A genetic–fuzzy approach for mobile robot navigation among moving obstacles

Dilip Kumar Pratihar *, Kalyanmoy Deb, Amitabha Ghosh

Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology, Kanpur 208 016, India

Received 1 February 1998; accepted 1 October 1998

Abstract

In this paper, a genetic–fuzzy approach is developed for solving the motion planning problem of a mobile robot in the presence of moving obstacles. The application of combined soft computing techniques – neural network, fuzzy logic, genetic algorithms, tabu search and others – is becoming increasingly popular among various researchers due to their ability to handle imprecision and uncertainties that are often present in many real-world problems. In this study, genetic algorithms are used for tuning the scaling factors of the state variables (keeping the relative spacing of the membership distributions constant) and rule sets of a fuzzy logic controller (FLC) which a robot uses to navigate among moving obstacles. The use of an FLC makes the approach easier to be used in practice. Although there exist many studies involving classical methods and using FLCs they are either computationally extensive or they do not attempt to find optimal controllers. The proposed genetic–fuzzy approach optimizes the travel time of a robot off-line by simultaneously finding an optimal fuzzy rule base and optimal scaling factors of the state variables. A mobile robot can then use this optimal FLC on-line to navigate in presence of moving obstacles. The results of this study on a number of problem scenarios show that the proposed genetic–fuzzy approach can produce efficient knowledge base of an FLC for controlling the motion of a robot among moving obstacles. © 1999 Elsevier Science Inc. All rights reserved.

Keywords: Genetic algorithms; Fuzzy logic controller; Genetic–fuzzy system; GA-based learning; Dynamic motion planning

*Corresponding author. E-mail: dkpra@iitk.ernet.in

0888-613X/99/$ see front matter © 1999 Elsevier Science Inc. All rights reserved.
PII: S0888-613X(98)10026-9
1. Introduction

Soft computing techniques enable handling of imprecision and uncertainty often encountered in solving practical problems requiring reasoning and learning. Soft computing techniques involve computations related to neural network (NN), fuzzy logic technique (FL), genetic algorithm (GA), and others. Over the years, researchers have found suitability in using hybrid techniques involving the above three methods, such as GA–NN, FL–NN, FL–GA and GA–FL–NN. Researchers and practitioners are finding these methods increasingly useful in various problem domains, not because they are new and interesting, but because they have inherent capabilities of handling imprecision and uncertainty with a reasonable amount of computational complexity.

In this paper, we describe a genetic–fuzzy technique based on a combined approach of genetic algorithm and fuzzy logic technique (GA–FL) to solve mobile robot navigation problems in the presence of moving obstacles. In the past, there were two basic approaches developed using FL and GA. In the first approach, fuzzy logic technique was used to improve the performance of a GA [1–4], whereas in the second approach, a GA was used to design optimum fuzzy logic controllers (FLCs) [5–13]. Our study falls in the second category, where a GA is used to find an optimal knowledge base needed to solve the motion planning problem of mobile robots.

In the area of robotics, one of the main areas of research is to build autonomous intelligent robots, which can plan their own motion during navigation through two-dimensional or three-dimensional terrains. A considerable amount of work has been carried out to develop suitable methods for motion planning in the presence of static as well as moving obstacles separately. Latombe [14] provides an extensive survey of different classical approaches of motion planning, particularly in the presence of stationary obstacles. Both graphical as well as analytical methods have been developed by several investigators to solve the mobile robot navigation problems among moving obstacles, known as dynamic motion planning problems. These methods include path velocity decomposition [15–19], accessibility graph technique [20], incremental planning [21,22], probabilistic approach [23–26], techniques using relative velocity paradigm [27], potential field approach [28–34], and others. Moreover, different learning techniques have also been used by researchers to improve the performance of conventional controllers [35–39]. Each of these methods has its own inherent limitations and is capable of solving only a particular type of problems. Canny and Reif [45] studied the computational complexity of some of these methods and showed that motion planning for a point robot in a two-dimensional plane with a bounded velocity is an NP-hard problem, even when the moving obstacles are convex polygons moving at a constant linear velocity without rotation.
Potential field method [28–34], in which a robot moves under the action of combined attractive and repulsive potentials created artificially, is the most widely used technique for solving dynamic motion planning problems. Since at every time step a new potential field must be created to find an obstacle-free direction, the method is local in nature and often has the chance of converging to a sub-optimal solution. Moreover, it is intuitive that many computations of such local travel directions using artificial potential field method may be computationally expensive.

Besides these, reactive control strategies are used by many investigators [40–44], in which robotic actions are decomposed into a collection of primitive motor behaviors. Such multiple motor behaviors act in a concurrent manner to yield a globally emergent behavior that strives to satisfy a robot's goals. The main disadvantage of the reactive control lies in the fact that its flexibility is less because hard-wired behaviors are unable to handle environments which the programmer did not foresee initially.

To reduce the computational complexity, some heuristics have also been developed by several researchers. Dynamic motion planning is based on sensor readings and future prediction of location of moving obstacles. Sensor readings are associated with imprecision and uncertainty. Therefore, an FLC is a natural choice for solving this type of problems. FLCs have been used by several investigators in the recent past [46–50] to solve the dynamic motion planning problem. However, in all such studies, no effort was made to find optimal FLCs (instead, an FLC was designed based on a particular user-defined membership function and rules). Thus, the obtained collision-free paths for the mobile robot need not be optimal. With the availability of a versatile yet efficient optimization method (GA), optimal FLCs for dynamic motion planning problems can be developed, like they have been used in other applications of FLCs, such as the cart-pole balancing [5], cart centering [7], and others.

