la presente tesis. A su vez dichos robots quedan en manos de INIFIM para el uso posterior de algún proyecto que algún alumno desarrolle en un futuro.

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

PRÓLOGO

Hoy en día, los robots son parte de nuestra vida. Los podemos encontrar como juguetes, móviles exploradores, brazos, humanoides, entre otros. Pero algo en común que requieren los robots es interrelacionarse con su entorno, el cual limita su accionar y le obliga a adecuarse a tales condiciones. En el caso de los móviles lo que se requiere primordialmente es trasladarlos de un lugar específico a otro requerido para posteriormente ejecutar alguna tarea. El problema de controlar un robot móvil y la planificación de su ruta esta dado por el hecho de que su ambiente de trabajo está condicionado a cambios, sea con alta o baja tendencia. Por tal motivo se han desarrollado a lo largo de los últimos 50 años diversos algoritmos y técnicas que permiten interactuar apropiadamente al robot con su medio. Por ejemplo, en el caso de robots manipuladores, sus articulaciones no deben colisionar con objeto alguno. Pero el cálculo de dichas trayectorias (de robots móviles y manipuladores) involucra factores tales como la geometría, cinemática, dinámica, entre otros que son los que limitan al robot. En la presente tesis se propone el uso de técnicas estocásticas en lo referente al planeamiento de rutas o caminos para robots móviles en ambientes no dinámicos, es decir, se considerará que el entorno no presenta obstáculos en movimiento. Para éste estudio, se ha organizado el trabajo en cinco capítulos que serán expuestos de la siguiente forma:

En el capítulo I, se realiza una introducción a esta tesis, con los antecedentes, el motivo de la investigación, los objetivos, alcances y limitaciones.

En el capítulo II, se presenta una revisión de la literatura sobre el planeamiento de trayectorias. Además se presentan definiciones básicas que han de permitir un mejor entendimiento, así como también una descripción de los principales algoritmos de planemiento de caminos.

En el capítulo III, se explica lo referente al algoritmo utilizado en la presente tesis desde sus elementos, el pseudocódigo y los algoritmos evolucionados a partir de éste.

En el capítulo IV, se presentan los resultados de las simulaciones realizadas así como la performance de evaluación.

En el capítulo V, se explicará lo referente a la implementación de la plataforma y la interfaz que se utilizó para validar el algoritmo en robots móviles reales.

Finalmente se presentan las conclusiones y recomendaciones que pueden servir para futuras investigaciones. El apéndice A presenta de manera suscinta algunos antecedentes históricos en robótica móvil a partir de la Segunda Guerra Mundial. En el apéndice B se aborda las principales caraterísticas del robot Moway; en el apéndice C, los recursos utilizados del campo de Visión Artificial y en el apéndice D los algoritmos y funciones principales codificadas.

Con este pequeño trabajo se espera generar experiencia académica en la planificación de movimiento para posteriores aplicaciones así como también tener una plataforma de pruebas para robots móviles con estados retroalimentados que son la posición y orientación de los mismos.

Con la compañía MiniRobots se firmó un convenio para poder utilizar los recursos de su software, sin costo alguno para aplicaciones académicas, es decir, la persona interesada debe ponerse en contacto con ellos.

CAPÍTULO 1
INTRODUCCIÓN

1.1. Antecedentes

Entre 1966 y 1972 se desarrollo el robot Shakey (ver Figura 1.1, tomada de [2]), desarrollado por el Centro de Inteligencia Artificial del SRI[1], el cual fue realmente fundamental no sólo para la robótica móvil sino para el desarrollo de la "Inteligencia Artificial", ya que fue el primer robot móvil en visualizar su ambiente y poder interpretarlo. El robot tenía una cámara de televisión, un telémetro, *bumpers*[2] y un sistema de video inalámbrico que le permitía localizar elementos y navegar por ellos. Si bien el robot tenía movimientos desiguales, a menudo absurdos; esto no impidió que dicho robot entrase en los libros de Historia (actualmente dicho robot se encuentra en el Museo de la Historia de la Computadora en California, Estados Unidos) por ser la primera máquina autónoma capaz de localizar objetos, moverse alrededor de ellos y luego explicar la lógica utilizada para hacerlo. Este trabajo sirvió de inspiración para muchos investigadores, en consecuencia, tanto la investigación y la variedad de diseños de robots móviles creció vertiginosamente (ver Apéndice A).

[1]"Stanford Research Institute", es el nombre de fundación(1946) de este centro que pertenecía a la Universidad de Standford. Posteriormente en 1970 se separó de dicha casa de estudio, siendo autónomo y se le llama desde entonces "SRI International".

[2]Término en inglés referido a los parachoques

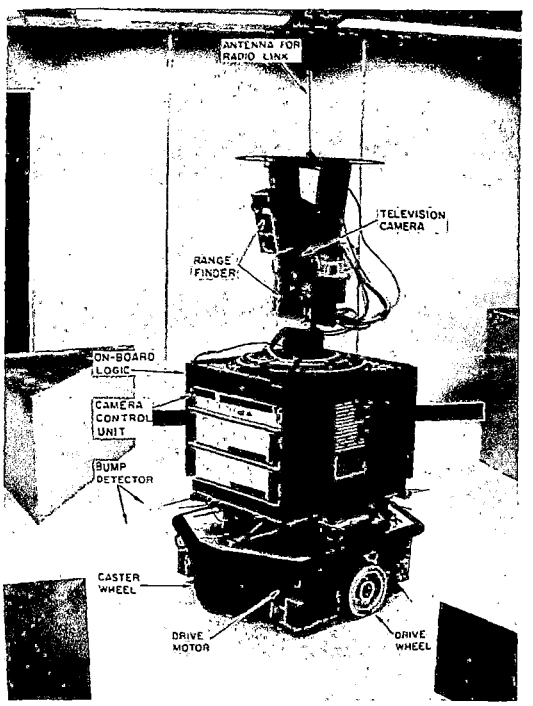

Figura 1.1: Robot Shakey(1968).

Siendo ***Motion Planning***[3], desde los años 70 una emergente y crucial área productiva de investigación en la robótica lo explica **Jean-Claude Latombe** [4] en [3]. El problema básico de la planificación de movimiento es: "Encontrar un camino libre de colisiones para un robot (rígido o articulado) dentro de un espacio con obstáculos rígidos y estáticos". Este problema que es puramente geométrico puede lucir geométricamente simple; salvo para robots con pocos grados de libertad, es completamente difícil. La planificación de movimiento se convirtió en un tópico de investigación durante la primera década, entre los principales trabajos se tiene el de Nilsson que introdujo el método de visibilidad de grafo (combinado con **A***[5]) para encontrar el caminos más corto para un robot representado por un punto con obstáculos poligonales convirtiéndose en una técnica enormemente popular.

[3]Planificación de Movimiento, término utilizado en robótica para el proceso de movimientos discretos de un robot en un mundo 2D o 3D con obstáculos

[4]Renombrado investigador mundial en Motion Planning y es el autor más citado en el campo. Es profesor en la escuela de Ingeniería de la Universidad de Stanford.

[5]A Star, es un algoritmo de Path Planning.

A inicios de los años 80, Lozano-Pérez introdujo el concepto de la "Configuración del espacio del robot", el cual impacto en *Motion Planning* más que alguna otra idea previa. El robot es representado como un punto llamado configuración, en un parámetro que contiene información de los grados de libertad del robot, la configuración del espacio. Los obstáculos son regiones prohibidas, siendo el complemento de éstos el espacio libre y la unión de éstos la configuración del espacio. El ***Path Planning***[6] de un robot con dimensiones es reducido al problema del planeamiento de un punto en un espacio que tiene muchas dimensiones tanto como los grados de libertad del robot. Posteriormente varias extensiones del problema básico han sido estudiados:

- Obstáculos en movimiento
- Restricciones cinemáticas y dinámicas
- Optimización de trayectorias
- Coordinación de múltiples robots

Asimismo las herramientas clásicas de geometría diferencial fueron usadas para estudiar las múltiples estructuras de una configuración de espacio, junto con su más específicas propiedades de topología, de geometría y de algebra. A mediados de los años 80, los planificadores de trayectoria más sofisticados lograron con dificultad calcular rutas de libre colisión para objetos planos trasladando y rotando en espacios de trabajos de 2 dimensiones.

Los conceptos físicos, tales como la fuerza y fricción, fueron elegantemente mapeados en esta representación. La mayoría de estos resultados tomaron un rol crucial en el entendimiento de los problemas de *Motion Planning*, especialmente en **sistemas no holónomos**[7] y de planeamiento óptimo.

[6]Planificador de camino o ruta, término referido al cálculo de la trayectoria. Esta incluido en Motion Planning.

[7]referido a sistemas que presentan restricciones cinemáticas y/o dinámicas que hacen que los grados de libertad sean dependientes

Es importante mencionar también que en esta década el problema de *Path Planning* atrajo el interés de la comunidad de Ciencias de la Computación, obteniéndose algoritmos que crecían exponencialmente con el número de grados de libertad, la mayoría de estos no fueron implementados, pero ayudaron a calibrar su complejidad y el entendimiento de su naturaleza. Con respecto a las técnicas populares de los 80's, básicamente son 2:

- Aproximación de Descomposición de Celdas, donde el espacio libre es representado por una colección de celdas.
- Campo potencial, basado en atracción y repulsión.

Ambos enfoques trabajan bien en espacios de 2 y 3 dimensiones, pero para mayores grados de libertad no tuvieron el éxito esperado, también a partir de la década de los 90's se empezó a tomar con mayor interés los sistemas no holónomos, es por tal razón que nuevos algoritmos de planificación de tipo aleatorio se desarrollaron tales como PRM y RRT. Desde aquel entonces increíbles progresos han sido desarrollados, lográndose planeadores que resuelven problemas complejos de tipo práctico, algunos con robots de muchos grados de libertad en entornos complejos. Así como el problema básico se han encontrado soluciones usadas en la práctica, pero que aún requieren mayor investigación. Por ejemplo su costo computacional es aún alto, por ende se siguen desarrollando y optimizando dichas técnicas.

Tal como lo indica Jean-Claude Latombe, los cambios en la tecnología, así como nuevas aplicaciones, aumentarán nuevos problemas a investigar. Por ejemplo, los módulos de robots reconfigurables (ver Figura 1.2, tomadas de Biorobotics Laboratory BioRob) debido a sus prestaciones están siendo desarrollados con mayor interés a partir del año 2000. Éstos necesitarán nuevos tipos de planeadores para calcular sus movimientos a reconfigurar.

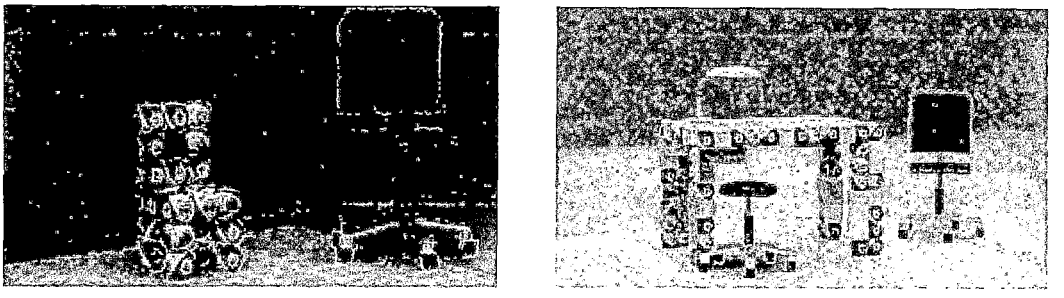

Figura 1.2: Robots reconfigurables.

1.2. Motivación de la investigación

Varios tipos de máquinas o dispositivos requieren interacción con su entorno, si bien es cierto con los diversos tipos de sensores se puede obtener data del entorno pero esto no es suficiente, ya que se requiere de un algoritmo que interprete dicha data para ejecutar ejecutar acciones pertinentes. El movimiento es una forma de interacción con el medio, ya que en los diversos entornos se presentan obstáculos, restricciones propias del sistema que dificultan en cierto modo el movimiento deseado. El campo de estudio es denominado *Motion Planning*, y lo que se va a enfocar en esta tesis es una parte de ese campo, lo referente a la planificación de rutas o caminos en robots móviles.

Lo que motivo la investigación en el presente tema es que las aplicaciones derivadas de su estudio son amplias como por ejemplo en el área de robótica, de manufactura, medicina y animaciones. La contribución al desarrollar este tipo de algoritmos es que se tendría una herramienta de planificación que permita interactuar con nuestro entorno.

1.3. Objetivos

- Desarrollar y validar un algoritmo de planificación de caminos de tipo estocástico para robots móviles que les permita navegar en sus respectivos entornos.
- Desarrollar y validar un algoritmo que tome en cuenta el modelamiento del sistema.
- Implementar una plataforma e interfaz de pruebas para robots móviles que permitan validar el resultado de algoritmos de planificación de caminos.

1.4. Alcances

El desarrollar algoritmos de *Path Planning* permite poder realizar aplicaciones futuras no sólo enfocadas a robótica móvil o manipuladores sino también a nuevos campos emergentes tales como:

- Diseño para manufactura
- Animación gráfica
- Software de videojuegos
- Cirugía robótica
- Biología computacional

1.5. Limitaciones

El algoritmo estocástico de *Path Planning* a desarrollar no puede ser aplicado en ambientes o entornos muy dinámicos ya que el costo computacional no lo permite, esto se verá a detalle con las simulaciones a realizar. La presente tesis desarrolla una implementación a pequeña escala con un robot móvil, lo cual servirá para validar el algoritmo en un entorno real.

CAPÍTULO 2

PLANIFICACIÓN DE TRAYECTORIA

Estrictamente hablando en la literatura de *Motion Planning* existe diferencia entre *Path* (camino) o *Trayectory* (trayectoria), debido a que una trayectoria es un camino o ruta parametrizada por el tiempo. Sin embargo en la traducción al español no.

La planificación de caminos o *Path Planning* surge en diversos campos como la robótica, el análisis de emsambles, prototipos virtuales, el diseño farmaceútico de drogas, la manufactura y la animación computacional. Dichos problemas generalmente involucran el cálculo de una secuencia (un camino o *Path*) de configuraciones (coordenadas generalizadas) entre una configuración a otra, mientras se respetan ciertas restricciones.

Es decir, *Path Planning* se refiere al cálculo de un camino a seguir para llevar a un sistema desde una configuración inicial a una final o deseada, en el presente caso, dicho sistema es un robot móvil que debe desplazarse por una trayectoria evitando los obstáculos de su entorno hasta llegar a la posición deseada. Por ende un planificador de trayectoria en una primera instancia reconoce o bosqueja una aproximación del entorno, esto debido a que en algunos sistemas el entorno es conocido a priori; en otros casos el robot va conociendo su entorno de a pocos utilizando sensores.

2.1. Definiciones y notaciones

Para tener un mayor entendimiento de los términos utilizados en lo referido a la literatura de la planificación de movimientos de los robots, es necesario desarrollar algunos conceptos así como también definir la nomenclatura a utilizar. Dichos conceptos han sido tomados de [4], [5] y [6]:

2.1.1. Robot

Esté término tiene muchas acepciones, se le puede definir como un dispositivo mecánico versátil equipado con actuadores y sensores bajo el control de un sistema. Russel y Norvig lo definen como un "activo y artificial agente cuyo ambiente es en mundo real". El robot será denotado con A para las formulaciones a seguir.

2.1.2. Entorno

Es el espacio de trabajo W (el cual es de 2 ó 3 dimensiones, $W = \mathbb{R}^2$ ó $W = \mathbb{R}^3$), el cual comprende tanto a los obstáculos como espacios por donde el robot puede desplazarse, es el dato principal sobre el cual se realizarán los algoritmos que calculen la ruta a seguir por el móvil para alcanzar la posición y orientación deseada, ya que en una primera instancia se reconoce o bosqueja una aproximación del entorno. En cuanto a la información referida al entorno se podría clasificar en:

- Entorno conocido, tanto el entorno como la configuración y orientación inicial del robot son conocidas.
- Entorno desconocido, es decir el robot no conoce su entorno a priori como el caso anterior, pero provee sensores que le permiten bosquejar un entorno progresivamente para posteriormente encontrar la ruta deseada.

2.1.3. Configuración de un objeto

Es una especificación de la posición de cada punto de un objeto, relativo a un fijo al sistema o marco de referencia. Para especificar la configuración de un objeto rígido A, es suficiente con especificar la posición y orientación del sistema de referencia F_A con respecto al sistema de referencia F_W (ver Figura 2.1). El subconjunto ocupado por A en la configuración q es denotado por $A(q)$, es decir, $A(q) \subset W$.

Si el robot es una cadena cinemática, q es típicamente un vector que contiene los ángulos de las articulaciones. Si el robot es un cuerpo rígido de desplazamiento libre, q consiste de una parte de traslación y otra de rotación.

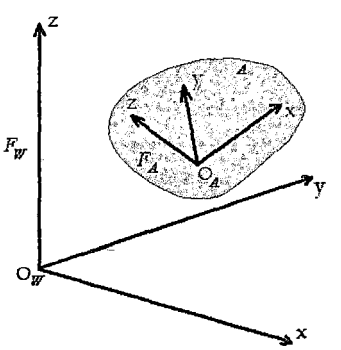

Figura 2.1: Sistemas de referencia

2.1.4. Región de obstáculos

Denotada por O, es definida como el conjunto de todos los puntos en W que contienen uno a más obstáculos. Si los obstáculos son representados por subconjuntos cerrados de W, la región de obstáculos es también un subconjunto cerrado, $O \subset W$.

2.1.5. Espacio de configuraciones de un cuerpo rígido

Es el espacio de todas las posibles configuraciones del robot, denotadas por C. El concepto del espacio de configuraciones es muy importante en *path planning*.

Para un brazo robótico con n juntas, donde cada junta tiene un rango limitado, el espacio de configuraciones es $\mathbb{R}^n$. Otros problemas pueden tener un espacio de configuraciones con una topología totalmente diferente, como en el caso de un mundo de 2 dimensiones en el cual un cuerpo rígido que puede trasladarse y rotar.

2.1.6. Configuración del espacio de obstáculos

El conjunto de todas las posibles configuraciones del robot que hagan que el robot intercepte con un obstáculo, es denotado por C_{obs}. Este conjunto es definido como:

$$C_{obs} = \left\{ q \in C \mid A(q) \bigcap O \neq \varnothing \right\} \tag{2.1}$$

La ecuación (2.1) puede ser vista como el mapeo de los obstáculos de la zona de trabajo (W) al espacio de configuraciones (C). Las regiones resultantes (ver Figura 2.2) son usualmente denotadas como el espacio de configuraciones de los obstáculos (C_{obs}). Para el caso de robots que constan de varios objetos, la definición de C_{obs} en la ecuación (2.1) tiene que ser extendida en incluir las auto-colisiones. La parte restante del espacio de configuraciones se llama espacio libre. Dicho espacio es definido y denotado como C_{free}: $C_{free} = C \setminus C_{obs}$

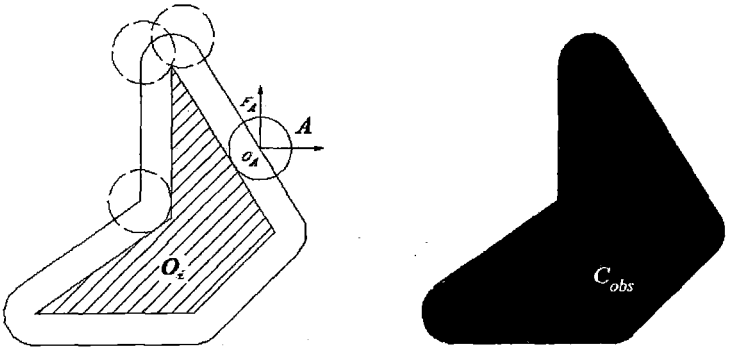

Figura 2.2: Mapeo de W a C

Desde el trabajo seminal de Tomás Lozano-Pérez[1], la mayoría de algoritmos de *path planning* trabajan en el espacio de configuraciones. Transformando el problema del espacio W al espacio C usando la ecuación (2.1) que es una útil abstracción. Los algoritmos de *path planning* que lucen muy diferentes en W mantienen un aspecto similar en el espacio C, permitiendo que el mismo tipo de algoritmo pueda resolver problemas muy diferentes. Sin embargo, la transformación en la ecuación (2.1) se lleva a cabo de forma explícita sólo para problemas muy sencillos de baja dimensión. La mayoría de los algoritmos trabajan probando C para descubrir si una configuración pertenece a C_{free}.

2.1.7. Métrica

La métrica no juega un papel o rol en la definición del problema básico de *motion planning.* Sin embargo algunos métodos de planificación usan la distancia a los obstáculos y la configuración final como las funciones heurísticas con el objetivo de direccionar el proceso de búsqueda.

La métrica es el valor real de una función ρ definida en un espacio topológico X tal que para cada x,y,z ∈X. La métrica debe cumplir con los siguientes requisitos:

- Ser positiva (ecuación 2.2)

$$\rho(x,y) \geq 0 \tag{2.2}$$

- Simétrica (ecuación 2.3)

$$\rho(x,y) = \rho(y,x) \tag{2.3}$$

- Ser cero si y solamente si los 2 puntos son iguales (ecuación 2.4)

[1]Catedrático en la Escuela de Ingeniería en el MIT (Instituto Tecnológico de Masachuset), donde es miembro del Laboratorio de Ciencias de la Computación e Inteligencia Artificial. Ha realizado investigaciones en el campo de robótica (enfoque de la configuración de espacio en Motion Planning), visión computacional, machine learning (aprendizaje de máquina), medical imaging y química computacional.

$$\rho(x, y) = 0 \Longleftrightarrow x = y \tag{2.4}$$

- La desigualdad del triángulo (ecuación 2.5)

$$\rho(x, y) + \rho(y, z) \geq \rho(x, z) \tag{2.5}$$

Entre las métricas mas utilizadas se tienen:

- La distancia euclidiana (ecuación 2.6)

$$\rho(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.6}$$

- Métrica de Manhattan (ecuación 2.7)

$$\rho(x, y) = \sum_{i=1}^{n} \mid x_i - y_i \mid \tag{2.7}$$

2.1.8. El problema básico

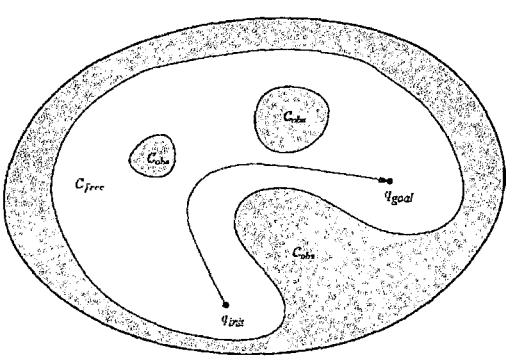

Figura 2.3: El problema básico de *motion planning*.

Jean-Claude Latombe explica que la meta de definir un problema básico de *motion planning* es de aislar algunas cuestiones centrales e investigarlas en profundidad antes de considerar dificultades adicionales. En el problema básico se asume que el robot es el único objeto que se mueve en el espacio de trabajo y se ignoran las propiedades dinámicas del robot, evitando así cuestiones de temporización. Además se restringen los movimientos a movimientos de no contacto, así

las cuestiones relacionadas a las interacciones mecánicas entre 2 objetos físicos en contacto pueden ser ignorados. Estas suposiciones esencialmente transforman el problema de *motion planning* a un problema púramente geométrico. Además se asume incluso las cuestiones geométricas asumiendo que el robot es un objeto rígido, es decir, los movimientos del objeto están restringidos únicamente por los obstáculos. El problema básico de *motion planning* (ver Figura 2.3)resultante de las simplificaciones es:

Sea A un simple cuerpo rígido – el robot- moviéndose en un espacio euclidiano W, llamado espacio de trabajo, representado como $\mathbb{R}^n$, con n=2 ó 3. Sea $O_1, O_2, \ldots .., Oq$ objetos rígidos fijos distribuidos en W, denominándose a O_i's obstáculos. Asumir para ambas geometrías A y O_i's y que la localización de O_i's son conocidas en W. Asumir además que las restricciones cinemáticas limitan los movimientos de A. El problema es: Dada una posición y orientación inicial (q_{init}) y una posición y orientación final(q_{goal}) de A en W, generar un camino τ especificando una secuencia continua de posiciones y orientaciones de A que eviten entrar en contacto con O_i's, iniciando en las condiciones iniciales dadas (posición y orientación) y alcanzando las condiciones finales requeridas. Reportar como falla si tal camino no existe. Jean-Claude Latombe explica que si bien el problema básico ha sido sobresimplificado, no obstante es un problema difícil y que varias soluciones han sido extendidas a otros problemas.

Este particular tipo de problema es usualmente referido al *"Problema de mover el piano"* en la literatura de *motion planning.* En la formulación de dicho problema no hay concepto del tiempo, es suficiente encontrar la secuencia de traslaciones y rotaciones que permitan al robot ir de la configuración inicial a la final, y las velocidades y aceleraciones no son de interés. Si se desease resolver el problema y al mismo tiempo respetar o tomar en cuenta las restricciones cinemáticas y/o dinámicas del sistema en particular, se tiene un ejemplo de lo que se denomina "*Kinodynamic Motion Planning Problem*".

Con respecto a los términos utilizados en la literatura: *motion planning* y *path planning*, James Kuffner[2] nos explica que el término *motion planning* es usualmente distinguido de *path planning*, en que el camino computado es parametrizado por el tiempo. La consideración de tiempo o sistemas dinámicos es usualmente importante para problemas que requieran parametrización de tiempo en la solución de sus trayectorias.

2.1.9. Tiempo

Todos los problemas de planificación implican una secuencia de decisiones que se deben aplicar en el tiempo. El tiempo puede ser modelado explícitamente, como en un problema, tales como conducir un coche tan rápido como sea posible a través de una carrera de obstáculos. Por otra parte, el tiempo puede ser implícito, simplemente refleja el hecho de que las acciones deben seguir en la sucesión, como en el caso de resolver el cubo de Rubik (ver Figura 2.4, tomada de [5]).

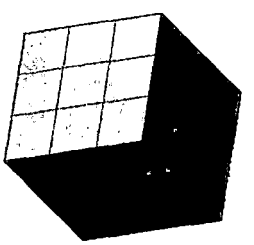

Figura 2.4: Cubo Rubik.

El tiempo en particular no es importante, pero la secuencia adecuada debe mantenerse. Otro ejemplo de tiempo implícito es una solución al problema del Mover el Piano (ver Figura 2.5, tomada de [5]), la solución a mover el piano se puede convertir en una animación con el tiempo, pero la velocidad en particular no se especifica en el plan. Al igual que en el caso de los espacios del estado, el tiempo puede ser discreto o continuo.

[2]Profesor asociado del Instituto de Robótica de la Escuela de Ciencias de la Computación de la Universidad de Carnegie Mellon

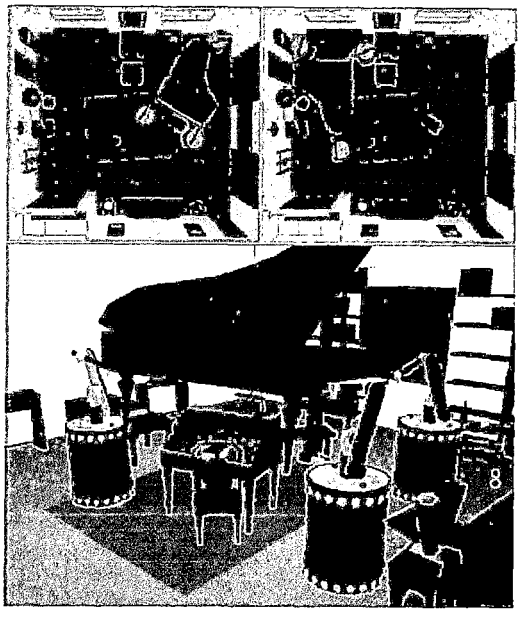

Figura 2.5: Moviendo piano usando robots móviles.

2.1.10. Modelo

Un modelo es la representación de un sistema y es usado con la finalidad de responder preguntas mediante análisis y simulación. El modelo que se elige depende del tipo de preguntas que se desee responder, y como tal, existe más de un tipo de modelo para un mismo sistema, con diferentes niveles de fidelidad dependiendo del fenómeno de interés.

