Unmanned Systems, Vol. 14, No. 4 (2026) 945–961

DOI: 10.1142/S2301385026500287

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

Joint Terrestrial-Aerial Path Planning for Tensegrone Robot

Songyuan Liu, Zhe Jing, Siyuan Hao, Jingshuo Lyu, Zichen Tao, Yun Gui, Hao Fang, Qingkai Yang*

School of Automation, Beijing Institute of Technology, Beijing 100081, P. R. China

This paper explores the joint terrestrial-aerial path planning challenge for the tensegrone robot, which combines a six-bar tensegrity structure with drone capabilities. Due to its unique construction, the tensegrone robot can both roll on the ground and fly in the air. To solve this joint path planning problem, we introduce a method based on probabilistic roadmaps. This approach leverages the topological properties of the feasible space and eliminates the necessity of calculating execution-specific trajectories. Moreover, to address the issue of achieving precise postures during the transition from rolling to flying mode, we propose a two-stage trajectory generation method. The effectiveness and robustness of the proposed methods are validated through simulation tests.

Keywords: Motion planning; terrestrial-aerial planning; tensegrity robot; UAV.

1. Introduction

With the continuous development of computer technology, control algorithms, and sensing technologies, mobile robots have been widely applied in real-world scenarios to assist or replace humans in performing complex and hazardous tasks [1–4]. To address the demands of challenging working conditions, robots have been designed to operate in various environments, including aerial, underwater, terrestrial, and pipeline contexts. Examples include quadrotor drones, unmanned submarines, and snake robots. These robots possess a certain degree of perception and autonomous decision-making capabilities, enabling them to complete their designated tasks. However, these robots typically have a single mode of movement, making them suitable for only one type of environment and limiting their operational capabilities to that specific domain. In contrast, robots capable of multiple modes of movement and functioning in multi-domain environments have significant practical implications. Among these, the research on cross-domain robots, particularly those capable of operating both on land and in the air, remains a prominent focus in the academic field [5–8]. We propose a novel robot named "tensegrone," which integrates a tensegrity robot with driving capabilities and a drone. The name "tensegrone" is a portmanteau of "tensegrity" and "drone". The diagram of tensegrone is shown in Fig. 1. This design leverages the impact resistance of the tensegrity structure and utilizes a cable-driven system to provide efficient and low-energy terrestrial rolling capabilities. It is capable of performing actions such as rolling and jumping [9]. Additionally, it benefits from an octocopter's agility and robust traversing abilities. Motion planning plays a crucial role in enabling the robot's autonomous movement in real-world tasks. Given the terrestrial-aerial mobility of tensegrone, the scenarios become increasingly complex.

Joint terrestrial-aerial planning is an effective approach to achieve dual-mode motion [5, 10]. A dynamic model is developed that applies to both terrestrial and aerial modes, enabling a seamless transition between the two [11]. The study [11] addresses the complexities involved in motion planning and control for vehicles capable of both terrestrial and aerial movement. It introduces a unified dynamic model for such vehicles, utilizing differential flatness to

Received 29 February 2024; Revised 11 March 2025; Accepted 11 March 2025; Published 18 June 2025. This paper was recommended for publication in its revised form by editorial board member, Hao Fang.
Email Addresses: *qingkai.yang@bit.edu.cn
*Corresponding author.

Fig. 1. The diagram of tensegrone robot.

facilitate the planning and control processes. This framework provides a robust solution for autonomous navigation in unknown environments. The literature [6] discusses the differential flatness of drones and two-wheeled vehicles. Specifically, it proposes a unified flight-rolling planner tailored for hybrid terrestrial and aerial quadcopters (HyTAQs). By mapping three-dimensional polynomial trajectories to two-dimensional surfaces, the authors derive motion primitives applicable to both aerial and ground movement modes, providing a consistent pipeline for both modes. Since the terrestrial rolling motion of tensegrity structures does not possess differential flatness, this trajectory mapping method cannot be used for them. The literature [5] employs a more generalized methodology in its frontend design. The authors build upon the concept of Uniform Visibility Deformation (UVD) from [12], extending it to encompass both terrestrial and aerial planning. This extension results in a geometric representation of feasible spaces suitable for ground and aerial navigation.

Due to the discrete nature of the rolling motion of tensegrity robots, explicit trajectory generation, such as polynomial or spline trajectories, is typically not employed in motion planning problems. The method proposed in [13] abstracts the terrestrial rolling path of tensegrity robots into concatenated regular hexagons and utilizes the A* algorithm for path planning. Owing to the existence of both equilateral and isosceles triangular surfaces on tensegrity robots, the rolling motion of these robots on the ground cannot be entirely modeled as the movement of regular hexagons. Therefore, when planning trajectories over considerable distances, this may lead to deviations from the center of the hexagon for tensegrity robots. The characteristics of tensegrity robots are analyzed in [14], and the tensegrity robots are modeled as an icosahedron. By constructing an unfolded icosahedron graph, they obtain a potential path graph for the tensegrity robots. This path graph better represents the rolling process of the tensegrity robots compared to the regular hexagon. Subsequently, the Dijkstra algorithm is employed for the target path search. This approach can be traced back to the research on the rolling of polyhedra mentioned in [15]. In [16], a breadth-first search method with pruning operations for potential path graphs of platonic solids on a plane is designed, enabling path searches for more general convex polyhedra. However, these methods [13–16] have a strong condition where certain groups of specific adjacent faces within the convex polyhedra cannot be transformed via rolling. While this condition simplifies the potential path graph, it significantly reduces the feasible paths for the robot, compromising part of its mobility. The physics simulation engine of tensegrity robots is utilized in [17]. In this approach, the motion of tensegrity robots is abstracted as multiple motion primitives involving transitions between different faces. These motion primitives are obtained through forward dynamic simulation. A sample-based path planning method is utilized to achieve rolling path planning for tensegrity robots. This method effectively leverages the motion characteristics of tensegrity robots. However, incorporating dynamics into planning requires validating the obtained paths through forward kinematics calculations at each sampling instance, leading to excessively long planning times, and reduced flexibility.

In octocopter trajectory planning, the authors of [18] adopted a two-tiered planning structure. It initially uses geometric path searching to obtain an initial path, which, after pruning, serves as the initial solution for the back-end B-spline trajectory optimization problem. Given the differential flatness, motion for octocopter can be represented by a polynomial trajectory and a function of yaw angle over time. The form of solutions to the optimal control problem is analyzed in [12], and the analytical form of the optimal solution for this specific problem is derived. Additionally, the authors examine common constraints and cost function forms. They provide methods for constraint handling and outline gradient computation with linear time complexity. In competitive settings, [19] employs weighted A* as the front-end path-search module and utilizes a rolling time-domain planning framework in the back-end for real-time trajectory replanning.

The tensegrone faces several challenges in its terrestrial-aerial path planning: The robot's terrestrial movement mode is unique, characterized by discrete rolling gaits, and existing terrestrial rolling planning algorithms have not perfectly solved its planning issues. Transitioning between terrestrial and aerial modes takes time, necessitating specific faces of the robot to make contact with the ground when

switching modes. No related work has been advanced on this aspect yet, and existing algorithms struggle to address the issue of discrete ground constraints and continuous position constraints coexisting in tensegrone.

