On the Moving-Obstacle Path-Planning Algorithm of Shih, Lee, and Gruver

Robert A. Conn and Moshe Kam

***Abstract*—We analyze a motion planning algorithm proposed by Shih, Lee, and Gruver [1]. In particular, we discuss the case of motion planning among moving obstacles, for which an algorithm is presented in [1]. The algorithm is claimed to always yield a collision-free path when one exists. We provide a specific counter-example to this claim, examine the possibility of repairing the algorithm, and discuss the computational complexity of a corrected algorithm.**

I. Overview of a Path-Planning Algorithm by Shih et al.

Shih *et al*. consider the problem of motion planning for a point robot among nonrotating polyhedral obstacles that are moving with piecewise constant velocities. The controlled robot also has piecewise constant velocity, and is subject to velocity bounds. The obstacles may overlap, and may have speeds greater than that of the controlled robot.

The authors of [1] present a decomposition of the free space into disjoint polytopes $C(i_1, \ldots, i_n)$. The notation reflects the fact that the polytope $C(i_1, \ldots, i_n)$ is constructed from the linear constraints that form face i_1 of obstacle 1, face i_2 of obstacle 2, and so on to face i_n of obstacle n. The graph of connections between the C polytopes encodes the topology of the free space of the problem. It is shown in [1] that the graph may be more efficiently obtained from the intersection of nonpolytope sets $S(i_1, \ldots, i_n)$ which are also constructed from the obstacle faces. In the sequel, we refer to the C polytopes, the S sets, and the graph exactly as they are defined in [1].

We shall use the notation of [1] to describe positions, i.e.,

$$\begin{aligned} \mathbf{x} &= \text{spatial position in } r-1 \text{ dimensions} \\ t &= \text{time} \\ \mathbf{z} = [\mathbf{x} \quad t]^T &= \text{ a vector of dimension } r \text{ fully describing} \\ &\qquad \text{the robot's spatial-temporal position.} \end{aligned} \tag{1}$$

Also

$z_0 =$ starting position (space-time) of robot, and

$z_g =$ goal position (space-time) of robot.

An algorithm is presented in [1] to obtain a solution to the path planning problem of a bounded-velocity point robot travelling among moving-obstacles. The algorithm begins in the polytope C_0 that contains the starting point z_0. It then searches for an adjacent polytope C_1 that contains a point z_1 that solves the following problem (Problem 1 from [1]).

Problem 1 (Feasible Linear Trajectory) Given z_{i-1}, C_{i-1}, and C_i, find the z_i that minimizes t_i subject to

$$z_i \in (S_{i-1} \cap S_i) \tag{2a}$$

$$\mathbf{v}_{\min} \leq \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{t_i - t_{i-1}} \leq \mathbf{v}_{\max} \tag{2b}$$

$$t_i - t_{i-1} \geq 0. \tag{2c}$$

The algorithm continues to find polytopes $C_2, C_3, \ldots$, etc., until the polytope C_g is reached that contains the goal point z_g. The linear trajectory from the space-time point z_{i-1} to the space-time point z_i is called a "feasible linear trajectory." The piecewise linear trajectory defined by the sequence of space-time points $z_0, z_1, \ldots, z_g$ is called a "feasible trajectory." The result of the algorithm is either a single feasible trajectory or a statement that no feasible trajectory exists.

The exact algorithm, as defined in [1], is:

> To plan a feasible trajectory from z_0 to z_g, we search a feasible path in the free-space graph as follows. First, locate C_0, which contains z_0. Then find an adjacent node C_1 of C_0 such that Problem 1 has a solution z_1. Repeat the procedure until a node C_k containing the goal point z_g has been found and the linear trajectory from z_k to z_g is feasible. If there is a collision-free trajectory from z_0 to z_g, then it will travel a set of feasible nodes in the free-space graph. Therefore, we always can find a solution, if one exists.

In the next section, we construct a counter-example that shows that this algorithm does not always generate a feasible trajectory when one exists.

Manuscript received December 1, 1994; revised February 21, 1996.

