RCS: a fast path planning algorithm for Unmanned Aerial Vehicles

Mohammad Reza Ranjbar Divkoti
   Ferdowsi University of Mashhad Faculty of Engineering

Mostafa Nouri-Baygi (✉ nouribaygi@um.ac.ir)
   Ferdowsi University of Mashhad Faculty of Engineering

Research Article

Keywords: Path planning, Unmanned aerial vehicles, Polygonal domains, Maximum turning angle, Minimum route leg length

Posted Date: February 2nd, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2503992/v1

License: ☺️ This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License
RCS: A FAST PATH PLANNING ALGORITHM FOR UNMANNED AERIAL VEHICLES

A PREPRINT

Mohammad Reza Ranjbar Divkoti
Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
mohammadreza.ranjbardivkoti@mail.um.ac.ir

Mostafa Nouri-Baygi*
Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
nouribaygi@um.ac.ir

April 18, 2020

ABSTRACT

Path planning is a major problem in autonomous vehicles. In recent years, with the increase in applications of Unmanned Aerial Vehicles (UAVs), one of the main challenges is path planning, particularly in adversarial environments. In this paper, we consider the problem of planning a collision-free path for a UAV in a polygonal domain from a source point to a target point. Based on the characteristics of UAVs, we assume two basic limitations on the generated paths: an upper bound on the turning angle at each turning point (maximum turning angle) and a lower bound on the distance between two consecutive turns (minimum route leg length).

We describe an algorithm that runs in \( O(n^2 \log n) \) time and finds a feasible path in accordance with the above limitations, where \( n \) is the number of obstacle vertices. As shown by experiments, the output of the algorithm is much close to the shortest path with this requirement (always below about 10% of the shortest path), in a much smaller time (up to 10000 quicker, in our experiments). We further demonstrate how to decompose the algorithm into two phases, a preprocessing and a query phase. In this way, given a fixed start point and a set of obstacles, we can preprocess a data structure of size \( O(n^2) \) in \( O(n^2 \log n) \) time, such that for any query target point we can find a path with the given requirements in \( O(n \log n) \) time. Finally, we modify the algorithm to find a feasible (almost shortest) path that reaches the target point within a given range of directions.

Keywords Path planning · Unmanned aerial vehicles · Polygonal domains · Maximum turning angle · Minimum route leg length

1 Introduction

An Unmanned Aerial Vehicle (UAV), is a flying machine without a human pilot. Due to their advantages in the military, civil and industrial markets, they are largely employed in situations dangerous for human life [1]. Since traditional remote piloting is not adequate for current complex missions, autonomous path planning are utilized extensively for UAVs [2].

Path planning is the problem of finding a feasible path from a given source to a given target for a moving object, that optimizes a desirable objective function and takes into account different constraints. The constraints are of various types, for example, non-accessible points in space, velocity and acceleration constraints, uncertainty about the environment, and so on. This problem in its generic form is NP-hard [3] and has polynomial time algorithms only in some special cases [4].

This problem has been extensively studied in robotics and various algorithms are presented for different forms of it. In general, the problem of path planning in robotics is more complex than in UAVs because of the high degrees of

*Corresponding author
freedom. For example, articulated robots can move their arms, or the shapes of some robots are too complex, and the robots must move in several directions to be able to pass through some regions in the environment (Piano Mover’s Problem [5]).

In contrast to robot path planning, UAV path planning has several characteristics. The three major differences in path planning for UAVs are as follows. First, due to the small size of the flying object compared to space, we usually model a UAV as a moving point in space. This makes the problem of path planning much simpler.

Second, because of the high speed of UAVs, it is not possible for a UAV to rotate by a sharp angle in its flight path [6]. This is completely different from a robot’s movement in which the robot can slow down or even stop to rotate. Because of this, the flight path of a UAV must be smooth and the path planner should avoid sharp turns. This limitation makes the path planning for UAVs different compared to the conventional path planning for robots and increases the complexity of the path planner. The complexity is magnified when the number of obstacles increases and the path planner may have to create a longer zigzag path with many rotations in order to make a smooth path. In addition to the limit on the turning angles, usually, a UAV cannot have two consecutive turns in a short time. This enforces the path planner to make enough distance between two successive turns [6].

The third difference in UAV path planning is the ability of a UAV to change its flight height. Robots usually move on a surface, so the problem of robot path planning is normally defined on the plane. But given the fact that UAVs fly in space, sometimes there is a need for a change in altitude to avoid obstacles (for example, a UAV flying over terrain, or a cruise missile over a sea near scattered islands). In these cases, the problem of designing a path should be solved in three-dimensional space which introduces new complexities [7].

In this paper, we take the first and second characteristics of UAVs path planning into account, and, as will be explained later, design a smooth path for a moving point among polygonal obstacles. However, in order to simplify explaining the algorithm in this paper, we choose the two-dimensional space. It is not too hard to generalize the approach to three-dimensional space [8].

The UAVs path planning algorithms are divided into two general categories, offline and online, based on the knowledge of the planner about the environment. A path planning algorithm is called offline if the designer has complete information about the environment and obstacles in it [9]. On the other hand, an online algorithm knows little or nothing at all about the environment in which the movement will take place [10, 11]. In the latter case, the path planning is done while the UAV moves, and based on the information we get from the sensors, the path is corrected. In this paper, we choose to work on offline algorithms.

In terms of the number of reported paths, the algorithms can be divided into three categories. In the first category, path planning is performed only for one device. Most algorithms are included in this category. In the second category, path planning is done simultaneously for a set of UAVs. In some applications, for example, a salvo attack against a warship, in order to prevent air defense systems activation, target missiles must simultaneously hit the same target from different angles. In this case, the path planner algorithm finds the best paths for several UAVs all at once [12, 13]. In the third category, path planning is performed for different targets, while the environment and the source point are fixed. This type of algorithm is useful when the target point is moving and we want to find the best time/place for the target to reach it, by taking advantage of the fixed environment to speed up the path planning.

In this paper, we first propose a path-planning algorithm for a single UAV. Later, with some changes, we split the algorithm into a preprocessing phase and a query phase in order to be able to find paths for multiple targets in a much shorter time.

In terms of the shape of the path, the algorithms are divided into two classes. In the first one, the output of the algorithm is composed of straight line segments that are connected together [14]. In the second class, the output consists of straight line segments and more complex curves such as arcs [7], Bezier curves [15] and B-Spline curves [16]. The advantage of the former is the simplicity of the path and the speed of the path planner, while the latter is better to adapt to the rotation limitations of UAVs and not having sharp angle rotations. In contrast, the latter algorithms have a high execution time and are rarely applied in practice.

With the assumptions for the path planner taken into account, we have both the simplicity and speed of the path planner as well as the smoothness of the path that UAVs can traverse.

The two limitations used in this paper were also previously considered by other researchers. Szczerba et al. [17] were the first who described these two limitations. They used a heuristic search algorithm to find a path in the two-dimensional space with the above limitations that has a length not more than a fixed given value and approaches the target from a desirable direction (approximately). The main problem with their algorithm, as will be explained in Section 6, is its running time. Later Zheng et al. [18] and Zheng et al. [6] considered the problem in the three-dimensional space and added two new limitations to the path, namely minimum flying height and maximum climbing/diving angle.
They used an evolutionary algorithm to solve the problem, which as Szczesna et al.’s algorithm [17] has a high running time when the complexity of the environment grows. He and Zhao [19] compared four different path planning algorithms, among which are A* and Dijkstra’s algorithms. Since we use a modified version of Dijkstra’s algorithm as the base of our algorithms, their results are especially of interest to us. As their experiments showed, Dijkstra’s algorithm, while achieving the shortest path, has the smallest running time.

When UAVs encounter complex and complicated missions, a higher level of intelligence is required. This makes self-learning and self-decision-making two essential prerequisites [20]. Recent advancements in machine learning techniques led to wide-ranging research in the field of UAV path planning. Machine learning techniques have adaptive learning ability, and they have been deeply studied and deployed in autonomous agents and unmanned systems [21, 22]. Depending on the interaction between an agent and the environment, reinforcement learning (a branch of machine learning) can improve the action policy that can be used in path planning [23].