The inspiration for this work stems from the aforementioned publications, with a focus on addressing the joint terrestrial-aerial path planning problem for tensegrone. To tackle this problem, we design a two-layer structure. In the first layer, a joint terrestrial-aerial geometric path generation module based on visib-PRM is used to generate candidate paths that combine both terrestrial and aerial routes. These candidate paths are then passed to the second layer for refinement. The second layer consists of two components: one is the trajectory generation algorithm for UAVs, where the geometric guiding path is integrated into the improved A* algorithm as part of its heuristic function to ensure a feasible trajectory. The other component is the terrestrial rolling trajectory generation algorithm, where points along the geometric path are used as centers for random sampling within the configuration space. This two-stage approach ensures that the generated paths satisfy the end state requirements while effectively integrating both terrestrial and aerial motion modes. The workflow of the tensegrone joint planner is shown in Fig. 2. The primary contributions of this work are summarized as follows:

(1) A specialized geometric path-search algorithm is devised to address the joint terrestrial-aerial planning challenge of tensegrone. The algorithm leverages sampling and UVD to create a probabilistic roadmap that captures the geometric characteristics of viable spaces.
(2) A sampling-based path planning algorithm for tensegrone is developed by utilizing a library of motion primitives instead of controlling inputs obtained through sampling. This approach enhances computational efficiency.
(3) A two-stage rolling planning method is devised to address discrete attitude and position constraints during takeoff of tensegrone.

This paper is arranged as follows. In Sec. 2, an introduction to the design details of the tensegrone robot is presented. In Sec. 3, we enhance the visiv-PRM and UVD concepts, devising a joint terrestrial-aerial planning algorithm. Section 4 details the trajectory generation algorithm. Here, for UAVs, we implement an enhanced A* algorithm and a 3D Bresenham algorithm [20] for initial aerial path pruning. Furthermore, we incorporate an optimization method that

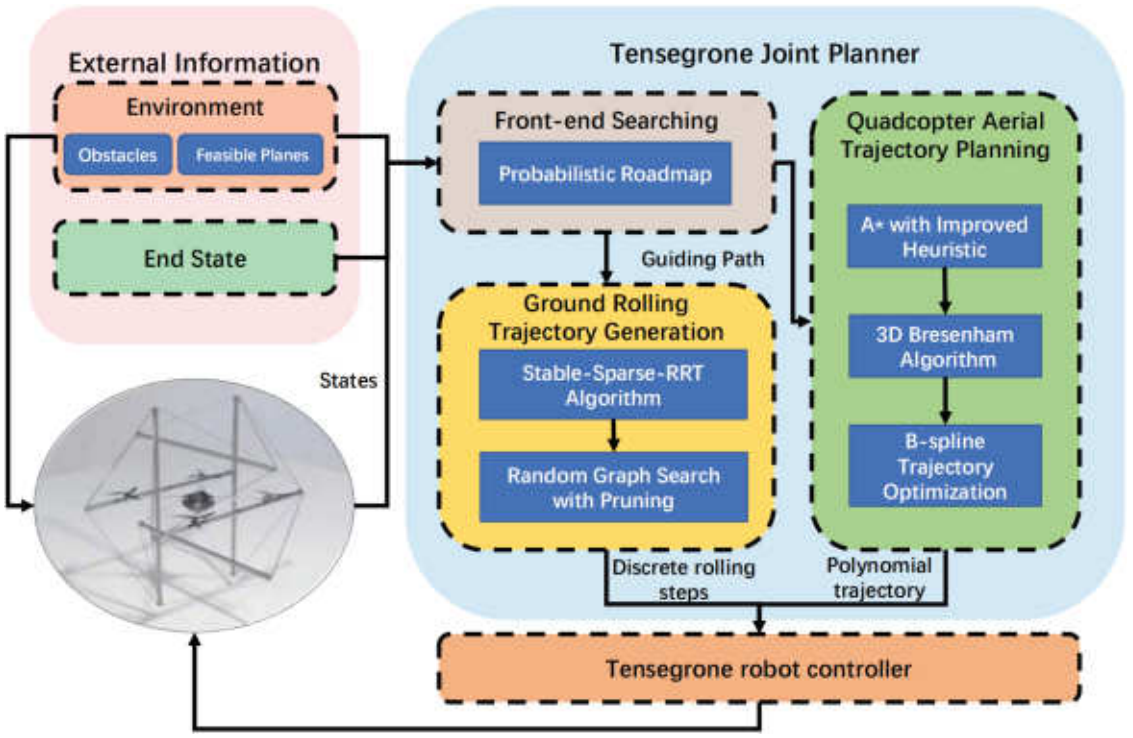


Fig. 2. System architecture of the tensegrone robot. The system processes external information and the end state through the Tensegrone Joint Planner. This planner includes two key modules: front-end searching, which constructs a probabilistic roadmap for the guiding path, and trajectory planning, which handles both terrestrial rolling and aerial motion. Terrestrial trajectory generation is achieved using Stable-Sparse-RRT and random graph search with pruning, while aerial motion is planned with an improved A* algorithm, 3D Bresenham, and B-spline trajectory optimization. The tensegrone robot controller integrates these modules to ensure seamless execution of both terrestrial and aerial trajectories.

leverages B-splines for trajectory generation (Sec. 4.1). Additionally, a terrestrial rolling trajectory generation algorithm is introduced, employing a two-stage trajectory generation approach. Initially, the Stable Sparse-RRT (SST) algorithm [21] based on motion primitives is utilized to generate a trajectory from the start to end point. Subsequently, an algorithm based on random graph search is employed to yield trajectories meeting end state requirements (Sec. 4.2). Section 5 presents the simulation results for tensegrone, while Sec. 6 showcases the physical experiments conducted on the robot. Finally, Sec. 7 provides the concluding remarks for this paper.

2. Introduction to Tensegrone

The "tensegrone" is a combination of a spherical six-rod tensegrity robot and an octocopter UAV, as shown in Fig. 1. Due to the octocopter's excellent symmetry, it is installed on the two parallel rods of the tensegrity structure. It is worth noting that although the tensegrone is an octocopter, its coaxial rotors provide lift in a unified direction. Since the flight stability of the octocopter requires the eight propellers to be fixed to a rigid body, while the motion characteristics of the tensegrity robot require that each rod be connected only by cables, a separating and merging mechanism has been designed, as shown in Fig. 3. The octocopter separates during the terrestrial rolling mode and merges during the aerial motion mode to meet the different motion requirements of the two robots.

Due to the structural characteristics of the tensegrone, it has a large moment of inertia. Therefore, when the propellers have difficulty providing reverse thrust, during takeoff it is required that the angle between the thrust direction of the octocopter blades and the direction of gravity be greater than $\pi/2$. Therefore, when switching from terrestrial mode to flight mode, the tensegrity robot on the ground needs to meet the takeoff requirements. Specifically, there are certain triangular faces in the icosahedron that satisfy the takeoff requirements. Taking Fig. 4 as example, its takeoff surface consists of two equilateral triangular faces enclosed by the red lines.