The authors are with the Data Fusion Laboratory, Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104 USA.

Publisher Item Identifier S 1083-4419(97)00023-X.

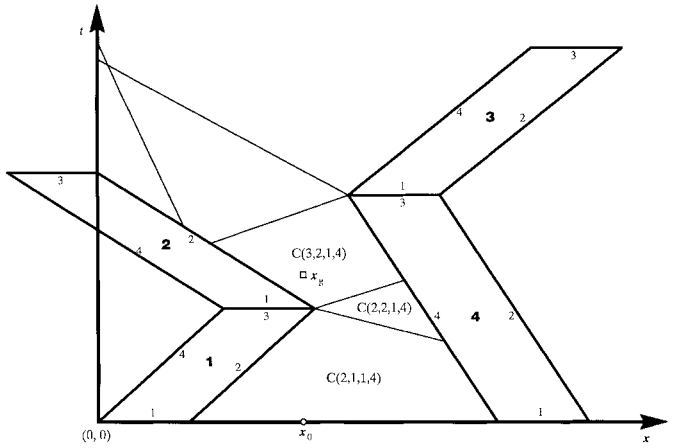

Fig. 1. The nonempty C polytopes for the counter-example.

II. Counter-Example

Consider the case $r = 2$, i.e., one spatial dimension and one time dimension. Fig. 1 shows the regions of space-time swept out by the movement of two physical objects. Each object undergoes one change of velocity, resulting in two polyhedral obstacles for each object (a total of four polyhedral space-time obstacles, labeled **1**, **2**, **3**, and **4**).

At $t_0 = 0$, our point robot is at $\mathbf{x}_0 = 18$. The goal is to reach $\mathbf{x}_g = \mathbf{x}_0$ at $t_g = 13$. The initial position of the robot is marked with a circle in Fig. 1, while the goal point is denoted by a small square. Let the speed bounds on the robot be $\mathbf{v}_{\min} = -1/3$ and $\mathbf{v}_{\max} = 1/3$, per (2b).

At $t = 0$, Object 1 occupies the $\mathbf{x}$-interval $[0, 8]$, and has a velocity of $11/10$. From $t = 0$ to $t = 10$, Object 1 generates obstacle **1**. At $t = 10$, Object 1 changes velocity to $-19/12$. From $t = 10$ to $t = 22$, Object 1 generates obstacle **2**.

At $t = 0$, Object 2 occupies the $\mathbf{x}$-interval $[35, 43]$, and has a velocity of $-13/20$. From $t = 0$ to $t = 20$, Object 2 generates obstacle **4**. At $t = 20$, Object 2 changes velocity to $16/13$. From $t = 20$ to $t = 33$, Object 2 generates obstacle **3**.

Three of the nonempty free-space polytopes are shown in Fig. 1. Fig. 2 shows the regions $S_a = S(2, 1, 1, 4) \cap S(2, 2, 1, 4)$ and $S_b =$

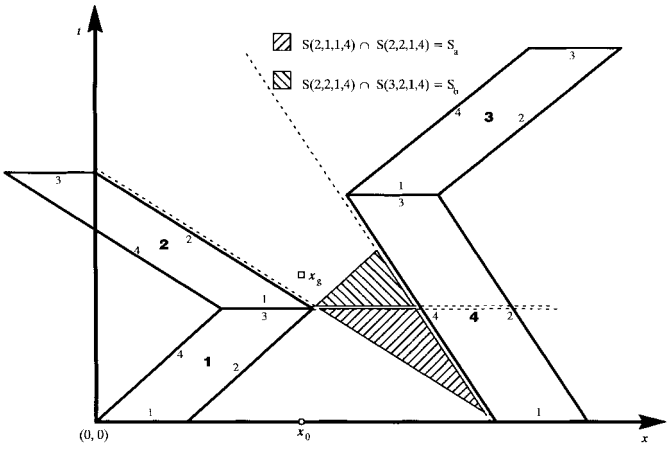

Fig. 2. The S sets for the counter-example.