In the remainder of this paper, we describe the genetic-fuzzy approach by drawing a simile of the motion planning problem with a natural learning process. The proposed approach incorporates some practical considerations, which, along with the use of an FLC makes the overall approach easier to be used in practice. The optimization procedure can be used off-line and an optimal FLC can be obtained before-hand by using a number of user-defined scenarios. Thereafter, the robot can use the obtained optimal FLC to navigate itself in real-world, unseen scenarios. The efficacy of the proposed approach is demonstrated by solving a number of motion planning problems.

2. Genetic-fuzzy approach

There is a natural connection between the dynamic motion planning (DMP) problem of robots and a combined approach of genetic algorithms and fuzzy
logic method. Before we discuss this connection, we state the assumptions made about the DMP problem:
1. The robot is considered to be a single point.
2. No kinematic constraints limit the motion of the robot. The motions are only constrained by the moving obstacles.
3. Each obstacle is represented by its bounding circle, although this is not a rigid limitation.
4. The obstacles are disjoint, that is, no two obstacles are allowed to overlap at any time.

The purpose of the DMP problem of a robot is to find an obstacle-free path which takes a robot from a point \( S \) to a point \( G \) with minimum time. There are essentially two parts of the problem:
1. learn to find any path from point \( S \) to \( G \) that avoids all obstacles; and
2. learn to choose that obstacle-free path which takes the robot in a minimum possible time.

Both these problems are somewhat similar to the growing-up (learning) process of a child. If a child is kept in a similar (albeit hypothetical) situation (that is, a child has to go from one corner of a room to another corner by avoiding a few moving objects), one of the probable approaches the child may follow is take each object at a time. When an object is very near to the child, he may either stop to let the object pass by or he may take a small detour so he avoids hitting the object. It is clear that when the child is taking a detour there is no particular angle by which he would turn when faced with a similar situation again. To avoid hitting an object, his objective is to deviate from his original path. This process of avoiding an object can be thought as if the child is using a rule of the following sort:

If an object is very near and is coming straight to me, then I turn right to my original path.

When such a situation happens, the most important thing for the child to do is to deviate from his path to avoid the imminent object. The exact angle of deviation is not that important. Thus, the angle of deviation can be imprecisely set, although an optimum angle of deviation can be computed using principles of physics and this computation will be extensive. Thus, it would make sense to use a fuzzy logic technique to find a suitable angle of deviation quickly than to use an exact angle calculated with unnecessary rigor. Let us now look at the second task.

The second task is to find an obstacle-free path which requires minimum possible time to reach from point \( S \) to \( G \). This task is similar to the above-mentioned child simile, but relates to the way an inexperienced and an experienced child will solve the same problem. An inexperienced child may take avoidance of each obstacle too seriously and deviate by a large angle each time he faces an obstacle. This way, this child may move away from the target point \( G \) and may finally reach \( G \) after traversing a long winding distance,
whereas an experienced child may deviate barely from each obstacle, thereby
taking the quickest route. If we think about how the experienced child has
learnt this trick, the answer is through experience of solving many such
problems in the past. If we assume again that the child uses rules to do the
task, the child has discovered (from experience) a set of efficient rules by
solving similar tasks in the past. This is precisely an optimization procedure
where an optimal set of rules are discovered which minimizes the travel time in
the presence of moving objects. We can simulate this learning process by using
an optimization algorithm – Genetic Algorithm – in training a robot to learn
to find an optimal set of rules by simulating its motion in a number of user-
defined scenarios.

Thus, the use of fuzzy logic technique helps in quickly determining imprecise
yet obstacle-free paths and the use of a genetic algorithm helps in learning an
optimal set of rules that a robot should use while navigating in presence of
moving obstacles. This process is illustrated in Fig. 1. A GA is used to create
the fuzzy knowledge base of a robot off-line. For on-line application, the robot
uses its optimal fuzzy rule base to find an obstacle-free path for a given input of
parameters depicting the state of moving obstacles and the state of the robot. It
is important to note that it would not be wise to solve both the above tasks:
(i) finding a deviation to avoid an obstacle and (ii) finding a complete obstacle-
free path which is shortest, independently. Both the problems are dependent on
each other; in fact, the child (in the simile above) also does not solve both
problems independently. We devise the following combined genetic-fuzzy
approach to solve the dynamic motion planning problem of a robot.

![Fig. 1. Genetic-fuzzy approach.](image-url)
2.1. Representation of a solution

A solution to the DMP problem is represented by a set of rules which a robot will use to navigate from point S to point G (Fig. 2). Each rule has three conditions: distance, angle, and relative velocity. The distance is the distance of the nearest obstacle forward from the robot. Four fuzzy values of distance are chosen: very near (VN), near (N), far (F), and very far (VF). Each of them is assumed to take a triangular membership function as shown in Fig. 3. The angle is the relative angle between the path joining the robot and the target point and the path to the nearest obstacle forward. The corresponding fuzzy values are left (L), ahead left (AL), ahead (A), ahead right (AR), and right (R). Each of them is also assumed to take a triangular membership function shown in Fig. 3. The relative velocity is the relative velocity vector of the nearest obstacle forward with respect to the robot. In our approach, we do not explicitly use this information as a fuzzy variable. Instead, we use a practical incremental procedure to eliminate its explicit consideration. In practice, the position and velocity of obstacles as a function of time may not be known a priori. However, the robot can find the position and velocity of each obstacle at a regular interval of time using sensors. Since at the end of each time step, the robot knows the position and relative velocity of each obstacle, the definition of the nearest obstacle forward can be modified by using the relative velocity information of obstacles. In such a case (Fig. 2), even if an obstacle $O_1$ is nearer compared to another obstacle $O_2$, and the relative velocity $\vec{v}_1$ of $O_1$ directs away from the robot’s path towards the target point G, whereas the relative velocity $\vec{v}_2$ of $O_2$ directs towards the robot (Position C), the obstacle $O_2$ is assumed to be the nearest obstacle forward. This practical consideration allows us to achieve two important tasks:

