Motion Planning in a Dynamic Domain*

Kikuo Fujimura
Center for Engineering Systems Advanced Research
Oak Ridge National Laboratory
P.O. Box 2008
Building 6025, MS-6364
Oak Ridge, TN 37831-6364

Hanan Samet
Center for Automation Research and
Computer Science Department and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

ABSTRACT

Motion planning is studied in a time-varying environment. Each obstacle is a convex polygon that moves in a fixed direction at a constant speed. The robot is a convex polygon that is subject to a speed bound. A method is presented to determine whether or not there is a translational collision-free motion for a polygonal robot from an initial position to a final position, and to plan such a motion, if it exists. Our method makes use of the concepts of configuration spaces and accessibility. An algorithm is given for motion planning in such an environment and its time complexity is analyzed.

Keywords and Phrases: time-varying environments, configuration spaces, accessibility, finite-sized robots, and motion planning.

TIME-VARYING ENVIRONMENTS

We consider the problem of motion planning in a 2-dimensional time-varying environment. The environment contains a number of moving polygonal obstacles whose motions are known. Each obstacle is a convex polygon that moves in a fixed direction at a constant speed. We treat edges constituting polygons as the basic units for our discussion, and we use the term *movement* to denote the straight motion of an edge. We assume that the environment is known completely (i.e., the shapes and trajectories of the obstacles are given a priori) and that the robot is subject only to a maximum speed that is greater than any of the obstacles' speeds. We would like to know whether there exists a colllision-free translational motion for a polygonal robot from an initial position to a final position, and if there is, we would like to find a motion that takes the robot to the final destination in the shortest time.

Many studies on motion planning concern the following two aspects of the problem:

(1) decide whether or not there exists a motion given an environment, and find a path if there is one, and
(2) optimize a motion in terms of some criterion (e.g., path length, number of turns, etc).

Note that time plays a crucial role in a time-varying environment. As an example, suppose that a robot is located inside a room that contains a number of moving and stationary obstacles. The room also has a door that is open for a certain period of time and the robot must pass through the door before it closes. The arrangement of the obstacles inside the room may be contrived in such a way that the robot can move to the door in time only if it makes a time-minimal motion to reach the door.

In this paper, we show how to find a translational time-minimal motion for a convex polygonal robot in an environment that contains moving obstacles. We assume that the robot can move faster than any of the obstacles. Section 2 reviews some prior research that serves as a basis for our approach. Section 3 describes motion planning among moving obstacles. Section 4 presents our algorithm as well as analyzes its execution time. Section 5 compares our work with some previous work, and Section 6 contains a few concluding remarks.

BACKGROUND

In this section, we review the concepts of a configuration space and accessibility. They are used in Sections 3 and 4.

Overview of the Configuration Space

One formulation for motion planning among stationary obstacles has been to use the configuration space [Upud76, Loza79]. This is a transformation from a physical space in which the robot is a polygon (or a polyhedron in 3-dimensions) into another space in which the robot is treated as a point. Intuitively, the configuration space is obtained by shrinking the robot to a point, while growing the obstacles by the size of the robot. Formally, the configuration space is described as follows. The position and orientation of a rigid object, say A, in the plane (or physical space) can be specified by a 3-tuple (x, y, θ) in a 3-dimensional space, called the *configuration space* of the object and denoted by $Cspace_A$. Here (x, y) represents the position of a *reference point* of A in physical space, and θ represents the angle made by a *reference angle* of A relative to the x-axis. When the orientation of the object is fixed, its configuration space is 2-dimensional because the pair (x, y) is sufficient to specify the location of A in physical space.

In $Cspace_A$, some points correspond to placements of A in which A overlaps other objects in physical space. Such points in $Cspace_A$ are called *illegal*, while points in $Cspace_A$ that correspond to placements of A where A does not overlap with any of the other objects in the physical space are called *legal*.

More specifically, the set of points in $Cspace_A$ that correspond to the placements of A where A overlaps with object B in physical space is called a *Cspace obstacle for A due to B* and denoted by $CO_A(B)$. The problem of motion planning for A in physical space is transformed into the problem of finding a path for the point in the configuration space such that every point on the path is legal.

* The support of the National Science Foundation under Grant IRI-88-02457 is gratefully acknowledged. This work is supported in part by an appointment to the U.S. Department of Energy Postgraduate Research Program administered by Oak Ridge Associated Universities and by the Engineering Research Program of the Office of Basic Energy Sciences, of the U.S. Department of Energy, under contract No. DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

Figure 1 illustrates the concept of configuration space in a 2-dimensional space. In Fig. 1a, P and Q are fixed objects in physical space, and R is the robot to be moved. We assume that the orientation of R in the physical space is fixed. Figure 1b represents R's configuration space or $Cspace_A$. The two objects delineated by thick lines are $CO_R(P)$ and $CO_R(Q)$ with respect to the reference point of R in Fig. 1a.

