SEP

SECRETARÍA DE EDUCACIÓN PÚBLICA

# INSTITUTO TECNOLÓGICO DE CD. MADERO

Sistema Nacional de Educación Superior Tecnológica

Dirección General de Educación Superior Tecnológica

**DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN**


# Planeación de Movimiento de Robots Móviles en Ambientes Estáticos Desconocidos

P R E S E N T A:

## ISC PEDRO TOMAS SOLIS

P A R A O B T E N E R E L G R A D O D E:

## MAESTRO EN CIENCIAS EN CIENCIAS DE LA COMPUTACIÓN

D I R E C T O R D E T E S I S:

## DR. ARTURO HERNÁNDEZ RAMÍREZ

C O – D I R E C T O R D E T E S I S:

## DR. JOSÉ GABRIEL RAMÍREZ TORRES

**CD. MADERO, TAM. MÉXICO** **DICIEMBRE DE 2007**

Instituto Tecnológico de Cd. Madero

D.I.

Cd. Madero, Tam., a 03 de Diciembre de 2007.

*Área*: Posgrado
*Nº Oficio*: U5.475/07
*Asunto*: Autorización de Impresión de Tesis

C. ING. PEDRO TOMAS SOLIS
P r e s e n t e.

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestro en Ciencias en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

**"PLANEACIÓN DE MOVIMIENTO DE ROBOTS MÓVILES EN AMBIENTES ESTÁTICOS DESCONOCIDOS"**

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A t e n t a m e n t e
***"POR MI PATRIA Y POR MI BIEN"***

M.P. María Yolanda Chávez Cinco
Jefa de la División

S.E.P.
DIVISION DE ESTUDIOS DE POSGRADO E INVESTIGACION
I T C M

MYCHC' NLCO ' cerc*

# Dedicatoria

Dedicada al guerrero más indómito que he conocido a lo largo de mi existencia, a la fuente de todas mis inspiraciones, y cuyo linaje de titanes corre con orgullo por mis venas, la señora **María Margarita Solis**, mi madre… Gracias por inculcar en mí desde pequeño el interés por la lectura y por permitirme desarmar todos los aparatos electrónicos de la casa, pero sobre todo, gracias por nunca dejar de creer en mi…

Lo siguiente, representa el legado de mis ancestros:

| | |
|---|---|
| ***Toxochihuehueuh huel in tzotzona,***<br>***tonequimilol in tlalticpac.***<br>***Ma oc cemilhuitl ye nican, antocnihuan.***<br>***Ayahue.***<br>***Tic ya on cahuazque in tlalli manic.***<br>***Yaohuaye Xon ahuiacan.***<br>***Ohuaya.***<br><br>***Ni nu kuatlaktli huehuetl xochiyoj.***<br>***Nu tlaneltokis ipan ni tlalticpactli.***<br>***Na ni mits makaj ni.***<br>***Tijpakisej tujuantij.*** | *"Id que **Sol is**, con el Sol fueron y la batalla vencieron"*<br>  |

# Agradecimientos

A Dios, por permitir que uno más de mis sueños se haga realidad…

A mi adorada abuelita Natividad Solis, por darme los principios que me permiten ser un hombre de bien…

A mi querido padre Mohawk Rarihokwats, por todos los consejos, apoyo y valores que sabiamente me ha otorgado, y de los cuales hoy soy resultado…

A todos mis hermanos de las cuatro flechas de México y Canada, en especial a mis compañeros Héctor, Javier, Horacio, Crescencio, Julio y Max, por estar siempre conmigo…

$\aleph_0$ gracias al Dr. Arturo Hernández Ramírez, por compartir conmigo sus conocimientos, brindarme un apoyo constante e infundir en mí el verdadero interés por la investigación…

Al Dr. Gabriel Ramírez Torres, por indicarme el sendero adecuado en este universo llamado Robótica…

A mis amigos incondicionales: Diana Nieto, Mago Mancilla, Rocío Tierrablanca, Carlos Quiroz, Laura Montes, Felipe Mora, Gabriela Aguilar, Frank Jegouzo, Valentín Ramírez y Liliana Vicenteño. Por acompañarme y estar conmigo en los momentos más críticos y difíciles de este proyecto…

A la familia Alvarado Tello: Alma, Roberto, Yamelith, Uriel, Elizabeth, y Bárbara…

A mis auxiliares Nubia y Javier, por ayudarme con mi experimentación en el LVR…

A todos los anteriormente citados, mi cariño, admiración y respeto por siempre...

# Resumen

La investigación en robótica móvil incluye muchas disciplinas de ciencia e ingeniería. Desde la ingeniería mecánica, eléctrica y electrónica hasta las ciencias computacionales, cognitivas y sociales. Como consecuencia, el diseño del robot involucra la integración de muchos campos de conocimiento a través de una metodología interdisciplinaria. Para resolver problemas de locomoción, se requieren nociones de mecanismos, cinemática, dinámica y teoría de control. Sistemas perceptuales robustos llevan al campo del análisis digital y temas especializados de estudio, tales como visión por computadora y el uso apropiado de tecnologías de sensores. Localización y navegación demandan la comprensión de algoritmos computacionales, teoría de la información, inteligencia artificial, y teoría de la probabilidad.

La principal contribución de esta investigación es un sistema robótico de navegación construido con la integración de tres componentes:

1) Un planeador de rutas local, donde los obstáculos detectados por los sensores del robot son representados como restricciones lineales en su espacio de velocidades, formando un subconjunto convexo llamado “el polígono de velocidades factibles (FVP)”. El planeador está compuesto de dos módulos: el primero representa un cálculo de distancia mínima entre un punto y el FVP; el segundo utiliza el FVP para escapar de una situación de estancamiento.
2) El simulador MobileSim de MobileRobots y,
3) Un robot móvil Pioneer 2DX con sensores ultrasónicos y codificadores de odometría en las ruedas.

Resultados experimentales muestran un tiempo de cálculo aceptable en el desempeño de los algoritmos y una estabilidad globalmente exponencialmente del sistema, logrando una navegación exitosa en todos los casos donde obstáculos poligonales convexos son involucrados.

# Summary

Research in mobile robotics includes many engineering and science disciplines, from mechanical, electrical and electronics engineering to computer, cognitive and social sciences. It follows that robot design involves the integration of many fields of knowledge through an interdisciplinary approach. To solve locomotion problems, notions of mechanisms, kinematics, dynamics and control theory are required. Robust perceptual systems lead to the fields of signal analysis and specialized themes of study such as computer vision and the appropriate use of sensor technologies. Localization and navigation demand understanding of computer algorithms, information theory, artificial intelligence, and probability theory.

The main contribution of this research is a navigation robotic system that is built with the integration of three components:

1) A local path planner where the objects detected by the sensors of the robot are represented as linear restrictions on its velocity space, forming a convex subset that is called the “feasible velocities polygon (FVP)”. The planner, in turn, is composed of two modules: the first one represents a minimum distance calculation between one point and the FVP; the second one uses the FVP in order to escape from a deadlock situation.
2) MobileSim simulator of MobileRobots and,
3) A mobile robot Pioneer 2DX with ultrasonic sensors and odometry encoders in the wheels.

Experimental results show an acceptable calculation time in the algorithms performance and a globally exponentially stability of the system, achieving successful navigation in all the cases where convex polygonal obstacles are involved.

# Contenido

# Índice de Figuras

## Índice de Tablas

*Robótica: Disciplina científica encargada de construir y programar robots*
*Enciclopedia Galáctica, manual de robótica, 56ª edición. Año 2058 D.C.*

# Capítulo 1

# Introducción

En este capítulo se induce de manera general al campo de la robótica, primero se definen algunas palabras y bibliografía que fueron parte fundamental del origen de este campo, y se muestran las clasificaciones de robots relevantes a esta investigación. Después se realiza un análisis de la planeación de movimiento y de los trabajos previos que la han abordado. Por último se realiza la descripción y justificación del problema, y se determinan los objetivos, restricciones y delimitaciones del mismo.

## 1.1. Antecedentes generales de Robótica

El origen de la palabra *robot* proviene de la palabra eslava *robota*, que se refiere al servicio o trabajo realizado de manera forzada. Esta palabra fue utilizada por primera vez en el año de 1921 por el escritor checo Karel Capek en el estreno de su obra de teatro *Rossum's Universal Robot* (R.U.R.). La trama era sencilla: el hombre crea robots que le ayudan a realizar el trabajo, con el paso del tiempo les da la capacidad de tomar decisiones y comportarse como humanos, pero cuando los robots logran superar física y mentalmente a la humanidad, estos la destruyen. Años después, en 1941 Isaac Asimov en su novela *Runaround* introdujo por

primera vez el término *robótica*, término que sería significativo en las series *The foundation* y *Robots*. En la primera serie, Asimov aborda la decadencia del imperio galáctico y su principal tecnología de apoyo que son los robots. El imperio galáctico es creado con base a la necesidad de expansión de los seres humanos al espacio exterior y abarca miles de planetas de nuestra galaxia. Esta saga ocurre en un futuro lejano y su tema principal es evitar la destrucción producida por la desaparición de un sistema federal del gobierno galáctico. Con el fin de producir el menor daño posible por la caída del imperio, se crean dos poderosas organizaciones: una pública (la primera fundación), y otra secreta (la segunda fundación). Estas organizaciones se encuentran situadas en lugares opuestos de la galaxia y su función principal es la de fungir como promotores para la creación de un segundo imperio galáctico, en el cual se aplicarían medidas dictadas por la ficticia *ciencia aplicada* y *la Psicohistoria*, permitiendo reducir el periodo inevitable de caos a sólo mil años. Por otra parte, en la serie *Robots*, Asimov define por primera vez las tres leyes de la robótica, mismas que establecen las restricciones para toda máquina considerada como robot y presenta a Elijah Baley, un detective de la policía de Nueva York el cual no tiene mucho afecto por las compañías que fabrican robots ni por quienes trabajan en ellas. Sin embargo, cuando un prominente científico es asesinado de una manera misteriosa, Baley es enviado a los mundos exteriores a buscar al asesino, llevándolo su travesía al planeta *Solaria*, un lugar poblado por colonias de humanos con robots como sirvientes en donde será apoyado por el robot Daneel Olivaw para tratar de solucionar el asesinato al que fue encomendado. No obstante, el jamás imaginó que tendría que enfrentar las complejidades psicológicas de un mundo en donde un robot puede tener una apariencia del todo humana.

Actualmente, la mayoría de la sociedad asocia la palabra robot con cualquier maquina hecha por el hombre que desarrolle el trabajo y tareas pertenecientes a los humanos. De una manera general, los robots pueden ser clasificados como:

a) **Androides**. Son robots cuyo propósito principal es reproducir total o parcialmente la forma y el comportamiento del ser humano. Aunque en la actualidad se han logrado algunos avances en diseño, todavía son dispositivos poco evolucionados, con poca

utilidad práctica, y destinados fundamentalmente al estudio y experimentación. Uno de los aspectos más complejos de estos robots, y sobre el que se centra la mayoría de los trabajos, es el de la locomoción bípeda. En este caso, el principal problema es controlar dinámica y coordinadamente en el tiempo real el proceso y mantener simultáneamente el equilibrio del robot.

b) **Móviles**. Estos robots perciben información del entorno que los rodea por medio de sus sensores, lo cual les permite llevar a cabo una determinada acción, como en el caso de la navegación de ambientes que ejecutan a través de actuadores como patas, ruedas u orugas. Son empleados en instalaciones industriales para tareas como el transporte de mercancías en cadenas de producción y almacenes, también son utilizados en la exploración espacial, y en la investigación de lugares peligrosos y distantes como cuevas submarinas.

c) **Zoomórficos**. Este tipo de robots se caracteriza principalmente por tratar de imitar los sistemas de locomoción de los seres vivos. Lo cual de una manera muy general, podría abarcar también a los androides. Actualmente se está experimentando con este tipo de máquinas en diversos laboratorios con el fin de llegar al resultado de tener sistemas robóticos completamente adaptables a todo tipo de terreno y cuyo funcionamiento sea de manera autónoma. Las aplicaciones de estos robots son principalmente orientadas a la exploración espacial y lugares donde existen superficies muy complicadas como en el caso de volcanes.

d) **Industriales.** Son robots que realizan de manera automática determinados procesos de manufactura o manipulación. Los países con mayor fabricación y uso de robots industriales son Japón y Estados Unidos. Resulta interesante ver la forma en cómo estos dos países definen al robot industrial:

Para la Asociación Japonesa de Robótica Industrial (JIRA): Los robots son "dispositivos capaces de moverse de modo flexible análogo al que poseen los

organismos vivos, con o sin funciones intelectuales, permitiendo operaciones en respuesta a las órdenes humanas".

Para el Instituto de Robótica de América (RIA): Un robot industrial es "un manipulador multifuncional y reprogramable diseñado para desplazar materiales, componentes, herramientas o dispositivos especializados por medio de movimientos programados variables con el fin de realizar tareas diversas".

En lo anterior se puede observar la amplitud de la definición japonesa en contraste con la concreta definición americana. Por ejemplo, un robot manipulador que requiere un operador "mecánicamente enlazado" a él se considera como un robot en Japón, pero no en la definición americana. De igual forma, una máquina automática no programable entraría en la definición japonesa y no en la americana. Una ventaja de la amplitud japonesa es que a varios dispositivos automáticos se les llama "robots" en Japón, lo que conlleva a pensar que los japoneses han aceptado al robot en su cultura mucho más fácilmente que los países occidentales, aunque la definición americana es la internacionalmente aceptada.

## 1.2. Clasificación de los robots móviles

En [Ramírez 2000] se hace mención de que la característica más remarcable de un robot móvil es evidentemente su medio de locomoción. En [Pruski 1996] se clasifican los robots móviles en cuatro grupos distintos, de acuerdo al tipo de locomoción que utilizan:

1. **Robots móviles con ruedas**. Tomando en cuenta la simplicidad del mecanismo de locomoción, este tipo de robot es el más difundido actualmente. Su principal operación es llevada a cabo en sitios industriales o ambientes interiores, aunque existe igualmente su aplicación en ambientes exteriores, como la exploración espacial. La gran mayoría de los robots de este tipo presentan las restricciones no holonómicas que limitan el movimiento instantáneo que el robot puede realizar. Estas restricciones tienen como

consecuencia el aumentar la complejidad del problema de planificación de trayectorias y de su control.

2. **Robots móviles con orugas**. Cuando el terreno es accidentado, las ruedas pierden la eficiencia de la locomoción. Lo cual limita la capacidad de evolución del robot móvil equipado con este tipo de sistema de locomoción. En estas condiciones, las orugas son más interesantes debido a que estas permiten aumentar la adherencia al terreno y atravesar por los obstáculos más importantes. Este tipo de robots presenta igualmente las restricciones de no holonomía.

3. **Robots móviles con patas**. En situaciones donde el terreno es todavía más incierto, con grandes diferencias de altura como por ejemplo un escalón o un terreno muy accidentado, los robots de ruedas o las orugas no son muy eficaces, por lo que se tiene que recurrir a los robots móviles con patas. Estos robots tienen los puntos de apoyo discretos sobre el terreno y son por lo tanto la solución a este problema de movimiento. Por el contrario, la concepción y el control de una maquina con patas es muy compleja. En adición, su velocidad de evolución es generalmente muy reducida.

4. **Otros medios de locomoción**. Es el grupo de todos los robots móviles que utilizan un sistema de locomoción diferente de los tres precedentes. Por ejemplo, los robots móviles que se desplazan por arrastre, los robots submarinos, los robots para exploración espacial, robots voladores, etc. Las aplicaciones de este tipo de robots son muy especializadas y su arquitectura es en general específica a la aplicación vista.

## 1.3. Planeación de movimiento

Desde el clásico problema de planeación de rutas para el movimiento del piano [Schwartz y Sharir 1983], la planeación de movimiento se ha utilizado en áreas como la animación de caracteres digitales, planeación quirúrgica, verificación automática de croquis de empresas, mapeo de ambientes inexplorados, navegación de ambientes cambiantes, ensamble secuencial

y diseño de medicamento. En robótica e inteligencia artificial, la manipulación de robots móviles representa un área de investigación altamente activa. *El problema básico de planeación de movimiento es encontrar una ruta libre de colisiones para el robot entre obstáculos rígidos estáticos*. Este es un problema geométrico que a excepto de los robots con pocos DOF (Degrees Of Freedom – Grados De Libertad), es computacionalmente Duro [Latombe 1999]. Se han estudiado varias extensiones del problema básico, por ejemplo; la construcción de mapas de entorno, la representación del espacio libre y obstáculos, el modelado de incertidumbre en los sensores, la evasión de obstáculos fijos y dinámicos, la planeación de trayectorias, restricciones cinemáticas y dinámicas que limitan los movimientos del robot, o múltiples robots que tienen que ser coordinados, etc. Algunos de estos problemas tienen ahora soluciones que pueden ser utilizadas en práctica, por ejemplo; la adquisición de modelos espaciales de ambientes físicos es abordada por el mapeo robótico, el cual es base para el desplazamiento de robots móviles verdaderamente autónomos. Aunque actualmente se cuenta con métodos robustos para construir mapas de ambientes estáticos, y técnicas eficientes para la interpretación y navegación de los mismos, la planeación de movimiento en tales ambientes, sigue representando un problema abierto de exploración.

### 1.3.1. Trabajo Previo de la Planeación de Movimiento

Algunas de las razones del éxito de la planeación en movimiento son que es lo suficientemente fácil de evaluar y que la mayoría de los resultados elevan la motivación por el desarrollo de nuevos temas de investigación. Parte de estos temas que han sido relevantes durante las últimas tres décadas son descritos en [Latombe 1999]:

En la planeación de movimiento, [Nilsson 1969] describió un sistema para un robot móvil con capacidades de planeación de movimiento, introduciendo el método de grafo de visibilidad combinado con el algoritmo de búsqueda *A** para encontrar la trayectoria más corta, libre de colisiones, para un robot representado por un punto entre obstáculos poligonales. [Udupa 1977] introdujo la idea de reducir un robot en un punto para evitar colisiones. [Lozano-Pérez 1979] explotó esta idea en una forma más general y sistemática y propusieron un planeador de

trayectorias completo para robots poligonales / poliédricos moviéndose en translación entre obstáculos poligonales / poliédricos. Este trabajo condujo al concepto de espacio de configuración.

En cuanto a fundamento matemático, [Lozano-Pérez 1983] introdujo el concepto de espacio de configuración del robot, en el cuál el robot se representa como un punto- llamado una configuración- en un espacio de parámetros que codifican los DOFs del robot, los obstáculos se representan en el mapa como regiones prohibidas en el espacio de configuración y el complemento de estas regiones es el espacio libre. Durante los 80s, las herramientas clásicas de geometría diferencial fueron utilizadas para estudiar la estructura de un espacio de transformaciones pertenecientes a un espacio de configuración (manifold), junto con sus propiedades topológicas, geométricas y algebraicas más específicas. La mayoría de estos resultados matemáticos son recopilados en [Latombe 1991], los cuales han representado un rol crucial en el entendimiento de problemas de planeación de movimiento, especialmente planeación no holonómica y planeación óptima.

En el análisis de complejidad computacional, [Reif 1979] mostró que la planeación de trayectorias de una articulación en 3D hecha de poliedros conectados es PSPACE-duro. La prueba utiliza los DOFs de un robot para codificar la configuración de una máquina de Turing delimitada a un espacio polinomial y diseña obstáculos que forzan a los movimientos del robot a simular el cálculo de esta máquina. En [Schwartz y Sharir 1983] se propuso un algoritmo completo de planeación de trayectorias de propósito general basado en una descomposición algebraica del espacio de configuración del robot conocido como la descomposición Collins. El algoritmo en [Halperin y Sharir 1996] toma un tiempo $O((kn)^{2+\varepsilon})$ para mover libremente un robot poligonal de $k$ lados en un espacio de trabajo poligonal.

La complejidad de planeadores de trayectorias completos y/o su falta de robustez han motivado el desarrollo de planeadores heurísticos. Dos métodos populares que pueden resolver problemas de planeación de trayectorias complejos en espacios de configuración en dos y tres dimensiones fueron introducidos en los 80s: 1) Descomposición en celdas de aproximación,

donde el espacio libre se representa por una colección de celdas simples [Brooks y Lozano-Pérez 1983]. 2) Campo potencial [Khatib 1986], que fue inicialmente propuesto para evitar colisiones en línea, pero puede combinarse con técnicas de búsqueda en rejillas para resolver problemas de planeación [Barraquand y Latombe 1991]. Un planeador aleatorio fue introducido [Barraquand y Latombe 1991], el cual es capaz de resolver problemas complejos de planeación de trayectorias para robots con varios DOFs por la alternación de movimientos para rastrear el gradiente negado de un campo potencial y movimientos aleatorios para escapar de mínimos locales. [Kavraki 1996] muestra otro planeador aleatorio que consiste en muestrear el espacio de configuración de forma aleatoria y conectar las muestras en el espacio libre por medio de trayectorias locales (trayectorias directas), y entonces crear un mapa de trayectorias probabilístico (PRM por sus siglas en ingles Probabilistic Road Map). Las muestras y trayectorias locales evitan los obstáculos mediante un verificador rápido de colisiones [Quinlan 1994] el cual evita el cálculo de una representación explícita del espacio libre.

## 1.4. Descripción del problema

El presente proyecto describe la investigación referente a la planeación de movimiento de robots móviles mediante la navegación incremental en ambientes estáticos desconocidos. Para esto, se utiliza un robot Pioneer 2DX con tres ruedas de Active Media Robotics, el cual debe ser capaz de salir de un punto de inicio y llegar a un punto meta evadiendo los obstáculos que se encuentren en la trayectoria de navegación entre estos dos puntos. El seguimiento de trayectoria hacia la meta y circunnavegación de los obstáculos se llevará a cabo por medio de la implementación de un planeador de rutas local inicialmente propuesto en [Ramírez y Zeghloul 2000]. En este método se representan los obstáculos detectados por el robot como restricciones lineales en su espacio de velocidades. Debido a que las restricciones de obstáculos son lineales, estas forman un subconjunto convexo que representa las velocidades que el robot puede utilizar sin colisionar con los objetos. Este subconjunto convexo es denominado polígono de velocidades factibles (FVP, por sus siglas en ingles: Feasible Velocity Polygon). El planeador está compuesto de dos módulos: el primero representa un problema de optimización, transformado en un problema de cálculo de distancia mínima en el

espacio de velocidades entre el FVP y un punto; y el segundo se utiliza en una situación de estancamiento, utilizando el FVP para encontrar localmente la mejor velocidad para escapar del mismo. Algunas de las ventajas de este método son su rápido cálculo y el comportamiento continuamente estable de las velocidades.

## 1.5. Justificación del problema

Aunque en la actualidad se han propuesto diferentes métodos y técnicas para resolver la planeación de movimiento de robots móviles, la implementación de nuevas estrategias y la combinación de las ya existentes sigue representando un tema abierto de investigación. En este contexto, la autonomía de los robots y su adaptación a las nuevas metodologías sigue representando un paso fundamental en el desarrollo de nuevas tecnologías.

## 1.6. Objetivos

### 1.6.1. Objetivo General

Desarrollar un sistema que permita controlar la navegación de un robot móvil del tipo Pionner 2DX entre dos puntos, inicio y meta, de un ambiente estático desconocido con obstáculos poligonales convexos mediante el uso de un planeador de rutas local basado en el método del Polígono de Velocidades Factibles.

### 1.6.2. Objetivos específicos

- Indicar los puntos de configuración de inicio $q_{start}$ y meta $q_{goal}$.
- Establecer el espacio de velocidades del robot, y computar la velocidad de avance.
- Mapear los objetos en la zona de influencia como restricciones lineales en el espacio de velocidades del robot.
- Formar el polígono de velocidades factibles.

- Resolver el problema de optimización por el cálculo de distancia mínima entre el FVP y $\mathbf{u}_{goal}$.
- Utilizar el módulo "alcanzando la meta" para aproximar al robot al punto $q_{goal}$.
- Utilizar el módulo "seguimiento de frontera" para permitirle al robot escapar de estancamientos.

## 1.7. Restricciones y delimitaciones

### 1.7.1. Restricciones

- La representación de obstáculos es en $\mathbb{R}^2$.
- Los obstáculos son considerados como polígonos convexos.
- Las restricciones de velocidad están sujetas a las velocidades del robot.
- La información de la zona de influencia está condicionada al rango de alcance de los sensores.
- La evasión de obstáculos en un ambiente está sujeta a las restricciones de distancia establecidas en el planeador de rutas local.

### 1.7.2. Delimitaciones

- Se consideran ambientes estructurados y entornos estáticos (obstáculos fijos).
- Las superficies de navegación deben ser planas.
- Se utilizará el simulador MobileSim y la interface de programación Aria para realizar pruebas de modelado y manipulación del robot.
- Se utiliza lenguaje C++ y la interfaz ARIA para la programación del planeador de rutas local.

*Matemáticas: La síntesis del cálculo de n-variables y de la geometría n-dimensional es la base de lo que Seldon alguna vez llamó "mi pequeña algebra de la humanidad"*
*Enciclopedia Galáctica, manual de robótica, 56ª edición. Año 2058 D.C.*

# Capítulo 2

# Marco Teórico

En esta sección se describe el fundamento matemático y, los conceptos y modelos computacionales necesarios para realizar esta investigación. Se comienza con las definiciones básicas necesarias para lograr que un sistema robótico sea globalmente exponencialmente estable según el teorema de Lyapunov. Después se analizan algunos de los métodos de navegación y evasión de obstáculos que actualmente resuelven el problema de planeación de movimiento para robots móviles.

## 2.1. Teoría de la estabilidad de Lyapunov

### 2.1.1. Definiciones básicas

**Definición 1.** Una función $f(x)$ es *continua por pedazos* (o *seccionalmente continua*), si cumple con las siguientes condiciones:

1. Es continua en cualquier subintervalo de definición.
2. El número de discontinuidades es finito o numerable.

3. En los puntos de discontinuidad existen los dos límites laterales.

**Definición 2.** Una función $f(x,t)$ es *continuamente Lipschitziana* (o *Lipschitz continua*) en *x* si dado $x_1, x_2 \in \mathbb{R}^n$ se cumple:

$$\|f(x_1,t) - f(x_2,t)\| \leq k\|x_1 - x_2\| \tag{1}$$

**Definición 3.** Una función $f(x,t)$ es *uniformemente continua* en *t* si para toda $\varepsilon > 0$, existe una $\delta > 0$ tal que:

$$\|f(x,t_1) - f(x,t_2)\| < \varepsilon \tag{2}$$

Siempre y cuando $\|t_1 - t_2\| < \delta$.

Si se considera un sistema dinámico que satisfaga a

$$\dot{x} = f(x,t) \qquad x(t_0) = x_0 \qquad x \in \mathbb{R}^n \tag{3}$$

Se asume que $f(x,t)$ satisface las condiciones estándar para la existencia y unicidad de las soluciones. Tales condiciones son, por ejemplo, que $f(x,t)$ sea Lipschitz continua con respecto a *x*, uniformemente continua en *t*, y continua por pedazos en *t.*

**Definición 4.** Un punto $x^* \in \mathbb{R}^n$ es un punto de equilibrio de (1) si $f(x^*,t) \equiv 0$.

Intuitivamente y de manera informal, se dice que un punto de equilibrio es *localmente estable* si todas las soluciones que comienzan cerca de *x** (lo cual significa que las condiciones iniciales están en una vecindad de *x**) permanecen cerca de *x** todo el tiempo.

Se dice que el punto de equilibrio $x^*$ es *localmente asintóticamente estable* si $x^*$ es localmente estable y, además, todas las soluciones que comienzan cerca de $x^*$ tienden hacia $x^*$ cuando $t \to \infty$.

Se utiliza el término informal por que la naturaleza de variación de tiempo de la ecuación (3) introduce todo tipo de sutilezas adicionales. No obstante, es intuitivo que un péndulo tenga un punto de equilibrio localmente estable cuando se esta columpiando hacia abajo y un punto de equilibrio inestable cuando lo hace hacia arriba. Si el péndulo es amortiguado, el punto de equilibrio estable es localmente asintóticamente estable.

Si se cambia el origen del sistema, se puede asumir que el punto de equilibrio de interés ocurre en $x^* = 0$. Si existen múltiples puntos de equilibrio, se necesitará estudiar la estabilidad de cada uno por el cambio apropiado del origen.

**Definición 5. Estabilidad en el sentido de Lyapunov**

El punto de equilibrio $x^* = 0$ de (3) es estable (en el sentido de Lyapunov) en $t = t_0$ si para cualquier $\varepsilon > 0$ existe una $\delta(t_0, \varepsilon) > 0$ de manera que:

$$\|x(t_0)\| < \delta \qquad \Rightarrow \qquad \|x(t)\| < \varepsilon, \qquad \forall t \geq t_0 \tag{4}$$

La estabilidad de Lyapunov es un requerimiento débil en los puntos de equilibrio. En particular, no requiere que las trayectorias que comienzan cerca del origen tiendan al origen asintóticamente. También, la estabilidad se define en un instante de tiempo $t_0$. La e*stabilidad uniforme* es un concepto que garantiza que el punto de equilibrio no está perdiendo estabilidad. Aquí se especifica que para un punto de equilibrio uniformemente estable $x^*$, $\delta$ en la definición 5, no es función de $t_0$, de manera que la ecuación (4) se mantiene para todo $t_0$.

**Definición 6. Estabilidad asintótica**

Un punto de equilibrio $x^* = 0$ de (3) es asintóticamente estable en $t = t_0$ si:

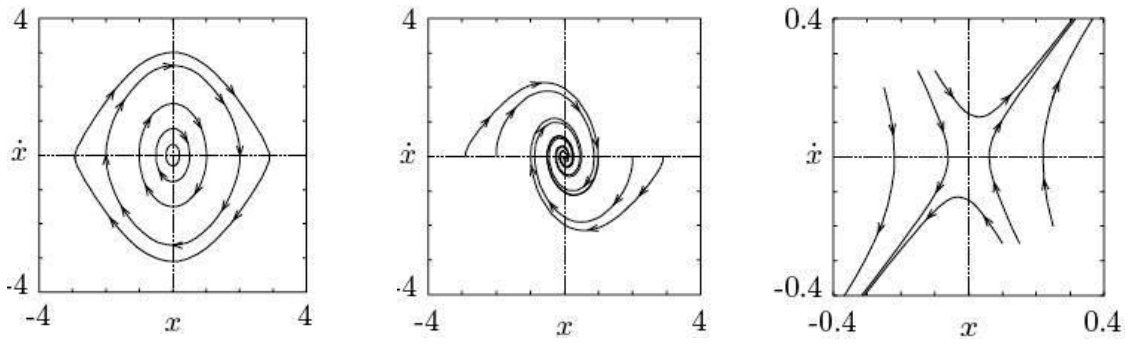
1. $x^* = 0$ es estable.
2. $x^* = 0$ es localmente atractivo; esto significa que existe $\delta(t_0)$ de manera que

$$\|x(t_0)\| < \delta \qquad \Rightarrow \qquad \lim_{t\to\infty} x(t) = 0 \tag{5}$$

Como se indicó previamente, la estabilidad asintótica es definida en $t_0$. La *estabilidad asintótica uniforme* requiere que:

1. $x^* = 0$ sea uniformemente estable.
2. $x^* = 0$ sea uniformemente localmente atractivo; lo que significa que existe una $\delta$ independiente de $t_0$ para que cada ecuación de (5) se mantenga. Además, se requiere que la convergencia en la ecuación (5) sea uniforme.

Finalmente, se dice que un punto de equilibrio es *inestable* si este no es estable. Esto es menos tautológico de lo que suena y se debe asegurar la negación de la definición de estabilidad en el sentido de Lyapunov para obtener la definición de inestabilidad. En Robótica, siempre existe un interés por un equilibrio uniformemente asintóticamente estable. Si se desea mover el robot hacia un punto, se deseará actualmente converger a ese punto, no solamente permanecer cerca. La figura 2.1 ilustra la diferencia entre la estabilidad en el sentido de Lyapunov y la estabilidad asintótica.




(a) Estable en el sentido de Lyapunov (b) Estable asintóticamente (c) Inestable

Figura 2.1. Puntos de equilibrio estables e inestables

Las definiciones 5 y 6 son definiciones *locales*; por que describen el comportamiento del sistema cerca de un punto de equilibrio.

Un punto de equilibrio $x^*$ es *globalmente estable* si es estable para todas las condiciones iniciales $x_0 \in \mathbb{R}^n$. La estabilidad global es muy deseable, pero en muchas aplicaciones es muy difícil de lograr. Nociones de uniformidad son solo importantes para sistemas de tiempo variante, por lo que, para sistemas de tiempo invariante, la estabilidad implica estabilidad uniforme y estabilidad asintótica que implica estabilidad uniforme asintótica.

Es importante notar que las definiciones de estabilidad asintótica no cuantifican el rango de convergencia, por lo que existe una forma más fuerte de estabilidad, la cual demanda un rango exponencial de convergencia:

**Definición 7. Estabilidad exponencial, rango de convergencia.**
El punto de equilibrio $x^* = 0$ es un punto de equilibrio *exponencialmente estable* de (3) si existen constantes $m, \alpha > 0$ y $\varepsilon > 0$ de manera que:

$$\|x(t)\| \leq me^{-\alpha(t-t_0)} \|x(t_0)\| \tag{6}$$

Para todas las $\|x(t_0)\| \leq \varepsilon$ y $t \geq t_0$. La constante $\alpha$ más larga que puede ser utilizada en (6) es llamada *rango de convergencia*.

La estabilidad exponencial es una forma fuerte de estabilidad; y en particular, implica estabilidad asintótica, uniforme. La convergencia exponencial es importante en las aplicaciones por que puede mostrar ser robusta hacia perturbaciones y es esencial para la consideración de algoritmos de control avanzados, como en el caso de los adaptativos.

Un sistema es *globalmente exponencialmente estable* si el límite en la ecuación (6) se mantiene para todos los $x_0 \in \mathbb{R}^n$.

Donde quiera que sea posible, se debe tratar de lograr probar la estabilidad exponencial, global.

### 2.1.2. El método directo de Lyapunov

El método directo de Lyapunov (también llamado método de Lyapunov), permite determinar la estabilidad de un sistema sin explícitamente integrar la ecuación diferencial de (3). El método es una generalización de la idea de que si existe alguna "medida de energía" en un sistema, entonces se puede estudiar el rango de cambio de la energía del sistema para determinar la estabilidad. Para hacer esto más preciso, se debe definir exactamente el significado de "medida de energía".