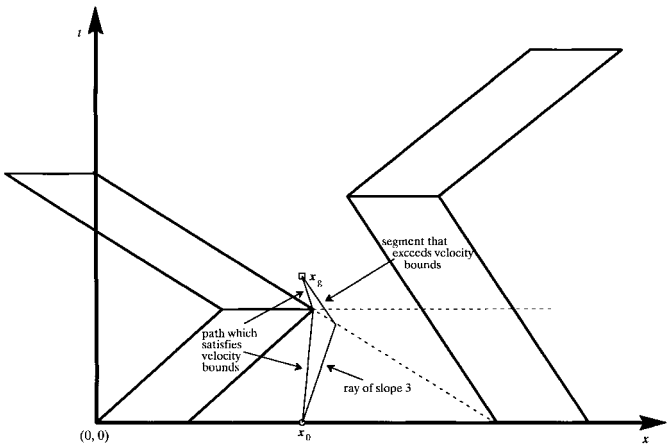

Fig. 3. Path on left is feasible path. Path on right shows the difficulty with the algorithm of [1].

$S(2,2,1,4) \cap S(3,2,1,4)$. We have chosen ε to be 0.0001; its value in Fig. 2 is exaggerated to make the various regions more visibly distinct. Since $S(2,1,1,4) \cap S(2,2,1,4)$ is nonempty, we connect $C(2,1,1,4)$ and $C(2,2,1,4)$ in our graph. Since $S(2,2,1,4) \cap S(3,2,1,4)$ is nonempty, we connect $C(2,2,1,4)$ and $C(3,2,1,4)$ in our graph. Note that $C(2,1,1,4)$ is connected to at least two other C polytopes (not shown) with $t < 0$, but since the intersections of $S(2,1,1,4)$ with these other S sets lie totally below the line $t = t_0 = 0$, they do not represent feasible connections (in the sense of Problem 1).

Since there is only one path in the graph between $C_0 = C(2,1,1,4)$ and $C_g = C(3,2,1,4)$, we know that $(\mathbf{x}_1, t_1) \in S_a$, and $(\mathbf{x}_2, t_2) \in S_b$. To find $\mathbf{x}_1$ and t_1, we turn to the definition of feasibility given in Problem 1 for $i = 1$.

Problem 1 (i=1): Given $(\mathbf{x}_0, t_0), C_0$, and C_1, find $(\mathbf{x}_1, t_1)$ that minimizes t_1 subject to

$$\begin{aligned} (\mathbf{x}_1, t_1) &\in S_a \\ \mathbf{v}_{\min} &\le \frac{\mathbf{x}_1 - \mathbf{x}_0}{t_1 - t_0} \le \mathbf{v}_{\max} \\ t_1 - t_0 &\ge 0. \end{aligned}$$

The solution is $(\mathbf{x}_1, t_1) = (20.927, 8.783)$. Geometrically, this means we follow a ray of slope 3 $(= 1/\mathbf{v}_{\max})$ from $(\mathbf{x}_0, t_0)$ until we intersect S_a. See Fig. 3.

Although the algorithm would proceed another two steps before determining that no feasible solution exists, there is no need to do those computations. To arrive at the goal point at the specified time would require us to travel at a speed of $(\mathbf{x}_g - \mathbf{x}_1)/(t_g - t_1) = 0.69 > \mathbf{v}_{\max}$. So by minimizing t_1, we have ensured that the goal cannot be reached. Again, see Fig. 3.

Let us now confirm that a piecewise linear path from the starting point to the goal which satisfies the speed bounds does, in fact, exist. At $t_0 = 0$, the position is $\mathbf{x}_0 = 18$. Let $t_1 = 10$, and $\mathbf{x}_1 = 19$. This corresponds to traveling from the starting point to the upper right tip of obstacle **1**. The speed for this portion of the path is $1/10$, where $\mathbf{v}_{\min} < 1/10 < \mathbf{v}_{\max}$. The second portion of the path involves traveling from $(t_1, \mathbf{x}_1) = (10, 19)$ to $(t_g, \mathbf{x}_g) = (13, 18)$. The speed for this portion of the path is $1/3 = \mathbf{v}_{\max}$. This feasible trajectory is shown in Fig. 3.