2.1.11. Estado

El estado de un sistema es la colección de variables que caracteriza completamente el movimiento de un sistema con el propósito de predecir el movimiento futuro. Al conjunto de todos los estados posibles se le denomina espacio de estados. Un estado puede representar la posición y orientación del robot. Un tema recurrente es que un estado es generalmente representado implícitamente por un algoritmo de planificación.

2.1.12. Criterio

Esto codifica el resultado deseado de un planeador en términos de su espacio estado y acciones ejecutadas. Hay generalmente 2 tipos diferentes de planeamientos basados en el tipo de criterio:

1. **Viabilidad**.- Encontrar un plan que permita llegar al estado deseado, independientemente de su eficiencia.

2. **Optimalidad**.- Encontrar un plan viable que optimice el desempeño en algunos, además de llegar a un estado deseado.

Para la mayoría de problemas la viabilidad es ya suficientemente difícil, lograr la optimización es considerablemente difícil para la mayoría de situaciones. Por lo tanto, gran parte de la atención se centra en la búsqueda de soluciones viables a los problemas, en lugar de soluciones óptimas. La mayoría de la literatura en robótica, teoría de control, y áreas relacionadas se centra en la optimización, pero esto no es necesariamente importante en muchos problemas de interés. En muchas aplicaciones, es difícil formular, incluso un criterio adecuado de optimización. Incluso si un criterio deseable puede formularse, puede ser imposible obtener un algoritmo práctico que compute óptimos planeadores. En tales casos, las soluciones viables son sin duda preferibles a no tener soluciones. Afortunadamente, para muchos algoritmos las soluciones producidas no están tan lejos de las óptimas en la práctica. Esto reduce parte de la motivación para encontrar soluciones óptimas. Para los problemas que involucran incertidumbre probabilística, sin embargo, la optimización surge con mayor frecuencia. Las probabilidades son a menudo utilizadas para obtener el mejor rendimiento en términos de los costos previstos. Viabilidad se asocia a menudo con la realización (el desempeño) de un análisis del peor caso de incertidumbres.

Este criterio es tomado no solo en cuenta en lo que concierne, por ejemplo lo que denomina David Lammers en la "Era del error-Tolerancia computacional",[6],

en el cual explica que la era perfeccionista del ordenador esta llegando a su fin. En el Simposio Internacional de Electrónica de Baja Potencia y Diseño, según los expertos se refiere a consumo de energía están impulsando la informática hacia una filosofía de diseño en el que los errores están permitidos y son ignorados, o se corregidos sólo cuando sea necesario. Los resultados probabilísticos sustituirán a la forma determinista de procesamiento de datos que ha prevalecido durante el último medio siglo.

Naresh Shanbhag, profesor en el departamento de Ingeniería Eléctrica y Computación de la Universidad de Illions, dice: "Si la aplicación es tal que pequeños errores pueden ser tolerados, nosotros permitiríamos que sucedan", agrega "Dependiendo de la aplicación, nosotros mantenemos frecuencias de error bajo un umbral, usando algoritmos o técnicas de circuito". Para muchas aplicaciones tales como procesamiento gráfico o inferencias gráficas desde una gran cantidad de data, los errores en un número razonable no impactan terriblemente en la calidad del resultado. Después de todo, nuestros ojos no pueden notar la presencia de un pixel errado dentro de la mayoría de imágenes.

Otro ejemplo puede darse en los sistemas de control, en los cuales ya hay incertidumbre a las señales a evaluar y buscar una precisión total para nuestra salida deseada, la cual de por si también presenta incertidumbre.

En cuanto a resultados aplicar el criterio expuesto ha logrado en los procesadores un menor consumo de energía; en el procesamiento gráfico y sistemas de control, mayor rapidez.

2.1.13. Plan

En general, un plan establece una estrategia específica o el comportamiento de un tomador de decisiones. Un plan sólo puede especificar una secuencia de acciones a tomar, sin embargo, podría ser más complicado. Si no es posible predecir los estados futuros, a continuación, el plan puede especificar las acciones en fun-

ción del estado. En este caso, con independencia de los estados futuros, la acción correspondiente se determinará. Utilizando la terminología de otros campos, lo que permite comentarios o planes de reactivos. Incluso podría darse el caso de que el Estado no puede ser medido. En este caso, la acción adecuada debe determinarse a partir de toda la información disponible hasta el momento actual. Esto generalmente se conoce como un estado de información, en la que las acciones de un plan están condicionadas.

2.2. Tipos de algoritmos de Path Planning

Principalmente desde los años 70 hasta ahora se han desarrollado una gran variedad de algoritmos de *path planning* que acorde a su naturaleza pueden clasificarse en:

- *Cell decomposition* (descomposición de celdas)
- *Potential Fields* (campos potenciales)
- *Roadmaps* (mapa de caminos)

2.2.1. Cell decomposition

Hasta hace poco, el método más común en *path planning* fue basado en la construcción de una descomposición de celdas (la más común es la trapezoidal,[7], ver Figuras 2.6 y 2.7, tomadas de [7].) del espacio libre del robot. Una celda es una región del espacio libre de forma sencilla de tal manera que un camino puede ser construido fácilmente entre 2 configuraciones dentro de la celda. El espacio libre es una colección de celdas, el *path planning* puede ser reducido a una búsqueda de la representación gráfica la relación adyacente entre celdas. *Cell decomposition* puede ser exacta o aproximada, la exacta tiene a ser compleja para su implementación e ineficiente por tal razón la aproximada tiene mayor acogida,

aunque hay parámetros que tienen que ser regulados como cuando el robot pasa a través de regiones de libre configuración cuyo tamaño es menor que la resolución de la celda, así nos explica Sean Quinlan[8].

Sin embargo cuando la dimensión del espacio libre es alta o cuando la complejidad de la escena es mayor, la representación de las celdas tiende a crecer exponencialmente y el método deja de ser práctico.

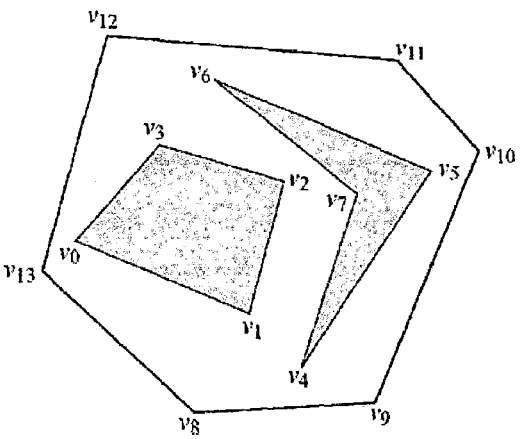

Figura 2.6: Muestreo poligonal de la configuración de espacio.

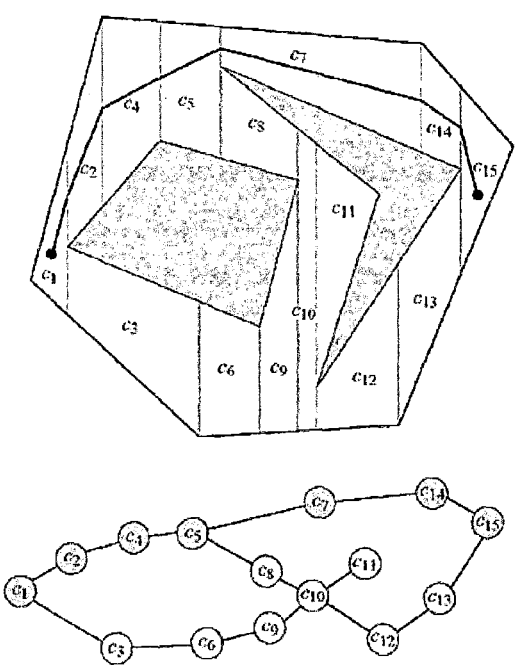

Figura 2.7: Calculo de la ruta.

2.2.2. Potential fields

El método de los campos potenciales artificiales tiene como principio atraer al robot a la posición deseada y de repelerlo de los obstáculos del entorno de navegación (ver Figuras 2.8 y 2.9, tomadas de [9]).

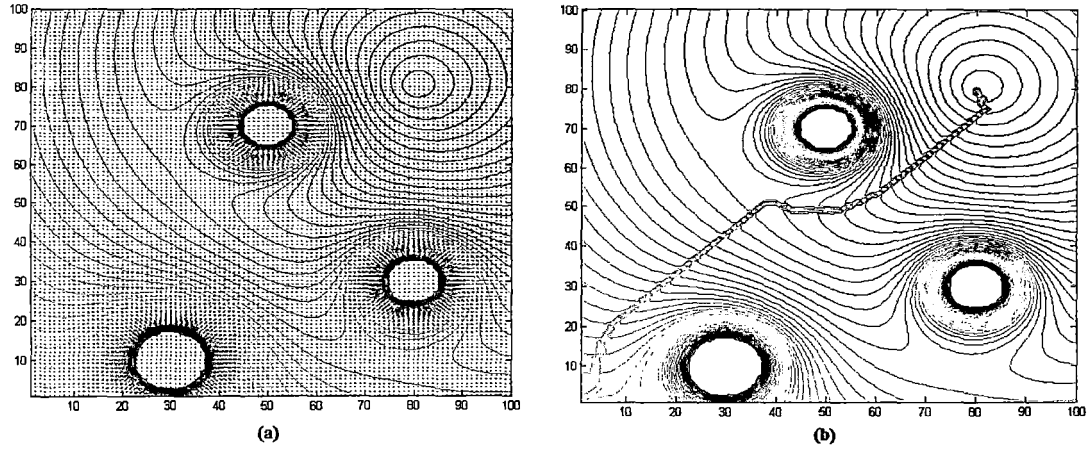

Figura 2.8: (a)Aplicación de campos potenciales en *path planning*. (b) Cálculo de la ruta en base al criterio del gradiente.

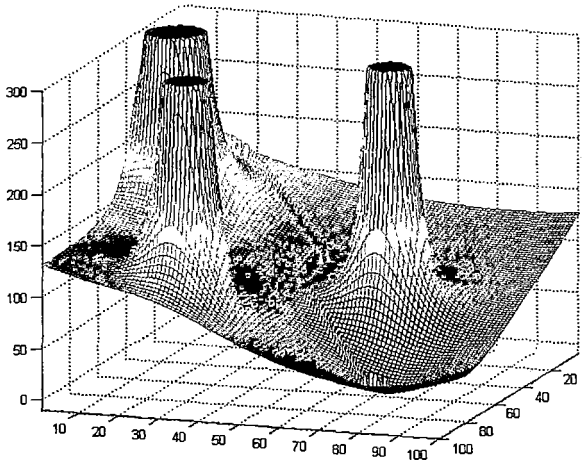

Figura 2.9: Vista del planeador.

La gran ventaja de este método es que puede ser implementado para operar en tiempo real y puede ser incorporado directamente en sistema de control del robot. Para muchas situaciones triviales, este método es muy satisfactorio pero la mayor dificultad se presenta cuando las funciones candidatas afrontan los llamados mínimos locales (ver Figura 3.1, tomada de [7]) que atascan al robot, de los cuales no es tan fácil escapar, así nos explica Sean Quinlan,[8].

Con respecto a este punto se han desarrollado "Local minima-free potential functions" (también llamadas Funciones de Navegación), sin embargo, los costos de dichas funciones son altos en configuraciones espaciales de grandes dimensiones y no han obtenido grandes resultados en robots de muchos grados de libertad. Otro alternativa propuesta era almacenar las probabilidades de las vecindades de las configuraciones sin caer en mínimos locales, con ello se obtuvieron buenos resultados hasta robots de 6 grados de libertad, pero conforme aumentaban los grados de libertad esta técnica se volvía impráctica como lo explica Latombe,[10].

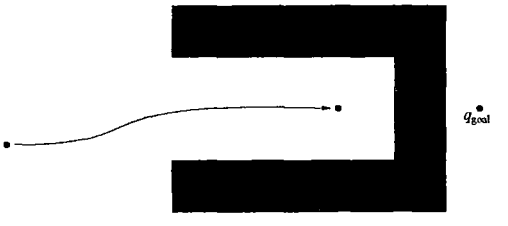

Figura 2.10: Ejemplo de mínimo local.

2.2.3. Roadmaps

Se enfoca el espacio libre del robot como la colección de caminos libres de colisión conectados a lo largo de dicho espacio. Con dicho conjunto, se construye la ruta desde el punto inicial al punto deseado que está en alguna parte del roadmap. Para un ambiente estático el roadmap es construido una vez para resolver múltiples problemas de planeamiento. Las muchas variaciones de roadmap se dan principalmente por el enfoque diverso en el método de construcción de dicho roadmap, estas variaciones incluyen [8]:

- Visibilidad de grafos
- Diagramas de Voronoi
- Roadmaps aleatorios

Los 3 primeros funcionan para robots de 2-3 grados de libertad pero no resultan muy adecuados para mayor grados de libertad tal como lo explica Latombe[10], que propone el uso de técnicas aleatorias, siendo los procesos estocásticos experimentalmente los que han tenido buenos resultados en robots complejos de muchos grados de libertad.

Se construye un grafo que representa la configuración libre de obstáculos del entorno; con el grafo ya definido se procede a encontrar la ruta que una la configuración inicial con la deseada, la cual debe ser modificada para cumplir con las restricciones no holonómicas del robot y la naturaleza del ambiente (dinámica o estática). La desventaja que posee es un alto costo computacional con respecto a otros métodos.

2.3. Algoritmo de path planning RRT

En la presente tesis se desarrollará el algoritmo RRT (*Rapidly Random Exploring Trees*[11]), que es pertenece al grupo de Roadmaps ya que explora el entorno del robot para encontrar la ruta deseada. A su vez es un algoritmo de tipo estocástico.

2.4. Metodología de trabajo

Básicamente consta de 3 partes (que son explicadas a detalle en el Capítulo 5):

1. Reconocimiento de Entorno, que será a priori para el robot, es decir se conocerán los obstáculos y configuraciones libres, así como la configuración inicial del robot.

2. Ejecutar el planeador de caminos, cuyos datos son el entorno, la configuración inicial del robot y la configuración deseada.

3. Supervisar que el robot siga la ruta de referencia mediante una interfaz que calcula la posición y orientación del robot mediante una cámara.

CAPÍTULO 3

ALGORITMO RRT

El algoritmo RRT es una estructura de datos que está diseñado para explorar de manera eficiente espacios no convexos de grandes dimensiones (ver Figura 3.1, tomada de [7]). El algoritmo se construye de forma incremental, reduciendo rápidamente la distancia de espera de un punto elegido al azar en el árbol. RRT es especialmente adecuado para problemas de planificación de ruta de acceso que implican los obstáculos y restricciones diferenciales. RRT puede ser considerado como una técnica para generar las trayectorias a lazo abierto para sistemas no lineales con restricciones de estado.

Hay que notar que de por sí solo, RRT es incapaz de resolver un problema de path planning. En consecuencia, puede considerarse como un componente ha ser incorporado en el desarrollo de una variedad de algoritmos de planificación diferentes.

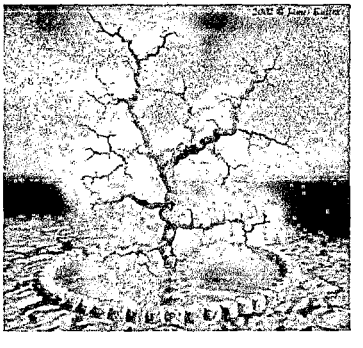

Figura 3.1: Naturaleza del algoritmo RRT.

3.1. Elementos

El algoritmo RRT original se basa en la construcción de un árbol de configuraciones que crece buscando a partir de un punto origen. Para entender el algoritmo se usará la nomenclatura expresada en el cuadro 3.1:

Elemento	Descripción
C	es el conjunto de todas las configuraciones posibles del entorno, es decir, los obstáculos y las configuraciones libres.
C_{free}	Subconjunto de C, configuraciones libres de obstáculos existentes.
R	Indicador de ponderación de proximidad al punto deseado. La distancia euclidiana es la más utilizada.
q_{init}	Configuración inicial
q_{fin}	Configuración que se desea alcanzar
q_{rand}	Configuración aleatoria dentro del espacio C_{free}
q_{near}	Es la conguración mas proxima a q_{rand} , en el sentido denido por R, de entre las existentes en un árbol. Se evalúa con el indicador de ponderación.
q_{new}	Configuración a añadir al árbol.
e	Longitud de segmento de crecimiento. Geométricamente, es la distancia entre un punto del árbol y el siguiente con el que esta conectado.
$Arbol$	Estructura de datos.

Cuadro 3.1: Elementos de una RRT

3.2. Pseudocódigo

El algoritmo viene expresado por (ver apéndice D.1.1 y D.1.2):

Algoritmo RRT

Función: RRT(q_{ini}, K_{max}, e)

1. $Arbol[0] = q_{init}$;
2. for $K = 1$ to K_{max}
3. q_{rand} = Configuración_Aleatoria();
4. q_{near}= Elemento_más_cercano ($Arbol$, q_{rand})
5. q_{new}= Nuevo_elemento (q_{rand}, q_{near}, e)
6. end for
7. Devuelve $Arbol$;

Función: Nuevo_elemento(q_{rand}, q_{near}, e)

1. $u_{qnear-qrand} = (q_{rand} - q_{near}) \,/\, \|q_{rand} - q_{near}\|$;
2. $q_{new} = q_{near} + e.u_{qnear-qrand}$;
3. Devuelve q_{new};

El pseudocódigo de RRT es explicado como sigue:

1. Se inicializa la estructura de datos llamada *Arbol* con la configuración inicial q_{init} (ver Figura 3.2)

2. Se inicia un búcle con un máximo número de iteraciones K_{max}

3. Se asigna a q_{rand} una configuración aleatoria del espacio C_{free} (ver Figura 3.3)

4. Se asigna a q_{near} el elemento más cercano de *Arbol* a q_{rand} (ver Figura 3.4)

5. Se asigna a q_{new} el nuevo elemento de *Arbol*, este elemento se calcula como la suma vectorial del producto del vector unitario (desde q_{near} a q_{rand}) con la longitud de crecimiento definida (e) y q_{near} (ver Figura 3.5).

6. Se verifica la condición del bucle.

7. Devolución de la estructura *Arbol*

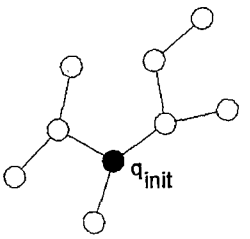

Figura 3.2: Árbol de exploración.

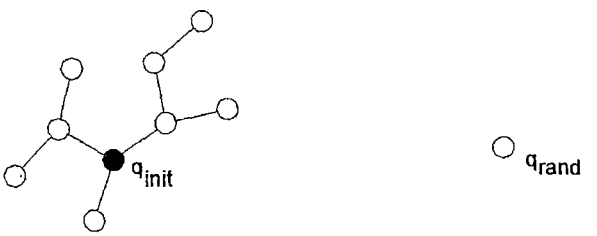

Figura 3.3: Seleccionando un punto aleatorio.

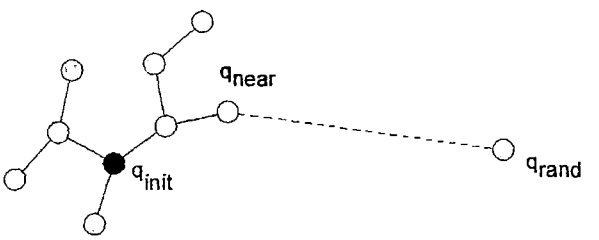

Figura 3.4: Calculando el punto el elemento más cercano del árbol al punto aleatorio.

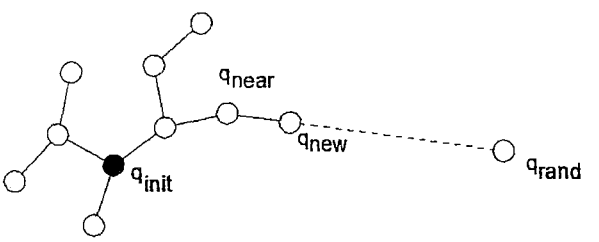

Figura 3.5: Adicionando nuevo elemento al árbol explorador.

3.3. Algoritmos evolucionados de RRT

Estos algoritmos permiten que el RRT pueda encontrar la ruta entre 2 configuraciones. Entre las principales destacan:

3.3.1. RRT Bidireccional Básica

Consiste en conectar los árboles exploradores que surgen desde el punto inicial y final. El algoritmo esta dado por (ver apéndice D.1.3):

Algoritmo RRT Bidireccional Básica

```
Función: RRT_BB(q_ini, q_goal, K_max, e)
1. Arbol_A[0] = q_init; Arbol_B[0] = q_goal; K = 1;
2. while (K < K_max|| state='Alcanzado')
3.  q_rand = Configuración_Aleatoria();
4.  if ( Extiende(Arbol_A, q_rand)≠'rechazado')
5.  a=Extiende(Arbol_A, q_rand); b=Extiende(Arbol_B, q_rand);
6.     if (a=='alcanzado' y b=='alcanzado')
7.       Devuelve Ruta;
8.     end if;
9.  end if;
10. Intercambiar (Arbol_A, Arbol_B);
11. K = K + 1;
12. end while;

Función: Extiende(Arbol, q_rand)
1. q_near= Vecino_mas_proximo(Arbol, q_rand);
2. q_new= Nuevo_elemento(q_rand, q_near);
3. if( q_new ϵ Configuracion_libre)
4.  Arbol=[Arbol; q_new] //se añade nuevo vértice
5.   if ( abs(q_new=q_rand)< tolerancia) Devuelve 'alcanzado';
6.   else Devuelve 'avanzado';
7.   endif
8. else Devuelve 'rechazado';
9. endif

Función: Nuevo_elemento(q_rand,q_near, e)
1. u_qnear-qrand = (q_rand − q_near) / ||q_rand − q_near||;
2. q_new = q_near + e.u_qnear-qrand;
3. Devuelve q_new;
```

El pseudocódigo es explicado como sigue:

1. Se inicializan los árboles exploradores con las configuraciones inicial (q_{init}) y final (q_{goal}), es decir con 2 estructuras de datos tipo *Arbol* (ver Figura 3.2)
2. Se inicia un búcle
3. Se asigna a q_{rand} una configuración aleatoria del espacio C_{free} (ver Figura 3.3)
4. Si el nuevo elemento del $Arbol\_A$ con respecto a q_{rand} está dentro de configuración C_{free} se procede al paso 5, caso contrario ir al paso 10.
5. Se calculan las nuevas configuraciones de los árboles $Arbol\_A$ y $Arbol\_B$ con respecto a q_{rand} si es que las hay mediante la función Extiende.
6. Si en ambos árboles se obtiene ´alcanzado´ ir al paso 7, caso contrario ir al paso 9.
7. Si las nuevas configuraciones de los 2 árboles alcanzan al punto q_{rand}, se procede a calcular y devolver la ruta contenida en la estructura *Arbol*, así como salir del bucle.
8. Se verifica la condición del bucle, si se termina se va al paso 7, caso contrario ir al paso 3.
9. Se intercambian los árboles.
10. Se aumenta la cuenta de la variable K.
11. Se verifica la condición del bucle.

3.3.2. RRT Ext-Ext

Es una mejora del algoritmo RRT Bidireccinal básico, el algoritmo viene expresado como (ver apéndice D.1.4):

Algoritmo RRT Ext-Ext

```
Función: RRT_Ext_Ext(q_ini, q_goal, K_max, e)
1. Arbol_A[0] = q_init; Arbol_B[0] = q_goal; K = 1;
2. while (K < K_max|| state='Alcanzado')
3.   q_rand = Configuración_Aleatoria();
4.   if ( Extiende(Arbol_A, q_rand)≠'rechazado')
5.    a=Extiende(Arbol_A, q_rand);
6.    q_new =last_element(Arbol_A);
7.    b=Extiende(Arbol_B, q_new);
8.     if (a=='alcanzado' y b=='alcanzado')
9.       Devuelve Ruta;
10.    end if;
11.  end if;
10. Intercambiar (Arbol_A, Arbol_B);
11. K = K + 1;
12. end while;

Función: Extiende(Arbol, q_rand)
1. q_near= Vecino_mas_proximo(Arbol, q_rand);
2. q_new= Nuevo_elemento(q_rand, q_near);
3. if( q_new ϵ Configuracion_libre)
4.  Arbol=[Arbol; q_new] //se añade nuevo vértice
5.   if ( abs(q_new=q_rand)< tolerancia) Devuelve 'alcanzado';
6.   else Devuelve 'avanzado';
7.   endif
8. else Devuelve 'rechazado';
9. endif

Función: Nuevo_elemento(q_rand,q_near, e)
1. u_qnear-qrand = (q_rand − q_near) / ||q_rand − q_near||;
2. q_new = q_near + e.u_qnear-qrand;
3. Devuelve q_new;
```

El pseudocódigo es explicado como sigue:

1. Se inicializan los árboles exploradores con las configuraciones inicial (q_{init}) y final (q_{goal}), es decir con 2 estructuras de datos tipo *Arbol* (ver Figura 3.6)

2. Se tiene búcle con un máximo número de iteraciones K_{max}.

3. Se asigna a q_{rand} una configuración aleatoria del espacio C_{free} (ver Figura 3.3).

4. Se calcula el nuevo elemento del $Arbol\_A$, si este elemento (q_{new}) no pertenece a la región de los obstáculos, se procede al paso 5, caso contrario ir al paso 2.

5. Se calcula el nuevo elemento del $Arbol\_B$, teniendo como objetivo el q_{new}de $Arbol\_A$ del paso 4.

6. Se verifica la condición de **'alcanzado'**para ambos árboles, si se cumple se devuelve la ruta y se finaliza el bucle, caso contrario se procede al paso 7.

7. Se intercambian los árboles.

8. Se aumenta la cuenta de la variable K.

9. Se va al paso 2.

3.3.3. RRT Ext-Con

Tiene el siguiente principio (ver apéndice D.1.5):

Algoritmo RRT Ext-Con

```
Función: RRT_Ext_Con(q_ini, q_goal, K_max, e)
1. Arbol_A[0] = q_init; Arbol_B[0] = q_goal; K = 1;
2. for K = 1 to K_max
3.    q_rand = Configuración_Aleatoria();
4.    if ( Extiende(Arbol_A, q_rand)≠'rechazado')
5.     a=Extiende(Arbol_A, q_rand);
6.     q_new =last_element(Arbol_A);
7.     b=Conecta(Arbol_B, q_new);
8.      if (a=='alcanzado' y b=='alcanzado')
9.        Devuelve Ruta;
10.     end if;
11.   end if;
10. Intercambiar (Arbol_A, Arbol_B);
11. K = K + 1;
12. endfor;

Función: Extiende(Arbol, q_rand)
1. q_near= Vecino_mas_proximo(Arbol, q_rand);
2. q_new= Nuevo_elemento(q_rand, q_near);
3. if( q_new ϵ Configuracion_libre)
4.  Arbol=[Arbol; q_new] //se añade nuevo vértice
5.   if ( abs(q_new=q_rand)< tolerancia) Devuelve 'alcanzado';
6.   else Devuelve 'avanzado';
7.   endif
8. else Devuelve 'rechazado';
9. endif

Función: Conecta(Arbol, q)
1. do {S=Extiende(Arbol, q); }
2. while (S='avanzado');
3. Devuelve S;

Función: Nuevo_elemento(q_rand,q_near, e)
1. u_qnear-qrand = (q_rand − q_near) / ||q_rand − q_near||;
2. q_new = q_near + e.u_qnear-qrand;
3. Devuelve q_new;
```

El pseudocódigo es explicado como sigue:

1. Se inicializan los árboles exploradores con las configuraciones inicial (q_{init}) y final (q_{goal}), es decir con 2 estructuras de datos tipo *Arbol* (ver Figura 3.6)

2. Se tiene un búcle con un máximo número de iteraciones K_{max}.

3. Se asigna a q_{rand} una configuración aleatoria del espacio C_{free} (ver Figuras 3.7 y 3.8).

4. Se calcula el nuevo elemento del $Arbol\_A$, si este elemento (q_{new}) no pertenece a la región de los obstáculos, se procede al paso 5, caso contrario al paso 2 (ver Figura 3.9 y 3.10).

