Planeamiento de Trayectorias en Contextos Dinámicos

Román Katz y Claudio Delrieux

Departamento de Ingeniería Eléctrica y de Computadoras
Universidad Nacional del Sur - Av. Alem 1253, (8000) Bahía Blanca, ARGENTINA.
e-mail: claudio@acm.org

Resumen

En este trabajo presentamos un sistema de planeamiento de trayectorias basada en el algoritmo de ruteo de Lee. Este algoritmo (concebido originariamente para obtener conexiones mínimas en circuitos VLSI) proporciona un mecanismo simple y robusto para computar caminos óptimos en espacios de configuración bidimensionales. El motor del sistema de planeamiento presentado en este trabajo se implementó mediante una versión basada en reglas del algoritmo de Lee, integrando una arquitectura híbrida que realiza inferencias lógicas mediante una componente del lenguaje Prolog, y cuyos resultados son compartidos por componentes de procesamiento númerico en un lenguaje imperativo convencional. Esto posibilita una clara factorización de las funcionalidades del sistema: el algoritmo de navegación se basa en la representación de alto nivel a través de su formulación lógica, y el estado del espacio de configuración se puede obtiener mediante técnicas de visión y procesamiento implementadas numéricamente. Por lo tanto la arquitectura propuesta sintetiza simultáneamente la velocidad y versatilidad en su entorno visual de aplicación, y el nivel de abstracción y modularidad de la descripción lógica de su motor de planeamiento.

Palabras Clave: Inteligencia artificial, planeamiento de trayectorias, agentes inteligentes, robótica móvil.

1 Introducción

La navegación robótica es usualmente estudiada desde dos perspectivas diferentes, ya sea desde un punto de vista de modelado cinemático y dinámico del robot o bien como una problemática geométrica-espacial [5]. En la primera, se genera una ley de control que proporciona las fuerzas y torques para controlar los manipuladores del robot, siguiendo una determinada trayectoria de referencia. Aquí suelen aplicarse satisfactoriamente los *campos potenciales* (artificial potential fields). En la segunda

perspectiva se genera una trayectoria libre en el espacio de configuración asociado a las configuraciones del robot, a los obstáculos del área de trabajo y a las posiciones inicial y final deseada para el mismo [5]

Indudablemente ambas perspectivas tienen ventajas y desventajas asociadas, y campos de aplicación bien definidos y ciertamente independientes entre sí. El modelado cinemático y dinámico —como el utilizado en los campos potenciales— se usa mayoritariamente en aplicaciones de navegación de tiempo real [5], donde las técnicas de programación utilizadas son en general de procesamiento numérico. Por el contrario, el modelado geométrico y espacial es considerado como tópico de planeamiento [11, 9], con áreas de aplicación principalmente en tareas de navegación off-line de alto nivel [15], y las técnicas utilizadas son cercanas al procesamiento simbólico [3, 16], en donde el conocimiento es representado de forma declarativa, por ejemplo en lógica de primer orden. Por lo tanto, parece conveniente y útil combinar ambas ventajas al considerar la implementación de un sistema de guiado y navegación híbrido, que integre un motor de planeamiento de trayectorias basado en una versión en lenguaje Prolog del algoritmo de ruteo de Lee [17] en un entorno de programación numérica.

Por dicha razón, en este trabajo presentamos un sistema de planeamiento de trayectorias en contextos dinámicos. El motor del sistema de se implementó mediante una versión basada en reglas del algoritmo de Lee, integrando una arquitectura híbrida que realiza inferencias lógicas, y cuyos resultados son compartidos por componentes de procesamiento númerico. La arquitectura propuesta sintetiza simultáneamente la velocidad y versatilidad en su entorno visual de aplicación, y el nivel de abstracción y modularidad de la descripción lógica de su motor de planeamiento. Este trabajo esta organizado de la siguiente manera: la Sección 2 presenta los conceptos del algoritmo de Lee y de cómo utilizarlo con propósitos de planeamiento de trayectorias. La Sección 3 considera la arquitectura original para el planeamiento de trayectorias en ambientes con obstáculos en movimiento, extendiendo luego la propuesta para soportar posiciones finales (targets) dinámicas. Las secciones 4 y 5 presentan las simulaciones y resultados experimentales y las conclusiones, respectivamente.