![Diagram](image-url)

Fig. 2. A schematic of condition (distance and angle) and action variables (deviation).
1. It eliminates the third condition variable (relative velocity) in the rule set. This reduces search space for finding the optimal rule base considerably.

2. It enables us to use an incremental approach, where the robot locates all obstacles at the end of a small time step. This makes the approach practical to be used in a real scenario.

The action variable is deviation of the robot from its path towards the target (Fig. 2). This variable is considered to have five fuzzy values: L, AL, A, AR, and R. The same triangular membership functions as those used for angle are used here. A typical rule will, thus, look like the following:

\[
\text{If distance is VN and angle is A, then deviation is AL.}
\]

With four choices for distance and five choices for angle, there could be a total of \(4 \times 5 = 20\) combinations of two different conditions possible. For each of these 20 combinations, there could be one value of the action variable. Thus, there are a total of \(20 \times 5 = 100\) different rules possible, but an arbitrary set from these 100 rules cannot be used to constitute a valid rule base. This is because for two rules having identical combinations of condition variables, there should be a unique value of the action variable. Thus, the maximum number of rules that may be present in a rule base is 20, each having a unique combination of condition variables. All 20 rules which are used in this study are shown in Table 1. We have assigned a particular value of the action variable for each combination of condition variables based on intuition. When an obstacle is very near and straight ahead, the robot deviates towards ahead-
All possible rules are shown

<table>
<thead>
<tr>
<th>angle</th>
<th>L</th>
<th>AL</th>
<th>A</th>
<th>AR</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>A</td>
<td>AR</td>
<td>AL</td>
<td>AL</td>
<td>A</td>
</tr>
<tr>
<td>N</td>
<td>AL</td>
<td>A</td>
<td>AL</td>
<td>A</td>
<td>AR</td>
</tr>
<tr>
<td>F</td>
<td>AL</td>
<td>A</td>
<td>AL</td>
<td>A</td>
<td>AR</td>
</tr>
<tr>
<td>VF</td>
<td>A</td>
<td>A</td>
<td>AL</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

left. However, when the obstacle is very near but on the left of the robot, the robot goes ahead. As the critical obstacle is away from the robot, it has a tendency to move ahead. This set of rule base is pretty good and we shall see later that an FLC with this rule base can navigate well in certain scenarios. However, currently we are also extending the genetic-fuzzy approach to adaptively find the best action variable for a particular combination of condition variables, thereby eliminating the need for such a user-defined rule base.

It is important to note that not all 20 rules are necessary for the robot to use during an obstacle avoidance. One of the tasks in this study is to find which (and how many) rules should be there in the rule base for the robot to find the quickest path between two points. We represent the presence of a rule by a 1 and the absence by a 0. Thus, a complete solution will have a 20-bit length string of 1 and 0. The value of the \( i \)th position along the string marks the presence or absence of the \( i \)th rule in the rule base. Thus, the following 20-bit string represents eight rules, as depicted in Table 2.

10011 01010 00010 01010

Thus, by using a 20-bit string we can represent any combination of rules in the rule set.

<table>
<thead>
<tr>
<th>angle</th>
<th>L</th>
<th>AL</th>
<th>A</th>
<th>AR</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>A</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VF</td>
<td>A</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.2. Evaluating a solution

A solution represented by a 20-bit string defines a set of rules which the robot uses to navigate through moving obstacles and attempts to reach point G (destination) from the point S (starting point). It is clear that some solutions will enable the robot to achieve the task quickly, whereas some will require longer time. In fact, not all solutions will necessarily take the robot to its destination within a fixed time ($T_f$). Thus, a solution is evaluated by calculating the total time ($T$) the robot takes to reach the destination. If $T < T_f$, the solution is assigned a value of $f = T$. If, however, the time of travel equals or exceeds $T_f$, the robot is halted at its current position and the Euclidean distance $d_{cm}$ between the halted position and the destination point is calculated. A value of $f = T_s + d_{cm}/V$ (where $V$ is the maximum velocity) is assigned to this solution.

It is important to note that the above optimization process will largely depend on the particular scenario (the placement and motion of obstacles) used for the motion planning problem. Thus, in order to find a genetic–fuzzy solution which is fairly generic to a wide variety of scenarios, we have evaluated a solution in $Q$ different scenarios and the average travel time ($T$) of all $Q$ scenarios is used as the actual objective function value, which is minimized during the optimization process.