5. Se calcula el nuevo elemento o nuevos elementos del $Arbol\_B$ (ver función **Conecta** y Figuras 3.10, 3.11 y 3.12), teniendo como objetivo el q_{new}de $Arbol\_A$ del paso 4. Se verifica la condición de **'alcanzado'**, si se cumple se devuelve la ruta (ver Figura 3.13) y se sale del bucle, caso contrario se procede al paso 6.

6. Se intercambian los árboles.

7. Se aumenta la cuenta de la variable K.

8. Se va al paso 2.

Los pasos explicados anteriormente explicados pueden entenderse gráficamente mediante las siguiente figuras:

Figura 3.6: Conección de árboles exploradores.

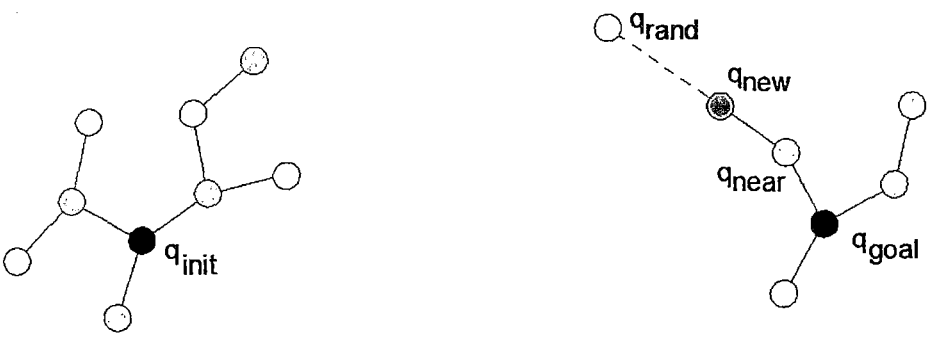

Figura 3.7: Un árbol genera un nuevo elemento (círculo gris oscuro).

Figura 3.8: El nuevo elemento del árbol se convierte en el objetivo (q_{target},) del otro árbol.

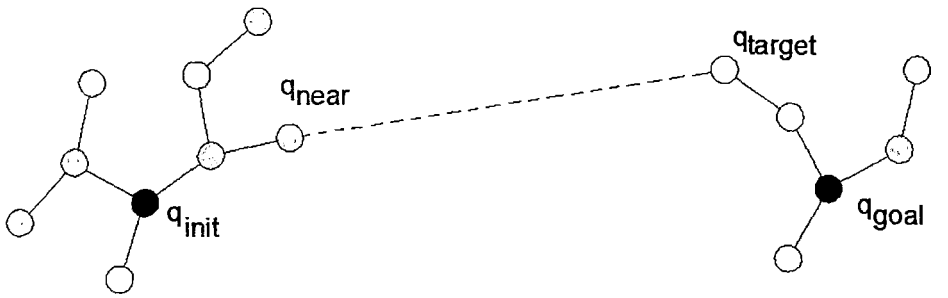

Figura 3.9: Calculando el elemento más cercano (q_{near}) a q_{target}.

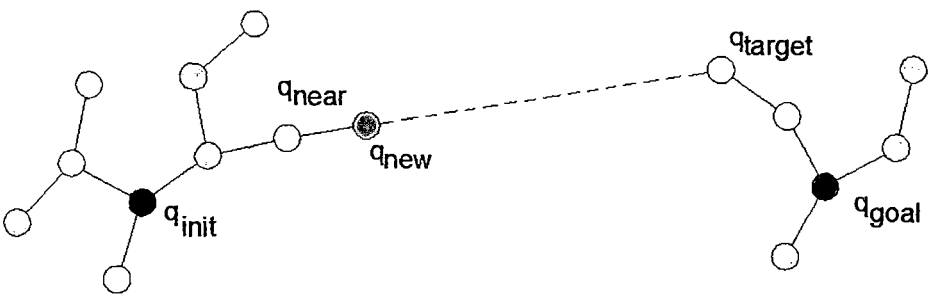

Figura 3.10: Cálculando nuevo elemento (q_{new}) del árbol de q_{init}.

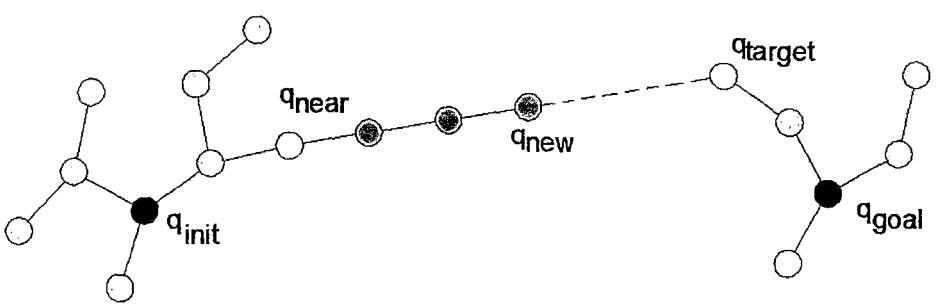

Figura 3.11: Adición consecutiva de elementos mientras no haya colisión.

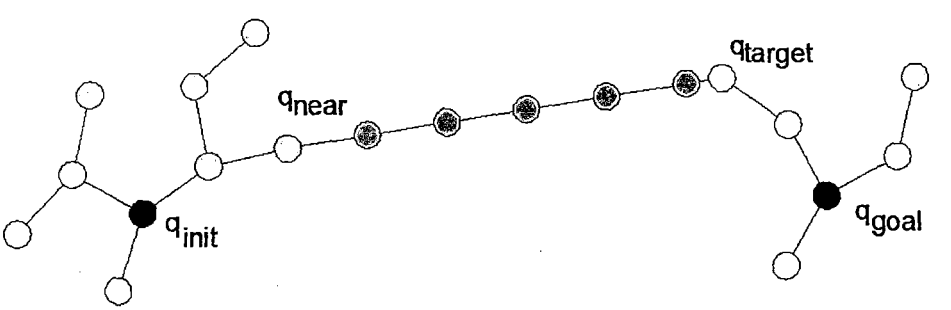

Figura 3.12: Conexión final del crecimiento.

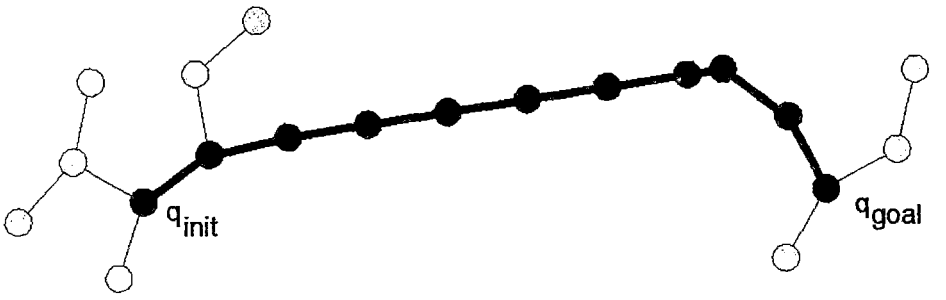

Figura 3.13: Tracking del camino de conexión.

3.4. Sistemas no holónomos

En la mecánica clásica un sistema puede ser definido como holonómico si todas sus restricciones son holonómicas y para que una restricción sea holonómica ésta debe ser expresada como una función:

$$f(x_1, x_2, x_3, ..., x_N, t) = 0 \tag{3.1}$$

Donde una restricción holonómica (ver Ecuación 3.1) se puede expresar solamente en términos de las variables de configuración (x) y del tiempo (t) mas no las derivadas de x. Es decir, una restricción no puede ser expresada como en la ecuación es un restricción no holonómica. Tales sistemas son por lo tanto, también conocidos como sistemas no integrables. Una característica clave de estos sistemas, es que no se puede pasar directamente de un configuración a otra, para realizar ello necesitan realizar una maniobra o una secuencia de movimientos como por ejemplo: robots móviles de 2 ruedas ó automóviles. Dichos sistemas no pueden realizar cualquier tipo de trayectoria, por ejemplo un carro de configuración Ackerman no puede rotar sobre su propio eje, en cambio un móvil de tipo omnidireccional puede realizar movimientos con radio de giro cero. Por tal razón se hace necesario tener en cuenta el modelo del sistema.

El algoritmo RRT se puede adaptar a sistemas de tipo no holonómico (ver Ecuación 3.2), representados de la siguiente manera (ver apéndice D.1.6):

$$\dot{x} = f(x, u) \tag{3.2}$$

Algoritmo RRT para sistemas no holónomos.

```
Función: RRT_non_hol(q_ini, K_max, Δt)
1. Arbol = q_init;
2. for 1 to K_max
3.   q_rand = Configuración_Aleatoria();
4.    a=Extiende(Arbol, q_rand);
5.     if (a=='alcanzado')
6.       Devuelve Ruta;
7.    endif;
8.  endif;
9. K = K + 1;
10. endfor;

Función: Extiende(Arbol, q_rand)
1. q_near= Vecino_mas_proximo(Arbol, q_rand);
2. u=seleccionar_entrada(q_rand, q_near);
3. q_new= Nuevo_elemento(q_near, u,Δt);
4. if( q_new ∈ Configuracion_libre)
5.  Arbol=[Arbol; q_new] //se añade nuevo vértice
6.   if ( abs(q_new=q_rand)< tolerancia) Devuelve 'alcanzado';
7.   else Devuelve 'avanzado';
8.   endif
9. else Devuelve 'rechazado';
10. endif

Función: Nuevo_elemento(q_near, u,Δt);
1. q_new=q_near+ẋ(u)*Δt;
2. Devuelve q_new;
```

Podemos notar que no se esta trabajando en la configuración de espacio sino en el espacio de estados y que el cálculo del nuevo elemento del árbol es recomendable utilizar el método de Runge-Kutta 4 para obtener mayor presición en sistemas de mayor número de grados de libertad, en el presente algoritmo se ha utilizado la aproximación de tangente para poder entender la idea del método.

CAPÍTULO 4

SIMULACIÓN Y PERFORMANCE

Para las simulaciones realizadas del algoritmo de *Path Planning* se ha utilizado el software Matlab, aunque el código generado también puede utilizarse en Octave (que es considerado como una de la versiones *Open Source*[1] de Matlab) ya que se han codificado las funciones sin utilizar toolbox especiales. Con respecto al equipo de procesamiento se ha utilizado una Core i5 con un clock de 2.4 Ghz.

En este capítulo se han de realizar las simulaciones correspondientes de los métodos derivados del algoritmo RRT, tanto en sistemas holónomos como en los no holónomos así como las consideraciones de simplificar al robot en un punto y tomar en cuenta sus dimensiones. A su vez, se apreciará lo referente al uso de algoritmo de optimización de búsqueda que permiten mejorar el computo de un algoritmo que requiera realizar búsqueda en estructuras de datos con alta cantidad de elementos.

4.1. Simulación del algoritmo RRT en sistemas holónomos

En esta sección se considera para las simulaciones a realizar un robot en el espacio 2D que no tenga restricción de moviento en cuanto a traslación y rotación.

[1]Es el término asociado al software distribuido y desarrollado libremente, que garantiza a cualquier persona el derecho de usar, modificar y redistribuir el código libremente. A su vez, se debe permitir crear trabajos derivados, que deben ser distribuidos bajo los mismos términos que la licencia original del software.

4.1.1. Algoritmo RRT

Tal como se había explicado en el capítulo anterior el algoritmo RRT es un algoritmo de exploración de las configuraciones libres. En la Figura 4.1 se tiene un espacio de 40mx40m con una longitud de segmento de crecimiento de 5cm y se puede apreciar que al aumentar el número de iteraciones el árbol explorador con origen en (0,0) tiende a crecer a lo largo del espacio libre, visualizándose así la naturaleza exploradora del algoritmo. Este mismo comportamiento se puede apreciar en la Figura 4.2, a través de un espacio que presenta obstáculos (representados por polígonos de color negro).

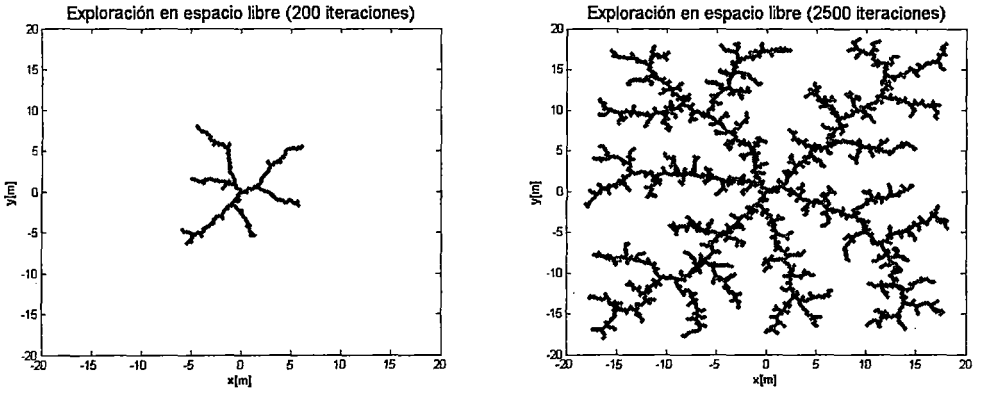

Figura 4.1: Algoritmo RRT explorando en configuración libre de obstáculos.

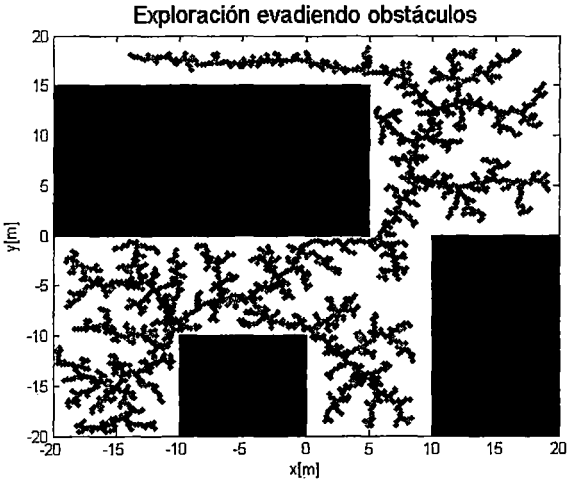

Figura 4.2: Algoritmo RRT explorando a traves de obstáculos.

4.1.2. Algoritmo RRT bidireccional

En este algoritmo la idea es encontrar un punto común de crecimiento entre los 2 árboles exploradores (ver Figura 4.3), siendo los orígenes de dichos árboles los puntos A y B. Aparentemente se puede apreciar muchos posibles puntos comunes de crecimiento, lo que permite intuir que el procedimiento puede mejorarse ya que el criterio de parada es que ambos árboles colisionen en el preciso instante que se expanden hacia el mismo punto.

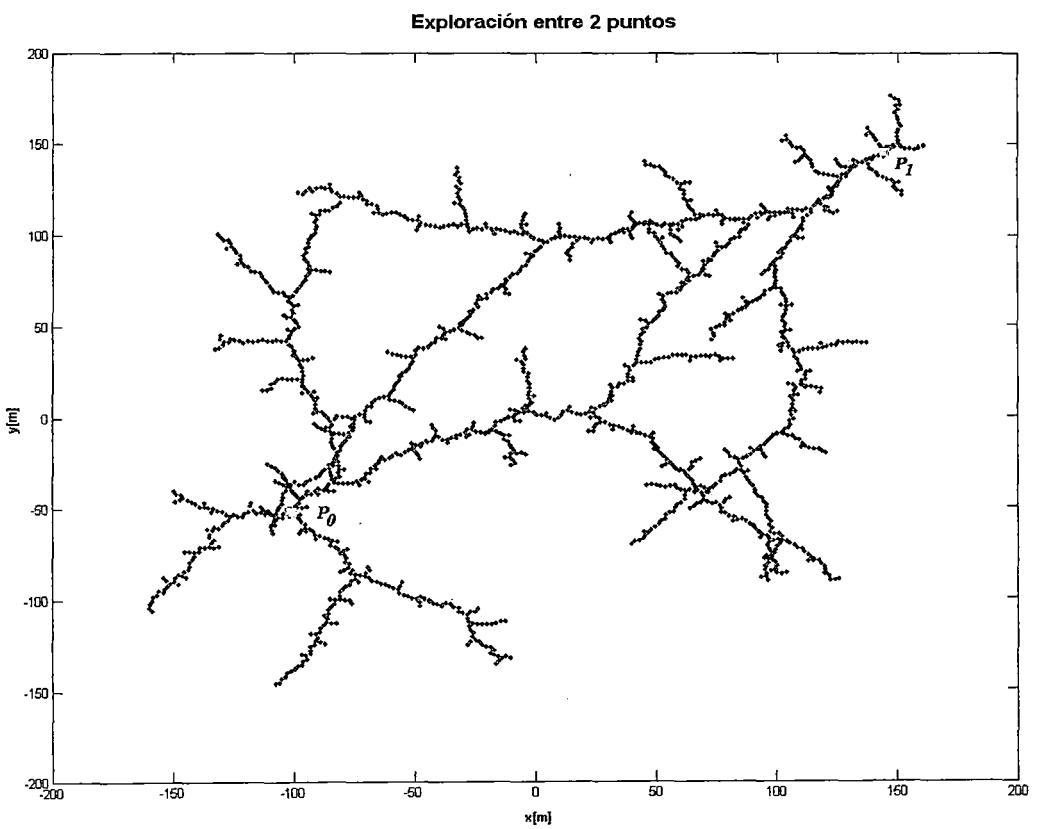

Figura 4.3: RRT bidireccional básico en espacio libre.

4.1.3. Algoritmo RRT Ext-Ext

Es una mejora del algoritmo RRT bidireccional, explicado en el capítulo 3. En la Figura 4.4 se aprecia una compación entre la búsqueda básica bidireccional y la búsqueda mediante RRT Ext-Ext. Si bien ambos métodos buscan la intersección de los árboles exploradores, el algoritmo RRT-Ext-Ext genera 385 iteraciones, en cambio el otro 781 iteraciones. A su vez el camino obtenido con el RRT-Ext-Ext tiende a ser una línea, como correspondería a 2 puntos en un espacio libre de obstáculos.

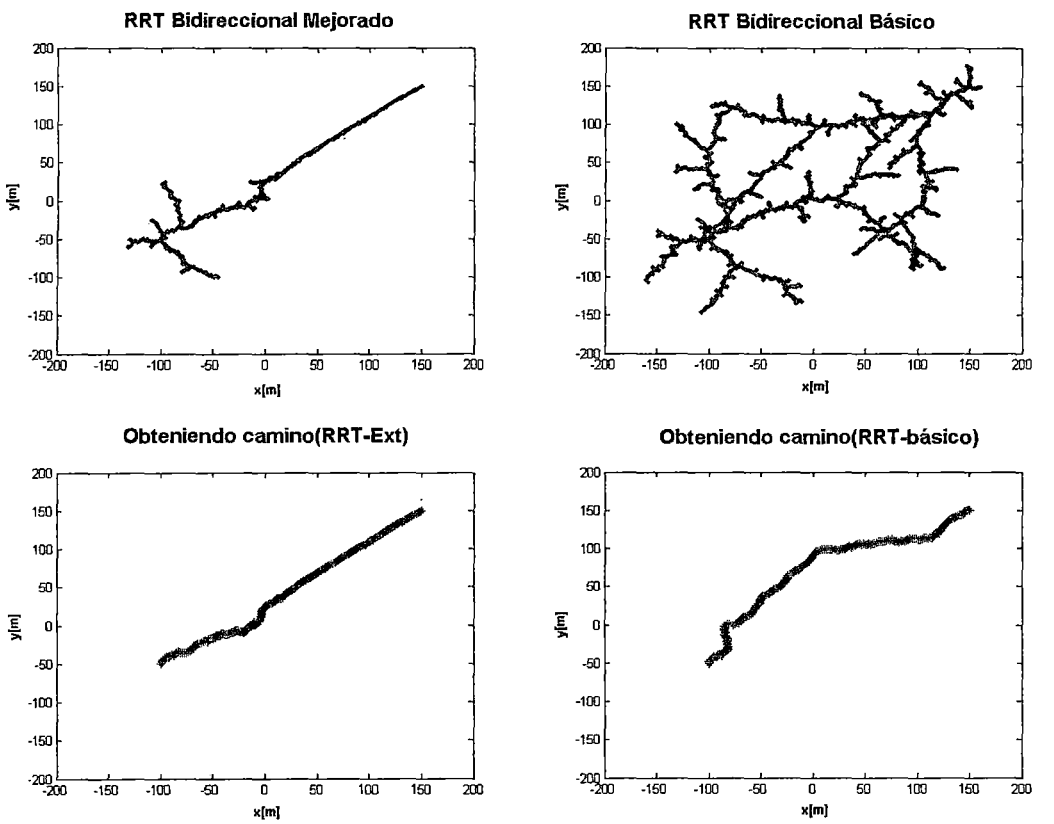

Figura 4.4: A la izquierda algoritmo RRT-Ext-Ext, a la derecha el algoritmo RRT básico bidireccional. Comparación entre ambas variaciones de RRT.

4.1.4. Algoritmo RRT Ext-Con

Es también otra mejora del algoritmo RRT bidireccional, el crecimiento del árbol es variable ya que la expansión del segmento de crecimiento se da consecutivamente hasta ser interrumpida por una configuración prohibida (obstáculo).

En la Figura 4.5 se puede apreciar los árboles explorados que a comparación de los métodos previos no presentan tantas ramas.

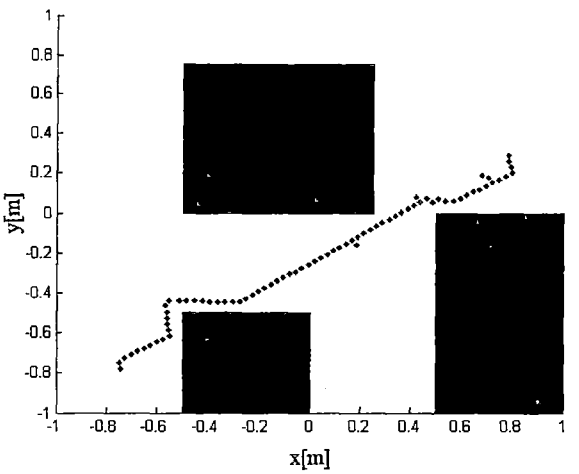

Figura 4.5: Árboles exploradores en RRT-Ext-Con

En la Figura 4.6 se aprecia el cálculo de las partes de los árboles que conforman el camino.

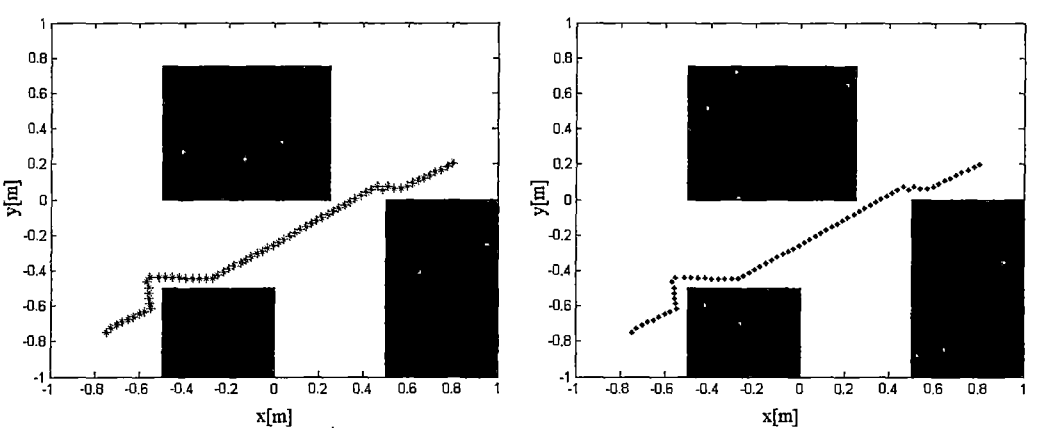

Figura 4.6: Tracking de árboles que forman la ruta entre los puntos deseados.

Se presenta a continuación una simulación considerando las dimensiones del robot moway con un rectángulo, es decir ya no se considera al robot como sólo un punto, sino que también se toma en cuenta sus dimensiones (ver Figura 4.7).

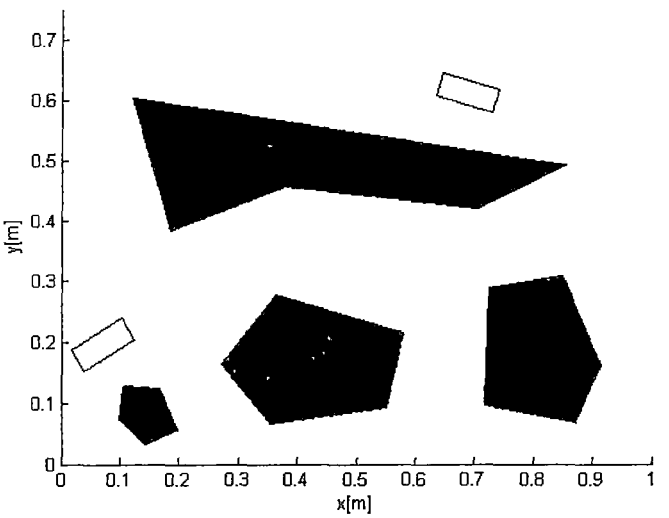

Figura 4.7: Configuración inicial (0,1;0,2) y final (0,7;0,6) del robot Moway así como su entorno con obstáculos.

En la Figura 4.8, apreciamos los árboles exploradores que conectan la configuración inicial y final.

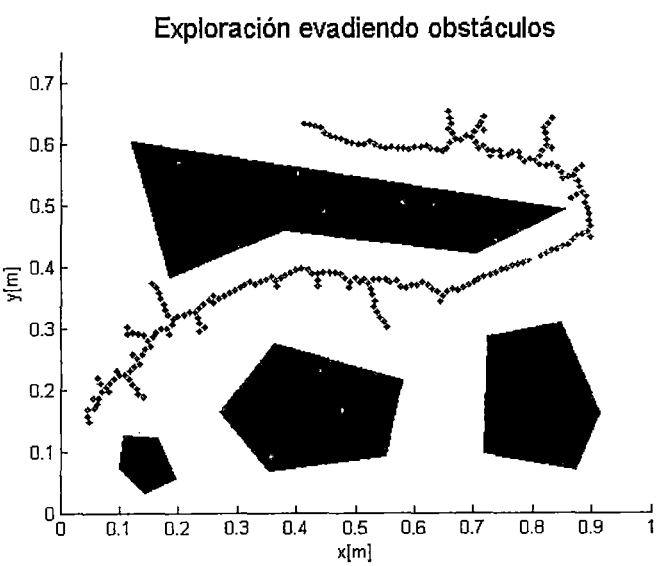

Figura 4.8: Árboles exploradores entre las configuraciones deseadas.

En la Figura 4.9 se aprecia el camino obtenido que une a las configuraciones en cuestión.

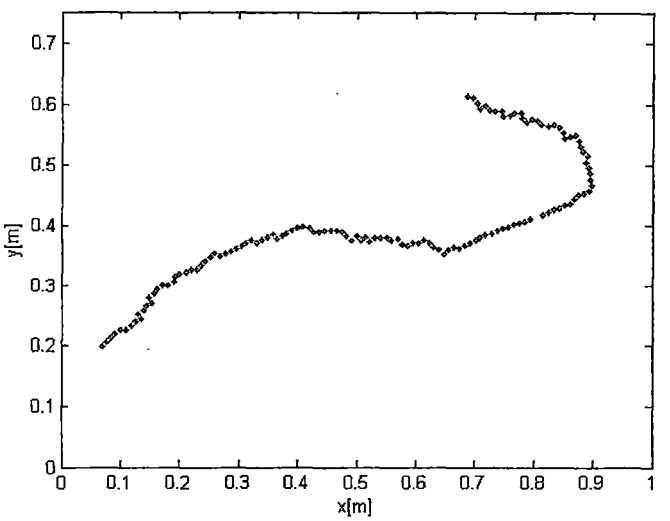

Figura 4.9: Tracking de la ruta entre las configuraciones inicial y final.

Los movimientos que realiza el robot para llegar de la configuración inicial a la final se aprecian en la Figura 4.10.