2 El Algoritmo de Ruteo de Lee

En esta Sección se presentan algunos detalles del algoritmo de ruteo de Lee y de cómo utilizarlo para propósitos de planeamiento de trayectorias. Este algoritmo fue concebido originalmente para obtener conexiones mínimas entre terminales de circuitos VLSI, y en el sistema propuesto provee un mecanismo simple y robusto para obtener caminos mínimos en espacios reales. Consideremos en primer lugar el espacio de configuración bidimensional representado por la grilla regular mostrada en la Fig. 1. El espacio de trabajo puede contener obstáculos (rectángulos azules) y luego el problema a resolver consiste en encontrar un camino libre mínimo entre

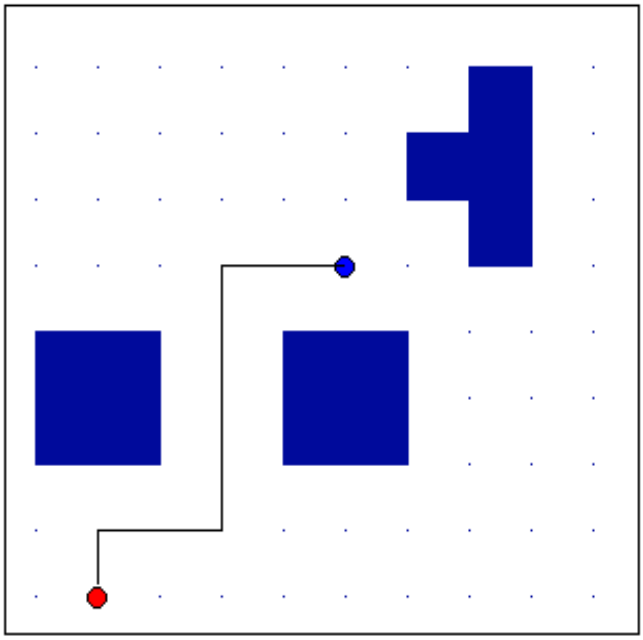


Figura 1: Configuración genérica del espacio de trabajo.

dos posiciones dadas. La línea negra uniendo estos dos puntos (círculos pequeños rojo y azul) representa el camino mínimo entre ellos.

Una versión en Prolog del algoritmo de Lee [17] adaptado para nuestro problema de planeamiento de trayectorias se muestra en la Fig. 2. En nuestro algoritmo, los puntos del espacio de configuración están representados por sus coordenadas cartesianas (indicadas X-Y) y las trayectorias son calculadas por el programa como una lista de puntos. Inicialmente el programa muestra la relación `lee_route` y sus dos reglas derivadas `waves` y `path`. El predicado `waves` genera sucesivos puntos vecinos, comenzando por el punto inicial, hasta que el punto final deseado es alcanzado. Estos `waves` son conjuntos de puntos que tienen como vecino a un punto del wave anterior y tal que aún no han aparecido. El predicado `path` obtiene el *path* utilizando los *waves* encontrados de forma reversa, es decir que el orden de los puntos de la trayectoria resultante progresan desde el punto final hasta el punto inicial.

En esta implementación se consideran obstáculos de forma rectangular y también de forma compleja. Los primeros están representados en términos de la forma `obstacle(L,R)`, donde L es el par de coordenadas del extremo inferior izquierdo y R el del extremo superior derecho. Los obstáculos complejos se manejan mediante uns lista de los bloques rectangulares que los componen, utilizando la estructura genérica recién descripta (por ejemplo `[obstacle((2-3),(4-5)), obstacle((3-3),(4-5))`). En ambos casos los obstáculos pueden ser fácilmente incorporados a la base de conocimiento del sistema a través de los predicados de Prolog `assert` o `retract` [4].