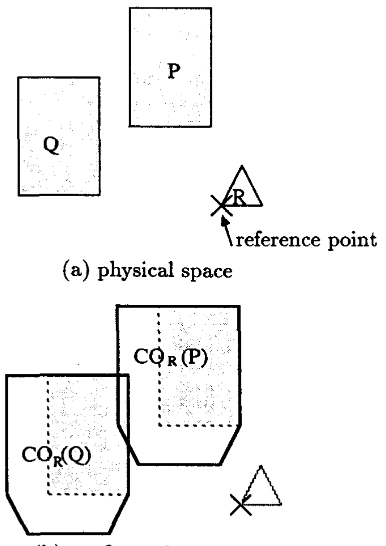


Fig. 1. Physical space and its configuration space. Obstacles delineated by thick lines represent configuration space obstacles corresponding to the shaded obstacles with respect to the reference point in R.

As in our problem, when the orientation of the robot, say A, is fixed, $Cspace_A$ is 2-dimensional. Also, when both A and B are convex polygons, $Cspace_A(B)$ is a convex polygon and the shape is given by taking the convex hull of $vert(B) - vert((A)_0)$. Here $vert(X)$ is the set of vertices of the polygon X; $X - Y = \{x - y | x \text{ in } X \text{ and } y \text{ in} Y\}$; and $(X)_0$ is the polygon X in its initial configuration, where its reference vertex is at the origin [Loza83]. For an n-sided polygon A and an m-sided polygon B, $CO_A(B)$ is at most $(n+m)$-sided [Kede85].

As a result of the expansion of obstacles, configuration space obstacles may overlap. In a time-varying environment, it is possible for two configuration space obstacles that do not overlap at their initial positions to begin to overlap for a certain period of time, then cease to overlap and move away from each other. This happens even when physical obstacles do not overlap at all. This motion of two overlapping obstacles can act as a gate-opening for a point-robot in configuration space when the robot needs to pass through between the two obstacles.

Alternatively, we can think of a situation in which a gate closes (e.g., two obstacles begin to overlap, thereby preventing the robot from passing through between them). In such a situation, if the robot is to pass through between the obstacles, then it must arrive at the gate just as it closes. Also, when two obstacles start overlapping, it is possible for the robot to be crushed between the obstacles. When the robot cannot escape from being crushed, the planner needs to report that there is no collision-free motion.

Overview of the Accessibility Approach

Fujimura and Samet [Fuji89b] propose an algorithm to produce a time-optimal motion for a point-robot among non-overlapping moving obstacles, assuming that the point-robot can move faster than the obstacles. This approach makes use of the concepts of accessibility and collision front. These concepts are used in this paper and are reviewed briefly below.

Accessibility: Let R be a point robot located initially at O at time t_0. Suppose that R starts moving at time t_0 at a speed v. After R starts moving, it moves in a fixed direction. A point V (V is either the destination point or a vertex of a polygonal obstacle) is *accessible* from O if there exists a direction of the motion of R such that R meets V without prior interception by any other movement. We say that V and R *meet* if there exists a location X through which both V and R pass at the same time t ($t_0 < t$). The location X is called an *accessible point* of V. The time t is called the *accessible time* of X with respect to V and is denoted by $t(V)$. The accessible point of a vertex varies for different values of the speed and the initial location of R. Note that the accessible point of a stationary vertex V is V itself, if applicable.

Collision Front: Consider an environment that contains one movement of an edge, L. Let V_a and V_b be the two endpoints of L; and let P_a and P_b be accessible points corresponding to V_a and V_b, respectively, with respect to R's initial location O, start time t_0, and speed v. The set of accessible points corresponding to all points in L forms a segment. This segment is known to be either a straight line or a quadratic curve [Fuji89a]. We will call this segment a *collision front* of L (with respect to O, t_0, and v). Figure 2 contains an example of the collision front. It can be shown that P_a and P_b are the two endpoints of the collision front. For an environment that contains more than one movement, there can be more than one collision front, each of which corresponds to some movement. In this case, however, it is possible that only a part of an obstacle is accessible. Notice that if two moving obstacles (e.g., edges) do not collide, then the corresponding collision fronts do not intersect.