Q-learning is a model-free, off-policy reinforcement learning algorithm that will find the best course of action, given the current state of the agent. Q-learning was first proposed by Watkins [24]. A number of researchers focus on solving the path planning problem using reinforcement learning [25]. Das et al. investigated an extended Q-learning algorithm [26] that has fast speed learning but has poor performance when faced with complex environments. Konar et al. presented an improved Q-learning algorithm [27]. The locked state is the main idea of this algorithm. There will be no exploration when the state is locked. This algorithm can be considered as an extension of Das et al.’s algorithm and has solved its problem.

In recent years, some surveys [28, 29, 30] overview the different UAV path planning approaches. There are a lot of UAV path planning works in these surveys. One of the new surveys on UAV path planning is in [31]. In this survey, Ait et al. proposed an overview of UAV path planning approaches. They classified these approaches into five main categories. These categories include classical methods [32, 33, 34], heuristics [35, 36, 37, 38, 39, 40], meta-heuristics [41, 42, 43], machine learning [44, 45, 46], and hybrid algorithms [47, 48, 49].

1.1 Contributions

We can summarize our contributions as follows:

- We present an offline, graph-based path planning algorithm, called RCS (stands for Regular Chains of Segments Path Planner). We first make two assumptions about the required path and then create a graph that models paths with those requirements, and finally apply a modified Dijkstra’s algorithm [50] on this graph to find the shortest feasible path from the source node to the target node. Because we find the shortest path with these properties, as we show in experimental results, the length of the result of the algorithm is much close to the length of the optimum path without the assumptions.

- We prove an upper bound on the time and space complexity of RCS. We show that if all inputs of the problem is given at the same time to the algorithm, we can solve the problem in \( O(n^3 \log n) \) time and \( O(n^2) \) space. In these upper bounds, \( n \) is the number of obstacle vertices.

- In another version of the problem, where several UAV’s need to be launched from a fixed source or we need to find the best time/path to reach a moving target point, we divide the RCS algorithm into two phases, such that given the set of obstacles and the source point, we preprocess a data structure of \( O(n^2) \) size in \( O(n^2 \log n) \) time, and for any query target point we can find the result in \( O(n \log n) \) time.

- We change the RCS algorithm in such a way as to find the best path that approaches the target point from a direction within a specified range of directions. The running time and the space usage of the algorithm do not change.

A preliminary version of this paper is presented at the 7th International Conference on Computer and Knowledge Engineering [51]. In this paper, the running time of the algorithm is extremely improved, from \( O(n^3 \log n) \) to \( O(n^2 \log n) \). Furthermore, we give a complete literature review, consider an extension of the problem in Section 5.2, and present a thorough evaluation of the algorithms in this paper.

The remaining paper is organized as follows. The problem is formally described in Section 2. In Section 3, we discuss the geometrical concepts needed to solve the problem. We present the algorithm to solve the problem in Section 4. The extensions of the algorithm are described in Section 5 and the experimental evaluations are presented in Section 6. Section 7 concludes the paper.
RCS: a fast path planning algorithm for
Unmanned Aerial Vehicles

Figure 1: Path planning in the presence of polygon obstacles, with connectedness, minimum route leg length, and maximum turning angle properties.

2 Problem Description

We here formally define the problem. In the plane, a set \( O = \{P_1, P_2, \ldots, P_h\} \) of \( h \) disjoint simple polygons, called obstacles, which totally have \( n \) vertices, a source point \( s \), a target point \( t \) and two constant values \( l \) and \( \alpha \) are given. Our goal is to find a path in the exterior space of polygons of \( O \), called the free space, from \( s \) to \( t \) with the following properties:

- **Minimum route leg length**: the path consists of straight line segments, each of length at least \( l \).
- **Connectedness**: the consecutive segments are connected at turning points.
- **Maximum turning angle**: the turning angles at the turning points are at most \( \alpha \).

From now on, we will refer to the above three properties as simply the path requirements. Figure 1 depicts a set \( O = \{P_1, P_2, P_3\} \) of three obstacles, with \( n = 13 \) vertices, source point \( s \) and target point \( t \). Each path from \( s \) to \( t \) is composed of a chain of segments. The illustrated path \( \{e_1, e_2, e_3\} \) in the figure satisfies the path requirements if \( l \leq |e_1|, |e_2|, |e_3| \) and \( \theta_1, \theta_2 \leq \alpha \).

3 Preliminaries

In this section, we introduce some geometric tools used for solving the problem. In Section 3.1 we define a regular chain of segments and in Section 3.2 the ray shooting data structure is briefly explained.

3.1 Regular chain of segments

In our problem we consider only paths consisting of a sequence of connected line segments. We call such a sequence a chain of segments. Each two consecutive segments have a common endpoint, called the turning point. The angle between a segment and the extension of the previous segment in the turning point is called the turning angle. (Figure 2)

We call a chain of segments with all turning angles equal to \( \alpha \) and equal leg lengths a regular chain of segments. Intuitively, it is part of a regular polygon with exterior angle \( \alpha \).

We here calculate the position of vertices of a regular chain of segments with different numbers of turning points, while the position of the start and the endpoints are known.

Let \( C \) denote a regular chain of segments with \( A_0 \) as the starting point, \( A_{k+1} \) as the endpoint, and \( k \) turning points. We assume the \( x \)-axis is the rightward horizontal line through the origin and the \( y \)-axis the upward vertical line through the origin. Let \( A_i, 1 \leq i \leq k \), be the \( i \)th turning point of \( C \). Let \( e \) denote the length of the segments of \( C \). For ease of illustration, we consider \( A_0 \) as an imaginary turning point and denote its previous point by \( A_{-1} \) (Figure 3). Let \( \beta \) denote the angle between the extension of \( A_{-1}A_0 \) and the \( x \)-axis. Since \( C \) is a regular chain of segments, the angle between the extension of \( A_{-1}A_0 \) and \( A_0A_1 \) is \( \alpha \). Let the angle between \( A_0A_1 \) and \( A_0A_{k+1} \) (the direct line segment from the starting point to the endpoint) be denoted by \( \theta \).
Figure 2: chain of segments from $p$ to $q$ with condition $l_1, l_2, l_3 \geq l, \theta_1, \theta_2 \leq \alpha$.

Figure 3: A regular chain of segments from point $A_0$ to $A_4$ with three turning points.

Draw a line from $A_0$ parallel to the $x$-axis and project $A_1$ on this line. Let $H$ denote the projection point (Figure 3). In the right triangle $\triangle A_0HA_1$ we have

$$x_{A_1} - x_{A_0} = |A_0A_1| \cos (\alpha + \beta),$$
$$y_{A_1} - y_{A_0} = |A_0A_1| \sin (\alpha + \beta).$$

Similarly, we can draw a horizontal line through $A_1$ and project $A_2$ on this line and get the following equations

$$x_{A_2} - x_{A_1} = |A_1A_2| \cos (2\alpha + \beta),$$
$$y_{A_2} - y_{A_1} = |A_1A_2| \sin (2\alpha + \beta).$$

As it follows, the position of each point $A_i$, can be calculated from the following recurrence relations

$$x_{A_i} - x_{A_{i-1}} = |A_{i-1}A_i| \cos (i\alpha + \beta),$$
$$y_{A_i} - y_{A_{i-1}} = |A_{i-1}A_i| \sin (i\alpha + \beta).$$

Since $e = |A_{i-1}A_i|$, for all $1 \leq i \leq k + 1$, we solve the above recurrence relations as follows

$$x_{A_j} = e \sum_{i=1}^{j} \cos (i\alpha + \beta) + x_{A_0},$$  \hspace{1cm} (1)
$$y_{A_j} = e \sum_{i=1}^{j} \sin (i\alpha + \beta) + y_{A_0},$$  \hspace{1cm} (2)

where $j = 1, \ldots, k + 1$.