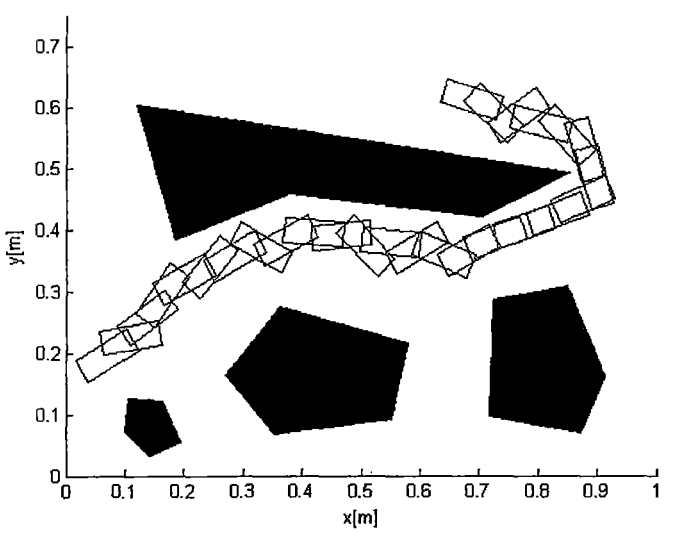

Figura 4.10: Secuencia de movimiento entre las configuraciones inicial y final en el entorno.

4.2. Aplicación de optimización de búsqueda

En este apartado se mostrará las simulaciones del tiempo de cómputo de los algoritmos RRT bidireccional y RRT-Ext al utilizar un método de búsqueda con el algoritmo **KNN Search**[2] . La búsqueda esta contenida en el cálculo de la configuración más cercana del árbol explorador a la configuración aleatoria. Esta búsqueda se hace crítica al tener un alta cantidad de datos, ya que no es lo mismo buscar en 10 datos que en 500 por ejemplo y considerando que el árbol de exploración aumenta de dimensión considerablemente es necesario tratar de optimizar la codificación del algoritmo. Esto influye notablemente en el tiempo de cálculo de los algoritmos.

En la Figura 4.11 se tiene un ambiente con 2 configuraciones a unir y se puede apreciar que no existe un único camino, en dicha gráfica se aprecian 2 posibles maneras.

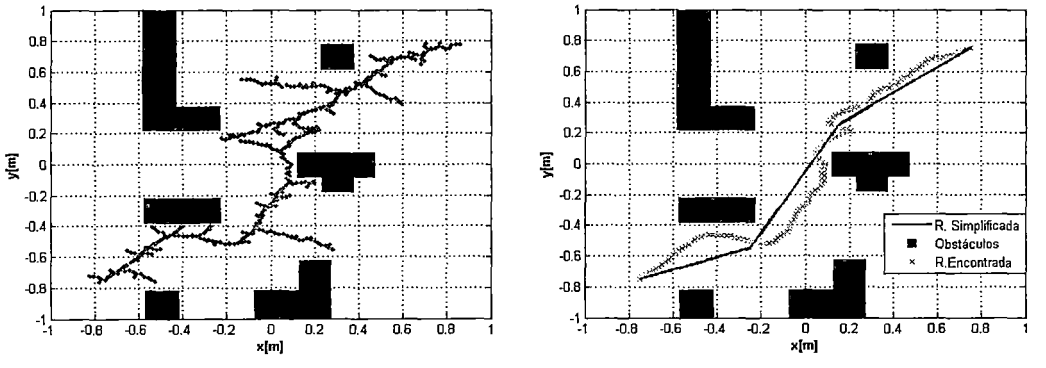

Figura 4.11: Algunas rutas encontradas en la exploración del algoritmo RRT.

Se han simulado 100 veces el cálculo de una ruta que permita conectar la configuración 1 (-0,7;-0,7) y la configuración 2 (0,7;0,7) en un entorno con obstáculos y se muestran los respectivos tiempos de cálculo obtenidos de cada iteración (ver Figura 4.12).

[2]Algoritmo de búsqueda K-nearest neighbour, el cual es un método que calcula a los K vecinos más cercanos de un elemento respecto a una data dada.

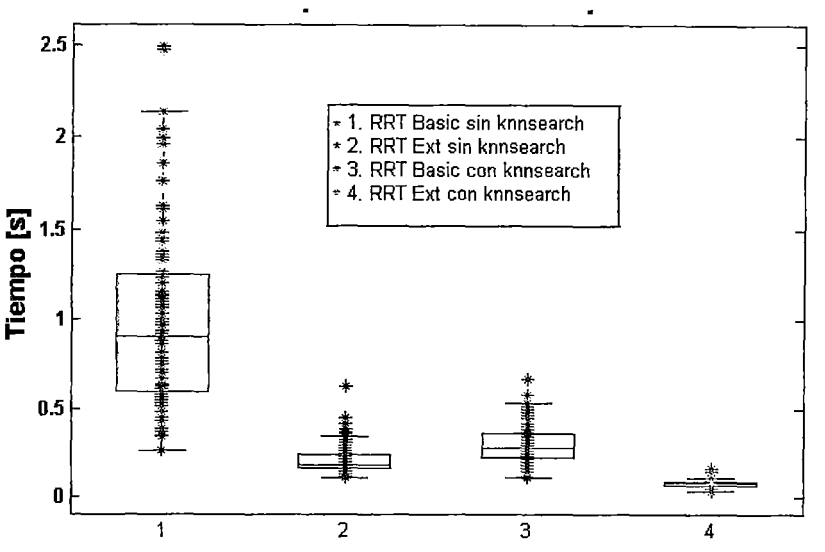

Figura 4.12: Comparación de tiempos de cálculo del algoritmo RRT en 100 simulaciones de un mapa.

Al utilizarse un algoritmo estocástico que obtiene una serie de caminos posibles con un variado tiempo de cálculo en la obtención de la ruta a seguir, se han utilizado tanto la media como la desviación estándar como indicadores. La Tabla 4.1 nos muestra dichos valores para los 4 casos considerados en esta pequeña prueba.

	Media	Desviación estándar
(1) RRT Basic sin knnsearch	0.905	0.48
(2) RRT Ext sin knnsearch	0.187	0.079
(3) RRT Basic con knnsearch	0.281	0.104
(4) RRT Ext con knnsearch	0.078	0.023

Cuadro 4.1: Tabla comparativa con los tiempo de cálculo sobre una data de 100 elementos.

Se puede notar que el algoritmo RRT básico en promedio, necesita alrededor de 0.905 segundos para encontrar el camino y el algoritmo RRT solo necesita 0.187 segundos, siendo esta, una gran mejora en el tiempo. En el otro lado, sus versiones optimizadas con el algoritmo de búsqueda de KNN muestran prestaciones, incluso mejor, que sólo necesitan 0.281 y 0.078 segundos respectivamente.

4.3. Simulación del algoritmo RRT en sistemas no holónomos

En este caso, se toma en consideración las restricciones que presentan los sistemas, en ésta caso del robot Moway. Por ello se hace necesario el modelar el sistema para poder determinar a priori que restricciones ha de presentar dicho sistema.

4.3.1. Modelamiento Cinemático del Sistema

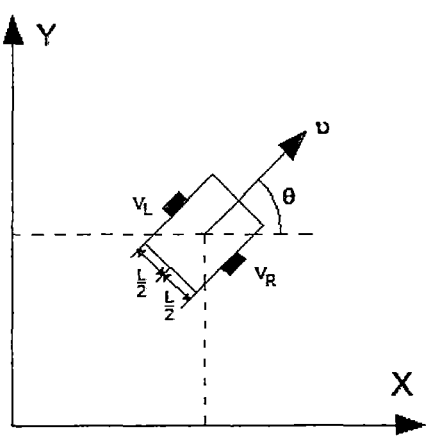

Figura 4.13: Ruta entre los puntos.

La cinemática del robot Moway esta expresada por:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \upsilon \\ \omega \end{bmatrix} \tag{4.1}$$

Pero para el presente caso se ha de expresar en función de las velocidades de cada rueda:

$$\upsilon = \frac{V_R + V_L}{L} \tag{4.2}$$

$$\omega = \frac{V_R - V_L}{2} \tag{4.3}$$

Reemplazando 4.2 y 4.3 en 4.1, se tiene:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{cos(\theta)}{2} & \frac{cos(\theta)}{2} \\ \frac{sin(\theta)}{2} & \frac{sin(\theta)}{2} \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix} \cdot \begin{bmatrix} V_R \\ V_L \end{bmatrix} \tag{4.4}$$

Simulando el sistema para que calcule un camino entre las configuraciones inicial y final, teniendo en cuenta la orientación del robot no solo la posición:

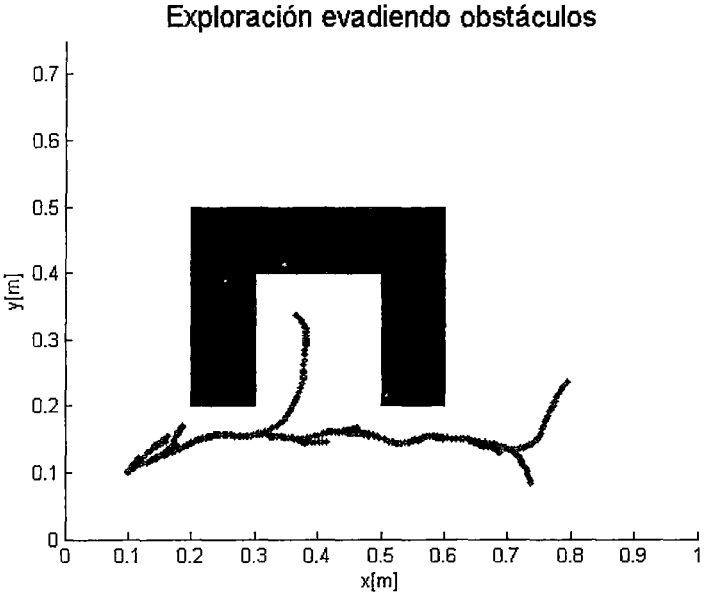

Figura 4.14: Ruta entre las configuraciones inicial (0,1;0,1;$\pi/4$) y final (0,8;0,25;$\pi/3$) en el entorno dado.

En la Figura 4.15 se aprecia el camino obtenido por el algoritmo RRT en un sistema no holónomo con las restricciones de movimiento del sistema en cuestión.

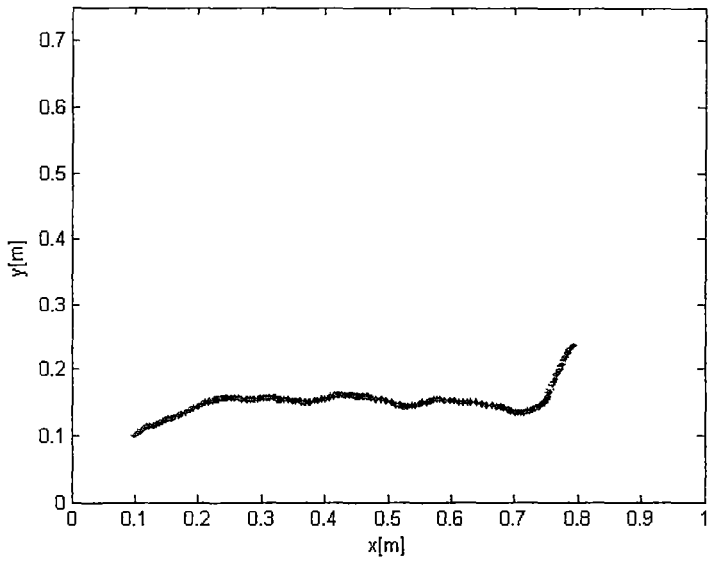

Figura 4.15: Ruta a seguir por el robot Moway.

En la Figura 4.16 se visualiza como sería la manera en la que el robot se desplazaría por el camino calculado.

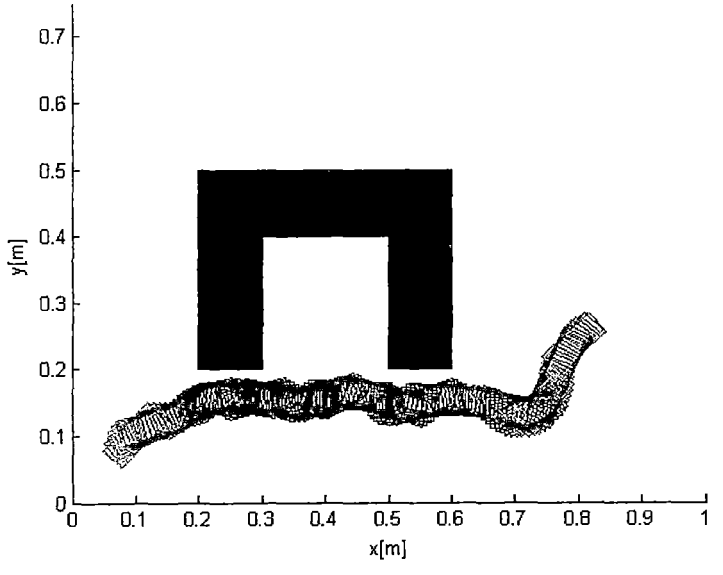

Figura 4.16: Tracking del movimiento del robot Moway desde la configuración inicial a la final.

CAPÍTULO 5

IMPLEMENTACIÓN DE UNA PLATAFORMA E INTERFAZ

Para la implementación física del planeador de rutas se ha desarrollado una interfaz de usuario, mediante la cual se ha de ejecutar tareas a especificar por un algoritmo dado. Asimismo, han realizado las pruebas físicas sobre la siguiente plataforma de madera que se detalla a continuación (ver Figuras 5.1, 5.2 y 5.3):

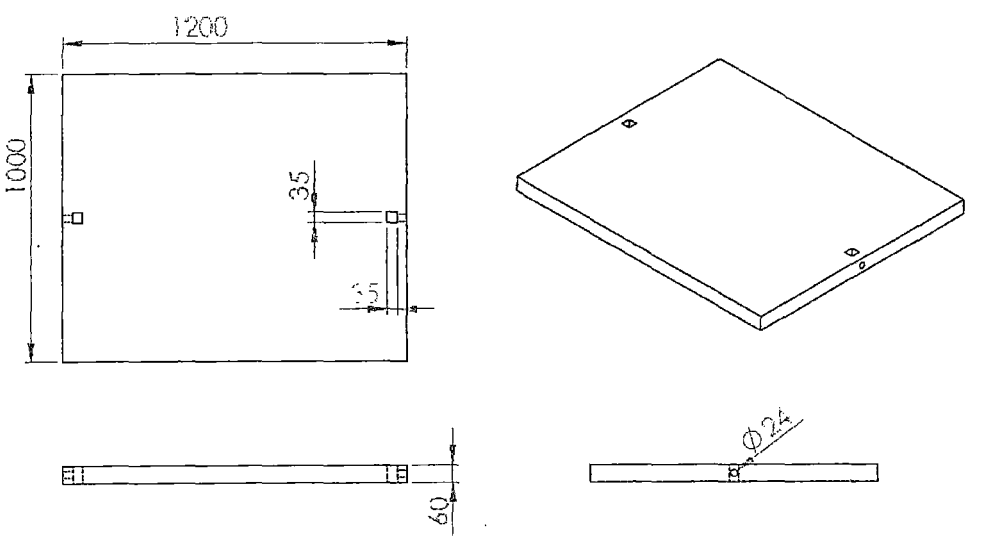

Figura 5.1: Plano de base de la plataforma. Unidades milímetros.

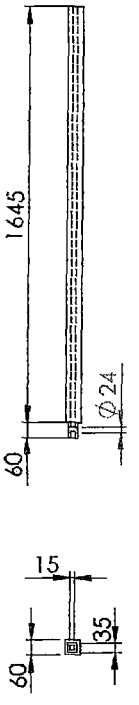

Figura 5.2: Plano de columna de soporte. Unidades milímetros.

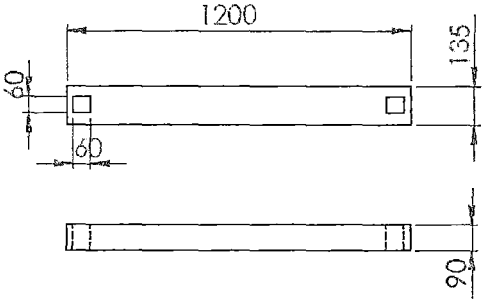

Figura 5.3: Plano de soporte de cámara. Unidades milímetros.

La plataforma descrita forma parte de una pequeña planta, cuya conformación está dada por (ver Figura 5.4):

- Robot Moway
- Cámara Hallion
- Plataforma

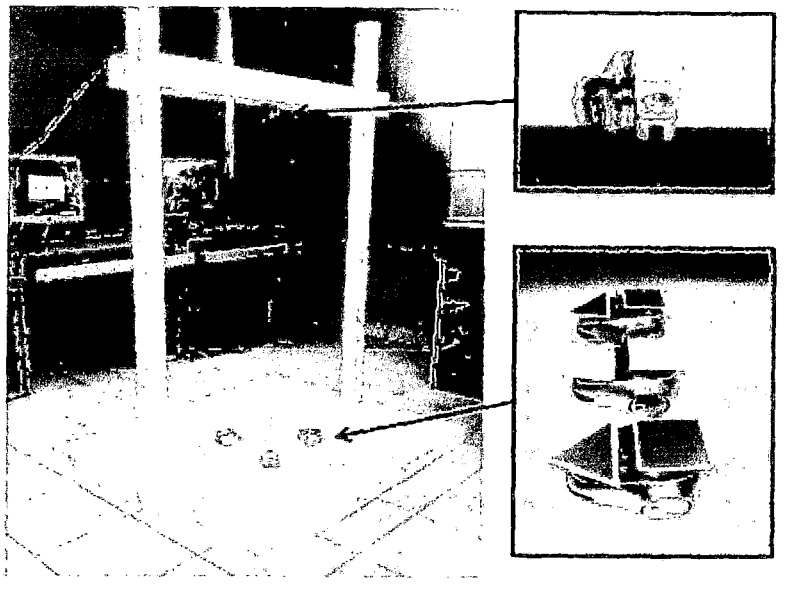

Figura 5.4: Planta de Prueba

En las pruebas se ha trabajado con el robot móvil Moway (ver apéndice B) de la compañía MiniRobots[1] por las prestaciones que tiene. La Facultad de Ingeniería Mecánica de la Universidad Nacional de Ingeniería a través de INIFIM[2] cuenta con 3 robots Moway de iguales características, por tal razón se ha considerado colocarles un color característico para poder diferenciarlos durante el trabajo con los mismos.

Por tal razón surge la necesidad de crear una interfaz que permita controlar a dichos robots para así poder gestionarlos con flujos de datos de otros programas o algoritmos que se desarrollen. Si bien es cierto que el robot Moway cuenta con un software propio para adiestramiento y manipulación inalámbrica (mediante tecnología de radiofrecuencia), dicho software no permite trasladar data o interactuar con otros programas así como tampoco permite enviar órdenes de movimiento a varios robots Moway. En este capítulo se explicará cómo se realizo dicha interfaz que tiene como principales características los siguientes puntos:

[1]Es la empresa fabricante del robot Moway con sede central en Barakaldo-España. La dirección web de su producto es www.moway-robot.com

[2]Instituto Nacional de Investigación de la Facultad de Ingeniría Mecánica.

- Comunicación inalámbrica con varios robots Moway
- Posibilidad de flujo de data con otros programas
- Reconocimiento de robots mediante visión artificial
- Indicadores de posición y orientación para 3 robots
- Planificador de rutas

Para lograr lo mencionado anteriormente se ha utilizado una cámara como sensor, cuya data es procesada por la interfaz para efectos del cálculo de rutas. La interfaz puede controlar no sólo un robot moway, ésta ha sido diseñada para reconocer hasta 3 robots, sin embargo, esto puede escalarse aumentando el número de pantallas y configurando los filtros.

5.1. Consideraciones

Las simulaciones del planeador de rutas fueron realizadas en el software Matlab, para el desarrollo del sistema de visión y la interfaz de comunicación con el robot Moway se empleó la plataforma de Visual C#[3](debido a que los controladores del robot moway están diseñados en ese lenguaje) y las librerías de visión Emgu[12] (wrapper[4] de OpenCV[5] para C#); la integración entre Matlab y Visual C# se dio mediante el uso de dll[6] s; sin embargo, no se descarta una posterior codificación completa del algoritmo de Path Planning en C#.

[3]Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET, similar al de Java aunque incluye mejoras derivadas de otros lenguajes (entre ellos Delphi).

[4]Envoltura, para este caso permite en C# accesar a las funciones contenidas en la libreria OpenCV.

[5]Open Source Computer Vision: es una librería con funciones para el desarrollo de visión computacional, escrita inicialmente en C y posteriormente en C++

[6]*Dynamic Library Link* (biblioteca de enlace dinámico), es la extensión con el que se identifican a los archivos con código ejecutable que se cargan bajo demanda de un programa por parte del sistema operativo. Esta denominación es exclusiva a los sistemas operativos Windows.

5.2. Funcionamiento de la interfaz

Para el presente caso el procedimiento a seguir es:

1. Obtener la data del entorno mediante la cámara, así como la posición y orientación inicial del robot Moway.

2. Calcular la ruta a seguir por el algoritmo de *Path Planning*, teniendo en cuenta la información recogida por la cámara (entorno del robot y configuración inicial) así como la configuración deseada.

3. Inicio de un controlador de trayectoria, el cual transmite las órdenes necesarias para que el robot realice la ruta calculada (ver Figura 5.5).

4. Calcular y enviar las órdenes de movimiento a través de la RF-USB desde la PC al robot.

5. Obtener la posición y orientación del móvil, esta data es enviada a la interfaz descrito en el paso 3 para corroborar si se sigue la ruta calculada por el planificador de caminos.

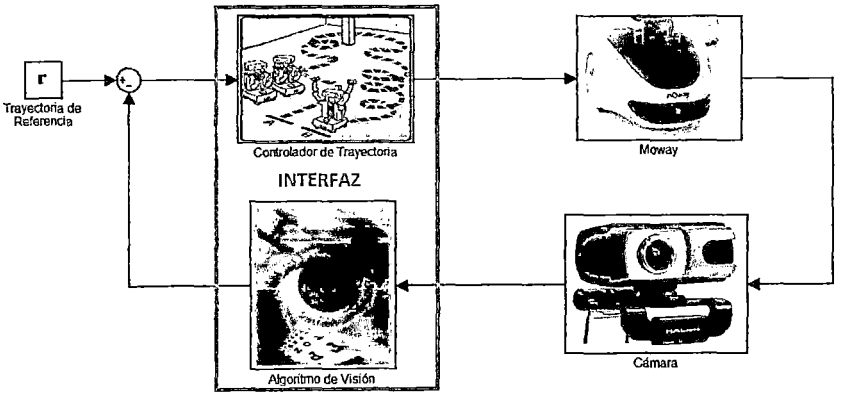

Figura 5.5: Retroalimentación para el seguimiento de la ruta calculada.

5.3. Características de la interfaz

La presente interfaz agrega visión computacional a la interfaz ejemplo de Moway, así como también un planificador de caminos que es el objetivo de la presente tesis (ver Figura 5.6).

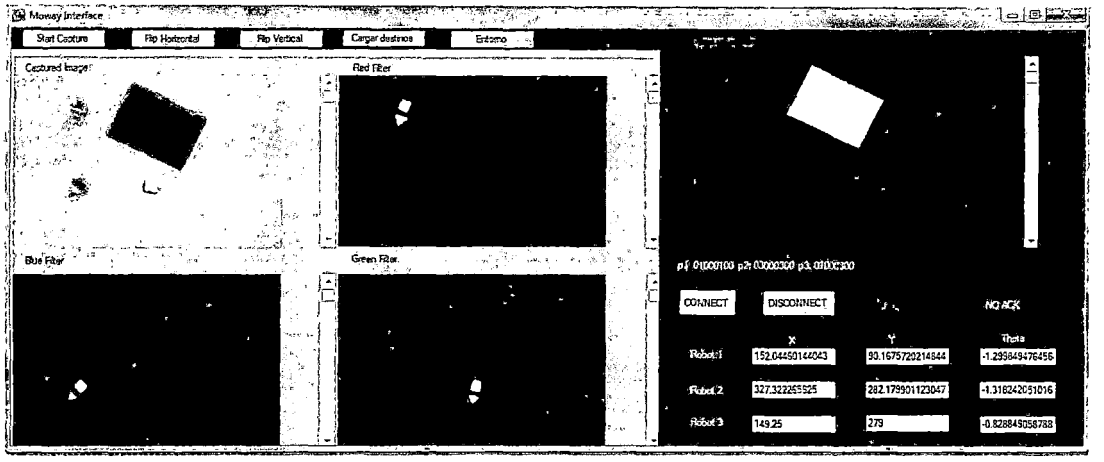

Figura 5.6: Interfaz de manipulación de robots móviles Moway con retroalimentación de posición y orientación.

La interfaz de usuario desarrollada presenta los siguientes elementos visuales:

- 5 pantallas de procesamiento de imágenes: 1 de lo captado por la cámara, 3 de reconocimiento de los móviles y 1 del entorno.
- Botón de **Flip Vertical**, que permite voltear la imagen verticalmente.
- Botón de **Flip Horizontal**, que permite voltear la imagen horizontalmente.
- Botón **Start Capture**, se inicia la captura y procesamiento de los frames (cuadros ó fotogramas) captados por la cámara.
- Botón **Cargar destinos**, la ruta calculada es cargada a la interfaz que tiene como objetivo que el robot la realice.
- Botón **Entorno**, captura los obstáculos presentes en el entorno.

- Indicadores de posición y orientación de los robots, siendo la posición expresada en pixeles y la orientación de los robots en radianes (ver Figura 5.7).

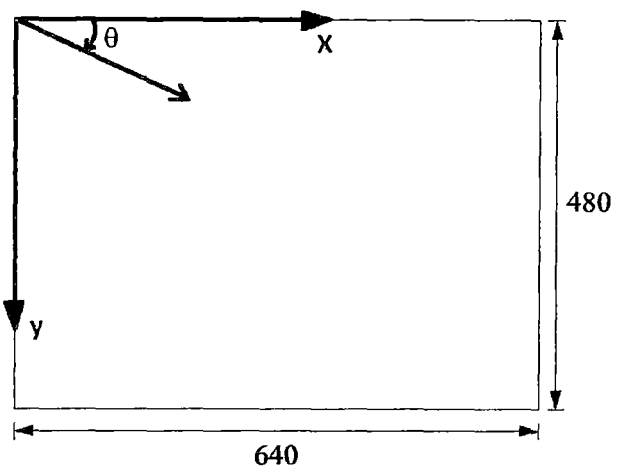

Figura 5.7: Sistema de referencia para la interfaz.

Para la manipulación de la interfaz se ejecutan las siguientes etapas (ver Figura 5.8):

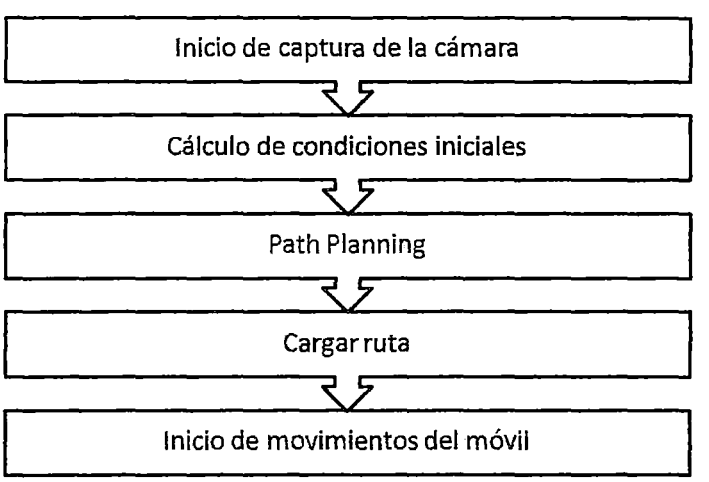

Figura 5.8: Secuencia de manipulación de la interfaz.

- Inicio de cáptura de la cámara, en esta etapa la interfaz inicializa la cámara y muestra en las pantallas correspondientes.

- Cálculo de condiciones iniciales, en este paso se calcula la posición y orientación inicial del móvil. Así como también la información de los obstáculos, que se expresa en una matriz binaria.