Now, we shall discuss some details which will be necessary to calculate the actual travel time $T$. As mentioned earlier, the robot's total path is a collection of a number of small straight line paths traveled for a constant time $\Delta T$ in each step. To make the matter as practical as possible, we have assumed that the robot starts from zero velocity and accelerates during the first quarter of the time $\Delta T$ and then maintains a constant velocity for the next one-half of $\Delta T$ and decelerates to zero velocity during the remaining quarter of the total time $\Delta T$ (Fig. 4). The acceleration and deceleration rates are assumed to be equal. If this rate is $a$, then the total distance covered during the small time step $\Delta T$ is $3a\Delta T^2/16$. At the end of the constant velocity travel (the point marked as E in Fig. 4) each time step ($\Delta T$), the robot senses the position and velocity of each obstacle and decides whether to continue moving in the same direction or to deviate from its path. This is achieved by first determining the predicted position of each obstacle, as follows:

$$P_{predicted} = P_{present} + (P_{present} - P_{previous}).$$

The predicted position is the linearly extrapolated position of an obstacle from its current position $P_{present}$ along the path formed by joining the previous $P_{previous}$ and present position. Thereafter, the nearest obstacle forward is determined based on $P_{predicted}$ values of all obstacles and the fuzzy logic technique is applied to find the obstacle-free direction using the rule base dictated by the corresponding 20-bit string. The details of this fuzzy logic technique are discussed in...
Appendix A. If the robot has to change its path, its velocity is reduced to zero at the end of the time step; otherwise the robot does not decelerate and continues in the same direction with the same velocity $a\Delta T/4$. It is interesting to note that when the latter case happens (the robot does not change its course) in two consecutive time steps, there is a saving of $\Delta T/4$ s in travel time per such occasion. Continuing in this fashion, when the robot comes closer to the destination and there is no critical obstacle in its way, the robot reaches its destination by starting its deceleration from a distance of $a\Delta T^2/32$. Overall time of travel ($T$) is then calculated by summing all intermediate time steps needed for the robot to reach its destination. This approach of robot navigation can be easily incorporated in a real-world scenario.

In all the simulations here, we have chosen $\Delta T = 4$ s and $a = 1$ m/s$^2$. These values make the velocity of the robot in the middle portion of each time step equal to 1 m/s.

2.3. Optimizing for minimum-time solution

The GA technique is used to find the optimal or a near-optimal obstacle-free path. The GA operators and the working principle of the GA are described in Appendix B. In short, the GA begins its search by randomly creating a number of solutions represented in binary-coded strings. Since solutions in this prob-
lem are represented in a 20-bit string, this problem is ideal to be solved using a GA. Each solution in the population is then evaluated to assign a fitness value. In our study, we have assigned a fitness value as follows. Each solution (20-bit string) is evaluated to calculate a function value \( f_i \) for \( Q \) different scenarios \( (i = 1, 2, \ldots, Q) \) of moving obstacles, starting positions and target positions. The fitness to the string is assigned as \( FS = \frac{\sum_{i=1}^{Q} f_i}{Q} \). Since the objective is to minimize the overall travel time, we use a GA to find a string which corresponds to the minimum fitness value.

After each solution in the population is evaluated and fitness is assigned, the population is modified by using three operators—tournament selection, one-point crossover, and bit-wise mutation (discussed in Appendix B). The tournament selection compares two solutions at a time from the population and chooses the solution having the smaller fitness value. Crossover operator exchanges bit information between two such strings obtained after tournament selection and creates two new strings (or solutions). The mutation operator compliments bit values at arbitrary places in a string to create a new string. After a new population of solutions is created, each of them is evaluated again to find a fitness value and all three operators are applied again. One iteration of these three operators followed by the evaluation procedure is called a generation. Generations proceed until a termination criterion is satisfied. In our study, we continue until a pre-specified number of generations have elapsed.

3. Results

In this section, we present simulation results of the motion planning problem of a mobile robot in a systematic manner. There are five different approaches studied here.

**Approach 1: Author-defined fuzzy-logic controller.** There are three components of the FLC knowledge base, namely scaling factors, membership functions, and rule set. It is important to note that both the scaling factors and membership functions taken together will represent the semantics of the symbols used by the FLC and the rule set represents the syntactic mapping among the symbols. In this approach, a fixed set of 20 rules (Table 1) and author-defined membership functions (Fig. 3) are used. Fig. 3 shows that \( b_1 \) is set to 2.3 corresponding to a scaling factor for distance \( SF_d = 6.9 \) and \( b_2 \) is set to 45 corresponding to a scaling factor for angle (deviation) \( SF_a = 90.0 \). No optimization method is used to find optimal knowledge base of the FLC.

**Approach 2: Tuning scaling factors of the state variables alone.** A set of author-defined rules base is assumed (Table 1) and the tuning of scaling factors for condition and action variables is done keeping the relative spacing of membership distributions constant. The shape of the membership function is
assumed to be triangular. The base coordinates of the membership functions are considered as variables. All 20 possible rules shown in Table 1 are used. Thus, the objective of this study is to tune the scaling factors for condition and action variables which, along with 20 rules presented in Table 1, will result in the obstacle-free path taking the smallest possible travel time between any two points. One parameter for each of the action variables is kept as the decision variable. The bases $b_1$ and $b_2$ (refer Fig. 3) are coded in 10-bit substrings each, thereby making a GA string equal to 20 bits. The base $b_1$ is decoded in the range (1.0, 4.0) m and the base $b_2$ is decoded in the range (25.0°, 60.0°). Thus, we are actually tuning the scaling factors, while leaving the term-set proportionally spaced. In all simulations here, the membership function distribution for deviation is kept the same as that in angle.