In a simple polygon, the sum of the interior angles with $d$ vertices is equal to $(d - 2)\pi$. If we connect $A_0$ to $A_{k+1}$, we have a simple polygon. In this polygon, the interior angle at $A_0$ and $A_{k+1}$ are equal, because $C$ is part of a regular polygon. Let $d$ be the number of vertices of the polygon. Since $k$ is the number of turning points in $C$, we have $k = d - 2$ and so the sum of the interior angles of the simple polygon is $(d - 2)\pi = k\pi$. The interior angle of the
first and the last vertices in the chain is $\theta$ and the other angles are $\pi - \alpha$. Therefore we have $k(\pi - \alpha) + 2\theta = k\pi$, or equivalently $\theta = \frac{k\alpha}{2}$.

Let the angle between $\overline{A_0A_{k+1}}$ and the $x$-axis be denoted by $\gamma$. If we know the position of $A_0$ and $A_{k+1}$, we have the value of $\gamma$. Since $\gamma = \beta + \alpha + \theta$, we can rewrite this equation to have $\beta$ on the left hand side as $\beta = \gamma - (\alpha + \theta)$. Therefore, we have the value of $\beta$. If we further know the number of turning points of $C$, i.e. $k$, the length of segments of $C$ can be computed with each of the following equations

$$e = \frac{x_{A_{k+1}} - x_{A_0}}{\sum_{i=1}^{k+1} \cos(i\alpha + \beta)} = \frac{y_{A_{k+1}} - y_{A_0}}{\sum_{i=1}^{k+1} \sin(i\alpha + \beta)}.$$

Obtaining the value of $e$, the position of each vertex in the chain will be identified by Equations 1 and 2.

With the above method, given the starting and endpoints and the number of turning points, we can easily determine the position of the turning points of the regular chain of segments.

### 3.2 Ray shooting

One of the problems we encounter in path planning is recognizing whether a segment has any intersection with a set of segments. Formally, a set of segments $S = \{s_1, s_2, \ldots, s_n\}$ are given. We are asked if segment $t$ intersects at least one of the segments of $S$.

It is obvious that this problem could be solved by checking the intersection of $t$ with each segment of $S$ in $\Theta(n)$ total time. But, if the problem must be solved a lot of times for a fixed $S$, an algorithm with less query time is more favorable.

We use a ray-shooting algorithm for this problem. In the ray shooting problem, a set of segments $S$ is known. We need to preprocess $S$ such that upon receiving a query ray (its start point and direction), the first intersection position of the ray with the segments of $S$ will be detected quickly. We should also detect the case in which the ray does not intersect any segment of $S$.

By the solution of the ray shooting problem, the problem of segment intersection with a set of segments will be solved as follow: First, we preprocess $S$ for ray shooting. During the query time by receiving segment $t$, we shoot a ray from one end-point of $t$ in the direction of the other end-point. If the first intersection point of the ray with $S$ is between the two end-points of $t$, the answer to our solution is positive, that is, $t$ has an intersection with $S$. On the other hand, if the intersection point is not between the end-points of $t$ or there is no any intersection at all, the answer to our problem is negative.

There are many different algorithms for the ray shooting problem. But an old algorithm due to Chazelle [52] is well suited to our purpose. In this algorithm, after preprocessing in $O(n^2 \log n)$ time and using $O(n^2)$ space, the ray shooting queries are answered in $O(\log n)$ time.

### 4 RCS: Path Planning Algorithm

In this section, we describe our algorithm for finding a path with the given requirements. Since we prefer to make the path smooth, we try to spread out the break points along the path evenly. Therefore, we use regular chains of segments as sub-paths when moving from one obstacle vertex to another one, and so we call our algorithm Regular Chains of Segments Path Planner, or simply RCS.

Let $u$ and $v$ be two obstacle vertices. We want to find all sub-paths in the free space from $u$ to $v$ not passing through any other obstacle vertices. To do so, we examine all regular chains of segments with a different number of breakpoints starting at $u$ and ending at $v$. If all edges of a chain do not intersect any edges of the obstacles, then the chain is considered as a valid sub-path from $u$ to $v$. In other words, we first look at the regular chain of segments without any breakpoint, that is the straight line segment from $u$ to $v$. If the only edge of this chain does not intersect any obstacle then we consider this chain as a valid path from $u$ to $v$. Similarly, we examine the regular chain of segments with one breakpoint, two break points, and so on. We continue this process until the distance between two adjacent break points in the chain gets smaller than $t$. If the first and the last points of a regular chain are fixed, the distance between two consecutive break points decreases when the number of breakpoints increases (the last result of Section 3.1). Therefore there are a limited number of valid regular chains of segments from $u$ to $v$.

Assume the moving object follows a given regular chain of segments from $u$ to $v$ when traveling from $s$ to $t$. Because of the maximum turning angle property, when the object moves away from $v$, its leaving direction must be within a fixed range, determined by the direction it enters $v$. This observation is depicted in Figure 4. Following chain $c_0$,
where the object moves from \( u \) directly towards \( v \), it can leave \( v \) in the same direction and without any rotation at \( v \), or rotate by an angle at most equal to \( \alpha \), clockwise or counterclockwise. Therefore, the object leaves \( v \) in a direction in range \([\theta_1, \theta_2 = \theta_1 + 2\alpha]\). Similarly, for chain \( c_2 \) with seven break points, the valid leaving direction is in the range \([\phi_1, \phi_2 = \phi_1 + 2\alpha]; \phi_1 \) is the angle of the path if the object rotates at \( v \) by an angle equal to \( \alpha \) clockwise, and \( \phi_2 \) is the angle of the path when the object rotates by an angle equal to \( \alpha \) counter-clockwise. We call this range of directions for a regular chain of segments \( c \), the valid leave range of \( c \), and denote it by \( \text{VLR}(c) \).

4.1 Graph construction

The key idea of our approach is to construct a weighted directed multigraph \( G = (V_G, E_G) \), such that any path from the starting point \( s \) to the target point \( t \) in the original configuration, consisting only of regular chains of segments, corresponds to a path between analogous nodes in \( G \), of the same length. The problem will then reduce to finding the shortest path in the graph.

Graph \( G \) is constructed as follows. We add the starting point \( s \) and the target point \( t \) to the set of obstacle vertices. For each obstacle vertex \( v \), we add a corresponding node to \( G \). For each pair of nodes \( u, v \) of \( G \), and for each valid regular chain of segments \( c \) between the corresponding vertices, we add two directed edges, one from \( u \) to \( v \), and the other from \( v \) to \( u \). We add two directed edges because \( c \) does not have a direction and the object can follow \( c \) in both directions, and each edge represents one of the directions. We say edge \( e = (u, v) \) is incident from \( u \) and incident to \( v \), which means \( u \) (resp. \( v \)) is the head (resp. tail) of \( e \). The weight of \( e \), denoted by \( e.w \) is equal to the length of \( c \). In addition, we store a reference to \( c \), denoted by \( e.c \), for further use. It is possible to have more than one valid regular chain of segments between two obstacle vertices. So, there may be multiple edges between two nodes of \( G \), and as we indicated before, \( G \) is a multigraph.

**Lemma 1.** For a polygonal domain of \( n \) vertices, the number of nodes of \( G \), \( |V_G| \), is \( n + 2 \) and the number of edges, \( |E_G| \), is \( O(n^2) \). We can construct \( G \) in \( O(n^2 \log n) \) time.

**Proof.** For each obstacle vertex, we add an equivalent node to \( G \). Furthermore, we add two equivalent nodes for \( s \) and \( t \). Therefore, the number of nodes of \( G \) is \( n + 2 \).

For each pair of vertices \( u \) and \( v \), we have at most \( \frac{2\pi}{\alpha} \) valid regular chains of segments. This is because the number of breakpoints in a regular polygon with exterior angle \( \alpha \) is \( \frac{2\pi}{\alpha} \). For each valid regular chain of segments, two edges are added to \( G \). In summary, the maximum number of edges is \( \binom{n}{2} \cdot 2 \cdot \frac{2\pi}{\alpha} \) which is \( O(n^2) \).