We conclude that the algorithm of [1], for the case of a bounded-velocity point robot travelling among moving obstacles, does not always yield a collision-free path when one exists, in contradiction to its claim.

III. Reason for Occasional Failure of the Moving-Obstacle Algorithm of [1]

The algorithm of [1] instructs the user to minimize the elapsed time at each step of the graph search. The claim is that this procedure will "maintain as many nodes as possible to find a path in the free-space graph" [1]. However, we know that in a multistage optimization process, optimization at each stage does not necessarily result in global optimization. We show by our counter-example that imposing stage-by-stage optimization on the problem of searching for a feasible trajectory in the free-space graph will sometimes result in not finding a solution at all even when one is known to exist.

IV. Repairing the Algorithm

The free-space decomposition proposed by Shih *et al.* allows an effective transformation of the path-planning problem into a graph-search problem for both the static- and moving-obstacle cases. For the static-obstacle case the definition of a "feasibile linear trajectory" provided by the statement of Problem 1 always yields a collision-free path when one exists. However, in the moving-obstacle case, the above definition of feasibile linear trajectory is overly restrictive, and as a result paths which should have been considered are pruned from the graph search.

The root of this deficiency is in the local nature of Problem 1. To repair the algorithm, we change the definition of Problem 1 to create

Problem 1:* Given z_0, z_g and a sequence of nodes $\{C_0, C_1, \ldots, C_k, C_g\}$ in the free-space graph, find $z_1, \ldots, z_k$ such that

$$z_i \in (S_{i-1} \cap S_i) \qquad i = 1, \ldots, k \quad \text{(3a)}$$

$$\mathbf{v}_{\min} \le \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{t_i - t_{i-1}} \le \mathbf{v}_{\max} \quad i = 1, \ldots, k \quad \text{(3b)}$$

$$t_i - t_{i-1} \ge 0$$

$$\mathbf{v}_{\min} \le \frac{\mathbf{x}_g - \mathbf{x}_k}{t_g - t_k} \le \mathbf{v}_{\max} \quad \text{(3c)}$$

$$t_g - t_k \ge 0.$$

Note that Problem 1* does not define a "feasible linear trajectory," since Problem 1* asks us to find a *sequence* of space-time points (as opposed to the *single* point found by Problem 1). The corrected algorithm to find a feasible trajectory is then:

To plan a feasible trajectory from z_0 to z_g, we first locate C_0 which contains z_0, and C_g which contains z_g. Next, we find all connected paths between C_0 and C_g. Let there be p such paths, and denote them by $\{C^{(j)}\} = \{C_0, C_1^j, \ldots, C_{kj}^j, C_g\}$ where $j = 1, \ldots, p$. If Problem 1* with sequence $\{C^{(j)}\}$

has a solution for any $j \in \{1, \ldots, p\}$, then we have found a collision-free path from z_0 to z_g. Since we explore every path through the graph, we will always find a solution if one exists.

The difference between Problem 1^* and Problem 1 is that Problem 1^* examines a candidate path for feasibility *globally* rather than *locally*. The penalty for obtaining this global information is that the determination of the existence of a feasible path with this method has an exponential worst-case computation time (in the number of obstacle faces—exhaustive search of a graph).

V. Conclusion

We have shown that the path-planning algorithm proposed by Shih et al. for a bounded-velocity point robot travelling among moving obstacles does not always yield a collision-free solution when one exists. We have offered a corrected version of the algorithm, which does always find a solution when one exists, but which possesses the unattractive feature of exponential worst-case computation time (in the number of obstacle faces).

References

[1] C. L. Shih, T.-T. Lee, and W. A. Gruver, "A unified approach for robot motion planning," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, pp. 903–915, July/Aug. 1990.