(a) (b)

Fig. 3. Separating and merging mechanism.

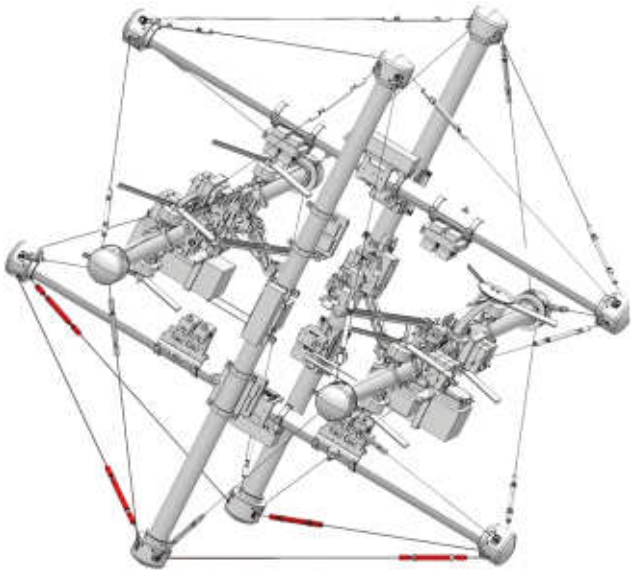
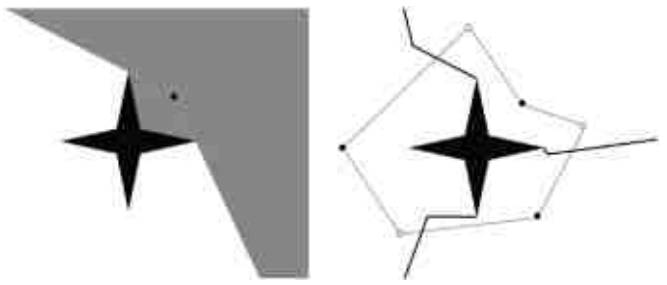


Fig. 4. Diagram of the takeoff surface of a tensegrone.

3. Joint Terrestrial-Aerial Geometric Path Planning

In the design of the joint terrestrial-aerial planning algorithm, to obtain potential mode transitions and energy consumption at unexpanded path nodes, the topological information of the free feasible space can be utilized. By retaining only the essential edges for node reachability, the data volume of the topological map can be significantly reduced. Additionally, integrating heuristic cost terms can alleviate computational burden.

In the design of the joint planning approach, the feasible spaces of different motion modes are decomposed into two parts: one part is the feasible airspace $C_{F_{air}}$ in the air, and the other part is the feasible space $C_{F_{obs}}$ on the bounded upper surface of obstacles (prisms and cylinders). Within the feasible space on the surface, it is classified into different labels based on connectivity, denoted as $C_{F_{obs_1}}$, $C_{F_{obs_2}}$, …, $C_{F_{obs_n}}$. These components, as part of the environmental representation, are added to the environment class Env. Therefore, the nodes p obtained through the sampling algorithm contain two pieces of information: their positional information p_{pos} and the label information of the space they belong to p_{flag}. A projection-based method is employed to map sample points onto manifolds within the free space, ensuring uniform sampling across all feasible spaces under different motion modes. When exploring feasible spaces, the guard class from visib-PRM is used to occupy free feasible spaces, and the bridge class is employed to connect adjacent spaces. Through the aforementioned nodes, we can obtain the connectivity relationships between feasible regions, as shown in Fig. 5.



(a) (b)

Fig. 5. Node visualization diagram in Visib-PRM. (a) Visibility domain of a configuration and (b) the visibility roadmap defined by guard nodes (black) and bridge nodes (white).

Algorithm 1. Uniform Sampling.

Input: $(C, C_{\text{obs}}, C_{F_{\text{obs}}})$

1: $Q = \emptyset$;
2: **while** $Q == \emptyset$ **do**
3: $pos_{rand} = uniformRandomSample(C)$;
4: **for** $obs_i \in Env.obs$
5: $d_i = minDistance(C_{F_{obsi}}, pos_{rand})$;
6: **if** $d_i < \theta_{local}$
7: $q_{proj} = planeProj(C_{F_{obsi}}, pos_{rand})$;
8: **if** $collisionCheck(C_{obsi}, q_{proj}.pos)$
9: $q_{proj}.flag = i$;
10: $add(Q, q_{proj})$;
11: **end**
12: **end**
13: **if not** $collisionCheck(C_{obs}, pos_{rand})$
14: $q_{proj}.flag = -1$;
15: $add(Q, q_{proj})$;
16: **end**
17: **end**
18: **end**

The function uniformRandomSample implements uniform sampling in the configuration space C, with the output being the three-dimensional coordinates pos_{rand} of the sampled node (line 3). Subsequently, the algorithm traverses all obstacles with surface feasible spaces and calculates the nearest distance from pos_{rand} to $C_{F_{obsi}}$. If this distance is less than a threshold, it triggers the subsequent mapping mechanism (lines 4–6). In the function planeProjection, the algorithm first maps pos_{rand} to the corresponding point on the plane where $C_{F_{obsi}}$ resides based on the surface normal vector of $C_{F_{obsi}}$. Then, using a guiding vector of random length and direction perpendicular to the normal vector, the corresponding point is mapped to $q_{project}$ (line 7). Subsequently, by checking the collision relationship between $q_{project}$ and the obstacle C_{obsi}, if a collision exists, the label of $q_{project}$ is set to the current obstacle's number, and the node is added to the output set Q (lines 8–10). Finally, to ensure that the threshold mapping mechanism does not affect the uniformity of the sampling algorithm, collision detection is performed on the original pos_{rand}. If it meets the safety requirements, the label of pos_{rand} is set to airspace, and it is added to the set Q (lines 13–15). Algorithm 1 yields a set of labeled nodes Q, which can be utilized for subsequent visual node determination.

In the visual node determination section (Algorithm 2), first, iterate through all nodes q_i in the set of sampled nodes Q, and consider the guard nodes in the roadmap G that are in the same feasible space as q_i as the current set of points to be computed. Find the set Q_{neigh} of points in the same feasible space within a radius δ of q_i (line 2). If Q_{neigh} is an empty set or if all points in Q_{neigh} are bridge nodes, then the current q_i is considered as a guard node to be added to the roadmap G (lines 3–5). Otherwise, if the number of visible nodes for the current node is greater than or equal to 2, it is considered that this node may become a bridge node and enters the bridge node update function (lines 6 and 7).

After obtaining a new candidate bridge node q_i, it is necessary to determine whether q_i satisfies the condition of uniform visible deformation within the same feasible space through a certain updating strategy. Utilize q_i and the guard nodes in Q_{neigh} to form a new local feasible path, and verify if there are other paths within the feasible space of this path. Based on this, determine whether a bridge node should be added or not.