For constructing the graph, we first preprocess the set of obstacle edges for ray shooting, as described in Section 3.2. There are \( n \) segments, and the preprocessing takes \( O(n^2 \log n) \) time. Then, for each pair of obstacle vertices, and each possible regular chain of segments between them, we see whether it is a valid regular chain. The validity test is done by checking the intersection of segments of the chain with obstacle edges. If any segment of the chain has an intersection with an obstacle edge, the chain is not valid. Otherwise, it is a valid chain and we add the corresponding edges to \( G \). There are \( \binom{n}{2} \cdot \frac{2\pi}{\alpha} \) possible chains, each consisting of at most \( 2\pi/\alpha \) segments, and executing a ray shooting query for each one takes \( O\left(\binom{n}{2} \cdot (2\pi/\alpha)^2 \log n\right) = O(n^2 \log n) \) total time.

The above bound for the number of edges of \( G \) is clearly the worst-case upper bound. This is because in most cases obstacle edges block regular chains and preclude them from being valid.

**Lemma 2.** Any path in the polygonal domain from vertex \( s \) to vertex \( t \) of length \( l \) that consists of only regular chains of segments has a corresponding path between the analogous nodes in graph \( G \) of length \( l \).
RCS: a fast path planning algorithm for Unmanned Aerial Vehicles

Proof. We construct $G$ such that any valid regular chain of segments from a vertex $u$ to a vertex $v$ has a corresponding edge between the analogous nodes in the graph with a weight equal to the length of the chain. Therefore, we can establish a one-to-one mapping between paths in the polygonal domain which consists of only regular chains of segments, and paths in $G$. \qed

4.2 Finding the shortest path in the graph

From Lemma 2, we conclude that if we need the shortest path between two vertices consisting of only regular chains of segments that satisfies some properties in the polygonal domain, we can look for the shortest path in $G$ between the analogous nodes with the same properties.

The best algorithm for finding the shortest path between two specific nodes in a weighted graph is Dijkstra’s algorithm [50], with $O(|E_G| + |V_G| \log |V_G|)$ running time using Fibonacci heap data structure. But the problem here is that Dijkstra’s algorithm is not directly applicable to $G$, because not all paths in $G$ are acceptable.

When the object moves on a path, because of the maximum turning angle property, it could not have sharp turns at obstacle vertices. Therefore, the angle between two consecutive regular chains of segments must be less than or equal to $\alpha$. Since in Dijkstra’s algorithm, edges are selected without any concern, the result of the algorithm may not satisfy the path requirements.

It is possible to transform $G$ into a new graph and apply Dijkstra’s algorithm to it to find the shortest path which satisfies the path requirements. But the large size of the new graph is another trouble. The graph has $O(|E_G|)$ vertices and $O(|\sum_{v \in V_G} \text{out-deg}^2(v))$ edges, where out-deg($v$) is the out-degree of $v$, i.e. the number of edges incident from $v$. If we apply Dijkstra’s algorithm on this graph, the running time of the algorithm will be $O(|E_G||V_G|)$ [53, 54], which in the worst case will lead to $O(n^3)$.

We here introduce an algorithm to find the shortest path with the path requirements, whose main idea is borrowed from Boroujerdi and Uhlmann [53]. In this algorithm, no new graph is generated and we only modify Dijkstra’s algorithm to find the shortest path with the given requirements. The running time of this algorithm will be $O(|E_G| \log |V_G|)$, which is $O(n^2 \log n)$ in the worst case.

Let $v$ be a node of $G$ and $e$ be an edge incident from $v$. Let $\text{angle}(e,v)$ denote the angle of the first segment of the regular chain of segments $e.c$ traversed when moving away from $v$. At the beginning of the algorithm, we sort the outgoing edges of each node $v$, according to their angles, in counter-clockwise order (see Figure 5). We store the sorted list in $v.out-edges$, a data structure capable of searching and deleting items in logarithmic time, like an AVL tree.

For each edge $e$ incident to $v$, when the object arrives at $v$ through $e.c$, only a subset of the outgoing edges of $v$ are applicable for departing from $v$. The angle of the first segment of these edges must be in the range $\text{VLR}(e.c)$. We denote these valid successors of $e$ by $\text{VS}(e)$. If the object comes to $v$ via $e.c$, it can leave $v$ following the chain of any edge in $\text{VS}(e)$, without violating the maximum turning angle requirement. It is easy to see that the edges of $\text{VS}(e)$ are placed in $v.out-edges$ consecutively. The following lemma gives an upper bound on the time needed to find $\text{VS}(e)$ for an edge $e$.

**Lemma 3.** Given a graph $G$ such that the outgoing edges of each vertex are sorted in counter-clockwise order based on the first edge of their chains, it is possible to find $\text{VS}(e)$ for an edge $e$ incident to $v$ in $O(\log(\text{out-deg}(v)) + |\text{VS}(e)|)$ time.
Proof. Let \( v \) be a vertex of \( G \), and \( e \) be an edge incident to \( v \). Let \( \text{VLR}(e.c) = [\theta_1, \theta_2] \). We can easily find the position of \( \theta_1 \) (resp. \( \theta_2 \)) in \( v.\text{out-edges} \) in \( O(|\log|\text{out-deg}(v)|) \) time. This is done by applying two binary searches with \( \theta_1 \) and \( \theta_2 \) as keys on \( v.\text{out-edges} \), and reporting all edges whose angles are between the keys. The binary search takes \( O(|\log|\text{out-deg}(v)|) \) time, and reporting edges in \( \text{VS}(e) \) sequentially takes \( O(|\text{VS}(e)|) \) time, which proves the claim. It is necessary to note that since the outgoing edges are arranged around \( v \) circularly, \( \theta_2 \) might be smaller than \( \theta_1 \). If this is the case, the report will consist of all edges whose angle is greater than \( \theta_1 \) or smaller than \( \theta_2 \). \( \square \)

We now describe the modified Dijkstra’s algorithm for this problem. Instead of processing nodes of the graph as in Dijkstra’s algorithm, we insert edges into a priority queue and process edges with minimum priority successively. For each edge \( e = (u, v) \), we assign a value \( \text{dist}(e) \), whose initial value is \( \infty \). This value denotes the length of the shortest path to \( v \), if the object starts from \( s \) and finally passes through \( e.c \). Furthermore, \( \text{dist}(e) \) plays the role of the priority in the priority queue used in the algorithm. We store another value \( \text{pred}(e) \), for each edge \( e \), which denotes the last edge before \( e \), in the shortest path to \( e \). The predecessor value is initially set to \( \text{null} \). This value helps to construct the shortest path at the end of the algorithm.

The algorithm starts by processing each edge \( e \) incident from \( s \), the starting point. We set \( \text{dist}(e) = e.w \), and insert \( e \) into the priority queue, \( Q \), with \( \text{dist}(e) \) as its priority. Now, as long as \( Q \) is not empty, we remove edge \( e \) with the minimum priority from \( Q \). If \( e \) is incident to \( t \), the target point, we are done, and report \( \text{dist}(e) \) as the shortest path distance from \( s \) to \( t \), and \( e \) and the sequence of predecessors as the shortest path. In the other case, where \( e \) is not incident to \( t \), but incident to some other node \( v \), we process edges of \( \text{VS}(e) \), one by one. For each edge \( e' \in \text{VS}(e) \), we remove \( e' \) from \( v.\text{out-edges} \), set \( \text{dist}(e') = \text{dist}(e) + e'.w \) and \( \text{pred}(e') = e \), and finally insert \( e' \) into \( Q \) with priority \( \text{dist}(e') \). We here remove \( e' \) from \( v.\text{out-edges} \) because we have found the shortest path to \( e' \) and \( \text{dist}(e') \) achieves its final value. Therefore, it cannot be updated furthermore, and it is not required to be checked for other segments incident to \( v \). This process continues and edges of \( G \) are explored successively until we arrive at \( t \). Because \( G \) is connected, we will eventually get into \( t \).

The outline of the RCS Path Planning algorithm is presented in Algorithm 1.