- *Path Planning*, con la información del entorno y la configuración inicial del móvil se procede a calcular la ruta ó camino que lleve al móvil a la configuración final o deseada.

- Cargar ruta, se procede a cargar la ruta calculada por el algoritmo de *Path Planning* a la interfaz. Se considera como ruta un conjunto de configuraciones progresivas hasta alcanzar la configuración final. En esta etapa se cálculan los movientos a realizar por el móvil entre una configuración y la siguiente.

- Inicio de movimientos del móvil, si bien en el paso anterior ya se habían calculado las órdenes de movimiento, es en este paso que se habilita el envío de dichas órdenes mediante la habilitación de la RF-USB. El móvil ha de realizar sucesivos movimientos hasta llegar a la configuración deseada.

Lo expresado anteriormente ha sido codificado en un máquina de estados (ver Figura 5.9 y apéndice D.2.1), la cual es ejecutada por la interfaz mediante un *timer*[7] (ver apéndice D.2.1) configurado cada 10 ms. Los estados de dicha máquina de estados están descrita por:

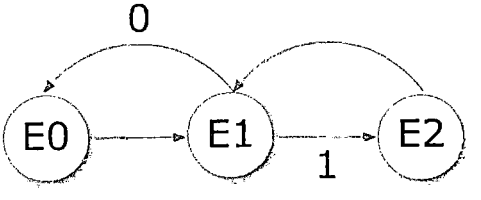

Figura 5.9: Secuencia de la máquina de estados

[7]Es un tipo especializado de reloj digital, que tiene un indicador cuando dicho *timer* alcanza el tiempo configurado.

- **E0:** Estado inicial
- **E1:** Estado de visión. En este estado se ejecuta la función ProcessFrame (nombre de la función en el código de la interfaz), la cual calcula la posición y orientación de los robots Moway así como también muestra en las pantallas el procesamiento realizado por la interfaz para entender el funcionamiento de la parte de visión artificial.
- **E2:** Control y envío RF. Se carga la ruta de referencia, se calcula el movimiento del robot y se envia dicha orden inalámbricamente mediante la RF.

Para que se pase del estado E1 al estado E2 se requiere de un indicador (que se denomina "flagstarcamera" en el código de la interfaz) que este en uno lógico, es decir, que la cámara este capturando el entorno. En caso que no haya conexión de la cámara se retornará al estado E0.

5.4. Componentes de la interfaz de usuario

5.4.1. Comunicación Inalámbrica

En esta sección se da la comunicación inalámbrica entre la PC y el robot Moway utilizando el módulo BZI-RF2GH4 de radiofrecuencia cuyas características técnicas se pueden apreciar en el cuadro 5.1.

Parámetro	Valor	Unidad
Tensión mínima de alimentación	1.9	V
Tensión máxima de alimentación	3.6	V
Potencia máxima de salida	0	dBm
Velocidad máxima de transmisión	2000	Kbps
Corriente en modo transmisión @ 0dbm potencia de salida	11.3	mA
Corriente en modo recepción @ 2000kbps	12.3	mA
Corriente en modo Power Down	900	nA
Frecuencia máxima del bus SPI	8	Mhz
Rango de temperatura	-40 a +85	^{o}C

Tabla 5.1: Parámetros principales del módulo BZI-RF2GH4, datos tomados de [1].

Las características principales del módulo BZI-RF2GH4 son:

- Bajo consumo.
- Frecuencia de trabajo de 2.4GHz
- Potencia de emisión entre -18 y 0 dBm
- Velocidad de transmisión entre 1 y 2 Mbps
- 128 canales de transmisión seleccionables por el bus SPI.

Con el objetivo de facilitar el manejo del módulo, BIZINTEK INNOVAS.L. ha desarrollado unas librerías tanto en ensamblador como en C que simplifican y acortan el tiempo de desarrollo de aplicaciones inalámbricas con estos módulos. A su vez para desarrollar una aplicación mediante una interfaz desde la PC se utilizan las dlls lib_nrf241u1_RF y lib_prog_mow2, las cuales están compiladas para C#. Desde la interfaz se ha de recibir la ruta calculada y se ha de calcular las órdenes de movimiento para realizar dicho camino, a su vez se esta considerando retroalimentar la posición y orientación del robot Moway (ver Figura 5.5).

Para la presente aplicación se han considerado las siguientes órdenes de movimiento para el robot moway (considerando que siempre avanza, ver apéndice D.2.2):

- Adelante
- Giro a la derecha
- Giro a la izquierda

Estas órdenes de movimientos se han definido debido a la restricción física que presente el robot en cuya hoja técnica se describe el comportamiento en lo que respecta a la rapidez de sus ruedas en función al *duty cycle* (ciclo de trabajo) del

motor DC que genera el movimiento rotacional. Ello se representa a través de la siguiente gráfica:

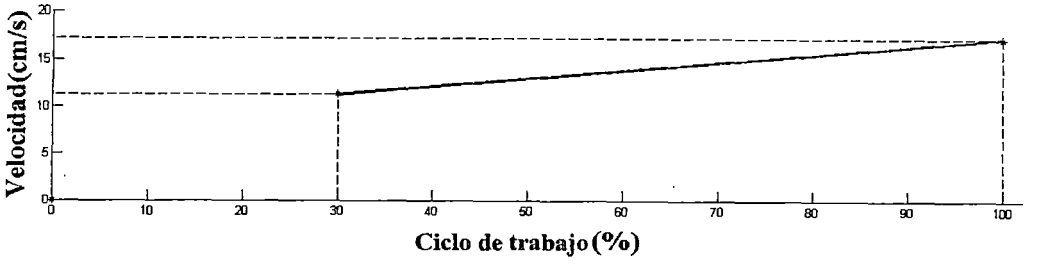

Figura 5.10: Rapidez de la rueda del robot Moway respecto al ciclo de trabajo del motor DC.

De la Figura 5.10 se puede apreciar que no contamos con el manejo completo de los duty cycle de las ruedas debido a que la rapidez de la rueda no está definida en el intervalo 1-29. A su vez, la rapidez al 30 % de duty cycle con el 100 % son próximos, por tal razón en vista de que no hay gran diferencia entre dichas rapideces se ha considerado trabajar con una representación equivalente (ver Figura 5.11), en ella se ha decido trabajar con la rapidez máxima y la cero.

Figura 5.11: Rapidez equivalente de la rueda del robot Moway respecto al ciclo de trabajo del motor DC.

Una vez aclarado el criterio tomado para los tipos de movimientos a utilizar en el robot, se procederá a explicar como se ha de enviar las órdenes de movimiento para que el robot moway las cumpla. Como se había mencionado anteriormente se ha tomado 4 órdenes principales para el movimiento robot asignándoles números a dichas órdenes, ver el cuadro 5.2.

Orden de movimiento	Número
Giro a la derecha	1
Giro a la izquierda	2
Adelante	3
Alto	4

Tabla 5.2: Asignación de números para las órdenes de movimiento

Con respecto al protocolo de comunicación que usa el robot Moway, se designa a todo el paquete de información transmitido por la radiofrecuencia (entiéndase por cabeceras y la información propiamente dicha) como "Data". La estructura de "Data" se muestra a continuación:

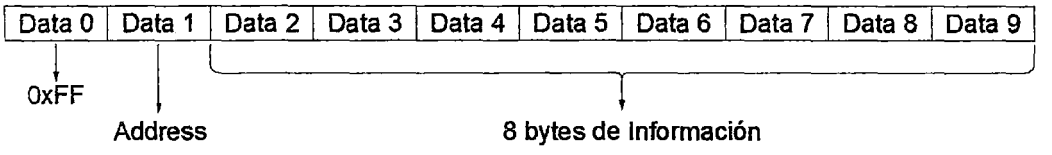

Figura 5.12: Estructura del paquete total enviado a traves del protocolo de la RF del robot Moway

Como se puede apreciar de la Figura 5.12, el paquete "Data" es en si una estructura de 10 bytes, siendo los 2 primeros de configuración y teniendo 8 bytes para envío de información desde la PC al robot Moway en cuestión. El primer byte de "Data"(**Data 0**) está seteado a una constante en hexadecimal 0xFF que es 255. El segundo byte de "Data" (**Data 1**) contiene el Address o dirección del dispositivo (robot Moway o PC) que recibirá la información. Finalmente desde **Data 2** a **Data 9** disponemos de estos bytes para la información que deseamos transmitir entre 2 puntos (sea PC-Moway ó Moway-Moway).

Si bien es cierto que se puede configurar el envío y recepción para la comunicación, el éxito de esta depende de un protocolo de comunicación el cual esta contenido en las librerías desarrolladas por Moway. Por ejemplo se describe que al enviar data de un punto a otro hay un tiempo de espera maximo de 16ms, tiempo en cual se espera recibir un flag o indicador del éxito de la comunicación

caso contrario se puede retransmitir la información. A su vez es importante indicar que para el desarrollo del presente proyecto se ha considerado el problema de la interferencia de otras ondas inalámbricas como las de WIFI, las cuales entorpecen el accionar de las RF de los robots Moway, si éstas son bastantes y si su intensidad es alta, para ello una recomendación del fabricante es el cambio de canal para evitar interferencias. Ello se puede apreciar en la siguiente figura:

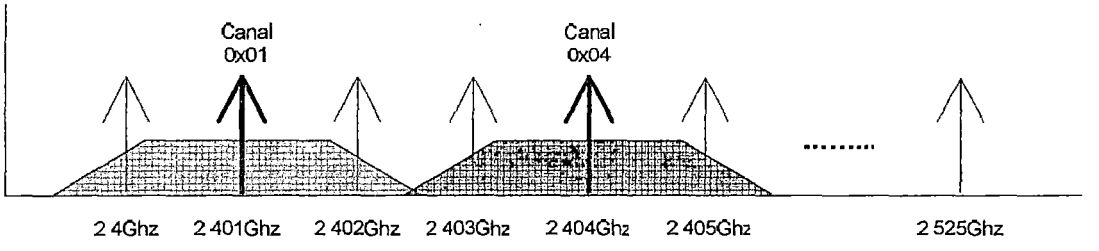

Figura 5.13: Canales del ancho de banda de la frecuencia del RF MOWAY.

5.4.2. Visión Artificial

La visión artificial, también conocida como visión por computador, es un subcampo de la inteligencia artificial. El propósito de la visión artificial es programar un dispositivo como un computador para que interprete una escena o las características de una imagen (en este caso sería de los frames captados por la cámara). Para el presente caso la data requerida es:

- El entorno del robot: obstáculos y espacios libres del entorno, esta información tendrá el formato de una matriz binaria, la cual es el dato del mapa para el desarrollo del *Path Planning.*

- Posición y orientación del robot Moway: se necesita de algún sensor que verifique si el robot sigue la ruta calculada para corregir su movimiento. Con ello se tendría un sistema retroalimentado.

Como se había mencionado anteriormente en la interfaz una función denominada ProcessFrame realiza tanto el cálculo de la posición y orientación de los robots Moway así como el reconocimiento de las características del entorno. Ahora se procederá a explicar lo referente a dicha función (ver Figura 5.14 y Figura 5.15):

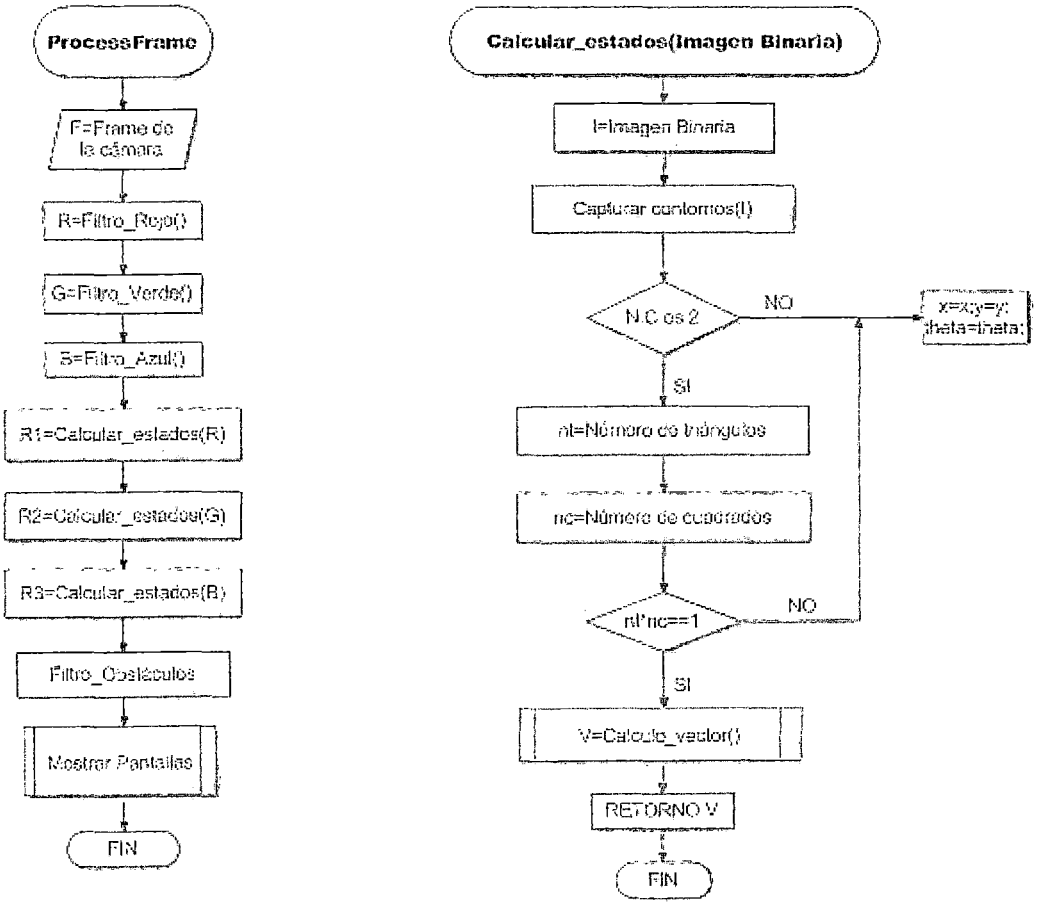

Figura 5.14: Función ProcessFrame

En el apéndice D.2.3 se puede apreciar lo referente a la codificación de la función ProcessFrame y las consideraciones tomadas como por ejemplo en el caso de los cuadriláteros la condición de que los ángulos definidos por las aristas estén comprendidos entre 80-100 grados sexagesimales.

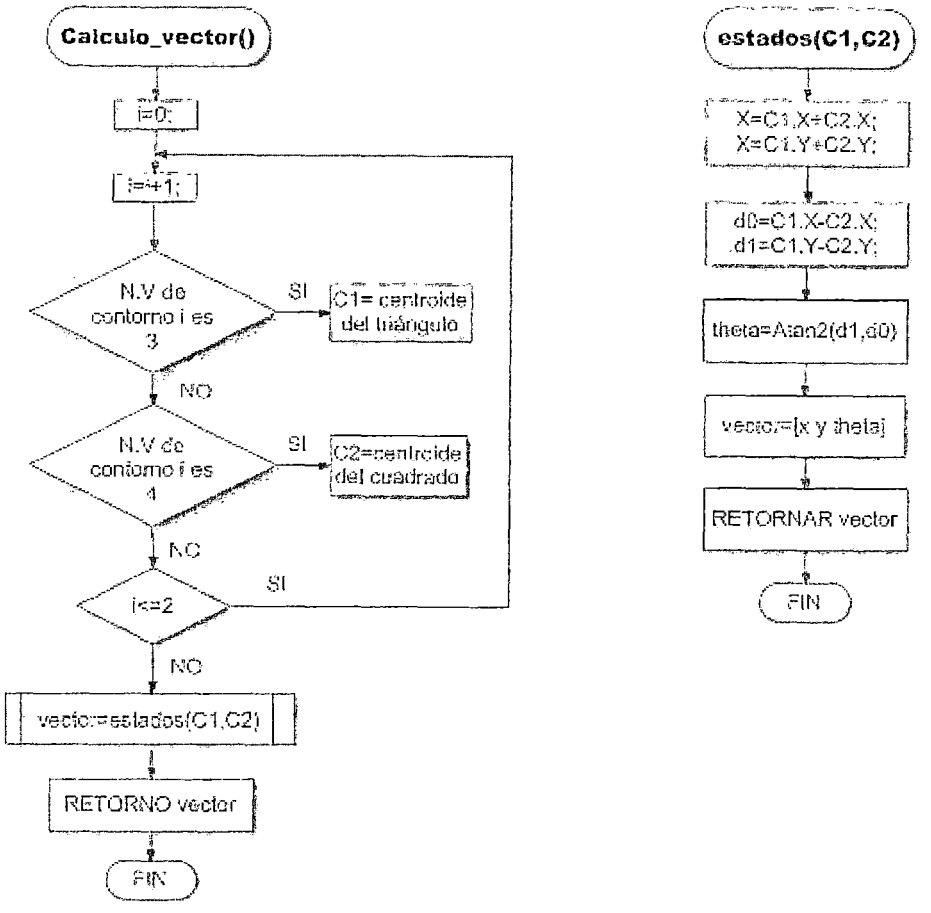

Figura 5.15: Rutina Calculo_vector perteneciente a ProcessFrame

En primer lugar se toma el frame de la cámara, el cual pasa a ser filtrado por 3 filtros de reconocimiento de los robots moway y 1 filtro para el entorno. Los filtros de reconocimiento de los Moway toman como entrada el frame de la cámara y entregan una imagen binaria, la cual se da como resultado tras filtrar un color en interés. En el presente caso son los colores rojo, verde y azul comprendidos en los siguientes rangos (ver tabla 5.3 y Figura 5.16) del sistema HSV (ver apéndice C):

	Robot 1 (Rojo)	Robot 2 (Verde)	Robot 3 (Azul)
H (Hue)	H<20;H>160	20<H<60	100<H<120
S (Saturation)	S>10	S>50	S>90

Tabla 5.3: Rango de colores en interés en el sistema HSV.

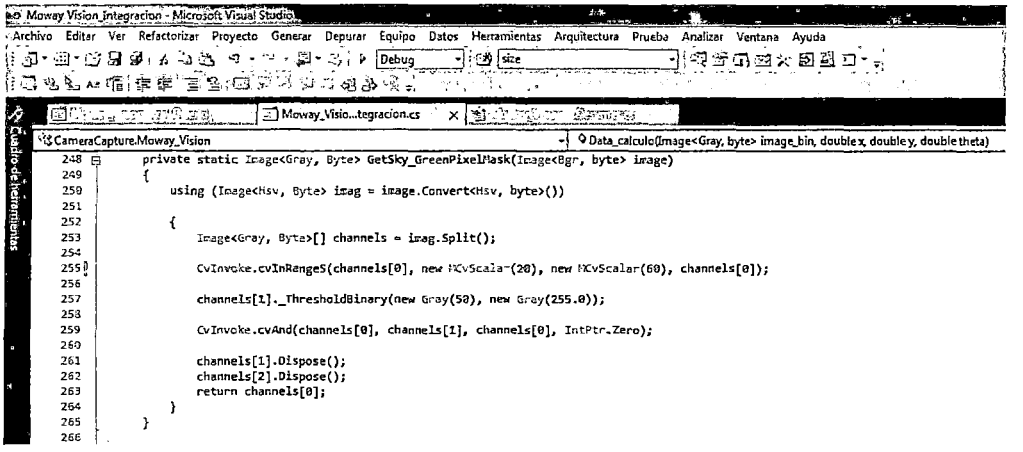

Figura 5.16: Filtro de obstáculos verdes, codificación en lenguaje C#.

Los filtros de colores se evalúan en el sistema HSV debido a que este sistema no es tan susceptible al brillo como en el caso del sistema de color RGB (ver Figura 5.17) . Esto queda corroborado con el histograma de dicha imagen (ver Figura 5.18), en dicho histograma se aprecia picos en la zona de luminosidad.

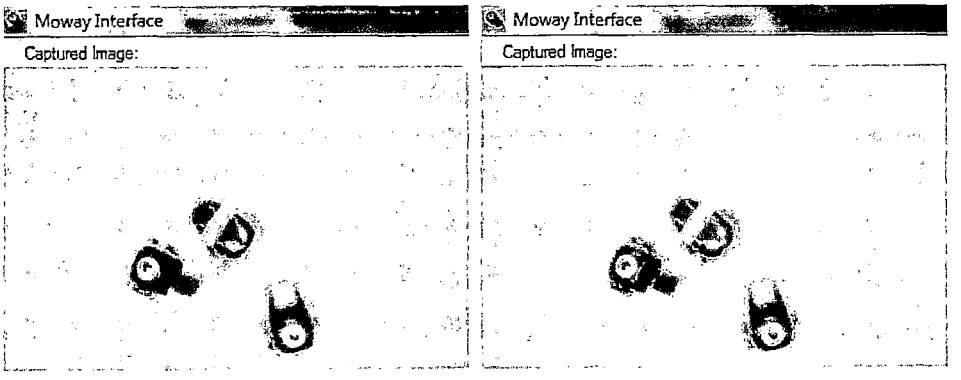

Figura 5.17: Captura del entorno por la interfaz.

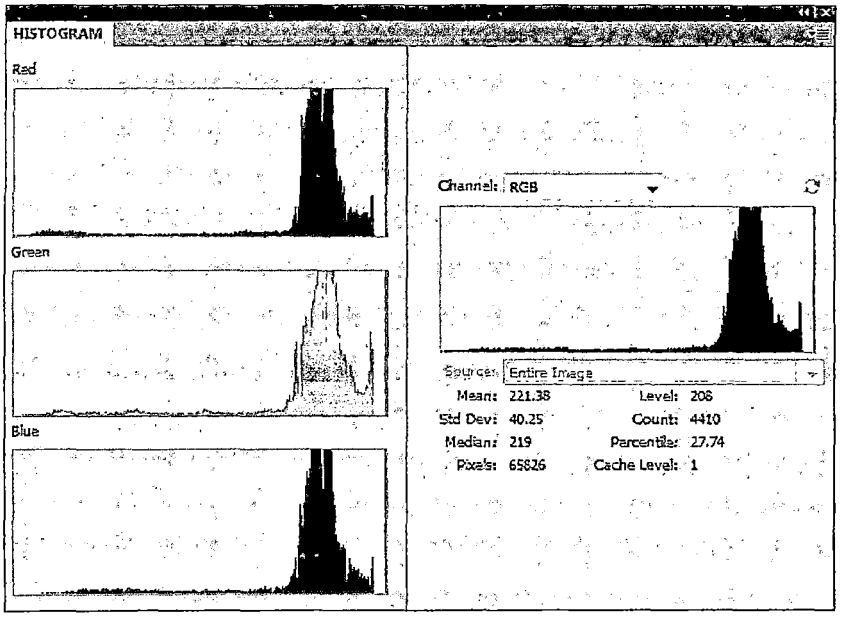

Figura 5.18: Histograma de la imagen capturada por la cámara

El procedimiento de este filtrado se describe como:

- Capturar la data de la cámara
- Convertir la imagen del Sistema RGB al sistema HSV
- Establecer los rangos en los canales H y S (ver apéndice D.2.4).
- Binarizar con los umbrales mencionados el color en interés

Si bien se aplican los filtros definidos por los rangos de los colores en interés del sistema HSV, los resultados presentan ciertas impurezas como podemos apreciar en la Figura 5.19.

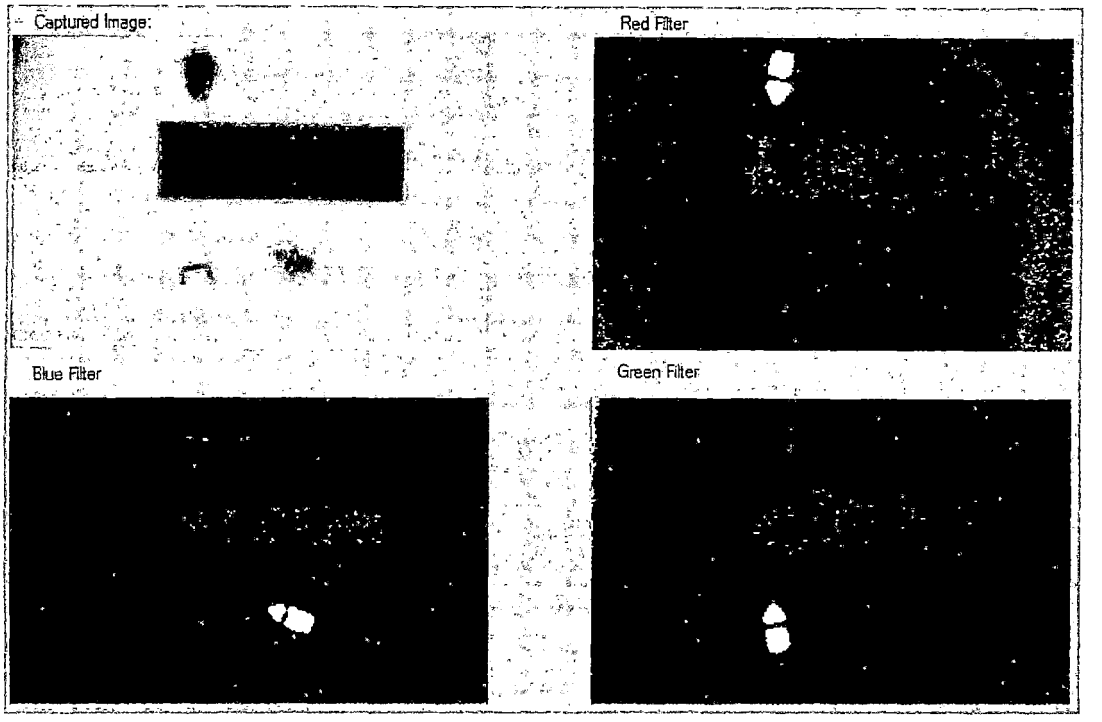

Figura 5.19: Filtrado de robots moway en sistema de color HSV, sin aplicar operaciones morfológicas.

Por ello se hace necesario utilizar algún otro criterio que en el presente caso se da mediante el uso de las operaciones morfológicas (ver apéndice C) sobre las imágenes binarias que resultan de aplicar el filtro de colores. Los resultados tras aplicar las operaciones morfológicas sobre las imágenes se pueden apreciar en las Figuras 5.20 y 5.21.

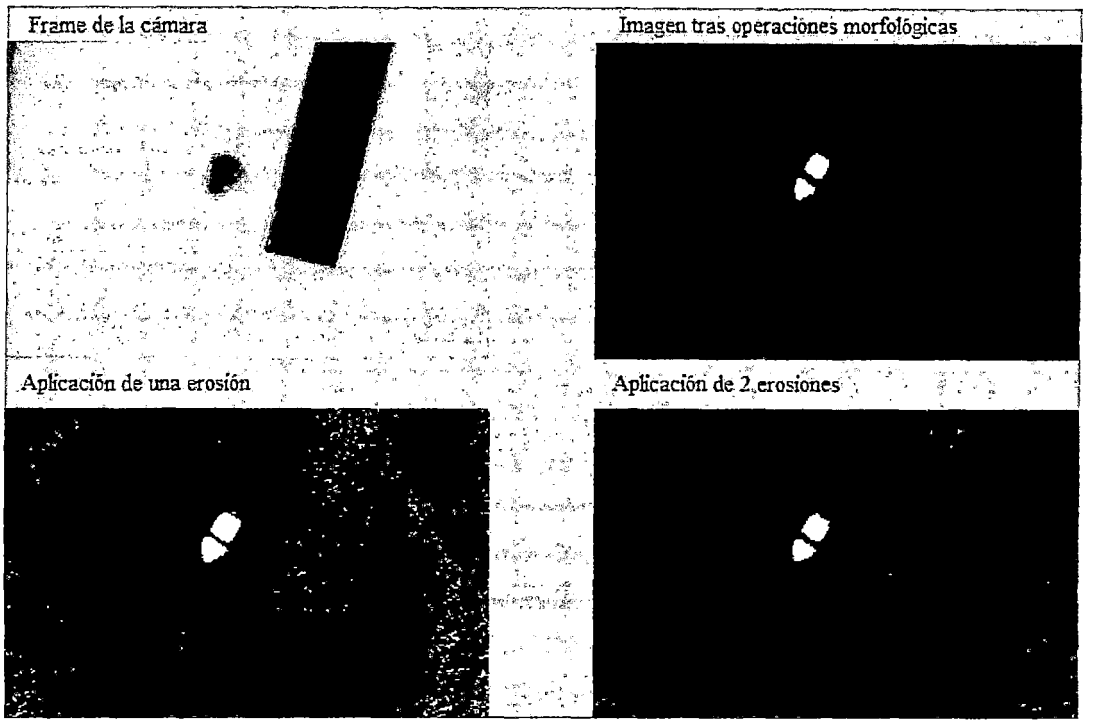

Figura 5.20: Filtrado de robot Moway rojo en sistema HSV, aplicando ciertas operaciones morfológicas.