**Approach 3:** Tuning rule base alone. The membership functions are defined by the authors, as shown in Fig. 3. The rule base is optimized in this study. The maximum number of possible rules is 20. Here, the GA string is a 20-bit string (of 1 and 0 denoting presence or absence of rules) as illustrated earlier in Table 2. But the objective of this study is to search for those rules (and how many) from these 20 that will result in an obstacle-free path taking the smallest possible travel time between any two points.

**Approach 4:** Tuning scaling factors of the state variables and rule base in stages. In this study, the optimized solutions obtained in Approaches 2 and 3 are combined together. Thus, the membership function used here is the same as that found in Approach 2 and the rule base is the same as that found in Approach 3.

**Approach 5:** Tuning scaling factors of the state variables and rule base simultaneously. In this study, both optimization of finding optimized base width of triangular membership functions and finding an optimized rule base are achieved simultaneously. Here, a GA string is a 40-bit string with the first 20 bits denoting the presence or absence of 20 possible rules, the next 10 bits representing the base $b_1$ and the final 10 bits representing the base $b_2$. Lower and upper bounds of these latter two variables are kept the same as that in Approach 2. The objective of this study is to find the size of triangular membership functions for condition and action variables, and the rule base which will result in the obstacle-free path taking the smallest possible travel time between any two points. This optimization is a more practical approach, since both optimizations are performed simultaneously.

In order to investigate the efficacy of the proposed approaches, we first study a scenario with only three moving obstacles. Thereafter, we present results for a more complicated scenario having eight obstacles. In all runs of the proposed approach, we use binary tournament selection (with replacement), the single-point crossover operator with a probability $p_c$ of 0.98 and the bit-wise mutation operator with a probability $p_m$ of 0.02. A maximum number of generations equal to 40 is used. In every case, a population size of 100 is used. For a brief
description of these operators, refer to Appendix B. In all approaches, \( Q = 10 \) different author-defined scenarios are used to evaluate a solution.

### 3.1. Three-obstacle problem

In this scenario, there are three obstacles moving independently in a grid of 16 \( \times \) 14 m\(^2\) in a 2-D space. The robot has to travel from point S to point G by avoiding all three obstacles. The results of all five proposed approaches are compared. The author-defined FLC has all 20 rules (Table 1) and membership functions as shown in Fig. 3. The traveling distance and time are presented for all five approaches (Approaches 1–5) in Table 3. In this table, three (out of 10) scenarios used during the optimization process are shown in the first three rows. The subsequent three rows show three new (and different) scenarios which were not used during the optimization process. These three experiments show how the robot behaves in unknown scenarios. In most cases, Approach 2 is better than Approach 1 (with author-defined FLC). The fact that in some cases the performance in terms of time is the same reveals that the author-defined rule base contains good rules to find shorter paths in some scenarios. The table also shows that, in general, Approaches 3–5 have found better paths (in terms of travel time) than the other two approaches. Importantly, the solution obtained in Approach 1 (with no optimization) is not better than results obtained in Approaches 2–5.

The paths obtained in all five approaches for the test scenario 4 (presented in Table 3) are shown in Fig. 5. The corresponding travel distance and time are also shown in Table 3. It is clear that Approaches 3–5 have obtained a better path than the other two methods.

The optimized rule base obtained using Approaches 3–5 are shown in Tables 4 and 5. These tables show that both rule bases are quite different from each other. There are only two rules that are common between the two rule bases. The optimized membership functions obtained using Approaches 2 and

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Approach 1</th>
<th>Approach 2</th>
<th>Approach 3</th>
<th>Approach 4</th>
<th>Approach 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D (m)</td>
<td>T (s)</td>
<td>D (m)</td>
<td>T (s)</td>
<td>D (m)</td>
<td>T (s)</td>
</tr>
<tr>
<td>1</td>
<td>16.179</td>
<td>20.0</td>
<td>15.560</td>
<td>20.0</td>
<td>14.610</td>
</tr>
<tr>
<td>2</td>
<td>15.568</td>
<td>20.0</td>
<td>14.974</td>
<td>17.0</td>
<td>14.318</td>
</tr>
<tr>
<td>3</td>
<td>17.870</td>
<td>22.0</td>
<td>16.560</td>
<td>22.0</td>
<td>14.804</td>
</tr>
<tr>
<td>4</td>
<td>18.920</td>
<td>26.0</td>
<td>18.144</td>
<td>23.0</td>
<td>14.675</td>
</tr>
<tr>
<td>6</td>
<td>17.091</td>
<td>22.0</td>
<td>16.308</td>
<td>22.0</td>
<td>14.594</td>
</tr>
</tbody>
</table>
Fig. 5. Optimized path found by all five approaches for the three-obstacle problem.