**Definición 8. Funciones definidas positivas localmente (lpdf)**

Sea $B_\varepsilon$ una bola de tamaño $\varepsilon$ alrededor del origen, $B_\varepsilon = \{x \in \mathbb{R}^n : \|x\| < \varepsilon\}$.

Una función continua $V : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ es una *función definida positiva localmente* si para algún $\varepsilon > 0$ y para alguna función estrictamente creciente, continua $\alpha : \mathbb{R}_+ \to \mathbb{R}$,

$$V(0,t) = 0 \qquad \text{y} \qquad V(x,t) \geq \alpha(\|x\|) \qquad \forall x \in B_\varepsilon, \forall t \geq 0 \tag{7}$$

Una función definida positiva localmente es localmente similar a una función de energía. Funciones que son globalmente similares a funciones de energía son llamadas funciones definidas positivas.

**Definición 9. Funciones definidas positivas (pdf)**

Una función continua $V : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ es una función definida positiva si satisface la condición de la definición 8 y adicionalmente $\alpha(p) \to \infty$ como $p \to \infty$.

Para el límite de la función de energía anterior, el decreciente se define como sigue:

**Definición 10. Funciones decrecientes**

Una función continua $V:\mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}$ es decreciente si para algún $\varepsilon > 0$ y alguna función estrictamente creciente, continua $\beta:\mathbb{R}_+ \to \mathbb{R}$,

$$V(x,t) \le \beta(\|x\|) \qquad \forall x \in B_\varepsilon, \forall t \ge 0 \tag{8}$$

Utilizando estas definiciones, el siguiente teorema permite determinar la estabilidad de un sistema por el estudio de una apropiada función de energía. A grandes rasgos, este teorema establece que cuando $V(x,t)$ es una función definida positiva localmente y $\dot{V}(x,t) \le 0$ se puede concluir la estabilidad del punto de equilibrio. La derivada del tiempo de $V$ se toma a lo largo de las trayectorias del sistema:

$$\dot{V}\Big|_{\dot{x}=f(x,t)} = \frac{\partial V}{\partial t} + \frac{\partial V}{\partial x} f \tag{9}$$

En lo que sigue, $\dot{V}$ tendrá el significado de $\dot{V}\Big|_{\dot{x}=f(x,t)}$.

### 2.1.3. Teorema básico de Lyapunov

Sea $V(x,t)$ una función no negativa con derivada $\dot{V}$ a lo largo de las trayectorias del sistema. Las condiciones de este teorema son resumidas en la tabla 2.1.

1. Si $V(x,t)$ es definida positiva localmente y $\dot{V}(x,t) \le 0$ localmente en $x$ y $\forall t$, entonces el origen del sistema es localmente estable (en el sentido de Lyapunov).
2. Si $V(x,t)$ es definida positiva localmente y decreciente, y $\dot{V}(x,t) \le 0$ localmente en $x$ y $\forall t$, entonces el origen del sistema es uniformemente localmente estable (en el sentido de Lyapunov).

3. Si $V(x,t)$ es definida positiva localmente y decreciente, y $-\dot{V}(x,t)$ es definida positiva localmente, entonces el origen del sistema es uniformemente localmente asintóticamente estable.
4. Si $V(x,t)$ es definida positiva y decreciente, y $-\dot{V}(x,t)$ es definida positiva, entonces el origen del sistema es globalmente uniformemente asintóticamente estable.

Tabla 2.1. Resumen del teorema básico de Lyapunov

| | Condiciones en $V(x,t)$ | Condiciones en $-\dot{V}(x,t)$ | Conclusiones |
|---|---|---|---|
| 1 | lpdf | $\geq 0$ localmente | Estable |
| 2 | lpdf, decreciente | $\geq 0$ localmente | Uniformemente estable |
| 3 | lpdf, decreciente | lpdf | Uniformemente asintóticamente estable |
| 4 | pdf, decreciente | pdf | Globalmente uniformemente asintóticamente estable |

El teorema básico de Lyapunov da condiciones suficientes para la estabilidad del origen de un sistema. Sin embargo, esto no da una prescripción para determinar la función de Lyapunov $V(x,t)$. Dado que el teorema ofrece tales condiciones, la búsqueda por una función de Lyapunov de establecimiento de estabilidad de un punto de equilibrio podría ser ardua. Sin embargo, es remarcable el hecho de que el reciproco del teorema básico de Lyapunov también existe: si un punto de equilibrio es estable, entonces existe una función $V(x,t)$ que satisface las condiciones del teorema. Sin embargo, la utilidad de este y otros teoremas recíprocos está limitada por la falta de una técnica computable para generar funciones de Lyapunov.

## 2.2. Métodos de evasión de obstáculos

### 2.2.1. Algoritmos Bug

*Algoritmos Bug1 y Bug2*

Los algoritmos Bug1 y Bug2 [Lumelsky y Stepanov 1987], inspirados en el comportamiento de los insectos para lograr sus objetivos, presentan básicamente dos comportamientos:

moverse en línea recta hacia la meta y seguir una frontera alrededor de un obstáculo. Durante estos movimientos, se asumen los siguientes puntos:

1. El robot está en un punto con perfecto posicionamiento (es decir, sin errores de posicionamiento), con un sensor de contacto que puede detectar la frontera de un obstáculo si el punto del robot la toca.
2. El robot puede también medir la distancia $d(\boldsymbol{x}, \mathbf{y})$ entre cualquier par de puntos $\boldsymbol{x}$ y $\boldsymbol{y}$.
3. El espacio de trabajo está delimitado.

**Definición 11.** Sea $B_r(\boldsymbol{x})$ una bola de radio $r$ centrado en $\boldsymbol{x}$, esto es, $B_r(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^2 \mid d(\boldsymbol{x}, \boldsymbol{y}) < r\}$. El hecho de que el espacio de trabajo este delimitado implica que para toda $\boldsymbol{x} \in W$, existe una $r < \infty$ tal que $W \subset B_r(\boldsymbol{x})$ (figura 2.2).

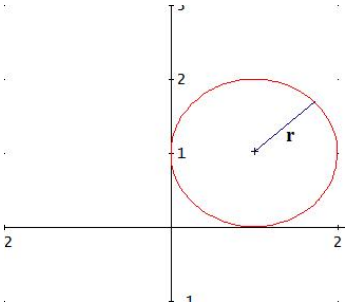


Figura 2.2. Br(x)

El inicio y la meta están etiquetados como $q_{\text{start}}$ y $q_{\text{goal}}$ respectivamente. Sea $q_0^L = q_{\text{start}}$ y sea la recta $m$ el segmento de recta que conecta a $q_i^L$ con $q_{\text{goal}}$. Inicialmente, $i = 0$.

En el algoritmo Bug1, durante el movimiento hacia la meta, el robot se mueve a lo largo de la recta $m$ hacia $q_{\text{goal}}$ hasta que encuentra la meta o un obstáculo. Si el robot encuentra un obstáculo, establece $q_1^H$ como el punto en donde el robot encuentra el primer obstáculo (a este punto se le denomina punto de choque - hit point), el robot entonces circunnavega el obstáculo hasta regresar a $q_1^H$. Entonces, el robot determina el punto más cercano a la meta en el

perímetro del obstáculo y navega hasta este punto, etiquetándolo con $q_1^L$ (a este punto se le denomina punto de partida - leave point). Desde $q_1^L$, el robot se dirige directo hacia la meta de nuevo, esto es, re-invocando el comportamiento del movimiento hacia la meta con una nueva recta *m*. Si la recta que conecta a $q_1^L$ y la meta intersecan al obstáculo actual, entonces no existe trayectoria hacia la meta; nótese que esta intersección ocurre inmediatamente después de partir de $q_1^L$. En caso contrario, el índice *i* se incrementa y se repite el procedimiento para $q_i^L$ y $q_i^H$ hasta que la meta se alcanza o el planeador determina que la meta no puede alcanzarse (figura 2.3 y figura 2.4). Finalmente, si la recta hacia la meta roza un obstáculo, el robot no necesita invocar un comportamiento de seguimiento de frontera, sino más bien continuar hacia delante, hacia la meta. El algoritmo 1 (figura 2.5) muestra una descripción general del método Bug1.

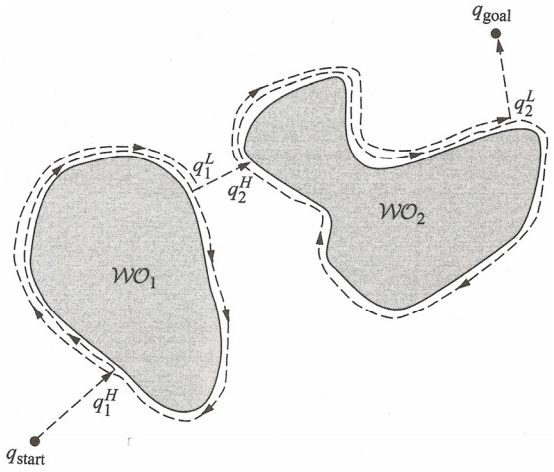



Figura 2.3. El algoritmo Bug1 encuentra la meta.

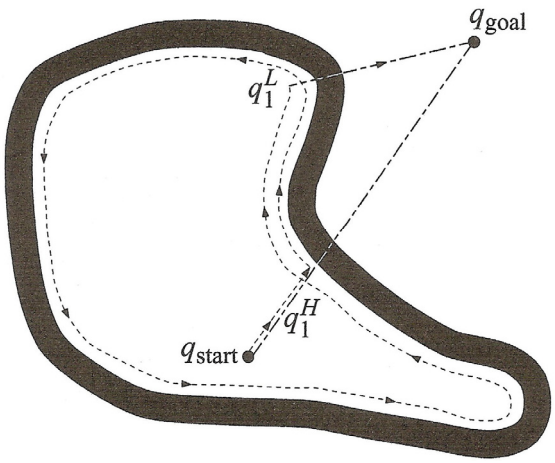



Figura 2.4. El algoritmo Bug1 reporta que la meta es inalcanzable**.**

**Algoritmo 1:** Algoritmo Bug1
**Entrada:** un punto del robot con un sensor táctil
**Salida:** una trayectoria hacia $q_{goal}$ o una conclusión de que la trayectoria no existe

1: **mientras** Por_Siempre **hacer**
2: **repetir**
3: Desde $q_{i-1}^L$, moverse hasta $q_{goal}$
4: **hasta** se alcance $q_{goal}$ **o** se encuentre un obstáculo en $q_i^H$
5: **si** se llega a $q_{goal}$ **entonces**

| | |
|---|---|
| 6: | Salir |
| 7: | **fin si** |
| 8: | **repetir** |
| 9: | Seguir la frontera del obstáculo |
| 10: | **hasta** se alcance $q_{\text{goal}}$ **o** se re-encuentre $q_i^H$ |
| 11: | Determinar el punto $q_i^L$ en el perímetro del obstáculo que tenga la distancia más corta hacia la meta. |
| 12: | Ir hacia $q_i^L$ |
| 13: | **si** el robot fuera a moverse hacia la meta **entonces** |
| 14: | Concluir que $q_{\text{goal}}$ no es alcanzable y salir |
| 15: | **fin si** |
| 16: | **fin mientras** |

Figura 2.5. Algoritmo Bug 1

Semejante al algoritmo Bug1, en el algoritmo Bug2, el robot se mueve hacia la meta sobre la recta *m*; sin embargo, en el Bug2 la recta *m* que conecta a $q_{\text{start}}$ y $q_{\text{goal}}$ permanece fija. Se invoca un seguimiento de frontera si se encuentra un obstáculo, pero este comportamiento es diferente del Bug1. Para el Bug2, el robot circunnavega el obstáculo hasta que este alcanza un nuevo punto en la recta *m* más cercano a la meta que el punto inicial de contacto con el obstáculo. Así, el robot continúa hacia la meta, repitiendo este proceso si este encuentra un objeto (figura 2.6). Si el robot encuentra el punto de partida original de la recta *m*, entonces no existe una trayectoria hacia la meta (figura 2.7).

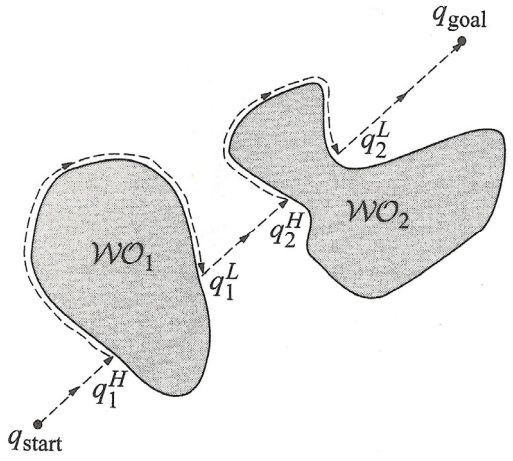



Figura 2.6. El algoritmo Bug2 encuentra la meta.

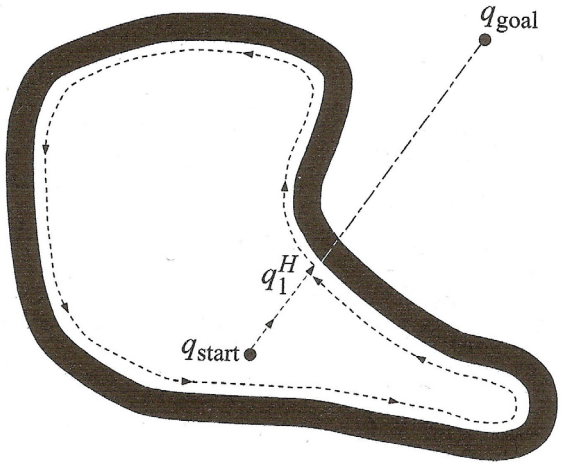



Figura 2.7. El algoritmo Bug2 reporta que la meta es inalcanzable.

Sea $\boldsymbol{x} \in W_{free} \subset \mathbb{R}^2$ la posición actual del robot, $i = 1$, y $q_0^L$ el punto de comienzo. El algoritmo 2 (figura 2.8) muestra una descripción general del método Bug2.

**Algoritmo 2:** Algoritmo Bug2
**Entrada:** un punto del robot con un sensor táctil
**Salida:** una trayectoria hacia $q_{\text{goal}}$ o una conclusión de que la trayectoria no existe

1: **mientras** Verdadero **hacer**
2: **repetir**
3: Desde $q_{i-1}^L$, moverse hacia $q_{\text{goal}}$ a lo largo de la recta *m*
4: **hasta** se alcance $q_{\text{goal}}$ **or** se encuentre un obstáculo en $q_i^H$
5: Girar a la izquierda, o a la derecha
6: **repetir**
7: Seguir la frontera del obstáculo
**hasta** se alcance $q_{\text{goal}}$ **o** se re-encuentre $q_i^H$ **o** se re-encuentre la recta *m* en un punto *m* tal que
8: $m \neq q_i^H$ (el robot no alcanza el punto de choque), $d(m, q_{\text{goal}}) < d(m, q_i^H)$ (el robot esta más cerca), y si el robot se mueve hacia la meta, este no debe golpear el obstáculo.
9: Hacer que $q_{i+1}^L = m$
10: Incrementar *i*
11: **fin mientras**

Figura 2.8. Algoritmo Bug 2

A primera vista, el algoritmo Bug2 parece ser más efectivo que el Bug1 por que el robot no tiene que circunnavegar completamente los obstáculos; sin embargo, este no es siempre el caso. Esto puede ser visto en la comparación de las longitudes de las trayectorias encontradas por los dos algoritmos. Para el Bug1, cuando el *i*-ésimo obstáculo es encontrado, el robot circunnavega completamente la frontera, y entonces regresa al punto de partida. En el peor caso, el robot debe viajar a través de la mitad del perímetro $p_i$ del obstáculo para alcanzar este punto de partida. Además, en el peor caso, el robot encuentra todos los *n* obstáculos. Si no hay ningún obstáculo, el robot debe recorrer una distancia de longitud $d(q_{\text{start}}, q_{\text{goal}})$. Entonces, se obtiene:

$$L_{Bug1} \leq d(q_{\text{start}}, q_{\text{goal}}) + \frac{3}{2}\sum_{i=1}^{n} p_i \qquad (10)$$

Para el Bug2, la longitud de la trayectoria es un poco más complicada. Suponiendo que la recta a través de $q_{\text{start}}$ y $q_{\text{goal}}$ interseca el $i$-ésimo obstáculo $n_i$ veces. Entonces, existen como máximo $n_i$ puntos de partida para este obstáculo, puesto que el robot puede dejar este obstáculo solo cuando regresa a un punto en esta recta. Es fácil ver que la mitad de estos puntos de intersección no son puntos de partida válidos por que ellos yacen en el lado erróneo del obstáculo, esto es, moverse hacia la meta podría causar una colisión. En el peor caso, el robot recorrerá cerca del perímetro entero del obstáculo por cada punto de partida. Entonces se obtiene:

$$L_{Bug2} \leq d(q_{\text{start}}, q_{\text{goal}}) + \frac{1}{2}\sum_{i=1}^{n} n_i p_i \qquad (11)$$

Un análisis de (10) y (11) muestra que $L_{Bug2}$ puede ser arbitrariamente más largo que $L_{Bug1}$. Esto se puede lograr por la construcción de un obstáculo cuya frontera tiene muchas intersecciones con la recta $m$. Entonces, como la complejidad del obstáculo se incrementa, es incrementalmente probable que el Bug1 pueda superar al Bug2 (figura 2.9).

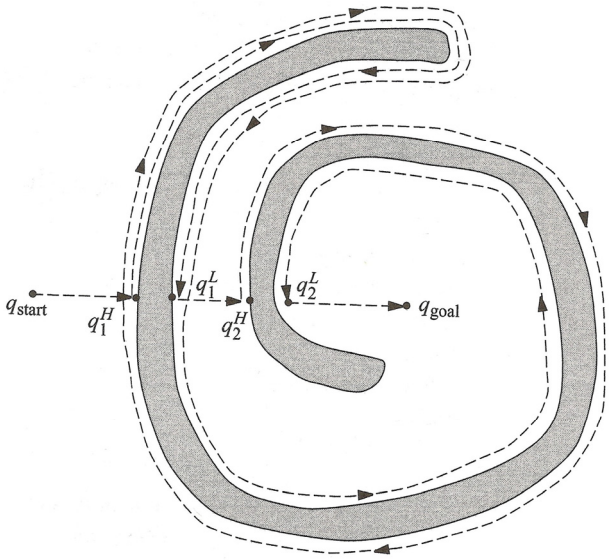



Figura 2.9. Análisis de l algoritmo Bug2

Bug1 y Bug2 ilustran dos métodos básicos para problemas de búsqueda. Para cada obstáculo que se encuentra, Bug1 desarrolla una búsqueda exhaustiva para encontrar el punto de partida óptimo. Esto requiere que el Bug1 recorra el perímetro entero del obstáculo, pero una vez hecho esto, es una certeza que habrá encontrado el punto de partida óptimo. En contraste, Bug2 utiliza un método oportunista. Cuando Bug2 encuentra un punto de partida que es mejor que cualquiera que ha visto antes, este ejecuta hasta ese punto de partida. Tal algoritmo es llamado también codicioso, a partir de que opta por la primera opción prometedora que es encontrada. Cuando los obstáculos son simples, el método codicioso del Bug2 da una resultado rápido, pero cuando los obstáculos son complejos, el método conservativo del Bug1 a menudo produce mejor desempeño.

*Algoritmo Bug Tangente*

El método del Bug Tangente [Kamon, Rivlin y Rimon 1996] sirve como un mejoramiento del algoritmo Bug2 por que este determina una trayectoria más corta hacia la meta utilizando un sensor de rango con 360 grados de resolución de orientación. Algunas veces la orientación es llamada acimut. Este sensor de rango puede ser modelado con la función de distancia simple (raw distance function) $\rho : \mathbb{R}^2 \times S^1 \to \mathbb{R}$. Considere el robot como un punto situado en $\boldsymbol{x} \in \mathbb{R}^2$ con rayos radiales emanando de él. Para cada $\theta \in S^1$, el valor $\rho(\boldsymbol{x}, \theta)$ es la distancia al obstáculo más cercano a lo largo del rayo de $\boldsymbol{x}$ en un ángulo $\theta$. Más formalmente:

$$\rho(\boldsymbol{x}, \theta) = \min_{\lambda \in [0, \infty]} d(\boldsymbol{x}, \boldsymbol{x} + \lambda[\cos\theta, \sin\theta]^T) \tag{12}$$

De manera que

$$\boldsymbol{x} + \lambda[\cos\theta, \sin\theta]^T \in \bigcup_i \boldsymbol{WO}_i$$

Note que existen muchos $\theta \in S^1$ y ahí la resolución infinita.

Esta es una suposición aproximada con un número finito de sensores de rango situados a lo largo de la circunferencia de un robot móvil circular el cuál es modelado como un punto.

Puesto que los sensores reales tienen rangos limitados, se define la función de distancia simple saturada (saturated raw distance function), denotada $\rho_R : \mathbb{R}^2 \times S^1 \to \mathbb{R}$, la cual toma los mismos valores de $\rho$ cuando el obstáculo esta dentro del rango de los sensores, y tiene un valor de infinito cuando las longitudes del rayo son más grandes que el rango de los sensores, $\boldsymbol{R}$, significando que los obstáculos están fuera del rango de los sensores. Más formalmente:

$$\rho_R(\boldsymbol{x},\theta) = \begin{cases} \rho(\boldsymbol{x},\theta), & \text{si } \rho(\boldsymbol{x},\theta) < \boldsymbol{R} \\ \infty, & \text{de otra forma} \end{cases} \tag{13}$$

El planeador del Bug tangente asume que el robot puede detectar discontinuidades en $\rho_R$ como se muestra en la figura 2.10. Para un $\boldsymbol{x} \in \mathbb{R}^2$ fijo, se define un intervalo de continuidad para conectar un conjunto de puntos $\boldsymbol{x} + \rho(\boldsymbol{x},\theta)[\cos\theta, sen\theta]^T$ en la frontera del espacio libre donde $\rho_R(\boldsymbol{x},\theta)$ es finito y continuo con respecto a $\theta$.

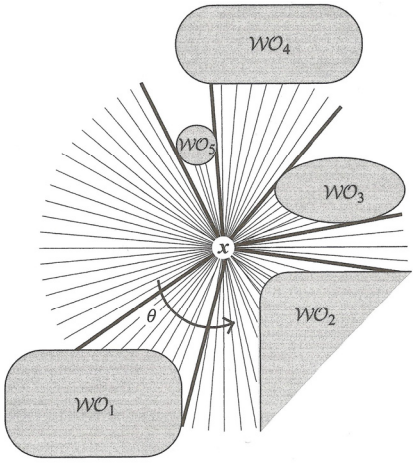



Figura 2.10. Detección de discontinuidades en $\rho_R(\boldsymbol{x},\theta)$.

Los límites de estos intervalos ocurren donde $\rho_R(\boldsymbol{x},\theta)$ pierde continuidad, o como un resultado de un obstáculo bloqueando otro obstáculo, o cuando el sensor alcanza su límite de

rango. Los límites están denotados como $O_i$. La figura 2.11 contiene un ejemplo donde $\rho_R$ pierde la continuidad. Los puntos $O_1$, $O_2$, $O_3$, $O_5$, $O_6$, $O_7$, y $O_8$ corresponden a la pérdida de continuidad asociada con los obstáculos bloqueando otras porciones de $W_{free}$; aquí los rayos son tangentes a los obstáculos. El punto $O_4$ es una discontinuidad porque la frontera del obstáculo cae fuera del rango del sensor. Los conjuntos de puntos en la frontera del espacio libre entre $O_1$ y $O_2$, $O_3$ y $O_4$, $O_5$ y $O_6$, $O_7$ y $O_8$, son los intervalos de continuidad.

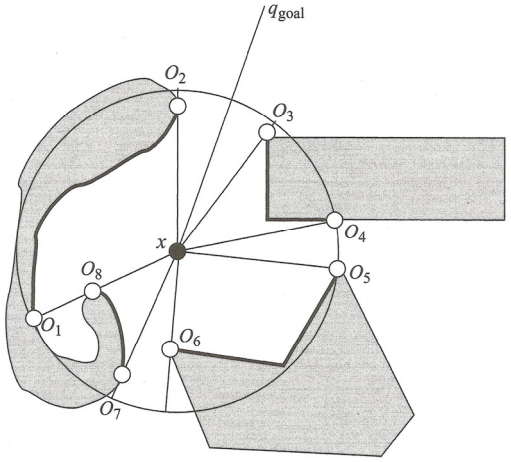



Figura 2.11. Puntos de discontinuidad de $\rho_R(\boldsymbol{x}, \theta)$ .

Al igual que los otros Bugs, el Bug Tangente interactúa entre dos comportamientos: movimiento hacia la meta y seguimiento de frontera. Sin embargo, estos comportamientos difieren de los métodos Bug1 y Bug2. Aunque el movimiento hacia la meta dirige el robot hacia la meta, este comportamiento puede tener una fase donde el robot siga la frontera. Asimismo, el comportamiento de seguimiento de frontera puede tener una fase donde el robot no siga la frontera.

El robot inicialmente invoca el comportamiento de movimiento hacia la meta, el cual tiene dos partes. Primero, el robot se mueve en línea recta hacia la meta hasta que los sensores detectan un obstáculo a $\boldsymbol{R}$ unidades de distancia y directamente entre este y la meta. Esto significa que un segmento de recta que conecta al robot y a la meta debe intersecar un intervalo de continuidad. Por ejemplo, en la figura 2.12, $\boldsymbol{WO_2}$ está dentro del rango de los sensores, pero no

bloquea la meta, sino que $\boldsymbol{WO_1}$ lo hace. Cuando el sensor del robot inicialmente detecta un obstáculo, el círculo de radio $\boldsymbol{R}$ se vuelve tangente al obstáculo. Inmediatamente después, este punto tangente se divide en dos $O_i$, los cuales son los límites del intervalo. Si el obstáculo está enfrente del robot, entonces este intervalo interseca el segmento que conecta al robot y la meta. El robot entonces se mueve hacia el $O_i$ que máximamente disminuya una distancia heurística hacia la meta. Un ejemplo de una distancia heurística es la suma $d(\boldsymbol{x}, O_i) + d(O_i, q_{\text{goal}})$.

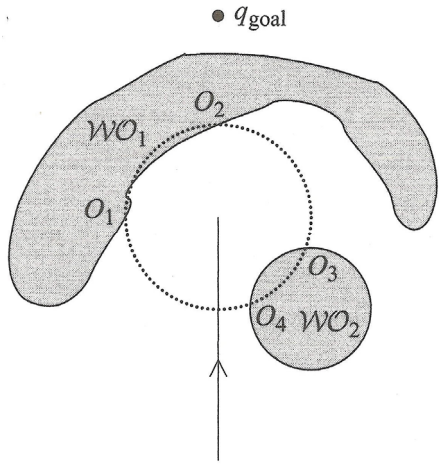



Figura 2.12. Representación de la trayectoria y el rango de los sensores del robot.

En la figura 2.13 (a), el robot percibe a $\boldsymbol{WO_1}$ y se dirige hacia $O_2$ por que $i = 2$ minimiza $d(\boldsymbol{x}, O_i) + d(O_i, q_{\text{goal}})$. Cuando el robot está situado en $\boldsymbol{x}$, no puede saber que $\boldsymbol{WO_2}$ bloquea la trayectoria de $O_2$ hacia la meta. En la figura 2.13 (b), cuando el robot está situado en $\boldsymbol{x}$ pero la meta es diferente, este tiene suficiente información del sensor para concluir que $\boldsymbol{WO_2}$ efectivamente bloquea una trayectoria de $O_2$ hacia la meta, y por lo tanto, se dirige hacia $O_4$.

Entonces, aunque dirigiéndose hacia $O_2$ puede inicialmente minimizar $d(\boldsymbol{x}, O_i) + d(O_i, q_{\text{goal}})$ más que dirigiéndose hacia $O_4$, el planeador asignará efectivamente un costo infinito a $d(O_2, q_{\text{goal}})$ por que este tiene suficiente información para concluir que cualquier trayectoria a través de $O_2$ será subóptima.

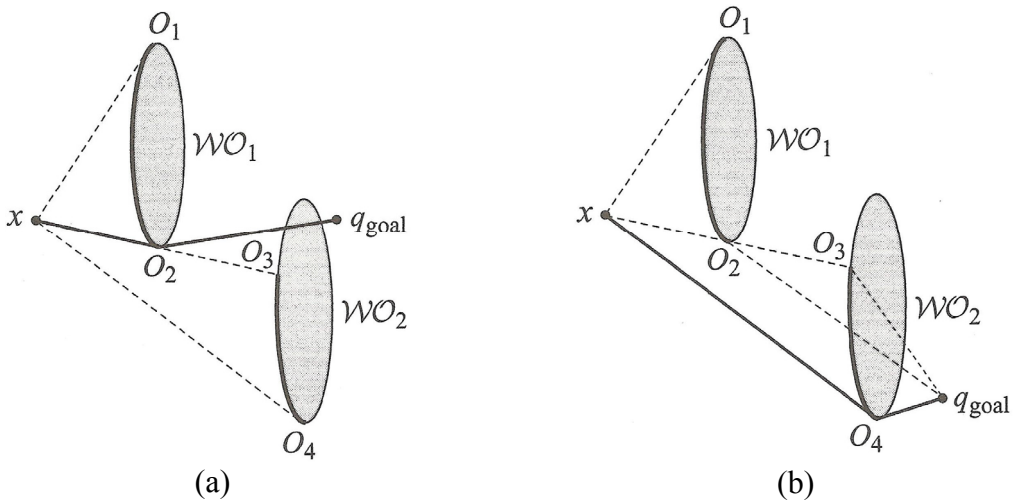



Figura 2.13. Selección de submetas $O_2$ (a) y $O_4$ (b) para el robot.

El conjunto $\{O_i\}$ se actualiza continuamente cuando el robot se mueve hacia un $O_i$ particular, como se observa en la figura 2.14. Cuando $t = 1$, el robot no ha sensado el obstáculo, por lo que el robot se mueve hacia la meta. Cuando $t = 2$, el robot inicialmente sensa el obstáculo, representado por una curva sólida gruesa. El robot continua moviéndose hacia la meta, pero fuera del lado del obstáculo dirigiéndose hacia la discontinuidad en $\rho$. Para $t = 3$ y $t = 4$, el robot sensa más del obstáculo y continua decrementando la distancia hacia la meta mientras se mantiene cerca de la frontera del obstáculo.

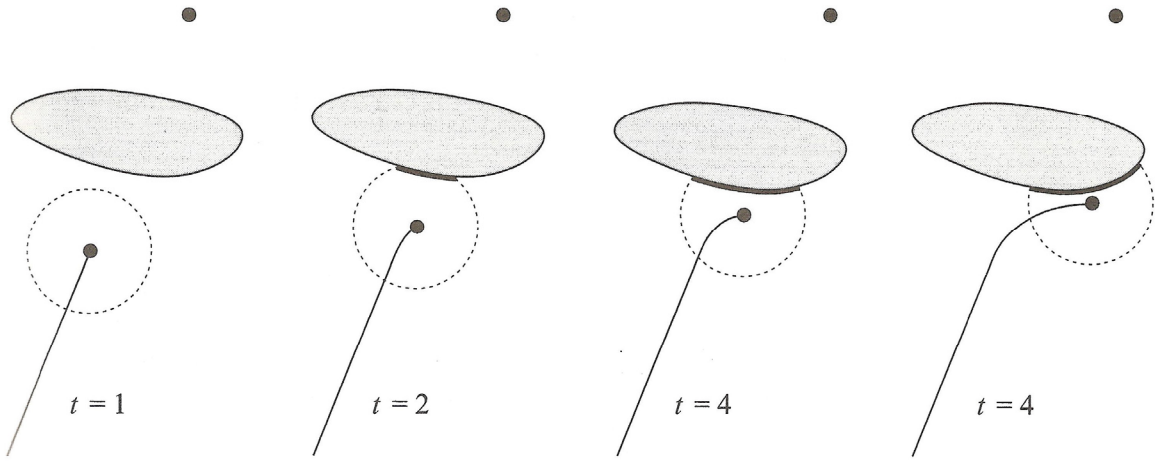



Figura 2.14. Movimiento hacia la meta de un robot con sensor de rango finito.

El robot experimenta el comportamiento de movimiento hacia la meta hasta que no puede decrementar más la distancia hacia la meta. Visto de forma diferente, el robot encuentra un

punto que es como un mínimo local de $d(\cdot, O_i) + d(O_i, q_{\text{goal}})$ restringido a la trayectoria que dicta el movimiento hacia la meta. Cuando el robot cambia del movimiento hacia la meta al seguimiento de frontera, este encuentra el punto *M* en la porción sensada del obstáculo, la cual tiene la distancia más corta del obstáculo hacia la meta. Note que si el rango del sensor es cero, entonces *M* es el mismo como el punto de golpe de los algoritmos Bug1 y Bug2. Al obstáculo que fue sensado se le llama también obstáculo seguido. Debe haber una distinción entre el obstáculo seguido y el obstáculo bloqueado. Sea $\boldsymbol{x}$ la posición actual del robot. El obstáculo bloqueado es el obstáculo más cercano dentro del rango del sensor que interseca el segmento $(1-\lambda)\boldsymbol{x} + \lambda q_{\text{goal}} \quad \forall \lambda \in [0,1]$. Inicialmente, el obstáculo bloqueado y el obstáculo seguido son el mismo.

Ahora el robot se mueve en la misma dirección como si este estuviera en el comportamiento de movimiento hacia la meta. Este se mueve continuamente hacia $O_i$ en el obstáculo seguido en la dirección elegida (figura 2.15). Mientras experimenta este movimiento, el planeador también actualiza dos valores: $d_{\text{followed}}$ y $d_{\text{reach}}$. El valor $d_{\text{followed}}$ es la distancia más corta entre la frontera sensada y la meta.

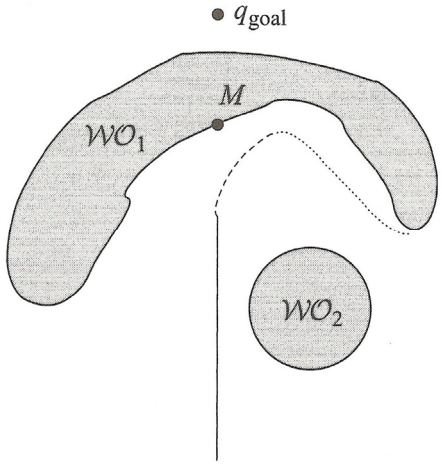



Figura 2.15. Movimiento hacia la meta y seguimiento de frontera del robot.