### 4.3 Analysis of the algorithm

In Lines 1-11, the graph is constructed, which takes \( O(n^2 \log n) \) time, as described before in Lemma 1. Lines 12-13 sort the outgoing edges of nodes and need \( O(\sum_{v \in V} \text{out-deg}(v) \log(|\text{out-deg}(v)|)) = O(|E_G| \log(|V_G|)) \) time. Lines 14-18 initialize the priority queue and process the outgoing edges of the starting point and take \( O(n \log s \log(\text{out-deg}(s))) \) time which is \( O(|V_G| \log |V_G|) \). In Lines 19-28, each edge of the graph is inserted into \( Q \) at most once, is removed from \( Q \) at most once, and removed from the list of the outgoing edges of its head at most once. These operations take \( O(|E_G| \log |V_G|) = O(|E_G| \log |V_G|) \) total time. Finding \( \text{VS}(e) \) for each edge \( e \), as proved in Lemma 3, takes \( O(|V_G| + |\text{VS}(e)|) \) time. Since each edge is considered only in one \( \text{VS}(e) \), and is removed from the list of outgoing edges of its head, the total time spent for lines 25-28 is \( O(|E_G| \log |V_G|) \). Therefore, the whole running time of the algorithm will be \( O(n^2 \log n) \), which is \( O(n^2 \log n) \). In the algorithm, the space is used for storing the ray shooting data structure, \( G \) and \( Q \), which is \( O(n^2 + |E_G|) = O(n^2) \).

The following theorem summarizes our results about finding the shortest path consisting of regular chains of segments.

**Theorem 1.** The RCS algorithm finds the shortest path from \( s \) to \( t \), which consists of regular chains of segments, and satisfies the path requirements, in \( O(n^2 \log n) \) time using \( O(n^2) \) space.

### 5 Extensions

In this section, we consider two extensions of the problem and try to use our technique to solve these extensions. In the first problem, there are several target points that are to be processed with a single starting point, and we have to find the best path to reach each target point. In the second extension, we solve the problem assuming that we have to arrive at the target point from a given range of directions.

#### 5.1 Problem in the query mode

In the original problem, it was assumed that the target point \( t \) is given at the same time as the other inputs. In this section, we divide the algorithm into two parts, a preprocessing phase, and a query phase. The set of obstacles and the starting point are given in the preprocessing phase, and in the query phase, the target point is determined. The problem in this mode is suitable when we want to find the best time/path to reach a moving target point in a fixed environment.
We solve the problem in the query mode as follows. Given the set of obstacles and the starting point in the preprocessing phase, we construct graph \( \mathcal{G} \) and run Dijkstra’s algorithm on \( \mathcal{G} \) to compute \( \text{dist}(e) \) for all edges of \( \mathcal{G} \). An edge \( e = (u, v) \) is a candidate predecessor for each edge \( e' \) incident from \( v \) if \( \text{angle}(e', v) \) lies in \( \text{VLR}(e.c) \). It is the actual predecessor of \( e' \), if, among all such candidates, \( \text{dist}(e) \) has the smallest value. With this in mind, for each node \( v \), we keep a sorted list of ranges around \( v \) such that the predecessor of edges in each range is fixed. We store this list, denoted by \( v.preds \), in an AVL tree, so that the insertion and searching take logarithmic time. This tree can easily be constructed, as Dijkstra’s algorithm is run on \( \mathcal{G} \). In the beginning, \( v.preds \) is empty for each node \( v \). When the first edge \( e \) incident to \( v \) is processed, \( \text{VLR}(e.c) \) is added to \( v.preds \), and \( e \) is linked to it. During the execution of the algorithm, when a new edge \( e' \) incident to \( v \) is being processed, it may add a new range to \( v.preds \). Since the edges are processed in increasing order of their distance, \( e' \) is predecessor only in a range where no previous predecessor is assigned. Furthermore, for each edge \( e \), \( \text{VLR}(e.c) \) covers a range of size \( \alpha \). Therefore, for each edge \( e \), if \( e \) is the predecessor for some range, it will be a connected range, which can easily be found and inserted to \( v.preds \) in \( O(\log n) \) time. Since the number of incoming edges to a vertex \( v \) is \( O(n) \), the number of ranges in \( v.preds \) is also \( O(n) \).

In the query phase, given the target point \( t \), we consider each node \( v \) in \( \mathcal{G} \) and find all valid regular chains of segments from \( v \) to \( t \). We here need to execute a ray shooting for each segment of a chain to see if it is valid. For each valid
chain, we add an edge \( e \) from \( v \) to \( t \) in \( G \). Furthermore, we find the predecessor of \( e \) using \( v.\text{preds} \) in \( O(\log n) \) time and accordingly assign \( \text{pred}(e) \). Finally, we add the distance of \( s \) to \( \text{pred}(e) \) to the length of \( [e.c] \) and store it into \( \text{dist}(e) \). After processing all edges to \( t \), the final solution is an edge \( e \) incident to \( t \) with the smallest value of \( \text{dist}(e) \), along with the sequence of predecessors.

**Theorem 2.** For the problem of path planning with the path requirements, we can preprocess a data structure in \( O(n^2 \log n) \) time using \( O(n^2) \) space, to answer the query for any target point \( t \) in \( O(n \log n) \) time. In these upper bounds, \( n \) is the total number of obstacle vertices.

**Proof.** We construct the graph in \( O(n^2 \log n) \) time using \( O(n^2) \) space and run Dijkstra’s algorithm in \( O(|E_G| \log |V_G|) \) time as proved in Lemma 1 and Theorem 1, respectively. In the query time, we need to find and process each edge to \( t \) in \( O(\log n) \) time to find a possible path to \( t \). Since there are at most \( O(n) \) edges to \( t \), processing all edges and choosing the shortest path to \( t \) takes \( O(n \log n) \) time. \( \square \)

### 5.2 Approaching from a given direction

In some path planning applications, it is desirable to reach the target from a given direction. For example, the most damage will take place when a missile hits the target orthogonally, or a UAV landing on an aircraft carrier needs to approach it from a fixed direction. In this section, we describe how we can use the method to achieve this goal.

In this version of the problem, in addition to the previous inputs, a range of directions like \( \Theta = [\theta_1, \theta_2] \) is given, and we want to approach the target from a direction in the range \( \Theta \).

In Section 4 we described how to construct a graph \( G \) that models paths only consisting of regular chains of segments. Each edge \( e = (u, v) \) in this graph has the following meaning: There exists a regular chain of segments \( e.c \) from obstacle vertex \( u \) to obstacle vertex \( v \) which leaves \( u \) and arrives at \( v \) in a fixed direction determined by \( e.c \).

Therefore, the solution is as follows. We continue the while loop of the algorithm until the edge that is processed comes into \( t \) with an angle in the range \( \Theta \), or \( Q \) becomes empty. It is obvious that in the former case the resulting path reaches \( t \) in a direction in the desirable range, and in the latter case, there is no path with the given requirements.

By the above argument we conclude the following theorem:

**Theorem 3.** In the problem of path planning with a fixed approaching direction, a path from the starting point \( s \) to the target point \( t \) with the required properties can be found in \( O(n^2 \log n) \) time using \( O(n^2) \) space, where \( n \) is the total number of obstacle vertices.

Furthermore, we can preprocess a data structure in \( O(n^2 \log n) \) time using \( O(n^2) \) space, to answer the query version of the problem for any target point \( t \) in \( O(n \log n) \) time.

### 6 Algorithm Evaluation

Even though the running time and the space usage of the proposed algorithms are bearable, the dependency in the upper bounds is on the number of obstacle vertices and not on the scene size. In most applications, the area in which the algorithm works is extremely large. But the complexity of the scene in terms of the number of obstacle vertices is not considerable. This can be made more rigorous if we apply a simplification step on the input to decrease the number of obstacle vertices.

In this section, we perform 6 experiments to evaluate the RCS algorithm and show its effectiveness in practice. We compare the results to the results of the algorithms designed by Śczerba et al. [17] and Lim et al. [55]. We chose the former, because they solved the most similar problem to the problem considered in this paper, and the latter, to compare our results with some newer algorithms. The algorithms selected for comparison are based on the A* search algorithm, and we called them A*, WA*, and DH hereinafter. The evaluations are performed based on several test cases. The source codes of the RCS, A*, WA*, and DH algorithms are available at the GitHub repository hosting service [56, 57, 58].

