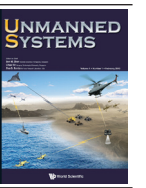
Unmanned Systems, Vol. 6, No. 2 (2018) 95–118

DOI: 10.1142/S2301385018400022

UNMANNED SYSTEMS
http://www.worldscientific.com/worldscinet/us



Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study

Mohammadreza Radmanesh*, Manish Kumar†,
Paul H. Guentert, Mohammad Sarim

Department of Mechanical and Materials Engineering,
University of Cincinnati, Cincinnati, OH 45221, USA

Unmanned aerial vehicles (UAVs) have recently attracted the attention of researchers due to their numerous potential civilian applications. However, current robot navigation technologies need further development for efficient application to various scenarios. One key issue is the "Sense and Avoid" capability, currently of immense interest to researchers. Such a capability is required for safe operation of UAVs in civilian domain. For autonomous decision making and control of UAVs, several path-planning and navigation algorithms have been proposed. This is a challenging task to be carried out in a 3D environment, especially while accounting for sensor noise, uncertainties in operating conditions, and real-time applicability. Heuristic and non-heuristic or exact techniques are the two solution methodologies that categorize path-planning algorithms. The aim of this paper is to carry out a comprehensive and comparative study of existing UAV path-planning algorithms for both methods. Three different obstacle scenarios test the performance of each algorithm. We have compared the computational time and solution optimality, and tested each algorithm with variations in the availability of global and local obstacle information.

Keywords: UAV; path-planning; obstacle avoidance; sense and avoid; algorithm efficiency.

US

1. Background and Overview

There are many popular path-planning methods for unmanned aerial vehicles (UAVs) to navigate in a three-dimensional domain filled with obstacles that formulate path-planning as an optimization problem. Two groups delineate the algorithms used to solve these optimization problems: heuristic and non-heuristic (or exact) methods. Heuristic methods trade optimality (or exactness) of solution for computational time-efficiency. Non-heuristic methods, on the other hand, use mathematical principles for obtaining the solutions which are often computationally expensive. The common process for path-planning algorithms starts by breaking down the detected area or map into computational domains using techniques such as area tessellation, matrix decomposition, or a combination of these two. These data structures help in the generation of possible UAV trajectories. After trajectory generation, due to the resolution of the computational domain, the generated trajectories require a process to smooth the path that the UAV can travel [1].

The paper carries out an evaluation of popular path-planning and obstacle avoidance algorithms available in the literature, applied to UAVs. The paper fills the gap in the literature by carrying out this extensive comparison while providing implementation details. This paper also considers general characteristics and requirements pertaining to UAVs which are typically modeled via velocity and acceleration constraints. Comparison of the algorithms is carried out with respect to the optimality and computational complexity. Once these properties are identified, the analysis presented in this paper will help in choosing an algorithm based on the requirements called for by a particular application. It may be noted that the comparison provided here is primarily intended to give the readers an overview of different algorithms and how they can be implemented and evaluated. Arriving at a generalized conclusion about

Received 24 February 2017; Accepted 1 February 2018; Published 8 June 2018. This paper was recommended for publication in revised form by editorial board member, Kai-Yew Lum.
Email Addresses: *Radmanma@mail.uc.edu, †manish.kumar@uc.edu

the best algorithm is elusive since the implementation of algorithms, constraints and environment plays a huge role in their overall performance. Before comparing the UAV path-planning algorithms for different scenarios, we start with a comprehensive overview of the algorithms available in the literature.

1.1. *Problem description*

Path-planning problem belongs to a class of non-deterministic polynomial-time (NP) hard problems which is usually solved for realistic problems by making some assumptions and using heuristics to reduce the complexity to that of polynomial time problems [1]. The following studies cover the multitude of different motion planning algorithms for robots. In [2–4], algorithms focus on polygonal obstacle field representations. Reference [5] provides a survey of all the mathematical improvements in the field of motion planning. Reference [6] considers the dynamic environment for robot path-planning and provides information on computational bounds for limited cases. More recently, [7,8] studied sensor-based path-planning and probabilistic path-planning methods. Reference [8] emphasizes the implementation of these methods in ground vehicles. The following works provide a comprehensive study of most of the available methods — Ref. [9] offers a review of all the applicable path-planning algorithms for different vehicles with an emphasis on ground vehicles, Ref. [10] presents the first comprehensive survey carried out on heuristic methods. Surveys for UAV path-planning have recently appeared in the literature and, among those, Ref. [11] provides a literature review for implementing path-planning for autonomous vehicles. Reference [1] specifically focuses on path-planning of UAVs while [12] focuses on progress made in path-planning and obstacle avoidance of Autonomous Underwater Vehicles (AUVs). The analysis structure for this paper attempts to follow a similar outline demonstrated in the literature review resources. The introduction covers a general overview of the path-planning problems solved by algorithms belonging to both heuristic and non-heuristic (exact) methods. The different methods are then explored from existing surveys by providing implementation details on a few of the popular algorithms and comparing their performances in three separate scenarios. For each of the algorithms, following information is provided: (i) technical idea of the algorithm, (ii) successful applications in UAV path-planning, (iii) pseudo code for the algorithm, (iv) potential issues with the algorithm, and (v) algorithm complexity.

The overall schematic of the problem considered in this paper is shown in Fig. 1. Here, it is assumed that the environment is tessellated into a number of regular grids and the objective of the path-planning algorithm is to find the collision-free, shortest path from the initial point to the goal point.

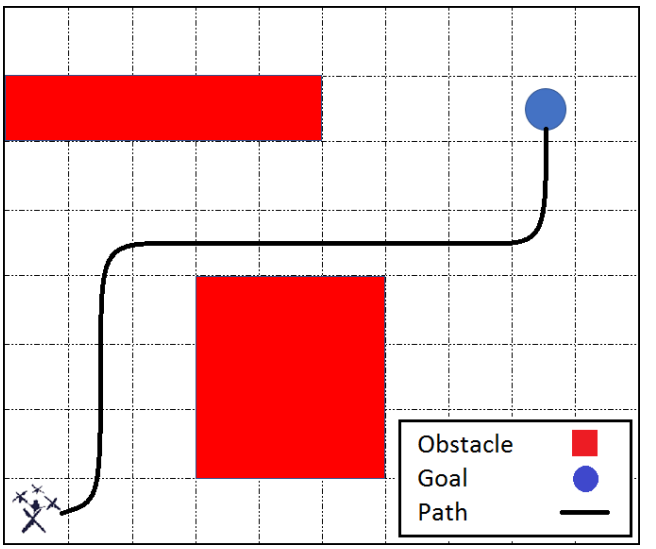


Fig. 1. Problem scheme in this paper.

1.2. *Basic definitions*

The terminology presented in [1, 5] are used to describe the path-planning algorithms in this paper. A brief summary of the terms and definitions are as follows:

World space is a space where the vehicle exists.

Configuration is a vector containing the parameters required for defining the positional state of the vehicle which usually consists of three position coordinates and three orientation coordinates of an UAV.

Configuration space or C-space is the set of all possible configurations of the vehicle.

State space is the set of all possible states of the system. Depending on the UAV dynamics, most elements of the state space are not relevant for the path-planning algorithm and usually are a subset of the possible state space (or higher-order derivatives of the state-space) that is considered for obtaining a solution [13].

Degrees of freedom are the minimum number of points or variables that represent a state or configuration.

Obstacle space is the physical space occupied by obstacles. Free space is the physical space which is not occupied by obstacles.

Path is the curve, in physical space, traced by the vehicle.

Trajectory also includes the information about the time unlike path that only represents the positional information about the curve [1].

Motion planning, also referred to as path-planning or trajectory planning, comprises determination of a path from the present state called the initial state to the final state called the goal state.

Local goal refers to the intermediate goal that the UAV tries to achieve in order to transition from the initial state to the goal state. This involves solving a number of sub-problems to obtain the solution of the general overall problem. The overall objective is finding the path or trajectory for navigating the UAV to the global goal.

Pop-up threat is referred to as the obstacles which are not known to the UAV *a priori*.

Safety margin or clearance is the minimum possible distance value that the path or trajectory can have with the obstacle.

1.3. *Algorithm complexity and performance*

In this paper, two types of analyses are used to compare the path-planning algorithms. The first analysis determines the computational resources required to run the algorithm. Time complexity describes the efficiency or run time of an algorithm. It relates the input length (or size of the problem that depends upon the number of decision variables) to the number of steps. Space complexity is the storage or memory requirements of a path-planning algorithm [14]. Algorithm analysis, an important part of the broader computational complexity theory, provides theoretical estimates of the required algorithmic resources to solve a given computational problem. These estimates serve as the criteria for determining the efficiency of the algorithms [15].

In theoretical analysis of algorithms, the estimate of complexity uses asymptotic analysis. O, Ω, and Θ notations indicate the complexity of an algorithm. O indicates that there exists an upper bound for the algorithm which restricts the number of computations needed for the algorithm. Ω denotes that there exists a lower bound for the algorithm and Θ represents that there exist a lower and upper bound for the algorithm. Computing the exact measures of efficiency, oftentimes, is quite challenging. This is because of the fact that the algorithm complexity highly depends on the implementation of the algorithm in actual programming codes. Also, the total number of computations often depend on the scenarios on which the algorithm is applied. This computation usually requires certain assumptions concerning an algorithm's implementation such as the operations used and their computational cost to the algorithm called model of computation [16].

Operational and computational aspects are among the criteria for judging whether an algorithm is efficient. Algorithms with lower computational complexity perform better in real time, which result in faster solution updates. For all the algorithms presented in this paper, the objective is to generate a trajectory that reaches the goal position while maintaining a safe distance from the obstacles.

Since solving for the optimal flight path of a UAV is an NP-Hard problem and often non-convex in nature, the problem is computationally expensive to solve. The discretization of the problem in space and time allows path-planning algorithms to obtain approximate optimal solutions in a more computationally efficient manner. However, operational requirements determine practical implementations of approximate solutions. This practicality dictates the trade-off between the optimality and the computational efficiency of the solution. Another approach to path-planning characterizes algorithms as informed or uninformed path-planning solutions. Informed path-planning involves heuristic functions to reduce the search space [17]. The informed search process typically results in faster but less optimal solutions. On the other hand, path-planning algorithms that do not use any heuristic functions are categorized as uninformed or *blind*. In most cases, the resulting path is based on the search in all directions from the current node [18].

Both informed and uninformed path-planning algorithms use methods such as iterative-deepening search [19], boundary search [20], bidirectional searches [21], and multi-goal searches [18, 22].

2. Potential Field Algorithm and Vector Force Field

One popular method used in path-planning applications is the Potential Field Algorithm. The Potential Field approach assigns a value calculated via an artificial potential function to every point in the world and simulates the reaction of the vehicle to the potential field as it navigates towards the minimum potential. The goal point has the lowest potential and thus the UAV is attracted towards the goal. The UAV's path avoids obstacles due to the obstacles being assigned repulsive force or positive potential. Reference [23] is one of the first publications on this approach. In [24–26], sensing with planning in an evidence grid, using the Vector Force Field (VFF) method, is coupled in order to provide motion planning inputs for ground robots. This reactive method, based heavily on localized sensor input and immediate response, uses a grid with VFF. Other similar approach, known as Vector Field Histogram, avoids obstacles by filtering the perceptions of sensors from 2D to 1D polar histogram [27]. In most publications related to the potential field algorithms, two types of functions are typically considered. The first, based on harmonic functions [28–31], solves partial differential equation with

a Laplacian term. The second function minimizes the distance-to-go function [32].

A challenge for each of these algorithms is the local minima trap issue. This occurs when all artificial forces (attractive and repelling) cancel each other out such as in situations when an obstacle is located between the UAV and the goal or when obstacles are closely spaced. Several approaches have been discussed in [33] to overcome this issue. The solution approach adds an imaginary obstacle in the local minima region to repel the traveling object.

Potential field algorithms require evaluating forces in the configuration space and the complexity of these algorithms can often be $O(M^D)$ where M is the total number of nodes in the space of computation and D is the dimension of the space. Successful application of this algorithm can be found in robot path-planning in [34–38]. The overview of this algorithm for UAV path-planning is described in Algorithm 1.

More details on the Potential Field Algorithm used in this paper for this comparative study are shown below. References [36] and [39] consider the following simple potential function for repulsion from boundaries:

$$P_{\mathrm{HA}} = \frac{1}{\delta + \sum_{i=1}^{s}(g_i + |g_i|)}, \tag{1}$$

where g_i is a linear function that represents the boundary of the convex region, δ is a constant number with a small value and $s = 4$ is a number of boundary face segments. The repulsive forces from an obstacle are derived from the following equation:

$$p_{ij} = \frac{p_{\max}}{1+g}, \tag{2}$$

where

$$\begin{aligned} g(x,y) = {} & (x_0 - l/2 - x) + |x_0 - l/2 - x| \\ & + (x - x_0 - l/2 + 1) + |x - x_0 - l/2 + 1| \\ & + (y_0 - l/2 - y) + |y_0 - l/2 - y| \\ & + (y - y_0 - l/2 + 1) + |y - y_0 - l/2 + 1|. \end{aligned} \tag{3}$$

$p_{\max}$ is the maximum potential and (x_0, y_0) is the coordinate of the center of the obstacle and l is the side length of the obstacle. The potential at any space in the environment is given by the maximum of the potentials due to individual obstacles: $P_0 = \mathrm{Max}\{p_i\}$. Here, $i \in [1, \ldots, M]$ where M is the total number of obstacles. The attractive force generated by the goal is represented by Eq. (30).

$$P_g = C\sqrt{|x - x_{\mathrm{goal}}|^2 + |y - y_{\mathrm{goal}}|^2}, \tag{4}$$

and the resultant force in the environment is represented by

$$P = P_0 + P_g, \tag{5}$$

where

$$P_0 = \mathrm{Max}\{p_i\}. \tag{6}$$

It may be noted that these functions are continuous in nature. Through the process of tessellation, the entire area is divided into cells. The only points considered in path-planning calculations are the centers of each cell. The UAV is constrained to travel only from the center of one cell to the center of cells connected to the UAV's current occupied cell. Virtual objects are used to avoid local minima traps in the domain [37, 40, 41]. The overview of this algorithm developed for the path-planning purpose is shown in Algorithm 10.