Como pretendemos un sistema capaz de capturar el espacio de configuración de forma dinámica, hemos introducido algunos cambios sobre el el algoritmo de Lee

```
lee_route(A,B,Obstacles,Path):-waves(B,[[A],[]],Obstacles,Waves),
                               path(A,B,Waves,Path).

waves(B,[Wave|Waves],Obstacles,Waves):-member(B,Wave),!.
waves(B,[Wave|[Lastwave|Lastwaves]],Obstacles,Waves):-
          next_wave(Wave,Lastwave,Obstacles,Nextwave),
          waves(B,[Nextwave,Wave,Lastwave|Lastwaves],Obstacles,Waves).

next_wave(Wave,Lastwave,Obstacles,Nextwave):-
          setof(X,admissible(X,Wave,Lastwave,Obstacles),Nextwave).

admissible(X,Wave,Lastwave,Obstacles):-adjacent(X,Wave,Obstacles),
                                       not member(X,Lastwave),
                                       not member(X,Wave).

adjacent(X,Wave,Obstacles):-member(X1,Wave),
                            neighbor(X1,X),
                            not obstructed(X,Obstacles).

neighbor((X1-Y),(X2-Y)):-next_to(X1, X2).
neighbor((X-Y1),(X-Y2)):-next_to(Y1, Y2).

next_to(X,X1):- X1 is (X+1).
next_to(X,X1):- X>0,X1 is (X-1).

obstructed(Point,Obstacles):-member(Obstacle,Obstacles),
                             obstructs(Point,Obstacle).

obstructs(X-Y,obstacle((X-Y1),(X2-Y2))):-Y1≤Y, Y≤Y2.
obstructs(X-Y,obstacle((X1-Y1),(X-Y2))):-Y1≤Y, Y≤Y2.
obstructs(X-Y,obstacle((X1-Y),(X2-Y2))):-X1≤X, X≤X2.
obstructs(X-Y,obstacle((X1-Y1),(X2-Y))):-X1≤X, X≤X2.

path(A,A,Waves,[A]):-!.
path(A,B,[Wave|Waves],[B|Path]):-member(B1,Wave),
                                 neighbor(B,B1),
                                 !, path(A,B1,Waves,Path).

get_obstacles(Previous,List):-obstacle(A,B),
                              not member(obstacle(A,B),Previous),
                              conc([obstacle(A,B)],Previous,Next),
                              get_obstacles(Next,List),!.
get_obstacles(List,List).

test_lee(Source,Destination, Path):-
          lee_route(Source, Destination, Obstacles, Path).
```

Figura 2: Versión en Prolog del algoritmo de Lee.

original presentado en [17]. Por ejemplo, definimos el predicado `get_obstacles` que simplifica el proceso de reconocimiento de obstáculos en cualquier momento de la ejecución. Esto es de especial interés para el esquema de guiado con obstáculos en movimiento que será descripto en detalle en la Sección 3. En particular, el predicado `get_obstacles` debe ser computado antes de `test_lee`, de forma de actualizar *obstacles* para la correcta ejecución de `lee_route` y también de `test_lee`.

Procedemos finalmente a formular el problema de navegación, de forma de utilizar el algoritmo propuesto para guiar un agente móvil desde una posición inicial a una posición final del espacio de configuración. Afortunadamente la conversión de la estrategia de resolución de trayectorias provista por el algoritmo de Lee en una arquitectura de planeamiento es casi inmediata. Por ejemplo, la posición actual del móvil robótico y la posición final deseada son respectivamente los anteriores puntos iniciales (círculo rojo) y finales (círculo azul) y el camino libre uniendo dicho puntos entre los obstáculos representa la trayectoria mínima buscada.

Se destaca que los métodos de adquisición del entorno de trabajo se independizan del motor de guiado provisto por el algoritmo en Prolog, puesto que cualquiera que sea el mecanismo de interfase, la base de conocimiento siempre puede ser actualizada a través de predicados `assert` o `assertz`. De hecho las formas de obtener el "layout" del espacio de configuración conteniendo información posicional de los obstáculos puede ser variada, siendo las provistas por técnicas de Visión Robótica y Procesamiento de Imágenes [10, 18] las más versátiles y usadas en la actualidad.

3 Navegación en Ambientes Dinámicos

En la Sección anterior presentamos el algoritmo de Lee con las adaptaciones adecuadas para su uso como generador de trayectorias en ambientes conteniendo obstáculos fijos. Sin embargo la situación común en ambientes reales de trabajo es mucho más compleja. Pueden existir errores e incertidumbre en el control y en el sensado [13] y los obstáculos y la posición final deseada pueden moverse durante el desarrollo de las tareas [14]. En esta Sección discutiremos específicamente cómo adaptar el algoritmo propuesto para contemplar situaciones en las cuales los obstáculos y/o la posición final se modifican de forma dinámica.