Table 4
Optimized rule base (having eight rules only) obtained using Approaches 3 and 4 for the three-obstacle problem

<table>
<thead>
<tr>
<th>angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
</tr>
<tr>
<td>VN</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>V F</td>
</tr>
</tbody>
</table>

5 are shown in Figs. 6 and 7, respectively. Fig. 6 shows that the scaling factors for distance (\(SF_d\)) and angle (\(SF_a\)) are found to be 9.0 and 57.6, respectively (using Approach 2). Similarly, \(SF_d\) and \(SF_a\) are determined (using Approach 5) to be 3.0 and 65.2, respectively, as shown in Fig. 7. These figures show that
Table 5
Optimized rule base (having eight rules only) obtained using Approach 5 for the three-obstacle problem

<table>
<thead>
<tr>
<th>Angle</th>
<th>L</th>
<th>AL</th>
<th>A</th>
<th>AR</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>AL</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>A</td>
<td>AL</td>
<td>A</td>
<td>AR</td>
<td></td>
</tr>
<tr>
<td>VF</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

although the optimized membership functions for angle and deviation are more or less similar, the optimized membership functions for distance are very different in both cases. In Approach 5, the membership functions are squeezed so that in most situations, distances will appear as far (F) or very far (VF). The corresponding rule base is adjusted so that there are more rules specifying rules related to far or very far distances (Table 5). It is interesting to note that although in some scenarios the same travel time is obtained, different approaches...
have used different combinations of rules and membership functions. Table 5 also shows that no rule exists in the rule base when the robot is very near (VN) to an obstacle. This is because of the small number of obstacles considered in the scenarios. Since there are only three obstacles, Approach 5 adjusts its membership functions (Fig. 7) so that the robot never comes very near (in the meaning of VN) to any obstacle. Since the critical obstacle is always found away from the robot, most of the rules specifying an action for far (F) or very far (VF) away obstacles are preferred.

3.2. Eight-obstacle problem

We now apply all five approaches to eight-obstacle problems (in a grid of 20 x 24 m$^2$). The optimized travel distance and time for Approaches 2–5 are presented in Table 6 along with that obtained for Approach 1 (once again, no optimization is used in this case, but the same 20 rules and membership function chosen in the three-obstacle problem are used). The first three rows in the table show the performance of all approaches on scenarios that were used during the optimization process and the last three rows show their performance on new test (unseen) scenarios. The table shows that in all cases, Approaches 3–5 have performed better than the other two approaches. Once again, in most cases Approach 2 is better than Approach 1.
Table 6
Travel distance $D$ and time $T$ obtained by five approaches for the eight-obstacle problem

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Approach 1</th>
<th>Approach 2</th>
<th>Approach 3</th>
<th>Approach 4</th>
<th>Approach 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$D$ (m)</td>
<td>$T$ (s)</td>
<td>$D$ (m)</td>
<td>$T$ (s)</td>
<td>$D$ (m)</td>
</tr>
<tr>
<td>1</td>
<td>27.203</td>
<td>32.0</td>
<td>26.488</td>
<td>29.0</td>
<td>26.155</td>
</tr>
<tr>
<td>2</td>
<td>26.957</td>
<td>29.0</td>
<td>26.310</td>
<td>29.0</td>
<td>26.026</td>
</tr>
<tr>
<td>3</td>
<td>30.228</td>
<td>40.0</td>
<td>29.017</td>
<td>37.0</td>
<td>26.660</td>
</tr>
<tr>
<td>4</td>
<td>32.717</td>
<td>43.0</td>
<td>27.391</td>
<td>31.0</td>
<td>26.243</td>
</tr>
<tr>
<td>5</td>
<td>30.296</td>
<td>41.0</td>
<td>29.491</td>
<td>37.0</td>
<td>26.543</td>
</tr>
<tr>
<td>6</td>
<td>32.386</td>
<td>42.0</td>
<td>27.287</td>
<td>34.0</td>
<td>26.574</td>
</tr>
</tbody>
</table>

Paths obtained using all five approaches for scenario 4 are shown in Fig. 8. It is clear that the paths obtained by Approaches 3–5 are shorter and quicker than those obtained by Approaches 1 and 2.

The optimized rule bases obtained using Approaches 3–5 are shown in Tables 3 and 6. The optimized membership functions obtained using Approaches 2 and 5 are shown in Figs. 9 and 10, respectively. It is to be noted that a GA has found (using Approach 2) the values of scaling factors, namely $SF_d$ and $SF_s$, to be 5.4 and 53.6, respectively, as shown in Fig. 9. Similarly, Fig. 10 shows that the values of scaling factors, namely $SF_d$ and $SF_s$, are to be 5.7 and 91.6, respectively, as obtained by Approach 5. Here, Approach 5 (simultaneous tuning of rules and scaling factors) has elongated the base width of the triangle (representing membership function distribution) so that distribution of relative angle is uniform in the range of $(-90^\circ, 90^\circ)$. Because only 10 scenarios are considered during the optimization process, it could have been that in most cases the critical obstacles come to the left of the robot, thereby causing more rules specifying $L$ or $A_L$ to appear in the optimized rule base. By considering more scenarios during the optimization process, such bias can be avoided and equal number of rules specifying left and right considerations can be obtained.

In Tables 3 and 6, it can be observed that Approach 3 (tuning of rule base only) has resulted in a much quicker path than Approach 2 (tuning scaling factors of the state variables only). This is because finding a good set of rules is more important for the robot than finding a good set of membership functions. Thus, the optimization of rule base is a rough-tuning process and the tuning of base width of the triangle representing the membership function distribution is a fine-tuning process. Among both the tables, in only one case (Scenario 6 in Table 6) the tuning of scaling factors for an optimized rule base has improved the solution slightly (Approach 4). In all other cases, the optimized solutions are already obtained during the optimization of rule-base only and tuning of scaling factors did not improve the solution any further.
Although the performance of Approaches 3–5 are more or less similar, we would like to highlight the point that Approach 5 is the most practical approach and also the most complex problem from the point of view of optimization. Approach 4 is a two-stage optimization process, and hence requires more computational effort. The similarity in the performances of Approaches 3 and 5 reveals that optimizing rule base has a significant effect and the tuning of scaling factors is only a secondary matter. Since the membership functions used in Approach 3 are developed by the authors and are reasonably good in these two problems, the performance of Approach 3 is good. However, for more complicated problems, we recommend using Approach 5, since it optimizes the knowledge base of an FLC needed for solving a problem. For more number of
Table 7
Optimized rule base (having seven rules only) obtained using Approaches 3 and 4 for the eight-obstacle problem