Algorithm 1. Overview of potential field algorithm.

```
Start with tessellating the area;
The Goal Weight ← 0
while next position is not goal do
    for all the obstacles do
        Evaluate the repelling function
    end
    Evaluate the attracting function for the goal;
    Apply the resultant forcing function to obtain next location of UAVs;
    if UAV encountered local minima then
        Assume a virtual obstacle there;
    end
    Go to next position
end
Print the path;
```

Algorithm 2. Potential field algorithm for UAV-path planning in tessellated area.

$t = 0, x_c(0) = x_{\text{start}}, Flag = 0$, Calculate the potential function
while *Next decision is not goal* **do**
  if $Flag = 0$ **then**
    Go to the next position with lowest potential,
    else if $Flag = 1$ **then**
      Change the cell weight and treat the cells as occupied by an obstacle
      Update the potential of each cell,
    end
  end
  if *the UAV is trapped in a cell or visit the same cells multiple times* **then**
    $Flag = 1$,
    Search for the trapping point/points,
  end
  if *the UAV flees from the local minima* **then**
    $Flag = 0$,
  end
  $x_{t+1} \leftarrow$ cell nearby with lowest potential function.
  $t \leftarrow t + 1$
end

3. Floyd–Warshall Algorithm

Contrary to the potential field approach, which is continuous and differentiable, the Floyd–Warshall method uses predefined discretized cells to determine the path. It obtains the shortest path through a weighted graph, a discretized domain where each cell has an associated weight or cost. The Floyd–Warshall method associates positive or negative edge weights (but with no negative cycles) unlike traditional weighted graphs that associate the cost or weight with the area of the cell [42]. A vertex represents a physical space in the environment and the edge represents the distance between two vertices. This algorithm solves the "all pairs shortest path problem (APSP)". This technique is used for offline path-planning and it is not suitable for the dynamic path-finding [43]. In [43], computational and timing effort of Floyd–Warshall to reach "only" the optimal answer in path-planning limits the authors to only implement this method in offline path-planning. The other application of this method can be found in [44] where they have shown the robustness and solution performance of this algorithm for a single facility location problem.

The objective of this problem is to minimize Eq. (7) used to calculate $L(i, j, k)$ that returns the shortest possible path from i to j using vertices only from the set $1, 2, \ldots, k$ as intermediate points along the way. k is the number of possible points that UAV can occupy. Now, given this function, the goal of this algorithm is to find the shortest path

Algorithm 3. The Floyd–Warshall algorithm.

for *each vertex* r *on the graph* **do**
  $distance(r, r) \leftarrow 0$
end
for k *in the cells of map* **do**
  for i *in cells of the map* **do**
    for j *cells of the map* **do**
      if $distance(i, j) > distance(i, k) + distance(k, j)$ **then**
        $distance(i, j) \leftarrow distance(i, k) + distance(k, j)$
      end
    end
  end
end

from each i to each j using only vertices 1 to $k+1$ [45–47]. Several different methods have solved the APSP in different ways. For example, Deng *et al.* presented a fuzzy parameter in order to solve the shortest path problem in the Floyd–Warshall algorithm [48]. Algorithm 3 presents the steps used to solve the problem.

$$L(i,j,k+1) = \min(L(i,j,k), L(i,k+1,k) + L(k+1,j,k)),$$
$$\text{Base Case: } L(i,j,0) = w(i,j). \tag{7}$$

4. Genetic Algorithm

A popular and efficient robot path-planning approach uses GA, which belong to the class of meta-heuristic search algorithms. GAs are search strategies based on models of evolution. This evolutionary computation strategy is based on the imitation of natural selection and survival of the fittest. Over the last decade, GA has gained interest from many researchers to solve complex optimization problems related to a variety of applications. Paths (or solutions, each of which are encoded as chromosomes) evolved by the GA parse trees whose lengths can change throughout the run. The GA optimizes a population of paths based on a fitness landscape specified by another function called as cost, fitness, or objective function. The objective function assesses the fitness of each solution. Different implementations of gene fitness functions, mutation functions, and crossover functions affect the performance of the algorithm. One advantage of using evolutionary algorithms is that the time of computation does not directly depend on the number of constraints [49].

The set of chromosomes originated in the program gives the resultant population. An algorithm is utilized that generates an evolution process based on operations such as crossover, reproduction, and mutation. These operations result into an iterative process where successive populations are generated until a satisfactory solution is obtained [50]. The operations carried out in GA for evaluation and creation of new successive generations are repeated until the satisfaction of a convenient termination condition is achieved in the algorithm.

In [51, 52], GA is implemented for both single and multiple UAS path-planning which has shown satisfactory results. Under certain circumstances, GAs can result in premature convergence. In [53], an immune system-based GA is developed, which results in a better convergence. Price and Lamont [51] used a GA design for self-organized search and attack of UAS swarms. In [54], the authors proposed a vibrational GA algorithm enhanced with a Voronoi Diagram in an effort to improve the convergence problem. Application of this method for task assignment with a novel encoding scheme is carried out in [55] and designing a software for path-planning of Raven fixed wing UAV is represented in [56]. Performance of parallelism with a Field Programmable Gate Array (FPGA) based implementation of the GA for autonomous path-planning is carried out in [57].

The work carried out in [58] presents one of the first approaches for robot path-planning that uses a GA. In this approach, the author used dynamic chromosome structures and modified crossover operators. Each chromosome represents one trajectory. The goal of the program was to minimize the deviation between the optimal path and the outcome of the program. The work presented in [59] examines unknown obstacles in the dynamic environment. References [60, 61] further study the application of classifier systems and genetic programming paradigm for robot path-planning. Other efforts for path-planning of a single robot can be found in [62, 63], and for multiple robots in [64–67]. A short description of the algorithm is provided in Algorithm 4.

GA as well as other evolutionary methods have shown success in different supervised learning applications. GA as an iterative evolutionary algorithm has been used in many different UAV path-planning methods [54, 68–71]. A GA path-planning solution represents a feasible shortest collision-free path, i.e. no cells on the path belong to the obstacle space, or none of the line segments of the path intersects an obstacle. The length of a chromosome fluctuates between 2 and a maximum length $N_{\max}$. After generating the path, the GA evaluates the path for feasibility.

In [72], a GA framework is presented, which considers changes in the texture of the ground. Another implementation of GA is presented in [73], which only considers the changing texture and hence the energy requirements when moving within a given energy constraint. In [74], an adaptive GA is developed based on a 2D digital map, and then an adaptive evolutionary planner is inserted based on the expectations from the path to be generated to avoid being detected by the ground surveillance radars. The other approach implemented on UAVs is shown in [75] where prior knowledge is added to the GA process to improve the

Algorithm 4. GA overview.

```
gen ← 0;
Evaluate population(gen);
while termination conditions are not satisfied do
    gen ← gen+1;
    Select population(gen) from population(gen−1);
    Crossover population(gen);
    Mutate population(gen);
    Evaluate population(gen);
end
```

fitness function. In their approach, by selecting essential points and moving strategy in advance, reduction in computation cost and obtaining the optimal path more efficiently are achieved. The other implementation of GA for UAVs can be found in [76] where the GA with TSP approach is used for finding the optimal path-planning for UAVs in a 3D environment.

5. Greedy Algorithm and Multi-Step Look-Ahead Policy (MSLAP)

MSLAP algorithm works by discretizing the UAV decision tree and evaluating different multi-step UAV path decisions for optimal performance. However, the problem is NP-hard and therefore the computation time is dependent on the number of decision variables. Defining the size of decision possibilities for the UAV modulates the cardinality of set of decision variables. A sub-optimal solution requires less computation time and hence has advantage when compared to the optimal trajectory planning methods. This makes heuristic approaches lucrative since these approaches make use of *a priori* knowledge of problem structure to arrive at approximate solutions. An example is the roll-out policy proposed in [77] based on a heuristic solution to the problem (called a base heuristic). The roll-out policy guarantees to find a solution that is no worse than the base heuristic. [78–80] show successful applications of the roll-out policy. By assuming that each cell represents a decision cell for a UAV, Fig. 2 illustrates the roll-out policy in a three-step look-ahead strategy for a single UAV. The greedy heuristic selects the decision that leads to the next cell with the best immediate result. In a three-step look-ahead roll-out policy, instead of starting from B, the greedy heuristic starts from $B+1$ to generate paths to $B+3$. The decision that leads to the objective functions lowest value is set as the best path. The paths' evaluation from B to $B+b$ are based on the information available at the time UAV is placed in B and the procedure is repeated at every time step with the updated information.

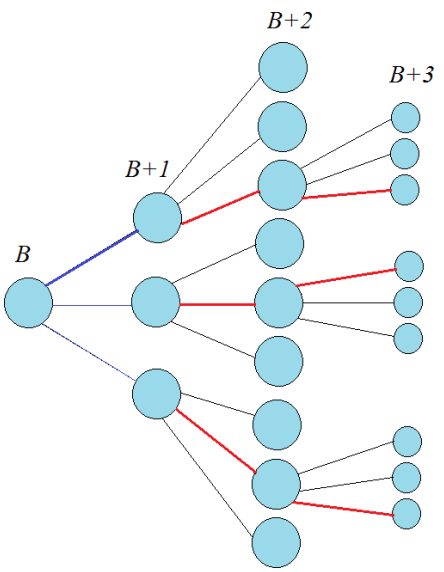


Fig. 2. Roll-out policy exerted on the cells for three-step look ahead.

Compared to exhaustive search methods, which require evaluation of the $\sum_{i=1}^{b} D^i$ cells, where D represents the number of cells the UAV could occupy as a result of the next decision, the roll-out policy only evaluates $D + (b-1)D^2$ cells. The computational cost increases linearly with the decision horizon b.

6. *A** Algorithm

A^* was first introduced in 1963 in [81] and has since become a popular method for robot path-planning and graph traversal. A^* uses best-first search in order to find the least-cost path from an initial node to the goal node [82–88]. The 2D implementation of this path-planning algorithm is quite mature and extensively developed. However, 3D implementation still provides computational challenges [89]. The experimental implementation of this algorithm for UAVs without considering many disturbances for UAVs is then studied without any feasibility, efficiency, or convergence proof in [90, 91]. Some researchers use two-dimensional Voronoi map to divide the target space into several sections to construct connected network graph [92]. In [93], an A^* algorithm is introduced for the real-time path-planning of UAVs in a 3D large-scale battlefield environment to obtain high survival rates of UAVs and low fuel consumption.

In general, the cost function is considered as Eq. (8) for this algorithm.

$$f(i) = s(i) + h(i). \tag{8}$$

In Eq. (8), parameter $s(i)$ is the known cost function of going from the initial node to node i and parameter $h(i)$ indicates the heuristic estimation of cost from node i to the goal point.

The parameter that distinguishes this method from a greedy algorithm is $s(i)$. This implements a priority queue of the nodes to be traversed, i.e. the higher priority is given to the lower value of $f(i)$. Hence, the value of f is then the length of the shortest path to the goal position. The overview of the algorithm is provided in Algorithm 5.

This type of programming's complexity is $O(b^d)$, in which b is the average number of the successors per state and d represents the length of the shortest path [17]. Several

Algorithm 5. A^* algorithm for path-planning.

```
O represents the openlist;
while O is not empty do
    Next position ← lowest f in O;
    if Next point is goal point then
        Print the path;
    else
        Current position ← Next position;
        Remove the Next position to C;
        for All the neighbors of the current position do
            if neighbor is not in O then
                save s, h and f;
                save the previous point and the neighbor to O;
                else if Neighbor is in O and Neighbor's s is better than previous s then
                    save s and f;
                    save previous position;
                end
            end
        end
    end
end
```


algorithms have been derived from A^* algorithm such as D^*, *IDA**, *FSA**, *GAA**, *SMA** and θ^*.

6.1. *D** *algorithm*

D^*-based algorithms provide an assumption-based solution to the UAV path-planning [94]. The implementation of these algorithms is useful when the UAV needs to navigate through the areas with unknown terrains. The method seeks the sequences of similar search problems by utilizing previous search patterns. This results in a faster computational algorithm for exploration and path-planning.

Since D^* was first developed by [95], multiple versions of it have appeared in the literature such as *Focused* D^* which combines informed heuristic algorithm of A^* with D^* [96]. Similarly, D^* *Lite* [97] is an incremental heuristic search algorithm that combines Dynamic Strict Weakly Superior Function Fix Point (SWSF-FP) and A^* algorithm [98–101]. The application of this algorithm in path-planning of single agents can be found in [102–105]. The important limitations of D^* and its origins from A^* algorithm for robot path-planning are unnecessary turns as well as assumptions of transition cost from particular grid node to each of its neighbors [106]. Reference [107] is founded on the fast marching method [108], an algorithm developed to combine the grids based on the surface flow equation. This method demonstrates a smoother path in navigating the robot through the obstacles [106, 109]. The *Field* D^* was developed to produce near-optimal solutions based on the consideration of the local variation in the cell traversal costs and linear interpolation method. This algorithm provides less costly paths than the grid-based algorithm [106, 110–113].