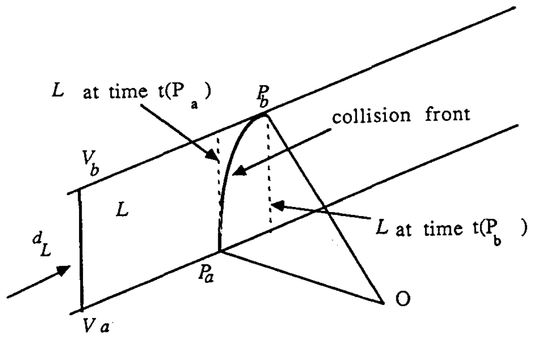


Fig. 2. Collision front.

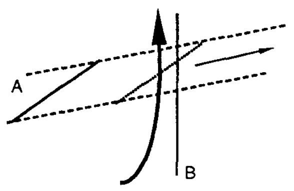
An alternative way of looking at the significance of a collision front is as follows. Suppose that R departs O at time t_0 and keeps moving at a constant speed v along a ray, say l, emanating at O. R does not meet any of the obstacles in motion if and only if l does not intersect any of the collision fronts. The collision front of a stationary edge L is L itself or subsets of L.

Previously, we demonstrated the following result [Fuji89b]. Along a time-optimal motion, the point-robot first moves straight from a start point to one of the endpoints of the collision front, say V, at its maximum speed. From V, it moves in the direction of another endpoint of the collision front generated at V having V as the start point while again moving at a maximum speed. This process is repeated until the destination point is finally reached.

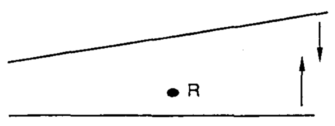
MOTION PLANNING AMONG MOVING OBSTACLES

From now on, we consider the problem of motion planning among moving obstacles in terms of the configuration space. In other words, the robot is treated as a point, and obstacles may overlap. In the following sections, we just use the term 'obstacle' to mean a configuration space obstacle. In this paper, we consider all types of overlaps between polygonal obstacles, thereby allowing types of overlaps that are impossible for configuration space obstacles. Thus, we are solving a slightly more general problem than the one stated in Section 1. As mentioned in Section 2, if obstacles do not overlap, then the robot only needs to move in the directions of vertices of the obstacles at its maximum speed. In an environment that contains overlapping obstacles, splits and merges of obstacles are critical events.

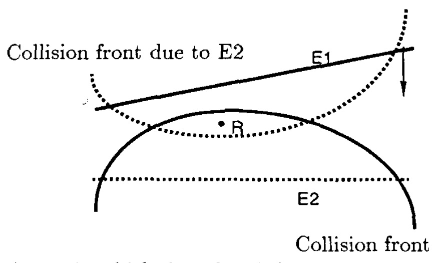
We define splits and merges in terms of movements (or edges in motion). A *merge* of two edges means that two disjoint edges begin to cross each other (Fig. 3a). It is possible that when a merge takes place, the robot is crushed between the two closing obstacles (Fig. 3b). This means that no matter what action is taken by the robot, it cannot escape from being crushed. In terms of accessibility, the event is indicated by crossing collision fronts (Fig. 3c).



(a) Edge A eventually intersects edge B. When A intersects the pathway indicated by the thick arrow disappears.



(b) It is possible that the robot (R in the figure) is crushed no matter what action it takes.



(c) A case in which the robot (R) cannot escape from being crushed. The situation is indicated by crossing collision fronts.

Fig. 3

A *split* of two edges, when intersecting edges cease to intersect, requires special handling. Such an event should not be missed, since a new pathway may be opened. For example, suppose that edges E_1 and E_2 intersect. A split occurs when a vertex of E_1 passes E_2 (or vice versa). The location at which the split takes place is called a ***split point***, and the time at which it occurs is termed a ***split time***. Note that even when a split of two edges occurs, the two obstacles having those edges as their sides may still overlap. It is important that the robot be at a split point at its split time, if it is possible (see [Fuji89a]). The question is how the robot arrives at a split point exactly at its split time. For this purpose, when moving obstacles overlap, we move the robot also in the direction of the following auxiliary points on an edge. These points serve as subgoals, in the process of planning a motion. We use two types of subgoals depending on whether or not the split point is accessible. Note that when we discuss splits, we are speaking about edges rather than obstacles.

Subgoal Type 1: Figure 4a shows two movements of edges AB and CD that are about to split. Let S be the point at which the point-robot R is initially found, and let G denote the destination point. The split point P (Fig. 4b) is not accessible from S, because edge CD lies between S and P. Let X be the intersection point between AB and CD. As edge CD moves, the intersection point X also moves towards the split point P (see Fig. 4b). In general, when an edge, say AB, crosses another edge, say CD in motion, their intersection point makes a straight motion (i.e., in a fixed direction with a fixed speed). We treat X as a subgoal; in other words, R aims at X. After it reaches X, the motion of R is identical with X until it reaches the split point. As X serves as a point to be aimed at, we call it a *pseudo-vertex*. In order to avoid confusion, we use the term *real-vertex* to indicate a vertex of a polygon.