In the first experiment, we run RCS, A*, WA*, and DH on a fixed complex polygonal domain from a given source point to several target points. The configuration is depicted in Figure 6. In this figure, \( s \) is the source point and \( t_1, \ldots, t_5 \) are five different target points. We use the maximum turning angle \( \alpha = \pi/6 \) and the minimum leg length \( l = 50 \) as the parameters for this test case. The target points are selected such that paths to each one have a different number of turning points.
RCS: a fast path planning algorithm for
Unmanned Aerial Vehicles

Figure 6: The scene used for the first experiment with $\alpha = \pi/6$ and $l = 50$. The source point $s$ and five different
target points $t_1, \ldots, t_5$ are denoted in the figure.

Table 1: Comparison of the results of the RCS, A*, WA*, and DH algorithms on the configuration of Figure 6.

| Target | RCS preproc. | RCS query | RCS total | A* | WA* | DH | RCS | A* | WA* | DH | RCS - A* | RCS - WA* | RCS - DH |
|--------|--------------|-----------|-----------|----|-----|----|-----|----|-----|----|---------|----------|---------|
| $t_1$  | 89.20        | 4.77      | 93.97     | 77.02 | 36.35 | 93.38 | 533.18 | 533.27 | 535.76 | 534.49 | 0.0%     | -0.5%    | -0.2%   |
| $t_2$  | 89.67        | 3.90      | 93.57     | 3460.86 | 77.02 | 8703.68 | 749.01 | 753.82 | 834.81 | 807.84 | -1.2%   | -10.3%   | -7.3%   |
| $t_3$  | 89.86        | 4.26      | 94.12     | 5747.98 | 77.02 | 8703.68 | 978.11 | 976.76 | 1031.58 | 1042.23 | 0.1%    | -5.2%    | -6.2%   |
| $t_4$  | 89.05        | 4.80      | 93.85     | 258215 | 77.02 | 8703.68 | 1740.42 | 1577.96 | 1544.24 | 1526.62 | 9.3%    | 11.3%    | 12.3%   |
| $t_5$  | 89.57        | 5.41      | 94.98     | 331021 | 77.02 | 8703.68 | 2106.7 | 1835.59 | 1930.62 | 1909.68 | 12.9%   | 9.4%     | 8.4%    |

The numerical results are shown in Table 1, and the paths generated by the RCS, A*, WA*, and DH algorithms for
each target point are drawn in Figure 7(a-e). In each figure, the blue path, the black path, the green path, and the red
path are produced by RCS, A*, WA*, and DH, respectively.

The first important fact derived from the results is that the running time of the A*, WA*, and DH algorithms are much
larger than the RCS algorithm, especially for farther targets. For example, as it can be seen in Table 1, in this simple
test case, the running time of A*, WA*, and DH could be more than 10,000 times of the running time of RCS.

In the last three columns of Table 1, we show the relative differences between the length of paths generated by RCS
and the other algorithms. For example, The relative difference with A* is defined as $100 \cdot (l_{\text{RCS}} - l_{\text{A*}}) / \max\{l_{\text{RCS}}, l_{\text{A*}}\}$. In this formula, $l_{\text{RCS}}$ and $l_{\text{A*}}$ are the length of the paths generated by RCS and A*, respectively. We used $l_{\text{WA*}}$ and $l_{\text{DH}}$ instead of $l_{\text{A*}}$ for WA* and DH, respectively. As it can be seen, the relative differences between the length of paths produced by RCS, and WA* and DH are negative, most of the time, that is, RCS found a shorter path. In the case
of A*, the relative differences are small (less than 7%), which means the paths found by RCS are not much longer
than the paths found by A*. Taking this into account and the fact that the running time of A*, WA*, and DH are huge
compared to RCS, the effectiveness of our method becomes apparent.

Another outcome of this experiment is that as the number of obstacle vertices around which the path must turn in-
creases, the running time of A*, WA*, and DH grows, whilst the running time of RCS is roughly constant. The
increase in the running time of A*, WA*, and DH might also be due to the increase in the path length, but as we will
see in the subsequent experiments, the impact of this increase is negligible here.

In the second experiment, we intend to see the effect of changing the maximum turning angle parameter on the
performance of the algorithms. We used a fixed simple scene of size $800 \times 800$ consisting of a line segment as the
only obstacle and two points as the starting and the target points. We chose $l = 50$ as the minimum leg length, and 4
different values for the maximum turning angle, namely $\alpha = 80^\circ, 40^\circ, 20^\circ, 10^\circ$.

The numerical results are shown in Table 2 and the paths generated by RCS, A*, WA*, and DH for each value of $\alpha$
are drawn in Figure 8(a-d). As it can be seen from the table, the running time of the A*, WA*, and DH algorithms
slowly decrease as the maximum turning angle is reduced. On the other hand, the running time of RCS is not much
dependent on the value of $\alpha$ and is less than the running time of A*, WA*, and DH algorithms. Please note that in
general, the running time of RCS is not fixed when the maximum turning angle is changing. On the other hand, the
value of the maximum turning angle is inversely proportional to the running time of RCS (Lemma 1).
Figure 7: Demonstration of paths generated by the RCS (blue), A* (black), WA* (green), and DH (red) algorithms from $s$ to $t_1, \ldots, t_5$. 
Figure 8: Demonstration of paths generated by the RCS (blue), A* (black), WA* (green), and DH (red) algorithms from $s$ to $t$ for different values of the maximum turning angle.
We carry out the next experiment to see the effect of changing the minimum leg length parameter on the performance of the algorithms. We used a fixed simple scene of size $800 \times 800$ consisting of a line segment as the obstacle and two points as the starting and the target points. We chose $\alpha = \pi/6$ as the maximum turning angle, and 4 different values for the minimum leg length, namely $l = 40, 20, 10, 5$.

The numerical results are shown in Table 3 and the paths generated by RCS, A*, WA*, and DH for each value of $l$ are drawn in Figure 9(a-d). As it can be seen from the figures, the paths generated by RCS are all the same, which is the shortest path with the given requirements. Additionally, when the minimum leg length decreases, the running times of the other algorithms, particularly the A* algorithm, increase. However, the running time of RCS is not much dependent on the value of $l$. Please note that in general, the running time of RCS is not independent of the value of $l$. On the other hand, the value of the maximum turning angle is inversely proportional to the running time of RCS, because the number of edges between two vertices increases, when $l$ is reduced.

In the fourth experiment, we want to illustrate the effect of increasing the complexity of the scene in terms of the number of vertices of the obstacles. In this experiment, each scene consists of a fixed starting point and a fixed target point, and a set of rod-shaped obstacles. We double the number of obstacles in each scene. The maximum turning angle is selected as $\alpha = \pi/6$, and the minimum leg length as $l = 30$.

The results are shown in Table 4 and Figure 10(a-c). From the results of the table, we see that the dependency of the running time of RCS to the complexity of the scene (the number of vertices of the obstacles) is almost quadratic, while the running time of A*, WA*, and DH are not much dependent to the complexity of the scene. Despite that, the query time of RCS is still less than the running time of A*.

In the fifth experiment, we show the effect of rescaling the workspace on the running time of the algorithms. As illustrated in Figure 11 the scene consists of a fixed starting and a target point and a single rod-shaped obstacle. In each test case, we rescale the scene by doubling each coordinate, as well as the minimum leg length $l$. The maximum turning angle is selected as $\alpha = \pi/6$.

The results are shown in Table 5. From these results, we can easily conclude that the running time of A*, WA*, and DH is dependent on the size of the scene, particularly A*, while the running time of RCS is independent.