6.2. *Iterative deepening A** *(IDA***)*

IDA^*[19] is a memory-efficient version of the A^* algorithm. It trades the space for time by eliminating the open-list and the closed-list from the A^* algorithm. Since the IDA^* iterates and repeatedly explores paths, it results in significant inefficiency. The IDA^* algorithms consider a starting threshold for the cost function h. The algorithm continues with a recursive depth first search, with a loop break if either the goal is found or a node has a value f more than the initial threshold. If the result of this loop is empty, then the threshold is increased till it finds the next point. The

application of this algorithm on the path-planning in robots can be found in [114–116].

6.3. *Fringe algorithm*

Path-planning approaches that apply fringe algorithms try to improve IDA^* inefficiencies by making data structures to iterate over two sets of data: frontier and fringe. In other words, there are two sets of lists, which store the current iteration and next iteration. This algorithm is shown to accelerate the search runs by 10–40% as compared to the A^* path-planning algorithm [20, 117].

6.4. θ^* *algorithm*

This type of algorithm searches for paths using a cell decomposition method. The algorithm checks for shortcuts during each node expansion. The algorithm's operation disallows the parent of a node in the search tree to be the neighbor. Depending on the geometry of the problem, the resulting path is the near optimal path [118–120]. The algorithm *Lazy* θ^* is based on reduction of line-of-sight calculations in θ^* [121]. ϕ^* is another incremental derivative of θ^* that makes it more efficient in uncertain 2D environments [122–124]. The efficiency of this algorithm has been compared to A^* algorithm in [92].

7. Dynamic Programming (DP)

Conceptually, dynamic programming breaks the problem into multiple related sub-problems. Solving the main status naturally requires solutions for a number of sub-problems. The solution of sub-problems is stored in a memory structure for future needs [125]. In terms of UAV path-planning, the outcome of the DP algorithm is to calculate the distance to the goal from all the points in the map and the sub-problem is the pre-computed distance to the nearby points [126, 127]. Recurrence relation is a common solution demonstrating the relationship between the problem and its sub-problems. DP algorithms are the basis for many types of algorithms and solution methods. In these solutions, the smallest sub-problem is considered first, then proceeding toward the higher sub-problems, the main solution is found in an *iterative fashion*. The shortest distance from any point to the destination at iteration $n+1$ using the shortest distances from the neighboring points at iteration n forms the basic calculation of the problem. The shortest path is now saved in the source equation and an **update function at each iteration** generates the shortest path as a result of this method [128, 129]. The most significant algorithm, derived from dynamic programming that solves for the shortest path, is Dijkstra's algorithm. This algorithm represents a successive approximation scheme for solving the dynamic programming equations for the shortest path [130, 131].

7.1. *Dijkstra's algorithm*

This algorithm was developed by Edsger W. Dijkstra in 1956 [132, 133]. The main purpose of this algorithm is to develop a greedy method for application such as the shortest path-planning [134]. Heavy modification has decreased the complexity of the algorithm. The original algorithm showed computational complexity of $O(n^{2^n})$ and with added simplifications, the complexity has been reduced to $O(n^3)$ in special cases [135, 136]. n is the number of possible nodes on the non-weighted graph. The basis for this algorithm is the principle of relaxation; values that are more accurate gradually replace approximations of the correct distance until eventually reaching the optimum solution. The Dijkstra's shortest path algorithm is described in Algorithm 6.

Applications of the DP method for path-planning have been studied in many papers [137–142].

7.2. *Bellman–Ford algorithm*

The Bellman–Ford algorithm computes the shortest paths from a single source vertex. This algorithm has been proven to be slower than Dijkstra's algorithm [143, 144]. This algorithm was developed by Richard Bellman and Lester Ford Jr., and also known as Bellman–Ford–Moore algorithm [145, 146]. Like Dijkstra's algorithm, the principle of relaxation is the basis of this algorithm. Unlike Dijkstra's algorithm, which uses a priority queue similar to a greedy algorithm to select the closest node and performs this relaxation on all of the neighbors, all the neighbors are relaxed via the Bellman–Ford algorithm. The algorithms complexity is found to be order of the number of nodes in the area given by $|v|$. This algorithm is useful for finding the shortest path in graphs with negative weights [147], while the application of this algorithm has been proven to be efficient in uncertain environment [148]. Pseudo code for Bellman–Ford algorithm for finding the path in a tessellated area is given in Algorithm 7. Here, the objective is to find a vector d as an output and d_i is the shortest distance from initial point s to node i in the tessellated area.

Two popular algorithm modifications, made by Yen and Bannister–Eppstien, accelerate the computation of the solution. Introduced in 1970, Yen's modification reduced the required operational number of steps. In this modification,

Algorithm 6. Dijkstra's algorithm for path-planning.

```
Initial point is s_1,
while next decision is not Goal do
    Create the node set S,
    for Node v in the area do
        distance from initial point: distance(v) ← ∞,
        Previous node in solution: previous(v) is undefined,
        Add v to solution array,
        distance(s_1) ← 0,
    end

    while |R| ≠ 0 do
        find the node with minimum distance function from Q and add to u,
        Remove u from solution array,
        for Neighbors r of node u do
            D ← dist(u)+ length of r to u,
            if D > dist(r) then
                dist(r) ← D,
                Previous(r) ← u
            end

        end
    end
     Print the path,
end
```

as the number of vertices grows, the number of outgoing edges, which need to be in relaxation, would shrink. This results in reducing the worst-case number of iterations of the algorithm from $|v| - 1$ to $\frac{v}{2}$ by assigning some arbitrary linear order on all the vertices and then partitioning the set of all edges into two sets [16]. The other improvement was made by Bannister and Eppstein by replacing the random permutation to arbitrary linear order of the vertices [149].

Algorithm 7. Bellman–Ford algorithm for path-planning.

```
d_s := 0 and d_v := inf
for All the remaining nodes in the area do
    for each of the possible connections of u to v with cost c do
        d_v := min(d_v, d_u + c)
    end
end
 for each possible edge from u to v with cost c do
    if d_v > d_u + c then
        Negative weight cycle is in the graph
    end
end
```

As a result, the number of iterations of the main loop decreased to $\frac{|v|}{3}$ [150].

8. Approximate Reinforcement Learning (RL)

RL program learns to take actions with the goal to maximize a reward signal. The earned reward is the feedback for the next action and is known as *exploitation*. The agent may also choose to explore the environment for better future action selection [151, 152]. Value functions, functions of the states, estimate the degree of importance that the agent gives to be in a given state [153]. Estimation of a value function is the main objective of the RL problem. The reinforcement learning model has five elements:

- S as the set of states,
- A as the set of actions,
- Policy or mechanism for transitioning between the states,
- Determination of the scalar reward of a transition,
- Method for the observing the agents.

In general, the methods are often stochastic [154, 155]. Furthermore, reinforcement learning is particularly more

Algorithm 8. Reinforcement learning algorithm for robot path-planning.

```
while current position ! = goal position do
    for given number of steps, k do
        while current position ! = goal position do
            Identify neighboring grids,
            Move to the next grid with minimum value among the neighboring grids.
            Update the previous position of UAV,
            current position← next grid
        end
    end
    Set the next decision of UAV as the current position
end
```

applicable for the problems which include a long-term versus short-term reward trade-off [156]. The overview of a practical method for UAV path-planning using RL is provided in Algorithm 8.

The choice of number of exploration steps depends on the size of the environment and the speed of computation [157]. The successful application of this algorithm for UAV path-planning can be seen in [158–161]. The other method of path-planning which is derived from the RL algorithm is Q-Learning. Path-planning of multi-agents vastly use this method.

Due to large state-space, it often becomes computationally difficult to learn and store value functions. Approximate dynamic programming was introduced to address this problem [162]. An example of such algorithms is neuro-dynamic programming. Neuro-dynamic programming uses neural networks and other approximation architectures to overcome such bottlenecks to the applicability of dynamic programming. The methodology allows systems to learn about their behavior through simulation and to improve their performance through iterative reinforcement.

8.1. *Reinforcement learning with Q-learning*

The Q-learning technique is a popular method for path-planning in the literature [163–165]. Q-learning (with a lookup table representation) can be viewed as a method for solving Bellman's equation using stochastic approximation. Convergence is established by first developing a stochastic approximation theory for the case where the iteration mapping is a contraction with respect to a weighted maximum norm [166]. Maximizing the sum of reinforcement function corresponds to rational allocation as the objective of robot. Let S be set of all the possible states and A be set of all actions. An action α_t by the robot yields a real return r_t. The objective of reinforcement learning is obtaining a

Algorithm 9. The algorithm for Q-learning.

```
Consider s = 0 and a = 0;
while current position! =Goal do
    select an action a using equation 10
    find the reward r
    update the value of cells using equation 9.
    s ← s'
end
```

strategy $\pi : S \rightarrow A$ that maximizes these returns. To learn the Q values (that maps state to action), training is carried out based on the immediate return and long-term return of the action as shown by Eq. (9):

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a'). \tag{9}$$

In this method, by using the probability process models such as Markov Decision Process (MDP), UAV repeatedly observes the current state s, selects and executes a certain action a, observes the returned result $r = r(s,a)$ and the new state $s' = \delta(s,a)$. Any action a can be found using Eq. (10).

$$a = \underset{a}{\operatorname{argmax}}[r(s,a) + Q(\delta(s,a), a')]. \tag{10}$$

The algorithm used for this purpose is given in Algorithm 9.

9. Mixed Integer Linear Programming (MILP)

MILP has been vastly used for mathematical modeling and known as a powerful tool to present optimal and near-optimal solutions [167–169]. Implementation of MILP as a solution to the UAV path-planning considering the probability of detection of other UAVs is done in [170]. Furthermore, optimizing the task allocation problem for a fleet of

UAVs with tightly coupled tasks and timing constraints has been studied in [171]. The use of planar kinematic model of a UAV resulting in linear constraints on rotational velocity by considering the differential flatness is addressed in [172, 173] presents a formulation for UAV mission scheduling for real-time applications in hostile environments. Terrain and communication constraints for UAVs as MILP constraints are treated in [174]. The complexity of MILP is dependent on the efficiency of the solver. Although the number of binary variables is often a poor indication of complexity of MILP, it is always useful to reduce the number of binary variables for faster solutions [175]. The solution space grows exponentially with the number of binary variables [167, 176–178] presents some heuristic solutions for path-planning of the UAVs. Furthermore, in [179], the computational effort for smaller number of binary variables using the iterative method is studied. A multi-UAV task allocation is done in [180] to solve a task assignment problem in hostile environment by adding dynamic constraint builder.

In general, path-planning problem formulated in an MILP framework involves minimizing a *linear* cost function involving energy consumption (Eq. 11):

$$J = \sum_{t=0}^{T} f_{x,t} + f_{y,t} + f_{z,t}, \tag{11}$$

where $f_{x,t}, f_{y,t}, f_{z,t}$ represent the forces exerted on the body of UAV along x, y, and z directions, respectively. In other formulations, the objective is to minimize the length of

Table 1. Features of the studied algorithms.

Method	Features
1. Potential Field	• Difficult to implement for real-world application • Poor performance in narrow passages • Poor performance in dynamic environment • Prone to get stuck in local minima situations • Problems in dealing with the symmetrical obstacles
2. Floyd–Warshall	• It is a subset of Dynamic Programming with optimal substructures • It can lead to optimal answer in grid-based map • Better performance compared to Dijkstra and Bellman–Ford algorithms [188] • Capable of solving hard path-planning problem • Due to high computational complexity, not a suggested method for on-board implementation
3. Genetic Algorithm	• Highly capable heuristic method • Time of computation and algorithm complexity is highly dependent on the algorithm implementation • Recent combination of this algorithm with learning process resulted in a very promising online application [189]
4. A^*	• It can provide a general heuristic approach for search of optimal path • It can potentially search a huge area of the map • The worst case for this algorithm results in considering all the nodes and edges in the graph • The solution time would benefit from static environment • Based on the scenarios, may lead to optimal or near-optimal solutions
5. Dynamic Programming	• It solves a complex problem by breaking into collection of subproblems • It requires memories to save all the solutions generated from iterations • Time of computation rises as the size of the problem grows • The solution guarantees the optimal answer based on the implementation
6. Approximate RL	• It involves estimating a value function • It explicitly considers the whole problem of an agent directed towards a goal in an uncertain environment • At any time step, the agent has a choice between *exploitation* and *exploration* • The agent must discover what actions to take so as to maximize the reward signal in long term • The solution obtained is often not optimal because value functions are not optimally obtained in limited learning environment
7. MILP	• The implementation specifics and constraints have high impacts on time of computation • Recent solvers accelerate the speed of computation • Capable of reaching to optimal answer for the well-defined problem

the path:

$$J = -\dot{p}_t(p_w - p_0 + ||p_t - p_w||), \tag{12}$$

where p_t represents the current position of the UAV, p_w is the position of the waypoint and p_0 is the initial point. In Eq. (12), $||p_t - p_w||$ represents the distance between the current position and goal position.