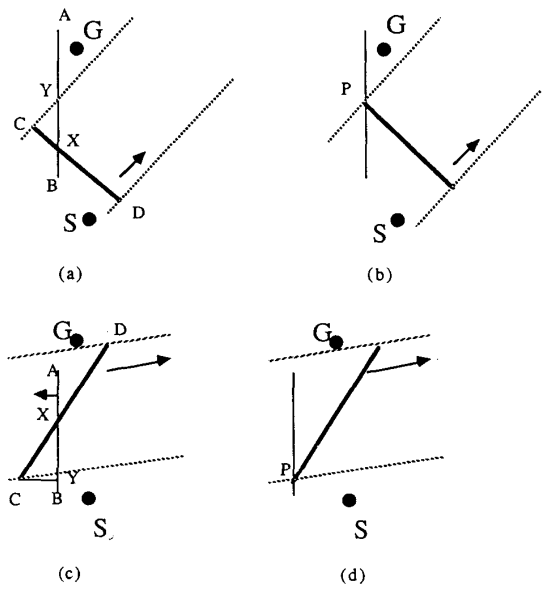
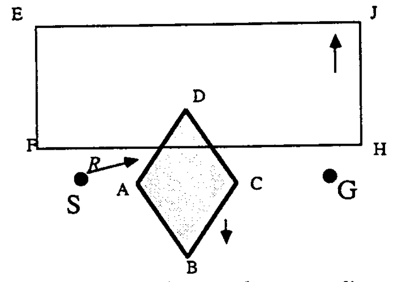
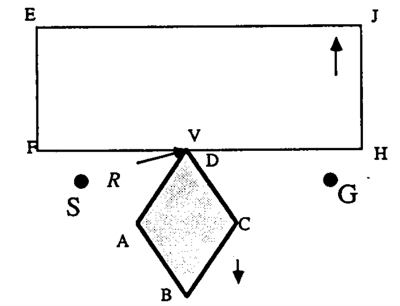


Fig. 4. P denotes the split point. In (a) and (b), the split is not accessible from S. In (c) and (d), the split point is accessible from S.

Consider polygons $ABCD$ and $EFHJ$ in Fig. 5. $ABCD$ and $EFHJ$ are about to separate. Edges AD and FH, and edges DC and FH split. Figure 5b shows the moment when these three edges split. At that moment, the split point V coincides with vertex D and lies on edge FH. Let us take the intersection point X of edges FH and AD as our subgoal. The intersection point of edges DC and FH is also a candidate for the subgoal, but this point is not accessible from S, the current location of the point-robot. (The procedure to determine accessible subgoals from the current robot location is described in the next section.) Intersection point X is chosen as a subgoal because it converges to the split point V. In fact, any point that lies in the free space (i.e., outside any of the obstacles) and converges to the split point can serve as a subgoal. It can be shown that of all the points that converge to the subgoal when the split point is not accessible, X moves the slowest. This means that if subgoal X moves faster than the robot, then all other choices of a subgoal would move faster than the robot.



(a) Two obstacles are about to split.



(b) The moment of split. V is the split point.

Fig. 5

Note that this type of subgoal may chain, as shown by the example in Fig. 6. In this case, polygon ABC intersects with stationary polygon $DEFHIJ$. Suppose that the robot (initially located at S in Fig. 6a) needs to reach point G. In Fig. 6a, edges AC and BC intersect with edge DE. The two intersection points can be considered as a subgoal. In this case, only one intersection (the intersection of AC and DE) is directly accessible from S. This subgoal moves along edge DE (Fig. 6b) until it reaches vertex E, where a split of edges AC and DE takes place. Next, we consider the intersection between edges AD and EF. The intersection point can then be considered as a subgoal (Fig. 6c) until a split of edges AC and FH occurs. Finally, the intersection point of edges AC and FH serves as a subgoal (Fig. 6d). In summary, we can consider this chain of intersection points as a path that eventually leads the robot to the split point of two obstacles at its split time.

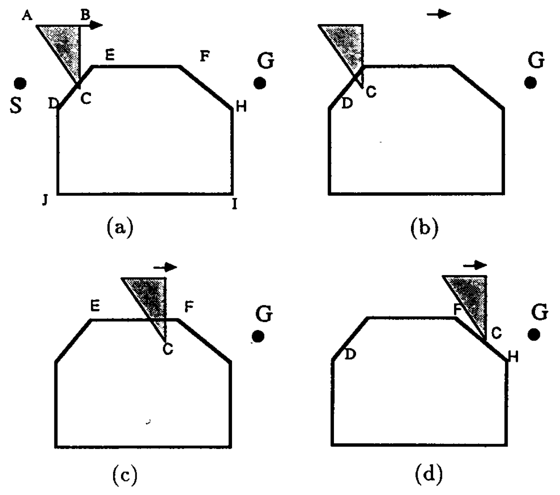


Fig. 6. A subgoal chain.