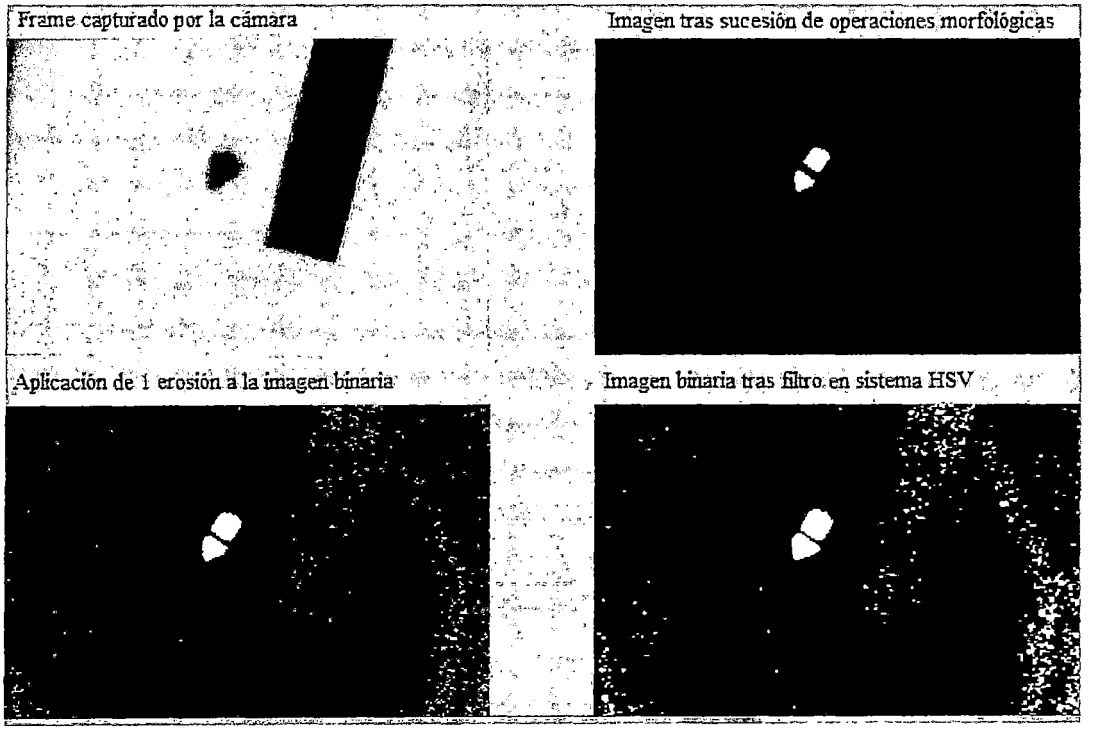

Figura 5.21: Uso de operaciones morfológicas en el procesamiento de imágenes para la obtención de posición y orientación del robot Moway.

Con las imágenes binarias sin impurezas se procede a calcular los estados (posición y orientación) de los moway, para ello el procedimiento es el siguiente:

- Capturar los contornos, si el número de estos es 2 se procede a calcular el número de triángulos y cuadrados presentes en la imagen binaria, para ello se considera que dichos objetos deben presentar un área mayor a 250 píxeles.
- Si el número de triángulos y cuadrados es 1 se procede a calcular los centroides de dichos objetos y posteriormente la posición y orientación del robot móvil.

A. Entorno del Robot El área de trabajo considerada para esta aplicación es de 1m de largo por 0.75m de ancho (ver Figura 5.22), siendo la superficie de color blanca. Los obstáculos a considerar son de color negro, por ende es necesario filtrar ese color.

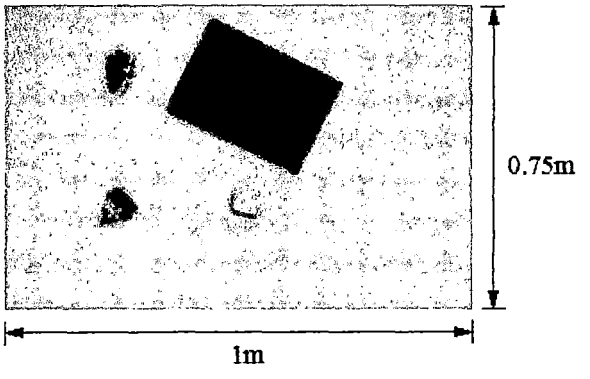

Figura 5.22: Dimensiones del área de trabajo.

B. Posición y Orientación Moway Ambos estados se calculan en base a los centros del triángulo y cuadrado denotados por C1 y C2 respectivamente (ver Figura 5.23). La orientación del robot móvil se halla en base a un vector definido por C1 y C2; en cuanto a la posición se halla con la semisuma de C1 y C2. Ello

se ejecuta en la subrutina **estados** (ver Figura 5.15) perteneciente a la función ProcessFrame (ver Figura 5.14).

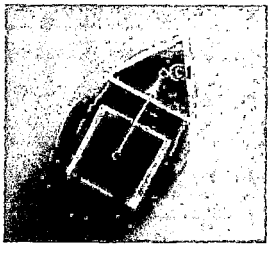

Figura 5.23: Orientación del Robot Moway.

Para el cálculo de los centroides del triángulo y cuadrado se tomará a partir de la imagen binaria obtenida tras el filtro de color en el sistema HSV y tras la aplicación de las operaciones morfológicas, luego se procede a calcular los estados mediante la función **Calcular_estados** (ver Figura 5.14 y Figura 5.15) cuyo argumento es una imagen binaria. Esta función captura los contornos presentes en la imagen binaria. Si el número de dichos contornos es mayor a 2, los estados se actualizarían con valores erróneos; caso contrario, cuando se detectan 2 contornos se procede a la actualización de los estados. La actualización de los estados se calcula en base a los 2 contornos detectados, realizándose una detección del número de triángulos y cuadrados. Si el número de ambos es 1 se procede al cálculo de los centroides, siendo los centroides los que permiten el cálculo de la posición y orientación del robot moway (ver Figura 5.15).

CONCLUSIONES

- La naturaleza estocástica del algoritmo RRT permite obtener varias rutas para un mismo problema, las cuales no son óptimas pero pueden ser usadas como rutas de referencia y podrían optimizarse mediante el uso de otras técnicas.

- En las Figuras 4.14, 4.15 y 4.16 se puede apreciar que la rutas generadas para el sistema son curvas suaves con respecto a las mostradas en las Figuras 4.3,4.5 y 4.8 (en las cuales no se consideraba el modelamiento del sistema). El algoritmo RRT tiene como ventaja el poder extenderse a sistemas del tipo no holónomo, que permite resolver cuestiones propias que caracterizan al sistema.

- Se ha construido una plataforma e interfaz, las mismas que nos han servido para validar los algoritmos tal como lo demuestran las simulaciones experimentales. En dicha interfaz se han monitoreado en línea los estados de posición y orientación del móvil lo que ha dado mayor robustez al sistema. El tener dicha plataforma permite probar una gran variedad de algoritmos. Ya que la implementación ha sido desarrollada, ha sido posible concentrarse en el desarrollo de los mismos.

- Se ha modificado el software del fabricante MiniRobots debido a que no se podía trasladar flujo de data, así como tampoco se podía manipular varios móviles al mismo tiempo, dicha modificación se realizó gracias a la información brindada de sus dlls.

- La optimización computacional de los algoritmos es un factor importante ya que permite acelerar el procesamiento de un algoritmo, esto queda demostrado con el uso del algoritmo de *KNN search*. En la simulación respectiva (ver Figura 4.12) se puede corroborar que utilizando este algoritmo la eficiencia (ver Tabla 4.1)de un algoritmo básico mejora notablemente.

- La factibilidad de aplicación de estos algoritmos se demuestra por el éxito de las implementaciones físicas realizadas mediante el sistema de visión artificial. Sin embargo, sigue siendo necesario mejorar el color de filtrado, como por ejemplo que sea autoregulable o el cambio de sensores de retroalimentación más sofisticados, tomando ventaja de la detección distribuida, especialmente en aquellos casos en que una vista completa de la vista de planta del espacio de trabajo con una cámara no es factible ni práctico. Sin embargo, para esta particular implementación, era plausible y adecuada.

- Por último, con respecto a la posibilidad de algún interesada desee realizar una aplicación comercial de la interfaz realizada, ante todo se debe pagar el concepto de Copyright de ciertos productos utilizados. Es decir, tanto a la compañía MiniRobots como a Emgu, ya que se ha utilizado recursos de software de ambos que son libres de manera académica mas no de manera comercial.

BIBLIOGRAFÍA

[1] BIZINTEK INNOVA S.L. *Manual Módulo BZI-RF2GH4*, Febrero 2007.

[2] Mark W. Greenia. *History of Computing: An Encyclopedia of the People and Machines that Made Computer History*. Lexikon Services Publishing, March 2001.

[3] Jean-Claude Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. In *International Journal of Robotics Research*, volume 18, pages 1119–1128, Stanford, CA 94305, USA, July 1999. Stanford University.

[4] Jean-Claude Latombe. *Robot Motion Planning*. B43X-XYZ-UEGU. Kluwer Academic Publishers, Stanford University, 1991.

[5] Steven M. LaValle. *PLANNING ALGORITHMS*. Cambridge University Press, 2006.

[6] David Lammers. The era of error-tolerant computing, November 2010.

[7] Seth Hutchinson George Kantor Wolfram Burgard Lydia Kavraki Sebastian Thrun Howie Choset, Kevin Lynch. *Principles of Robot Motion Theory, Algorithms, and Implementation*. The MIT Press, 2005.

[8] Sean Quinlan. *Real Time Modification of Collision-Free Paths*. PhD thesis, Stanford University, December 1994.

[9] José Mireles Jr. Kinematics of mobile robots. Technical report, University of Texas Arlington, 2004.

[10] Jean-Claude Latombe Lydia E. Kavraki, Petr Svestka and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE Transactions on Robotics and Automation*, pages 566–580. Robotics Laboratory, Department of Computer Science, Standford University and Department of Computer Science, Utrecht University., 1996.

[11] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Department of Computer Science; Iowa State University, 1998.

[12] Canming Huang. www.emgu.com.

[13] Moway. *MOWAY USER MANUAL*, June 2010.

[14] Moway. *Manual librería Base*, Abril 2008.

[15] Gary Bradski and Adrian Kaehler Beijing. *Learning OpenCV*. O'Reilly Media, first edition, September 2008.

[16] Toby Breckon Chris Solomon. *Fundamentals of Digital Image Processing*. Wiley-Blackwell, 2011.

APÉNDICE A

Antecedentes históricos

A.1. Reseña histórica de robots móviles

En éste apéndice se aboradará a modo suscinto parte de la historia referida a la evolución de la robótica móvil desde 1948. Cabe recalcar que no se detallan todos los avances durante este periodo a tratar debido a la cantidad y varieda de trabajos que se han desarrollados. Por tal motivo lo explicado continuación es solo una breve reseña que nos puede dar un panorama de lo que concierne a la evolución de la robótica móvil:

Como se había mencionado en el Capítulo 1, antes del robot Shakey se habían desarrollado ciertos avances en robótica móvil, como por ejemplo en la detonación dentro de cierto rango de bombas con el uso de sistemas de guiado y control radar durante la Segunda Guerra Mundial (ver Figura A.1).

Figura A.1: Soldados británicos junto a los vehículos alemanes Goliath (tras la batalla de Normandia, en 1944). Estos móviles eran vehículos de demolición controlados remotamente.

Elmer y Elsie (ver Figura A.2), los primeros robots autónomos electrónicos fueron creados por Willian Grey Walter en Bristol (Inglaterra) en 1948 y 1949. Dichos robots considerados turtle (por su caparazón similar al de una tortuga) que exploraban su entorno utilizando un sensor de luz para moverse en dirección a la fuente de luz que encontrasen, contaban con sensores de colisión. Estos robots demostraron que el comportamiento complejo puede surgir a partir de un diseño simple, como en el caso de la evasión de obstáculos en su camino (ver Figura A.3).

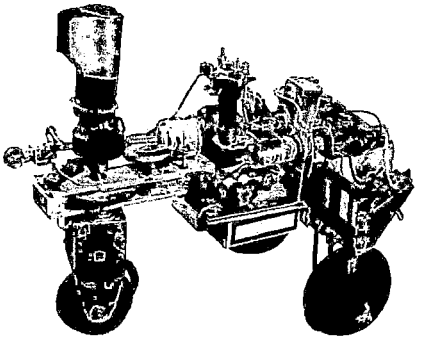

Figura A.2: Robot Elsie(1948) sin caparazón.

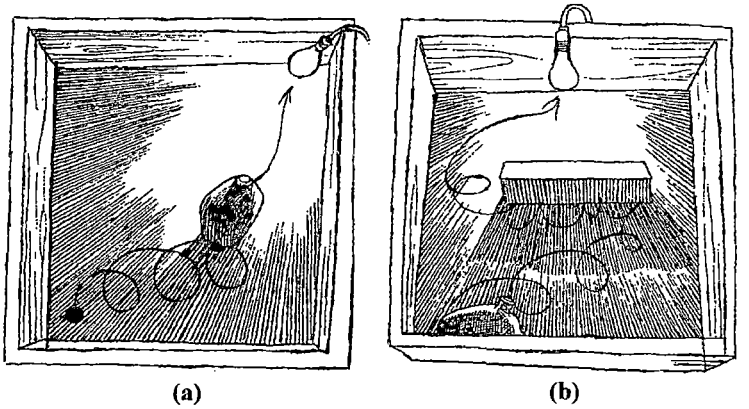

Figura A.3: (a)La "tortuga de Grey", que el denominaba "tortoise" buscando una luz (b) Reacción de "tortoise" frente a un pequeño obstáculo.

Sin embargo Unimate (ver Figura A.4) fue realmente el primer robot industrial, teleoperado digitalmente, además de ser programable y académico. Este robot fue inventado en 1954 por George Devol quien después cofundaría la primera compañía de robótica en el mundo.

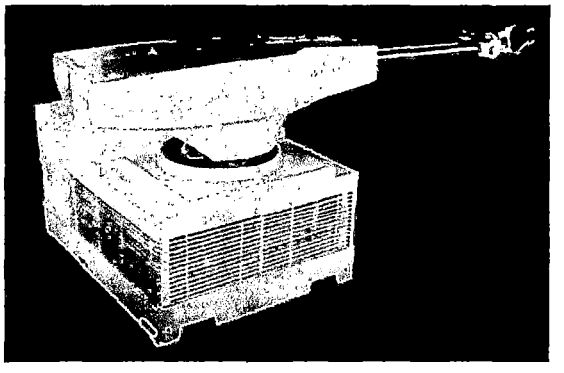

Figura A.4: Robot Unimate, presentado en 1961.

La Universidad Johns Hopkins entre 1961 y 1963 desarrolla el robot Beast (ver Figura A.5), que utiliza un sonar para moverse. Cuando sus baterías se descargaban buscaba una fuente de energía, dicha búsqueda consistía en vagar por los pasillos blancos del laboratorio hasta encontrar un bloque color negro donde se conectaba y recargaba.

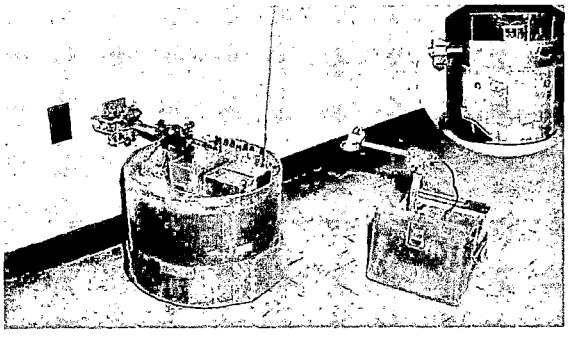

Figura A.5: De izquierda a derecha: Robot Beast junto al Robot Ferdinand (1961-1963).

En 1970 Stanford desarrollo Stanford's Cart (ver Figura A.6), un carrito que sigue líneas de color blanco con una cámara y cuyo procesamiento lo realizaba una computadora *mainframe*[1] de la época, posteriormente 10 años después este vehículo atraviesa obstáculos y hace mapas de su entorno.

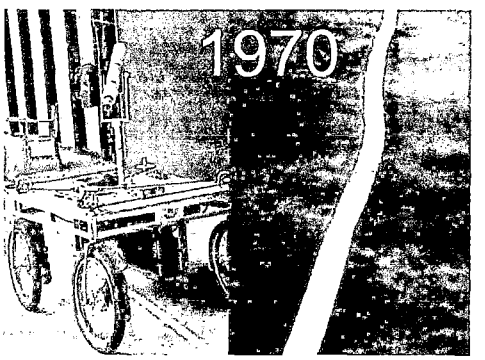

Figura A.6: Robot móvil Stanford's Cart (1970).

La Unión Soviética en 1970 envía el primer robot explorador espacial, el Lunokhod 1 (ver Figura A.7). A su vez la NASA en 1976 envia a Marte 2 naves espaciales no tripuladas en su programa Viking.

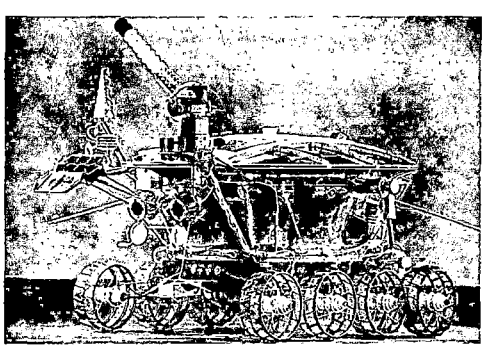

Figura A.7: Robot Lunokhod 1 en 1970.

[1]Computadora poderosa y costosa utilizada principalmente por cooporaciones u organos guvernamentales para procesar gran cantidad de datos o soportar gran cantidad de usuarios.

Ya a partir de 1980 el interés del público en los robots se eleva, dando lugar a los robots que se podían comprar para uso doméstico. Estos robots sirven de entretenimiento o educativos. Los ejemplos incluyen el RB5X (ver Figura A.8), que todavía existe hoy y la serie HERO.

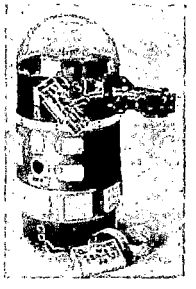

Figura A.8: Robot RB5X en la década de los 80.

En los inicios de los 80, el equipo de Ernst Dickmanns en la Univesidad Bundeswehr de Munich fabrica los primeros automóviles robóticos (en si utilizó automóviles Mercedenz Benz a los cuales equipo con sensores), logrando conducir hasta 55 millas por hora en las calles vacías. Posteriormente en 1995 se logra realizar un tramo desde Munich a Copenhagen, alcanzado velocidades de 120 millas por hora y ejecutando ocasionalmente maniobras para pasar a otros vehículos en situaciones de tráfico. **Visión Activa**[2] (ver Figura A.9) fue utilizada como herramienta para hacer frente a la rápida evolución de escenas de la calle.

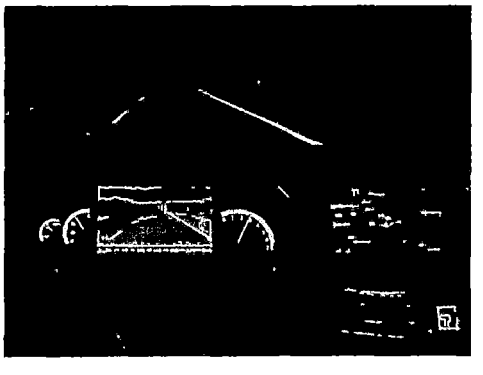

Figura A.9: Sistema de Visión Activa Nocturna implementado en un Mercedes-Benz en Estados Unidos.

[2]Área de Visión Computacional, con ésta técnica se puede manipular el punto de vista de cámara con el objetivo de investigar el ambiente y conseguir mejor información del mismo.

En 1991, Edo. Franzi, André Guignard y Mondada Francesco desarrollaron Khepera (ver Figura A.10.a), un pequeño robot autónomo móvil, dedicado a actividades de investigación, siendo hoy en día muy utilizado para el desarrollo de técnicas de navegación autónoma. El proyecto fue financiado por el laboratorio de LAMI-EPFL. Posteriormente, en 1995, el robot móvil programable Pioneer (ver Figura A.10.b) se hace comercialmente disponible para el mercado a un precio asequible, lo que permite un aumento generalizado en la robótica de investigación y estudio en la universidad durante la siguiente década de tal manera que la robótica móvil se convierte en una parte estándar del currículum universitario.

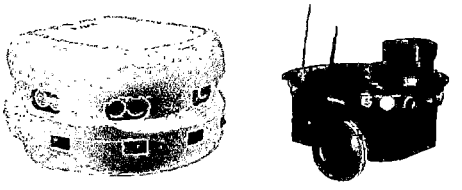

Figura A.10: (a)Robot autónomo móvil Khepera. (b) Robot móvil Pioneer.

Entre 1996 1997, la NASA envía al robot explorador espacial Sojourner (ver Figura A.11) a Marte. El robot explora la superficie, comandado desde la tierra. Sojourner estaba equipado con un sistema de prevención de riesgos. Esto permitió a Sojourner encontrar de forma autónoma su camino a través del desconocido terreno marciano.

Figura A.11: Robot exploradpor espacial Sokourner en el planeta Marte.

A inicios del 2001 se inicia el proyecto Swarm-bots son similares a las colonias de insectos. Por lo general comprendidos con un alto número de robots individuales simples, que pueden interactuar entre sí y juntos realizar tareas complejas. En el 2010 se estableció la competencia *Multi Autonomous Ground-robotic International Challenge (MAGIC*, ver Figura A.12*)* cuyo objetivo es la creación de equipos robóticos múltiples vehículo que puede ejecuten una misión de inteligencia, vigilancia y reconocimiento en un entorno urbano dinámico.

Figura A.12: Swarm-bots en la competencia MAGIC.

En el 2004 y 2005 se desarrollola competición del DARPA Grand Challenge (ver Figura A.13.a), vehículos totalmente autónomos competir unos contra otros en un campo desierto. Posteriormente en el 2007 se realizó el DARPA Urban Challenge (ver Figura A.13.b), que se enfoco en que los vehículos puedan ser capaces de obedecer las leyes de tránsito, mientras que detectan y evitan a otros robots en el camino.

(a) **(b)**

Figura A.13: (a)Vehículo en el DARPA Grand Challenge en el 2004 (b) Vehículo en el DARPA Urban Challenge en el 2007 .

En el 2008, Boston Dynamics publicó un video de una nueva generación de BigDog (ver Figura A.14) capaz de caminar sobre terrenos irregulares tales como pedregosos, fangosos, cubiertos de nieve e incluso poder recuperar su equilibrio cuando fue desestabilizado desde el costado por una persona.

Figura A.14: Robot Bigdog en la nieve (2008).

Se han nombrado algunos de la larga lista de desarrollos en robótica móvil principalmente, sin embargo es necesario recalcar la influencia hoy en día de los *Smartphone* (ver Figura A.15) ya que por las prestaciones y recursos (que tienen a la fecha y que seguirán desarrollando y mejorando con el tiempo con una disminución paulatina de su costo) están siendo tomados como minicomputadoras portables.

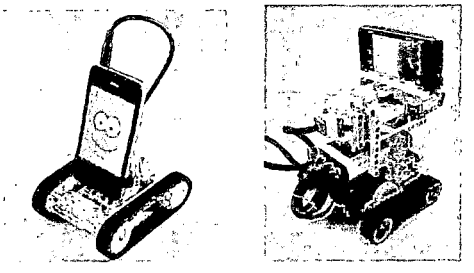

Figura A.15: Robots utilizando *smartphone* como procesador.

APÉNDICE B

Robot Moway

El robot Moway (ver Figura B.1, tomada de [13]) es un robot programable diseñado principalmente para desarrollar aplicaciones prácticas de robótica móvil. Está equipado con una serie de sensores que le permiten moverse en un entorno real, así como también puede comunicarse inalámbricamente (mediante tecnología de radiofrecuencia) con la PC. Sus dimensiones son aproximadamente de 6cm x 4cm y con 3cm de alto, posee 2 ruedas motrices y una esfera metálica que se utiliza como rueda loca para el robot móvil.

Figura B.1: Robots Moway.

B.1. Arquitectura

El robot Moway esta conformado por los siguientes elementos (ver Figura B.2, tomada de [13]):

- Procesador
- Sistema de navegación
- Sensores en indicadores de grupos
- Sistema de potencia
- Conector de expansión

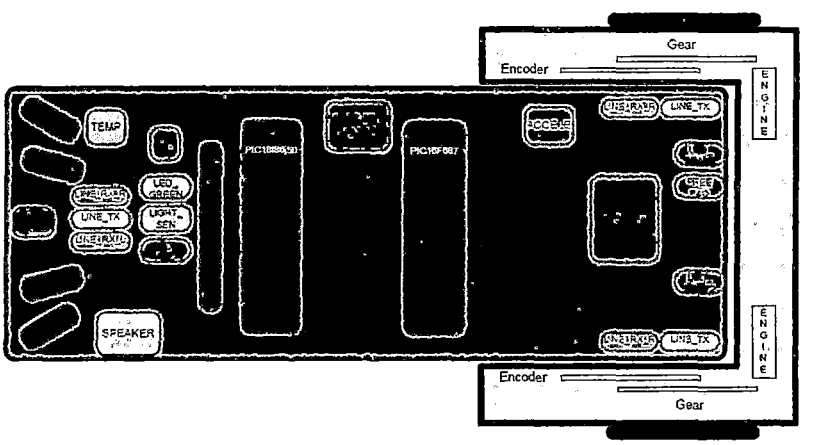

Figura B.2: Partes del robot Moway.

B.1.1. Procesadores

Los Moways son gobernados por el microcontrolador PIC18F87J50 de 4MHz, este procesador se encarga tanto de las comunicaciones, de procesar la data recibida por los sensores y el que gobierna el sistema de navegación del móvil.

B.1.2. Sistema de navegación

Esta dado por el PIC16F887 que mediante un puente H varia la velocidad de las ruedas a través de los motores DC que posee el robot Moway (ver Figura B.3, tomada de [13]). Este microcontrolador es el esclavo del microcontrolador principal PIC18F87J50.

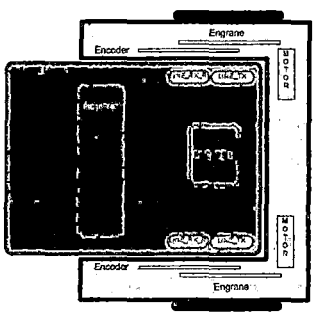

Figura B.3: Drive System: electrónica y mecánica.

B.1.3. Sensores e indicadores

Permiten al Moway relacionarse con su entorno y dicha información se puede enviar inalámbricamente a la PC (ver Figura B.4, tomada de [13]).

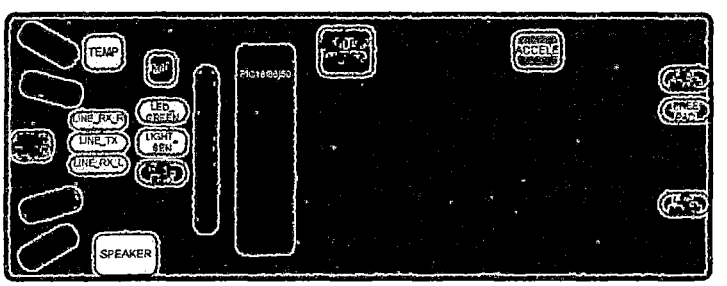

Figura B.4: Sensores y grupo de indicadores.

B.1.4. Sistema de potencia

El robot tiene una bateria LiPo recargable a través del puerto USB del Moway desde la computadora (ver Figura B.5, tomada de [13]).