En primer lugar consideramos la posibilidad de modificación en los obstáculos, restringiéndonos a aquellos que efectúan movimientos de velocidad acotada. Aunque esta es una restricción que podría ser importante, la mayoría de las situaciones de interés la verifican. Por lo tanto, solamente son tolerables cambios pequeños en la posición de los obstáculos de un momento a otro, ya sea por alteración en sus dimensiones o por variaciones posicionales. La relativa baja velocidad de cambio del ambiente posibilita la implementación de nuestro sistema de guiado híbrido. En el mismo se integra el motor de planeamiento con información sensorial —la cual actualiza dinámicamente la situación del espacio de configuración de la base de conocimientos— permitiendo un ajuste en tiempo de corrida de la mejor trayectoria a realizar.

```
set(Initial_point,Final_point)
get_obstacles(Obstacles)
test_lee(Path)
Actual_point←Initial_point
(Start moving)
repeat        (Show path)
      move_nextpoint
      Actual_point←Next_point
      get_obstacles(Obstacles)
      If interfer(Obstacles,Path) then test_lee(Path)
until (Actual_point=Final_point)
```

Figura 3: Pseudo-código del procedimiento para navegación con obstáculos móviles.

Bajo las condiciones mencionadas, proponemos un esquema que complementa componentes de alto nivel con interfases sensoriales, al tiempo que las herramientas de procesamiento numérico brindan facilidades de procesamiento de señales y visualización del proceso en desarrollo. El resultado es un arquitectura de navegación para ambientes dinámicos altamente simple y robusta, que combina el uso de alguno de los predicados de la Fig. 2 en un ciclo de tipo *repeat ... until* en el cual *obstacles* y/o la posición destino son actualizadas iterativamente y *path* es recalculado en caso que sea necesario. El pseudo-código del procedimiento para navegación con obstáculos móviles es mostrado en la Fig. 3.

En primer lugar se definen las condiciones extremas de la trayectoria requerida, definiendo los puntos iniciales y finales (*Initial_point* y *Final_point*) con el comando `set`. Luego, se obtiene el layout inicial en *Obstacles* mediante (`get_obstacles(Obstacles)`) y la primer trayectoria *Path* es obtenida con (`test_lee(Path)`). Una vez concluida la secuencia inicial, se define *Actual_point* como *Initial_point* y el robot móvil puede comenzar el movimiento (Start moving). El proceso ingresa ahora al ciclo *repeat ... until*, efectuando iterativamente varios pasos hasta alcanzar la condición de parada.

En cada iteración se realizan varias tareas. Primero, se muestra la posición actual del móvil (Show path). Luego, se conduce al móvil a la posición siguiente del espacio de configuración 2-D (`move_nextpoint`), actualizando posteriormente la posición actual *Actual_point* con la nueva posición *Next_point*. Finalmente el algoritmo realiza la tarea más importante del ciclo (y también de todo el procedimiento), que es la evaluación del impacto de cambios en el espacio de trabajo. Esto es realizado en dos pasos consecutivos: primero se recarga el layout (`get_obstacles(Obstacles)`), y luego se prueba mediante `interfer(Obstacles,Path)` la posible interferencia entre este nuevo conjunto *Obstacles* y la trayectoria ya calculada *Path*. Si esta evaluación muestra la existencia de interferencia, una nueva trayectoria se computa desde la posición actual del móvil considerando el arreglo actualizado de obtáculos mediante

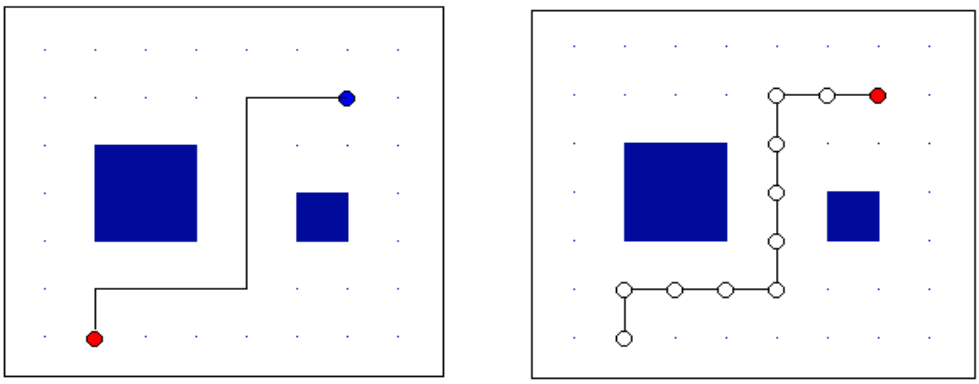
```
set(Initial_point)
set(Final_point)
get_obstacles(Obstacles)
test_lee(Path)
Actual_point←Initial_point
(Start moving)
repeat        (Show path)
      move_nextpoint
      Actual_point←Next_point
      get_obstacles(Obstacles)
      get_target(New_target)
      If interfer(Obstacles,Path) or New_target ≠ Final_point then
           begin
           Final_point ← New_target
           test_lee(Path)
           end
until (Actual_point=Final_point)
```

Figura 4: Pseudo-código del procedimiento para navegación con posición final dinámica y obstáculos móviles.