A practical method for navigating the UAV in a tessellated area can be found in [181, 182]. This method uses a distance-based value function that represents the weight of the cell of the tessellated region:

$$\begin{aligned} &\forall i \in V, \\ &\text{weight(goal)} = 0, \\ &\text{weight(i)} = \min_{j \in \text{ neighbor of cell } 'i'} \text{weight}(j) + 1, \end{aligned} \tag{13}$$

V represents the number of cells in tessellated area. The objective function can be written as follows:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} \zeta_{ij} \text{weight}_{ij}, \tag{14}$$

where ζ_{ij} is a binary variable and weight_{ij} is the cost of traveling from i to j [181]. The solution is achieved via Fast-Floating Point (FFP) developed in [176].

Among the other approaches that were implemented on mostly ground robots are Bug Algorithm where the robot is given two decision commands: (i) move toward the goal and (ii) avoid the obstacles [183]. After avoiding obstacle, the robot again restarts moving towards the goal without considering any other parameter or past behavior. The robot is usually considered as a point object without any dimension and the path is assumed to be a function of distance to the goal position. Due to the unidirectional obstacle avoidance approach, trajectories followed are sometimes very long and thus the robot may take more time to reach the goal [184, 185].

The other approach is Bubble-Based Technique (BBT) [186]. In this approach, a bubble is defined as a local region of free space around a configuration of the robot and an initial path is generated as a solution to the problem of moving a robot between the start and goal configuration. Then, adjustments to the path are made while maintaining a global path while taking into account the condition of bubbles. Inspired by this algorithm, in [187], a sensitive bubble is defined when the robot detects obstacles within an area. Then, the robot moves in the direction of lowest density. The robot continues to follow this direction unless another obstacle is detected or destination is achieved. The dimensions of the bubble depend on the kinematic characteristics of the robot.

As a summary of all the approaches provided in this paper, Table 1 provides a comparison of various features of the algorithms and methods studied.

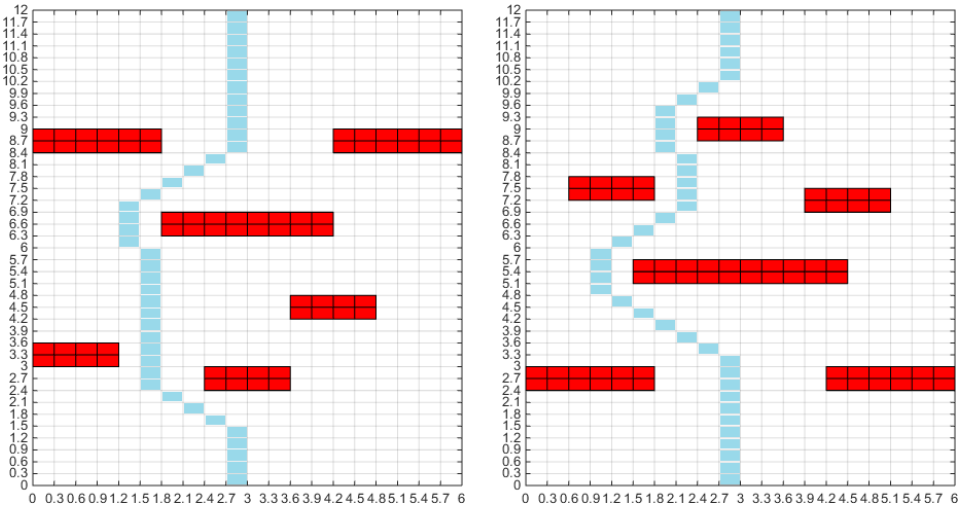
10. Results and Discussions

Three different obstacle layouts are used to compare the described algorithms' time of computation and optimality of the solution. It is important to note that the aim of this simulation study using limited number of scenarios is to provide examples to substantiate some of the qualitative properties mentioned in this paper. It is not intended to serve as a basis to arrive at any conclusion about superiority of an algorithm over the rest. Indeed, the comparative performance of the algorithms would differ for different kinds of scenarios based on complexity of the problem and obstacle structure. The dynamics of the UAV, more specifically, a quadcopter, is considered for this problem and can be found in [190]. A brute-force algorithm obtains a globally optimal solution [191]. Figure 3 shows the optimal paths for these three obstacles obtained using an exhaustive search method.

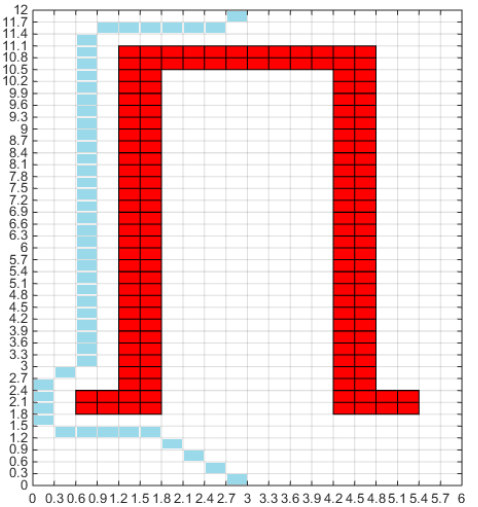
Tables 2–4 represent the time of computation (TC) and solution's optimality for the three obstacle layouts. Each layout has a different number of cells and positions of obstacles. Maximum error values represent the percentage increase in the path length provided by the algorithm compared to the optimal path length obtained via exhaustive or brute-force search. Considering that there may be different results or paths with the same number of steps, the table provides the optimal path length based on the number of steps UAV takes to reach the goal. Each layout undergoes three different tessellation resolutions resulting in different total number of cells as: Scenario 1 — 800 cells, Scenario 2 — 80,000 cells, and Scenario 3 — 8,000,000 cells. These resolutions help us to study the scalability properties of the algorithms. The algorithms were run three times for each scenario and the average tabulated. The maximum error values used for these tables are based on Eq. (15) and are calculated for the worst-case results from the algorithms from the three scenarios.

$$\text{Error} = \frac{\text{Path Length} \ - \ \text{Optimal path length}}{\text{Optimal path length}} \times 100\%. \tag{15}$$

Obstacle Layout 1: The algorithms' results for the first layout are shown in Table 2. Some of the algorithms provide sub-optimal solutions. These algorithms are potential field algorithm, MSLAP, and GA. The GA was stopped during the simulation for 8,000,000 cells as it reached the terminal number of generation but still showed acceptable result with approximately 5% error. The maximum error occurred in the cases of the MSLAP and potential field algorithms with 15% error. The other algorithms resulted in the optimal path with zero percent error. The most reasonable algorithm from time and optimality point of view was found to be the MILP algorithm. The optimum path lengths for Scenarios 1, 2, and 3 are 39, 392, and 3901 cells, respectively.



(a) Obstacle layout 1 (b) Obstacle layout 2



(c) Obstacle layout 3

Fig. 3. Different obstacle layouts for optimization tests and corresponding globally optimal paths.

Table 2. Time of Computation (TC) in seconds for each method for obstacle layout 1.

	Scenario 1: 800 cells		Scenario 2: 80,000 cells		Scenario 3: 8,000,000 cells	
Method	TC	Max Error (%)	TC	Max Error (%)	TC	Max Error (%)
Potential Field	1.352	9.2	5.786	12.6	7.653	15.4
Floyd–Warshall	2.352	0	95.652	0	10410.784	0
Genetic	2.355	2.2	67.766	3.6	2659.653	5.1
MSLAP	0.926	6.4	2.352	9.5	3.301	15.4
A^*	1.310	0	86.239	0	9859.253	0
Dijkstra	2.256	0	109.237	0	9962.012	0
Approximate RL	22.012	0	625.326	0	14471.749	0
MILP	1.065	0	36.562	0	3502.359	0

Table 3. Time of Computation (TC) in seconds for each method for obstacle layout 2.

	Scenario 1: 800 cells		Scenario 2: 80,000 cells		Scenario 3: 8,000,000 cells	
Method	TC	Max Error (%)	TC	Max Error (%)	TC	Max Error (%)
Potential Field	1.436	16.4	6.002	18.7	7.647	20.2
Floyd–Warshall	2.352	0	103.602	0	11410.784	0
Genetic	3.851	16.2	72.253	18.1	2659.653	20.0
MSLAP	0.874	19.0	1.253	19.8	2.469	20.0
A^*	1.712	0	93.678	0	10256.238	0
Dijkstra	2.987	0	114.453	0	10257.253	0
Approximate RL	24.578	0	768.235	0	16253.258	0
MILP	1.464	0	42.150	0	3518.215	0

Table 4. Time of Computation (TC) in seconds for each method for obstacle layout 3.

	Scenario 1: 800 cells		Scenario 2: 80,000 cells		Scenario 3: 8,000,000 cells	
Method	TC	Max Error (%)	TC	Max Error (%)	TC	Max Error (%)
Potential Field	4.436	16.2	13.002	18.0	101.647	20.4
Floyd–Warshall	4.357	0	148.326	0	14253.657	0
Genetic	4.782	15.9	106.326	16.0	2659.653	16.3
MSLAP	0.971	20.1	1.745	26.2	4.326	28.6
A^*	2.326	0	146.326	0	14253.985	0
Dijkstra	3.000	0	198.325	0	17452.326	0
Approximate RL	27.578	0	854.235	0	203289.258	0
MILP	2.356	0	62.857	0	3512.457	0

Obstacle Layout 2: Obstacle layout 2 differs from layout 1 in the sense that the obstacles are symmetric. The right-hand side of the layout 1 has more empty space but results in the longer path length. The results in Table 3 show that since both potential field algorithm and GA chose their path from the right-hand side of the map, it resulted in high error in the calculation. The GA reached the terminal iteration that caused high error. A higher threshold for the terminal iteration would result in a more optimal solution but would take longer computation time. Again, MILP solution performed better than the other algorithms. The exhaustive search algorithm finds the optimal path length 40 cells long for Scenario 1. The maximum error is 20% for MSLAP, potential field algorithm as well as GA.

Obstacle Layout 3: The third obstacle layout design forces the algorithms to encounter a region of local minima. The potential field algorithm took longer time as compared to the other paths since it faced multiple local minima during the solution. Since MSLAP does not consider the complete map, it became stuck in local minima as well and therefore needed to adjust its path. The GA stopped at the terminal generation count for the 8,000,000 cells. The maximum error, found during the simulations using the GA approach, was 16.3%. The optimal path length is 49 cells for Scenario 1. Again, MILP appeared to provide the most efficient solution methodology.

The results obtained using different algorithms applied to the three layouts are in accordance with the expectations. As the complexity of the problem increases, the computational time also increases [192]. In previous studies, it has been shown that the algorithms such as Floyd–Warshall and Dynamic Programming consume more time to find the exact answers as the size of the problem grows [193]. On the other hand, the heuristic methods such as the GA and then RL are more powerful tools to find the answer more efficiently [49]. However, there is no guarantee for finding the most optimal answer using these heuristic methods. In other studies, it has been shown that the A^* compared to Dijkstra achieves lower time by using the heuristic function [194] which is also shown in this paper.

11. Conclusions

This paper presented an exhaustive overview of different popular algorithms used for path-planning of UAVs. The paper also presented a comparative study of the algorithms

for path-planning for different scenarios and obstacle layouts in terms of time of computation and optimality of solution. From the simulation study of these specific scenarios with different obstacle layouts, it was found that the GA showed less sensitivity to time with respect to the increase in number of cells. MSLAP is the fastest solution but often not optimal. Furthermore, the potential field algorithm shows a reasonable time of solution, but it shows poor ability to overcome the local minima and provides non-optimal results. MILP shows the ability to provide solutions for complex scenarios in reasonable computational time without compromising on the optimality of the solution. Also, the A^*, Dijkstra's algorithm and Floyd–Warshall algorithm show the ability to solve the scenarios optimally by spending considerably higher computational time than MILP. It may be noted that these results are for the specific scenarios studied in this paper. Different scenarios may yield different results in terms of the performance of the algorithms. It is clear from the results that there is a trade-off between the optimality and computational time requirements. Choice of algorithm in practice needs to be based on operational requirements in terms of computational time and optimality.

Acknowledgment

This material is based upon the work supported by the National Science Foundation under Grant No. IIP-1526677. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Appendix A. Quadcopter Model

Quadrotors are one of the most widely used UAVs in various applications mainly due to their vertical take off and landing (VTOL) capability, simplicity of construction, maneuverability, and ability to negotiate in tight spaces. Hence, it is very important to predict their flight behavior under various conditions so that they could be controlled with precision. This section presents the dynamic model of a quadrotor vehicle.

A.1. *Dynamic model*

The coordinate systems and the free body diagram of the UAV are shown in Fig. A.1. The world frame (W) denotes the fixed reference frame with respect to which all motion can be referred to and the body-frame (B) is a frame

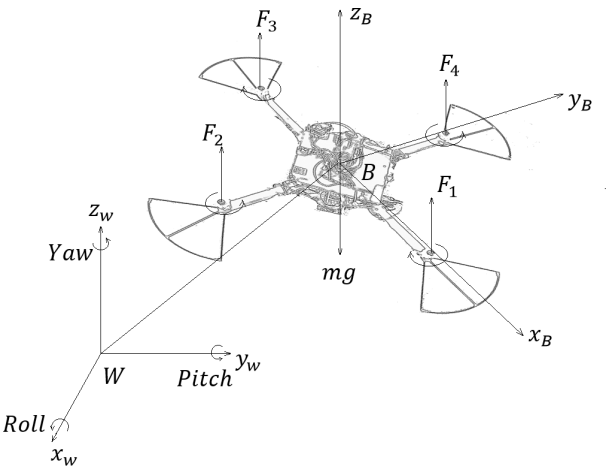


Fig. A.1. The coordinate systems and the free body diagram.

attached to the center of mass of the vehicle. There is a vertical force for each rotor that comes from the rotation of the rotor. In addition to the forces, each rotor produces a moment perpendicular to the plane of propeller rotation. The Z–X–Y Euler angle convention is used to model the rotation of the quadrotor in the W frame. To get to the B frame, we first rotate through z_W by yaw angle, ψ, then rotate about the intermediate x-axis by the roll angle, ϕ, and then about the y_B by the pitch angle, θ. Hence, the rotation matrix R from W to B frame can be written as

$$R = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\phi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix}, \quad (A.1)$$

where $c\psi$ and $s\psi$ denote $\cos(\psi)$ and $\sin(\psi)$, respectively, and similarly for other angles.