Sea $\Lambda$ todos los puntos dentro de la línea de visión de $\boldsymbol{x}$ con rango *R* que están en el obstáculo seguido $\boldsymbol{WO_f}$, esto es $\Lambda = \{\boldsymbol{y} \in \partial \boldsymbol{WO}_f : \lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in Q_{\text{free}} \quad \forall \lambda \in [0,1]\}$. El valor $d_{\text{reach}}$ es la

distancia entre la meta y el punto más cercano en el siguiente obstáculo que está dentro de la línea de visión del robot, esto es $d_{\text{reach}} = \min_{c \in \Lambda} d(q_{\text{goal}}, c)$. Cuando $d_{\text{reach}} < d_{\text{followed}}$, el robot termina el comportamiento de seguimiento de frontera.

Sea $T$ el punto donde un círculo centrado en $\boldsymbol{x}$ de radio $R$ interseca el segmento que conecta a $\boldsymbol{x}$ y $q_{\text{goal}}$. Este es el punto en la periferia del rango de sensado que es el más cercano a la meta cuando el robot está localizado en $\boldsymbol{x}$. Comenzando con $\boldsymbol{x} = q_{\text{start}}$ y $d_{\text{leave}} = (q_{\text{start}}, q_{\text{goal}})$, como se muestra en el algoritmo 3 (figura 2.16).

**Algoritmo 3:** Algoritmo Bug Tangente
**Entrada:** un punto del robot con un sensor de rango
**Salida:** una trayectoria hacia $q_{\text{goal}}$ o una conclusión de que la trayectoria no existe

1: **mientras** Verdadero **hacer**
2: **repetir**
3: Moverse continuamente hacia el punto $n \in \{T, O_i\}$ el cual minimiza $d(\boldsymbol{x}, n) + d(n, q_{\text{goal}})$
**hasta** la meta sea encontrada **or** la dirección que minimiza $d(\boldsymbol{x}, n) + d(n, q_{\text{goal}})$
4: comience a incrementar $d(\boldsymbol{x}, q_{\text{goal}})$, esto es, el robot detecta un mínimo local de $d(\cdot, q_{\text{goal}})$.
5: Elegir una dirección de seguimiento de frontera que continúe en la misma dirección como la más reciente dirección de movimiento hacia la meta.
6: **repetir**
7: Actualizar continuamente $d_{\text{reach}}$, $d_{\text{followed}}$, y $\{O_i\}$
8: Moverse continuamente hacia $n \in \{O_i\}$ que está en la dirección de la frontera elegida.
9: **hasta** se alcance $q_{\text{goal}}$ **or** el robot complete un ciclo alrededor del obstáculo, en tal caso la meta no puede ser lograda **or** $d_{\text{reach}} < d_{\text{followed}}$
10: **fin mientras**

Figura 2.16. Algoritmo Bug Tangente

La figura 2.17 contiene una trayectoria para un robot con sensor de rango cero. Aquí el robot invoca un comportamiento de movimiento hacia la meta hasta que encuentra el primer obstáculo en el punto de choque $H_1$. A diferencia del Bug1 y Bug2, encontrar un punto de choque no cambia el modo de comportamiento del robot. El robot continúa con el comportamiento de movimiento hacia la meta girando a la derecha y siguiendo la frontera del primer obstáculo. El robot gira a la derecha por que esa dirección minimiza su distancia heurística hacia la meta. El robot parte de esta frontera en el punto de salida $D_1$. El robot continúa con el comportamiento de movimiento hacia la meta, maniobrando alrededor de un

segundo obstáculo hasta que encuentra el tercer obstáculo en $H_3$. El robot gira a la izquierda y continúa invocando el comportamiento de movimiento hacia la meta hasta que alcanza $M_3$, un punto mínimo. Entonces el planeador invoca el comportamiento de seguimiento de frontera hasta que el robot alcanza $L_3$. Nótese que desde que se tiene un sensor de rango cero, $d_{\text{reach}}$ es la distancia entre el robot y la meta. El procedimiento continúa hasta que el robot alcanza la meta. Sólo en $M_i$ y $L_i$ el robot intercambia comportamientos. Las figuras 2.18 y 2.19 contienen ejemplos donde el robot tiene rangos de sensores finitos e infinitos, respectivamente.

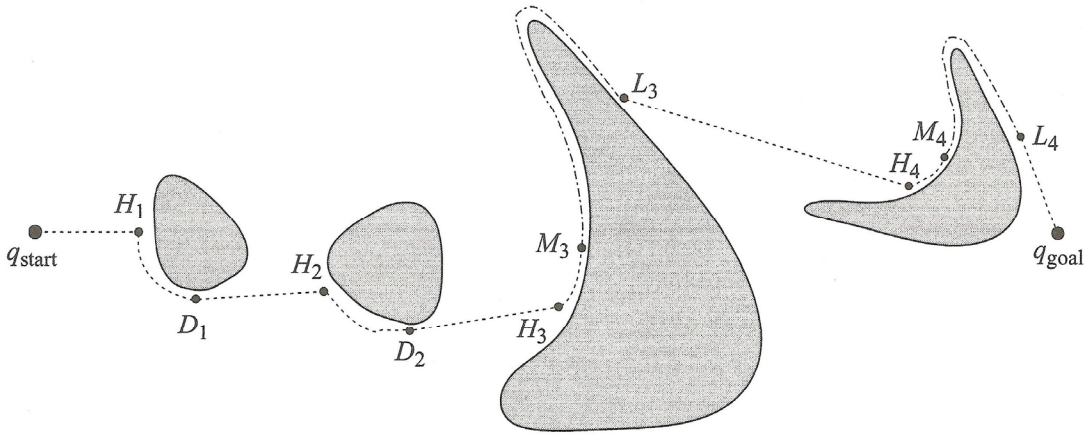



Figura 2.17. La trayectoria generada por el Bug Tangente con rango de sensor cero.

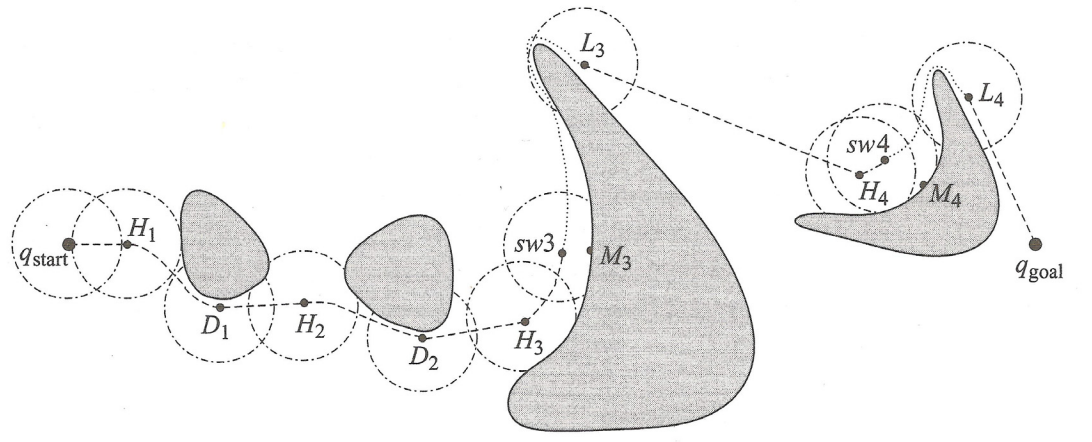



Figura 2.18. La trayectoria generada por el Bug tangente con rango de sensor finito.

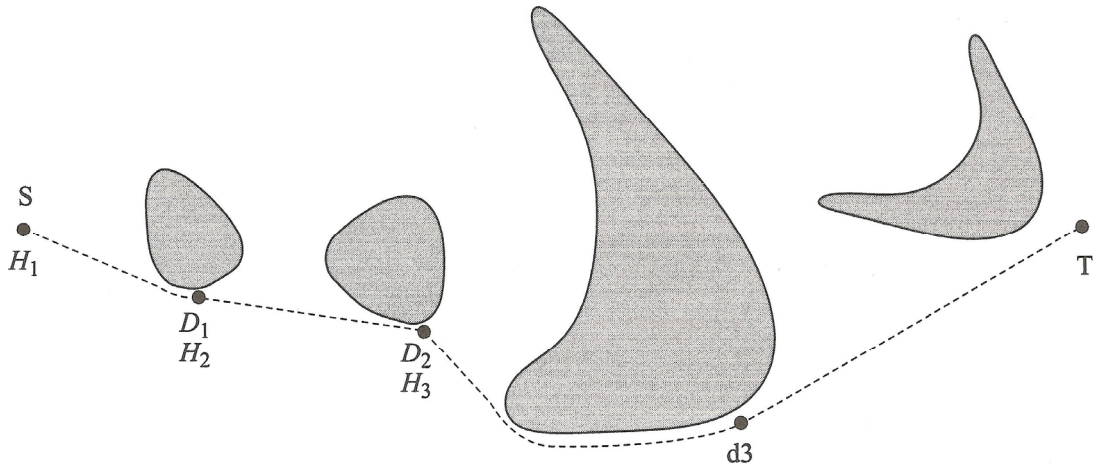



Figura 2.19. Trayectoria generada por el Bug tangente con rango de sensor infinito.

**Implementación del algoritmo Bug**

Esencialmente, los algoritmos Bug tienen dos comportamientos: dirigirse hacia un punto y seguir un obstáculo. El primer comportamiento es simplemente una forma de gradiente descendente de $d(\cdot, n)$ donde $n$ es $q_{\text{goal}}$ o un $O_i$. El segundo comportamiento, seguimiento de frontera, representa un reto por que la frontera del obstáculo no es conocida *a priori*. Por consiguiente, el planeador del robot debe basarse en información del sensor para determinar la ruta. Sin embargo, se debe admitir que la trayectoria completa hacia la meta no es determinada por una lectura del sensor: el rango del sensor del robot puede ser limitado y el robot puede no ser capaz de ver el mundo entero desde un punto ventajoso. Entonces, el planeador del robot tiene que ser incremental. Se debe determinar primero que información requiere el robot y entonces hacia donde se debe mover el robot para adquirir más información. Este es de hecho el reto de la planeación basada en sensores. Idealmente, se desea que este método sea reactivo con información de sensado alimentando un algoritmo simple cuya salida sea una velocidad de translación y rotación para el robot. Existen tres preguntas: ¿Qué información requiere el robot para circunnavegar el obstáculo? ¿Cómo infiere el robot esta información de los datos del sensor? ¿Cómo utiliza el robot esta información para determinar (localmente) una trayectoria?

**La línea tangente**

Si el obstáculo fuera plano, tal como un muro en un corredor, entonces seguir el obstáculo consistiría en mover el robot en paralelo al obstáculo. Esto se implementa utilizando un

sistema de sensores que pueda determinar la superficie normal del obstáculo $n(\boldsymbol{x})$, y por lo tanto una dirección paralela a esta superficie. Sin embargo, el mundo no esta poblado con obstáculos planos; muchos tienen curvatura no cero. El robot puede seguir una trayectoria que es consistentemente ortogonal a la superficie normal; esta dirección puede ser escrita como $n(\boldsymbol{x})^{\perp}$ y la trayectoria resultante satisface a $\dot{c}(t) = v$ donde $v$ es un vector base en $(n(c(t)))^{\perp}$. El signo de $v$ esta basado en la dirección previa de $\dot{c}$. Regularmente, determinar la superficie normal puede ser muy retador y por lo tanto, para la implementación, se puede asumir que los obstáculos son "localmente planos". Esto significa que el sistema de sensores determina la superficie normal, el robot se mueve ortogonalmente hacia esta normal por una distancia corta, y entonces el proceso se repite. En un sentido, el robot determina la secuencia de los segmentos de línea cortos a seguir, basado en la información del sensor. Esta línea plana, es la tangente (figura 2.20).

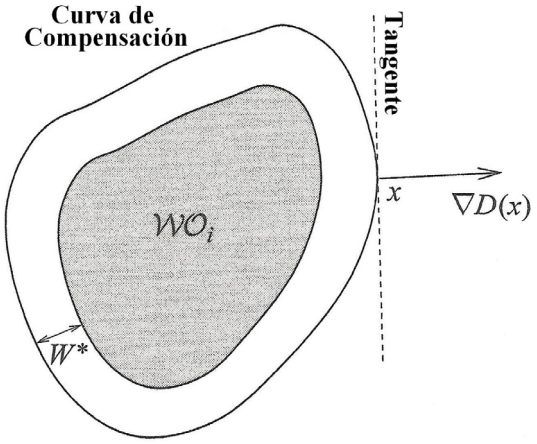



Figura 2.20. La línea guión representa la tangente a la curva de compensación en *x*.

Es una aproximación lineal de la curva al punto donde la tangente interseca la curva. La tangente puede ser también vista como una aproximación de primer orden a la función que describe la curva. Sea $c:[0,1] \to W_{\text{free}}$ la función que define una trayectoria. Sea $\boldsymbol{x} = c(S_0)$ para una $s_0 \in [0,1]$. La tangente en $\boldsymbol{x}$ es $\frac{dc}{ds}|_{s=s_0}$. El espacio de la tangente puede ser visto como una línea cuyo vector base es $\frac{dc}{ds}|_{s=s_0}$, esto es $\left\{\alpha \frac{dc}{ds}|_{s=s_0} | \alpha \in \mathbb{R}\right\}$.

**Como inferir información con los sensores: distancia y gradiente**

El siguiente paso es inferir la tangente del sensor de datos. En lugar de pensar en el robot como un punto en el plano, tómese como una base circular que tiene un arreglo de sensores táctiles radialmente distribuidos a lo largo de su circunferencia (figura 2.21). Cuando el robot contacta un obstáculo, la dirección del sensor contactado hacia el centro del robot aproxima la superficie normal. Con esta información, el robot puede determinar una secuencia de tangentes para seguir el obstáculo.

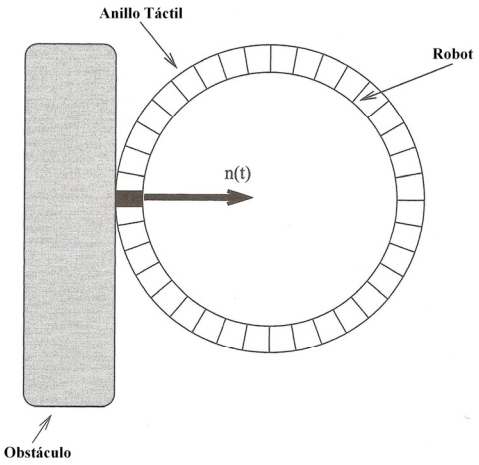



Figura 2.21. Una resolución fina del sensor táctil.

Desafortunadamente, utilizar un sensor táctil para prescribir una trayectoria requiere una colisión entre el robot y los obstáculos, los cual pone en peligro los obstáculos y al robot. En lugar de esto, el robot debe seguir una trayectoria en una distancia segura $W^* \in \mathbb{R}$ desde el obstáculo más cercano. Tal trayectoria es llamada curva de compensación. Sea $D(\boldsymbol{x})$ la distancia de $\boldsymbol{x}$ al obstáculo más cercano, esto es:

$$D(\boldsymbol{x}) = \min_{c \in \bigcup_i \boldsymbol{WO}_i} d(\boldsymbol{x}, c) \tag{14}$$

Para medir esta distancia con un robot móvil equipado con un anillo como sensor de rango a bordo, se utiliza la función de distancia simple de nuevo. Sin embargo, en lugar de buscar discontinuidades, se busca el mínimo global. En otras palabras, $D(\boldsymbol{x}) = \min_s \rho(\boldsymbol{x}, s)$ (como se

muestra en la figura 2.22). Se necesitará utilizar la gradiente de distancia. En general, la gradiente es un vector que apunta en la dirección que incrementa de manera máxima el valor de una función. Típicamente, el i-ésimo componente del vector gradiente es la derivada parcial de la función con respecto a su i-ésima coordenada. En el plano, $\nabla D(\boldsymbol{x}) = \left[\frac{\partial D(\boldsymbol{x})}{\partial \boldsymbol{x}_1} \frac{\partial D(\boldsymbol{x})}{\partial \boldsymbol{x}_2}\right]^T$ la cual apunta en la dirección que incrementa la distancia al máximo.

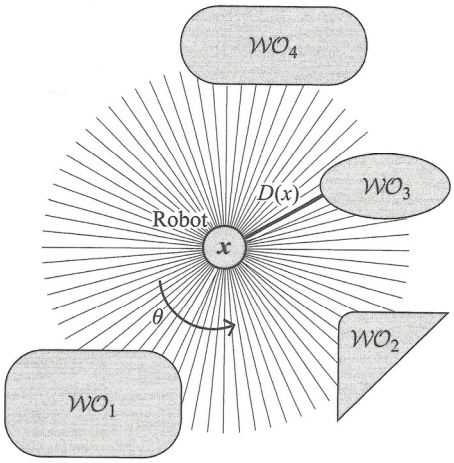



Figura 2.22. Búsqueda del mínimo global.

Finalmente, la gradiente es la dirección de unidad asociada con el más pequeño valor de la función de distancia simple. A partir de que la función de distancia simple aparentemente aproxima un sistema de sensado con rango individual sensando elementos radialmente distribuidos alrededor del perímetro del robot, se puede implementar un algoritmo definido en términos de *D* utilizando sensores reales.

Los sensores ultrasónicos convencionales miden la distancia utilizando tiempo de vuelo. Cuando la velocidad del sonido en el aire es constante, el tiempo que el ultrasonido requiere para dejar el transductor, golpea un obstáculo, y regresa es proporcional a la distancia hacia el punto de reflexión en el obstáculo. Este obstáculo, sin embargo, puede ser localizado en cualquier lado a lo largo de la propagación angular del patrón del sonar del rayo del sensor (figura 2.23).

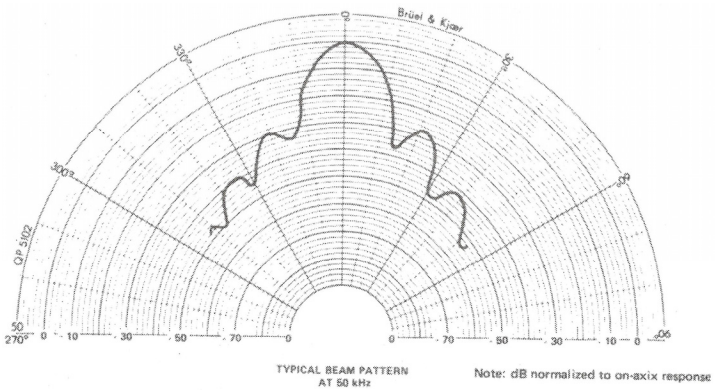



Figura 2.23. Patrón del rayo para un transductor

Por lo tanto, la información de distancia que los sonares proveen es precisa en profundidad, pero no en el azimut. El patrón del rayo puede ser aproximado con un cono (figura 2.24). Inicialmente, se asume que el eco se origina desde el centro del cono del sonar. A esto se le llama modelo de la línea central (figura 2.24).

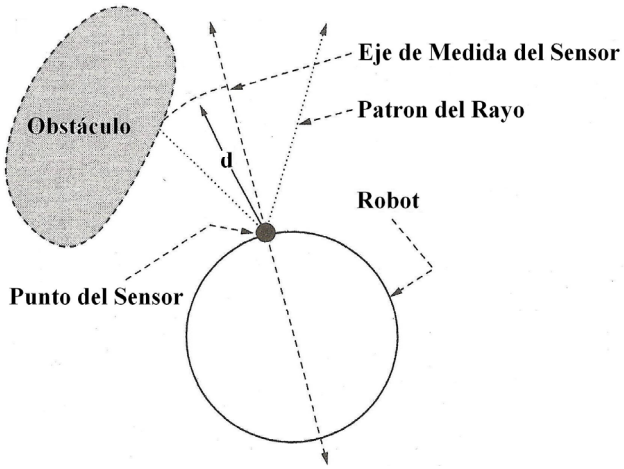



Figura 2.24. Modelo de la línea central

El sensor ultrasónico con la lectura más pequeña se aproxima al mínimo global de la función de distancia simple, y por lo tanto $D(\boldsymbol{x})$. La dirección a la que este sensor se orienta se aproxima a la gradiente negativa $-\nabla D(\boldsymbol{x})$ por que este sensor confronta el obstáculo más cercano. La tangente es entonces la línea ortogonal hacia la dirección asociada con la lectura más pequeña del sensor.

**Como procesar información del sensor**

La tangente de la curva de compensación es $(\nabla D(\boldsymbol{x}))^{\perp}$, la línea ortogonal a $\nabla D(\boldsymbol{x})$ (figura 2.20). El vector $\nabla D(\boldsymbol{x})$ apunta en la dirección que incrementa de manera máxima la distancia; así mismo, el vector $-\nabla D(\boldsymbol{x})$ apunta en la dirección que decrementa de manera máxima la distancia; ambos vectores apuntan en la misma línea pero en direcciones opuestas. En consecuencia, el vector $(\nabla D(\boldsymbol{x}))^{\perp}$ apunta en la dirección que mantiene localmente la distancia; esta es perpendicular a ambos $\nabla D(\boldsymbol{x})$ y $-\nabla D(\boldsymbol{x})$. Este sería la tangente de la curva de compensación que mantiene la distancia a un obstáculo cercano.

Otra forma de ver por que $(\nabla D(\boldsymbol{x}))^{\perp}$ es la tangente es observar a la definición de la curva de compensación. Para una distancia segura $\boldsymbol{W}^*$, se puede definir la curva de compensación implícitamente como el conjunto de puntos donde $G(\boldsymbol{x}) = D(\boldsymbol{x}) - \boldsymbol{W}^*$ mapea a cero. El conjunto de puntos no cero (o vectores) que mapea a cero es llamado el espacio nulo de un mapa. Para una curva implícitamente definida por *G*, el espacio de la tangente en un punto $\boldsymbol{x}$ es el espacio nulo de $DG(\boldsymbol{x})$, el Jacobiano de *G*. En general, el *i*, *j*-ésimo componente de la matriz Jacobiana es la derivada parcial de la función del *i*-ésimo componente con respecto a la *j*-ésima coordenada y entonces el Jacobiano es un mapeo entre los espacios de la tangente. Puesto que en este caso, *G* es una función de valor real ($i = 1$), el Jacobiano es sólo un vector unidimensional $D\ D(\boldsymbol{x})$. En este contexto, se reutiliza el símbolo *D* y se debe identificar cuándo se utiliza como distancia y como diferencial.

En espacios Euclidianos, el *i*-ésimo componente de un Jacobiano unidimensional es igual al *i*-ésimo componente de la gradiente y entonces $\nabla D(\boldsymbol{x}) = (D\ D(\boldsymbol{x}))^T$. Por lo tanto, dado que el espacio de la tangente es el espacio nulo de $D\ D(\boldsymbol{x})$, la tangente para el seguimiento de frontera en el plano es la línea ortogonal para $\nabla D(\boldsymbol{x})$, esto es $(\nabla D(\boldsymbol{x}))^{\perp}$, y puede ser deducido de la información del sensor.

Utilizando información de la distancia, el robot puede determinar la dirección de la tangente hacia la curva de compensación.

Si los obstáculos son planos, entonces la curva de compensación también es plana, y con simplemente seguir la tangente es suficiente para seguir la frontera de un obstáculo desconocido. Si el obstáculo tiene curvatura, el robot se puede mover a lo largo de la tangente por una distancia corta, pero, dado que el obstáculo tiene curvatura, el robot no seguirá la curva de compensación, es decir, se separará de la curva de compensación. Para el re-acceso a la curva de compensación, el robot se mueve hacia o fuera del obstáculo hasta que alcanza la distancia de seguridad $\boldsymbol{W}^*$. Al hacer esto, el robot se mueve a lo largo de una recta definida por $\nabla D(\boldsymbol{x})$, la cual puede ser deducida de la información del sensor.

Esencialmente, el robot desarrolla un procedimiento numérico de predicción y corrección. El robot utiliza la tangente para predecir localmente la figura de la curva de compensación y entonces invoca un procedimiento de corrección una vez que la aproximación a la tangente no es válida. Nótese que el robot no sigue explícitamente la trayectoria, sino que en su lugar, ronda alrededor de ella resultando un muestreo de la trayectoria, no la trayectoria en sí (figura 2.25).

Un procedimiento de trazo numérico puede ser aquel que trace las raíces de la expresión $G(\boldsymbol{x}) = 0$, donde en este caso $G(\boldsymbol{x}) = D(\boldsymbol{x}) - \boldsymbol{W}^*$. Las técnicas de trazado de curvas numéricas se basan en el teorema de la función implícita que define localmente una curva que es implícitamente definida por un mapa $G : Y \times \mathbb{R} \to Y$. Específicamente, las raíces de $G$ definen localmente una curva parametrizada por $\lambda \in \mathbb{R}$.

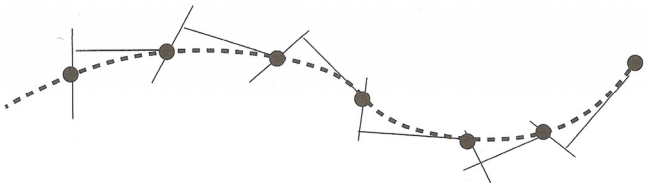


Figura 2.25. Procedimiento de predicción y corrección del robot.

Para el seguimiento de frontera en una distancia segura $\boldsymbol{W}^*$, la función $G(y,\lambda) = D(y,\lambda) - \boldsymbol{W}^*$ define implícitamente la curva de compensación. Nótese que la coordenada $\lambda$ corresponde a la dirección de la tangente y las coordenadas $y$ a la recta o hiper

plano ortogonal a la tangente. Si $Y$ denota este hiper plano y $D_yG$ la matriz formada por tomar la derivada de $G(\boldsymbol{x}) = D(\boldsymbol{x}) - \boldsymbol{W^*} = 0$ con respecto a las coordenadas $y$. Esto toma la forma $D_yG(\boldsymbol{x}) = D_yD(\boldsymbol{x})$ donde $D_y$ denota la diferencial con respecto a las coordenadas $y$. Si $D_yG(y, \lambda)$ es sobreyectivo en $\boldsymbol{x} = (\lambda, y)^T$, entonces el teorema de la función implícita establece que las raíces de $G(y, \lambda)$ definen una curva que sigue la frontera en una distancia $\boldsymbol{W^*}$ como $\lambda$ es variada, es decir, $y(\lambda)$.

Por trazar numéricamente las raíces de $G$, se puede construir localmente una trayectoria. Mientras existe un número de técnicas para el trazo de curvas, considérese una adaptación de un esquema predictor-corrector común. Si se asume que el robot está situado en el punto $\boldsymbol{x}$ que es una distancia fija $\boldsymbol{W^*}$ fuera de la frontera, el robot toma un pequeño paso, $\Delta\lambda$, en la dirección $\lambda$ (la tangente de la trayectoria local). En general, este paso de predicción conduce al robot fuera de la trayectoria inicial. Siguiente, un método de corrección es utilizado para traer al robot de regreso a la trayectoria inicial. Si $\Delta\lambda$ es pequeño, entonces la trayectoria local intersecará un plano de corrección, el cuál es un plano ortogonal para la dirección $\lambda$ en una distancia $\Delta\lambda$ fuera del origen. El paso de corrección busca el lugar en donde la trayectoria inicial interseca el plano de corrección y es una aplicación del teorema de convergencia de Newton. El teorema de convergencia de Newton también requiere que $D_YG(y,\lambda)$ sea completamente clasificado en cada $(y,\lambda)$ en un vecindario de la trayectoria inicial. Esto es verdadero por que $G(x) = D(\boldsymbol{x}) - \boldsymbol{W^*}$, $[0 \;\; D_YG(y,\lambda)]^T = DG(y,\lambda)$. Dado que $DG(y,\lambda)$ es completamente clasificado, entonces debe estar $D_YG(y,\lambda)$ en la curva de compensación.

Puesto que el conjunto de matrices no singulares es un conjunto abierto, se sabe que existe un vecindario alrededor de cada $(y,\lambda)$ en la trayectoria inicial donde $DG(y,\lambda)$ es completamente clasificado y por lo tanto se puede utilizar el método de Newton iterativo para implementar el paso corrector. Si $y^h$ y $\lambda^h$ son los $h$-ésimos estimados de $y$ y $\lambda$, la primera iteración $h + 1$ es definida como $y^{h+1} = y^h - (D_YG)^{-1}G(y^h, \lambda^h)$, donde $D^YG$ es evaluado en $(y^h, \lambda^h)$.

Note que dado que trabaja en un espacio Euclidiano, se puede determinar $D_Y G$ solamente de la distancia de la gradiente, y por lo tanto, de la información del sensor.

### 2.2.2. Método de descomposición trapezoidal.

Una forma de representar el espacio libre es mediante la descomposición de celdas exactas. Estas estructuras representan el espacio libre por la unión de simples regiones llamadas celdas. Las fronteras compartidas de las celdas a menudo tienen un significado físico como un cambio en un obstáculo cerrado o un cambio en una línea de visión para rodear un obstáculo. Dos celdas son adyacentes si ellas comparten una frontera común. Un grafo de adyacencia codifica la relación de adyacencia de las celdas, donde un nodo corresponde a una celda y un borde conecta a los nodos de las celdas adyacentes.

Asumiendo que la descomposición es computada, la planeación de rutas con una descomposición de celdas es usualmente hecha en dos pasos: primero, el planeador determina las celdas que contienen el comienzo y la meta, respectivamente, y entonces el planeador busca una ruta dentro del grafo de adyacencia. Nótese que el grafo de adyacencia también podría servir como un mapa de caminos del espacio libre. Por lo tanto, el mapeo podría ser logrado por construir incrementalmente el grafo de adyacencia.

El método de descomposición de celdas más popular es la descomposición trapezoidal [Preparata and Shamos 1985], este método comprende celdas de dos dimensiones que son diseñadas como trapezoides. Algunas celdas pueden ser diseñadas como triángulos, los cuáles se pueden ver como trapezoides degenerados donde uno de los lados paralelos tiene un borde de longitud cero. En un sistema de coordenadas ($x$, $y$) para el espacio de configuración en el plano, el espacio libre se limita por un polígono y todos los obstáculos son poligonales, esto es, cada vértice $v_i$ en todos los polígonos tiene una coordenada $x$ única, esto es, para toda $i \neq j, v_{i_x} \neq v_{j_x}$ (figura 2.26).

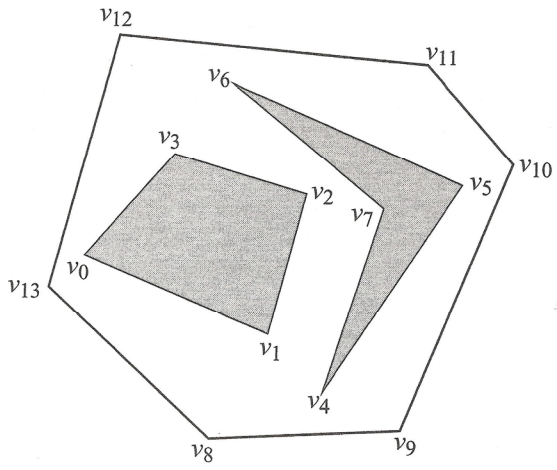



Figura 2.26. Ejemplo de un espacio de configuración poligonal

Para formar la descomposición, en cada vértice *v* se dibujan dos segmentos, uno llamado extensión vertical superior y el otro llamado extensión vertical inferior. Aquí, superior corresponde a incrementar la coordenada *y*, y similarmente, inferior significa un decremento. Las extensiones verticales superior e inferior comienzan en el vértice y terminan cuando ellos intersecan por primera vez un borde de un polígono que yace inmediatamente arriba y debajo de *v*, respectivamente. Nótese que muchos vértices tendrán solo una extensión vertical superior o inferior. La figura 2.27 contiene la descomposición trapezoidal y el grafo de adyacencia para el espacio de trabajo de la figura 2.26. Recuérdese que dos celdas son adyacentes si ellas comparten una frontera común, es decir, una extensión vertical.

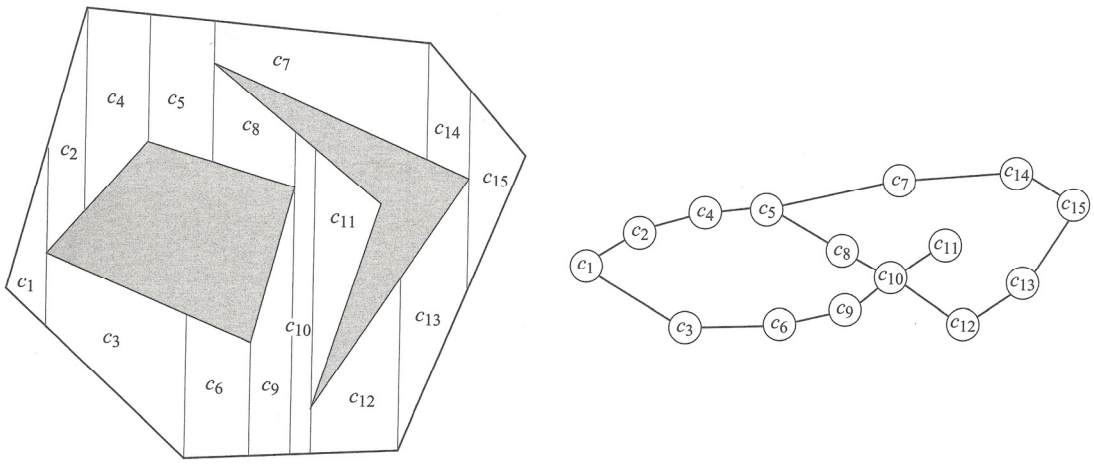



Figura 2.27. Descomposición trapezoidal para la configuración de la Figura 2.26

Una vez que las celdas de inicio y meta son determinadas, el planeador busca en el grafo de adyacencia para determinar la ruta. Sin embargo, el resultado de la búsqueda en el grafo es sólo una secuencia de nodos, no una secuencia de nodos incrustados en el espacio libre, y entonces el siguiente paso es determinar la ruta explícita. Dado que un trapezoide es un conjunto convexo, cualquier par de puntos en la frontera de una celda trapezoidal se puede conectar por un segmento de recta que no interseca ningún obstáculo. El planeador construye la ruta, un trapezoide a la vez, conectando los puntos medios de las extensiones verticales a los centroides de cada trapezoide. Esto mantiene una ruta conectada libre de colisiones a través del espacio libre que se deriva del grafo de adyacencia. Para conectar los puntos de inicio y meta, simplemente dibuje una recta en los puntos medios de las extensiones verticales del trapezoide apropiado (figura 2.28).