For the last experiment, we want to test the scalability of the RCS algorithm. We use six different polygonal domains of various sizes, and run the algorithm to find the required preprocessing and query time of the algorithm for each one.
Figure 9: Demonstration of paths generated by the RCS (blue), A* (black), WA* (green), and DH (red) algorithms from $s$ to $t$ for different values of the minimum leg length.
Figure 10: Demonstration of paths generated by the RCS (blue), A* (black), WA* (green), and DH (red) algorithms from $s$ to $t$ for scenes of different complexities.
Figure 11: Demonstration of paths generated by the RCS (blue) and A* (red) algorithms from $s$ to $t$ for scenes of different sizes.
Table 5: Illustration of the effect of changing the size of the scene.

| Test case | Scene size | l | RCS | preproc. | RCS | query | RCS | total | A* | WA* | DH | RCS | A* | WA* | DH | RCS - A* | RCS - WA* | RCS - DH |
|-----------|------------|---|-----|---------|-----|-------|-----|-------|----|-----|----|-----|----|-----|----|---------|-----------|---------|
| t1        | 100 × 100  | 10| 2.65| 1.33    | 3.98| 7.98  | 5.16| 10.19 | 101.37| 98.96| 102.38| 102.38| 2.4%| -1.0%| -1.0%|
| t2        | 200 × 200  | 20| 1.38| 2.96    | 4.34| 40.82 | 6.81| 10.48 | 236.23| 236.84| 265.78| 265.15| -0.3%| -11.1%| -10.9%|
| t3        | 400 × 400  | 40| 1.10| 2.08    | 3.18| 166.75| 8.89| 20.36 | 522.43| 519.26| 588.28| 564.50| 0.6%| -11.2%| -7.5%|
| t4        | 800 × 800  | 80| 1.06| 1.98    | 3.04| 1166.33| 14.14| 23.03 | 1093.44| 1083.79| 1180.17| 1161.50| 0.9%| -7.3%| -5.9%|

Table 6: Preprocessing and query time of the RCS algorithm for scenes of various complexities.

| Test case | #Obstacles | #Vertices | Preprocessing time (ms) | Query time (ms) |
|-----------|------------|----------|-------------------------|----------------|
| 1         | 1          | 3        | 2                       | 3              |
| 2         | 2          | 7        | 25                      | 3              |
| 3         | 9          | 39       | 1166                    | 18             |
| 4         | 26         | 100      | 2033                    | 26             |
| 5         | 84         | 300      | 28860                   | 188            |
| 6         | 180        | 500      | 223134                  | 570            |

Figure 12: A polygonal domain with 9 obstacles and 39 vertices is illustrated in which we find a path with $\alpha = \pi/18$ and $l = 60$. In these test cases, we choose the maximum turning angle $\alpha = \pi/9$ and the minimum leg length $l = 60$. For each test case, the number of obstacles and vertices, and the preprocessing and query time in milliseconds are shown in Table 6. One of the configurations used in this experiment is depicted in Figure 12. In this figure, a polygonal domain with 9 obstacles and 39 vertices is illustrated. The result of the algorithm on this configuration is shown when $\alpha = \pi/18$ and $l = 60$.

7 Conclusion

In this paper, we studied the problem of path planning for a robot with a maximum turning angle and minimum leg length in a polygonal domain. We first proposed an algorithm to find a path avoiding obstacles with the aforementioned requirements. We then proved that the required space and the running time of the algorithm are $O(n^2)$ and $O(n^2 \log n)$, respectively. In order to decrease the path planning time, we further decomposed the algorithm into two parts, the preprocessing phase, and the query phase. This helps to reduce the running time of the algorithm when a set of target points are going to be used in a fixed polygonal domain. In another extension, we changed the algorithm to answer path-planning queries that require the object to approach the target from a given direction (or a range of directions).

Although our main goal was not to find the shortest path, as presented by the experiments, the solution is not much longer than the shortest path. Furthermore, compared to most of the previous work on this problem, which uses heuristic and evolutionary methods, the running time of the algorithm is much less, and the algorithm is greatly scalable to large complex environments. For future work, we try to improve the algorithm and find the shortest path with the given requirements.
Compliance with Ethical Standards

- **Funding:** This study was NOT funded
- **Conflict of Interest:** Author Mohammad Reza Ranjbar Divkoti declares that he has no conflict of interest. Author Mostafa Nouri Baygi declares that he has no conflict of interest.
- **Ethical approval:** This article does not contain any studies with human participants performed by any of the authors.

References

[1] Bruce Enderle. Commercial applications of UAV's in japanese agriculture. In *AIAA 1st UAV Conference*, pages 2002–3400, 2002.

[2] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65, 2010.

[3] John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987.*, 28th Annual Symposium on, pages 49–60. IEEE, 1987.

[4] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[5] John H Reif. Complexity of the mover’s problem and generalizations. In *Foundations of Computer Science, 1979.*, 20th Annual Symposium on, pages 421–427. IEEE, 1979.

[6] Changwen Zheng, Lei Li, Fanjiang Xu, Fuchun Sun, and Mingyue Ding. Evolutionary route planner for unmanned air vehicles. *IEEE Transactions on Robotics*, 21(4):609–620, 2005.

[7] Vincent Roberge, Mohammed Tarbouchi, and Gilles Labonté. Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. *IEEE Transactions on Industrial Informatics*, 9(1):132–141, 2013.

[8] Halit Ergezer and Kemal Leblebicioglu. 3d path planning for multiple uavs for maximum information collection. *Journal of Intelligent & Robotic Systems*, 73(1):737–762, 2014.

[9] Qian Xue, Peng Cheng, and Nong Cheng. Offline path planning and online replanning of UAVs in complex terrain. In *Guidance, Navigation and Control Conference (CGNCC), 2014 IEEE Chinese*, pages 2287–2292. IEEE, 2014.

[10] Naifeng Wen, Lingling Zhao, Xiaohong Su, and Peijun Ma. UAV online path planning algorithm in a low altitude dangerous environment. *IEEE/CAA Journal of Automatica Sinica*, 2(2):173–185, 2015.

[11] Naifeng Wen, Xiaohong Su, Peijun Ma, Lingling Zhao, and Yanhang Zhang. Online UAV path planning in uncertain and hostile environments. *International Journal of Machine Learning and Cybernetics*, 8(2):469–487, 2017.

[12] Peng Yao, Honglun Wang, and Zikang Su. Cooperative path planning with applications to target tracking and obstacle avoidance for multi-UAVs. *Aerospace Science and Technology*, 54:10–22, 2016.

[13] Arman Nedjati, Gokhan Izbirak, Bela Vizvari, and Jamal Arkat. Complete coverage path planning for a multi-UAV response system in post-earthquake assessment. *Robotics*, 5(4):26, 2016.

[14] Yangguang Fu, Mingyue Ding, and Chengping Zhou. Phase angle-encoded and quantum-behaved particle swarm optimization applied to three-dimensional route planning for UAV. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(2):511–526, 2012.

[15] Liang Yang, Dailei Song, Jizhong Xiao, Jianda Han, Liying Yang, and Yang Cao. Generation of dynamically feasible and collision free trajectory by applying six-order bezier curve and local optimal reshaping. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 643–648. IEEE, 2015.

[16] Shashi Mittal and Kalyanmoy Deb. Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3195–3202. IEEE, 2007.

[17] Robert J Szczepański, Peggy Galkowski, Ira S Glickstein, and Noah Ternullo. Robust algorithm for real-time route planning. *IEEE Transactions on Aerospace and Electronic Systems*, 36(3):869–878, 2000.

[18] Changwen Zheng, Mingyue Ding, and Chengping Zhou. Real-time route planning for unmanned air vehicle with an evolutionary algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(01):63–81, 2003.
[19] Z. He and L. Zhao. The comparison of four UAV path planning algorithms based on geometry search algorithm. In 2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), volume 2, pages 33–36, 2017.

[20] Zhengyang Cui and Yong Wang. Uav path planning based on multi-layer reinforcement learning technique. IEEE Access, 9:59486–59497, 2021.

[21] Su Yeon Choi and Dowan Cha. Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art. Advanced Robotics, 33(6):265–277, 2019.

[22] Asharul Islam Khan and Yaseen Al-Mulla. Unmanned aerial vehicle in the machine learning environment. Procedia computer science, 160:46–53, 2019.