The equations of motion governing the acceleration of the center of mass can be written as follows:

$$m\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R\begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix}, \quad (A.2)$$

where m is the mass of the quadrotor and g is the acceleration due to gravity.

A.1.1. *Motor control*

Each rotor produces a vertical force F_i due to the rotation given by

$$F_i = k_F \omega_i^2, \quad (A.3)$$

where ω_i is the rotational speed of rotor i, and k_F is a constant whose value is given in [195] as $k_F = 6.11 \times 10^{-8}\text{N/rpm}^2$. In addition, Euler equations are written in order to obtain

angular accelerations of the vehicle given by

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}L(F_2-F_4)\\ L(F_1-F_3)\\ M_1-M_2+M_3-M_4\end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix}\times I\begin{bmatrix}p\\ q\\ r\end{bmatrix}, \quad (A.4)$$

where L is distance of each rotor from the vehicle's center of gravity. I is the matrix for moment of inertia along x_B, y_B, and z_B directions, respectively. M_i, $(i = 1, 2, 3, 4)$ are rotor moments produced by angular velocity of rotors and are given by

$$M_i = k_M \omega_i^2, \quad (A.5)$$

where ω_i is the angular velocity of i^{th} rotor and k_M is the constant, given in [195] as $k_M = 1.5 \times 10^{-9}\text{Nm/rpm}^2$. For hover state, the thrusts from the propellers must satisfy

$$F_{i,0} = \frac{mg}{4}, \quad (A.6)$$

and the motor speeds are given by

$$\omega_{i,0} = \omega_H = \sqrt{\frac{mg}{4k_F}}. \quad (A.7)$$

A.1.2. *Attitude control*

The controller inputs are four independent speeds of propellers. The dynamic model of the quadcopter, described in (A.2) and (A.4), is used to design a PD controller.

Substituting (A.3) and (A.5) in (A.4) and assuming that the component of angular velocity in z_B direction is small compared to other terms, we get the vector of desired rotor speeds as linear combination of four terms:

$$\begin{bmatrix}\omega_1^{des}\\ \omega_2^{des}\\ \omega_3^{des}\\ \omega_4^{des}\end{bmatrix} = \begin{bmatrix}1 & 0 & -1 & 1\\ 1 & 1 & 0 & -1\\ 1 & 0 & 1 & 1\\ 1 & -1 & 0 & -1\end{bmatrix}\begin{bmatrix}\omega_H + \Delta\omega_F\\ \Delta\omega_\phi\\ \Delta\omega_\theta\\ \Delta\omega_\psi\end{bmatrix}, \quad (A.8)$$

where ω_i^{des}, $(i = 1, 2, 3, 4)$ are the desired angular velocities of the respective rotors, and ω_H is hovering speed. The proportional-derivative control laws are used to control $\Delta\omega_\phi$, $\Delta\omega_\theta$, $\Delta\omega_\psi$, and $\Delta\omega_F$, which are deviations that result into forces/moments causing roll, pitch, yaw, and a net force along the z_B axis, respectively, and are calculated as

$$\begin{aligned}\Delta\omega_\phi &= k_{p,\phi}(\phi^{\text{des}} - \phi) + k_{d,\phi}(p^{\text{des}} - p),\\ \Delta\omega_\theta &= k_{p,\theta}(\theta^{\text{des}} - \theta) + k_{d,\theta}(q^{\text{des}} - q),\\ \Delta\omega_\psi &= k_{p,\psi}(\psi^{\text{des}} - \psi) + k_{d,\psi}(r^{\text{des}} - r),\\ \Delta\omega_F &= \frac{m}{8k_F\omega_H}\ddot{Z}^{\text{des}},\end{aligned} \quad (A.9)$$

where p, q, and r are the components of angular velocities of the vehicle in the body frame. The relationship between these components and the pitch, roll, and yaw are provided in [196].

A.1.3. *Position control*

In order to have the quadrotor track a desired trajectory $r_{i,T}$, the command acceleration, $\ddot{r}_i^{\text{des}}$ is calculated from proportional-derivative controller based on position error as [197]

$$(\ddot{r}_{i,T} - \ddot{r}_i^{\text{des}}) + k_{d,i}(\dot{r}_{i,T} - \dot{r}_i) + k_{p,i}(r_{i,T} - r_i) = 0, \quad (A.10)$$

where r_i and $r_{i,T}$ $(i = 1, 2, 3)$ are the three-dimensional position of the quadrotor and desired trajectory, respectively. It may be noted that $\dot{r}_{i,T} = \ddot{r}_{i,T} = 0$ for hover.

During the flight, the orientation of the vehicle needs to be set close to zero. This can be obtained by linearizing the equation of motion that corresponds, to the nominal hover states. The nominal hover state is ($r = r_0$, $\phi = \theta = 0$, $\psi = \psi_T$, $\dot{r} = 0$ and $\dot{\theta} = \dot{\psi} = \dot{\phi} = 0$). The change of the pitch and roll angles are supposed to be small during flight. By linearizing Eq. (17) about these nominal hovering states, desired pitch and roll angles to cause the motion can be derived as given by the following equations:

$$\begin{aligned}\phi^{\text{des}} &= \frac{1}{g}(\ddot{X}^{\text{des}}\sin\psi_T - \ddot{Y}^{\text{des}}\cos\psi_T),\\ \theta^{\text{des}} &= \frac{1}{g}(\ddot{X}^{\text{des}}\cos\psi_T + \ddot{Y}^{\text{des}}\sin\psi_T),\end{aligned} \quad (A.11)$$

where $\ddot{X}^{\text{des}}$ and $\ddot{Y}^{\text{des}}$ are the desired acceleration in X and Y directions, respectively. ψ_T is the yaw angle to be tracked which is the same as the desired yaw angle ψ^{des}.

This dynamic is exposed to the path planning algorithms. As it can be seen, Fig. A.2 is one of the results of the GA method with violation over the solution.

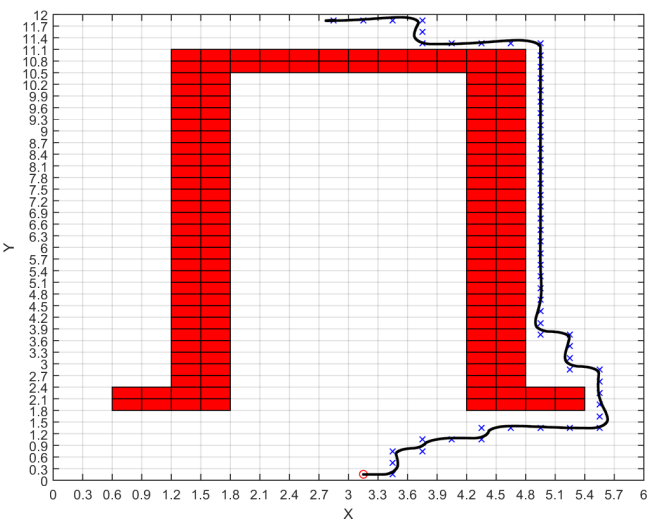


Fig. A.2. Quadrotor path generated via GA algorithm.

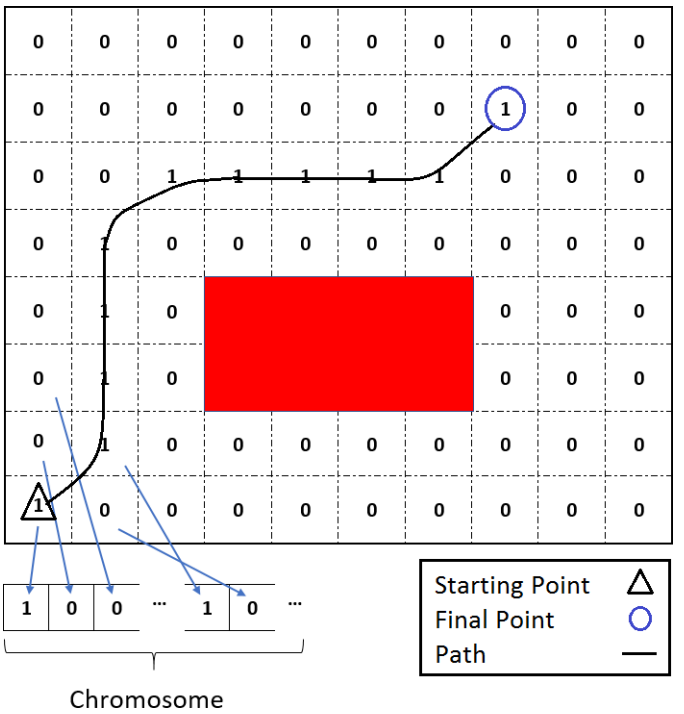


Fig. B.1. Scheme of the chromosome for GA.

Appendix B. Implementation of GA

For implementing GA, this paper considers a binary array with the length of 1 × Number of Cells as a chromosome and each chromosome represents a complete path. The cells in the tessellated area has been already labeled and addressed in the array. If a cell is a part of a particular path, then the binary variable corresponding to that cell in the chromosome is assigned a value of 1. Otherwise, it is assigned a value of 0. A schematic diagram showing a particular path and corresponding chromosome is shown in Fig. B.1. The rest of the implementation details is provided in Algorithm 4.

Appendix C. Implementation of Artificial Potential Field

In this approach, a simple potential function for repulsion from boundaries is considered [36, 39]:

$$P_{HA} = \frac{1}{\delta + \sum_{i=1}^{s}(g_i + |g_i|)}, \tag{C.1}$$

where g_i is a linear function that represents the boundary of convex region, δ is a constant number with a small value

Algorithm 10. Potential Field algorithm for UAV path planning in tessellated area.

$t = 0, x_c(0) = x_{\text{start}}, Flag = 0$, Calculate the potential function
while *Next decision is not goal* **do**
  if $Flag = 0$ **then**
    Go to the next position with lowest potential,
    else if $Flag = 1$ **then**
      Change the cell weight and treat the cells as there is obstacle there
      Update the potential of each cell,
    end
  end
  if *the UAV is trapped in a cell or visit specific cells multiple times* **then**
    $Flag = 1$,
    Search for the trapping point/points,
  end
  if *the UAV flee from the local minima* **then**
    $Flag = 0$,
  end
  $x_{t+1} \leftarrow$ cell nearby with lowest potential function.
  $t \leftarrow t + 1$
end

and $s = 4$ is a number of boundary face segments. As for repulsive force from an obstacle, the following equation was used:

$$p_{i,j} = \frac{p_{\max}}{1+g}, \tag{C.2}$$

where

$$\begin{aligned} g(x,y) = {} & (x_0 - l/2 - x) + |x_0 - l/2 - x| \\ & + (x - x_0 - l/2 + 1) + |x - x_0 - l/2 + 1| \\ & + (y_0 - l/2 - y) + |y_0 - l/2 - y| \\ & + (y - y_0 - l/2 + 1) + |y - y_0 - l/2 + 1|, \end{aligned} \tag{C.3}$$

here $p_{\max}$ is the maximum potential, (x_0, y_0) is the center coordinate of the obstacle and l is the side length of the obstacle. The potential at any cell in the environment is given by the maximum of the potentials due to individual cells. Furthermore, the attractive force generated by the goal is represented by Eq. (30).

$$P = C\sqrt{|x - x_{\text{goal}}|^2 + |y - y_{\text{goal}}|^2}, \tag{C.4}$$

and the potential in the environment is represented by $P = P_0 + P_g$, where

$$P_0 = \text{Max}\{p_i\}, \tag{C.5}$$

in which $i \in [1, \ldots, M]$ and M represents the number of obstacles.

It may be noted that these functions are continuous in nature. For the calculations, the center of each cell is considered and the UAV is constrained to travel only from one cell to the center of the cells connected to each other. For avoiding the local minima, the virtual obstacle is considered [40, 41]. The overview of the algorithm used for the path planning is shown in Algorithm 10.

References

[1] C. Goerzen, Z. Kong and B. Mettler, A survey of motion planning algorithms from the perspective of autonomous UAV guidance, *Journal of Intelligent and Robotic Systems* **57**(1–4) (2010) 65–100.

[2] J. Canny, *The Complexity of Robot Motion Planning* (MIT Press, Cambridge, 1988).

[3] J. T. Schwartz and C. K. Yap, *Advances in Robotics: Algorithmic and Geometric Aspects of Robotics*, Vol. 1 (Erlbaum Associates Inc, Hillsclale, NJ, 1986).

[4] J. C. Latombe, *Robot Motion Planning* (Kluwer Academic, Boston, MA, 1991).

[5] Y. K. Hwang and N. Ahuja, Gross motion planning — a survey. *ACM Comput. Surv.* **24**(3) (1992) 219–291.

[6] K. Fujimura, *Motion Planning in Dynamic Environments* (Springer Science & Business Media, New York, 2012).

[7] K. Tarabanis *et al.*, A survey of sensor planning in computer vision, *Robot. Autom. IEEE Trans.* **11**(1) (1995) 86–104.