In the bridge node update strategy (Algorithm 3), the node in Q_{neigh} closest to q_{bridge} is denoted as q_{nearest} (line 1). q_{nearest} is set as the base point for judging the uniqueness and uniform visible deformation of the bridge, i.e. all starting points of the paths to be checked are q_{nearest}. This design ensures that the newly added bridge node exists only within the coverage range of the closest node, preventing confusion in path priorities caused by newly added bridge nodes. Then, traverse through the nodes in set Q_{neigh} other than q_{nearest}, and connect q_{nearest}, q_{bridge}, and q_i in sequence to form a local feasible path $path_1$ (line 2 and 3). To determine

Algorithm 2. Visual Node Determination.

Input: $(Q, C, C_{obs}, \mathcal{G})$

1: **for** $q_i \in Q$
2: $Q_{neigh} = knnSearch(G.guard, q_i, \delta)$;
3: **if** $sempty(Q_{neigh})$ || $\forall q \in Q_{neigh}, q.type == bridge$
4: $q_i.type = guard$;
5: $addGuard(G, q_i)$;
6: **else if** $size(Q_{neigh}) \geq 2$
7: $bridgeUpdate(G, q_i, Q_{neigh})$;
8: **end**
9: **end**

Algorithm 3. Bridge Node Update.

Input: $(\mathcal{G}, q_{\text{bridge}}, Q_{neigh})$

1: $q_{\text{nearest}} = Q_{neigh}(1)$;
2: **for** $q_i \in Q_{neigh} \backslash q_{\text{nearest}}$
3: $path_1 = getPath(q_{\text{nearest}}, q_{\text{bridge}}, q_i)$;
4: $Distinct = true$;
5: $N_s = shareNeighs(G, q_{\text{nearest}}, q_i)$;
6: **for** $n_s \in N_s$
7: $path_2 = getPath(q_{\text{nearest}}, n_s, q_i)$;
8: **if** $UVDAble(path_1, path_2)$
9: $Distinct = false$;
10: **if** $len(path_1) < len(path_2)$
11: $replace(G, q_{\text{bridge}}, n_s)$;
12: **break;**
13: **end**
14: **end**
15: **end**
16: **if** $Distinct$
17: $addBridge(G,\ q_{\text{nearest}}, q_{\text{bridge}}, q_i)$;
18: **end**
19: **end**

the uniqueness of q_{bridge}, let N_s denote the common bridge nodes between q_{nearest} and p_i (line 5). Traverse through N_s to form a local feasible path $path_2$ composed of q_{nearest}, n_s, and q_i (line 6 and 7). If $path_2$ does not have a uniform visible deformation relationship with $path_1$ (line 8), then it is known that there is no reachable path between q_{nearest} and q_i, and a new bridge node is needed to connect the two points (lines 16 and 17). If $path_2$ has a uniform visible deformation relationship with $path_1$, then the current node is not considered as a bridge node to be added, and the lengths of the two paths need to be compared to choose the optimal local feasible path (lines 10 and 11). The relationships between the nodes are illustrated in Fig. 6.

The pseudocode for the uniform visible deformation check algorithm is shown in Algorithm 4. First, it checks whether the input two paths satisfy the conditions of the same type (line 2). Since the starting and ending points of $path_1$ and $path_2$ are the same, only the bridge nodes between the two points need to be compared, i.e. whether the labels of q_{bridge} and q_i are the same. If they are different, return false directly. During the UVD check process, normalize $path_1$ and $path_2$ to the scale range of [0,1], and compute the relationship between the line checkPath formed by $path_1(t)$ and $path_2(t)$ and the obstacles. If there exists a time t such that checkPath(t) collides with an obstacle, it is considered that the two paths do not have a uniform visible deformation relationship, and false is returned (lines 9 and 10).

Combining the above, the overall probabilistic roadmap search algorithm is shown in Algorithm 5. The probabilistic roadmap search algorithm first obtains a set of candidate nodes with positions and feasible space labels through uniform sampling within the feasible space (line 5). Then, it iterates through the node set, classifies the current node using the visual node determination strategy (lines 6–11), and checks the subsequent node update strategy for nodes with bridge attributes. During node updates, the nearest neighbor node obtained from the visual node determination process is used as the common starting point, and local feasible paths with the same bridge node are traversed to determine uniform visible deformation characteristics and costs (lines 12–28).

Finally, we use the Dijkstra algorithm to search for paths from the starting point to the destination point on graphs containing only aerial nodes, only terrestrial nodes, and graphs containing feasible planes, obtaining guiding paths for use in the subsequent path planning module.

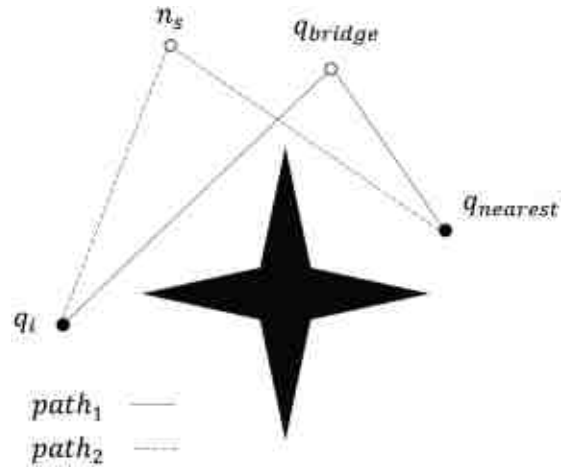


Fig. 6. Illustration of the relationships between the nodes.

Algorithm 4. UVD Able.

Input: $(path_1, path_2)$

1: $deformable = true$;
2: **if** $typeCheck(path_1, path_2)$
3: $deformable = false$;
4: **return;**
5: **end**
6: **for** $t = 0 : 1/\delta_t$
7: $checkPath = genPath(path_1(t * \delta_t),\ path_2(t * \delta_t))$;
8: **for** $t_c = 0 : 1/\delta_c$
9: **if** $collisionCheck(checkPath(t_c)$
10: $deformable = false$;
11: **break;**
12: **end**
13: **end**

Algorithm 5. Probabilistic Roadmap.

Input: $(\mathcal{G}, s, g)$

Initialize$(\mathcal{G}, Env)$;
AddGuard$(\mathcal{G}, s)$;
AddGuard$(\mathcal{G}, g)$;
while $t \leq t_{max} \wedge N_{\text{sample}} \leq N_{\max}$ **do**
    $p_s \leftarrow$ **Sample**();
    $g_{vis} \leftarrow$ **VisibleGuards**$(\mathcal{G}, p_s)$;
    $p_{same} =$ **FindSamePlane**(p_s, g_{vis});
    if $g_{vis}.size == 0$ **or** p_{same}== NULL **then**
        AddGuard$(\mathcal{G}, p_s)$;
    elseif $g_{vis}.size \geq 2$ **then**
    for $p_{iter} \in g_{vis} \backslash p_{same}$
        $path_1 \leftarrow$ **Path**$(p_{same}, p_s, p_{iter})$;
        Distinct ← True
        $\mathcal{N}_s \leftarrow$ **ShareNeighs**$(\mathcal{G}, p_{same}, p_{iter})$;
        for each $n_s \in \mathcal{N}_s$
            $path_2 \leftarrow$ **Path**$(p_{same}, n_s, p_{iter})$;
            if UVD Able ($path_1, path_2$**) then**
                Distinct ← False;
                if Len$(path_1) <$ **Len**$(path_2)$ **then**
                    Replace$(\mathcal{G}, p_s, n_s)$;
                    break;
                end
            end
        end
        if Distinct **then**
            AddBridge$(\mathcal{G}, p_s, p_{same}, p_{iter})$;
        end
    end
end