[23] Beomjoon Kim and Joelle Pineau. Socially adaptive path planning in human environments using inverse reinforcement learning. International Journal of Social Robotics, 8(1):51–66, 2016.

[24] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[25] William D Smart and L Pack Kaelbling. Effective reinforcement learning for mobile robots. In Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292), volume 4, pages 3404–3410. IEEE, 2002.

[26] Pradipta Kumar Das, Amit Konar, R Janarthanan, et al. Extended q-learning algorithm for path-planning of a mobile robot. In Asia-Pacific Conference on Simulated Evolution and Learning, pages 379–383. Springer, 2010.

[27] Amit Konar, Indrani Goswami Chakraborty, Sapam Jitu Singh, Lakhami C Jain, and Atulya K Nagar. A deterministic improved q-learning for path planning of a mobile robot. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 43(5):1141–1153, 2013.

[28] Yiying Zhao, Zheng Zheng, and Yang Liu. Survey on computational-intelligence-based uav path planning. Knowledge-Based Systems, 158:54–64, 2018.

[29] Mohammadreza Radmanesh, Manish Kumar, Paul H Guentert, and Mohammad Sarim. Overview of path-planning and obstacle avoidance algorithms for uavs: A comparative study. Unmanned systems, 6(02):95–118, 2018.

[30] Shubhani Aggarwal and Neeraj Kumar. Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges. Computer Communications, 149:270–299, 2020.

[31] Amylia Ait Saadi, Assia Soukane, Yassine Meraïhi, Asma Benmessaooud Gabis, Seyedali Mirjalili, and Amar Ramdane-Cherif. Uav path planning using optimization approaches: A survey. Archives of Computational Methods in Engineering, pages 1–52, 2022.

[32] Imane Nizar, Adil Jaafar, Zineb Hidila, Mohamed Barki, El Hossein Illoussamen, and Mohammed Mestari. Effective and safe trajectory planning for an autonomous uav using a decomposition-coordination method. Journal of Intelligent & Robotic Systems, 103(3):1–16, 2021.

[33] Xia Chen and Miaoyan Zhao. Collaborative path planning for multiple unmanned aerial vehicles to avoid sudden threats. In 2019 Chinese automation congress (CAC), pages 2196–2201. IEEE, 2019.

[34] Egidio D’Amato, Massimiliano Mattei, and Immacolata Notaro. Bi-level flight path planning of uav formations with collision avoidance. Journal of Intelligent & Robotic Systems, 93(1):193–211, 2019.

[35] Iago Z Biundini, Aurelio G Melo, Fabrício O Coelho, Leonardo M Honório, André LM Marcato, and Milena Faria Pinto. Experimentation and simulation with autonomous coverage path planning for uavs. Journal of Intelligent & Robotic Systems, 105(2):1–20, 2022.

[36] Guohao Zhang and Li-Ta Hsu. A new path planning algorithm using a gnss localization error map for uavs in an urban area. Journal of Intelligent & Robotic Systems, 94(1):219–235, 2019.

[37] Xia Chen, Miaoyan Zhao, and Liyuan Yin. Dynamic path planning of the uav avoiding static and moving obstacles. Journal of Intelligent & Robotic Systems, 99(3):909–931, 2020.

[38] Xueli Wu, Lei Xu, Ran Zhen, and Xiaoqing Wu. Bi-directional adaptive a* algorithm toward optimal path planning for large-scale uav under multi-constraints. IEEE Access, 8:85431–85440, 2020.

[39] Zhe Zhang, Jian Wu, Jiyang Dai, and Cheng He. A novel real-time penetration path planning algorithm for stealth uav in 3d complex dynamic environment. IEEE Access, 8:122757–122771, 2020.

[40] Zhe Zhang, Jian Wu, Jiyang Dai, and Cheng He. Optimal path planning with modified a-star algorithm for stealth unmanned aerial vehicles in 3d network radar environment. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering, 236(1):72–81, 2022.
[41] Samira Hayat, Evşen Yanmaz, Christian Bettstetter, and Timothy X Brown. Multi-objective drone path planning for search and rescue with quality-of-service requirements. *Autonomous Robots*, 44(7):1183–1198, 2020.

[42] Manh Duong Phung and Quang Phuc Ha. Safety-enhanced uav path planning with spherical vector-based particle swarm optimization. *Applied Soft Computing*, 107:107376, 2021.

[43] Xianjin Zhou, Fei Gao, Xi Fang, and Zehong Lan. Improved bat algorithm for uav path planning in three-dimensional space. *IEEE Access*, 9:20100–20116, 2021.

[44] Chao Yan, Xiaojia Xiang, and Chang Wang. Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments. *Journal of Intelligent & Robotic Systems*, 98(2):297–309, 2020.

[45] Ronglei Xie, Zhijun Meng, Yaoming Zhou, Yunpeng Ma, and Zhe Wu. Heuristic q-learning based on experience replay for three-dimensional path planning of the unmanned aerial vehicle. *Science Progress*, 103(1):0036850419879024, 2020.

[46] Ronglei Xie, Zhijun Meng, Lifeng Wang, Haochen Li, Kaipeng Wang, and Zhe Wu. Unmanned aerial vehicle path planning algorithm based on deep reinforcement learning in large-scale and dynamic environments. *IEEE Access*, 9:24884–24900, 2021.

[47] Gewen Huang, Yanguang Cai, Jianqi Liu, Yuanhang Qi, and Xiaozhou Liu. A novel hybrid discrete grey wolf optimizer algorithm for multi-uav path planning. *Journal of Intelligent & Robotic Systems*, 103(3):1–18, 2021.

[48] Zain Anwar Ali, Han Zhangang, and Wang Bo Hang. Cooperative path planning of multiple uavs by using max–min ant colony optimization along with cauchy mutant operator. *Fluctuation and Noise Letters*, 20(01):2150002, 2021.

[49] Yuwen Pan, Yuanwang Yang, and Wenzao Li. A deep learning trained by genetic algorithm to improve the efficiency of path planning for data collection with multi-uav. *IEEE Access*, 9:7994–8005, 2021.

[50] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[51] Mohammad Reza Ranjbar Divkoti and Mostafa Nouri-Baygi. Path planning in polygonal domains for robots with limited turning abilities. In *7th International Conference on Computer and Knowledge Engineering (ICCKE)*, 2017, pages 308–313, 2017.

[52] Bernard Chazelle. Filtering search: A new approach to query answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

[53] Ali Boroujerdi and Jeffrey Uhlmann. An efficient algorithm for computing least cost paths with turn constraints. *Information Processing Letters*, 67(6):317–321, 1998.

[54] D. [de Caen]. An upper bound on the sum of squares of degrees in a graph. *Discrete Mathematics*, 185(1):245–248, 1998.

[55] Dahee Lim, Jihoon Park, Dongin Han, Hwanchoel Jang, Wontae Park, and Daewoo Lee. Uav path planning with derivative of the heuristic angle. *International Journal of Aeronautical and Space Sciences*, 22(1):140–150, 2021.

[56] Mohammad Reza Ranjbar Divkoti. Optimal path planning algorithm. https://github.com/mrrranjbar/Optimal-Path-Planning-Algorithm, 2019.

[57] Mohammad Reza Ranjbar Divkoti. A* path planning. https://github.com/mrrranjbar/Astar-Path-Planning, 2019.

[58] Mohammad Reza Ranjbar Divkoti. Wa* and wa-dh path planning. https://github.com/mrrranjbar/WAStar-DH-Path-Planning, 2022.

**Declarations**

- Funding (Not applicable)
- Competing interests (Not applicable)
- Authors’ contributions:
  Mohammad Reza Ranjbar Divkoti (first author)
  Mostafa Nouri Baygi (second author) All authors contributed to the study’s conception and design. Material preparation, data collection, and analysis were performed by Mohammad Reza Ranjbar Divkoti and Mostafa Nouri Baygi. The first draft of the manuscript was written by Mohammad Reza Ranjbar Divkoti and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.
• Data Availability (Not applicable)