[8] M. H. Overmars *et al.*, Coordinated motion planning for multiple car-like robots using probabilistic roadmaps, in *Robotics and Automation, 1995. Proc. 1995 IEEE Int. Conf.*, Vol. 2, (IEEE, 1995), pp. 1631–1636.

[9] S. M. LaValle, *Planning Algorithms* (Cambridge University Press, Cambridge, 2006).

[10] D. Ferguson, M. Likhachev and A. Stentz, A guide to heuristic-based path planning, in *Proc. Int. Workshop on Planning Under Uncertainty for Autonomous Systems, International Conf. Automated Planning and Scheduling (ICAPS)*, pp. 9–18, (2005).

[11] K. P. Valavanis, *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy*, Vol. 33 (Springer Science & Business Media, New York, 2008).

[12] L. Paull, S. Saeedi, M. Seto and H. Li, AUV navigation and localization: A review, *IEEE J. Oceanic Eng.* **39**(1) (2014) 131–149.

[13] H. Voos, Nonlinear state-dependent riccati equation control of a quadrotor UAV, in *Computer Aided Control System Design, 2006 IEEE Int. Conf. Control Applications, 2006 IEEE Int. Symp. Intelligent Control, 2006 IEEE*, (IEEE, 2006), pp. 2547–2552.

[14] O. Goldreich, Computational complexity: A conceptual perspective, *ACM SIGACT News* **39**(3) (2008) 35–39.

[15] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Vol. 3 (Pearson Education, London, 1998).

[16] T. H. Cormen, *Introduction to Algorithms* (MIT Press, Cambridge, 2009).

[17] S. Russell and P. Norvig, Artificial intelligence: A modern approach. 1995.

[18] K. L. Lim, L. S. Yeong, S. I. Ch'Ng, K. P. Seng and L.-M. Ang, Uninformed multigoal pathfinding on grid maps, in *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 Int. Conf.*, Vol. 3 (IEEE, 2014), pp. 1552–1556.

[19] R. E. Korf, Depth-first iterative-deepening: An optimal admissible tree search, *Artificial Intelligence* **27**(1) (1985) 97–109.

[20] Y. Björnsson, M. Enzenberger, R. C. Holte and J. Schaeffer, Fringe search: Beating A* at pathfinding on game maps, *CIG*, **5** (2005) 125–132.

[21] A. Felner, C. Moldenhauer, N. R. Sturtevant and J. Schaeffer, Single-frontier bidirectional search, in *AAAI*, 2010.

[22] L. Hongyun, J. Xiao and J. Hehua, Multi-goal path planning algorithm for mobile robots in grid space, in *Control and Decision Conf. (CCDC), 2013 25th Chinese*, (IEEE, 2013), pp. 2872–2876.

[23] O. Khatib and L. M. Mampey, Fonction decision-commande dun robot manipulateur, *Rep* **2**(7) (1978) 156.

[24] J. Borenstein and Y. Koren, Real-time obstacle avoidance for fast mobile robots, *IEEE Trans. Systems Man Cybern.* **19**(5) (1989) 1179–1187.

[25] J. Borenstein and Y. Koren, Real-time obstacle avoidance for fast mobile robots in cluttered environments, in *Robotics and Automation, 1990. Procs., 1990 IEEE Int. Conf.*, (IEEE, 1990), pp. 572–577.

[26] J. Borenstein and Y. Koren, The vector field histogram-fast obstacle avoidance for mobile robots, *IEEE Trans. Robot. Autom.* **7**(3) (1991) 278–288.

[27] A. Babinec, M. Dekan, F. Duchoň and A. Vitko, Modifications of vfh navigation methods for mobile robots, *Procedia Eng.* **48** (2012) 10–14.

[28] C. I. Connolly, J. B. Burns and R. Weiss, Path planning using Laplace's equation, in *Robotics and Automation, 1990. Proc. 1990 IEEE Int. Conf.*, (IEEE, 1990), pp. 2102–2106.

[29] S. Akishita, S. Kawamura and K. Hayashi, Laplace potential for moving obstacle avoidance and approach of a mobile robot, in *Proc. I990 Japan-USA Symposium on Flexible Automation*, A Pacific Rim Conference, pp. 139–142, Kyoto, Japan, 1990.

[30] C. Connolly *et al.*, A hamiltonian framework for kinodynamic planning and control, in *Robotics and Automation, 1995. Proc. 1995 IEEE Int. Conf.* Vol. 3 (IEEE, 1995), pp. 2746–2751.

[31] J. S. Zelek, Dynamic path planning, in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE Int. Conf.*, Vol. 2 (IEEE, 1995) pp. 1285–1290.

[32] M. W. Hirsch, *Differential Topology*, Vol. 33 (Springer Science & Business Media, New York, 2012).

[33] N. Ahuja and J.-H. Chuang, Shape representation using a generalized potential field model, *IEEE Trans. Pattern Anal. Mach. Intelli.* **19**(2) (1997) 169–176.

[34] M. C. Lee and M. G. Park, Artificial potential field based path planning for mobile robots using a virtual obstacle concept, in *Advanced Intelligent Mechatronics, 2003. AIM 2003. Proc. 2003 IEEE/ASME Int. Conf.*, Vol. 2 (IEEE, 2003), pp. 735–740.

[35] J. Barraquand, B. Langlois and J.-C. Latombe, Numerical potential field techniques for robot path planning, *IEEE Trans. Syst. Man Cybern.* **22**(2) (1992) 224–241.

[36] Y. K. Hwang and N. Ahuja, A potential field approach to path planning, *IEEE Trans. Robot. Autom.* **8**(1) (1992) 23–32.

[37] Y. Koren and J. Borenstein, Potential field methods and their inherent limitations for mobile robot navigation, in *Robotics and Automation, 1991. Proc, 1991 IEEE Int. Conf.* (IEEE, 1991), pp. 1398–1404.

[38] S. S. Ge and Y. J. Cui, Dynamic motion planning for mobile robots using potential field method, *Autonomous Robots* **13**(3) (2002) 207–222.

[39] Y. Kitamura, T. Tanaka, F. Kishino and M. Yachida, 3-d path planning in a dynamic environment using an octree and an artificial potential field, in *Intelligent Robots and Systems 95.* '*Human Robot Interaction and Cooperative Robots*', *Proc. 1995 IEEE/RSJ Int. Conf.*, Vol. 2 (IEEE, 1995), pp. 474–481.

[40] M. G. Park and M. C. Lee, A new technique to escape local minimum in artificial potential field based path planning, *KSME Int. J.* **17**(12) (2003) 1876–1885.

[41] G. Faria, R. A. F. Romero, E. Prestes and M. A. P. Idiart, Comparing harmonic functions and potential fields in the trajectory control of mobile robots, in *Robotics, Automation and Mechatronics, 2004 IEEE Conf.*, Vol. 2 (IEEE, 2004), pp. 762–767.

[42] K. Rosen, *Discrete Mathematics and Its Applications*, 7th edn. (McGraw-Hill Science, New York, 2011).

[43] B. Banerjee, A. Abukmail and L. Kraemer, Advancing the layered approach to agent-based crowd simulation, in *Proc. 22nd Workshop on Principles of Advanced and Distributed Simulation*, (IEEE Computer Society, 2008) pp. 185–192.

[44] E. Melachrinoudis and M. E. Helander, A single facility location problem on a tree with unreliable edges, *Networks* **27**(3) (1996) 219–237.

[45] S. Hougardy, The Floyd–Warshall algorithm on graphs with negative cycles, *Inform. Proc. Lett.* **110**(8) (2010) 279–281.

[46] T. M. Chan, More algorithms for all-pairs shortest paths in weighted graphs, *SIAM J. Comput.* **39**(5) (2010) 2075–2089.

[47] L. Szirmay-Kalos and G. Márton, Worst-case versus average case complexity of ray-shooting, *Comput.* **61**(2) (1998) 103–131.

[48] X. Zhang, Q. Wang, A. Adamatzky, F. T. S. Chan, S. Mahadevan and Y. Deng, A biologically inspired optimization algorithm for solving fuzzy shortest path problems with mixed fuzzy arc lengths, *J. Optim. Theory Appl.* **163**(3) (2014) 1049–1056.

[49] A. Sathyan, N. Boone and K. Cohen, Comparison of approximate approaches to solving the travelling salesman problem & its application to UAV swarming, *Int. J. Unmanned Syst. Eng* **3**(1) (2015) 1–16.

[50] C. M. Eaton, E. K. P. Chong and A. A. Maciejewski, Multiple-scenario unmanned aerial system control: A systems engineering approach and review of existing control methods, *Aerospace* **3**(1) (2016) 1.

[51] O. K. Sahingoz, Flyable path planning for a multi-uav system with genetic algorithms and bezier curves, in *Unmanned Aircraft Systems (ICUAS), 2013 Int. Conf.* (IEEE, 2013), pp. 41–48.

[52] U. Cekmez, M. Ozsiginan and O. K. Sahingoz, Adapting the ga approach to solve traveling salesman problems on cuda architecture, in *Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th Int. Symp.* (IEEE, 2013), pp. 423–428.

[53] Z. Cheng, Y. Sun and Y. Liu, Path planning based on immune genetic algorithm for uav, in *Electric Information and Control Engineering (ICEICE), 2011 Int. Conf.* (IEEE, 2011) pp. 590–593.

[54] Y. V. Pehlivanoglu, A new vibrational genetic algorithm enhanced with a voronoi diagram for path planning of autonomous UAV, *Aerospace Science and Technology* **16**(1) (2012) 47–55.

[55] T. Shima, S. J. Rasmussen, A. G. Sparks and K. M. Passino, Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms, *Computers & Oper. Res.* **33**(11) (2006) 3252–3269.

[56] M. Darrah, E. Fuller, T. Munasinghe, K. Duling, M. Gautam and M. Wathen, Using genetic algorithms for tasking teams of raven UAVs, *J. Intell. Robot. Syst.* **70**(1–4) (2013) 361–371.

[57] F. C. Allaire, M. Tarbouchi, G. Labonte and G. Fusina, FPGA implementation of genetic algorithm for UAV real-time path planning, *J. Intell. Robot. Syst.* **54**(1–3) (2009) 495–510.

[58] Y. Davidor, A genetic algorithm applied to robot trajectory generation, in *Handbook of Genetic Algorithms*, pp. 144–165, 1991.

[59] H.-S. Lin, J. Xiao and Z. Michalewicz, Evolutionary navigator for a mobile robot, in *Robotics and Automation, 1994. Proc, 1994 IEEE Int. Conf.*, (IEEE, 1994), pp. 2199–2204.

[60] H. H. Zhou and J. J. Grefenstette, Learning by analogy in genetic classifier systems, in *Proc. Third Int. Conf. Genetic algorithms*, (Morgan Kaufmann Publishers Inc., 1989), pp. 291–297.

[61] S. Handley, The genetic planner: The automatic generation of plans for a mobile robot via genetic programming, in *Intelligent Control, 1993, Proc. 1993 IEEE Int. Symp.* (IEEE, 1993), pp. 190–195.

[62] K. Sugihara and J. Smith, Genetic algorithms for adaptive planning of path and trajectory of a mobile robot in 2D terrains, *IEICE Trans. Inform. Syst.* **82**(1) (1999) 309–317.

[63] A. C. Nearchou, Path planning of a mobile robot using genetic heuristics, *Robotica* **16**(5) (1998) 575–588.

[64] Z. Cai and Z. Peng, Cooperative coevolutionary adaptive genetic algorithm in path planning of cooperative multi-mobile robot systems, *J. Intell. Robot. Syst.* **33**(1) (2002) 61–71.

[65] J. Chakraborty, A. Konar, L. C. Jain and U. K. Chakraborty, Cooperative multi-robot path planning using differential evolution, *J. Intell. Fuzzy Syst.* **20**(1, 2) (2009) 13–27.

[66] S. X. Yang, Y. Hu and M. Q. H. Meng, A knowledge based GA for path planning of multiple mobile robots in dynamic environments, in *Robotics, Automation and Mechatronics, 2006 IEEE Conf.* (IEEE, 2006), pp. 1–6.

[67] K. S. Senthilkumar and K. K. Bharadwaj, An efficient global optimization approach to multi robot path exploration problem using hybrid genetic algorithm, in *Information and Automation for Sustainability, 2008. ICIAFS 2008. 4th Int. Conf.* (IEEE, 2008), pp. 7–12.

[68] I. K. Nikolos, E. S. Zografos and A. N. Brintaki, UAV path planning using evolutionary algorithms, in *Innovations in Intelligent Machines-1* (Springer, New York, 2007), pp. 77–111.

[69] S. E. H. A. M. Zarbaf, M. Norouzi, R. J. Allemang, V. J. Hunt and A. Helmicki, Stay cable tension estimation of cable-stayed bridges using genetic algorithm and particle swarm optimization, *J. Bridge Eng.* **22**(10) (2017) 05017008.

[70] S. Mittal and K. Deb, Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms, in *Proc.*

Congress on Evolutionary Computation (CEC-2007), (Singapore) pp. 3195–3202. *Proc. Congress on Evolutionary Computation* (CEC-2007), (Singapore, 2007).

[71] D. Rathbun, S. Kragelund, A. Pongpunwattana and B. Capozzi, An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments, in *Digital Avionics Systems Conf. 2002. Proc. The 21st*, Vol. 2 (IEEE, 2002), pp. 8D2–1.

[72] M. Gemeinder and M. Gerke, GA-based search for paths with minimum energy consumption for mobile robot systems, in *Int. Conf. Computational Intelligence* (Springer, New York, 2001), pp. 599–607.