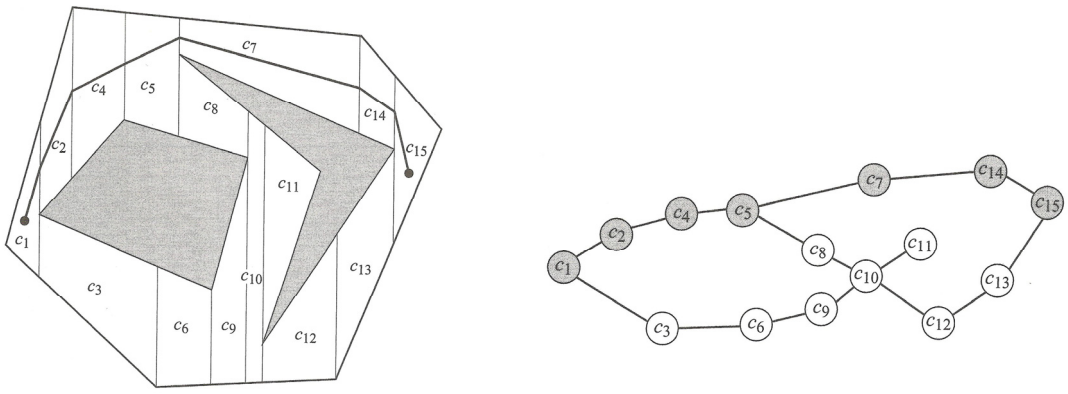



Figura 2.28. Trayectoria resultante en el grafo de adyacencia y espacio libre

### 2.2.3. Método del campo potencial artificial

El método del campo potencial artificial se propuso originalmente por [Kathib 1986] y es una de las técnicas más utilizadas por robots móviles para evitar colisiones locales y controlar el movimiento en tiempo real. El método del campo potencial utiliza una función escalar llamada la función potencial, la cual tiene un valor mínimo cuando el robot está en la configuración meta y tiene un valor alto en los obstáculos. En cualquier otro lado, el valor de la función baja conforme se acerca a la configuración meta, de manera que el robot puede alcanzar el objetivo por seguir la gradiente negativa del campo potencial.

Los altos valores del campo potencial previenen al robot de ir cerca de los obstáculos.

Si el robot se mueve en un espacio en $\mathbb{R}^2$, la generación de movimiento en un campo potencial artificial (Figura 2.29) se produce por:

1) una fuerza potencial atractiva que se genera por la meta.
2) fuerzas potenciales repulsivas que son generadas por los obstáculos.

La función potencial artificial en la posición del robot $x$ se define en la forma:

$$\boldsymbol{U}(x) = \boldsymbol{U}_{atr}(x) + \boldsymbol{U}_{rep}(x) \tag{15}$$

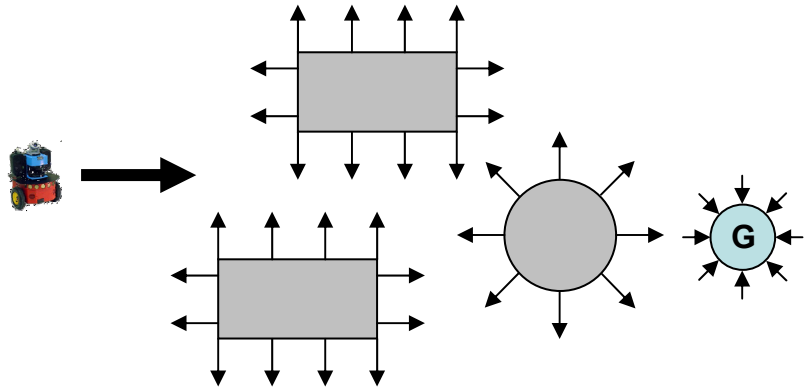



Figura 2.29. Fuerzas atractivas y repulsivas en un campo potencial artificial

Donde $\boldsymbol{U}_{atr}(x)$ es el campo potencial atractivo que produce la meta en $x$, y $\boldsymbol{U}_{rep}(x)$ es el campo potencial repulsivo causado por los obstáculos en $x$. La fuerza resultante $\boldsymbol{F}$ en la posición $x$ es entonces:

$$\boldsymbol{F}(x) = \boldsymbol{F}_{atr}(x) + \boldsymbol{F}_{rep}(x) \tag{16}$$

Donde

$$\boldsymbol{F}_{atr}(x) = -\nabla \boldsymbol{U}_{atr}(x)$$
$$\boldsymbol{F}_{rep}(x) = -\nabla \boldsymbol{U}_{rep}(x)$$

$\boldsymbol{F}_{atr}$ es la fuerza atractiva que guía al robot a la meta.

$\boldsymbol{F}_{rep}$ es la fuerza de repulsión que producen los obstáculos.

El operador Nabla $\nabla = \frac{\partial}{\partial x}\hat{i} + \frac{\partial}{\partial x}\hat{j} + \frac{\partial}{\partial x}\hat{k}$ .

Una vez que se calcula la fuerza resultante, $\boldsymbol{F}(x)$ se transforma en un comando de movimiento del robot.

La función potencial atractiva utilizada por [Kathib 1986] es:

$$\boldsymbol{U}_{\text{atr}}(q) = \frac{1}{2}\delta\rho^{r}(q,q_{meta}) \tag{17}$$

Donde

$q$ = es una configuración $[x, y]$ que refleja una posición en el espacio de trabajo.

$\delta$ = es un factor escalar positivo.

$\rho(q, q_{meta}) = \|q_{meta} - q\|$ Es la distancia entre el robot $q$ y la meta $q_{meta}$ calculada con la métrica $r$ ($r = 2$ en este caso).

La fuerza potencial atractiva es la gradiente negativa del campo potencial atractivo:

$$\boldsymbol{F}_{atr}(x) = -\nabla \boldsymbol{U}_{atr}(x) = \delta(x_{meta} - x) \tag{18}$$

La función potencial repulsiva es:

$$U_{rep}(obs_i) = \begin{cases} \frac{1}{2}\alpha_i\left(\frac{1}{\rho(q,q_{obs_i})} - \frac{1}{\rho_0}\right)^2 & \text{si } \rho(q,q_{obs_i}) \le \rho_0 \\ 0 & \text{si } \rho(q,q_{obs_i}) > \rho_0 \end{cases} \tag{19}$$

Donde

$i$ = 1 hasta $n$ numero de obstáculos.

$\alpha_i$ = Factor escalar positivo.

$\rho_0$ = constante positiva que denota las influencias del obstáculo en el robot.

$\rho(q, q_{obs_i})$ = es la distancia mínima desde el robot $q$ hasta el obstáculo $q_{obs_i}$ .

Similar a $\boldsymbol{F}_{atr}$, la fuerza potencial repulsiva del obstáculo $i$ es dada por:

$$\boldsymbol{F}_{rep}(obs_i) = -\nabla \boldsymbol{U}_{rep}(obs_i) \tag{20}$$

La dificultad de estos métodos depende de la definición de los diversos potenciales donde las funciones basadas en física simple (como el campo de gravedad), no trabajan muy bien para robots que tienen restricciones cinemáticas (no holonómicas).

### 2.2.4. Método del polígono de velocidades factibles

Uno de los métodos más eficientes para la planeación de movimiento de robots móviles es el desarrollo de planeadores de rutas que permitan la navegación y evasión de colisiones en ambientes estáticos de una manera factible. En el método de planeación de rutas local propuesto en [Ramírez y Zeghloul 2000], los obstáculos y el robot se modelan en base a polígonos convexos, y el robot se considera como un sistema con propiedades no holonómicas. Este método utiliza dos diferentes módulos: el primer módulo es una ley de control y un optimizador de velocidad para aproximar el robot a la meta, siguiendo una trayectoria estable mientras se evaden los obstáculos. El objetivo de la ley de control es calcular una velocidad de referencia que estabiliza el robot en una posición fija de un ambiente libre, a pesar de su propiedad no holonómica. El optimizador de velocidad se utiliza para evadir los obstáculos mediante una formulación similar a la propuesta por [Faverjon y Tournassaud 1987], donde los obstáculos se mapean como restricciones lineales sobre la

velocidad del robot para formar un polígono de velocidades factible. El módulo, vía cálculo de distancia, encuentra la velocidad más cercana en el polígono a la velocidad de referencia. El segundo módulo utiliza el FVP para escapar de las situaciones de estancamiento que podrían aparecer en la trayectoria.

**Cinemática de un robot móvil diferencial**

La configuración de un robot móvil (figura 2.30) en un plano se puede definir por el siguiente vector de estados:

$$[x, y, \theta]^T \tag{21}$$

Donde ($x$, $y$) es la posición de un punto fijo en el robot y θ la orientación del marco ligado al objeto con respecto al eje X. Se puede definir el centro **R** del eje de las ruedas como el punto fijo en el robot.

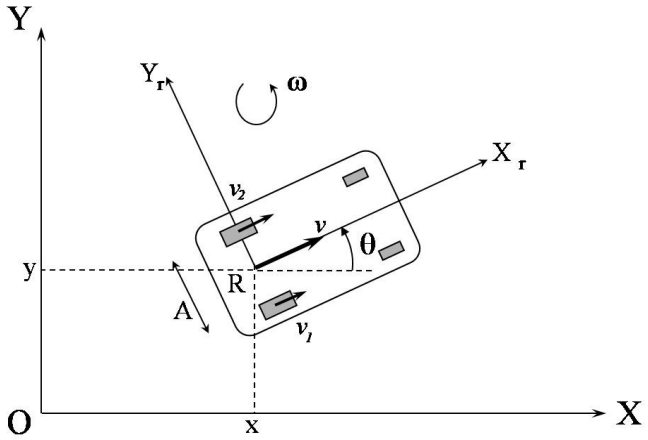



Figura 2.30. Un robot móvil diferencial

La velocidad lineal $v$ y la velocidad angular $\omega$ del robot se calculan mediante:

$$v = \frac{v_1 + v_2}{2} \quad , \quad \omega = \frac{v_1 - v_2}{A} \tag{22}$$

Donde $v_1$ y $v_2$ son respectivamente las velocidades de la rueda derecha y la rueda izquierda. El modelo de cinemática del robot móvil puede ser dado como:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}; \text{ o bien } \dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})\mathbf{u} \tag{23}$$

Lo que conlleva a los valores:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \theta &= \omega \end{aligned} \tag{24}$$

Donde $\mathbf{J}(\mathbf{q})$ es la matriz Jacobiana del robot y $\mathbf{u} = [v, w]^T$ es el vector de entrada (o vector de control). Los sistemas no holonómicos son sistemas con restricciones de velocidad no integrables. Para este sistema, la restricción no holonómica es la siguiente:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \frac{\dot{y}}{\dot{x}} &= \frac{v\sin\theta}{v\cos\theta} = \tan\theta \\ \dot{y} &= \dot{x}\tan\theta \\ \dot{y} - \dot{x}\tan\theta &= 0 \end{aligned} \tag{25}$$

**Ley de control**

Los sistemas no holonómicos son el centro de atención en la comunidad de control no lineal durante los últimos años. Estos sistemas no se pueden estabilizar en un punto de equilibrio utilizando una ley de control de retroalimentación de tiempo-invariante discreta (o aún continua), por lo que muchos de los resultados existentes de teoría de sistemas lineales y no lineales no son directamente aplicables en este caso. Las teorías de control de tiempo-variante se pueden utilizar pero tales teorías de control pueden bajar los rangos de convergencia. Los rangos de convergencia rápida necesitan leyes típicas de control no-discretas/no-continuas.

Si $\mathbf{q}_\mathrm{f} = [x_\mathrm{f}, y_\mathrm{f}, 0]^T$ es la posición meta deseada (independientemente de la posición final) y $\mathbf{q} = [x, y, \theta]^T$ la posición actual del robot. Utilizando las coordenadas polares $(a, \alpha)$ de la meta con respecto al marco del robot, se define el vector de error **z**($t$) (figura 2.31), como sigue:

$$\mathbf{z}(t) = \begin{bmatrix} a \\ \alpha \end{bmatrix} \tag{26}$$

Donde:

$$\begin{aligned} a &= \sqrt{x_e^2 + y_e^2} \\ \alpha &= \operatorname{atan2}(y_e, x_e) - \theta \qquad \alpha \in [-\pi, \pi] \end{aligned} \tag{27}$$

La dinámica del vector **z** es:

$$\begin{aligned} \dot{a} &= -v\cos\alpha \\ \dot{\alpha} &= \frac{1}{a} v \sin\alpha - \omega \end{aligned} \tag{28}$$

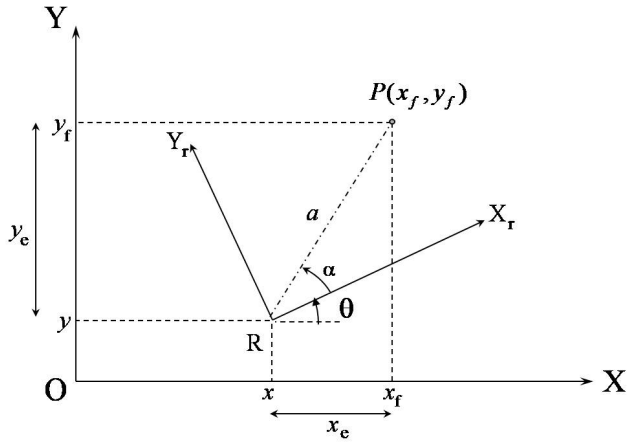



Figura 2.31. Definición del error

Se propone la siguiente ley de control, con dirección hacia una convergencia exponencial:

$$\begin{aligned} v &= k_1 a \cos\alpha \\ \omega &= k_2\alpha + k_1 \sin\alpha \cos\alpha \end{aligned} \tag{29}$$

Con $k_1$, $k_2 > 0$, el sistema de lazo cerrado se vuelve:

$$\begin{aligned} \dot{a} &= -(k_1 a \cos\alpha)\cos\alpha \\ \dot{\alpha} &= \frac{1}{a}(k_1 a \cos\alpha)\sin\alpha - (k_2\alpha + k_1 \sin\alpha\cos\alpha) \\ \dot{a} &= -(k_1 \cos^2\alpha)a \\ \dot{\alpha} &= -k_2\alpha \end{aligned} \tag{30}$$

Para probar que el origen (el cual representa la meta) es de forma global exponencialmente estable, se utiliza la siguiente función de Lyapunov:

$$\mathbf{V}(\mathbf{z}) = \frac{1}{2}a^2 + \frac{1}{2}\alpha^2 \geq 0 \qquad \forall \mathbf{z} \neq 0 \tag{31}$$

De forma que:

$$\begin{aligned} \mathbf{V}(\mathbf{z}) &= \frac{1}{2}a^2 + \frac{1}{2}\alpha^2 \\ \dot{\mathbf{V}}(\mathbf{z}) &= \frac{1}{\cancel{2}}\cancel{2}a\cdot\dot{a} + \frac{1}{\cancel{2}}\cancel{2}\alpha\cdot\dot{\alpha} \\ \dot{\mathbf{V}}(\mathbf{z}) &= a(-k_1\cos^2\alpha)a + \alpha(-k_2\alpha) \\ \dot{\mathbf{V}}(\mathbf{z}) &= -k_1 a^2\cos^2\alpha - k_2\alpha^2 < 0 \qquad \forall \mathbf{z} \neq 0 \end{aligned} \tag{32}$$

Debido a que tal función de Lyapunov existe, se garantiza la convergencia estable a 0 de $(a,\alpha)$, y $a = 0$ implica que el robot alcanzó la posición final.

**Módulo alcanzando_la_meta**

En [Faverjon y Tournassoud 1987] se propone un método sustituto de un campo potencial para definir la trayectoria en un ambiente ocupado de un robot manipulador con un alto número de

grados de libertad. Este método se llama método de restricciones, y su nombre es por que los obstáculos se mapean como restricciones lineales sobre las velocidades comunes del robot (figura 2.32). Este método se basa en un esquema interactivo donde el vector de configuración se encuentra utilizando la siguiente fórmula:

$$\mathbf{q}_{\mathrm{i}} = \mathbf{q}_{\mathrm{i-1}} + \Delta t \dot{\mathbf{q}} \tag{33}$$

El vector $\dot{\mathbf{q}}$ es el vector de velocidades comunes que minimizan la siguiente función:

$$f = \frac{1}{2}\left\| \dot{\mathbf{q}} - \dot{\mathbf{q}}_{goal} \right\|^2 \quad \text{con} \quad \dot{\mathbf{q}}_{goal} = \frac{\mathbf{q} - \mathbf{q}_f}{\left\| \mathbf{q} - \mathbf{q}_f \right\|} \tag{34}$$

Sujeta a las restricciones del obstáculo dadas por el modelo de velocidad de amortiguamiento:

$$\dot{d} \geq -\xi \frac{d - d_s}{d_i - d_s} \tag{35}$$

Donde $\mathbf{q}$ y $\mathbf{q}_f$ son respectivamente los vectores de configuración actual y final, $\dot{\mathbf{q}}_{goal}$ es el vector de velocidades comunes deseado, calculado para obtener una línea recta en el espacio común, $d$ es la distancia mínima entre el robot y el objeto considerado, $d_i$ es la distancia de influencia desde la cual una restricción se vuelve activa, $d_s$ es la distancia de seguridad que se debe respetar, y $\xi$ representa un coeficiente para adaptar la velocidad de convergencia.

La optimización del problema se puede establecer como sigue: encontrar un vector $\mathbf{u}$ que minimice la siguiente función:

$$\min_{\mathbf{u}} f_r = \frac{1}{2}\left\| \mathbf{u} - \mathbf{u}_{goal} \right\|^2 \tag{36}$$

Donde

$$\mathbf{u}_{goal} = \begin{bmatrix} k_1 a \cos\alpha \\ k_2\alpha + k_1 \sin\alpha \cos\alpha \end{bmatrix} \tag{37}$$

Es el vector velocidad que se obtiene de la ley de control exponencial propuesta en (29) y (30).

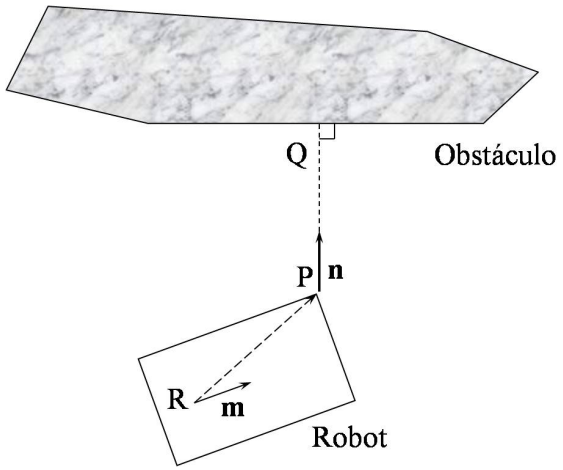



Figura 2.32. Estableciendo las restricciones del obstáculo

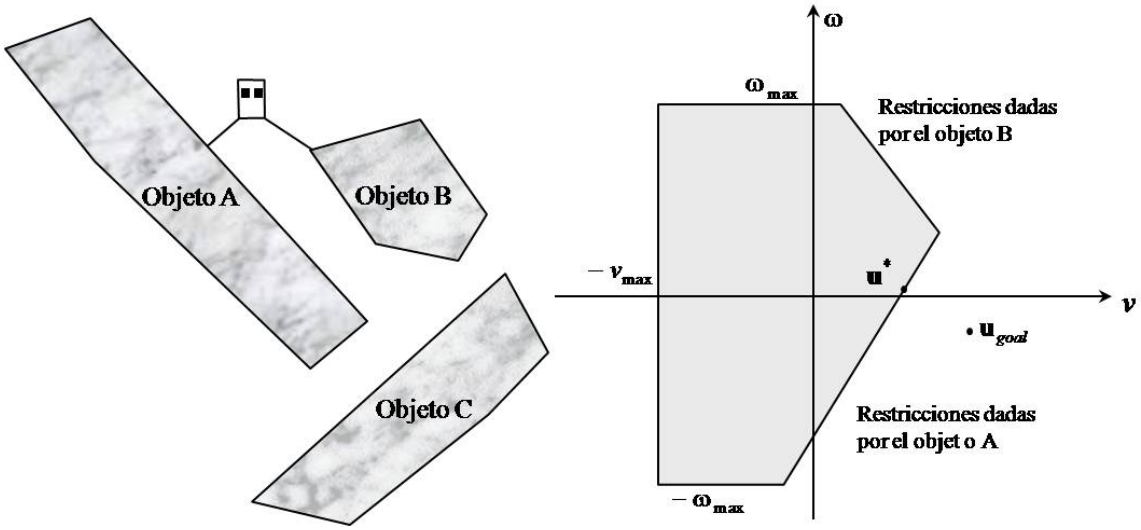
Para cualquier obstáculo a una distancia $d$ menor que $d_i$, se puede expresar la restricción de velocidad de amortiguamiento como una función del vector de control **u**:

$$(v\mathbf{m} + \omega\hat{k} \times \overline{RP}) \cdot \mathbf{n} \leq \xi \frac{d - d_s}{d_i - d_s} \tag{38}$$

Donde **m** es el vector unidad a lo largo del eje $X_r$ del robot y **n** es el vector unidad a lo largo del segmento PQ de distancia mínima entre el robot y el obstáculo.

El conjunto de restricciones del obstáculo y las restricciones de límite de velocidad son lineales en $(v, \omega)$, así que ellos definen un subconjunto convexo en el espacio $(v, \omega)$. Debido a que este subconjunto define el conjunto bidimensional de velocidades que el robot puede utilizar sin colisionar con los obstáculos, se le denomina "polígono de velocidades factible" o FVP.

Por ejemplo, considérese la situación mostrada en la figura 2.33(a). El obstáculo C está a una distancia más grande que $d_i$ del robot, así que solo los obstáculos A y B son considerados en el proceso de cálculo, esto es, ellos se mapean como restricciones sobre la velocidad del robot. El FVP resultante, en el espacio $(v, \omega)$ se muestra en la figura 2.33 (b), donde las restricciones de límite de velocidad y las dos restricciones de obstáculo definen el subconjunto convexo que garantiza que el robot pueda moverse sin colisionar. La velocidad para alcanzar la meta en un ambiente libre se representa por el punto $\mathbf{u}_{goal}$.




a) Configuración del obstáculo b) Polígono de velocidades factible

Figura 2.33. construcción del FVP

Se puede observar que el problema de minimización de (36) sujeto a las restricciones del obstáculo mostradas en (38) es un problema de cálculo de distancia mínima entre el FVP y el punto de referencia $\mathbf{u}_{goal}$. La solución a este problema se representa por el punto **u***, el cual es el punto más cercano en el polígono hacia $\mathbf{u}_{goal}$. Para resolver este problema, se utiliza un algoritmo de cálculo de distancia mínima desarrollado por [Zeghloul y Rambeaud 1996].

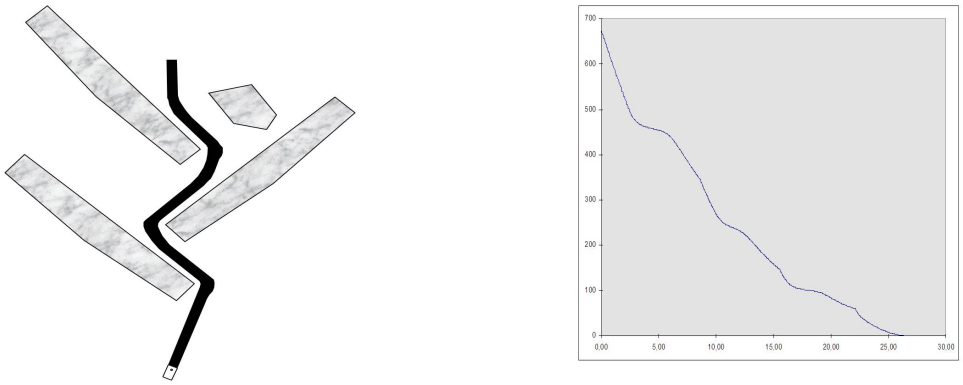
La variación del vector de configuración se calcula utilizando el vector solución, **u***, como sigue:

$$\Delta \mathbf{q}\mathrm{i} = \Delta t \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u}^* \tag{39}$$

El primer módulo del planeador de rutas, llamado "alcanzando_la_meta", se define por el siguiente proceso interactivo:

1) Computar la velocidad de referencia $\mathbf{u}_{goal}$, utilizando la ley de control exponencial.
2) Construir el FVP, por el mapeo de los obstáculos en la zona de influencia como restricciones lineales sobre las velocidades del robot.
3) Resolver el problema de optimización por el cálculo de distancia mínima.
4) Aplicar el vector **u***.

Este módulo le permite al robot resolver situaciones simples donde la configuración de los obstáculos no crea puntos que son mínimos locales. Un ejemplo se muestra en la figura 2.34. Mientras que el robot utiliza la solución **u***, la función de distancia (31) es estrictamente decreciente, dado que el robot esta continuamente aproximándose a la meta.




a) movimiento del robot b) función de distancia V(**z**)

Figura 2.34. Ejemplo de alcanzando_la_meta

Esta propiedad se utiliza en el segundo módulo para resolver las situaciones de estancamiento.

Cuando la solución del problema de optimización **u*** se localiza en el origen del plano $(v, \omega)$, el robot no puede continuar moviéndose utilizando el módulo de alcanzando_la_meta. Esta condición corresponde a un estancamiento, por lo que se aplica el segundo módulo, llamado "seguimiento_de_frontera".

**Módulo seguimiento_de_frontera**

Este módulo al igual que el primero están inspirados por el algoritmo Bug reportado en [Skewis and Lumelsky 1992]. Este módulo utiliza la representación del FVP y el valor de la función de distancia para permitirle al robot seguir la frontera del obstáculo para escapar del estancamiento. De acuerdo a la locación de **u*** en el FVP, existen dos tipos de estancamiento: el estancamiento de obstáculo simple, el cual ocurre cuando **u*** se localiza sobre una restricción del FVP, y el estancamiento de dos obstáculos, el cual ocurre cuando **u*** se localiza en un vértice del FVP, como se describe en la figura 2.35, y figura 2.36 respectivamente.

A pesar del número de obstáculos implicados en una situación de bloqueo, el proceso de solución es el mismo en ambos casos. Cuando se detecta una situación de estancamiento (**u*=0**), el valor actual de la función de distancia, $V_{block}$, es guardado y los dos vértices adyacentes a **u***, $s_R$ *y* $s_L$, son identificados. La posición del bloqueo de obstáculos, con respecto a la posición actual del robot define cual restricción de obstáculo será elegida para rodear, y cuál de los vértices $s_R$ *y* $s_L$ será el utilizado.

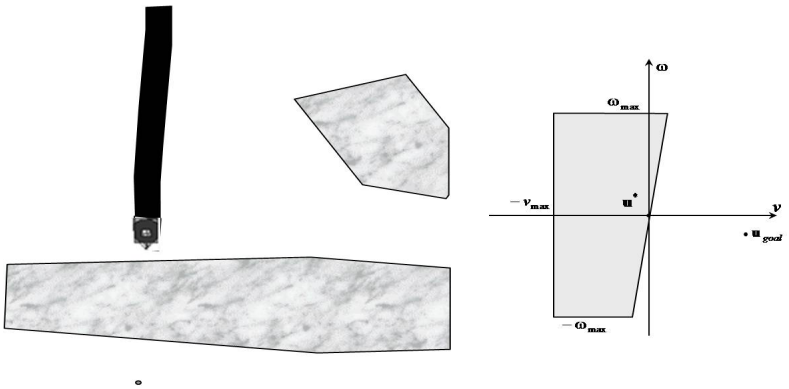



Figura 2.35. Estancamiento de obstáculo simple

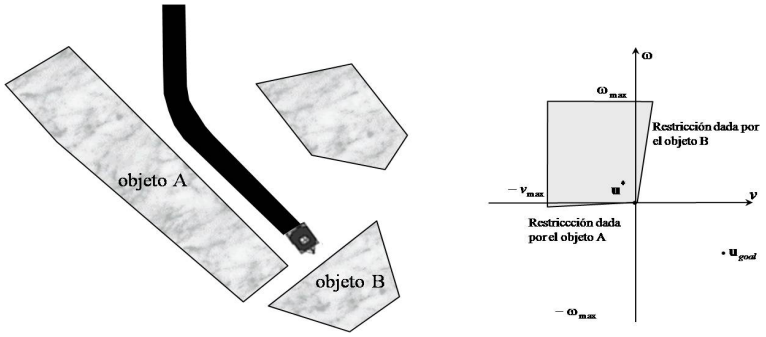



Figura 2.36. Estancamiento de dos obstáculos

Por lo tanto, el método rastrea la evolución de la restricción elegida en el FVP y aplica las velocidades representadas para el vértice elegido. Esto permite al robot seguir la frontera del obstáculo bloqueado a una distancia mayor o igual a $d_s$.

Mientras el robot rodea los obstáculos, el valor de la función de distancia se compara continuamente con $V_{block}$. Debido a que esta función es una medida de distancia entre el robot y el obstáculo, cuando el valor de la función de distancia es menor que $V_{block}$, el robot ha encontrado un punto sobre la frontera del obstáculo convexo que está más cerca de la meta que el punto de estancamiento. En este momento, el robot deja el módulo de seguimiento_de_frontera para continuar utilizando el módulo de alcanzando_la_meta.

Es posible que, mientras el robot esta siguiendo la frontera del obstáculo, el vértice elegido converja con el origen del plano $(v,\omega)$, esto significa que un nuevo obstáculo previene al robot de seguir la frontera del obstáculo. Cuando esto ocurre, el método simplemente cambia de la restricción seguida a una nueva, por lo que el robot rodeará el nuevo obstáculo.

Para ilustrar los dos módulos y su interacción, considérese la situación mostrada en la figura 2.37, con un valor de distancia inicial V(**z**) = 299.08.

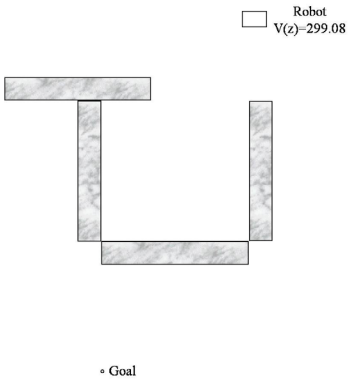



Figura 2.37. un ejemplo de un punto muerto

Como se muestra en la figura 2.38, el robot utilizando el módulo de alcanzando_la_meta se moverá hacia la posición final siguiendo una trayectoria estable, hasta que esta trayectoria sea

bloqueada por un obstáculo. Esta situación se detecta como una condición de estancamiento (**u* = 0**) y se registra el valor actual de la función de distancia, $V_{block} = 52.43$.

Después de esto, se identifican los vértices $s_R$ y $s_L$. Debido a que el bloqueo del obstáculo se localiza a la izquierda del robot, el vértice elegido es $s_R$ puesto que este permite al robot girar a la derecha.

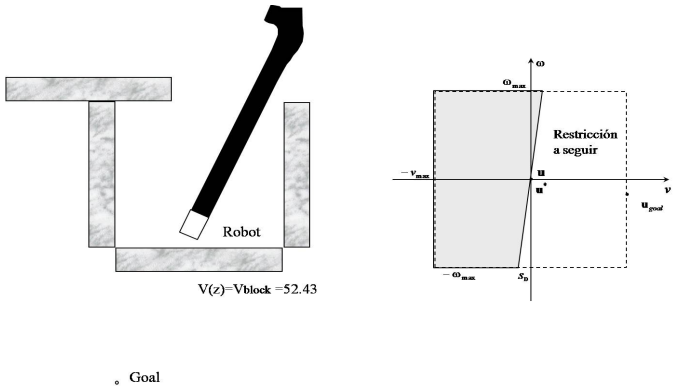



Figura 2.38. situación de estancamiento

El método rastreará la evolución de la restricción seguida y aplicará las velocidades representadas por $s_R$. En la situación que se muestra en la figura 2.39, el vértice $s_R$ converge en el origen del plano $(v, \omega)$, entonces el robot encuentra un nuevo obstáculo de bloqueo.

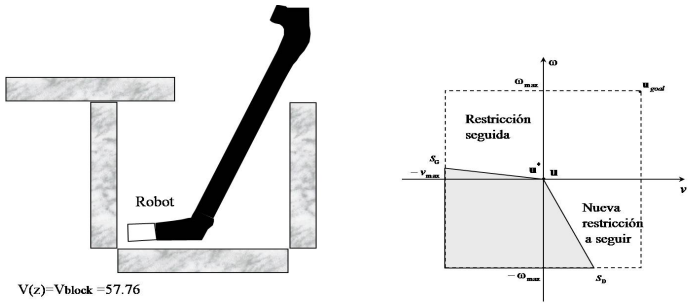



Figura 2.39. Cambiando la restricción a seguir

El método cambia a la siguiente restricción y continúa rodeando el nuevo obstáculo, utilizando el nuevo vértice $s_R$. Cada vez que el vértice elegido converge a cero, el método cambia a la

siguiente restricción, de manera que el robot pueda rodear los obstáculos de bloqueo, como se muestra en la figura 2.40.