Subgoal Type 2: Figures 4c and 4d show another type of split. A split occurs when edge CD passes through P. P in Fig. 4d is the split point, and Y is a point on edge AB that passes through P. In this case, the split point P is accessible from the current location of the robot. We consider Y as a subgoal, since it converges to the split point. Such a point on an edge is also termed a pseudo-vertex. We aim the robot at Y. Once it reaches Y, the motion of the robot is identical to that of Y until it reaches the split point. It is always possible to follow Y because the speed of Y is the same as that of the obstacle on which Y is found. In this case, the intersection point X could have also been used as a subgoal. However, it is possible that X moves faster than the maximum speed of the point (depending on the angles between the two edges), and in this case, X cannot serve as a subgoal. Note that in Fig. 4a, it is not possible for the robot to aim at point Y because the path would be obstructed by edge CD itself.

Figure 7 illustrates this type of subgoal using two polygons. Rectangle $ABCD$ moves in direction d and triangle EFH is a stationary obstacle. Vertex F is a split point whose split time is the instant at which edge BC passes vertex F. Y is the pseudo-vertex on edge BC and γ indicates Y's trajectory. Suppose that S is the starting point of the robot R. Motion π, a straight-line motion from S to F, seems to be a natural choice for R's motion in approaching the split point. Along SF, R moves at a constant speed such that R can reach the split point exactly at its split time. This motion π can also serve as a subgoal that R can aim at. Clearly, there may be many ways of moving a robot from a current position to a split point at its split time. Our strategy is first to move to a subgoal with a maximum speed, and then to follow the motion of the subgoal. This choice of motion for the robot makes it easier to design an algorithm.

When polygonal obstacles do not overlap, the robot can reach the final destination without having to visit the same vertex of a polygonal obstacle more than once [Fuji89b]. However, in a situation that allows overlaps among obstacles, the robot may have to visit the same vertex more than once. Figure 8 illustrates such a situation. Triangle ABC is an obstacle that is moving towards the right. DEF and HJK are stationary obstacles. S is the starting point for the robot R. We assume that edge DE

is sufficiently far from S so that moving toward DE is not an adequate solution. We assume that edge HJ is also sufficiently far from S. Figure 8a shows the moment when R arrives at vertex C. After leaving vertex C, R moves to F and meets with vertex C again (Fig. 8b). Next, R moves to vertex K (Fig. 8c), after which it meets with C again (Fig. 8d).

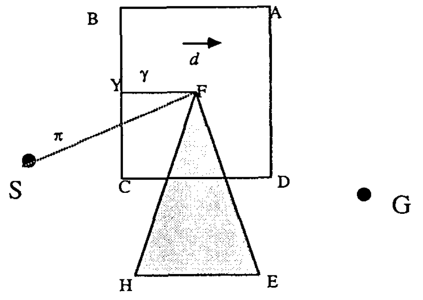


Fig. 7. Example of an alternative motion.

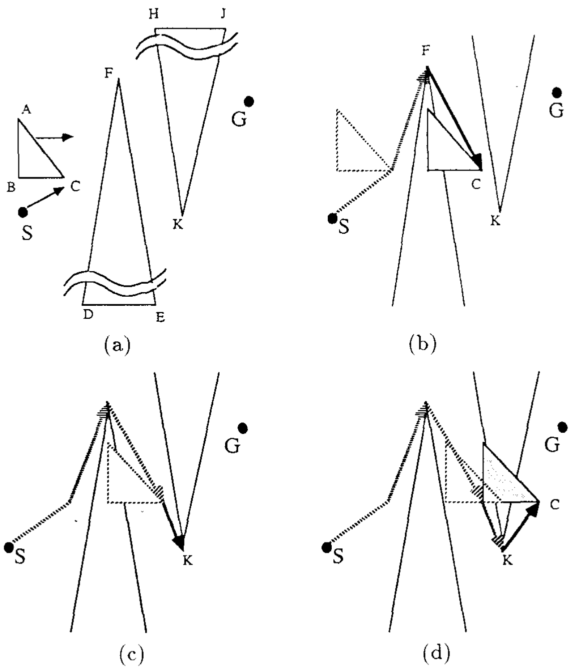


Fig. 8. A scene where the robot visits the same vertex more than once.