Fuzzy Identification of Systems with Unsupervised Learning

A. M. Luciano and M. Savastano

***Abstract*— The present paper describes a mathematical tool to build a fuzzy model whose membership functions and consequent parameters rely on the estimates of a data set. The proposed method proved to be capable of approximating any real continuous function, also if a strongly nonlinear one, on a compact set to arbitrary accuracy. Without resorting to domain experts, the algorithm constructs a model-free, complete function approximation system. Applications to the modeling of several functions among which classical nonlinear ones such as the Rosenbrock and the sinc (x, y) functions are reported. The proposed algorithm can find applications in the development of fuzzy logic controllers (FLC).**

I. Introduction

The number of applications based on fuzzy logic theory [1], [2] has rapidly grown in these last years. One of the reasons of such a large diffusion consists certainly in being closer to human thinking in the sense that fuzzy logic is similar to the experience of the human controller, usually expressed as some linguistic "IF-THEN" rules stating in which situation(s) which action(s) should be taken. It must be however emphasized that one point of weakness of the Fuzzy Logic approach is the lack of fixed rules to decide the optimal membership function shape and overlap.

Several methods [3]–[7], [10]–[14] for creating fuzzy systems from data have been proposed in the literature, but neither a standard nor a general procedure exists for constructing membership functions automatically from the example data in order to maximize the performance index.

The present paper introduces a method for generating unambiguously fuzzy rules from numerical data so as to produce an accurate mapping from the input to the output space.

Manuscript received January 15, 1995; revised November 12, 1995.

A. M. Luciano is with the Department of Electronics, "Federico II," University of Napoli, 80123 Napoli, Italy.

M. Savastano is with I.R.S.I.P., National Research Council of Italy (C.N.R.), 80123 Napoli, Italy (e-mail: ISCS@nadis.unina.it).

Publisher Item Identifier S 1083-4419(97)00026-5.

II. Description of the Methodology

The typical steps of a "fuzzy reasoning" consist of:

—*Fuzzification*: comparison of the input variables with the "membership functions" of the premise part in order to obtain the membership values.

—*Weighing*: combination, by means of a specific operator, normally multiplication or minimum, of the premise part membership values to get the firing strength of each rule.

—*Generation*: creation of the consequents relative to each rule.

—*Defuzzification*: aggregation of the consequents to produce the output.

Using the symbolism proposed by Wang [9] and with reference to a multi-input single-output (MISO) application, let us suppose that:

- The system, in fuzzy rule terms, has the form

$$R_j\text{: IF } x_1 \text{ is } A_1^j \text{ and } x_2 \text{ is } A_2^j \text{ and } \cdots \text{ and } x_n \text{ is } A_n^j \text{ THEN } z \text{ is } c^j \tag{1}$$

where:

R_j is the jth rule of the fuzzy system.
n is the number of input variables.
$x_i (i = 1, 2, \ldots, n)$ are the inputs to the fuzzy system.
$A_i^j (j = 1, 2, \ldots, k)$ is the jth fuzzy set defined in the ith input subspace.
$c^j (j = 1, 2, \ldots, k)$ is the output value inferred by the jth rule.
k is the number of the implications.

- All the membership functions are in Gaussian form

$$\mu_{A_i^j}(x_i) = \exp\left[-\frac{1}{2}\left(\frac{x_i - \bar{x}_i}{\sigma_i}\right)^2\right]. \tag{2}$$

- The consequence part is characterized by crisp values of the output.
- The fuzzy system is defined with the algebraic product according to the triangular norms (Table I). Then the proposition x_1 is A_i^j and$\cdots$and x_n is A_n^j is true as

$$\left|x_1 \text{is } A_i^j \text{and } \cdots \text{ and } x_n \text{ is } A_n^j\right| = A_1^j(x_1) \bullet \cdots \bullet A_n^j(x_n).$$

- The defuzzification strategy is the centroid method specified by a relation as follows:

$$f(x) = \frac{\sum_{j=1}^{K}\left(c^j \prod_{i=1}^{n} \mu_{A_i^j}(x_i)\right)}{\sum_{j=1}^{K}\left(\prod_{i=1}^{n} \mu_{A_i^j}(x_i)\right)}. \tag{3}$$