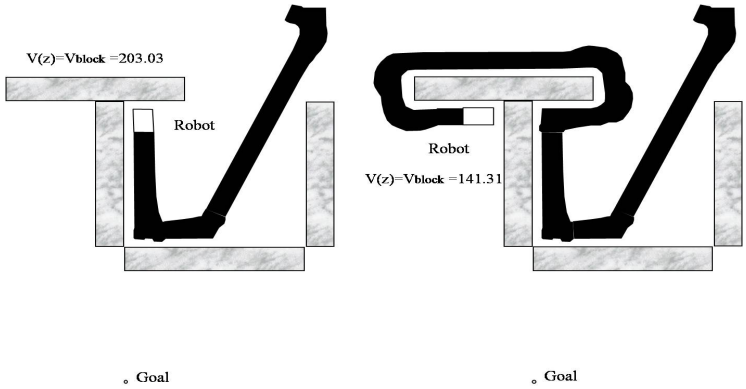



Figura 2.40. Rodeando obstáculos

Cuando el valor actual de la función de distancia es menor que el valor del estancamiento, $V_{block}$, el planeador deja el módulo de seguimiento_de_frontera y utiliza el módulo de alcanzando_la_meta, hasta que se alcanza el objetivo (figura 2.41). La trayectoria resultante es una secuencia de intervalos de alcanzando_la_meta y seguimiento_de_frontera. Por construcción, por cada intervalo de seguimiento_de_frontera, el valor final de la función de distancia es menor que el inicial ($V_{block}$). Además, la función de distancia es estrictamente decreciente en los intervalos de alcanzando_la_meta. Si los intervalos de seguimiento_de_frontera son finitos, se puede afirmar que la función de distancia converge a cero. Entonces, el robot alcanza la posición final.

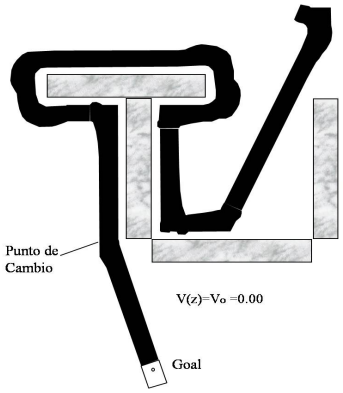



Figura 2.41. Solución del problema de planeación de rutas

Es importante notar que los dos módulos están basados en solo el polígono de velocidades factible, el cual se construye de la información de distancia entre el robot y los obstáculos, entonces el método se debe adaptar bien a un robot bien equipado con sensores de distancia, como sensores ultrasónicos, láser o cámaras de video.

### 2.2.5. Método de la ventana activa

Es un método diseñado para utilizar restricciones impuestas por velocidades limitadas y aceleraciones debido a que este se origina directamente de la dinámica de movimiento de robots móviles dirigidos en forma sincrónica.

En resumen, este método considera periódicamente un corto intervalo de tiempo cuando computa el siguiente comando de dirección para evitar la enorme complejidad del problema de planeación de movimiento general. La aproximación de trayectorias durante tal intervalo de tiempo es dado por resultados de curvaturas circulares en un espacio de búsqueda de dos dimensiones de velocidades de traslación y de rotación. Este espacio de búsqueda es reducido a velocidades admisibles permitiéndole al robot parar de manera segura. Debido a las aceleraciones limitadas de los motores, una restricción más es impuesta en las velocidades: el robot sólo considera velocidades que pueden ser alcanzadas dentro del siguiente intervalo de tiempo. Estas velocidades forman *la ventana dinámica*, la cual se centra alrededor de las velocidades actuales del robot en el espacio de velocidades. En medio de las velocidades admisibles dentro de la ventana dinámica, la combinación de velocidad de rotación y traslación se elige por medio de la maximización de una función objetivo, la cual incluye una medida del progreso hacia el punto meta, la velocidad hacia delante del robot, y la distancia al siguiente obstáculo en la trayectoria. Por la combinación de estas, el robot intercambia entre la decisión de moverse más rápido hacia la meta y circunnavegar los obstáculos (lo cual decrementa el espacio libre). La combinación de todos estos objetivos conlleva a una estrategia de evasión de colisiones elegante y robusta.

**Ecuaciones de movimiento**

Esta sección describe las ecuaciones de movimiento fundamental de robots móviles dirigidos sincrónicamente. Se comienza con leyes dinámicas, asumiendo que la velocidad de rotación y traslación pueden ser controladas independientemente (con torsión limitada). Para hacer las ecuaciones más prácticas, se propone un acercamiento que modele la velocidad como una función inteligente constante en el tiempo. Bajo esta suposición, las trayectorias del robot consisten de secuencias de segmentos finitos de círculos. Estas representaciones son muy convenientes para la identificación de colisiones, donde las intersecciones de los obstáculos con círculos son fáciles de detectar. También se propone un límite superior para la aproximación del error. La representación circular forma la base del método de la ventana dinámica para evasión de colisiones.

**Ecuaciones de movimiento general**

Sea $x(t)$ y $y(t)$ las coordenadas del robot en el tiempo $t$ en el mismo sistema de coordenadas global, y sea la orientación del robot (rumbo de dirección) descrita por $\theta(t)$. La tripleta $< x, y, \theta >$ describe la configuración cinemática del robot. El movimiento de un robot dirigido en forma síncrona es restringido de manera tal que la velocidad de traslación $v$ siempre lleva en la dirección $\theta$ del robot, lo cual es una restricción no holonómica.

Sea $x(t_0)$ y $x(t_n)$ las coordenadas $x$ del robot en el tiempo $(t_0)$ y $(t_n)$ respectivamente. Sea $v(t)$ la velocidad de translación del robot en el tiempo $t$, y $\omega(t)$ la velocidad de rotación. Entonces $x(t_n)$ y $y(t_n)$ se pueden expresar como una función de $x(t_0)$, $v(t)$ y $\theta(t)$:

$$x(t_n) = x(t_0) + \int_{t_0}^{t_n} v(t) \cdot \cos\theta(t) dt \tag{40}$$

$$y(t_n) = y(t_0) + \int_{t_0}^{t_n} v(t) \cdot \sin\theta(t) dt \tag{41}$$

Las ecuaciones (40) y (41) dependen de las velocidades del robot, las cuales usualmente no pueden ser establecidas directamente.

En su lugar, la velocidad $v(t)$ depende de la velocidad de traslación inicial $v(t_0)$ en $t_0$, y la aceleración de traslación $\dot{v}(\hat{t})$ en el intervalo de tiempo $\hat{t} \in [t_0, t]$. De manera similar, la orientación $\theta(t)$ es una función de la orientación inicial $\theta(t_0)$, la velocidad de rotación inicial $\omega(t_0)$ en $t_0$, y la aceleración de rotación $\dot{\omega}(\hat{t})$ con $\hat{t} \in [t, t_0]$. Substituyendo $v(t)$ y $\theta(t)$ por la correspondiente configuración inicial de cinemática y dinámica $v(t_0), \theta(t_0), \omega(t_0)$ y las aceleraciones $\dot{v}(\hat{t})$ y $\dot{\omega}(\hat{t})$ conllevan a la expresión (nótese que la derivación de $y(t_n)$ es análoga, por lo que se describe la derivación para la coordenada *x*):

$$x(t_n) = x(t_0) + \int_{t_0}^{t_n} \left( v(t_0) + \int_{t_0}^{t_n} \dot{v}(\hat{t}) d\hat{t} \right) \cdot \cos\left( \theta(t_0) + \int_{t_0}^{t} (\omega(t_0) + \int_{t_0}^{\hat{t}} \dot{\omega}(\tilde{t}) d\tilde{t}) d\tilde{t} \right) dt \qquad (42)$$

Las ecuaciones están ahora de forma que la trayectoria del robot depende exclusivamente de su configuración dinámica inicial en el tiempo $t_0$ y las aceleraciones, las cuales se asumen son controlables (para muchos robots las aceleraciones que determinan su movimiento son funciones monótonas de los flujos de corriente a través del motor. Por lo tanto, los límites en las corrientes corresponden directamente a los límites en las aceleraciones).

La ecuación (42) puede ser simplificada si se asume que entre dos puntos arbitrarios en el tiempo, $t_0$ y $t_n$, el robot puede ser solo controlado por comandos finitos de aceleración. Sea *n* este número de instantes de tiempo. Entonces, las aceleraciones $\dot{v}_i$ y $\dot{\omega}_i$ para $i = 1 \ldots n$ se mantienen constantes en un intervalo de tiempo $[t_i, t_i + 1](i = 1...n)$. Sea $\Delta t_t^l$ definido como $\Delta t_t^l = t - t_i$ para algún índice del intervalo $i = 1 \ldots n$. Utilizando esta forma discreta, el comportamiento dinámico del robot se expresa por la siguiente ecuación, la cual proviene directamente de la ecuación (42) bajo la suposición de aceleraciones constantes para cada pieza:

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} \left( v(t_i) + \dot{v}_i \cdot \Delta_t^i \frac{1}{2} \right) \cdot \cos\left( \theta(t_i) + \omega(t_i) \cdot \Delta_t^i + \frac{1}{2} \dot{\omega}_i \cdot (\Delta_t^i)^2 \right) dt \qquad (43)$$

**Ecuaciones de movimiento aproximado**

Aunque la ecuación (43) describe el caso general de control de un robot móvil, esta no es particularmente de ayuda cuando se trata de determinar la dirección de navegación, esto es por que las trayectorias generadas por estas ecuaciones son complejas. La ecuación (43) se puede simplificar por la aproximación de las velocidades del robot dentro del intervalo de tiempo $[t_i, t_{i+1}]$ por un valor constante. La ecuación de movimiento resultante (44) converge con la ecuación (43) tal como la longitud de los intervalos de tiempo va a cero.

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} v_i \cdot \cos\left(\theta(t_i) + \omega_i \cdot (\hat{t} - t_i)^2\right) d\hat{t} \tag{44}$$

La cual, por la solución de la integral, queda simplificada en:

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1} \left(F_x^i(t_{i+1})\right) \tag{45}$$

Donde

$$F_x^i(t) = \begin{cases} \frac{v_i}{\omega_i}\left(\sin\theta(t_i) - \sin(\theta(t_i) + \omega_i \cdot (t - t_i))\right), & \omega_i \neq 0 \\ v_i \cos(\theta(t_i)) \cdot t, & \omega_i = 0 \end{cases} \tag{46}$$

Las ecuaciones correspondientes para la coordenada *y* son:

$$y(t_n) = y(t_0) + \sum_{i=0}^{n-1} \left(F_y^i(t_{i+1})\right) \tag{47}$$

$$F_y^i(t) = \begin{matrix} -\frac{v_i}{\omega_i}\left(\cos\theta(t_i) - \cos(\theta(t_i) + \omega_i \cdot (t - t_i))\right), & \omega_i \neq 0 \\ v_i \sin(\theta(t_i)) \cdot t, & \omega_i = 0 \end{matrix} \tag{48}$$

Nótese que si $\omega_i = 0$, el robot seguirá una línea recta. Por el contrario, si $\omega_i \neq 0$, la trayectoria del robot describirá un círculo, por lo que se puede considerar:

$$M_x^i = -\frac{v_i}{\omega_i} \cdot \sin\theta(t_i) \tag{49}$$

$$M_y^i = \frac{v_i}{\omega_i} \cdot \cos\theta(t_i) \tag{50}$$

Por lo cual, la siguiente relación se mantiene:

$$\left(F_x^i - M_x^i\right)^2 + \left(F_y^i - M_y^i\right)^2 = \left(\frac{v_i}{\omega_i}\right)^2 \tag{51}$$

Esto muestra que la *i*-ésima trayectoria es un círculo $M_i$ alrededor de $\left(M_x^i, M_y^i\right)$ con un radio $M_r^i = \frac{v_i}{\omega_i}$.

**Componentes principales del método de la ventana activa**

En este método la búsqueda de comandos que controlan el robot es llevada a cabo directamente en el espacio de velocidades. La dinámica del robot se incorpora en el método por la reducción del espacio de búsqueda a aquellas velocidades que son alcanzables bajo las restricciones dinámicas. En adición a esta restricción, se consideran solo aquellas velocidades que son seguras con respecto a los obstáculos. Este recorte del espacio de búsqueda se realiza en el primer paso del algoritmo, en el segundo paso, se elige de entre las velocidades restantes, la velocidad que maximiza la función objetivo. En la figura 2.42 se muestra un breve bosquejo de las diferentes partes de un ciclo del algoritmo.

1. **Espacio de búsqueda:** El espacio de búsqueda de las velocidades posibles se reduce a tres pasos:
   a) ***Trayectorias circulares:*** el método de la ventana dinámica considera solo trayectorias circulares (curvaturas), determinadas únicamente por pares $(v, \omega)$ de velocidades de rotación y translación. Esto da como resultado un espacio de búsqueda de velocidades de dos dimensiones.
   b) ***Velocidades admisibles:*** La restricción para velocidades admisibles asegura que solo las trayectorias seguras sean consideradas. Un par $(v, \omega)$ se considera admisible, si el robot es capaz de parar antes de que este alcance el obstáculo más cercano en la curvatura correspondiente.
   c) ***Ventana dinámica:*** La ventana dinámica restringe las velocidades admisibles a aquellas que puedan ser alcanzadas dentro de un corto intervalo de tiempo dadas las aceleraciones limitadas del robot.
2. **Optimización:** se maximiza la siguiente función objetivo:

$$G(v,\omega) = \sigma\left(\alpha \cdot \text{guía}(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot \text{vel}(v,\omega)\right) \tag{52}$$

Con respecto a la posición y orientación actual del robot, esta función cambia los siguiente aspectos:
   a) ***Guía meta:*** *guía* es una medida del progreso hacia el punto meta. Esta es máxima si el robot se mueve directamente hacia la meta.
   b) ***Despejado:*** *dist* es la distancia al objeto más cercano en la trayectoria. Entre más pequeña sea la distancia a un obstáculo, más alto será el deseo del robot de circunnavegarlo.
   c) ***Velocidad:*** *vel* es la velocidad hacia delante del robot y soporta movimientos rápidos.

La función $\sigma$ suaviza la suma de pesos de los tres componentes y da como resultado un lado más despejado del obstáculo.

Figura 2.42. Diferentes partes del método de la ventana dinámica

*Primera ley de la robótica: Un robot no puede dañar a un ser humano,*
*o por inacción, permitir que un ser humano sea dañado*
*Enciclopedia Galáctica, manual de robótica, 56ª edición. Año 2058 D.C.*

# Capítulo 3

# Estado del arte

Este capítulo presenta una investigación actualizada sobre la planeación de movimiento de robots móviles. En como los investigadores han abordado el tema y los métodos y técnicas que recientemente se han utilizado para la solución del problema.

En [Ramírez y Zeghloul 2000], se presenta un nuevo planeador de rutas local, el cual es utilizado para la navegación y evasión de colisiones de robots móviles (Robosoft Robuter) en ambientes con obstáculos. Este método es aplicable para cualquier robot móvil autónomo que cuente con un sistema de sensores que permita captar la información suficiente para determinar la distancia entre el robot y los obstáculos; entre estos captores se encuentran los sensores ultrasónicos, láser y cámaras de video. Al realizar una navegación progresiva en un ambiente, el robot mapea los obstáculos que sus captores detectan representándolos como restricciones lineales en su espacio de velocidades. Debido a que las restricciones de obstáculos son lineales, estas forman un subconjunto convexo que representa las velocidades que el robot puede utilizar sin colisionar con los objetos. Este subconjunto convexo es

denominado polígono de velocidades factibles (FVP, por sus siglas en inglés de Feasible Velocities Polygon). El planeador está compuesto de dos módulos: el primer módulo representa un problema de optimización, que se transforma en un problema de cálculo de distancia mínima en el espacio de velocidad entre el FVP y un punto; el segundo módulo, que se utiliza cuando una situación de estancamiento ocurre, utiliza la representación del FVP para encontrar localmente la mejor velocidad para escapar del estancamiento. Algunas de las ventajas de este método son que se puede calcular en corto tiempo y que mantiene un comportamiento continuamente estable de las velocidades.

En [Parag and Nourbakhsh 2000] se desarrolla la planeación de trayectorias y navegación en base a dead reckoning de un robot Cye de dos ruedas con sensores codificadores que determinan la odometría y de esta forma mantener la posición del robot en el mundo. Para mantener una precisión en el rastreo, Cye utiliza superficies de calibración, llamadas puntos de chequeo, que son superficies conocidas como muros o esquinas con las cuales el robot llega a un punto cercano a la colisión para re-calibrarse así mismo. Cye utiliza una técnica de rejillas de evidencias [Martin y Moravec 1996], en la cual el mundo es discretizado en pequeñas celdas de rejillas que contienen una medida de ocupación. En los extremos, una celda tiene un 0 si tiene un espacio libre con un 100% de certidumbre, caso contrario, tiene un 255 reflejando un área ocupada con 100% de certidumbre. Para la planeación de trayectorias y navegación, el robot incorpora un campo potencial basado en rejillas que se construye en dos etapas. En la primera etapa, cada celda contiene un valor que denota la transversabilidad en ese punto, para ello se crea un campo que contiene las distancias de cualquier punto del mundo al objeto más cercano. Este campo se crea en base a una transformación grassfire. En la segunda etapa, se crea un campo potencial que contiene la distancia de cualquier punto del mundo a la meta. Este campo se genera utilizando una expansión de onda frontal desde el punto meta del robot. Una trayectoria es creada a partir de este campo garantizando la minimización de la longitud de la trayectoria y del costo de transversabilidad.

En [Clark y El-Osery 2004] se describen dos algoritmos complementarios para la operación de robots móviles en ambientes tipo laberinto. Los algoritmos fueron implementados en un

simulador modular UMBRA utilizando una plataforma RTLinux. El primer algoritmo se utiliza para realizar el mapeo ambiental y navegación de manera que se cubra por completo el laberinto con locaciones de muros desconocidas a priori. Este algoritmo emplea un esquema de seguimiento de muros para explorar el laberinto, y utiliza un detector de ocurrencias basado en estados para evitar ambientes repetidos y ciclos. También utiliza una variante de una estrategia de búsqueda en profundidad (conocida como flood fill) para la planeación de trayectorias en el mapa. Esta estrategia, realiza una búsqueda incremental a partir de la posición actual hasta lograr la meta y traza la trayectoria más corta que sea encontrada. Para ello, cada celda es etiquetada con un valor entero positivo representando su distancia al punto origen. La celda origen, por lo tanto, tiene un valor de 0, las celdas adyacentes un valor de 1 y así sucesivamente. El segundo algoritmo implementa un método autómata de aprendizaje estocástico (SLA) para evitar la colisión con obstáculos no estructurados dentro del laberinto. Este algoritmo utiliza un sistema de multas y recompensas que se utiliza para adaptar las probabilidades de acción. El premio o multa se asigna con base a la distancia de la iteración actual, si la distancia actual es menor o igual a la iteración previa, se otorga un premio, de otra forma la acción es penalizada.

Debido a que los errores en el sensado y control son imposibles de eliminar y que la incertidumbre generada por los mismos es crucial para lograr una correcta navegación, es importante generar planes robustos para estos errores que tomen en cuenta la incertidumbre generada y garanticen que la meta será alcanzada. El método de planeación de movimiento parcial (PMP) es un esquema de planeación reactiva utilizado para calcular iterativamente planos parciales seguros por medio de exploración incremental del espacio de tiempo-estados del ambiente hasta que el robot alcance su meta. En [Petti y Fraichard 2005] se aborda el problema de navegación en ambientes dinámicos con incertidumbre, en el cual se utiliza una extensión del método PMP con cálculo de incertidumbre para la planeación de trayectorias robustas en la simulación de errores de un robot de tipo carro. Para ello se utiliza un ciclo de planeación constante que permite regularmente obtener y actualizar el modelo. Este ciclo consiste iterativamente en: 1) el modelo actualizado de futuros estados, 2) la construcción de un árbol mediante la exploración incremental del espacio de tiempo-estados del robot, 3) la

selección y seguimiento de la mejor trayectoria parcial en el árbol de acuerdo a ciertos criterios (métrica, seguridad) una vez que el tiempo para la iteración actual ha terminado. Esta extensión del PMP utiliza una representación probabilística de los errores que aparecen en la ejecución y el control, garantizando que la meta será alcanzada.

En [Hun y Shin 2004] se utiliza una nueva función potencial repulsiva (RPF) para establecer la planeación de trayectorias del robot. Esta RPF tiene una magnitud diferente para cada dirección, basada en el ángulo entre una meta y un obstáculo, lo cual difiere de una RPF convencional, en la cual la magnitud en cualquier dirección es la misma. Se presenta un conjunto de análisis para diseñar funciones potenciales que eviten mínimos locales en un número determinado de escenarios. Específicamente, se prueban mediante simulación los siguientes casos: 1) un problema meta no alcanzable (caso en el cual el potencial de la meta es superado por el potencial de un obstáculo), 2) un problema de colisión con obstáculos (caso en el cual el potencial del obstáculo es superado por el potencial de la meta), 3) un problema de un pasaje estrecho (caso en el cual el potencial de la meta es superado por el potencial de los dos obstáculos). El esquema RPF utilizado elimina el área no factible para los tres casos propuestos con la ayuda de una magnitud de ángulo variante entre una meta y un obstáculo, construyendo de forma eficaz un sistema de planeación de trayectorias con la capacidad de alcanzar una meta y evadir obstáculos a pesar de posibles mínimos locales.

## 3.1. Análisis del estado del arte

Con base en un detallado escrutinio de los trabajos relacionados con la planeación de movimiento previamente citados, y teniendo como fundamento el requerir un tiempo de cálculo aceptable y la necesidad de un comportamiento continuamente estable del sistema, además de la adaptación a la infraestructura existente, se toma como base de esta investigación el planeador de rutas local mediante el método del FVP propuesto por [Ramírez y Zeghloul 2000]. Extendiendo su aplicación primero al simulador MobileSim de MobileRobots (basado en player/stage), y después a una infraestructura de robot Pioneer 2DX de ActiveMediaRobotics con sensores ultrasónicos y codificadores de odometría.

*Segunda ley de la robótica: Un robot debe obedecer las órdenes dadas por los seres humanos, excepto cuando tales órdenes entran en conflicto con la primera ley*
*Enciclopedia Galáctica, manual de robótica, 56ª edición. Año 2058 D.C.*

# Capítulo 4

# Metodología de solución

En esta sección se describe la metodología utilizada para solucionar el problema de la planeación de movimiento de un robot móvil mediante el uso de un planeador de rutas local que utiliza el método del FVP para la navegación y evasión de obstáculos en un ambiente estático desconocido con obstáculos poligonales convexos. Se comienza con la especificación de la infraestructura robótica con la que se realiza la experimentación. Después se define la interfaz de programación que se utiliza para controlar el robot. Se indica el simulador y editor de mapas utilizados en las pruebas. Por último se describen las clases principales y algoritmo pertenecientes a los módulos del planeador desarrollado.

## 4.1. El robot Pioneer 2DX

El robot Pioneer 2DX (figura 4.1) posee una corona de ultrasónicos, con un total de dieciséis sonares dispuestos radialmente. También tiene un par de cuenta vueltas (encoders) de 500 tics por vuelta en ambas ruedas.

Figura 4.1. Pioneer DX2 de Active Media Robotics

Para el movimiento, el robot incorpora dos ruedas motoras, y una rueda loca (para la estabilización). Los dos potentes motores incorporados le permiten desplazarse a velocidades de hasta 1.8 m/s o transportar hasta 23 kg de carga. Como elemento de proceso, incorpora un pequeño microcontrolador Hitachi HS8 encargado del manejo a bajo nivel de los distintos periféricos conectados, y de la comunicación con la unidad de proceso, la cual puede ser una laptop o cualquiera de los dispositivos mostrados en las configuraciones de la figura 4.2.

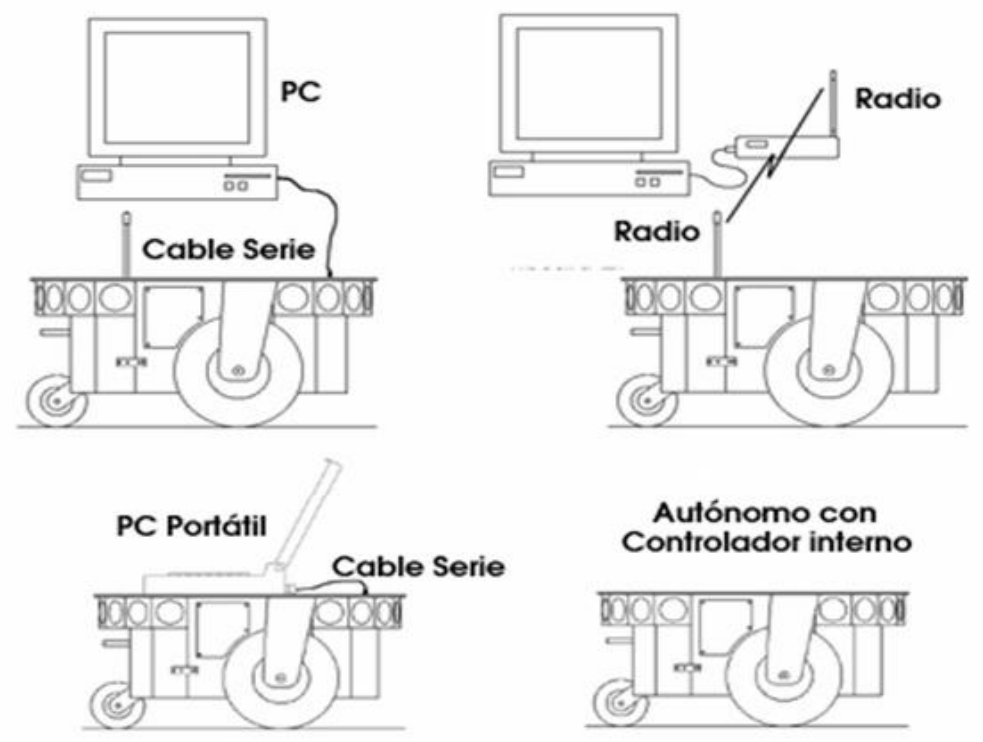



Figura 4.2. Configuraciones de control para el Pioneer

Un sensor sonar, basa su funcionamiento en las propiedades de las ondas ultrasónicas. Dichas ondas, son ondas mecánicas, ya que necesitan de un medio material para su transmisión, que viajan por el espacio a una velocidad próxima a 330 m/s a 0oC y aire seco. Como toda onda, una onda ultrasónica rebota en las superficies, efecto que conocemos como eco. Así, el funcionamiento básico de un sonar, que se observa en la figura 4.3, utiliza esta propiedad de la siguiente manera:

1) Un transductor emite una onda sonora de frecuencia ultrasónica (por encima de los 44kHz) y a la vez activa un cronometro.
2) Se espera la vuelta del eco.
3) Cuando se detecta el eco, se calcula el tiempo de ida y vuelta, llamado tiempo de vuelo. Puesto que conocemos la velocidad del ultrasonido, podemos calcular la distancia recorrida por la onda, que será exactamente el doble de la distancia al obstáculo. Dicho mecanismo, es el utilizado en los sonares del robot Pioneer. Sus sensores de ultrasonidos, desarrollados por Polaroid Co., son capaces de emitir ondas sonoras de 40 a 100 KHz, y pueden medir distancias a obstáculos desde 15cm a 5m.

El espacio captado por los sonares no se limita a la recta normal al sensor, sino que se expande como un cono de ángulo de 30°, por lo que el ángulo de dicho cono depende del cono de propagación de la onda ultrasónica.

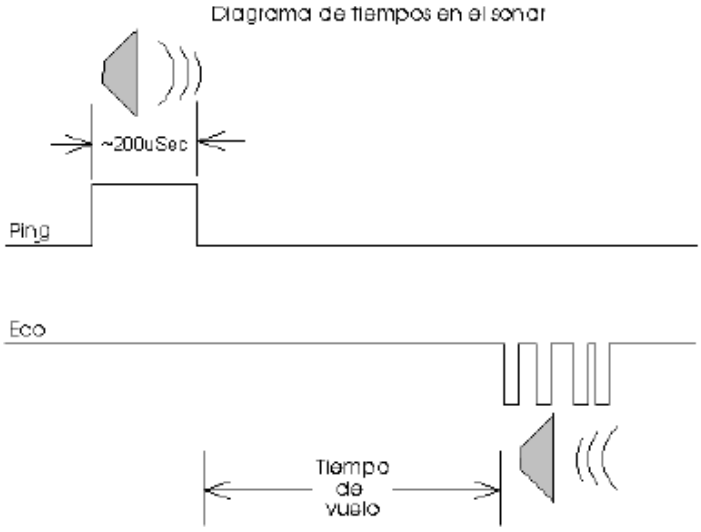



Figura 4.3. Funcionamiento de un sonar

Aunque las características de los sonares en la detección de obstáculos son bastante efectivas con respecto a otros sensores, como los infrarrojos, también tienen limitaciones. Por ejemplo, este tipo de sensores sufren de ruidos radiales (estimación inexacta de la distancia), reflexión especular sobre ciertas superficies y ángulos de incidencia (múltiples ecos que no permiten detectar el obstáculo real) y de la denominada incertidumbre angular. Este último defecto se refiere al ángulo $\alpha$ de propagación del sonar. Así, dada una medición $d$ no podemos concretar mas que el obstáculo que produjo la reflexión se encuentra en algún punto del arco con origen en el sensor, radio $d$ y ángulo $\alpha$. Si se toma un enfoque conservador para las lecturas de este sensor, se interpretará como ocupado todo el arco entre estos puntos.

## 4.2. La interfaz ARIA (Advanced Robotics Interface for Applications)

En este proyecto se utiliza una interfaz de acceso al hardware: ARIA, la cual ofrece una colección de clases que configuran una API articulada (figura 4.4).

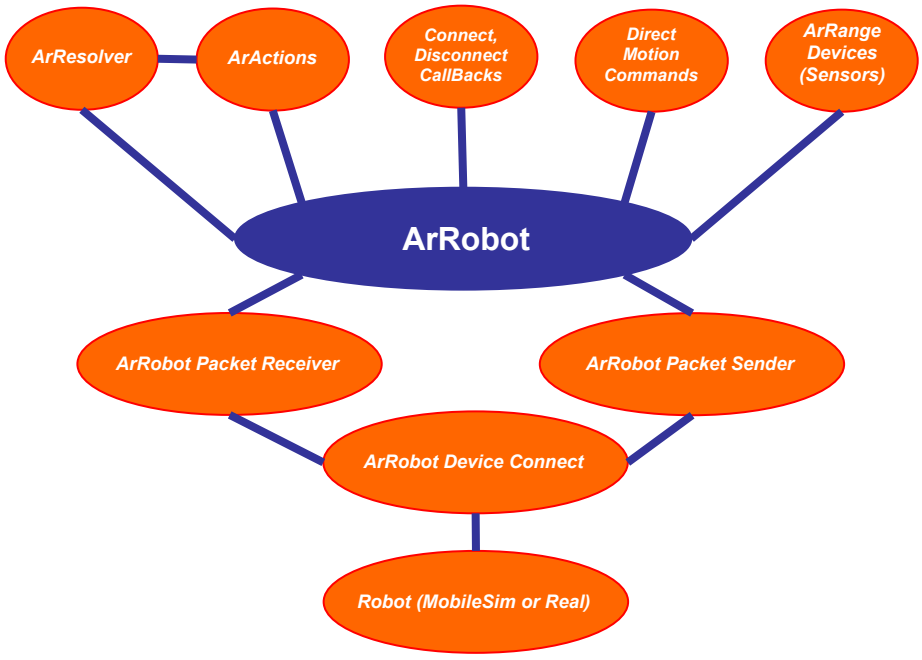



Figura 4.4. API de ARIA

Las características principales de ARIA son las siguientes:

1) Ofrece un ambiente orientado a objetos.
2) Incluye soporte para programación multitarea.
3) Incluye soporte para las comunicaciones a través de la red.
4) Las aplicaciones en esta plataforma se deben escribir en C++.
5) En el bajo nivel ARIA tiene un diseño de cliente-servidor: el robot esta gobernado por un microcontrolador que hace de servidor.

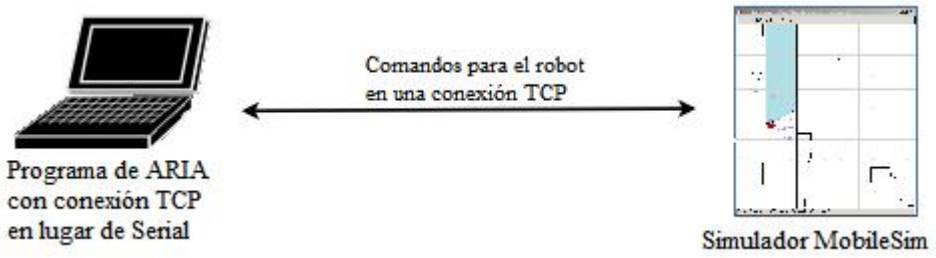
La clase principal *ArRobot* contiene varios métodos y objetos asociados. *Packet Receiver* y *Packet Sender* están relacionados con el envío y recepción de paquetes con el servidor por el puerto serie. Dentro de la clase *ArRangeDevices*, se tienen clases más concretas como *ArSonarDevice* o *ArSick* cuyos objetos contienen métodos que permiten a la aplicación acceder a las lecturas de los sensores de proximidad, tanto si estos son sonares o escáner láser. Además, la clase *DirectMotionCommands* de *ArRobot* contiene métodos para comandar consignas a los motores del robot.

### 4.2.1. Comunicación con el robot

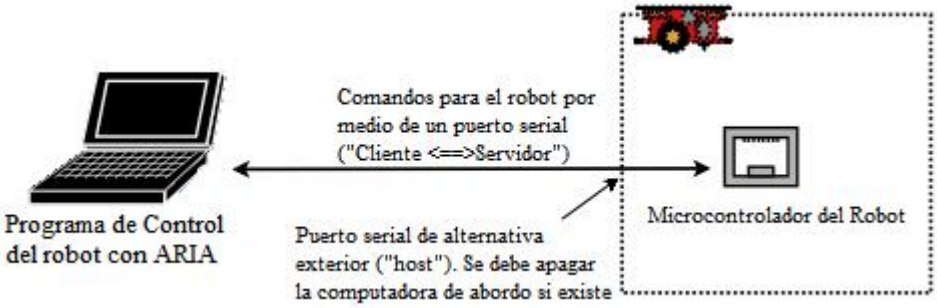
Una de las funciones más importantes de ARIA y una de las cosas más importantes que el programa de aplicación debe realizar es establecer la conexión entre la instancia del objeto *ArRobot* y el sistema operativo de la plataforma del robot (firmware).

Además del robot móvil, algunos accesorios como el sonar, el sujetador del Pioneer, las cámaras PTZ, el brazo del Pioneer, el compás y otros, están internamente conectados al microcontrolador auxiliar (AUX) o líneas digitales de entrada/salida del robot, y también utilizan la conexión del robot (por lo tanto, las clases de interfaz para estos objetos requieren una referencia hacia un objeto *ArRobot* que debe estar conectado para que trabajen los dispositivos). Otros accesorios, como el *SICK Laser*, las tarjetas para captar video, etc., están conectadas directamente a bordo de la computadora.