In general, a vertex of an obstacle can be treated as a new vertex after it has moved through the inside of an obstacle (e.g., vertex C in Fig. 8). Let us call a vertex *live* when it is not covered by any of the obstacles, and let a *live-period* of a vertex be a time interval during which the vertex is live. In Fig. 8, vertex C has the following three live-periods. The period before C intersects edge DF. The period after C intersects edge EF, and before it intersects edge HK. The period after C intersects edge JK. The robot does not have to visit the same vertex more than once while the vertex is in the same live period.

So far we have introduced two types of new subgoals and defined motions to be taken when a split occur. As noted in Fig. 7, there are other ways of moving the point to a split point at its split time. We treat the motion defined in Section 3 as a canonical form for motion planning. It can be shown that whenever there is a motion to reach the split point at its split time, there is a canonical motion to reach the same split point.

In our approach, a motion from a given start point to a destination point takes place as follows. After the robot leaves its start point, it moves to one of the subgoals in a shortest time. After it reaches a subgoal, it follows the motion of the subgoal to its split point. After the split time, the robot again moves to another subgoal in a shortest time. This process is repeated until the robot eventually reaches the final destination point.

ALGORITHM

We now describe an algorithm to find a collision-free motion and analyze its complexity. Let n be the total number of vertices in the original environment. The number of total vertices in the configuration space produced from the given environment is $O(n)$, assuming a bounded number of vertices of the robot [Kede85]. Let k be the total number of merges and splits that occur between a vertex and an edge. The value of k depends on the nature of the obstacles. It is zero when configuration space obstacles (i.e., enlarged obstacles) do not overlap at any instance of time. In the worst case, the size becomes $O(n^2)$, since every obstacle can overlap every other obstacle.

The idea behind our algorithm is as follows. When obstacles do not overlap as in [Fuji89b], the robot need only move in the directions of the vertices of the polygonal obstacles along a straight line at its maximum speed. If obstacles do overlap, then the robot also needs to move in the direction of the two types of subgoals. After the robot reaches a subgoal, it moves along it until it comes to a split point. At a split point, it needs to change its direction towards its next objective (a vertex of an obstacle or another subgoal), until it reaches its final destination point. During this process, the robot may encounter the same vertex of a polygonal obstacle more than once, if necessary.

Description

(1) Create configuration space obstacles from the given set of physical polygonal obstacles and the given polygonal robot. See [Loza83, Whit86] for a detailed algorithm. It takes $O(n_i+m)$ time to create a configuration space for an input polygon of n_i vertices and a polygonal robot with m vertices. Therefore, it takes $O(n_1 + ...n_l + ml)$ time to create configuration space obstacles from a given environment with a total of l obstacles. Thus, it amounts to $O(n)$ time, assuming that the number of vertices of the robot is bounded.

(2) Enumerate all the pseudo-vertices. This can be done naively by checking each vertex against all the edges in the environment. By doing so, all the split points and split times are determined. From a split point and its split time, a pseudo-vertex and its motion are also determined. At the same time, determine the live-periods of each real-vertex. A real-vertex has up to $O(n)$ live-periods, since a real-vertex can be covered by obstacles at most $O(n)$ times. Since the total number of live-periods of real-vertices does not exceed the total number of merges and splits, it is $O(k)$.

When a vertex of an obstacle intersects an edge, the intersection point on the edge becomes a pseudo-vertex.

Since there are k such intersections, the number of pseudo-vertices is $O(k)$. It is also necessary to identify the trajectory of a pseudo-vertex. Since a pseudo-vertex is always coincident with a point on an edge, its motion is determined by that edge. However, it is necessary to determine when and where the pseudo-vertex starts. To determine the starting point, the trajectory is checked against all edges in the environment. This takes $O(n)$ time.

(3) There are $n + k$ real and pseudo-vertices. Let $N = n + k$. Sort the real and pseudo-vertices in a clock-wise manner with respect to the current point. This takes $O(N \log N)$ time.

(4) Rotate a ray emanating from the current point about the current point. The ray halts each time it intersects a vertex. At this time, check whether the vertex is accessible from the current point. This can be done by using a balanced binary tree (e.g., a 2-3 tree [Aho74]). After $O(N \log N)$ time, all accessible vertices from the current point are determined. If no vertices are accessible, then the current point is a dead end.

(5) Maintain a priority queue of vertices, where the priority corresponds to the accessible time of the vertex. Choose the vertex whose associated time is the youngest, say Y. If Y is the destination point, then stop. When Y is a real-vertex, repeat steps (3) and (4) with Y as the current point. When Y is a pseudo-vertex, pick the split point Z, and time associated with Z, and repeat steps (3) and (4) with Z as the current point.

(6) Repeat steps (3), (4) and (5) at most N times.