<table>
<thead>
<tr>
<th>distance</th>
<th>angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>AR</td>
</tr>
<tr>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>F</td>
<td>A</td>
</tr>
<tr>
<td>VF</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 8
Optimized rule base (having eight rules only) obtained using Approach 5 for the eight-obstacle problem

<table>
<thead>
<tr>
<th>distance</th>
<th>angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>A</td>
</tr>
<tr>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>F</td>
<td>AL</td>
</tr>
<tr>
<td>VF</td>
<td>A</td>
</tr>
</tbody>
</table>

obstacles and more complicated scenarios, a good guess of a rule base may not be possible. The methods of this study can be easily (and in a practical sense) used to obtain an optimal or a near-optimal rule base.

4. Conclusions

In this study, learning capability of a genetic-fuzzy approach has been demonstrated by finding optimal/near-optimal FLCs for solving the motion planning problem of a mobile robot. In the genetic-fuzzy approach, obstacle-free paths are found locally by using the fuzzy logic technique, where optimal scaling factors (determining the base width of the triangle) for condition and action variables and an optimal rule base are found using genetic algorithms. Based on this basic approach, four different approaches are developed and compared with an author-defined (non-optimized) fuzzy-logic controller (FLC).
Fig. 9. The optimized membership function obtained using Approach 2 for the eight-obstacle problem.

Fig. 10. The optimized membership function obtained using Approach 5 for the eight-obstacle problem.
In three-obstacle and eight-obstacle problems, the merit of the using genetic-fuzzy approach is demonstrated. In all cases, the genetic-fuzzy approaches have found obstacle-free paths which take shorter time to travel between two points. The genetic-fuzzy approach developed here is also highly practical to be used in a real-world situation. One of the major advantages of the proposed method is that the optimization is performed off-line and an optimal rule base is obtained before-hand. Robots can then use this optimal rule base to navigate in the presence of unseen scenarios in an optimal or a near-optimal manner. This paper shows how such a rule base can be achieved. However, to create a truly optimal rule base, we recommend that the approach of this paper be used with as many scenarios as possible. More scenarios during the optimization process will test each and every rule in the rule base for its utility better and will lead to a truly optimal rule base.

From this study, the following conclusions can be made:

1. The proposed algorithm is able to solve the dynamic motion planning problems effectively. Simulation results show that GA-designed FLCs always perform better than an author-defined FLC. It is obvious because author-defined knowledge base for an FLC may not be optimum always.

2. The performance of FLC depends on the selection of rule base and membership function distribution. As both the rule base as well as the shape of the membership function distribution are interdependent, it is more practical to consider the GA-designed FLC where both rule base as well as scaling factors of the state variables have been optimized simultaneously.

3. As optimization is done off-line, the proposed algorithm is computationally tractable and on-line implementation is easy.

4. Rule-base optimization involves the problem of dealing with discrete variables and GA is a powerful tool for solving this type of problems.

5. It is also observed that optimizing rule base of an FLC is a rough-tuning process whereas optimizing scaling factors of the state variables (which indicates the base width of triangular membership functions) is a fine-tuning process.

This study opens a number of useful extensions which need immediate attention:

More obstacles and training scenarios: As discussed earlier, the optimized rule base largely depends on the number of obstacles and the corresponding number of scenarios used during the optimization process. Using more obstacles and scenarios during the optimization process will test various rules for their worth to be included in the optimized rule base better. The approach of this study can be easily extended to obtain a better rule base for more complicated scenarios.

Varying time step: It is intuitive that the optimized travel time will depend on the incremental time $\Delta T$ used in each step. Using a large value of $\Delta T$ reduces the steps necessary to reach the target, but increases the chance of
collision with obstacles. On the other hand, using a small value of $\Delta T$ reduces the chance of collision, but increases the number of time steps the robot needs to take measure of the position and velocity of moving obstacles. Since this should ideally depend on the location and motion of obstacles, the time step $\Delta T$ can also be kept as an action variable in the rule base. This way, the FLC will decide how much incremental time to leap every time the robot takes a decision about its deviation.

Eliminating author-defined rule base: In this study, we have developed a set of 20 rules (based on intuition) for each combination of distance and angle option. Since each such combination can cause five different options for the action variable (deviation), it will be fair to have a slightly different GA coding of representing a solution and eliminate the use of any author-defined information. One of the ways to represent a solution is shown in the following:

201500130...4

The above string has 20 positions (one of each combination) and each position can take one of six values (0–5). The value 0 means the absence of that combination (or rule). The value 1 means that the first option (L) of the action variable is associated with the combination of condition variables, and so on. This way every solution represented by a 20-position vector signifies a valid rule base. Better crossover and mutation operators need to be devised to make an efficient GA search.

Acknowledgements

The first author acknowledges financial support from Quality Improvement Program of All India Council of Technical Education (AICTE), Government of India.