(`test_lee(Path)`). El ciclo itera hasta que el móvil alcanza la posición final, es decir cuando *Actual_point* es igual a *Final_point*. Mientras el procedimiento está en ejecución, el sistema obtiene continuamente información de los obstáculos y actualiza convenientemente la base de conocimiento. Esto se realiza mediante los predicados de Prolog `assert` y `retract` (como se describió en la Sección 2) y dispositivos sensores.

Cuando la posición final cambia durante la ejecución del proceso es probable que la trayectoria previa calculada deje de ser la óptima. Por esta razón, el algoritmo debe considerar una nueva trayectoria cada vez que la posición final deseada varía. Esto tiene un impacto negativo en la eficiencia del procedimiento, aunque varias optimizaciones pueden ser implementadas en tal sentido (se discute esto luego en las conclusiones). El pseudo-código del procedimiento para planeamiento de trayectorias con posición final dinámica y obstáculos móviles es mostrado en la Fig. 4.

Cómo dijimos antes, una interfaz de sensado adecuada la proporcionan las técnicas de Visión Robótica y Procesamiento de Imágenes (*CVIP*). Esta es en efecto una estrategia más que conveniente, puesto que muchos de los robots móviles experimentales y comerciales disponibles utilizados en navegación poseen incorporadas cámaras, placas de adquisición y dispositivos DSP para procesamiento [12, 10]. Por lo tanto, una arquitectura basada en componentes como la propuesta parece una solución apropiada para integrar los dispositivos de hardware para adquisición con la representación lógica del motor de planeamiento. Es de destacar que aquellas ar-



(a) (b)

Figura 5: Simulación del sistema para navegación con obstáculos fijos.

quitecturas híbridas de *CVIP* que combinen procesamiento numérico y lógico como las propuestas en [2, 6, 7] se integrarán más naturalmente a nuestro sistema.

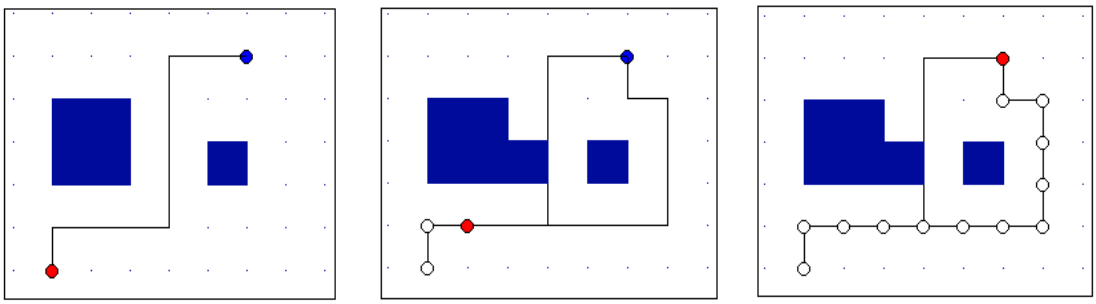
4 Simulaciones y Resultados Experimentales

La integración de unidades de software de diferentes lenguajes de programación puede ser compleja. Es por esto que utilizamos un sistema de programación basado en componentes como Kylix$^{(TM)}$ (o Delphi$^{(TM)}$ bajo MS Windows$^{(TM)}$). De esta forma la plataforma de desarrollo IDE permite incorporar junto con las utilidades numéricas propias, un interprete de Prolog a través de una componente de software. En esta etapa de nuestro desarrollo estamos utilizando software libre y componentes freeware (la versión Open Edition de Kylix$^{(TM)}$ es de distribución libre para fines académicos y el interprete de Prolog se obtuvo como freeware en `www.trinc.nl`). En esta Sección presentaremos algunas simulaciones junto con los resultados obtenidos.