4. Terrestrial-Aerial Trajectory Generation

After obtaining the geometrically guiding path, we utilize this path as prior information, feeding it into terrestrial and aerial trajectory generation modules to derive a feasible trajectory that meets the requirements for controller execution. In the aerial trajectory generation module, we use an improved A* algorithm to generate collision-free initial paths, then apply the 3D Bresenham algorithm for path reduction. The processed path is utilized as the initial solution for trajectory optimization problems and inputted into the B-spline trajectory optimization algorithm to generate feasible aerial paths. In the terrestrial trajectory generation module, we employ the SST algorithm based on motion primitives to achieve continuous terrestrial-rolling trajectory generation. Additionally, to address endpoint posture constraints, we utilize a two-stage path planning approach to decouple the problem and conduct random graph search with pruning in the second stage.

4.1. *Octocopter aerial trajectory generation*

4.1.1. *Improved A* algorithm*

In this paper, we use an improved A* algorithm to achieve real-time path planning from the current position to the target position of a drone. A* is a commonly used algorithm for pathfinding and graph traversal, merging characteristics of both depth-first and breadth-first search algorithms. When provided with appropriate heuristics, it demonstrates good performance and accuracy. The crux of the A* algorithm lies in the design of its evaluation function. It maintains a heuristic evaluation function as

$$f(n) = g(n) + h(n). \tag{1}$$

This function aims for the shortest path, where $g(n)$ represents the cost from the starting node to the current node, and $h(n)$ is a heuristic function indicating the cost from the current node to the goal. A heuristic algorithm employing the strategy $f(n) = g(n) + h(n)$ meets certain conditions to qualify as an A* algorithm:

(1) There exists an optimal path from the start point to the endpoint in the search tree.
(2) The problem domain is finite.
(3) The search cost of all nodes' child nodes is larger than 0.
(4) $h(n) \leq h^*(n)$($h^*(n)$ represents the actual cost of the problem).

When these conditions are met, a heuristic algorithm with the $f(n) = g(n) + h(n)$ strategy is deemed an A* algorithm, guaranteeing the discovery of an optimal solution. Considering computational speed and path length, the heuristic function is set as the sum of the Euclidean distance from the current point to the endpoint and the offset distance from the guiding path as

$$h_1(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2 + (n_z - g_z)^2}, \tag{2}$$

$$h_2(n) = \min d(n, \text{GuidingPath}(\theta)), \tag{3}$$

where $d()$ represents the Euclidean distance. To balance optimality against computational speed and inspired by the Weighted A* concept, an improvement is made to the heuristic estimation function as

$$f(n) = g(n) + \varepsilon_1 * h_1(n) + \varepsilon_2 * h_2(n), \tag{4}$$

where $\epsilon_1, \epsilon_2 > 1$ is the heuristic weight coefficient. The node with the lowest value (highest priority) is selected as the next node to be traversed. Maintaining an optimal path tree enables real-time optimal path planning from the current position to the endpoint.

4.1.2. *Path reduction algorithm based on longest visible path*

As the A* algorithm operates on a grid map, the resulting path is grid-based and highly discrete, which is not conducive to backend processing. Furthermore, due to the use of stronger heuristics, the generated path is challenging to guarantee as optimal. Therefore, after obtaining a discrete path, it requires specific treatment.

Considering the process of A* generating a three-dimensional discrete path, a collision detection algorithm is designed for path reduction. Drawing from the Bresenham algorithm [20] in computer graphics, it is extended for higher dimensions. The Bresenham line algorithm is an incremental error algorithm used to generate straight lines on a grid, determining the n-dimensional grid points to approximate the line between two points. It's commonly employed in drawing line primitives in bitmap images (e.g. on computer screens) using only integer addition, subtraction, and shifting, which consume less in standard computer architectures.

In the two-dimensional case, let $(x_1, y_1), (x_2, y_2)$ be the endpoints of the line segment, Δx and Δy denote offsets, and m represents the slope written as

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}. \tag{5}$$

When $|m| \leq 1$, for a given x increment Δx,

$$\begin{cases} \Delta y = m\Delta x, \\ \Delta x = x_2 - x_1. \end{cases}$$

When $|m| > 1$, for a given y increment Δy,

$$\begin{cases} \Delta x = \frac{\Delta y}{m}, \\ \Delta y = y_2 - y_1. \end{cases}$$

Assuming the pixel $P_k(x_k, y_k)$ is determined to be displayed, the next point P_{k+1} needs to be determined whether to draw at (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}).

According to the equation of the line, the y coordinate at x_{k+1} on the line is

$$\begin{aligned} f(x) &= m(x_k + 1) + b, \\ d_1 &= f(x_k + 1) - y_k, \\ d_2 &= (y_k + 1) - f(x_k + 1). \end{aligned} \tag{6}$$

Substitute m into $d_1 - d_2$, and transform it to

$$p_k = \Delta x(d_1 - d_2) = 2x_k\Delta y - 2y_k\Delta x + c, \tag{7}$$

where p_k represents the decision parameter at step k, and c is a constant with a value of $2\Delta y + \Delta x(2b - 1)$.

Expanding the two-dimensional Bresenham algorithm to three dimensions yields Algorithm 6.

After generating the three-dimensional discrete line, optimizations are made to the A* pathfinding operation. Traditionally, since A* reaches the target point, the common approach involves tracing back from the current node to its parent node in reverse until the current node becomes the starting point. In our method, the reverse traversal seeks the parent node of the current node and then evaluates the visual relationship between the parent node and the starting point. If the generated discrete line between the nodes doesn't collide with obstacles, the current node is deemed in line with the longest visual path principle, and it's added to the path to be generated. Subsequently, using this current node as the new starting point, the reverse traversal starts again from the endpoint, repeating the process until the current node becomes the endpoint. In this process, the A* generated path is reduced to several key nodes, improving the quality of the initial solution.

Simultaneously, to ensure safety in the drone planning process, real-time assessments are made on the path generated initially. If the endpoint lies within an obstacle, the path is truncated to ensure the drone can remain in a safe position.

Algorithm 6. 3D Bresenham algorithm.