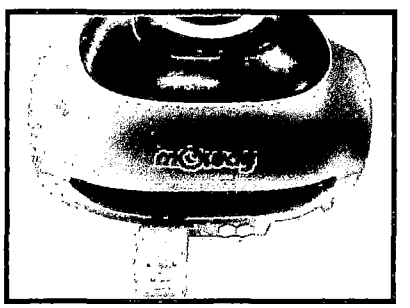

Figura B.5: Fuente de sistema de alimentación.

B.1.5. Conector de expansión

Es un módulo RF (ver Figura B.6, tomada de [13]) que permite interacción entre el Moway con la PC a través de un RF-USB (ver Figura B.7, tomada de [13]), la cual es en sí un modulo de RF con una extensión de RS232 a USB.

Figura B.6: Módulo RF, que se conecta al robot Moway.

Figura B.7: RF-USB, módulo que se conecta al puerto USB de la computadora.

B.2. Moway GUI

Es una interfaz (ver Figura B.8) de interacción con el robot, pudiendo programarlo en asembler, C18 o mediante bloques en un diagrama de flujo; debiéndose referenciar las librerías del robot moway para accesar tanto a la data como periféricos de robot.

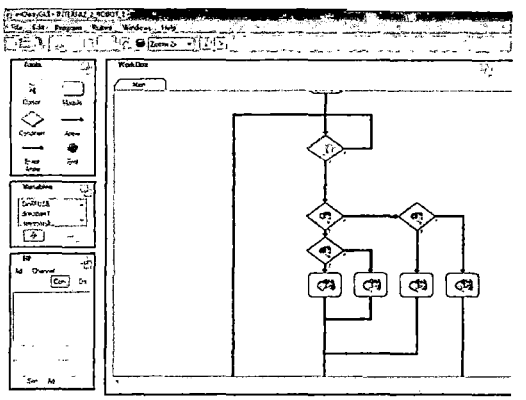

Figura B.8: Interfaz gráfica del robot Moway.

Moway GUI tiene una sección para realizar pruebas con la radiofrecuencia BZI-RF2GH4[1], la cual permite la comunicación inalámbrica entre el robot Moway y la PC.

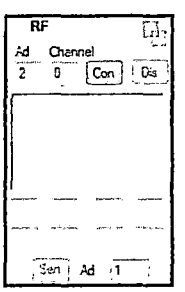

Figura B.9: Sección de Control de RF en la Interfaz gráfica Moway GUI.

Pero esta sección (ver Figura B.9) permite mandar data en un paquete conformado por 8 bytes pero que tienen que ser seteados manualmente en la interfaz de Moway GUI, por esta razón solo se utilizará la aplicación Moway GUI para

programar a los Moway y realizar pequeñas pruebas de comunicación pero no será la interfaz de la presente tesis.

B.3. Interfaz C#

Es un proyecto ejemplo de Moway escrito en C#(ver Figura B.10), el cual posee las funcionalidades de Moway GUI debido a que incluye las 3 dlls que permiten controlan al robot:

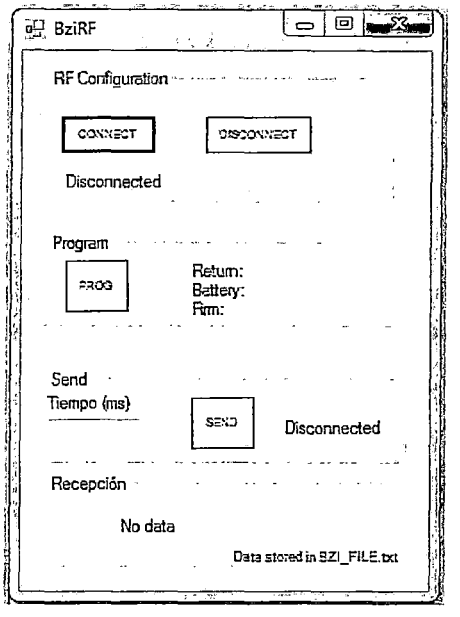

Figura B.10: Interface Moway en C#

- lib_nrf24lu1_RF, desarrolla el protocolo de comunicación inalámbrico que utilizan los robots Moway, es decir define por ejemplo el tiempo de confirmación del envió de data así como el empaquetamiento de la información y cabeceras de la data a enviar.

- lib_prog_mow2, permite grabar sobre el robot moway.

- LibUsbDotNet, es una librería USB escrita en .NET C# para drivers WinUsb, libusb-win32 y Linux libusb v1.x. Todas las funcionalidades básicas del dispositivo USB lo que le permiten interactuar con el sistema operativo.

Esta interfaz es la base de la que se ha desarrollado en la presente tesis, la cual agrega visión computacional y el cálculo de comandos de movimiento para que el robot Moway siga la ruta calculada.

En este documento no se pondrá el código de esta interfaz por lo referente a derechos de autor pero esto no limita a la persona interesada a contactarse con la compañía MiniRobots para poder obtener la interfaz. Para entender las funciones de la interfaz del fabricante, se recomienda revisar [14].

APÉNDICE C

Visión Artificial

Visión Artificial es la transformación de data desde una cámara fotográfica o de video en algún otro tipo de representación para el logro de algún objetivo en particular. El ser humano realiza dicha transformación dividiendo la señal de visión en varios canales que proveen diferentes tipos de información al cerebro. El cerebro tiene un sistema de atención que identifica, en una tarea de modo dependiente, partes importantes de una imagen a examinar mientras sorpresivamente examina otras áreas. Hay una masiva retroalimentación en el flujo visual que es aún no entendido por completo, así como también hay una amplia asociación de entradas de varios sentidos que permiten al cerebro bosquejar situaciones aprendidas por la experiencia. En los sistemas de visión artificial, sin embargo, una computadora recibe una matriz de números (por ejemplo lo mostrado en la Figura C.1, tomado de [15]) desde la cámara donde por defecto no hay un reconocimiento de patrones, no hay un control automático de focus y apertura, no hay asociaciones cruzadas con los años de experiencia. Es por tal razón que a partir de la información se aplican diversos algoritmos para lograr el objetivo en particular. En este caso el objetivo será encontrar la posición y orientación del robot Moway en su entorno.

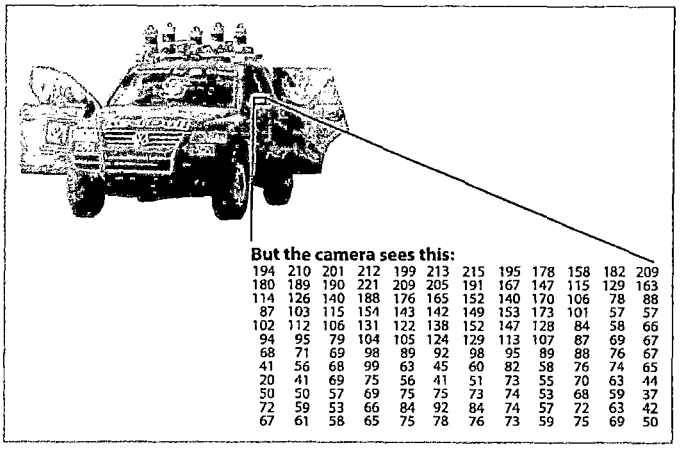

Figura C.1: Para la computadora, lo que capta la cámara es una matriz de números.

C.1. Data de la imagen

Lo captado por la cámara de video es una secuencia de *frames* (fotogramas). Cada frame es una matriz, o un array bidimensional de números siendo cada celda un píxel[1]. Un píxel contiene la información de la combinación de cada uno de los canales del sistema de colores que se utiliza, por ejemplo en el sistema RGB cada canal toma valores entre el rango de 0 a 255.

C.1.1. Representación de imágenes digitales

1. Imagen Binaria, 1 píxel es representado por 1 bit: 1 ó 0.

2. Imagen en escala de grises, 1 píxel es representado por 1 byte: 0-255.

3. Imagen en color, por ejemplo en el sistema RGB cada píxel consta de 3 valores: Rojo, Verde, Azul. Con un byte para cada canal se puede obtener 16,7 millones posibles colores.

Los 3 representaciones mencionadas se aprecian en la Figura C.2.

[1]La palabra píxel es una abreviación de 'picture element', es la unidad básica de información en una imagen.

Figura C.2: De izquierda a derecha: a colores en RGB, en escala de grises e imagen binaria, con un *threshold* (umbral) 28).

C.2. Histograma de una imagen

Un histograma es un gráfico que muestra la distribución de los valores de intensidad de cada píxel en una imagen, donde la abcisa horizontal indica la intensidad y la abcisa vertical indica la cantidad de píxeles con dicha intensidad. La intensidad podría expresarse en escala de grises teniendo 256 elementos cuyos valores van desde 0 a 255 ó también podría normalizarse dicha frecuencia, pudiéndose tratar el histograma como una función de densidad de la probabilidad que define la vecindad del valor de un píxel dentro de la imagen. Una inspección visual del histograma permite revelar los constraste básicos presentes en la imagen. En el ámbito de la fotografía se le utiliza como herramienta para mejorar la exposición de las fotos, corrigiendo los colores con técnicas de ecualización.

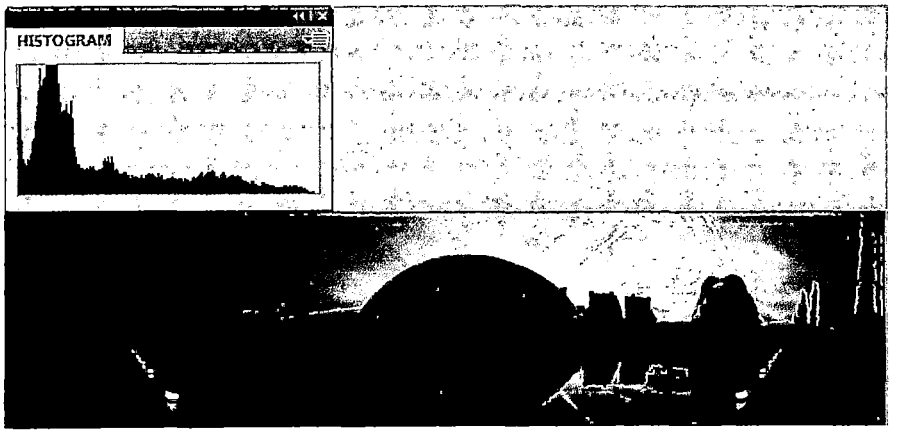

Figura C.3: Imagen sub-expuesta.

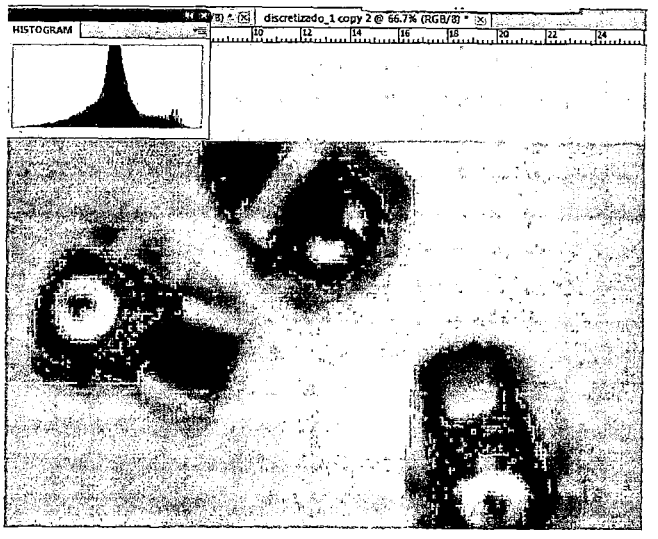

Figura C.4: Imagen propiamente expuesta.

Figura C.5: Imagen sobre-expuesta.

C.3. Sistema HSV

El modelo HSV (conocido también como HSB) fue creado en 1978 por Alvy Ray Smith. Se trata de una transformación no lineal del espacio de color RGB, y se puede usar en progresiones de color. El formato HSV no es muy sensible al cambio de iluminación, es decir nos conviene en ambientes donde la iluminación es muy inestable (ver Figura C.6, tomada de [16]).

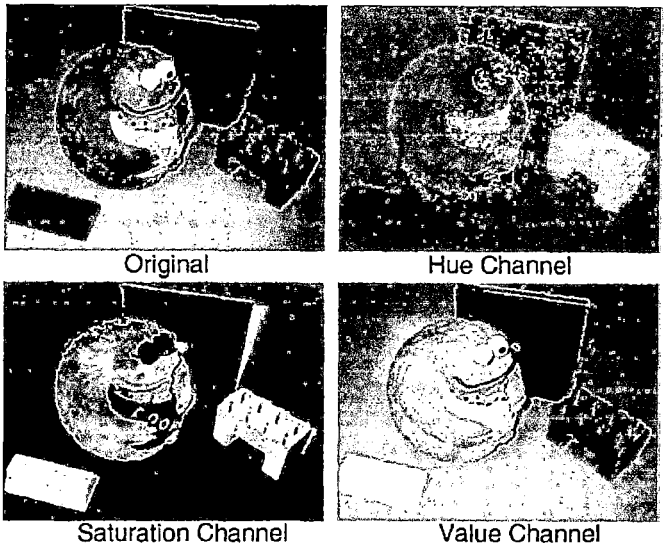

Figura C.6: Imagen transformada y mostrada en el espacio de color HSV.

Los 3 canales de este formato de color son:

- **H:** Hue (Tonalidad), se representa como un grado de ángulo cuyos valores posibles van de 0 a 360° (aunque para algunas aplicaciones se normalizan del 0 al 100 %). Cada valor corresponde a un color. Ejemplos: 0 es rojo, 60 es amarillo y 120 es verde.

- **S:** Saturation (Saturación), se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van del 0 al 100 %. A este parámetro también se le suele llamar "pureza" por la analogía con la pureza de excitación y la pureza colorimétrica de la colorimetría. Cuanto menor sea la saturación de un color, mayor tonalidad grisácea habrá y más decolorado estará. Por eso es útil definir la insaturación como la inversa cualitativa de la saturación.

- **V:** Value (Valor), el brillo del color. Representa la altura en el eje blanco-negro. Los valores posibles van del 0 al 100 %. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado.

- **B**: Brightness (Brillo).

Algunas formas gráficas de la representación de este sistema (ver Figuras C.7 y C.8):

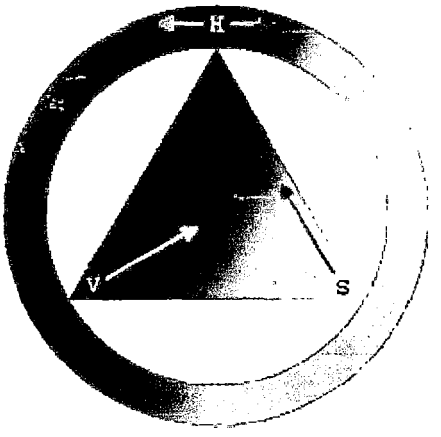

Figura C.7: Espacio de color HSV como una rueda de color.

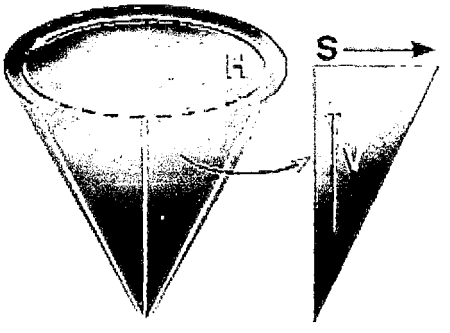

Figura C.8: Cono de colores del espacio HSV.

En EMGU, los rangos de valores de los canales del modelo HSV son:

- H:0-255
- S:0-255
- V:0-255

C.4. Operaciones morfológicas en imágenes

La palabra morfología se refiere al estudio de la forma o estructura de un objeto. En el procesamiento de imágenes se utiliza la morfología matemática para la identificación y extracción de descriptores significativos de las imágenes; tales como contornos, esqueletos, entre otros. Es decir se simplifican las imágenes y se conservan las principales características de forma de los objetos.

La morfología en imágenes abarca un poderoso e importante grupo de métodos que pueden ser tratados matemáticamente dentro del marco de la teoría de conjuntos, la teoría de retículos, la topología y las funciones aleatorias. Conceptos topológicos y geométricos de espacio continuo, tales como tamaño, forma, convexidad, conectividad y distancia geodésica, se pueden caracterizar por la morfología matemática en espacios continuos y discretos.

Las operaciones morfológicas pueden ser aplicadas a imágenes de todo tipo, pero su uso es extendido principalmente en imágenes binarias siendo las operaciones clave la dilatación y erosión ya que muchos procedimientos morfológicos sofisticados pueden ser reducidos a una secuencia de dilataciones y erosiones.

El resultado de una operación morfológica depende de 3 cosas: la imagen de entrada, el tipo de operación y el elemento estructural. El elemento estructural (ver Figura C.9, tomada de [16]) es quien define la forma y el tamaño de la vecindad del píxel que será analizado para posteriormente alterar su valor. Su tamaño es muy inferior al tamaño de la matriz original que define el conjunto a modificar. Esta compuesto de unos y ceros en imagenes binarias de forma y tamaño arbitrario en la cual las posiciones donde está el uno define la vecindad.

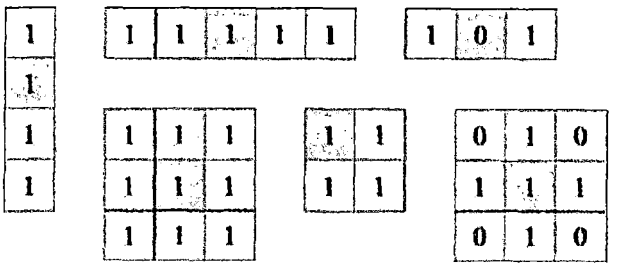

Figura C.9: Algunos ejemplos de elementos morfológicos estructurales. El píxel centro de cada elemento estructural esta sombreado.

C.4.1. Dilatación

$$\delta_C(A) = A \oplus C = \{x|(\hat{C})_x \subset A \neq 0\}$$

Se obtiene en base a la reflexión de C con respecto a su origen y un desplazamiento x.

La salida de la dilatación es el conjunto de puntos barridos por el centro del elemento estructural mientras algún punto de C coincide con alguno de A. Alternativamente (véase Figuras C.10 y C.11), puede interpretarse la dilatación como el resultado de reemplazar cada píxel blanco de la imagen original por una réplica del elemento estructural. La dilatación añade todos los puntos del fondo que tocan el borde de un objeto, es decir, es extensiva.

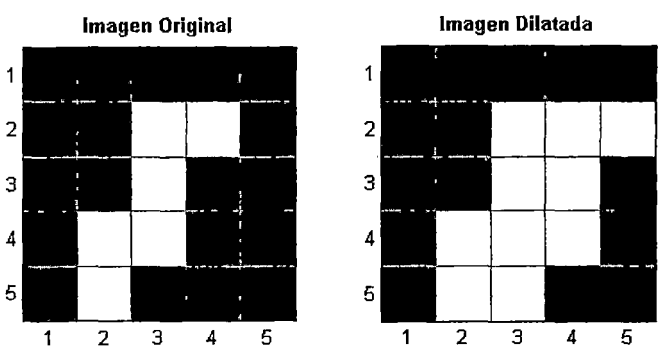

Figura C.10: Aplicación de dilatación.

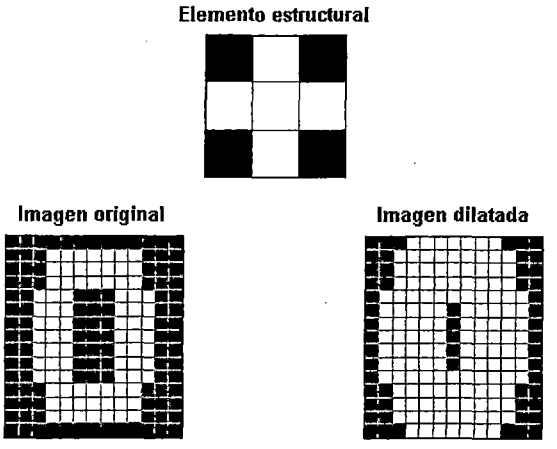

Figura C.11: Otra aplicación de dilatación.

C.4.2. Erosión

$$\varepsilon_C(A) = A \ominus C = \{x|(C_x \subset A\}$$

La salida de la erosión es el conjunto de puntos barridos por el centro del elemento estructural mientras se cumpla que todos los puntos de C estaban contenidos en A. Elimina grupos de píxeles donde el elemento estructural no está contenido (ver Figuras C.12 y C.13). La erosión reduce reduce el tamaño del objeto, es decir, es antiextensiva. La aplicación más común de la erosión es la eliminación de detalles irrelevantes.

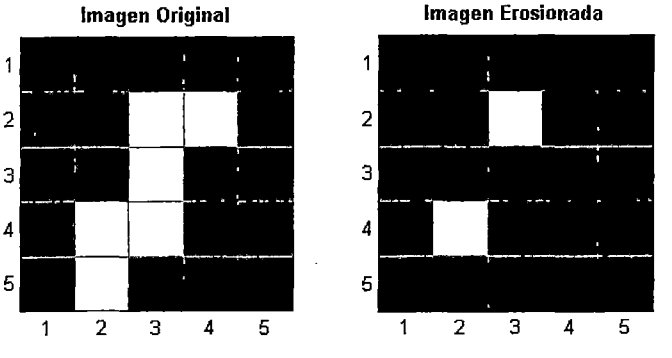

Figura C.12: Aplicación de erosión.

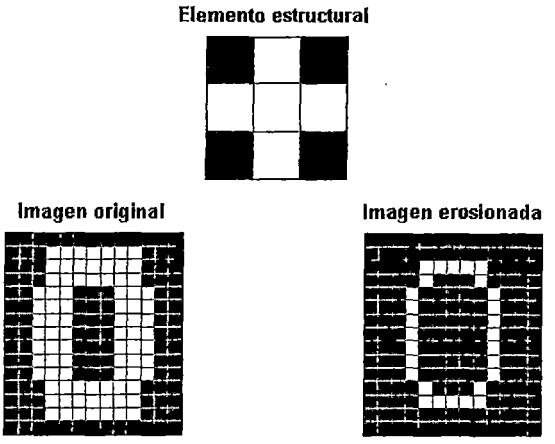

Figura C.13: Otra aplicación de erosión.

C.4.3. Apertura

$$\gamma_C(A) = A \circ C = (A \ominus C) \oplus C = \delta_C(\varepsilon_C(A))$$

Es la composición de un operador de erosión y otro de dilatación con el mismo elemento estructurante (ver Figura C.14).

Se obtiene desplazando el elemento estructural C por el interior del conjunto y eliminando las zonas por las que C no pueda "pasar". Entre sus efectos se tiende a alisar contornos (redondear esquinas que no contengan al elemento estructural) así como también a separar objetos en puntos estrechos.

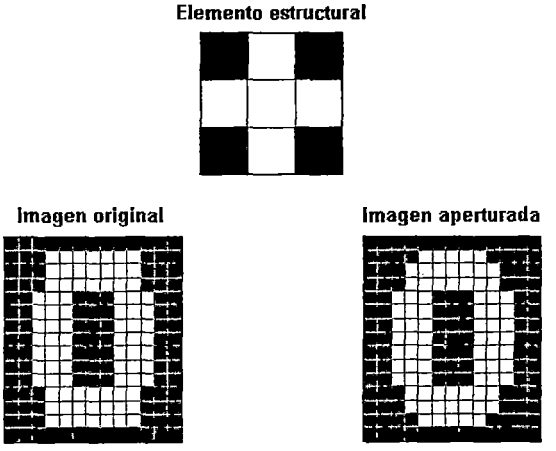

Figura C.14: Aplicación de apertura.

C.4.4. Cierre

$$\varphi_C(A) = A \bullet C = (A \oplus C) \ominus C = \varepsilon_C(\delta_C(A))$$

Es la composición de un operador de dilatación seguido de otro de erosión con el mismo elemento estructural (ver Figura C.15). El cierre tiende a alisar porciones del contorno, a fusionar grietas estrechas, rellenar vacíos del contorno (agujeros) y conectar objetos vecinos.

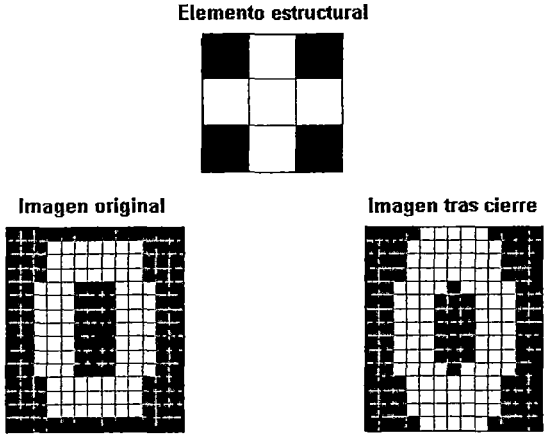

Figura C.15: Aplicación de cierre.

APÉNDICE D

Códigos de Programas

En este apéndice se adjuntan los códigos principales utilizados tanto en lo referente al *Path Planning* como en la interfaz de los robots Moway

D.1. Path Planning

En esta sección se muestran las principales funciones utilizadas, las cuales han sido escritas en Matlab. En primer lugar se muestran 2 funciones frecuentes que se han utilizado:

- linked_list_m.m, estructura de dato creada para representar a los árboles exploradores del Algoritmo RRT, esta estructura contiene 4 funciones las cuales le permiten mostrar sus elementos, adicionar un nuevo elemento en una posición determinada o al final de dicha estructura así como también nos entrega los valores contenidos.