Existen diferentes formas de conectar una computadora (ejecutando ARIA) al microcontrolador del robot o al simulador. La figura 4.5 provee una vista esquemática de las opciones de comunicación ARIA-Robot que se han utilizado en esta investigación.




(a) Comunicación ARIA-Simulador MobileSim




(b) Comunicación ARIA-Robot Pioneer

Figura 4.5. Esquemas de comunicación de ARIA.

### 4.2.2. Estableciendo una comunicación con el robot o con el simulador

La clase *ArSimpleConnector* puede ser utilizada para realizar la comunicación con el robot, basándose en una configuración de tiempo de ejecución vía argumentos de comandos en línea y un ambiente detectado. Entre otros beneficios, *ArSimpleConnector* tratará primero de conectarse a un simulador por medio de un puerto TCP local, y si el simulador no está en ejecución, entonces se tratará de conectar al robot por medio de un puerto serial local. Esto hace más fácil la manera de desarrollar y depurar un programa por medio del simulador, entonces simplemente se copia el programa en la computadora del robot y se ejecuta sin ninguna modificación. La Figura 4.6 muestra un ejemplo sencillo del uso de la clase *ArSimpleConnector* para conectar las clases *ArRobot* y *ArSick* a su respectivo hardware

(*ArRobot* al robot vía puerto serial o al simulador vía puerto TCP, y el objeto *ArSick* al láser vía el puerto serial del láser).

```
#include "Aria.h"
int main(int argc, char** argv)
{
  Aria::init();
  ArSimpleConnector connector(&argc, argv);
  ArRobot robot;
  ArSick sickLaser;

  //Checa argumentos de línea de comandos utilizada por SimpleConnector
  if(!connector.parseArgs())
  {
    connector.logOptions();
    exit(1);
  }

  //Conecta con el robot, ejecuta los dispositivos en hilos de segundo
    plano y conecta al  oubl
  if(!connector.connectRobot(&robot))
  {
    // Error!
    …
  }
  robot.runAsync(true);
  sickLaser.runAsync();
  if(!connector.connectLaser(&sickLaser))
  {
    // Error!
    ...
  }

  //Conexión establecida, los objetos robot y sicklaser están en
    ejecución en hilos de segundo plano
  …
```

Figura 4.6. Ejemplo de conexión de ARIA

### 4.2.3. Comandos cliente y servicio de paquetes de información

Las comunicaciones cliente-servidor entre ARIA y una plataforma de robot móvil o simulador utilizan protocolos basados en paquetes. (En este contexto, el cliente es el software utilizando ARIA para operar el robot, y el servidor es el firmware de la plataforma del robot). La clase *ArRobot* (por medio de las clases *ArDeviceConnection*, *ArRobotPacketReceiver*, *ArRobotPacketSender*, *ArRobotPacket*, y A*rSerialConnection*) maneja los detalles de

construcción y envío de paquetes de comandos hacia el robot, así como la recepción y decodificación de paquetes recibidos desde el servidor del robot.

El Servidor de Paquetes de Información (SIPs) envía paquetes para el servidor del robot conteniendo actualizaciones de información acerca del robot y sus accesorios. El SIP estándar es enviado por el robot a un cliente conectado cada 100 milisegundos. Estos paquetes contienen la posición actual del robot y la estimación de las velocidades actuales traslacionales y rotacionales, las actualizaciones de lectura de los sonares, el voltaje de la batería, los estados de entrada/salida análoga y digital, y más. Estos datos son almacenados y utilizados por el *state reflection* de *ArRobot* y son accesibles vía métodos de la clase *ArRobot*.

### 4.2.4. Paquetes de Comandos

Para controlar el robot, un programa cliente envía paquetes de comandos a través de la conexión del robot y utiliza, de la clase *ArRobot*, las funciones de comandos de movimiento, acciones, o en el nivel más básico, comandos directos que dan como resultado paquetes de comandos enviados al robot. Cabe aclarar que puede existir un conflicto si se utilizan tanto acciones como comandos de movimiento al mismo tiempo.

### 4.2.5. Ciclo de sincronización del robot

El SIP estándar se envía en un ciclo constante, y su recepción dispara una nueva iteración del ciclo de procesamiento de tareas de *ArRobot*, el cual consiste de una serie de tareas sincronizadas en las que se incluyen: el manejo de paquetes SIP, la invocación de tareas de interpretación de los sensores, el manejo de acciones y resolución, el estado de reflexión, y la invocación de tareas de usuario, en el orden en que se mencionan. Dado que el ciclo de tareas se dispara (normalmente) por la recepción de cada SIP (a menos que el robot comience a fallar en el envío de SIPs o el ciclo de tareas no este explícitamente asociado a la conexión del robot, Figura 4.7), cada tarea es invocada en un orden predecible con los datos más recientes

para actuar; ninguna tarea perderá la oportunidad de utilizar un SIP; y como las tareas no toman mucho tiempo en ejecutarse, cada SIP es manejado tan pronto como sea posible después de que el robot lo manda.

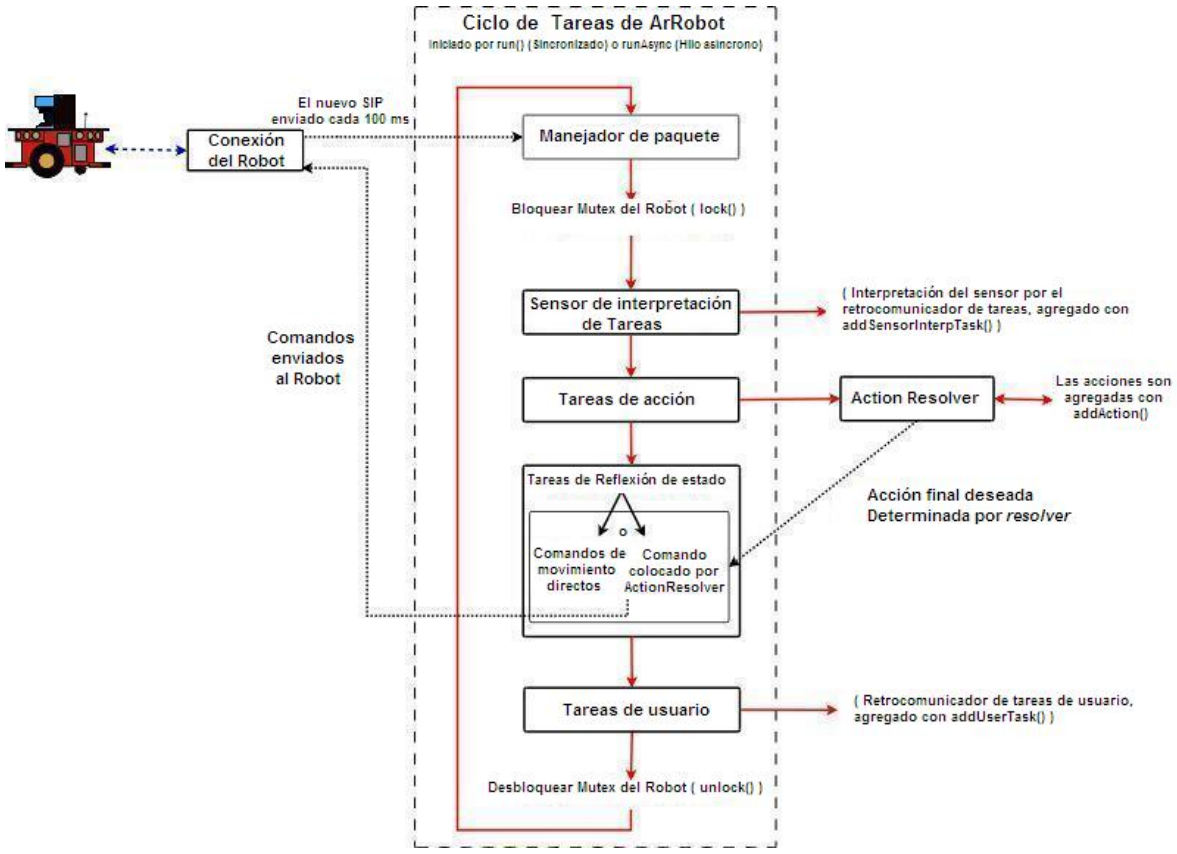



Figura 4.7. Ciclo de tareas de ArRobot

Para comenzar el ciclo de procesamiento, se puede utilizar *ArRobot::run()* para entrar al ciclo sincrónicamente, o *ArRobot::runAsync()* para ejecutar el ciclo en un nuevo hilo en segundo plano. *ArRobot::stopRunning()* para el ciclo de procesamiento.

*ArRobot* provee métodos que permiten agregar interpretadores de sensores propios y retro-comunicadores de tareas de usuario. Para agregar un retro-comunicador de tareas, se debe crear un objeto *ArFunctor*, y agregar este utilizando *ArRobot::addSensorInterpTask()* o *ArRobot::addUserTask()*.

También es posible ejecutar el ciclo de procesamiento sin una conexión con el robot. Este ciclo alternativo tiene su propio tiempo de 100 milisegundos el cual puede ser examinado y reiniciado con *ArRobot::getCycleTime()* y *ArRobot::setCycleTime()*. Durante el ciclo de sincronizado normal, *ArRobot* espera hasta dos veces el tiempo del ciclo de un SIP estándar del robot antes de abandonar y comenzar el ciclo automáticamente.

### 4.2.6. Reflexión de estado del robot

La reflexión de estado en la clase *ArRobot* es la forma en cómo Aria mantiene una vista de las condiciones operativas del robot y sus valores, tales como posición estimada, velocidad actual, voltaje de la batería, etc., tal como se extrajo del último SIP estándar. El SIP estándar también contiene actualizaciones de lectura del sonar, los cuales se reflejan en *ArRobot* y se examinan con métodos como *ArRobot::getSonarRange()*, y *ArRobot::getSonarReading()*. La clase de interfaz del sonar *ArSonarDevice* también recibe esta información.

*ArRobot* también utiliza la tarea de la reflexión de estado para enviar previamente comandos de movimiento requeridos para el robot, de manera que los comandos reflejan esos valores deseados establecidos en la reflexión de estado de *ArRobot* por métodos de acciones o comandos de movimiento, y también de manera que el vigilante (watchdog) en el robot no este fuera de tiempo y deshabilite el robot ( si ningún comando de movimiento es establecido, se envía un *ArCommands::PULSE* en cada ciclo). Si se desea, se puede cambiar el comando de movimiento de la reflexión de estado

### 4.2.7. Controlando el robot con comandos y acciones

En ARIA existen tres formas de manipular un robot y sus accesorios:

1) utilizando comandos directos de la clase *ArRobot*.
2) comandos de movimiento.
3) acciones.

**Comandos directos**

En el más bajo nivel de acceso al robot, se puede mandar cualquier paquete de comandos directamente al robot o plataforma de simulación a través de la clase *ArRobot*. Comandos directos consisten de un numero de comando de 1 byte seguido por ninguno o varios argumentos, como se define en el sistema operativo del robot (ARCOS, AROS, P2OS, AmigOS, etc). Por ejemplo, el comando numero 4: ENABLE, habilita los motores del robot si este es acompañado por el argumento 1, y deshabilita los motores con el argumento 0. La clase *ArCommands* contiene un enum con todos los comandos directos, por ejemplo *ArCommands::ENABLE*. No todos los robots de MobileRobots y ActiveMedia Robotics soportan todos los comandos directos, por lo que los comandos que no sean soportados, serán simplemente ignorados.

**Funciones de comandos de movimiento**

Son comandos posicionados un nivel arriba de los comandos directos de *ArRobot*. Estos son explícitos comandos de movimiento simples envidos por la tarea de reflexión de estados de *ArRobot*. Por ejemplo, *ArRobot::setVel()* establece la velocidad de traslación, *ArRobot::setRotVel()* establece la velocidad rotacional, *ArRobot::setVel2()* establece la velocidad de cada rueda por separado, *ArRobot::setHeading()* para establecer un ángulo de giro para una dirección global, *ArRobot::Move()* para mover el robot una distancia dada, *ArRobot::stop()* para parar todo el movimiento.

Se debe estar consciente que los comandos de movimiento y comandos directos pueden causar conflictos con controles de acciones u otros procesos de nivel superior y llevar a consecuencias inesperadas. Para esto, se debe utilizar *ArRobot::clearDirectMotion()* para cancelar el efecto de sobrecarga de comandos de movimiento establecidos previamente, de manera que la acción que se va a ejecutar sea capaz de retomar el control del robot. O se puede limitar el tiempo de un comando de movimiento de forma que prevenga otras acciones con *ArRobot::setDirectMotionPrecedenceTime()*. De otra forma, el comando de movimiento prevendrá acciones para siempre.

También se puede utilizar *ArRobot::getDirectMotionPrecedenceTime()* para ver que tanta precedencia tiene un comando una vez establecido.

**Acciones**

Mientras las secuencias simples de comandos de movimiento pueden ser fáciles de usar, tratar de lograr movimientos más sofisticados por medio de estos se puede volver rápidamente complicado. Para hacer esto posible se deben definir comportamientos complejos independientes y componentes reutilizables, por lo que ARIA provee el sistema de alto nivel *Actionsi*. Las acciones son objetos individuales que proveen de manera independiente requisiciones de movimiento, las cuales son evaluadas y entonces combinadas durante cada ciclo para producir un conjunto final de comandos de movimiento. Esto permite construir comportamientos complejos a partir de simples bloques para controles de movimientos continuos y dinámicos.

Las acciones son definidas por la creación de una subclase de *ArAction*, la clase base que sobrecarga el método *ArAction::fire()*. Las acciones son agregadas a un objeto *ArRobot* con *ArRobot::addAction()*, junto con una prioridad que determina su posición en la lista de acciones. Un llamado a *ArAction::setRobot()* se establece al momento de agregar al robot el objeto *action*. Las acciones son evaluadas por el *action resolver* de **ArRobot** en orden descendiente de prioridad (máxima prioridad primero, mínima prioridad hasta el final), en cada ciclo de tareas previo al *state reflection*. El *action resolver* invoca a cada acción del método *fire()*, combinando sus comandos de movimiento deseados (los objetos *ArActionDesired* que ellos retornan) en un simple objeto *ArActionDesired*, el cual es entonces utilizado por *state reflection* para enviar comandos de movimiento al robot.

### 4.2.8. Funciones de Aria

Algunas de las clases y métodos principales para el movimiento del Pioneer que se han utilizado en el laboratorio de visión y robótica (LVR) son los siguientes:

*void Aria::init()*: Ejecuta una configuración de socket inicial especifica del sistema operativo. Debe ser llamada antes que cualquier otra función de Aria.

*void Aria::exit(int error)*: Permite a Aria a salir de todos sus hilos y procesos, asociando un numero de error.

*void Aria::shutdown()*: Apaga o cierra todos los procesos o hilos de Aria. Internamente llama a la función *stop()*, para detener todos los hilos y tareas.

*bool ArSimpleConnector::connectRobot(ArRobot *robot)*: Permite indicar el robot al que se desea conectar.

*bool ArRobot::addAction(ArAction *action ,int priority)*: Esta función permite agregar una acción al robot, esto lo realiza mediante el paso de un puntero a un objeto *ArAction* que es la acción a tomar cuando se presenta determinado evento, y su prioridad de uso.

*void ArRobot::run(bool stoprunifnotconnected)*: Inicia la instancia para procesar en este hilo. Si el parámetro *stoprunifnotconnected* es *true*, si inicia la instancia conectando al robot, si es *false* no lo hace.

*void ArRobot::runAsync(bool stoprunifnotconnected)*: Inicia la instancia para procesar en un nuevo hilo.

*void ArRobot::setVel(double r1, double r2)*: Establece la velocidad de tracción de ambas ruedas a la vez.
*void ArRobot::setRotVel()*: Permite establecer la velocidad de rotación de ambas ruedas a la vez.

*void ArRobot::setHeading(double heading)*: Permite fijar un ángulo para desplazar al robot, esto es, permite al robot rotar en su propio eje, *heading* grados.

*void ArRobot::setDeltaHeading(double delta)*: Permite fijar una cierta cantidad delta de cambio en la cabecera del robot.

*void ArRobot::setVel2(double leftvelocity, double rightvelocity)*: Permite fijar las velocidades de las ruedas del robot de forma independiente. Donde *leftvelocity*, es la velocidad asignada para la rueda izquierda en mm/s; y *rightvelocity*, es la velocidad asignada para la rueda derecha en mm/s. Esto es funcional en muchos casos cuando se requiere que el robot avance con cierto ángulo de desplazamiento al mismo tiempo que recorre una distancia.

*void ArRobot::stop()*: Permite al programador detener al robot completamente.

*bool ArRobot::setAbsoluteMaxRotVel(double vel)*: Fija la velocidad máxima de rotación. Esta velocidad es la máxima que el robot podrá utilizar y no será permitido fijar una velocidad de rotación mayor a este valor. Regresa *true* si el valor a fijar es bueno, *false* en otro caso.

*bool ArRobot::setAbsoluteMaxTransVel(double vel)*: fija la velocidad máxima de traslación. Esta velocidad es la máxima que el robot podrá utilizar y no será permitido fijar una velocidad de traslación mayor a este valor. Regresa *true* si el valor a fijar es bueno, *false* en otro caso.

*void ArUtil::sleep(unsigned int milisegundos)*: Permite a cualquier robot de Aria estar en estado inactivo o simplemente no aceptar ninguna instrucción por un tiempo dado en milisegundos.

### 4.2.9. Ejemplo de ARIA

En la figura 4.8 se muestra un ejemplo para mover al robot de forma con comandos directos como los detallados previamente.

```
#include "Aria.h"
#include <conio.h>
#include <string.h>

int main(int argc, char **argv){
```

```
  ArRobot robot;  // robot

  ArTime start;
 // Sonar utilizado para los limites, para que este trabaje, debe ser
 //agregado al robot
 ArSonarDevice sonar;

 ArSimpleConnector connector(&argc, argv);

 connector.parseArgs();

 if (argc > 1)  {
    connector.logOptions();
    exit(1);
 }

  Aria::init();// Inicializar

  printf("Este programa permite que se utilice el joystick o el teclado
para controlar el robot. Se pueden utilizar las flechas del cursor para
navegar, y la barra espaciadora para parar, para el control por
joystick presione el botón de disparo y entonces navegue, presione Esc
para salir.\n\n");

  // Agrega el sonar al robot
  robot.addRangeDevice(&sonar);

  // trata de conectarse, si falla se sale
  if (¡connector.connectRobot(&robot)) {
    printf("No se puede conectar al robot... saliendo\n");
    Aria::shutdown();
    return 1;
  }

  // Establece la velocidad máxima del robot (el sonar no trabaja bien
  //si se va a rápida velocidad)
  robot.setAbsoluteMaxTransVel(200);

  robot.comInt(ArCommands::ENABLE, 1); //habilitar los motores,
                                       //deshabilita sonidos amigobot

  printf("\n Fijando velocidad 200");
  robot.setVel2(200,200);

  robot.lock();
  robot.setHeading(45);
  robot.unlock();
  getchar();
  robot.run(true);
  Aria::shutdown();
  return 0;
}
```

Figura 4.8. Ejemplo con comandos directos de ARIA

## 4.3. El simulador MobileSim

MobileSim (figura 4.9) es un software para simular robots móviles y sus ambientes, permitiendo depurar y experimentar con programas de control basados principalmente en ARIA. MobileSim esta basado en la librería Stage, creada por el proyecto Placer/Stage/Gazebo. Su manera de trabajar consiste en la carga de un mapa de ActiveMedia (un archivo .map creado por Mapper3, o Mapper3Basic) en el Stage como el ambiente de simulación, y entonces coloca un robot simulado en ese ambiente. Después se proporciona un control para manipular un Pioneer simulado por una conexión accesible vía puerto TCP 8108 (similar a la conexión real del puerto serial del Pioneer ). La mayoría de los programas basados en ARIA se conectarán automáticamente a ese puerto TCP si está disponible.

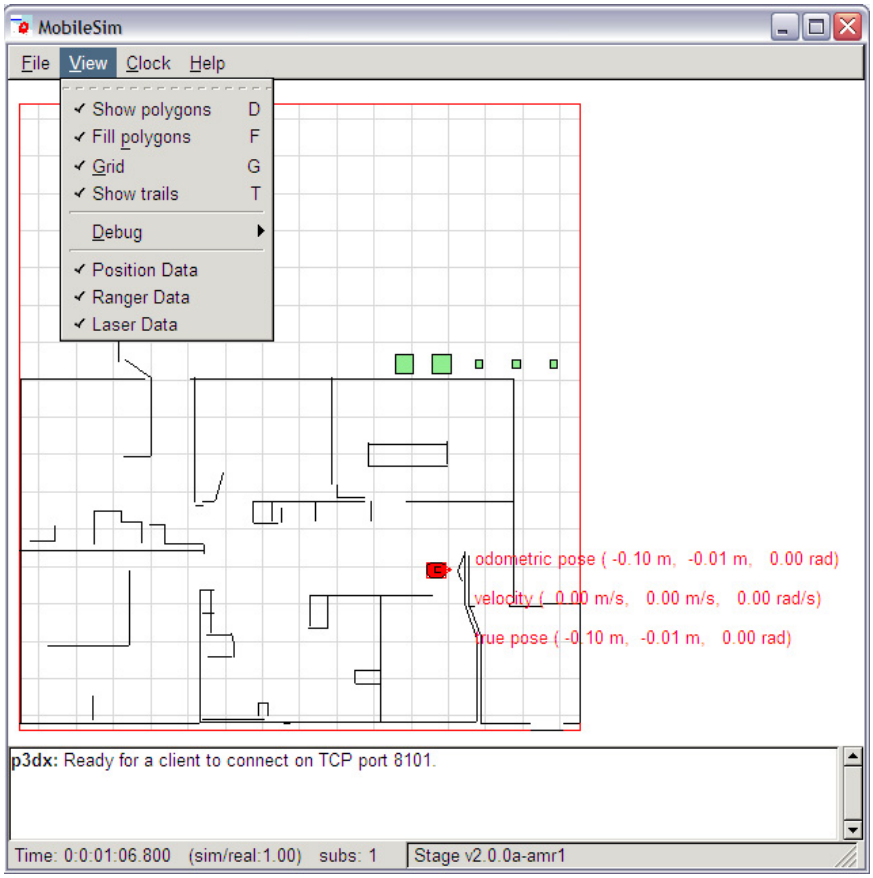



Figura 4.9. Pantalla principal del simulador MobileSim

## 4.4. El editor de mapas Mapper3

Mapper3 es una aplicación de interfaz de usuario gráfica (GUI) para el despliegue y edición de archivos de tipo mapa para robots MobileRobots y ActiveMedia Robotics que puedan ser simulados en MobileSim y Saphira. Estos robots utilizan estos archivos de mapas para localizar, establecer planeación de navegación y localizar metas, puntos de estancamiento y tareas basadas en localizaciones.

El editor permite crear ambientes con base en líneas para formar polígonos y el establecimiento de puntos de salida, metas y estancamientos. La interfaz gráfica de esta aplicación puede observarse en la figura 4.10.

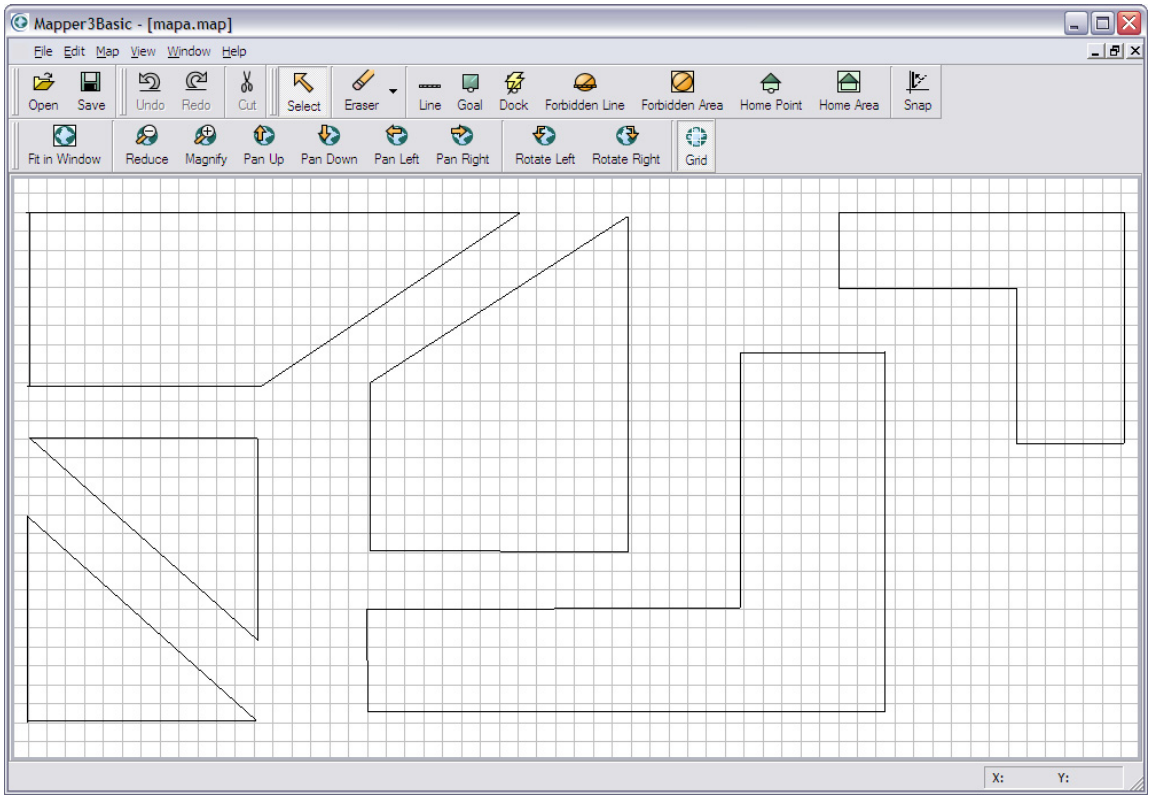



Figura 4.10. Pantalla principal de Mapper3Basic

## 4.5. El planeador de rutas local mediante acciones de ARIA

En la implementación del planeador de rutas local basado en [Ramírez y Zeghloul 2000], se desarrollaron las siguientes acciones:

*ArActionLimiterForwards*

Acción que limita el movimiento hacia delante del robot basado en las lecturas del sensor de rango. Utiliza el sensor para encontrar una velocidad máxima hacia adelante a la cual viajar, cuando el sensor de rango (sonar, láser, etc.) detecta obstáculos más cerca que los parámetros dados, se solicita al robot que desacelere o pare (figura 4.11).

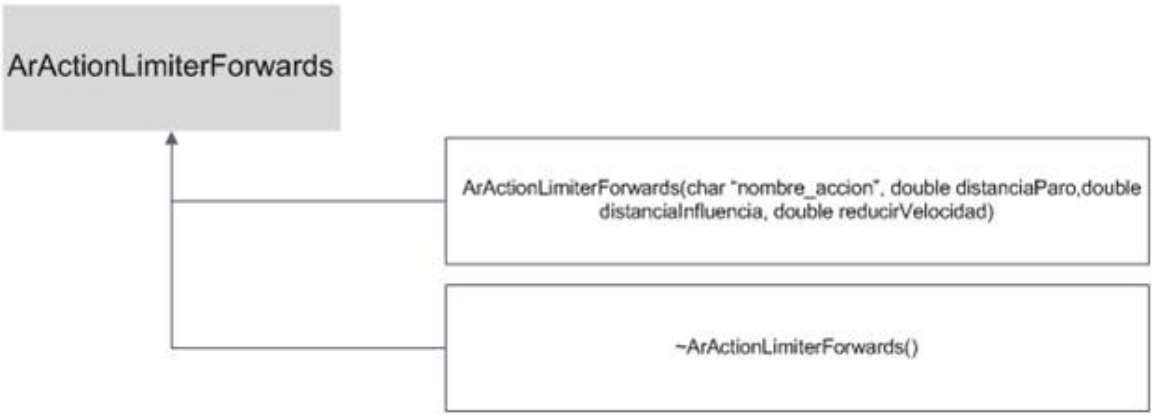



Figura 4.11. Diagrama de clase de *ArActionLimiterForwards*

*ArActionLimiterTableSensor*

Acción que limita la velocidad (y paro) basado en si la tabla de sensores registra algo. De ser así, limita la velocidad a 0. Esta acción trabajará solo si el robot tiene tabla de sensores, lo cual significa que el archivo de parámetros del robot debe tener establecidos los parámetros de sensores a *true* (figura 4.12).

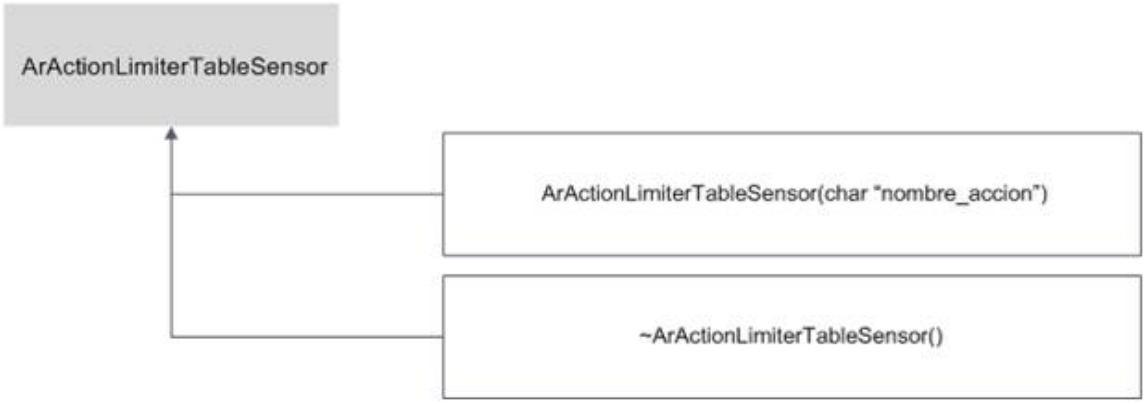



Figura 4.12. Diagrama de clase de *ArActionLimiterTableSensor*

*ArActionAvoidFront*

Su función principal es la de evadir obstáculos de frente (figura 4.13). Se basa en las siguientes acciones:

1) Empieza la ejecución cuando se encuentra un obstáculo en frente del robot.
2) Checa si la distancia de lectura es menor que la distancia de tolerancia con el obstáculo.
3) Determina hacia que lado es posible avanzar (es decir, que la distancia de tolerancia es mayor) y gira *n* grados ya sea en contra de las manecillas del reloj o a favor.

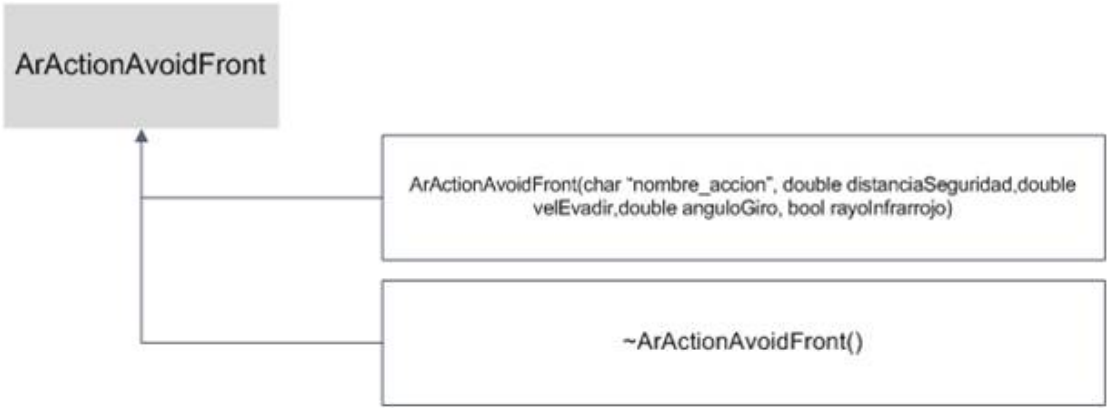



Figura 4.13. Diagrama de clase de *ArActionAvoidFront*

*ArActionAvoidSide*

Su función principal es la de evadir de lado los obstáculos (figura 4.14):

1) Empieza la ejecución cuando se encuentra un obstáculo en los lados del robot.
2) Checa si la distancia de lectura es menor que la distancia de tolerancia con el obstáculo.
3) Determina hacia que lado es posible avanzar (es decir, cual distancia de tolerancia es mayor) y gira *n* grados ya sea en contra de las manecillas del reloj o a favor.

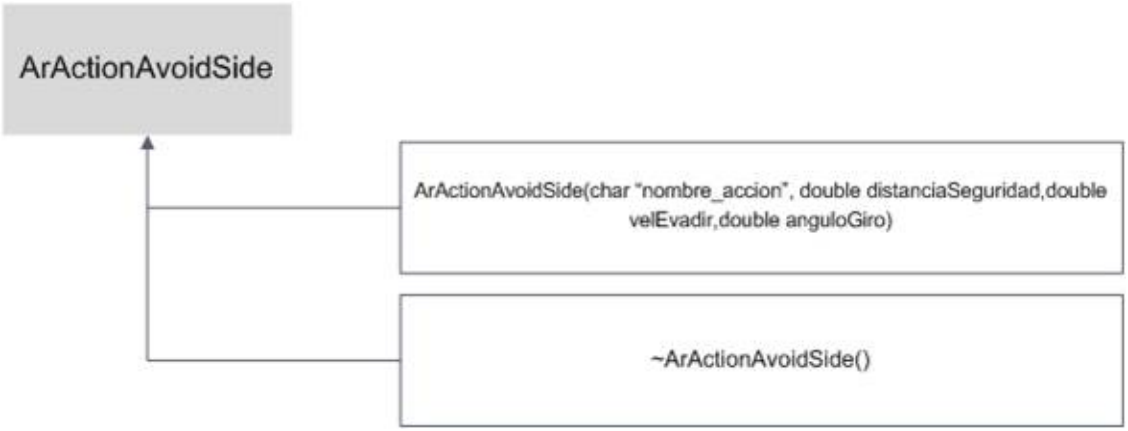



Figura 4.14. Diagrama de clase de *ArActionAvoidSide*

*ArActionGoto*

Su función principal es la de trasladar al robot a una determinada posición *Arpose* (figura 4.15). Sus acciones principales son:

1) Encuentra un ángulo de desplazamiento, según la posición de inicio y meta.
2) Calcula la distancia en línea recta desde el punto inicio hasta la posición meta.
3) Avanza hacia la meta hasta que topa con un obstáculo.
4) Cuando se encuentra un obstáculo, se detiene el robot.