Input: $(StartPoint, EndPoint, res)$

Initialize();
v_{ori} = (EndPoint - StartPoint)/res;
v_{seq} = **sort**(abs(v_{ori}),descend);
for i in 1:3
    $R_{3\times3}(v_{seq}(i), i) = 1$;
end
$v = v_{ori} * R$;
$s = sign(v)$;
$v = s. * v$;
$currPoint = [-s.x, 0, 0]$
$addY = 2 * v.y$;
$addZ = 2 * v.z$;
$flagY = -v.x$;
$flagZ = -v.x$;
$d = -2 * v.x$;
while $(currPoint(1)! = v(1)||$
        $currPoint(2)! = v(2)||$
        $currPoint(3)! = v(3))$
    $currPoint(1) = currPoint(1) + s.x$;
    $currPoint(2) = currPoint(2) + s.y * (flagY > 0)$;
    $currPoint(3) = currPoint(3) + s.z * (flagZ > 0)$;
    $flagY = flagY + d * (flagY > 0) + addY$;
    $flagZ = flagZ + d * (flagZ > 0) + addZ$;
    $currPos = res * currPoint * R^T + StartPoint$;
end
return currPos;

4.1.3. *Opti-based curve generation*

Due to the differential flatness property, we only need to know the expression of the desired flat-output trajectory to

derive the expected expressions for all state variables and control quantities in the system. This will effectively reduce the complexity of the planning. Here, we describe the octocopter's trajectory in three-dimensional space using B-spline curves. The B-spline basis functions are defined by the Cox-de Boor recursive formula, as shown in

$$N_{i,k}(\rho) = \frac{\rho - \rho_i}{\rho_{i+k} - \rho_i} N_{i,k-1}(\rho) + \frac{\rho_{i+k+1} - \rho}{\rho_{i+k+1} - \rho_{i+1}} N_{i+1,k-1}(\rho), \tag{8}$$

where $i \in 0, 1, \ldots, n, k \geq 1$, and

$$N_{i,0}(\rho) = \begin{cases} 1, & \rho_i \leq \rho \leq \rho_{i+1}, \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

which represents a high-degree B-spline basis function as a linear combination of lower-degree B-spline basis functions. B-spline basis functions possess properties of nonnegativity, local support, continuity in derivatives, and linear independence. A B-spline curve can be represented as

$$C(\rho) = \sum_{i=0}^{n} Q_i N_{i,k}(\rho), \tag{10}$$

where $Q_i = (x_i, y_i, z_i)^T \in \mathbb{R}^3, i = 0, 1, \ldots, n$ represents the ith control point of the curve, $N_{i,k}(\rho)$ stands for the ith k-degree B-spline basis function determined by the knot vector U, where the number of knots satisfies $m = n + k + 1$.

The trajectory is parameterized by a uniform B-spline curve ϕ, with its degree denoted as p_b. It consists of N_c control points $Q_1, Q_2, \ldots, Q_{N_c}$, and a knot vector $t_1, t_2, \ldots, t_m$, where $Q_1 \in \mathbb{R}^3, t_m \in \mathbb{R}$, and $M = N_c + p_b$.

With reference to [22], we formulate the problem of obtaining a smooth, safe, and dynamically feasible trajectory as an optimization problem written as

$$\begin{aligned} f_{p2} &= \lambda_s f_s + \lambda_c f_c + \lambda_d (f_v + f_a) \\ &= \lambda_s \sum_{i=p_b-1}^{N-p_b+1} \|\mathbf{Q}_{i+1} - 2\mathbf{Q}_i + \mathbf{Q}_{i-1}\|^2 \\ &\quad + \lambda_c \sum_{i=p_b}^{N-p_b} \mathcal{F}(d(\mathbf{Q}_i), d_{\min}) \\ &\quad + \lambda_d \sum_{\substack{\mu \in \\ \{x,y,z\}}} \left\{ \sum_{i=p_b-1}^{N-p_b} \mathcal{F}(v_{\max}^2, \dot{Q}_{i,\mu}^2) \right. \\ &\quad \left. + \sum_{i=p_b-2}^{N-p_b} \mathcal{F}(a_{\max}^2, \ddot{Q}_{i,\mu}^2) \right\}, \end{aligned} \tag{11}$$

where $v_{\max}$ and $a_{\max}$ are single-axis maximum velocity and acceleration, $\mathcal{F}(x, y)$ is a penalty function

$$\mathcal{F}(x, y) = \begin{cases} (x - y)^2, & x \leq y, \\ 0, & x > y, \end{cases} \tag{12}$$

f_c stands for the collision cost, $d(q)$ is the distance of point q in ESDF. f_v and f_a penalize infeasible velocity and acceleration, where $\dot{Q}_i = [\dot{Q}_{i,x}, \dot{Q}_{i,y}, \dot{Q}_{i,z}]^T$ and $\ddot{Q}_i = [\ddot{Q}_{i,x}, \ddot{Q}_{i,y}, \ddot{Q}_{i,z}]^T$ are the control points of velocity and acceleration, which can be evaluated by

$$\dot{\mathbf{Q}}_i = \frac{\mathbf{Q}_{i+1} - \mathbf{Q}_i}{\Delta t}, \quad \ddot{\mathbf{Q}}_i = \frac{\mathbf{Q}_{i+2} - 2\mathbf{Q}_{i+1} + \mathbf{Q}_i}{\Delta t^2}. \tag{13}$$

4.2. *Terrestrial rolling trajectory generation algorithm within the tensegrity*

Due to the presence of mode switching points in the entire aerial-terrestrial trajectory generation problem, the structural characteristics of the tensegrone demand that it has one of two fixed surfaces in contact with the ground during takeoff. This introduces new constraints for the terrestrial rolling problem, i.e. discrete face constraints. The tensegrity rolling problem without considering contact surface constraints involves only an inequality constraint concerning the endpoint position, constituting a continuous constraint. Hence, this problem becomes intricate, encompassing both continuous and discrete state variables, along with continuous and discrete constraints. We propose a rolling planning method encompassing two planning stages: the first stage uses the SST algorithm, guided by the geometric path, to generate a preliminary trajectory within a certain range of the endpoint. In the second stage, the endpoint from the first stage serves as the starting point for a random graph search, refining the trajectory while considering the end state to ensure the vehicle lands on the desired takeoff surface.

4.2.1. *Rolling representation of tensegrity based on motion primitives*

In order to fully utilize the guidance path information obtained in the joint land-air planning, a guiding path-based sampling algorithm was designed, as shown in Algorithm 7. The guiding path, GuidingPath, abstracted into a sequence of points after undergoing path pruning algorithm, with paths between the points being straight lines that do not intersect with obstacles. To enhance the algorithm's robustness, a threshold-switching guiding path sampling was designed. If the current iteration count exceeds leadT, the algorithm degrades to regular configuration space random sampling; if the iteration count is not greater than leadT, the algorithm calculates the corresponding point on the guiding path based on the current iteration count and uses it as the center for

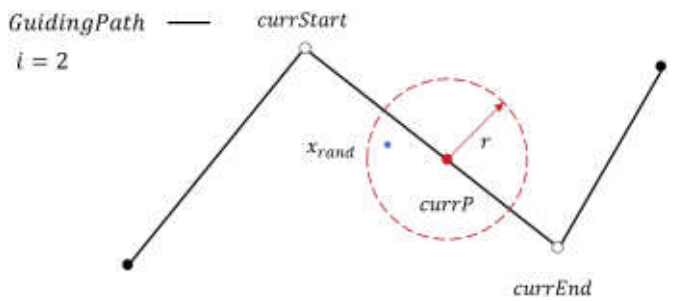


Fig. 7. Illustration of guiding path sampling.