- knnsearch.m, función codificada por Yi Cao de la Universidad de Cranfield del algoritmo KNN, esta herramienta nos permite reducir el costo computacional de una búsqueda.

```
function listObject = linked_list_m(values)

  data = values;
  listObject = struct('display',@display_list,...
                      'addAfter',@add_element,...
                      'cont',@contenido,...
                      'delete',@delete_element);

  function display_list
    disp(data);
  end

  function add_element(values,index)
    [¬, c]=size(data);
    if nargin<2
        index=c;
    else
        index = min(index,c);
    end
    data = [data(:,1:index) values' data(:,index+1:end)];
  end

  function delete_element(index)
    data(:,index)=[];
  end

  function y=contenido()
  y=data;
  end

end
%% Example
% A=[2 5 7;4 6 9]
% L=linked_list_m(A)
%
% L =
%      display: @linked_list_m/display_list
%     addAfter: @linked_list_m/add_element
%         cont: @linked_list_m/contenido
%       delete: @linked_list_m/delete_element
%
% L.addAfter([4 5], 2)
% L.display()
%      2     5     4     7
%      4     6     5     9
%
% L.addAfter([2 3])
% L.display()
%      2     5     4     7     2
%      4     6     5     9     3
```

```
function [idx,D]=knnsearch(varargin)
% By Yi Cao at Cranfield University on 25 March 2008
[Q,R,K,fident] = parseinputs(varargin{:}); % Check inputs
error(nargoutchk(0,2,nargout)); % Check outputs
% C2 = sum(C.*C,2)';
[N,M] = size(Q);  L=size(R,1);  idx = zeros(N,K);
D = idx;
if K==1
    for k=1:N  % Loop for each query point
        d=zeros(L,1);
        for t=1:M
            d=d+(R(:,t)-Q(k,t)).^2;
        end
        if fident
            d(k)=inf;
        end
        [D(k),idx(k)]=min(d);
    end
else
    for k=1:N
        d=zeros(L,1);
        for t=1:M
            d=d+(R(:,t)-Q(k,t)).^2;
        end
        if fident
            d(k)=inf;
        end
        [s,t]=sort(d); idx(k,:)=t(1:K); D(k,:)=s(1:K);
    end
end
if nargout>1
    D=sqrt(D);
end

function [Q,R,K,fident] = parseinputs(varargin)
error(nargchk(1,3,nargin)); % Check input and output
Q=varargin{1};
if nargin<2
    R=Q;
    fident = true;
else
    fident = false;  R=varargin{2};
end
if isempty(R)
    fident = true;  R=Q;
end
if ~fident
    fident = isequal(Q,R);
end
if nargin<3
    K=1;
else
    K=varargin{3};
end
```

D.1.1. Algoritmo RRT en exploración libre

```
clc, clear all
close, close all
 % Configuración Inicial
x0=0.1;y0=0.1; theta0=pi/4;
 % Iniciando Arbol
Arbol=linked_list_m([x0 y0 theta0]);
 % Paso incremental de la rama
e=0.25;

Kmax=700;
for i=1:Kmax
    qrand=Config_Aleatoria();
    Extiende(Arbol,qrand,e);
end

figure()
hold on
arbol=Arbol.cont();
plot(arbol(1,:),arbol(2,:),'.','color','r')
title('Exploración del Algoritmo RRT','FontSize',15)
xlabel('x[m]') ylabel('y[m]')
axis([0 1 0 0.75])
```

```
function qrand=Config_Aleatoria()
% Se realiza un random dentro del mapa, X=Y=[-20 20]
% para este caso en todos los puntos ya que no hay obstaculos
    xrand=40*rand(1)-20;
    yrand=40*rand(1)-20;
qrand=[xrand yrand];
```

```
function qnear=Vecino_mas_Proximo(Arbol,qrand)
% %% Computacionalmente mas costoso
% n=length(Arbol);
% for i=1:n-1
%     d(i)=sqrt((Arbol(i,1)-qrand(1))^2+(Arbol(i,2)-qrand(2))^2);
% %     d(i)=Arbol(i,1)-qrand(1))^2+(Arbol(i,2)-qrand(2);
% end
% [¬, indice]=min(d);
% qnear=Arbol(indice,:);
%% Metodo KNNS del grupo NNS: nearest neighbour search
idx=knnsearch(qrand,Arbol);
qnear=Arbol(idx,:);
```

```
function []=Extiende(Arbol,qrand,e)
qnear=Vecino_mas_Proximo(Arbol,qrand)
u_vector=(qrand-qnear)/sqrt((qrand(1)-qnear(1))^2+...
            (qrand(2)-qnear(2))^2);
qnew=qnear+u_vector*e;
Arbol.add(qnew)
```

D.1.2. Algoritmo RRT de exploración con obstáculos

```
function []=Extiende(Arbol,qrand,e)
% Obtencion de qnew
qnear=Vecino_mas_Proximo(Arbol,qrand)
u_vector=(qrand-qnear)/sqrt((qrand(1)-qnear(1))^2+...
            (qrand(2)-qnear(2))^2);
qnew=qnear+u_vector*e;
% Verificando si qnew pertenece a una configuracion libre: s=0
s=colision(qnew);
if s==0
Arbol.addAfter(qnew)
end
```

```
function s=colision(q)
x=q(1)
y=q(2)
s=0;
es=0.5;
% Compruebo choque con obstáculos
if((x>(-10-es) && x<(0+es))&&(y>(-20-es)&& y<(-10+es)))
    s=1;
end

if((x>(10-es) && x<(20+es))&&(y>(-20-es) && y<(0+es)))
    s=1;
end

if((x>(-20-es) && x<(5+es))&&(y>(0-es) && y<(15+es)))
    s=1;
end
```

D.1.3. Algoritmo Bidireccional Básico

```
clc, clear all
close, close all

%% Configuración Inicial y Final
global nodes; %variable referida a los obstáculos
[P1,P2,nodes]=data_entorno();

%% Iniciando Arboles
Arbol_1=linked_list_m(P1);
Arbol_2=linked_list_m(P2);
Arbol_aux=linked_list_m([0 0 0]');
Q_1=linked_list_m(zeros(6,1));
Q_2=linked_list_m(zeros(6,1));
%% Paso incremental de la rama
e=0.01;

imax=500; i=0
state_2='NC';
while (state_2≠'A' || i=imax)
   qrand=Config_Aleatoria();
   state_1=Extiende(Arbol_1,Q_1,qrand,e);
   state_2=Extiende(Arbol_2,Q_2,qrand,e);
   %Cambio de Arboles
   [Arbol_1 Arbol_2]=Intercambia(Arbol_1, Arbol_2);
   [Q_1 Q_2]=Intercambia(Q_1, Q_2);
   [P1 P2]=Intercambia(P1, P2);
   i=i+1;
end

% Ramas exploradoras
figure();hold on;plot_obst();
a1=Arbol_1.cont();a2=Arbol_2.cont();
plot(a1(:,1),a1(:,2),'.','color','b')
plot(a2(:,1),a2(:,2),'.','color','r')

% Tracking  de los caminos
Sint_Arbol_1=Simpl_Arbol(Q_1,P1);
Sint_Arbol_2=Simpl_Arbol(Q_2,P2);

% Camino obtenido, formado por ramos de 2 árboles
figure();hold on;plot_obst();
plot(Sint_Arbol_1(:,1),Sint_Arbol_1(:,2),'*','color','b')
plot(Sint_Arbol_2(:,1),Sint_Arbol_2(:,2),'*','color','r')

% Camino obtenido, esquivando obstáculos
Arbol=Sint_Arbol(Sint_Arbol_1,Sint_Arbol_2,Start);
figure();plot_obst();
plot(Arbol(:,1),Arbol(:,2),'.','color','k')
```

```
function [state]=Extiende(Arbol,track,qrand,e)
%% Obtencion de qnew: criterio distancia euclidiana
qnear=Vecino_mas_Proximo(Arbol.cont(),qrand);
%% u: entradas del sistema para ir de qnear a qrand
qnew=zeros(3,1);
u_vector=(qrand(1:2,1)-qnear(1:2,1))/...
          sqrt((qrand(1)-qnear(1))^2+(qrand(2)-qnear(2))^2);
qnew(1:2,1)=qnear(1:2,1)+u_vector*e;
qnew(3,1)=atan(u_vector(2)/u_vector(1));

state=colision(qnew); % Verifico si hay colision
if state==0 % No hay colision si se cumple esta condicion
  Arbol.addAfter(qnew');
  track.addAfter([qnear' qnew']);
  if abs(qnew-qrand)<0.03 % tolerancia de proximidad
    state='A'; % Alcanzado
  else
    state='B'; % Avanzado
  end
end
```

```
function Sint_Arbol=Simpl_Arbol(Qe,ini_Arbol,p)

Qe.delete(1);
Q=Qe.cont();

if nargin<2
    n=length(Q);
    punto=Q(n,1:2);
else
    punto=p;
end

Sint_Arbol=[Q(n,3:4); Q(n,1:2)];
while (punto≠ini_Arbol)
    stop=0;
    j = 1;
    while (j<n && stop==0)
        if(Q(j,3:4)==punto)
            stop=1;
            punto_prev=Q(j,1:2);
        end
        j=j+1;
    end
    Sint_Arbol=[Sint_Arbol; punto_prev];
    punto=punto_prev;
end
```

```
function Arbol=Sint_Arbol(A,B,Start)
if(Start==A(length(A),:))
    Arbol=[Voltea_Arbol(A);B];
else
    Arbol=[Voltea_Arbol(B);A];
end
```

```
function [nA,nB]=Intercambia(A,B)
   nA=B;
   nB=A;
```

```
function Arbol=Voltea_Arbol(A)
n=length(A);
for i=1:n
    Arbol(i,:)=A(n+1-i,:);
end
```

D.1.4. Algoritmo RRT-Ext-Ext

```
clc, clear all
close, close all

%% Configuración Inicial y Final
global nodes;
[P1,P2,nodes]=data_entorno();

%% Iniciando Arboles
Arbol_1=linked_list_m(P1);
Arbol_2=linked_list_m(P2);
Q_1=linked_list_m(zeros(6,1));
Q_2=linked_list_m(zeros(6,1));
%% Paso incremental de la rama
e=0.01;

imax=500; i=0
state_2='NC';
while (state_2≠'A' || i=imax)
   qrand=Config_Aleatoria();
   state_1=Extiende(Arbol_1,Q_1,qrand,e);
   arbol1=Arbol_1.cont();
   qnew_1=arbol1(:,end);
   state_2=Extiende(Arbol_2,Q_2,qnew_1,e);
   %Cambio de Arboles
   [Arbol_1 Arbol_2]=Intercambia(Arbol_1, Arbol_2);
   [Q_1 Q_2]=Intercambia(Q_1, Q_2);
   [P1 P2]=Intercambia(P1, P2);
   i=i+1;
end

% Ramas exploradoras
figure();plot_obst();hold on
a1=Arbol_1.cont()
a2=Arbol_2.cont()
plot(a1(:,1),a1(:,2),'.','color','b')
plot(a2(:,1),a2(:,2),'.','color','r')

% Tracking  de los caminos
Sint_Arbol_1=Simpl_Arbol(Q_1,P1);
Sint_Arbol_2=Simpl_Arbol(Q_2,P2);

% Camino obtenido, formado por ramos de 2 árboles
figure();plot_obst();hold on
plot(Sint_Arbol_1(:,1),Sint_Arbol_1(:,2),'*','color','b')
plot(Sint_Arbol_2(:,1),Sint_Arbol_2(:,2),'*','color','r')

% Camino obtenido, esquivando obstáculos
Arbol=Sint_Arbol(Sint_Arbol_1,Sint_Arbol_2,Start);
figure();plot_obst();
plot(Arbol(:,1),Arbol(:,2),'.','color','k')
```

D.1.5. Algoritmo RRT-Ext-Con

```
clc, clear all
close, close all

%% Configuración Inicial y Final
global nodes;
[P1,P2,nodes]=data_entorno();

%% Iniciando Arboles
Arbol_1=linked_list_m(P1);
Arbol_2=linked_list_m(P2);
Q_1=linked_list_m(zeros(6,1));
Q_2=linked_list_m(zeros(6,1));
%% Paso incremental de la rama
e=0.01;

i=0;imax=500;
state_2='NC';
while (state_2 ≠'A'|| i=imax)
   qrand=Config_Aleatoria();
   state_1=Extiende(Arbol_1,Q_1,qrand,e);
   arbol1=Arbol_1.cont();
   qnew_1=arbol1(:,end);
   % Aplicando Conecta
   if state_2≠'C'
       state_2=Conecta(Arbol_2,Q_2,qnew_1,e);
   end
   %Cambio de Arboles
   [Arbol_1 Arbol_2]=Intercambia(Arbol_1, Arbol_2);
   [Q_1 Q_2]=Intercambia(Q_1, Q_2);
   [P1 P2]=Intercambia(P1, P2);
   i=i+1;
end

% Ramas exploradoras
figure();hold on;plot_obst();
a1=Arbol_1.cont();a2=Arbol_2.cont();
plot(a1(:,1),a1(:,2),'.','color','b')
plot(a2(:,1),a2(:,2),'.','color','r')
% Tracking  de los caminos
Sint_Arbol_1=Simpl_Arbol(Q_1,P1);
Sint_Arbol_2=Simpl_Arbol(Q_2,P2);
% Camino obtenido, formado por ramos de 2 árboles
figure();hold on;plot_obst();
plot(Sint_Arbol_1(:,1),Sint_Arbol_1(:,2),'*','color','b')
plot(Sint_Arbol_2(:,1),Sint_Arbol_2(:,2),'*','color','r')
% Camino obtenido, esquivando obstáculos
Arbol=Sint_Arbol(Sint_Arbol_1,Sint_Arbol_2,Start);
figure();plot_obst();
plot(Arbol(:,1),Arbol(:,2),'.','color','k')
```

```
function [state]=Extiende(Arbol,track,qrand,e)
%% Obtencion de qnew: criterio distancia euclidiana
qnear=Vecino_mas_Proximo(Arbol.cont(),qrand);
%% u: entradas del sistema para ir de qnear a qrand
qnew=zeros(3,1);
u_vector=(qrand(1:2,1)-qnear(1:2,1))/...
          sqrt((qrand(1)-qnear(1))^2+(qrand(2)-qnear(2))^2);
qnew(1:2,1)=qnear(1:2,1)+u_vector*e;
qnew(3,1)=atan(u_vector(2)/u_vector(1));

state=colision(qnew); % Verifico si hay colision
if state==0 % No hay colision si se cumple esta condicion
  Arbol.addAfter(qnew');
  track.addAfter([qnear' qnew']);
  if abs(qnew-qrand)<0.03 % tolerancia de proximidad
    state='A'; % Alcanzado
  else
    state='B'; % Avanzado
  end
end
```

```
function [state]=Conecta(Arbol,Q,qrand,e)
state='B';
while state=='B'
    qnear=Vecino_mas_Proximo(Arbol.cont(),qrand);
    u_vector=(qrand-qnear)/sqrt((qrand(1)-qnear(1))^2+...
                                (qrand(2)-qnear(2))^2);
    qnew=qnear+u_vector*e;
    state=colision(qnew);
        if state=='NC'
            if abs(qnew-qrand)<0.06 % tolerancia de proximidad
              state='A'; % Alcanzado
            else
              state='B'; % Avanzado
            end
        end

        if state~='C'
             Q.addAfter([qnear' qnew']);
             Arbol.addAfter(qnew');
        end
end
```

D.1.6. Algoritmo RRT para sistemas no holónomos

```
clc, clear all, close, close all
x0=0.1;y0=0.1; theta0=pi/4; % Configuración Inicial
P1=[x0 y0 theta0]';
% Iniciando Arboles
Arbol1=linked_list_m(P1);  Q_1=linked_list_m(zeros(6,1));

Kmax=200;
for i=1:Kmax
    qrand=Config_Aleatoria();
    e=Extiende(Arbol1,qrand,Q_1);
end

% Ramas exploradoras
figure();plot_obst();
a1=Arbol_1.cont()
plot(a1(:,1),a1(:,2),'.','color','b')
% Tracking  del camino
p=[0.6 0.6 pi/4]   % Configuracion deseada p
pnear=knnsearch(p,Arbol1.cont())
Sint_Arbol_1=Simpl_Arbol(Q_1,P1,pnear);
% Camino obtenido
figure();plot_obst();
plot(Sint_Arbol_1(:,1),Sint_Arbol_1(:,2),'*','color','b')
```

```
function [e]=Extiende(Arbol,qrand,track)
qnear=Vecino_mas_Proximo(Arbol.cont(),qrand);
i=1; colision=0;
%% u: entradas del sistema para ir de qnear a qrand
u=[rand(1); rand(1)];
e=norm([0.8 0.4 pi/4]-[0.1 0.1 pi/4]);
while (i<6 && colision==0 && e>0.03)
% Calculo de X_der (derivada de x): X_der=Ru
theta=qnear(3); L=0.06;  Δ_T=0.009;
R=[cos(theta)/2 cos(theta)/2;sin(theta)/2 sin(theta)/2;1/L -1/L];
qnew=qnear+Δ_T*R*u; %qnew=qnear+Δ_T*X_der
% Verificando si qnew esta libre de choque: s=0
s=colision(qnew);
 if s==0
    Arbol.addAfter(qnew');
    track.addAfter([qnear' qnew']);
    % Calculo de e: e=norm([1 1 0.3]'.*([0.8 0.4 pi/4]'-qnew));
    a=[1 1 0.8]'.*([0.8 0.4 pi/4]'-qnew);
    e=a'*a;
 end
qnear=qnew;  % Actualización para evaluar mas puntos
i=i+1;
end
```

D.2. Interfaz Moway

Como se había mencionado en el Capítulo 5 la interfaz está escrita en C# y además se han utilizado dlls para el manejo del robot Moway así como también las referidas a Emgu, que permite utilizar de manera análoga lo referido a OpenCV en C# (ver Figura D.1).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.Util;
using lib_nrf24lu1_RF;
using lib_prog_mow2;
using System.IO;
```

Figure D.1: Dlls utilizadas en la interfaz

A su vez en esta sección se muestran a continuación las principales funciones utilizadas:

D.2.1. Elementos de la interfaz

La interfaz consta principalmente de un timer (ver Figura D.2) a cuyo desbordamiento se ejecuta una máquina de estados (ver Figura D.3) que permite realizar una secuencia de cálculo de estados y envío de ordenes de movimiento mientras el móvil se desplaza.

```
public Moway_Vision() // Form de la Interfaz
 {
   vector1[0] = 0; vector1[1] = 0; vector1[2] = 0;
   vector2[0] = 0; vector2[1] = 0; vector2[2] = 0;
   vector3[0] = 0; vector3[1] = 0; vector3[2] = 0;
   InitializeComponent();

   bzirf = new Bzirf2Manager();
   this.Controls.Add(bzirf);

   bzirf.Read += new NewBziRF2DataEventHandler(bziRFReadEvent);

   bziprog = new BziprogManager2();

   timer2.Enabled = true;
   timer2.Interval = timer2_interval;
   timer2.Start();
}
```

Figure D.2: Habilitación del timer que invoca a la máquina de estados

```
public void maquina_estados()
{  int j;
   String posiciones1anterior;
   switch (estado)
   {
    case 0: break;
    case 1: // Estado Vision
            if (flagstartcamera)
            { ProcessFrame(); estado = 2;}
            else { MessageBox.Show("Inicializa captura"); estado = 0; }
            break;
    case 2: // Estado control y envio rf
            if (flagcargadatos)
            { string send1 = control_moway(vector1[0], vector2[0], vector3[0],
                vector1[1], vector2[1], vector3[1], vector1[2], vector2[2], vector3[2]);
              if (Convert.ToInt16(send1[0]) == 52)
               { sReaderLine = sReaderLine + 1;
                 StreamReader sReader = new StreamReader(filepath);
                 posiciones1anterior = posiciones1;
                 for (j = 0; j < sReaderLine; j++)
                 { posiciones1 = sReader.ReadLine();}

                 if (posiciones1 == null)
                 { sReaderLine = sReaderLine-1;  posiciones1 = posiciones1anterior; }
                    lblpd.Text = String.Concat("p1: ", posiciones1);
              enviardatos(send1);
            }
            estado = 1;
            break;
   }
}
```

Figure D.3: Desarrollo de la máquina de estados

A su vez la interfaz también realiza una captura de la información del entorno, considerando para este caso demostrativo que los obstáculos son de color negro.

```
private void btnEntorno_Click(object sender, EventArgs e)
{
    File.Delete("G:\\entorno.png");
    File.Delete("G:\\entorno.txt");
    string ruta = "G:\\entorno.txt";
    StreamWriter escritura = new StreamWriter(ruta);
    Image<Bgr, Byte> frame = _capture.QueryFrame();
    Image<Gray, Byte> bin_obst = Get_Obstacles(frame.Clone()).Erode(4);
    bin_obst.Save("G:\\entorno.png");

    string ruta_1 = "G:\\pos_ini.txt";
    StreamWriter posiciones_iniciales = new StreamWriter(ruta_1);
    posiciones_iniciales.Write(textBox1.Text);
    posiciones_iniciales.Write(" ");
    posiciones_iniciales.Write(textBox2.Text);
    posiciones_iniciales.Write(" ");
    posiciones_iniciales.Write(textBox3.Text);
    posiciones_iniciales.Close();
}
```

Figure D.4: Captura de la información del entorno

```
private static Image<Gray, Byte> Get_Obstacles(Image<Bgr, byte> image)
{
    using (Image<Hsv, Byte> imag = image.Convert<Hsv, byte>())
    {
        Image<Gray, Byte>[] channels = imag.Split();

        channels[2]._ThresholdBinaryInv(new Gray(50), new Gray(255.0));

        channels[0].Dispose();
        channels[1].Dispose();
        return channels[2];
    }
}
```

Figure D.5: Filtro de Obstáculos

D.2.2. Funciones de Control

Son funciones contenidas en el estado 2 de la máquina de estados, dichas funciones permiten el cálculo de las órdenes de movimiento requeridas para ir de una posición actual a una posición dada.

```
public string control_moway(double x1, double x2, double x3, double y1, double y2, double y3, double t1, double t2, double t3)
{
    string sendcontrol;
    string s1, s2, s3;
    s1 = control1_moway(posiciones1, x1, y1, t1, i1);
    s2 = control1_moway(posiciones1, x2, y2, t2, i2);
    s3 = control1_moway(posiciones1, x3, y3, t3, i3);
    sendcontrol = String.Concat(s1, s2, s3,"0000000000");
    return sendcontrol;
}

public string control1_moway(string posiciones, double x, double y, double theta, int i)
{
    string salida;
    double xd, yd, thetad, xe, ye, thetae, de;

    if (posiciones.Length / 8 - 1 >= i)
    {
        xd = System.Convert.ToDouble(posiciones.Substring(8 * i, 4));
        yd = System.Convert.ToDouble(posiciones.Substring(8 * i + 4, 4));
        xe = xd - x;
        ye = yd - y;
        de = Math.Sqrt(xe * xe + ye * ye);
        thetad = Math.Atan2(-ye, -xe);
        Console.WriteLine(String.Concat("Thetad: ", System.Convert.ToString(thetad), " Theta: ", System.Convert.ToString(theta)));
        thetae = thetad - theta;
        Console.WriteLine(String.Concat("Theta e: ", System.Convert.ToString(thetae), "Dist e: ", System.Convert.ToString(de)));

        if (Math.Abs(de) <= errordistancia)
        {   salida = alto;
            i = i + 1; }
        else
        {   if (Math.Abs(thetae) <= errortheta) { salida = adelante; }
            else
            {   if (thetae > 0) { salida = derecha; }
                else { salida = izquierda; } }  }
    }
    else { salida = alto; }
    return salida;
}
```

Figure D.6: Calculo de orden de movimiento

D.2.3. Función ProcessFrame

Es la función encargada del procesamiento de la información recibida por la cámara, es la que calcula los estados del móvil.

```
public void ProcessFrame()
{
   Image<Bgr, Byte> frame = _capture.QueryFrame();
   Image<Gray, Byte> bin_red = GetRedPixelMask(frame.Clone()).Erode(3).Dilate(1);
   Image<Gray, Byte> bin_green = GetSky_GreenPixelMask(frame.Clone()).Erode(2);
   Image<Gray, Byte> bin_blue = GetSky_BluePixelMask(frame.Clone()).Erode(3).Dilate(5).Erode(3);
   Image<Gray, Byte> bin_obst = Get_Obstacles(frame.Clone()).Erode(7);

   vector1 = Data_calculo(bin_red, vector1[0], vector1[1],vector1[2]);
   textBox1.Text = Convert.ToString(vector1[0]); textBox2.Text = Convert.ToString(vector1[1]); textBox3.Text = Convert.ToString(vector1[2]);

   vector2 = Data_calculo(bin_green, vector2[0], vector2[1],vector2[2]);
   textBox4.Text = Convert.ToString(vector2[0]); textBox5.Text = Convert.ToString(vector2[1]); textBox6.Text = Convert.ToString(vector2[2]);

   vector3 = Data_calculo(bin_blue, vector3[0], vector3[1],vector3[2]);
   textBox7.Text = Convert.ToString(vector3[0]); textBox8.Text = Convert.ToString(vector3[1]); textBox9.Text = Convert.ToString(vector3[2]);

   captureImageBox.Image = frame;
   redfilterImageBox.Image = bin_red;
   greenfilterImageBox.Image= bin_green;
   bluefilterImageBox.Image = bin_blue;
   obstacleBox1.Image = bin_obst;
}
```

Figure D.7: Codificación de función ProcessFrame

```
public static double[] Data_calculo(Image<Gray, byte> image_bin,double x, double y, double theta)
{
   float[] data= new float[2];
   double[] salida = new double[3];
   Find triangles and rectangles

   Find Directional Vector

   return salida;
}
```

Figure D.8: Rutina de cálculo de estados del móvil, posición y orientación.

```
#region Find triangles and rectangles
List<Triangle2DF> triangleList = new List<Triangle2DF>();
List<MCvBox2D> boxList = new List<MCvBox2D>();
using (MemStorage storage = new MemStorage()) //allocate storage for contour approximation
    for (Contour<Point> contours = image_bin.FindContours(); contours != null; contours = contours.HNext)
    {
        Contour<Point> currentContour = contours.ApproxPoly(contours.Perimeter * 0.10, storage);

        if (contours.Area > 250) //only consider contours with area greater than 250
        {
            if (currentContour.Total == 3) //The contour has 3 vertices, it is a triangle
            {
                Point[] pts = currentContour.ToArray();
                triangleList.Add(new Triangle2DF(pts[0], pts[1], pts[2]));
            }
            else if (currentContour.Total == 4) //The contour has 4 vertices.
            {
                determine if all the angles in the contour are within the range of [80, 100] degree

                if (isRectangle) boxList.Add(currentContour.GetMinAreaRect());
            }
        }
    }
#endregion
```

Figure D.9: Cálculo de triángulos y cuadriláteros.

```
#region determine if all the angles in the contour are within the range of [80, 100] degree
bool isRectangle = true;
Point[] pts = currentContour.ToArray();
LineSegment2D[] edges = PointCollection.PolyLine(pts, true);

for (int i = 0; i < edges.Length; i++)
{
    double angle = Math.Abs(
       edges[(i + 1) % edges.Length].GetExteriorAngleDegree(edges[i]));
    if (angle < 80 || angle > 100)
    {
        isRectangle = false;
        break;
    }
}
#endregion
```

Figure D.10: Condición de ángulos de contorno del cuadrilátero

```
#region Find Directional Vector
if (triangleList.Count == 1 && boxList.Count == 1)
{
        PointF C2 = triangleList[0].Centeroid;
        PointF C1 = boxList[0].center;

    data[0] = Convert.ToInt32(C1.X - C2.X);
    data[1] = Convert.ToInt32(C1.Y - C2.Y);
    salida[0] = Convert.ToDouble((C1.X + C2.X)/2);
    salida[1] = Convert.ToDouble((C1.Y + C2.Y)/2);
    salida[2] = Convert.ToDouble(Math.Atan2(data[1], data[0]));
}
else
{
    salida[0] = x;
    salida[1] = y;
    salida[2] = theta;
}
#endregion
```

Figure D.11: Cálculo de orientación del robot

D.2.4. Filtro HSV

A continuación se muestra lo referente al filtrado de los colores rojo, verde y azul mencionados en el Capítulo 5, como se aprecia en las Figuras D.12, D.13 y D.14 se trabaja principalmente en los canales 0 y 1 que corresponde a los de tonalidad y saturación respectivamente.

Por ejemplo en el filtro del color rojo se puede apreciar que se ha definido dicho color cuyo canal 0 sea numéricamente menor a 20 y mayor a 160. Y en el canal 1 se como mínimo 10. Para los otros colores se aplica el mismo criterio salvo con otros valores de ajuste.

```
private static Image<Gray, Byte> GetRedPixelMask(Image<Bgr, byte> image)
{
    using (Image<Hsv, Byte> hsv = image.Convert<Hsv, Byte>())
    {
        Image<Gray, Byte>[] channels = hsv.Split();

        CvInvoke.cvInRangeS(channels[0], new MCvScalar(20), new MCvScalar(160), channels[0]);
        channels[0]._Not();

        channels[1]._ThresholdBinary(new Gray(10), new Gray(255.0));

        CvInvoke.cvAnd(channels[0], channels[1], channels[0], IntPtr.Zero);

        channels[1].Dispose();
        channels[2].Dispose();
        return channels[0];
    }
}
```

Figure D.12: Filtrado HSV del color rojo

```
private static Image<Gray, Byte> GetSky_GreenPixelMask(Image<Bgr, byte> image)
{
    using (Image<Hsv, Byte> imag = image.Convert<Hsv, byte>())

    {
        Image<Gray, Byte>[] channels = imag.Split();

        CvInvoke.cvInRangeS(channels[0], new MCvScalar(20), new MCvScalar(60), channels[0]);

        channels[1]._ThresholdBinary(new Gray(50), new Gray(255.0));

        CvInvoke.cvAnd(channels[0], channels[1], channels[0], IntPtr.Zero);

        channels[1].Dispose();
        channels[2].Dispose();
        return channels[0];
    }
}
```

Figure D.13: Filtrado HSV del color verde

```
private static Image<Gray, Byte> GetSky_BluePixelMask(Image<Bgr, byte> image)
{
    using (Image<Hsv, Byte> imag = image.Convert<Hsv,byte>())

    {
        Image<Gray, Byte>[] channels = imag.Split();

        CvInvoke.cvInRangeS(channels[0], new MCvScalar(100), new MCvScalar(120), channels[0]);

        channels[1]._ThresholdBinary(new Gray(90), new Gray(255.0));

        CvInvoke.cvAnd(channels[0], channels[1], channels[0], IntPtr.Zero);

        channels[1].Dispose();
        channels[2].Dispose();
        return channels[0];
    }
}
```

Figure D.14: Filtrado HSV del color azul