[73] M. Gemeinder and M. Gerke, GA-based path planning for mobile robot systems employing an active search algorithm, *Appl. Soft Comput.* **3**(2) (2003) 149–158.

[74] S.-Y. Fu, L.-W. Han, Y. Tian and G.-S. Yang, Path planning for unmanned aerial vehicle based on genetic algorithm, in *Cognitive Informatics & Cognitive Computing (ICCI* CC), 2012 IEEE 11th Int. Conf.* (IEEE, 2012), pp. 140–144.

[75] Y. Wang and W. Chen, Path planning and obstacle avoidance of unmanned aerial vehicle based on improved genetic algorithms, in *Control Conf. (CCC), 2014 33rd Chinese* (IEEE, 2014), pp. 8612–8616.

[76] A. Sonmez, E. Kocyigit and E. Kugu, Optimal path planning for UAVs using genetic algorithm, in *Unmanned Aircraft Systems (ICUAS), 2015 Int. Conf.* (IEEE, 2015), pp. 50–55.

[77] R. W. Beard *et al.*, Coordinated target assignment and intercept for unmanned air vehicles, *IEEE Trans. Robot. Autom.* **18**(6) (2002) 911–922.

[78] D. P. Bertsekas, J. N. Tsitsiklis and C. Wu, Rollout algorithms for combinatorial optimization, *J. Heuristics* **3**(3) (1997) 245–262.

[79] X. Tian, Y. Bar-Shalom, K. R. Pattipati, and A Sinha, Surveillance by multiple cooperative UAVs in adversarial environments, in *SPIE Defense and Security Symp.* (International Society for Optics and Photonics, 2008), pp. 69691E–69691E.

[80] X. Tian, Y. Bar-Shalom and K. R. Pattipati, Multi-step look-ahead policy for autonomous cooperative surveillance by UAVs in hostile environments, in *Decision and Control, 2008. CDC 2008. 47th IEEE Conf.* (IEEE, 2008), pp. 2438–2443.

[81] P. E. Hart, N. J. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.* **4**(2) (1968) 100–107.

[82] D. Delling, P. Sanders, D. Schultes and D. Wagner, Engineering route planning algorithms, J. Lerner, D. Wagner, K. A. Zweig, (eds.), in *Algorithmics of Large and Complex Networks* (Springer-Verlag, Berlin/Heidelberg, 2009), pp. 117–139.

[83] R. Dechter and J. Pearl, Generalized best-first search strategies and the optimality of A*, *J. ACM* **32**(3) (1985) 505–536.

[84] W. Zeng and R. L. Church, Finding shortest paths on real road networks: The case for A*, *Int. J. Geog. Inf. Sci.* **23**(4) (2009) 531–543.

[85] J. Yao, C. Lin, X. Xie, A. J. Wang and C.-C. Hung, Path planning for virtual human motion using improved A* star algorithm, in *Information Technology: New Generations (ITNG), 2010 Seventh Int. Conf.* (IEEE, 2010), pp. 1154–1158.

[86] Z. Xu, K.-S. Choi, Y.-G. Kim, J. An and S.-G. Lee, An enhanced formation of multi-robot based on A* algorithm for data relay transmission, in *International Conference in Swarm Intelligence* (Springer, Berlin, Heidelberg, 2011), pp. 91–98.

[87] Z. Zhao and R. Liu, An optimized method for A* algorithm based on directional guidance, in *Software Engineering and Service Science (ICSESS), 2015 6th IEEE Int. Conf.* (IEEE, 2015), pp. 986–989.

[88] B. M. ElHalawany, H. M. Abdel-Kader, A. TagEldeen, A. E. Elsayed and Z. B. Nossair, Modified A* algorithm for safer mobile robot navigation, in *Modelling, Identification & Control (ICMIC), 2013 Proc. Int. Conf.* (IEEE, 2013), pp. 74–78.

[89] R. Samar and A. Rehman, Autonomous terrain-following for unmanned air vehicles, *Mechatronics* **21**(5) (2011) 844–860.

[90] H. Cao, N. E. Brener and S. S. Iyengar, 3D large grid route planner for the autonomous underwater vehicles, *Int. J. Intell. Comp. Cybern.* **2**(3) (2009) 455–476.

[91] M. G. H. Bell, Hyperstar: A multi-path astar algorithm for risk averse vehicle navigation, *Transport. Res. B Meth.* **43**(1) (2009) 97–107.

[92] L. D. Filippis, G. Guglieri and F. Quagliotti, Path planning strategies for UAVs in 3D environments, *J. Intell. Robot. Syst.* **65**(1) (2012) 247–264.

[93] W. Zhan, W. Wang, N. Chen and C. Wang, Efficient UAV path planning with multiconstraints in a 3D large battlefield environment, *Math. Probl. Eng.* **2014** (2014) 1–12.

[94] S. Koenig, C. Tovey and Y. Smirnov, Performance bounds for planning in unknown terrain, *Artificial Intelligence* **147**(1) (2003) 253–279.

[95] A. Stentz, Optimal and efficient path planning for partially-known environments, in *Robotics and Automation, 1994. Proc. 1994 IEEE Int. Conf.* (IEEE, 1994), pp. 3310–3317.

[96] A. Stentz *et al.*, The focussed D* algorithm for real-time replanning, in *IJCAI* 95 (1995) 1652–1659.

[97] S. Koenig and M. Likhachev, D* lite, in *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, Edmonton, AB, Canada (2002), pp. 476–483.

[98] S. Koenig and M. Likhachev, Fast replanning for navigation in unknown terrain, *IEEE Trans. Robot.* **21**(3) (2005) 354–363.

[99] S. Koenig, M. Likhachev and D. Furcy, Lifelong planning A*, *Artificial Intelligence* **155**(1) (2004) 93–146.

[100] G. Ramalingam and T. Reps, An incremental algorithm for a generalization of the shortest-path problem, *J. Algorithms* **21**(2) (1996) 267–305.

[101] Y. Liu, S. Koenig and D. Furcy, Speeding up the calculation of heuristics for heuristic search-based planning, in *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, (2002), pp. 484–491.

[102] W. Burgard, M. Moors, C. Stachniss and F. E. Schneider, Coordinated multi-robot exploration, *IEEE Trans. Robot.* **21**(3) (2005) 376–386.

[103] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans and D. Lane, Path planning for autonomous underwater vehicles, *IEEE Trans. Robot.* **23**(2) (2007) 331–341.

[104] L. Jaillet, J. Cortés and T. Siméon, Sampling-based path planning on configuration-space costmaps, *IEEE Trans. Robot.* **26**(4) (2010) 635–646.

[105] A. Yahja, S. Singh and A. Stentz, An efficient on-line path planner for outdoor mobile robots, *Robotics and Autonomous Systems* **32**(2) (2000) 129–143.

[106] D. Ferguson and A. Stentz, Field D*: An interpolation-based path planner and replanner, in *Proc. of International Symposium on Robotics Research*, San Francisco, CA, Oct 12, 2005, pp. 239–253.

[107] R. Philippsen and R. Siegwart, An interpolated dynamic navigation function, in *Robotics and Automation, 2005. ICRA 2005. Proc. 2005 IEEE Int. Conf.* (IEEE, 2005), pp. 3782–3789.

[108] J. A. Sethian, A fast marching level set method for monotonically advancing fronts, *Proc. Nat. Acad. Sci.* **93**(4) (1996) 1591–1595.

[109] R. Philippsen, Motion planning and obstacle avoidance for mobile robots in highly cluttered dynamic environments, PhD thesis, École Polytechnique Fédérale De Lausanne, (2004).

[110] A. Kelly *et al.*, Toward reliable off road autonomous vehicles operating in challenging environments, *Int. J. Robot. Res.* **25**(5–6) (2006) 449–483.

[111] J. Carsten, D. Ferguson and A. Stentz, 3D field D*: Improved path planning and replanning in three dimensions, in *Intelligent Robots and Systems, 2006 IEEE/RSJ Int. Conf.* (IEEE, 2006), pp. 3381–3386.

[112] S. Hrabar, 3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs, in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ Int. Conf.* (IEEE, 2008), pp. 807–814.

[113] S. Perkins, P. Marais, J. Gain and M. Berman, Field D* path-finding on weighted triangulated and tetrahedral meshes, *Autonomous Agents and Multi-Agent Systems* **26**(3) (2013) 354–388.

[114] P. Yap, Grid-based path-finding, in *Conference of the Canadian Society for Computational Studies of Intelligence* (Springer, Berlin, Heidelberg, 2002), pp. 44–55.

[115] A. Botea, M. Müller and J. Schaeffer, Near optimal hierarchical path-finding, *Journal of Game Development* **1**(1) (2004) 7–28.

[116] A. Zelinsky, A mobile robot exploration algorithm, *IEEE Trans. Robot. Autom.* **8**(6) (1992) 707–717.

[117] T. Cazenave, Optimizations of data structures, heuristics and algorithms for path-finding on maps, in *Computational Intelligence and Games, 2006 IEEE Symp.* (IEEE, 2006), pp. 27–33.

[118] A. Nash, K. Daniel, S. Koenig and A. Felner, Theta*: Any-angle path planning on grids, in *Proc. Nat. Conf. Artif. Intell.*, Vol. 22 (2007), p. 1177.

[119] K. Daniel, A. Nash, S. Koenig and A. Felner, Theta*: Any-angle path planning on grids, *J. of Artificial Intell. Research* **39** (2010) 533–579.

[120] A. Nash, Any-angle path planning, University of Southern California (2012).

[121] A. Nash, S. Koenig and C. Tovey, Lazy Theta*: Any-angle path planning and path length analysis in 3D, in *Proceedings of the 24th AAAI Conference on Artificial Intelligence* (Menlo Park, CA: AAAI Press, 2010).

[122] A. Nash, S. Koenig and M. Likhachev, Incremental $\phi*$: Incremental any-angle path planning on grids, in *Proc. 21st Int. Joint Conf. Artifical Intelligence*, (Morgan Kaufmann Publishers Inc., 2009), pp. 1824–1830.

[123] S. Choi and W. Yu, Any-angle path planning on non-uniform cost-maps, in *Robotics and Automation (ICRA), 2011 IEEE Int. Conf.* (IEEE, 2011), pp. 5615–5621.

[124] F. Duchon, P. Hubinsky, A. Babinec, T. Fico and D. Hunady, Real-time path planning for the robot in known environment, in *Robotics in Alpe-Adria-Danube Region (RAAD), 2014 23rd Int. Conf.* (IEEE, 2014), pp. 1–8.

[125] J. Li, M. Dridi and A. El-Moudni, A cooperative traffic control of vehicle–intersection (ctcvi) for the reduction of traffic delays and fuel consumption, *Sensors* **16**(12) (2016) 2175.

[126] R. Bellman, Dynamic programming and lagrange multipliers, *Proc. Nat. Acad. Sci. USA* **42**(10) (1956) 767.

[127] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas and D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Vol. 1 (Athena Scientific Belmont, MA, 1995).

[128] N. L. Stokey, *Recursive Methods in Economic Dynamics* (Harvard University Press, Cambridge, 1989).

[129] R. Giegerich and C. Meyer, Algebraic dynamic programming, in *Algebraic Methodology and Software Technology* (Springer, New York, 2002), pp. 349–364.

[130] M. Sniedovich, Dijkstra's algorithm revisited: The dynamic programming connexion, *Control and Cybernetics* **35**(3) (2006) 599.

[131] M. Sniedovich, *Dynamic Programming: Foundations and Principles* (CRC Press, Boca Raton, 2010).

[132] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik* **1**(1) (1959) 269–271.

[133] T. J. Misa and P. L. Frana, An interview with Edsger W. Dijkstra, *Commun. ACM* **53**(8) (2010) 41–47.

[134] M. T. Goodrich and R. Tamassia, *Algorithm Design* (Wiely India, 2002).

[135] J. Edmonds and R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, *J. ACM* **19**(2) (1972) 248–264.

[136] D. B. Johnson, A note on Dijkstra's shortest path algorithm, *J. ACM* **20**(3) (1973) 385–388.

[137] B. Golden, Technical note — shortest-path algorithms: A comparison, *Oper. Res.* **24**(6) (1976) 1164–1168.

[138] C. Sommer, Shortest-path queries in static networks, *ACM Comput. Surv.* **46**(4) (2014) 45.

[139] B. Liu, S.-H. Choo, S.-L. Lok, S.-M. Leong, S.-C. Lee, F.-P. Poon and H.-H. Tan, Integrating case-based reasoning, knowledge-based approach and Dijkstra algorithm for route finding, in *Proc. Tenth Conf. Artificial Intelligence for Applications, 1994* (IEEE, 1994), pp. 149–155.

[140] M. Noto and H. Sato, A method for the shortest path search by extended dijkstra algorithm, in *IEEE Int. Conf. Syst. Man Cybern.* **3** (2000) 2316–2320.

[141] J. L. Solka, J. C. Perry, B. R. Poellinger and G. W. Rogers, Fast computation of optimal paths using a parallel Dijkstra algorithm with embedded constraints, *Neurocomputing* **8**(2) (1995) 195–212.

[142] Y. Li and W. Liu, Analysis of the shortest route in network on Dijkstra algorithm, *Microcomputer Applications* **3** (2004) 007.

[143] S. Pettie, V. Ramachandran and S. Sridhar, Experimental evaluation of a new shortest path algorithm, in *Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX)* (2002), pp. 126–142.

[144] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Vol. 6 (MIT Press, Cambridge, 2001).