Algorithm 7. Guiding Path Sampling.

Input: $GuidingPath, X, t$

1: **if** $t \leq leadT$
2: $[currT, i] = calSegGuidingPath, t, leadT;$
3: $currStart = GuidingPath(i);$
4: $currEnd = GuidingPath(i+1);$
5: $urrP = currT * currStart + (1 - currT) * currEnd;$
6: $\theta_{rand} = rand() * 2\pi;$
7: $r_{rand} = rand() * r;$
8: $x_{rand}.x = cos(\theta_{rand}) * r_{rand} + currP(x);$
9: $x_{rand}.y = sin(\theta_{rand}) * r_{rand} + currP(y);$
10: **else**
11: $xrand(X);$
12: **end**

random sampling. By combining the guiding path with configuration space random sampling, the utilization of feasible region connectivity in rolling planning can be improved without sacrificing the inherent randomness of the sampling algorithm. The guiding path sampling is illustrated in Fig. 7.

For the tensegrity terrestrial rolling problem, due to the tensegrone surface being a nonstandard icosahedron, it cannot cover the entire feasible area on the rolling surface. Algorithms such as A* that rely on constructing a representation map of the feasible space before performing graph search struggle to address such issues. Hence, we employ a sampling-based planning algorithm. This approach effectively utilizes the geometric path obtained from the second layer as a guiding path, enhancing sampling efficiency. Additionally, we opt for an algorithm based on SST [21], which can achieve probabilistic convergence to the optimal solution while ensuring computational efficiency.

We adopt a motion primitive-based approach to convert the tensegrity rolling control into a face-to-face rolling problem, thereby improving computational speed. The basic rolling motion of the tensegrity robots is achieved by transitioning from one face on the ground to an adjacent face. The tensegrity robots have a surface geometry comprising 24 edges occupied by elastic cables and six additional virtual

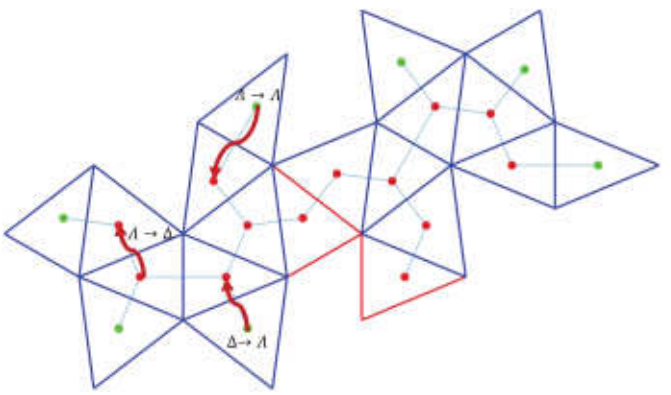


Fig. 8. Faces of an unfolded icosahedron tensegrity, where the red face serves as a surface that can switch to aerial modes.

edges forming 20 triangular faces. As a result, there could potentially be 60 motion primitives to consider in terms of rolling directionality. However, owing to the symmetrical topology of the tensegrity robots, these motion primitives can be reduced to three equivalent ones: (i) $\Lambda \leftarrow \Delta$, (ii) $\Delta \leftarrow \Lambda$, and (iii) $\Lambda \leftarrow \Lambda$. Additionally, for smooth transitions between triangular faces, longer motion primitives such as $\Delta \leftarrow \Lambda$ and $\leftarrow \Delta$ are also considered. Hence, designing motion primitives for the following four rolling actions enables the tensegrity robot to switch between any two faces, as described in [17]:

$$\mathcal{U} = \{u_{\Lambda\Delta}, u_{\Delta\Lambda}, u_{\Lambda\Lambda}, u_{\Delta\Lambda\Delta}\}. \tag{14}$$

To address the issue of slow computation speed in Monte Carlo forward propagation algorithms based on dynamics and control sampling, a forward propagation algorithm based on motion primitives is designed. After obtaining the optimal parent node $x_{selected}$ within the radius δ_{BN} on the tree, x_{rand} is updated to x_{tg} with the same orientation as $x_{selected}$ and a step size of random step $Step_{prop}$. Using a greedy approach, the shortest path to reach x_{tg} is computed, as shown in Algorithm 8.

By using the vector from x_{rand} to $x_{selected}$ as the growth direction and setting the target length as the random step $Step_{prop}$, the current target position x_{tg} is calculated (lines 1 and 2). The iteration terminates when the distance between the current path endpoint x_{curr} and x_{tg}, represented by $currDist$, is less than the threshold δ_e or when $currDist$ exceeds the threshold $\delta_e + targetDist$ (line 5). During each iteration, the algorithm selects an appropriate motion primitive using a greedy strategy, with the length of the control set U being determined by the random step count t (line 10). If the stopping criterion is met before completing the traversal of the random step count t, the loop is exited prematurely (lines 17–19).

Sampling-based motion planning algorithms like RRT aim to quickly find feasible paths but do not inherently possess an “anytime” property, meaning they cannot incrementally improve trajectory quality over time. Additionally, these

Algorithm 8. Motion Primitive Forward Propagation.

Input: $x_{selected}, U, Step_{prop}$

1: $\theta_{target} = \arctan(x_{rand} - x_{selected})$;
2: $x_{tg} = x_{selected} + Step_{prop} * [\cos(\theta_{target}), \sin(\theta_{target})]$;
3: $targetDist = norm(x_{tg} - x_{selected})$;
4: $currDist = targetDist$;
5: **while**
$currDist > \delta_e \cap countIter < maxIter \cap !killFlag$
6: $p_{new} = \emptyset$;
7: $U_{new} = \emptyset$;
8: $cost_{path} = 0$;
9: $x_{curr} = x_{selected}$;
10: $t = floor(rand() * tLengthMax) + minStep$;
11: **for** $i = 2 : t + 1$
12:
$[x_{curr}, cost_{step}, ctrlU] = StepDirec(x_{curr}, x_{rand})$;
13: $p_{new}.cost_{path} = p_{new}.cost_{path} + cost_{step}$;
14: $p_{new}.push(x_{curr})$;
15: $U_{new}.push(ctrlU)$;
16: $currDist = norm(x_{tg} - x_{curr})$;
17: **if** $currDist \leq \delta_e$ || $currDist$
18: $t = i - 1$;
19: $killFlag = True$;
20: **end**
21: **end**
22: $countIter++$;
23: **end**

Algorithm 9. SST using motion primitives.