Appendix A. A fuzzy reasoning approach

The concept of fuzzy set theory was first introduced by Zadeh [51], in the year 1965. A fuzzy logic controller (FLC) is one of the most successful applications of fuzzy set theory. A fuzzy rule-base system works depending on the concept of membership function distribution [51,52]. The fuzzy reasoning process is illustrated in Fig. 11. For simplicity, we assume that there are two fuzzy control rules as follows:

RULE 1: IF sl is A1 and s2 is B1 THEN f is C1
RULE 2: IF sl is A2 and s2 is B2 THEN f is C2.
If $s_1^*$ and $s_2^*$ are the inputs for fuzzy variables $s_1$ and $s_2$ and if $u_{A_1}$ and $u_{B_1}$ are the membership functions for $A$ and $B$, respectively, then the grade of membership of $s_1^*$ in $A_1$ and the grade of membership of $s_2^*$ in $B_1$ are represented by $u_{A_1}(s_1^*)$ and $u_{B_1}(s_2^*)$, respectively for rule 1. Similarly, for rule 2, $u_{A_2}(s_1^*)$ and $u_{B_2}(s_2^*)$ are used for the grades of membership. The firing strengths of the first and second rules are calculated as follows:

$$\alpha_1 = \min (u_{A_1}(s_1^*), u_{B_1}(s_2^*)), \quad (A.1)$$

$$\alpha_2 = \min (u_{A_2}(s_1^*), u_{B_2}(s_2^*)). \quad (A.2)$$

The membership function of the combined control action $C$ is given by

$$u_C(f) = \max (u_{C_1}(f), u_{C_2}(f)). \quad (A.3)$$

The center of area method is employed for defuzzification and the method can be represented by

$$U_f = \frac{\sum_{j=1}^{p} A(x_j) \times f_j}{\sum_{j=1}^{p} A(x_j)}, \quad (A.4)$$

where $U_f$ is the output of the controller, $A(x_j)$ represents the firing area of the $j$-th rule, $p$ is the total number of the firing rules, and $f_j$ represents the centroid of a membership function. For details of the above method, see Ref. [53].
Appendix B. Binary-coded genetic algorithms

Genetic algorithms (GAS) are population-based search and optimization algorithms which mimic the principles of natural genetics and natural selection [54-56]. GAS are very different from classical search and optimization methods. There are several versions of GAS, such as binary-coded GA, real-coded GA, messy GA and others, which are used to solve different kinds of search and optimization problems.

In a binary-coded GA, all problem variables are coded in finite-length binary strings. For example, three variables $x_1$, $x_2$ and $x_3$ can be represented by 4, 3, and 5 bit substrings as follows:

\[
\begin{bmatrix}
0111 \\
101 \\
00110
\end{bmatrix}
\]

$x_1$, $x_2$, $x_3$

The total string length ($L$) of a solution is then 12. The length of each substring is determined by the required accuracy in each variable. In order to retrieve the corresponding variable values, the following decoding scheme is usually used:

\[
x_t = x_{\text{lower}}^t + \frac{x_{\text{upper}}^t - x_{\text{lower}}^t}{2^L - 1} \times d,
\]

where $d$ indicates the decoded value of the string. Once the values of the variables are known, the objective function value can be determined. This objective function value is treated as the fitness value of the string.

The working principle of GA is shown in the form of a flowchart (Fig. 12). The operation of a GA begins with a population of random strings representing design variables. Thereafter, each string is evaluated to find the fitness value. The population is then operated by three main operators, namely reproduction, crossover and mutation, to create a new population. The new population is further evaluated and tested for termination.

Reproduction operator selects good strings from the population using fitness information. In a binary tournament selection, two strings are chosen at random from the population and the best string is selected and copied in an intermediate population, called mating pool. This process continues by comparing two strings at a time till the mating pool has the same size as the original population size. Thus, this operator emphasizes the good strings of a population and makes duplicate copies of them in a mating pool.

In the crossover, new strings are created by exchanging information among strings of the mating pool. In a single-point crossover, two strings are chosen from the mating pool, and also a crossing site is chosen along the string. Thereafter, all bits on the right side of the crossing site are exchanged between both the strings. This operator allows partial information to be exchanged between two good strings found using the reproduction operator. The cross-
BEGIN

Initialize a population of strings
Gen = 0

Gen > max_gen?

No

Assign fitness to all strings in the population

REPRODUCTION

CROSSOVER

MUTATION

Yes

END

Gen = Gen + 1

Fig. 12. Schematic of the working principle of a GA.

over operator is mainly responsible for the search of new strings. In order to reduce the chance of destructing already-found good strings, crossover is usually performed with a probability $p_c$ slightly smaller than one.

Mutation operator changes 1 to 0 and vice versa with a small probability, $p_m$. Mutation is used for achieving a local change around the current solution. In short, reproduction operator selects good strings and crossover operator recombines two good strings to hopefully create better strings. The mutation operator alters a string locally to create a new string.

The above operation of a GA is very similar to the evolutionary principle. If good strings are created by crossover and mutation operators, reproduction emphasizes them and they have more chance of getting mated with other good solutions. On the other hand, if crossover and mutation creates bad strings, reproduction ruthlessly eliminates them from further processing. The string copying and substring exchange operations in GAs may seem at first to be
random operators, a careful thought will prevail that although random numbers are used extensively in a GA, the search is not a random search. Instead, the randomness in the search operators provide the necessary stochasticities for a GA not to get stuck at suboptimal solutions.

References