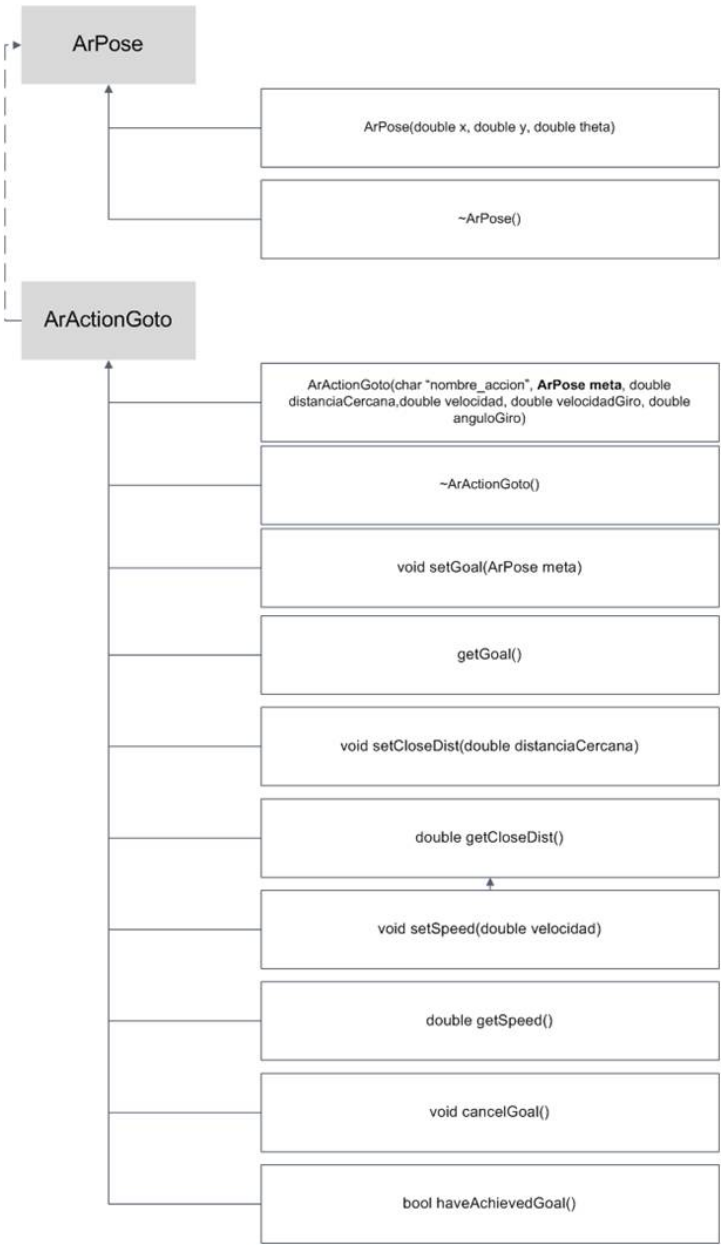



Figura 4.15. Diagrama de clase de *ArActionGoto*

El algoritmo 4 (figura 4.16) muestra una definición del planeador de rutas local mediante acciones de ARIA y comandos de movimiento (Para una descripción más detallada de la ley de control y Acciones de evasión de obstáculos, consulte el anexo C, que contiene la programación de las funciones y clases más importantes de los módulos en cuestión, o en su caso, consulte la programación contenida en el CD anexado a esta investigación).

**Algoritmo 4:** Planeador de rutas local mediante acciones de ARIA
**Entrada:** un punto del robot con un sensor táctil
**Salida:** una trayectoria hacia el punto meta o una conclusión de que la trayectoria no existe

1: Declarar objetos de conexión y manipulación para **robot**
   Objetos de las clases *ArRobot, ArTime, ArKeyHandler, ArSonarDevice y ArSimpleConnector*
2: Declarar **acciones** para evadir obstáculos
   Acciones de las clases *ArActionLimiterForwards, ArActionLimiterTableSensor, ArActionGoto, ArActionIrA y ArActionEvadirDeFrente*
3: Establecer prioridad a la **acciones**
   ArActionLimiterTableSensor::tableLimiterAction ← 100
   ArActionLimiterForwards::limiterAction ← 95
   ArActionLimiterForwards::limiterFarAction ← 90
   ArActionIrA::gotoPoseAction ← 50
   ArActionEvadirDeFrente::AvoidFrontAction ← 40
4: Agregar **acciones** al **robot** (objeto *ArRobot*)
5: Inicializar manejador de teclado (objeto *ArKeyHandler*)
6: Inicializar conexión con ARIA (hilo)
7: Agregar sensores al **robot** (Agregar objetos *ArSonarDevice* al objeto *ArRobot*)
8: **si no** conexión con **robot** (puerto serie – Pioneer, puerto TCP 8010 – MobileSim)
9: "No se puede conectar a **robot**"
10: Terminar conexión ARIA (Aria::shutdown)
11: Salir
12: **fin si**
13: Ejecutar **acciones** del robot en forma asíncrona (ArRobot::runAsync)
14: Habilitar motores (ArRobot::enablemotors)
15: Establecer posición **meta** (posición *ArPose* ($x$, $y$))
16: Ejecutar Ley de Control
17: Establecer restricciones de velocidad lineal y angular
18: **mientras** (ARIA este en ejecución – hilo)
19: Bloquear **robot** (solo se ejecutara lo que este entre bloquear y desbloquear)
20: **si** se logro la meta
21: Obtener posición actual de **robot**
22: Imprimir posición actual de **robot**
23: Detener **robot**
24: Salir del ciclo **mientras**
25: **fin si**
26: Realizar navegación incremental con las acciones de ARIA
27: Obtener posición actual de **robot**
28: Imprimir posición actual de **robot**
29: Desbloquear **robot**
30: **fin mientras**
31: Obtener posición final de **robot**
32: Imprimir posición final de **robot**
33: Finalizar conexión con ARIA (hilo)

Figura 4.16. Algoritmo del Planeador de rutas local mediante ARIA

*Tercera ley de la robótica: Un robot debe proteger su propia existencia, siempre y cuando dicha protección no entre en conflicto con la primera y segunda ley*
*Enciclopedia Galáctica, manual de robótica, 56ª edición. Año 2058 D.C.*

# Capítulo 5

# Pruebas y Resultados

Durante el desarrollo de la presente investigación, se realizó la siguiente experimentación:

1) Pruebas utilizando el simulador MobileSim y la interfaz de ARIA.
2) Pruebas en tiempo real utilizando el robot Pioneer 2DX y la interfaz de ARIA.

El hardware utilizado en las pruebas fue:

- Robot Pioneer 2DX de ActiveMedia Robotics
- Desktop, procesador Intel Celeron 2.66GHz, RAM de 512 Mb, HD de 37.2 Gb
- Laptop Toshiba M45-S359, procesador Intel Centrino 2 GHz, RAM 1Gb, HD 100 Gb.

El software necesario para realizar las experimentaciones fue el siguiente:

- Sistema Operativo Microsoft Windows XP SP2

- Matlab ver. 7.0
- Microsoft Visual C++ ver. 6.0
- Simulador MobileSim de Mobile Robots
- Interfaz Aria de Active Media Robotics y Mobile Robots
- Mapper3Basics de Mobile Robots

**Pruebas iniciales**

Como primera etapa en el desarrollo de esta investigación, se llevaron a cabo pruebas de programación en Matlab correspondientes a la ley de control (29) y (30) en un ambiente libre de obstáculos (figura 5.1).

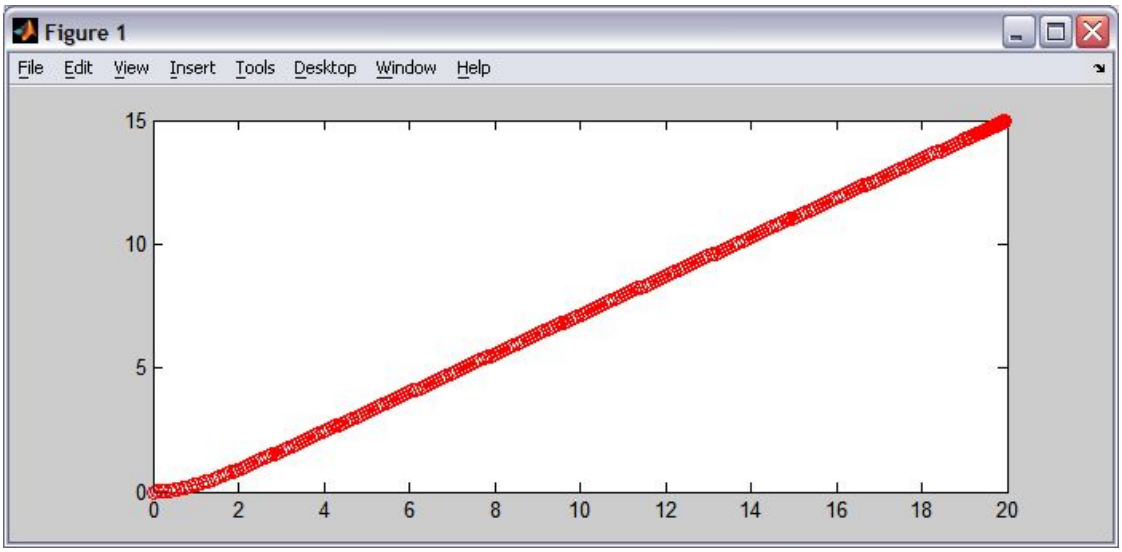



Figura 5.1. Trayectoria en Matlab generada por la ley de control (29) y (30).

Posteriormente, se realizaron pruebas de movimientos generales con el robot y el simulador MobileSim. Para ello se estuvo practicando con diferentes ejemplos sobre comandos directos, funciones de movimiento y acciones de ARIA. Se tomaron como base algunos códigos proporcionados por MobileSim y ActiveMedia Robotics para la implementación de dichas pruebas. Un modelo típico acerca de esta gamma de ejemplos es el programa que muestra la figura 5.2, donde el robot realiza una navegación a sus alrededores a una velocidad constante y trata de no chocar con los obstáculos que se encuentra en su camino.

```
#include "Aria.h"
//Este programa hace que el robot navegue por los alrededores utilizando
//algunas rutinas de evasion y a una velocidad constante.
int main(int argc, char **argv)
{
  ArRobot robot; // robot
  ArSonarDevice sonar; // sonar, debe ser agregado al robot
  // Acciones que se utilizaran para deambular
  ArActionStallRecover recover;
  ArActionBumpers bumpers;
  ArActionAvoidFront avoidFrontNear("Avoid Front Near", 225, 0);
  ArActionAvoidFront avoidFrontFar;
  ArActionConstantVelocity constantVelocity("Constant Velocity", 400);
  // Manejador de entradas de teclado
  ArKeyHandler keyHandler;
  // Inicializa ARIA
  Aria::init();
  ArSimpleConnector connector(&argc, argv);
  connector.parseArgs();
  if (argc > 1){
    connector.logOptions();
    exit(1);
  }
  // Agrega el manejador del teclado a ARIA
  Aria::setKeyHandler(&keyHandler);
  // Agrega el manejador del teclado al robot
  robot.attachKeyHandler(&keyHandler);
  // Agrega el sonar al robot
  robot.addRangeDevice(&sonar);
  // Trata de conectarse con el robot, si falla se sale
  if (!connector.connectRobot(&robot))
  {
    printf("No se pudo conectar con el robot... saliendo\n");
    Aria::shutdown();
    return 1;
  }
  // enciende los motores, apaga los sonidos de amigobot
  robot.comInt(ArCommands::ENABLE, 1);
  robot.comInt(ArCommands::SOUNDTOG, 0);
  // Agrega las acciones
  robot.addAction(&recover, 100);
  robot.addAction(&bumpers, 75);
  robot.addAction(&avoidFrontNear, 50);
  robot.addAction(&avoidFrontFar, 49);
  robot.addAction(&constantVelocity, 25);
  // comienza a caminar el robot si se pierde la conexion, el robot se para
  robot.run(true);
  // Salir de ARIA
  Aria::shutdown();
  return 0;
}
```

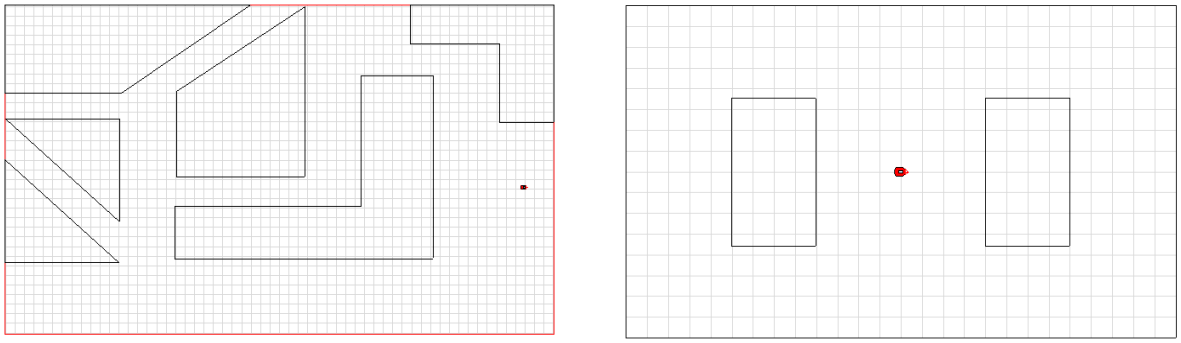
Figura 5.2. Ejemplo de acciones en ARIA

## 5.1. Pruebas con el simulador MobileSim

Durante la experimentación con el simulador MobileSim de ActiveMedia Robotics y MobileRobots, se realizaron dos tipos de pruebas:

1) Pruebas con un mismo punto de partida $\mathbf{q_0}$ y un mismo punto meta $\mathbf{q_{goal}}$.
2) Pruebas con un mismo punto de partida $\mathbf{q_0}$ y diferentes puntos meta $\mathbf{q_{goal}}$.

En cada una de las pruebas se utilizaron dos ambientes: uno completamente inventado y que simula las cuadras de una ciudad (figura 5.3 (a)), y otro construido en base a un ambiente real sencillo, simulando dos jardineras en una plaza (figura 5.3 (b)). Se realizaron seis simulaciones para cada ambiente propuesto y seis simulaciones para cada prueba.



(a) (b)

Figura 5.3. Mapas de ambientes utilizados para experimentación

Debido a que en un simulador un robot opera con condiciones ideales y no enfrenta restricciones de ningún tipo como en una situación real, se observará que en ocasiones el comportamiento del planeador de rutas en el robot simulado muestra más estabilidad que en un robot físico enfrentando un ambiente real. Sin embargo, si se toman en cuenta las delimitaciones y restricciones del robot real y del ambiente en el cual se llevará a cabo la navegación, se garantiza el desempeño del planeador de una manera eficiente.

### 5.1.1. Experimento 1: Pruebas con mapa 5.3 (a), mismo punto de partida $q_0$ y mismo punto meta $q_{goal}$.

**Objetivo**

Dado un robot móvil del tipo Pioneer 2DX, realizar una navegación incremental (mediante simulación) entre dos puntos, inicio y meta, utilizando el ambiente de la figura 5.3 (a) y aplicando el planeador de rutas local y el método del FVP.

**Procedimiento**

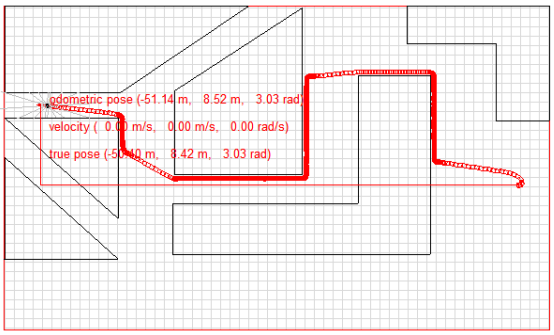
Utilizar el Simulador MobileSim para realizar seis pruebas de navegación incremental entre dos puntos del ambiente de la figura 5.3 (a). Partir de un mismo punto $q_0$ y tener como objetivo el mismo punto meta $q_{goal}$. Para la navegación y evasión de obstáculos se aplica el planeador de rutas local y el método del FVP utilizados en esta investigación. Se registra el tiempo de cada trayectoria y se indica si el robot logró su objetivo en cada una de las pruebas.

**Resultados**

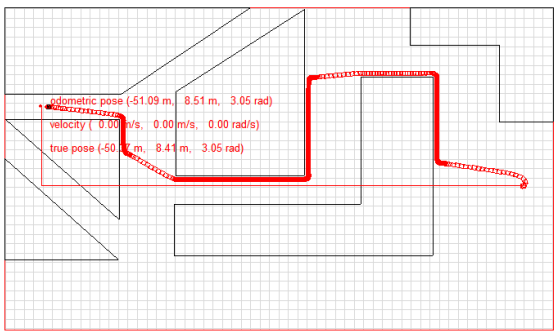
Las trayectorias generadas por esta prueba se presentan en la tabla 5.1., y en la figura 5.4.

Tabla 5.1. Trayectorias con mapa 5.3 (a), mismo punto $\mathbf{q_0}$ y mismo punto $\mathbf{q_{goal}}$

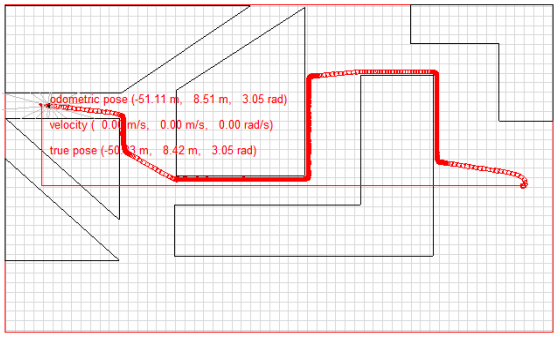
| No | Punto Inicio ($x$, $y$) | Punto Meta ($x$, $y$) | ¿Meta Alcanzada? | Tiempo |
|---|---|---|---|---|
| 1 | 0, 0 | -51.00, 8.50 | Si | 13:00 |
| 2 | 0, 0 | -51.00, 8.50 | Si | 12:00 |
| 3 | 0, 0 | -51.00, 8.50 | Si | 12.20 |
| 4 | 0, 0 | -51.00, 8.50 | Si | 12:43 |
| 5 | 0, 0 | -51.00, 8.50 | Si | 13:55 |
| 6 | 0, 0 | -51.00, 8.50 | Si | 15:00 |



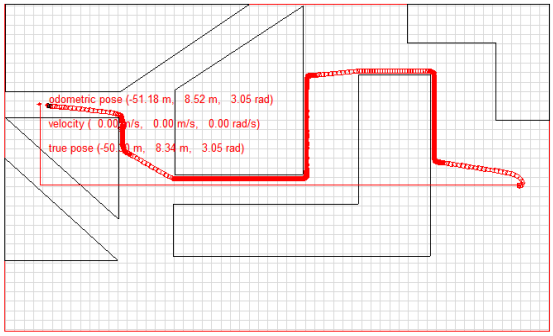

Trayectoria 1



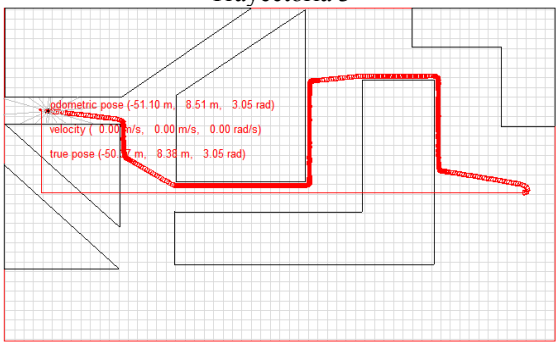

Trayectoria 2



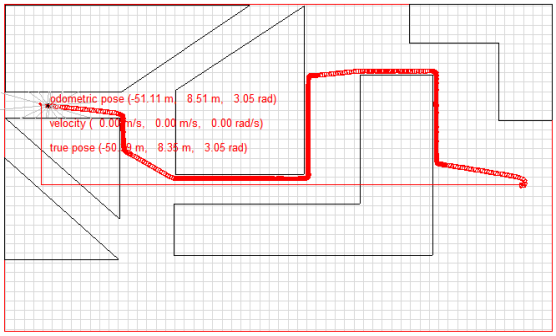

Trayectoria 3




Trayectoria 4




Trayectoria 5




Trayectoria 6

Figura 5.4. Trayectorias con mapa 5.3 (a), mismo punto $\mathbf{q_0}$ y mismo punto $\mathbf{q_{goal}}$

**Análisis de resultados**

La tabla 5.1 nos muestra un resultado exitoso en la todas las pruebas de simulación del experimento 1, lo cual puede ser corroborado en las 6 graficas de trayectorias generadas por el simulador MobileSim y mostradas en la figura 5.4. El tiempo promedio de recorrido de trayectoria es de 13:03 minutos para las seis pruebas similares.

### 5.1.2. Experimento 2: Pruebas con mapa 5.3 (a), mismo punto de partida $q_0$ y diferente punto meta $q_{goal}$.

**Objetivo**

Dado un robot móvil del tipo Pioneer 2DX, realizar una navegación incremental (mediante simulación) entre dos puntos, inicio y meta, utilizando el ambiente de la figura 5.3 (a) y aplicando el planeador de rutas local y el método del FVP.

**Procedimiento**

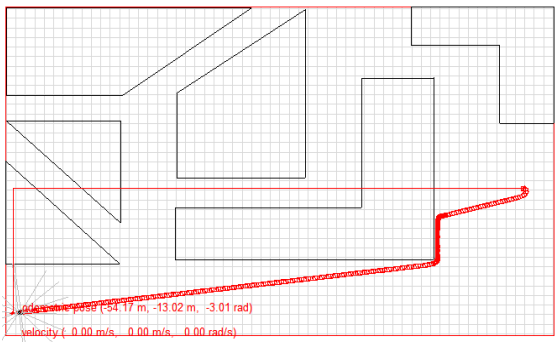
Utilizar el Simulador MobileSim para realizar seis pruebas de navegación incremental entre dos puntos del ambiente de la figura 5.3 (a). Partir de un mismo punto $q_0$ y tener como objetivo un punto meta diferente $q_{goal}$. La navegación y evasión de obstáculos resulta de la aplicación del planeador de rutas local y el método del FVP utilizados en esta investigación. Se registra el tiempo de cada trayectoria y se indica si el robot logró su objetivo en cada una de las pruebas.

**Resultados**

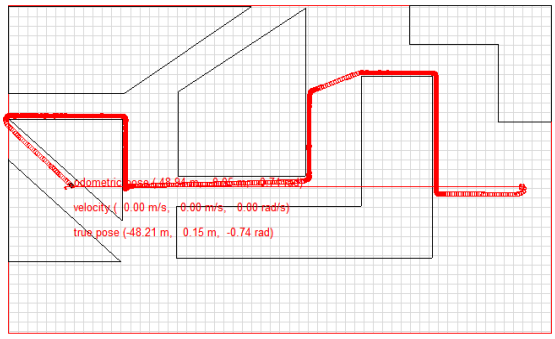
Las trayectorias generadas por esta prueba se presentan en la tabla 5.2., y en la figura 5.5.

Tabla 5.2. Trayectorias con mapa 5.3 (a), mismo punto $\mathbf{q_0}$ y diferente punto $\mathbf{q_{goal}}$

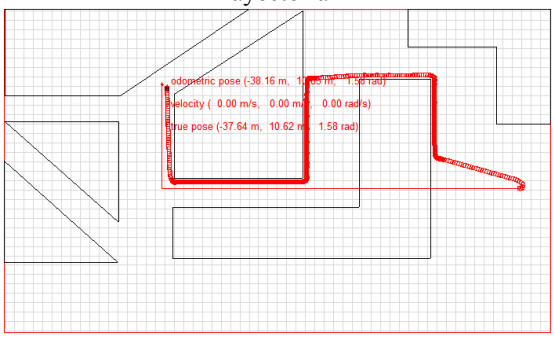
| No | Punto Inicio ($x$, $y$) | Punto Meta ($x$, $y$) | ¿Meta Alcanzada? | Tiempo |
|---|---|---|---|---|
| 1 | 0,0 | -54.00, -13.00 | Si | 03:50 |
| 2 | 0,0 | -49.00, -0.36 | Si | 25:00 |
| 3 | 0,0 | -38.16, 10.76 | Si | 14:30 |
| 4 | 0,0 | -46.00, -3.00 | Si | 07:16 |
| 5 | 0,0 | -40.00, 0.00 | Si | 10:25 |
| 6 | 0,0 | -31.42, 15.27 | Si | 03:35 |



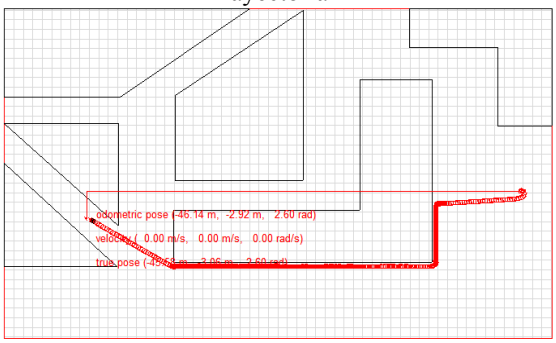
Trayectoria 1



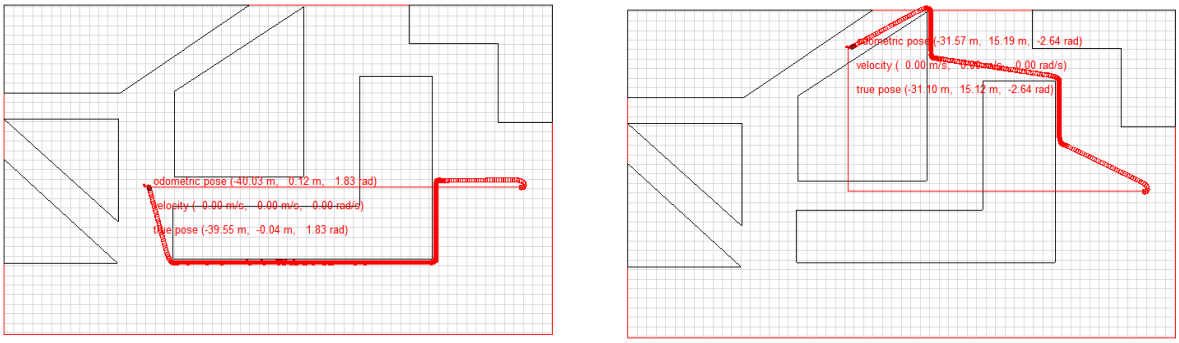
Trayectoria 2



Trayectoria 3



Trayectoria 4




Trayectoria 5 Trayectoria 6

Figura 5.5. Trayectorias con mapa 5.3 (a), mismo punto $\mathbf{q_0}$ y diferente punto $\mathbf{q_{goal}}$

**Análisis de resultados**

La tabla 5.2 nos muestra un resultado exitoso en la todas las pruebas de simulación del experimento 2, lo cual puede ser corroborado en las 6 graficas de trayectorias generadas por el simulador MobileSim y mostradas en la figura 5.5. En esta serie de pruebas no se calcula el tiempo promedio debido a que el punto meta es diferente en cada uno de los casos.

### 5.1.3. Experimento 3: Pruebas con mapa 5.3 (b), mismo punto de partida $q_0$ y mismo punto meta $q_{goal}$.

**Objetivo**

Dado un robot móvil del tipo Pioneer 2DX, realizar una navegación incremental (mediante simulación) entre dos puntos, inicio y meta, utilizando el ambiente de la figura 5.3 (b) y aplicando el planeador de rutas local y el método del FVP.

**Procedimiento**

Utilizar el Simulador MobileSim para realizar seis pruebas de navegación incremental entre dos puntos del ambiente de la figura 5.3 (b). Partir de un mismo punto $q_0$ y tener como objetivo el mismo punto meta $q_{goal}$. Para la navegación y evasión de obstáculos se aplica el planeador de rutas local y el método del FVP utilizados en esta investigación. Se registra el tiempo de cada trayectoria y se indica si el robot logró su objetivo en cada una de las pruebas.

## Resultados

Las trayectorias generadas por esta prueba se presentan en la tabla 5.3., y en la figura 5.6.

Tabla 5.3. Trayectorias con mapa 5.3 (b), mismo punto $\mathbf{q_0}$ y mismo punto $\mathbf{q_{goal}}$

| No | Punto Inicio ($x$, $y$) | Punto Meta ($x$, $y$) | ¿Meta Alcanzada? | Tiempo |
|---|---|---|---|---|
| 1 | 0, 0 | 20.60, 1.00 | Si | 04:31 |
| 2 | 0, 0 | 20.60, 1.00 | Si | 04:35 |
| 3 | 0, 0 | 20.60, 1.00 | Si | 04:10 |
| 4 | 0, 0 | 20.60, 1.00 | Si | 04:25 |
| 5 | 0, 0 | 20.60, 1.00 | Si | 03:57 |
| 6 | 0, 0 | 20.60, 1.00 | Si | 04:00 |

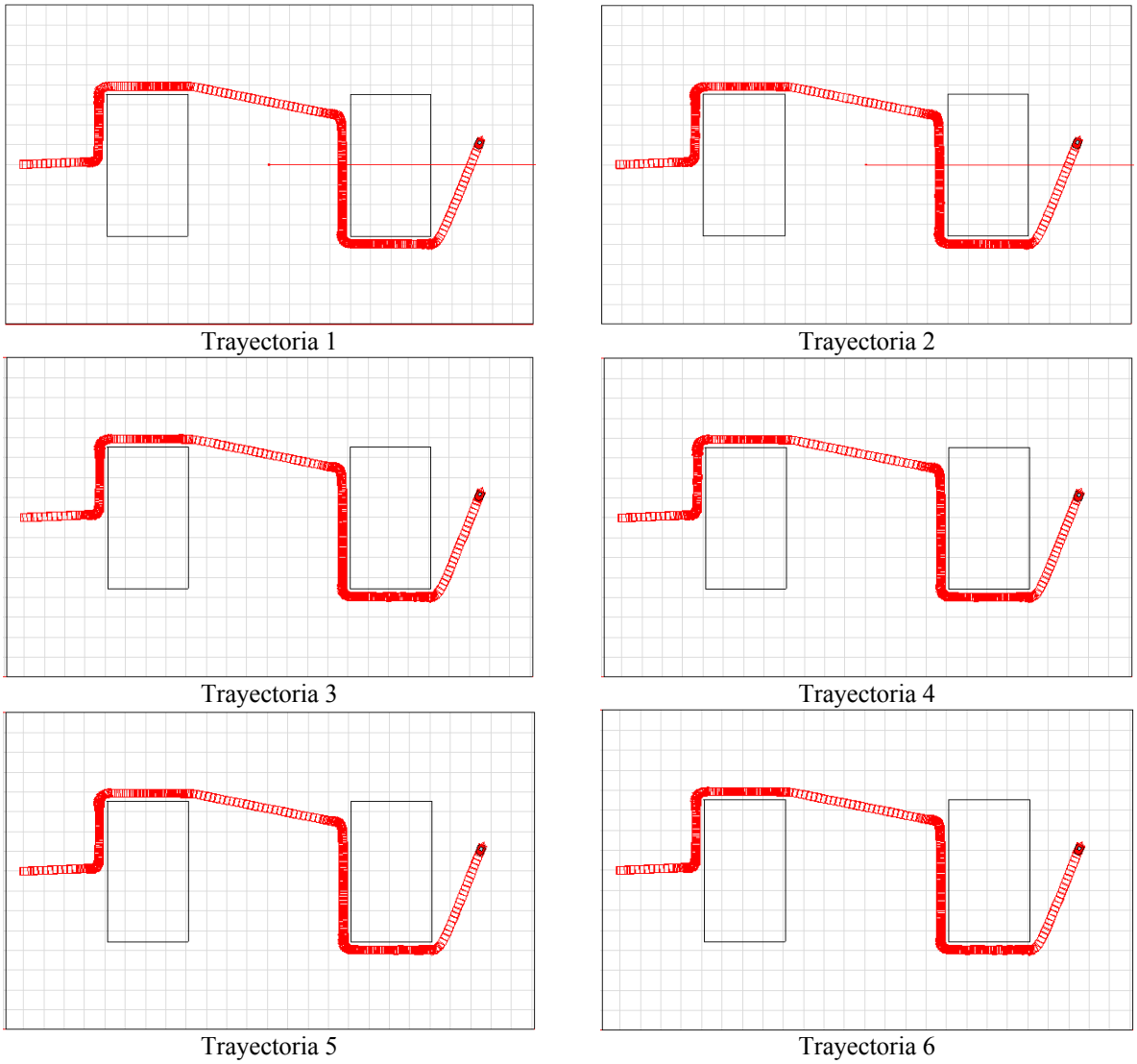



Figura 5.6. Trayectorias con mapa 5.3 (b), mismo punto $\mathbf{q_0}$ y mismo punto $\mathbf{q_{goal}}$

**Análisis de resultados**

La tabla 5.3 nos muestra un resultado exitoso en la todas las pruebas de simulación del experimento 3, lo cual puede ser corroborado en las 6 graficas de trayectorias generadas por el simulador MobileSim y mostradas en la figura 5.6. El tiempo promedio de recorrido de trayectoria es de 4:09 minutos para las seis pruebas similares.

### 5.1.4. Experimento 4: Pruebas con mapa 5.3 (b), mismo punto de partida $q_0$ y diferente punto meta $q_{goal}$.

**Objetivo**

Dado un robot móvil del tipo Pioneer 2DX, realizar una navegación incremental (mediante simulación) entre dos puntos, inicio y meta, utilizando el ambiente de la figura 5.3 (b) y aplicando el planeador de rutas local y el método del FVP.

**Procedimiento**

Utilizar el Simulador MobileSim para realizar seis pruebas de navegación incremental entre dos puntos del ambiente de la figura 5.3 (b). Partir de un mismo punto $q_0$ y tener como objetivo un punto meta diferente $q_{goal}$. La navegación y evasión de obstáculos resulta de la aplicación del planeador de rutas local y el método del FVP utilizados en esta investigación. Se registra el tiempo de cada trayectoria y se indica si el robot logró su objetivo en cada una de las pruebas.