En la Fig. 5 mostramos imágenes de una simulación que considera un conjunto de obstáculos estáticos. La Fig. 5(a) ilustra el layout estático, con la posición actual del móvil (círculo rojo), el target o posición final deseada (círculo azul) y la trayectoria inicial calculada uniendo ambos puntos. En la Fig. 5(b) se observa al móvil alcanzando la posición deseada (círculo rojo), luego de haber recorrido los puntos de la trayectoria libre calculada (círculos blancos).

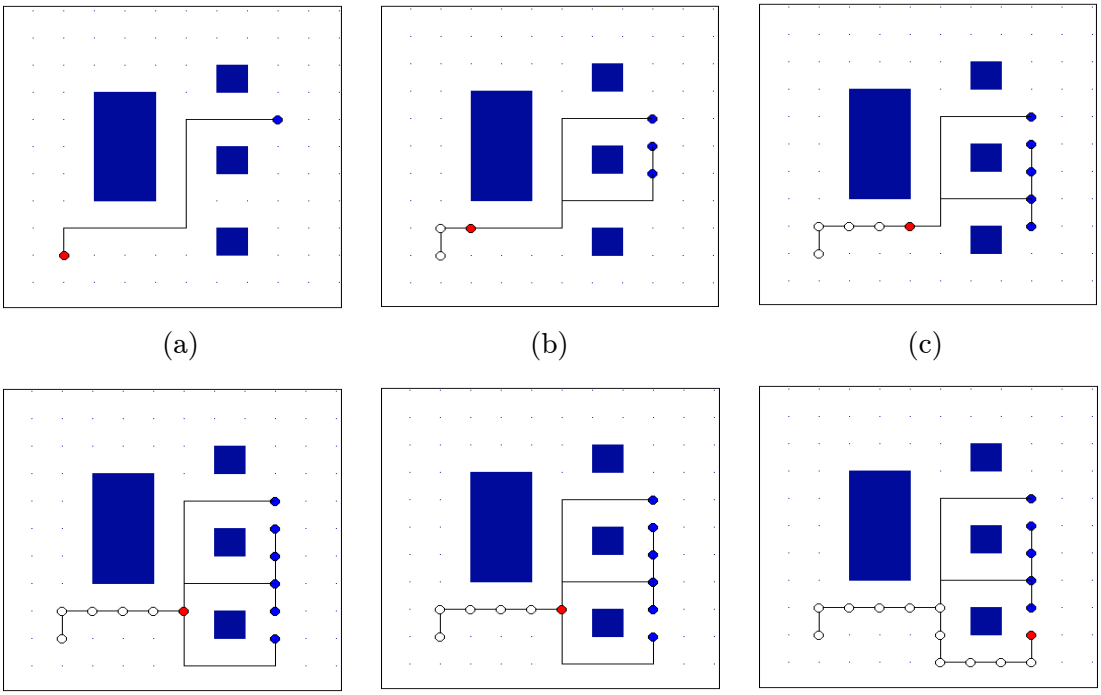
En la Fig. 6 mostramos el sistema resolviendo una situación con obstáculos móviles. En Fig. 6(a) se observa el layout inicial, la posición actual (círculo rojo), la posición final deseada (círculo azul) y la trayectoria inicial calculada. La Fig. 6(b) muestra el móvil en una situación intermedia, cuando un cambio en la estructura de uno de los obstáculos genera un interferencia con la trayectoria en desarrollo. Es por esto que se computa un nuevo camino libre, de forma que el móvil puede evitar el obstáculo y alcanzar la posición final deseada en Fig. 6(c).

La Fig. 7 ilustra el sistema funcionando cuando las posición final deseada (target) es dinámica, es decir evoluciona durante el desarrollo de la ejecución. En Fig. 7(a) se observa el layout inicial, la posición actual (círculo rojo), el target (círculo azul)



(a) (b) (c)

Figura 6: Simulación del sistema para navegación con obstáculos móviles.



(a) (b) (c)

(d) (e) (f)

Figura 7: Simulación del sistema para navegación con posición final dinámica.

y la trayectoria inicial calculada. Fig. 7(b) a Fig. 7(e) muestran la evolución en el desplazamiento del target y del móvil. Se observa que en la Fig. 7(b) y Fig. 7(d) se recalcula la trayectoria de forma notable (no sólo en el tramo final en donde cambió la posición del target), puesto que la que está en ejecución resulta no ser la óptima. En la Fig. 7(f) se observa al móvil alcanzar el target.