Figure 9 is a simple example of motion planning for a triangle robot (R) among two rectangular moving obstacles (P and Q). The five parts of Fig. 9a show the motion in $Cspace_R$, and the five parts of Fig. 9b show the corresponding motion in physical space. R first moves to X (see Fig. 9a (ii), (iii)), a pseudo-vertex. After R reaches X, it stays on X until it reaches Y, which is a split point. After P and Q have separated, R moves in the direction of real-vertex in the scene until it reaches the destination point G. Note that in Fig. 9b (iii) and (iv), the robot R touches both of the obstacles P and Q.

The execution time required for each step in the procedure is noted with each step. Altogether, it takes $O(n^2) + O(nk) + O(N^2 \log N) = O((n+k)^2 \log(n+k))$ to produce a motion. The value of k can be as large as n^2, and thus, the worst-case execution time is $O(n^4 \log n)$.

PRIOR WORK

Dynamic motion planning has been shown to be computationally harder than motion planning in a stationary environment. Canny and Reif [Cann87] show that the problem is NP-hard when some obstacles can move faster than the robot. Reif and Sharir [Reif85] show that the problem is solvable in a polynomial time when the number of obstacles is bounded. Kant and Zucker [Kant86] use a two-step algorithm. They plan a path to avoid stationary obstacles in the first step, and determine a velocity profile along the path to avoid moving obstacles in the second step. Erdmann and Lozano-Perez [Erdm86] represent the motions of the obstacles as a set of slices in space-time. These slices represent configuration spaces at particular times. The times are those instants at which some moving obstacle changes its velocity. A path of the robot consists of a set of path-segments which starts at a vertex of an obstacle in one slice and terminates at a vertex of an obstacle in the next slice. Their approach is successful when the topology of the free space does not change (i.e., the obstacles are not allowed to merge or split as opposed to our approach) and requires $O(rn^3)$ time, where n is the total number of edges in the environment and r is the number of slices constructed. None of the above approaches consider splits and merges of moving obstacles.

Kedem and Sharir [Kede85] have studied the complexity of motion planning among stationary obstacles with robot under translation. They developed an algorithm that runs in time $O(n \log^2 n)$, where n is the total number of corners in the obstacles. Bhattacharya and Zorbas [Bhat88] have proposed a new approach that takes $O(n \log n)$ to compute a translational motion among stationary convex polygons. Finding a shortest path among stationary obstacles for a point-robot takes $O(n^2)$ time [Asan85, Gosh87, Welz85].

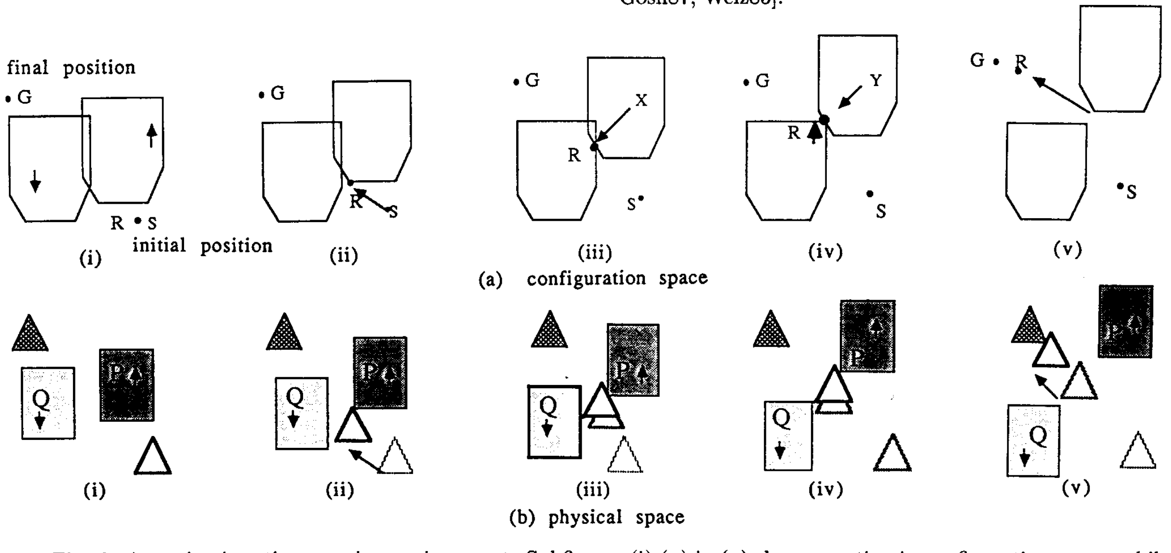


Fig. 9. A motion in a time-varying environment. Subfigures (i)-(v) in (a) show a motion in configuration space, while subfigures (i)-(v) in (b) show the corresponding motion in physical space.