**Resultados**

Las trayectorias generadas por esta prueba se presentan en la tabla 5.4., y en la figura 5.7.

Tabla 5.4. Trayectorias con mapa 5.3 (b), mismo punto $\mathbf{q_0}$ y diferente punto $\mathbf{q_{goal}}$

| No | Punto Inicio (*x*, *y*) | Punto Meta (*x*, *y*) | ¿Meta Alcanzada? | Tiempo |
|---|---|---|---|---|
| 1 | 0,0 | 23.00, 6.30 | Si | 01:23 |
| 2 | 0,0 | 22.00, 0.00 | Si | 06:17 |
| 3 | 0,0 | 23.00, -6.30 | Si | 01:23 |
| 4 | 0,0 | 15.00, 2.00 | Si | 01:30 |
| 5 | 0,0 | 23.50, -2.50 | Si | 03:50 |
| 6 | 0,0 | 21.00, 1.50 | Si | 03:78 |

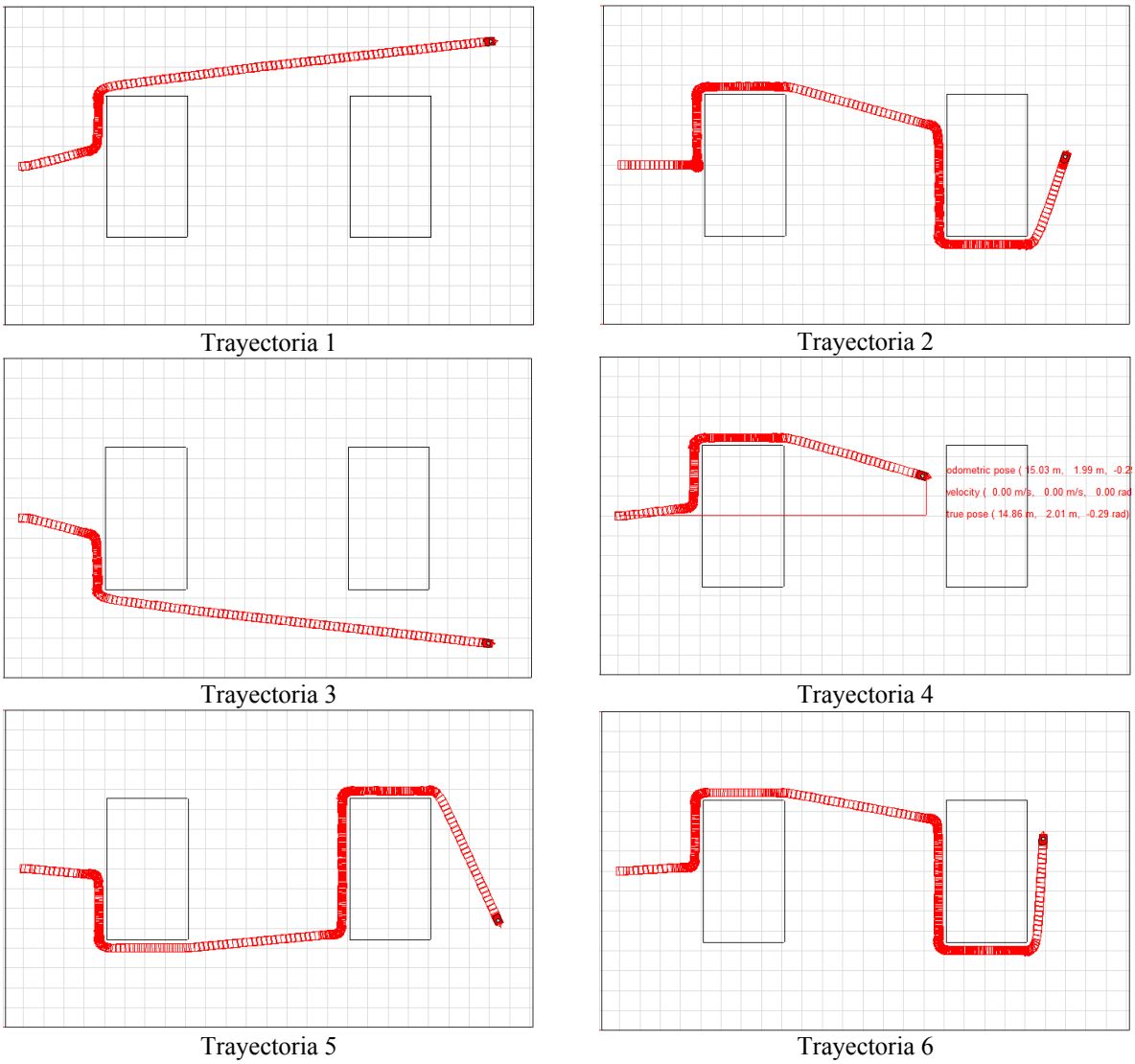



Figura 5.7. Trayectorias con mapa 5.3 (b), mismo punto $\mathbf{q_0}$ y diferente punto $\mathbf{q_{goal}}$

**Análisis de resultados**

La tabla 5.4 nos muestra un resultado exitoso en la todas las pruebas de simulación del experimento 4, lo cual puede ser corroborado en las 6 graficas de trayectorias generadas por el simulador MobileSim y mostradas en la figura 5.7. En esta serie de pruebas no se calcula el tiempo promedio debido a que el punto meta es diferente en cada uno de los casos.

## 5.2. Pruebas con el robot físico Pioneer 2DX

Una vez realizada la experimentación con el planeador de rutas local y el simulador MobileSim, se procedió a llevar a cabo la experimentación con el robot físico Pioneer 2DX (figura 4.1), utilizando una configuración de control del microprocesador del robot por medio de una laptop montada en la base del Pioneer y conectada a este por un cable serial (DB9). Al igual que en la simulación, se realizaron dos tipos de ensayo:

1) Pruebas con un mismo punto de partida $\mathbf{q_0}$ y un mismo punto meta $\mathbf{q_{goal}}$.
2) Pruebas con un mismo punto de partida $\mathbf{q_0}$ y diferentes puntos meta $\mathbf{q_{goal}}$.

Para realizar la experimentación se monto un ambiente artificial (figura 5.8), con las siguientes características:

- Se buscó una superficie lo más plana posible.
- Se construyeron objetos artificiales con forma poligonal convexa con una altura de 40cm (lo suficientemente altos como para ser detectados por los sonares).
- Se posicionaron los objetos a distancias razonables entre si, de manera que se cumplan las restricciones de distancia establecidas en las acciones de ARIA y el planeador de rutas local.

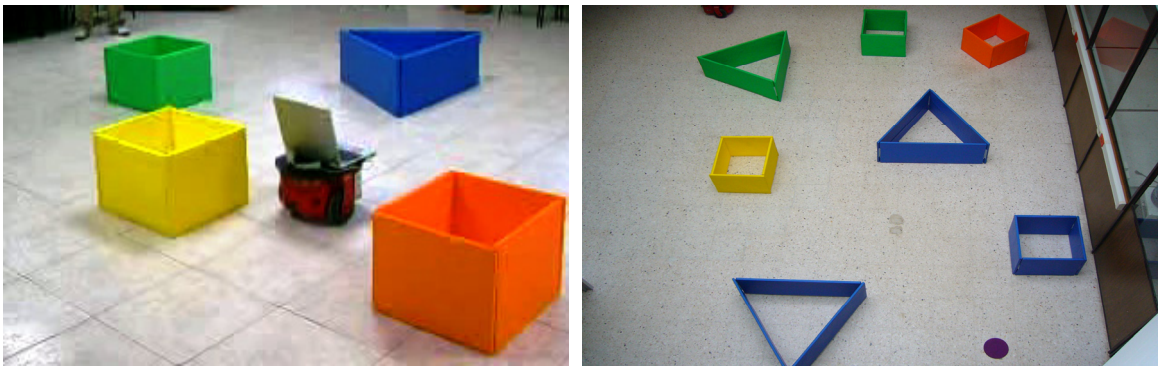


Figura 5.8. Ambiente artificial utilizado para experimentación

La figura 5.9 muestra una secuencia de fotografías pertenecientes a uno de los videos tomados al robot Pioneer DX2 durante su navegación incremental a través del ambiente artificial de la figura 5.8.

Se trazó una trayectoria sobre la secuencia de fotografías con el fin de identificar la ruta de navegación que realizó el robot al solucionar el problema de planeación de movimiento mediante el planeador de rutas local. Las secuencias de video completas y software de programación son incluidos en el CD anexo a esta investigación.

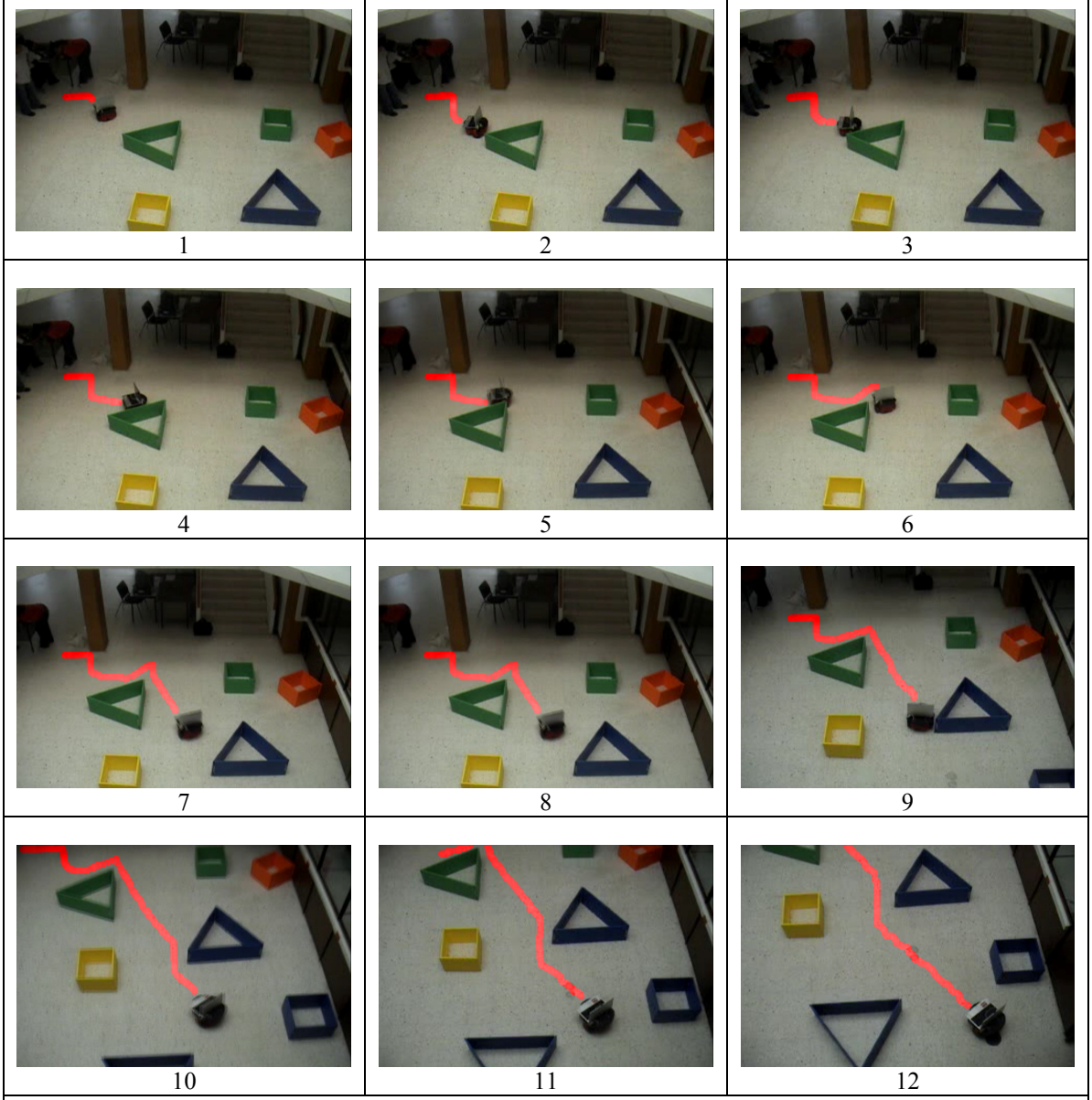



Figura 5.9. Planeación de movimiento del robot Pioneer 2DX

## 5.3. Comentarios generales sobre la experimentación

Los comentarios más relevantes sobre la investigación y experimentación a priori a este documento son los siguientes:

1) Aunque en el simulador MobileSim el robot siempre genera una trayectoria de solución en la mayoría de los casos, tanto para ambientes con objetos poligonales convexos como no convexos, con el robot físico la solución de los ambientes durante la navegación incremental es solo llevada a cabo con éxito a través de obstáculos en forma de polígonos convexos.

2) Enviar comandos de movimiento simple (por ejemplo *ArRobot::setVel()*) mientras algunas acciones de ARIA se encuentran activas puede resultar en conflictos entre los comandos de movimiento simple y los comandos del *action resolver*.

3) Al realizar pruebas de comandos en el LVR con el simulador MobileSim y el robot Pioneer 2, ambos de ActiveMedia Robotics, se observaron algunas discrepancias en el comportamiento tanto del robot simulado como del robot real. Esto es, algunas instrucciones que funcionan como se indica en la guía de referencia de ARIA en el simulador no funcionan de igual forma en el robot real. Lo anterior indica que probablemente las tareas que se llevan a cabo con éxito en el simulador no siempre sean exitosas en el robot real.

*1. La población bajo escrutinio es abstracta a la existencia de la ciencia de la Psicohistoria*
*2. Los periodos de tiempo tienen un alcance de tres generaciones*
*3. La población debe ser contada en billones para que la probabilidad estadística tenga validez Psicohistórica*
*Teoremas de Seldon sobre psicohistoria cuantitativa, conferencia de matemáticas, Trantor. Año 12020 D.C.*

# Capítulo 6

# Conclusiones

## 6.1. Conclusión

En esta investigación se desarrolló un sistema para controlar la navegación de un robot Pioneer 2DX entre dos puntos de un ambiente estático desconocido con obstáculos poligonales convexos. Esto mediante dos coordenadas base: inicio y meta, y la utilización de un método de evasión de obstáculos cuyo resultado proviene de la aplicación de la metodología utilizada por [Ramírez y Zeghloul 2000], la cual consiste en el desarrollo de un planeador de rutas local libre de colisiones. El método consta de dos módulos: el primer módulo, "alcanzando_la_meta", acerca al robot continuamente a la meta mientras evade los obstáculos siguiendo una trayectoria estable obtenida de una ley de control; el segundo módulo, "seguimiento_de_frontera", permite al robot salir de puntos de estancamiento detectados por el primer módulo. Ambos módulos están basados únicamente en el "Polígono de Velocidades Factible", el cual se construye de la información de distancia entre el robot y

los obstáculos, por lo que el método se adapta bien para cualquier robot móvil de dos ruedas equipado con un sistema de sensores de tipo sonar, láser o cámara. El planeador de trayectorias se ha implantado de manera a priori en un simulador (MobileSim de Mobile Robots). Subsecuentemente, los resultados de la simulación han sido probados en un robot móvil real (Pioneer 2DX de ActiveMedia Robotics). Las pruebas preliminares muestran que el planeador de rutas da una solución para todos los casos donde se involucran obstáculos poligonales convexos. El tiempo corto de cálculo y el comportamiento estable del sistema hacen que este método sea confiable para la solución del problema de planeación de movimiento, además de permitir su implementación en aplicaciones reales donde el uso de un robot comercial puede ser aplicado, por ejemplo: como guías de turistas en museos y bibliotecas, sistemas rescatadores en ambientes peligrosos como minas y lugares poco accesibles, además del apoyo para personas con capacidades diferentes, entre otros.

## 6.2. Trabajos futuros

- Establecer la navegación incremental consecutiva entre varios puntos meta.

- Realizar la programación necesaria para que el planeador de rutas local con base en el FVP lleve a cabo la evasión de obstáculos de tipo poligonal no convexo.

- Realizar un mapeo en tiempo real del ambiente en el que se realiza la navegación.

- Aplicar un sistema de aprendizaje reforzado al planeador de rutas local a manera de mejorar la solución de rutas.

# Referencias Bibliográficas

[Avnaim 1988] Avnaim, F. Boissonat J.D., Faverjon, B. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacle. Proceedings of the 1988 IEEE International Conference on Robotics and Automation, pp. 1656-1660 (1988).

[Baase 1998] Baase, S., Computer Algorithms, Introduction to Design and Analysis, Editorial Addison-Wesley Publishing Company, (1998).

[Barraquand y Latombe 1989] Barraquand, J. and Latombe, J. C. On Nonholonomic Mobile Robots and Optimal Maneuvering. Revue d'Intelligence Articielle, 3(2): 77-103 (1989).

[Barraquand y Latombe 1991] Barraquand, J. and Latombe, J. C. Robot Motion Planning: A Distributed Representation Approach. Int. J. of Rob. Res., 10(6):628-649 (1991).

[Barraquand y Latombe 1993] Barraquand, J. and Latombe, J. C. Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles. Algorithmica,10(2-3-4):121-155 (1993).

[Brooks y Lozano-Pérez 1983] Brooks, R. and Lozano-Pérez, T. A Subdvision Algorithm in Conguration Space for Findpath with Rotation. Proc. 8th Int. Joint Conf. on Articial Intelligence (ICAI), 799-806 (1983).

[Canny y Reif 1987] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In IEEE Conference on Foundations of Computer Science, pages 39-48, (1987).

[Choset et al] Choset H., Lynch K.M., Hutchinson S., Kantor G., Burgard W., Kavraki L., Thrun S. Principles of Robot Motion: Theory, Algorithms and Implementations. A brandford book, The MIT Press, England. 2005.

[Clark y El-Osery 2004] Clark, R. El-Osery, K. Wedeward, and Bruder, S. A Navigation and Obstacle Avoidance Algorithm for Mobile Robots Operating in Unknown, Maze-Type Environments, Intelligent Systems and Robotics Group (ISRG), New Mexico Tech, Socorro, N.M., (2004).

[Collins 1975] Collins, G. E. Quantier elimination for the elementary theory of real closed cells by cylindrical algebraic decomposition. In H. Brakhage, editor, Automata Theory and Formal Languages, volume 33 of Lecture Notes in Computer Science, pages 134-183 Springer (1975).

[Faverjon y Tournassoud 1987] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipualtors with high number of degrees of freedom," IEEE Procs. Int. Conf. Robot. Autom., pp. 1152-1159, 1987.

[Halperin y Sharir 1996] Halperin, D. and Sharir, M. A. Near-Quadratic Algorithm for Planning the Motion of a Polygon in a Polygonal Environment. Discrete Comput. Geom., 16:121-134 (1996).

[Hopcroft 1986] Hopcroft, J. Wilfong, G. Motions of Objects in Contact. The International Journal of Robotics Research, Vol. 4, 4, pp. 32-46 (1986).

[Hun y Shin 2004] Hun, K. D. Shin, S. New Repulsive Potential Functions with Angle Distributions for Local Path Planning, Division of Electronic and Electric Engineering, Kyungnam University, Masan South Korea , School of Information Science and Technology, University of Tokyo, Tokyo Japan, (2004).

[Kamon, Rivlin y Rimon 1996] I. Kamon, E. Rivlin and E. Rimon. A new range-sensor based globally convergent navigation for mobile robots. In IEEE Int'l. Conf. on Robotics and Automation, Minneapolis, MN, April 1996.

[Kavraki 1996] Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. Probabilistic Roadmaps for Path Planning in High-Dimensional Conguration Spaces. IEEE Tr. On Robotics and Automation, 12(4): 566-580 (1996).

[Khatib 1986] Khatib, O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. Int. J. of Robotics Research, 5(1), 90-98 (1986).

[Latombe 1991] Latombe, J.C. Robot Motion Planning, Kluwer Academic Pub., Boston M. A. EEUU (1991).

[Latombe 1999] Latombe, Jean-Claude, Motion Planning: A Journey of Robots, Molecules, Digital Actors, and Other Artifacts, Stanford University, Stanford, California, EEUU (1999).

[LaValle y Hutchinson 1998] LaValle, S. M., Hutchinson, S.A., An objective-based framework for motion planning under under sensing and controls uncertainties. International Journal of Robotics Research, 17(1): 19-42, (1998).

[LaValle 2000] LaValle, S. M., Robot Motion Planning: A Game-Theoretic Foundation, Department of Computer Science, Iowa State Universiity, Ames, Iowa, EEUU (1996).

[LaValle 2006] LaValle, S. M., Planning Algorithms, Chapters 3,4,5, University of Illinois, Ed. Cambridge University Press, EEUU, (2006).

[Leven y Sharir 1985] Leven, D. Sharir, M. An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers, In Proceedings of the 1st Annual ACM Symposium on Computational Geometry, pp-221-227 (1985).

[Lozano-Pérez 1976] Lozano-Pérez, T. The Design of a Mechanical Assembly System. Tech. Rep. AI-TR 397, AI Lab, MIT, Cambridge, M.A. EEUU (1976).

[Lozano-Pérez y Wesley 1979] Lozano-Pérez, T. and Wesley M. A. An Algorithm for Planning Collision Free Paths Among Polyhedral Obstacles. Comm. ACM, 22(10):560-570 (1979).

[Lozano-Pérez 1983] Lozano-Pérez, T. Statial Planning: A Configuration Space Approach IEEE Tr. Computers, C-32(2):108-120 (1983).

[Lumelsky y Stepanov 1987] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknowk obstacles of arbitrary shape. Algorithmica, 2:403-430, 1987.

[Nilsson 1969] Nilsson, N.J. A Mobile Automaton: An Application of Artificial Intelligence Techniques. Proc. 1st Int. Joint Conf. on Artificial Intelligence, Wasington D.C., 509-520 (1969).

[Parag and Nourbakhsh 2000] Parag, H. and Nourbakhsh, I. Path Planning for the Cye Personal Robot, Carnegie Mellon University, Robotics Institute, Pittsburg, PA, EEUU (2000), Online: http://www.cs.cmu.edu/~illah/PAPERS/cye.pdf

[Petti y Fraichard 2005] Petti, S. Fraichard, T. Reactive Planning under Uncertainty among Moving Obstacles, INRIA, Cedex, France, (2005).

[Preparata and Shamos 1985] F. Preparata and M.I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, 1985. p198-257.

[Pruski 1996] A. Pruski, Robotique mobile - La planification de trajectoire, Ed. Hermes, 1996.

[Quinlan 1994] Quinlan, S. Efficient Distance Computation Between Non-Convex Objects. Proc. IEEE Int. Conf. on Robotics and Automation, 3324-3329 (1994).

[Ramírez 2000] Ramírez, G. Docteur Thèse: Contribution à la planification de trajectoires sans collition de robots mobiles non holonomes – Approche basée sur le calcul de distance dans l'espace des vitesses, Université de Poitiers, France. (2000)

[Ramírez y Zeghloul 2000] G. Ramírez and S. Zeghloul, "A new local path planner for nonholonomic mobile robot navigation in cluttered environments," IEEE Procs. Int. Conf. Robot. Autom., pp. 2058-2063, 2000.

[Reif 1979] Reif, J. H. Complexity of the Movers Problem and Generalizations. Proc. FOCS, 421-427 (1979).

[Rief y Wang 1998] Rief, J., Wang, H. The complexity of the two dimensional curvature-constrained shortest-path problem. In Proc. 1998 Workshop Algorithmic Found. Robot., page to appear, (1998).

[Schwartz y Sharir 1983] Schwartz, J. T. and Sharir, M. On the "Piano Movers" Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds. Advances in Applied Mathematics, 4:298-351 (1983).

[Skewis and Lumelsky 1992] T. Skewis and V. Lumelsky, "Experiments with a mobile robot operating in a cluttered unknown environment," Procs. Int. Conf. Rob. Autom., pp. 1482-1487, 1992.

[Taylor 1976] Taylor, R. H. Synthesis of Manipulator Control Programs from Task-Level Specications. Ph. D. Dissertation, Dept. of Computer Science, Stanford U., Stanford, CA. EEUU, (1976).

[Udupa 1977] Udupa, S. Collision Detection and Avoidance in Computer Controlled Manipulators. Ph. D. Dissertation, Dept. of Electrical Engineering, California Institute of Technology, Pasadena CA. EEUU, (1977).

[Zeghloul y Rambeaud 1996] S. Zeghloul and P. Rambeaud, "A fast algorithm for distance calculation between convex objects using the optimization approach," Robotica, pp. 355-363, 1996.

# Anexo A

## Glosario de términos

**Acción**: Movimiento o transformación de un robot a partir de un estado.

**Ambiente**: Circunstancias, objetos y condiciones que rodean al robot.

**API**: Application Program Interface – Interfaz de Programas para Aplicaciones, parte del sistema operativo que provee una interfaz de uso común a las aplicaciones.

**Dead reckoning**: Método que determina la posición de un robot por medio de la realización de un grafo de su curso y velocidad a partir de una posición previamente conocida.

**Espacio**: Parte que ocupa cada objeto dentro del ambiente.

**Estado**: Representa la posición y orientación de un robot en un ambiente.

**Espacio de estados**: Es el conjunto de todos los estados posibles.

**Espacio de acción**: Es el conjunto de todas las acciones que pueden ser aplicadas a un estado.

**Espacio de configuración**: Es un conjunto en el cual cada elemento representa una transformación que especifica completamente la posición y orientación de todas las partes del robot.

**Espacio libre**: Es el espacio libre de colisiones con objetos.

**Espacio métrico**: Función que mide la distancia entre dos puntos del espacio de configuración.

**Espacio topológico**: Es una colección de subconjuntos del espacio de estados llamados conjuntos abiertos.

**Especular**: Dicho de dos cosas simétricas: que guardan la misma relación que la que tiene un objeto con su imagen en un espejo.

**Grados de Libertad** (DOF): Se refiere al máximo número de parámetros independientes que son necesarios para caracterizar completamente la transformación aplicada al robot.

**Incertidumbre**: Algo que es incierto.

**Manifold**: Es un espacio topológico que representa las propiedades de la estructura de un espacio de transformaciones.

**Mapa**: Es la representación geométrica del espacio.

**Obstáculo**: Espacio ocupado en el ambiente por objetos.

**Odometría**: medición de distancia entre dos puntos.

**Planeación**: Es la tarea de seguir una secuencia de acciones que lograran una meta.

**Planeación de movimiento**: Es el proceso de detallar una tarea en movimientos robóticos. Es utilizado para problemas que envuelven tiempo, restricciones dinámicas, coordinación de objetos, interacción con los sensores, etc.

**Planeación de trayectorias**: Es el problema geométrico de calcular una trayectoria libre de colisiones para un robot entre obstáculos estáticos.

**Sensor**: Cualquier pieza de equipo que recupera datos o propiedades del ambiente para el robot.

**Robot holonómico**: Se refiere a la relación que existe entre los grados de libertad controlables y los totales de un robot. Si los grados de libertad controlables es igual a los grados de libertad totales, el robot es holonómico.

**Transformación Grassfire**: Es un método que permite identificar los bordes de una región mediante la generación de su esqueleto.

# Anexo B

## Características del robot Pioneer 2DX

| **Características físicas** | **Pioneer 2DX/CE** |
|---|---|
| Longitud | 44cm |
| Anchura | 33cm |
| Altura (cuerpo) | 22 cm |
| Cuerpo | 5.1 cm |
| Peso | 9 Kg |
| Peso de carga | 23 Kg |
| **Construcción** | |
| Cuerpo | 1.6mm CNC fabricado de aluminio |
| Cubierta principal y consola | 2.4mm CNC fabricado de aluminio |
| Ensamble | Tornillos Allen hex (métrico) |
| **Poder** | |
| Batería | 3x12V acido de plomo (Sólo DX) |
| Carga | 252 watt-hr (DX) |
| Tiempo de ejecución | 8-10 hrs (3 hrs CE) |
| Tiempo de recarga del cargador std | 12 hrs (8 hrs CE) |
| Tiempo de recarga del cargador hs | 2.4 hrs (1hrs CE) |
| **Movilidad** | |
| Ruedas | 2 de caucho sólido, 1 rueda tonta |
| Diámetro de rueda | 16.5 cm |
| Anchura de rueda | 3.7 cm |
| Dirección | Diferencial |
| Radio de la rueda | 19.7:1 |
| Radio de oscilación | 32 cm |
| Radio de giro | 0 cm |

| | |
|---|---|
| Velocidad máxima de traslación | 1.6 m/sec |
| Velocidad máxima de rotación | 300 grados/seg |
| Paso transversal máximo | 2 cm |
| Apertura transversal máxima | 8.9 cm |
| Inclinación transversal máxima | Pendiente 25% |
| Terreno transversal | Todos los accesibles para una silla de ruedas |

**Sensores**

| | |
|---|---|
| Ultrasónicos frontales | 8<br>1 en cada lado<br>6 delanteros @ intervalos de 20° |
| Ultrasónicos traseros | 8<br>1 en cada lado<br>6 traseros @ intervalos de 20° |
| Posición del codificador | 76 señales por mm<br>9850 señales por revolución |

| | |
|---|---|
| **Electrónicos (microcontrolador básico a bordo)** | **Pioneer 2DX/CE** |
| Procesador | Siemens 8C166(20 MHz) |
| Entradas de posición | 4 |
| Entradas de sonar | 2x8 (multiplexado) |
| I/O digital | 16 puertos lógicos |
| A/D | 5 @ 0.5 VDC |
| Salida en tiempo digital | 8@resolución 1 µsec |
| Entradas en tiempo digital | 8@resolución 1 µsec |
| Puerto Comm | 2 RS-232 serial |
| FLASH PROM | 32 Kb; P2OS- Software codificado |
| RAM | 32 Kb; programable por el usuario |
| Switches de encendido | 1 principal y 2 auxiliares |

**Controles y puertos**

| | |
|---|---|
| Encendido principal | Encendido/apagado del Robot |
| Carga | Sistema de encendido/batería recargable |
| Display LCD | Sistema de estado y mensajes |
| Botón RESET | Reseteado rápido/descarga |
| Botón MOTORS | Motores/ joydrive/descarga/auto-prueba |

# Anexo C

## Programación del Planeador de rutas local

**La ley de control.** La función que calcula las velocidades lineal y angular correspondientes a la ley de control expuesta en (29) se indica en la figura C1.

```
void Ley_de_Control(){
  double X,Y,Theta,Xf,Yf,Xe,Ye,Thetae,k1,k2;
  // Calculo de Velocidades V y W
  X=0;
  Y=0;
  Theta = 0;
  Xf = meta.getX();
  Yf = meta.getY();
  k1 = 0.6;
  k2 = 0.6;
  printf("\tX\tY\tTheta\n");
  printf("\t%f\t%f\t%f\n",X,Y,Theta);
  int i=0;
  double Inc_t=0.9;
  double a=1;
  //while (abs(a)>.1){} Para cuando se necesita ir registrado odometria
      Xe = Xf-X;
      Ye = Yf-Y;
      Thetae = -Theta;
      a = sqrt(pow(Xe,2)+pow(Ye,2));
      double alpha = atan2(Ye,Xe) + Thetae;
      V = k1 * a * cos(alpha);
      if (V > V_Max)
            V = V_Max;
      if (V < -V_Max)
            V = -V_Max;

      W = k1 * sin(alpha) * cos(alpha) + k2 * alpha;
      if (W > W_Max)
            W = W_Max;
      if (W < -W_Max)
            W = -W_Max;
}
```

Figura C1. Ley de control del planeador de rutas local

**El planeador de rutas local.** En las figuras C2, y C3 se muestra la definición de clases principales para los módulos pertenecientes al planeador de rutas local. El código completo se incluye en el CD anexo a esta investigación.

```
#ifndef ARACTIONGOTO_H
#define ARACTIONGOTO_H

 class ArActionIrA : public ArAction
 {

 public:
    ///Constructor
    ArActionIrA(const char *name = "goto",
             ArPose goal = ArPose(0.0, 0.0, 0.0),
             double closeDist = 100, double speed = V_Max,
             double speedToTurnAt = 150, double turnAmount = 7);
    ///Destructor
    virtual ~ArActionIrA();

    bool haveAchievedGoal(void);
    void cancelGoal(void);
    void setGoal(ArPose goal);
    ArPose getGoal(void) { return myGoal; }
    void setCloseDist(double closeDist) { myCloseDist = closeDist; }
    double getCloseDist(void) { return myCloseDist; }
    void setSpeed(double speed) { mySpeed = speed; }
    double getSpeed(void) { return mySpeed; }
    virtual ArActionDesired *fire(ArActionDesired currentDesired);
    virtual ArActionDesired *getDesired(void) { return &myDesired; }

 #ifndef SWIG
    virtual const ArActionDesired *getDesired(void) const
    { return &myDesired; }
 #endif

protected:

   ArPose myGoal;
   double myCloseDist;
   double mySpeed;
   double mySpeedToTurnAt;
   double myDirectionToTurn;
   double myCurTurnDir;
   double myTurnAmount;
   ArActionDesired myDesired;
   bool myTurnedBack;
   bool myPrinting;
   ArPose myOldGoal;

   enum State
   {
     STATE_NO_GOAL,
     STATE_ACHIEVED_GOAL,
```

```
      STATE_GOING_TO_GOAL
   };
   State myState;
};
#endif // ARACTIONGOTO
```

Figura C2. Clase *ArActionIrA* utilizada para el modulo Alcanzando_la_meta

```
class ArActionEvadirDeFrente : public ArAction
{

public:

  /// Constructor
  ArActionEvadirDeFrente(const char *name = "avoid front obstacles",
                double obstacleDistance = 200, double avoidVelocity = 200,
                double turnAmount = 15, bool useTableIRIfAvail = true);
  /// Destructor
   virtual ~ArActionEvadirDeFrente();
   virtual ArActionDesired *fire(ArActionDesired currentDesired);
   virtual ArActionDesired *getDesired(void) { return &myDesired; }

protected:

  double myTurnAmount;
  double myObsDist;
  double myAvoidVel;
  double myTurnAmountParam;
  bool myUseTableIRIfAvail;
  int myTurning; // 1 for left, 0 not turning, -1 for right
  ArActionDesired myDesired;
  ArSectors myQuadrants;
  ArFunctorC<ArActionAvoidFront> myConnectCB;

};
```

Figura C3. Clase *ArActionEvadirDeFrente* utilizada para el modulo Seguimiento_de_frontera