[145] R. Bellman, On a routing problem, Technical report, DTIC Document (1956).

[146] G. Laporte, The vehicle routing problem: An overview of exact and approximate algorithms, *Eur. J. Oper. Res.* **59**(3) (1992) 345–358.

[147] G. Hajela and M. Pandey, A fine tuned hybrid implementation for solving shortest path problems using Bellman–Ford, *Int. J. Comput. Appl.* **99**(2) (2014) 29–33.

[148] Z. Chong and Z. Fan, Study on UAV path planning based on Bellman–Ford algorithm, *Journal of Projectiles, Rockets, Missiles and Guidance* **5** (2007) 077.

[149] M. J. Bannister and D. Eppstein, Randomized speedup of the Bellman–Ford algorithm, *ANALCO* **2012** (2012) 41–47.

[150] G. Hajela and M. Pandey, Parallel implementations for solving shortest path problem using Bellman–Ford, *Int. J. Comput. Appl.* **95**(15) (2014) 1–6.

[151] J. Kober, J. A. Bagnell and J. Peters, Reinforcement learning in robotics: A survey, *The International Journal of Robotics Research* **32**(11) (2013) 1238–1274.

[152] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-art*, Vol. 12 (Springer Science & Business Media, New York, 2012).

[153] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, Vol. 135 (MIT Press, Cambridge, 1998).

[154] R. S. Sutton, Learning to predict by the methods of temporal differences, *Machine Learning* **3**(1) (1988) 9–44.

[155] J. Peters, S. Vijayakumar and S. Schaal, Reinforcement learning for humanoid robotics, in *Proc. Third IEEE-RAS Int. Conf. Humanoid Robots (HUMANOIDS)* (2003), pp. 103–123.

[156] R. S. Sutton and A. G. Barto, *Proceedings of the IEEE-RAS international conference on humanoid robots (HUMANOIDS)*, Vol. 1 (MIT Press, Cambridge, 1998), pp. 103–123.

[157] H. Bou-Ammar, H. Voos and W. Ertel, Controller design for quadrotor UAVs using reinforcement learning, *InControl Applications (CCA), IEEE International Conference on 2010* (IEEE, 2010), pp. 2130–2135.

[158] D. Bauso, L. Giarré and R. Pesenti, Multiple UAV cooperative path planning via neuro-dynamic programming, in *43rd IEEE Conf. Decision and Control, 2004. CDC.* Vol. 1 (IEEE, 2004), pp. 1087–1092.

[159] B. Zhang, W. Liu, Z. Mao, J. Liu and L. Shen, Cooperative and Geometric Learning Algorithm (CGLA) for path planning of UAVs with limited information, *Automatica* **50**(3) (2014) 809–820.

[160] B. Zhang, Z. Mao, W. Liu and J. Liu, Geometric reinforcement learning for path planning of UAVs, *J. Intell. Robot. Syst.* **77**(2) (2015) 391–409.

[161] S. R. B. dos Santos, S. N. Givigi and C. L. Nascimento, Autonomous construction of structures in a dynamic environment using reinforcement learning, in *2013 IEEE Int. Systems Conf. (SysCon)* (IEEE, 2013), pp. 452–459.

[162] D. P. Bertsekas and J. N. Tsitsiklis, Neuro-dynamic programming: An overview, in *Proc. 34th IEEE Conf. Decision and Control, 1995,* Vol. 1 (IEEE, 1995), pp. 560–564.

[163] S. D. Whitehead, A complexity analysis of cooperative mechanisms in reinforcement learning, in *AAAI* (1991), pp. 607–613.

[164] C. Chen, H.-X. Li and D. Dong, Hybrid control for robot navigation — a hierarchical Q-learning algorithm, *Roboti. Autom. Mag.* 15(2) (2008) 37–47.

[165] Q. Zhang, M. Li, X. Wang and Y. Zhang, Reinforcement learning in robot path optimization, *J. Softw.* **7**(3) (2012) 657–662.

[166] J. N. Tsitsiklis, Asynchronous stochastic approximation and Q-learning, *Machine Learning* **16**(3) (1994) 185–202.

[167] H. P. Williams, *Model Building in Mathematical Programming* (John Wiley & Sons, New York, 2013).

[168] A. Bemporad and M. Morari, Control of systems integrating logic, dynamics, and constraints, *Automatica* **35**(3) (1999) 407–427.

[169] A. Richards and J. How, Mixed-integer programming for control, in *Proc. American Control Conf. 2005.* (IEEE, 2005), pp. 2676–2683.

[170] A. Chaudhry, K. Misovec and R. D'Andrea, Low observability path planning for an unmanned air vehicle using mixed integer linear programming, in *43rd IEEE Conf. Decision and Control, 2004. CDC.* Vol. 4 (IEEE, 2004), pp. 3823–3829.

[171] M. Alighanbari, Y. Kuwata and J. P. How, Coordination and control of multiple UAVs with timing constraints and loitering, in *Proc. American Control Conf. 2003.* Vol. 6 (IEEE, 2003), pp. 5311–5316.

[172] Y. Hao, A. Davari and A. Manesh, Differential flatness-based trajectory planning for multiple unmanned aerial vehicles using mixed-integer linear programming, in *Proc. American Control Conf.*, Vol. 1 (2005), p. 104.

[173] Y. Kim, D.-W. Gu and I. Postlethwaite, Real-time optimal mission scheduling and flight path selection, *IEEE Trans. Automa. Control* **52** (6) (2007) 1119–1123.

[174] C. Reinl and O. Von Stryk, Optimal control of multi-vehicle-systems under communication constraints using mixed-integer linear programming, in *Proc. 1st Int. Conf. Robot Communication and Coordination* (IEEE Press, 2007), p. 3.

[175] E. I. Grøtli and T. A. Johansen, Path planning for UAVs under communication constraints using SPLAT! and MILP, *J. Intelligent & Robotic Systems* **65**(1–4) (2012) 265–282.

[176] M. Radmanesh and M. Kumar, Flight formation of UAVs in presence of moving obstacles using fast-dynamic mixed integer linear programming, *Aerospace Science and Technology* **50** (2016) 149–160.

[177] M. Radmanesh, M. Kumar, A. Nemati and M. Sarim, Dynamic optimal UAV trajectory planning in the national airspace system via mixed integer linear programming, *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* **230**(9) (2015) 1668–1682.

[178] C. Branca and R. Fierro, A hierarchical optimization algorithm for cooperative vehicle networks, in *American Control Conf.* (IEEE, 2006).

[179] M. G. Earl and R. D'Andrea, Iterative MILP methods for vehicle-control problems, *IEEE Trans. Robot.* **21**(6) (2005) 1158–1167.

[180] M. A. Darrah, W. M. Niland and B. M. Stolarik, Multiple UAV dynamic task allocation using mixed integer linear programming in a SEAD mission, *Infotech@Aerospace* (2005) 26–29.

[181] M. Radmanesh, M. Kumar, P. H. Guentert and M. Sarim, Grey Wolf optimization based sense and avoid algorithm in a Bayesian framework for multiple UAV path planning in an uncertain environment, *Aerospace Science and Technology* **77** (2018) 168–179.

[182] M. Radmanesh and M. Kumar, Grey wolf optimization based sense and avoid algorithm for UAV path planning in uncertain environment using a Bayesian framework, in *2016 Int. Conf. Unmanned Aircraft Systems (ICUAS)* (IEEE, 2016), pp. 68–76.

[183] A. Yufka and O. Parlaktuna, Performance comparison of bug algorithms for mobile robots, in *Proc. 5th International Advanced Technologies Symposium (IATS'09)*, May 13–15, 2009, Karabuk, Turkey.

[184] M. Zohaib, M. Pasha, R. A. Riaz, N. Javaid, M. Ilahi and R. D. Khan, Control strategies for mobile robot with obstacle avoidance, *arXiv preprint arXiv:1306.1144* (2013).

[185] M. Zohaib, S. M. Pasha, N. Javaid, A. Salaam and J. Iqbal, An improved algorithm for collision avoidance in environments having u and h shaped obstacles, *Studies in Informatics and Control* **23**(1) (2014) 97–106.

[186] S. Quinlan and O. Khatib, Elastic bands: Connecting path planning and control, in *Proc. IEEE Int. Conf. Robotics and Automation, 1993.* (IEEE, 1993), pp. 802–807.

[187] I. Susnea, A. Filipescu, G. Vasiliu, G. Coman and A. Radaschin, The bubble rebound obstacle avoidance algorithm for mobile robots, in *2010 8th IEEE Int. Conf. Control and Automation (ICCA)* (IEEE, 2010), pp. 540–545.

[188] B. M. Sathyaraj, L. C. Jain, A. Finn and S. Drake, Multiple UAVs path planning algorithms: A comparative study, *Fuzzy Optimization and Decision Making* **7**(3) (2008) 257–267.

[189] N. Ernest, D. Carroll, C. Schumacher, M. Clark, K. Cohen and G. Lee, Genetic fuzzy based artificial intelligence for unmanned combat aerial vehicle control in simulated air combat missions, *J. Def. Manag*. **6**(144) (2016) 2167–0374.

[190] M. Radmanesh, M. Kumar and M. Sarim, On the effect of different splines on way-point navigation of quad-copters, in *ASME 2016 Dynamic Systems and Control Conf.* (American Society of Mechanical Engineers, 2016), pp. V001T05A004–V001T05A004.

[191] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners* (Springer Science & Business Media, New York, 2009).

[192] Q.-K. Pan, M. F. Tasgetiren and Y.-C. Liang, A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem, *Comput. Oper. Res.* **35**(9) (2008) 2807–2839.

[193] M. Radmanesh, P. H. Guentert, M. Kumar and K. Cohen, Analytical pde based trajectory planning for unmanned air vehicles in dynamic hostile environments, in *American Control Conf. (ACC), 2017* (IEEE, 2017), pp. 4248–4253.

[194] I.-C. Chang, H.-T. Tai, F.-H. Yeh, D.-L. Hsieh and S.-H. Chang, A VANET-Based A* route planning algorithm for travelling time-and energy-efficient GPS navigation app, *International Journal of Distributed Sensor Networks* **9**(7) (2013) 794521.

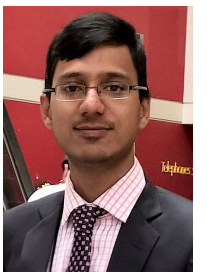
[195] C. Powers, D. Mellinger and V. Kumar, Quadrotor kinematics and dynamics, in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos (eds.) (Springer, Netherlands, 2014), pp. 307–328.

[196] A. Nemati and M. Kumar, Modeling and control of a single axis tilting quadcopter, in *American Control Conf., (ACC), 2014* (IEEE, 2014), pp. 3077–3082.

[197] R. Tan and M. Kumar, Proportional navigation (PN) based tracking of ground targets by quadrotor UAVs, in *ASME 2013 Dynamic Systems and Control Conf.* (American Society of Mechanical Engineers, 2013), pp. V001T01A004–V001T01A004.

Mohammadreza Radmanesh is a PhD candidate in the University of Cincinnati after receiving his masters in Mechanical Engineering and Aerospace Engineering in 2016 and 2018, respectively. He has published several journal and conference papers. His current research interests include multi-disciplinary decision-making, complex systems and control in large-scale systems, development of novel techniques for a collaborative control of UAVs, robotics, swarm systems, and multiple robot coordination and control. Mohammadreza is currently serving as a Lecturer in the University of Cincinnati. He is a member of ASME, AIAA and IEEE. Furthermore, he has coauthored a book titled, *Multi-Rotor Platform Based UAV Systems*.

Manish Kumar received his Bachelor of Technology degree in Mechanical Engineering from Indian Institute of Technology, Kharagpur, India in 1998, and his M.S. and Ph.D. degrees in Mechanical Engineering from Duke University, NC, USA in 2002 and 2004, respectively. After finishing his Ph.D., he worked as a postdoctoral research associate in the Department of Mechanical Engineering and Materials Science at Duke University from 2004 to 2005. In 2005, he received the Research Associateship Award from National Research Council (NRC). This award allowed him to work as a postdoctoral Research Associate with the Army Research Office, NC, USA from 2005 to 2007. As a part of his NRC Associateship program, he was a visiting scholar at General Robotics, Automation, Sensing, and Perception (GRASP) laboratory at the University of Pennsylvania, PA, USA. Subsequently, he worked as an Assistant Professor in the School of Dynamic Systems at the University of Cincinnati, OH, USA where he directed the Cooperative Distributed Systems (CDS) Laboratory and co-directed the Center for Robotics Research. He is currently an Associate Professor in the Department of Mechanical and Materials Engineering in University of Cincinnati. His current research interests include complex systems, decision-making and control in large-scale systems, development of novel techniques to fuse data from multiple sources, robotics, swarm systems, and multiple robot coordination and control. He is a member of American Society of Mechanical Engineers. In addition to coauthoring a book titled, *Ant Colony Inspired Robotic Swarm Systems*, in 2011, Kumar actively reviews multiple scholarly journals and also publishes his own papers for journals, conferences, and research offices. He also mentors many PhD and MS candidates and has served as the chair of multiple professional conferences and organizations as well as worked as a consultant for Edaptive Computing, Inc.



Mohammad Sarim is a PhD Candidate in the Department of Mechanical and Materials Engineering at the University of Cincinnati in Cincinnati, OH. He completed his Master of Technology and Bachelor of Technology in Mechanical Engineering from Aligarh Muslim University in India. He is currently working towards developing a hardware-based brain-inspired computing for robot learning problem. His research interests include computational neuroscience, machine learning, robotics, and controls.