CONCLUDING REMARKS

Motion planning for a convex robot has been described. The concepts of configuration spaces and accessibility have been used to produce an algorithm to find a collision-free motion among moving polygons in a 2-dimensional plane. Our algorithm takes $O((n+k)^2 \log(n+k))$ to determine a motion, where n is the number of total vertices and k is the number of occurrences of overlaps in the configuration space. In addition to vertices in the input obstacles, two types of pseudo-vertices are introduced to handle configuration space obstacles. We have assumed that the obstacles move in fixed directions at constants speed. Our approach can be extended to allow piecewise linear motions of the obstacles. See [Fuji89a] for more details.

In our formulation, we permit arbitrary overlaps among moving obstacles. This situation is not allowed by the conventional approach to configuration space obstacles. Thus, we have solved a more general class of the problem. It is not clear yet whether it is possible to reduce the amount of computation for motion planning by examining the types of overlaps produced by configuration space obstacles.

Our formulation, however, has another interpretation. We can view obstacles as search lights projected on the ground. The projections of the lights sweep the ground in a scheduled manner. Given the schedule of the search lights, our motion planning algorithms provide a way for a culprit to move from one place to another without being detected by any of the lights.

It should be pointed out that our approach can be adapted to an environment with uncertainty. Such a situation arises due to errors in the measurement of the motion of the obstacles. Consider an obstacle that moves in a fixed direction at a speed v, where $v_1 \leq v \leq v_2$ for some known v_1 and v_2. In such a case, collision fronts of two obstacles may overlap even when the obstacles themselves do not overlap. When we wish to plan a motion under such uncertainty, our approach can be used as it allows overlaps between obstacles. Note also that often the destination point is temporarily covered by some of the obstacles. Our algorithm could be extended to handle such a situation.

REFERENCES

[Aho74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[Asan85] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility of Disjoint Polygons," *Algorithmica* **1**(1), pp. 49-63 (1988).

[Bhat88] B. K. Bhattacharya and J. Zorbas, "Solving the Two-Dimensional Findpath Problem Using a Line-Triangle Representation of the Robot," *Journal of Algorithms* **9**, pp. 449-469 (1988).

[Cann87] J. Canny and J. Reif, "New Lover Bound Techniques for Robot Motion Planning Problems," *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science,* pp. 49-60, Los Angeles, CA, October 1987.

[Erdm87] M. Erdmann and T. Lozanno-Perez, "On Multiple Moving Objects, *Algorithmica* **9**(4), pp. 477-522 (1987).

[Fuji89a] K. Fujimura, "Motion Planning in Dynamic Domains," Ph.D. dissertation, TR-2377, Computer Science Department, University of Maryland, College Park, MD, December 1989.

[Fuji89b] K. Fujimura and H. Samet, "Time-Minimal Paths Among Moving Obstacles, *Proceedings of IEEE International Conference on Robotics and Automation* **2**, pp. 1110-1115, Scottsdale, AZ, May 1989.

[Ghos87] S. K. Ghosh and D. M. Mount, "An Output Sensitive Algorithm for Computing Visibility Graphs," *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science,* pp. 11-19, Los Angeles, CA, October 1987.

[Kant86] K. Kant and S. W. Zucker, "Toward Efficient Planning: The Path-Velocity Decomposition," *International Journal of Robotics Research* **5**(3), pp. 11-19, Fall 1986.

[Kede85] K. Kedem and M. Sharir, "An Efficient Algorithm for Planning Collision-Free Translational Motion of a Convex Polygonal Object in 2-Dimensional Space Amidst Polygonal Obstacles," *Proceedings of the Symposium on Computational Geometry,* pp. 75-80, Baltimore, MD, June 1985.

[Loza79] T. Lozano-Perez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM* **2**(10), pp. 560-570, October 1979.

[Loza83] T. Lozano-Perez, "The Configuration Space Approach," *IEEE Transactions on Computers* **92**(2), pp. 108-120, February 1983.

[Reif85] J. Reif and M. Sharir, "Motion Planning in the Presence of Moving Obstacles," *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science,* pp. 144-154, Portland, OR, October 1985.

[Upud77] S. Upuda, "Collision Detection and Avoidance in Computer Controlled Manipulators," *Proceedings of the 5th International Joint Conference on Artificial Intelligence,* pp. 737-748, Cambridge, MA, 1977.

[Welz85] E. Welzl, "Constructing the Visibility Graph for n Line Segments in $O(n^2)$ Time," *Information Processing Letters* **20**(4), pp. 167-171, May 1985.

[Whit85] S. H. Whitesides, "Computational Geometry and Motion Planning, *Computational Geometry* (Ed., G. T. Toussaint), North-Holland, Amsterdam, The Netherlands, pp. 377-427, 1985.