Además de las mostradas se efectuaron varias simulaciones del sistema, testeando el planeamiento bajo condiciones diversas de obstáculos fijos y móviles, targets dinámicos y problemáticas combinadas. En todos los casos los resultados fueron óptimos, no sólo en resultados sino también en tiempo de cómputo. La arquitectura planteada permitió resolver de forma robusta situaciones generales en donde los obstáculos se movían en forma lineal. Sin embargo, en algunas situaciones se originaron "deadlocks" cuando los objetos oscilaban alrededor de una posición determinada, y donde era esperable obtener una trayectoria definitiva. Una posible solución para esto se discute en las Conclusiones.

5 Conclusiones y Trabajo Futuro

Se presentó una arquitectura que permite el planeamiento de trayectorias para guiar agentes móviles en contextos dinámicos. El sistema puede ser utilizado para generar trayectorias que eviten obstáculos en movimiento, y también para alcanzar objetivos móviles. Se utiliza como base el algoritmo de ruteo de Lee, implementado con un sistema basado en reglas, el cual genera trayectorias óptimas para un estado dado del espacio de configuraciones. Estas trayectorias son actualizadas cada vez que se detecta un cambio en el contexto, cuya actualización es monitoreada por medio de un sistema que puede incorporar sensores y visión robótica.

Durante el desarrollo surgieron dos problemas interesantes para su posterior estudio. El primero concierne la optimización del cómputo realizado cada vez que el objetivo del agente móvil se desplaza. En efecto, si bien el algoritmo está en condiciones de reutilizar lo calculado cuando existe un cambio en la posición de los obstáculos, esto no sucede cuando es el *target* el que se desplaza, y por lo tanto la trayectoria completa debe ser recalculada. El segundo problema se origina cuando un obstáculo se desplaza en forma oscilatoria. En esta situación, puede ocurrir que el algoritmo de ruteo genere cíclicamente alternativas para rodear dicho obstáculo por caminos diferentes, ocasionando el surgimiento de un "deadlock" en el cual el agente móvil queda atrapado.

Para estas dos situaciones problemáticas, estamos considerando incorporar a la arquitectura una tercera componente de monitoreo. Esta componente tendrá un registro de la historia de las trayectorias generadas y los cambios en el contexto, con el objetivo de poder reutilizar cómputos útiles ya efectuados, y además evitar los casos más obvios de "deadlocks" (aunque su total eliminación probablemente sea un problema indecidible). Otro desarrollo de gran importancia consiste en considerar un espacio de configuración contínuo, de manera que las trayectorias poligonales

generadas por el algoritmo de Lee sean ahora consideradas como grafo de control de un mecanismo de generación de curvas [1, 8].

Referencias

[1] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*. Springer-Verlag, New York, 1987.

[2] Bruce G. Batchelor. *Intelligent Image Processing in Prolog*. Springer-Verlag, London, 1991.

[3] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Addison Wesley, Singapore, 1990.

[4] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag, third edition, 1988.

[5] Thomas Dean and Michael Wellman. *Planning and Control*. Morgan Kaufmann Publishers, Los Altos, CA, 1990.

[6] C. Delrieux and R. Katz. Hybrid image recognition architecture. In *SPIE's 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Florida USA, 2002. SPIE-The International Society for Optical Engineering.

[7] C. Delrieux and R. Katz. Integrating Symbolic and Numerical Image Processing. In *The 2002 International Conference on Imaging Science, Systems, and Technology (CISST'02)*, Las Vegas USA, 2002. Proceedings of the CISST'02.

[8] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, New York, 1988.

[9] Michael Genesereth and Nils Nilson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Los Altos, California, 1987.

[10] R. C. González K. S. Fu and C. S. G. Lee. *Robotics: Control, Detection, Vision and Intelligence*. McGraw-Hill, New York, 1988.

[11] R. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[12] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, Massachusetts, 1990.

[13] Jean-Claude Latombe, Anthony Lazanas, and Shashank Shekhar. Robot Motion Planning with Uncertainty in Control and Sensing. *Artificial Intelligence*, 52(1):1–47, 1991.

[14] Nils Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[15] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: a Logical Approach*. Oxford University Press, New York, 1998.

[16] Elaine Rich. *Artificial Intelligence*. McGraw-Hill, 1983.

[17] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. The MIT Press, Cambridge, Massachusetts, second edition, 1994.

[18] S. E. Umbaugh. *Computer Vision and Image Processing*. Prentice-Hall, New York, 1998.