Input: $(\mathbb{X}, \mathcal{U}, x_\circ, \mathbb{X}_G, T_{\text{prop}}, \delta_{BN}, \delta_s, N)$

1: G = $\mathbb{V}_{\text{active}} \leftarrow \{x_\circ\}, \mathbb{V}_{\text{inactive}} \leftarrow \emptyset, \mathbb{E} \leftarrow \emptyset$;
2: $s_0 \leftarrow x_\circ, s_0 \cdot \text{rep} = x_\circ, S \leftarrow \{s_0\}$;
3: NN.Add($x_\circ$); $\text{NN}_S \cdot \text{Add}(s_0)$;
4: **for** N iterations **do**
5: $x_{\text{new}} \leftarrow$ **RandSel**$(GuidingPath, \mathbb{X})$;
6: $x_{\text{selected}} \leftarrow$ **BestNearSel**$(\mathbb{V}_{\text{active}}, x_{\text{new}}, \delta_{BN})$;
7: $\pi_{\text{new}} \leftarrow$ **MPProp**$(x_{\text{selected}}, \mathcal{U}, T_{\text{prop}})$;
8: $s_{\text{new}} \leftarrow$ **FindWitness**$(\pi_{\text{new}}(t), G, S, \delta_s)$;
9: **if** not Coll(π_{new}) and BetterRep($\pi_{\text{new}}(t), s_{\text{new}}$)
10: $\mathbb{V}_{\text{active}} \leftarrow \mathbb{V}_{\text{active}} \cup \{\pi_{\text{new}}(t)\}$;
11: $\mathbb{E} \leftarrow \mathbb{E} \cup \{\pi_{\text{new}}\}$;
12: NN.Add($\pi_{\text{new}}(t)$);
13: SSTPrune(s_{new}.rep, G);
14: s_{new}.rep $\leftarrow \pi_{\text{new}}(t)$;
15: **end**
16: **end**
17: return G($\mathbb{V}_{\text{active}}, \mathbb{V}_{\text{inactive}}, \mathbb{E}$);

methods do not harness heuristic approaches to guide the search process. We adopt a sampling-based trajectory generation method derived from SST and adapt it to make use of the motion primitive library constructed earlier in this text.

Algorithm 9 presents the SST trajectory generation method based on motion primitives. The input of this algorithm is the initial state x_o of the system, the goal condition $\mathbb{X}_G$, the motion primitives $\mathcal{U}$, and the total number of iterations N. The method maintains a search tree T_{prop}. During the algorithm iterations, the **RandSel** function randomly samples within the region determined by the $GuidingPath$ to obtain an extension point x_{new}. From the active nodes of the existing tree G, the algorithm searches for the nearest neighbor x_{selected} to serve as the extensional node. The function **MPProp** is used to select the combination of motion primitives with the minimum extension cost in the motion primitive library to extend x_{selected}, resulting in the current local trajectory π_{new} (lines 5–7). At this point, the endpoint of the newly generated trajectory π_{new} may not necessarily coincide with x_{new}, hence the need to check if the current trajectory endpoint $\pi_{\text{new}}(t)$ lies within the range of other witnesses (line 8). If the current local trajectory does not collide with obstacles and is the lowest cost within the current witness, the endpoint of the current trajectory is added to the set of points V_{active} and the current trajectory is added to the edge set $\mathbb{E}$ (lines 9–12). Additionally, an update to the witnesses is required. If the endpoint of the current trajectory is not within the range of other witnesses, $\pi_{\text{new}}(t)$ is directly added as a new witness to the set. Otherwise, based on the rules for handling active point sets in SST, the points within the current witness are processed accordingly (lines 13, 14).

4.2.2. *Second-stage algorithm*

The second-stage algorithm is implemented in Algorithm 10. When the trajectory generation in the first stage reaches the endpoint threshold range, Algorithm 10 begins to run. Similar to Algorithm 9, the input of Algorithm 10 includes the motion primitive library $\mathcal{U}$, the initial state x_o of the tensegrone, and the target state X_G. Similarly, this algorithm maintains a graph structure G, initially with only one node x_o (line 1). During the iterations, if the list of valid controls ($AbleList$) becomes too large, a scaling flag is activated (line 3 and 4). When the size of AbleList drops below a threshold, the tabu lists ($CtrlTabu$) is reset, and $AbleList$ is merged with the out-of-range list ($AbleListOutRange$) to incorporate previously out-of-bound controls (lines 6–8). Next, a new control u_{new} is selected by resetting the tabu list based on the guiding path and control space U (line 10). If the selected control exists in the tabu list, a random control is chosen as an alternative (lines 12 and 13). If this alternative control is also found in the tabu list, the step flag is set to false, indicating no further action (lines 14 and 15). Otherwise, the tabu list is updated with the newly selected control, and it is removed from AbleList (lines 17 and 18). The algorithm then

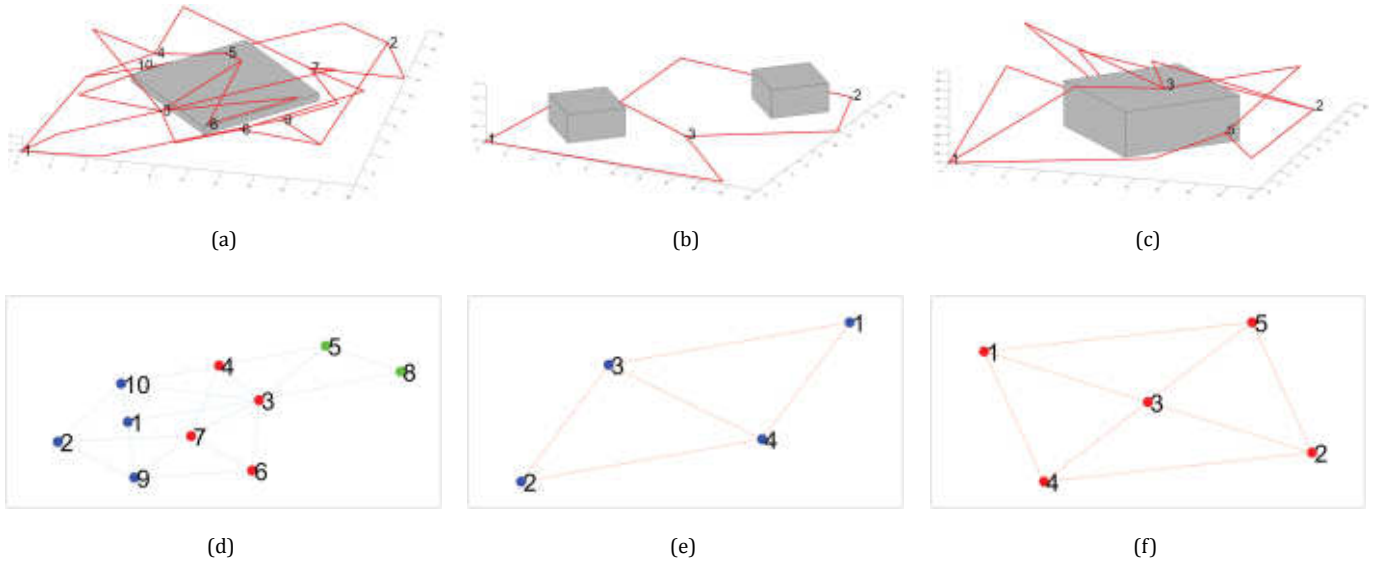


Fig. 9. The probability roadmap obtained through geometric path searching and its abstracted form as an undirected graph. (a) Visualized results of the route map containing aerial and ground guards. (b) Visualized results of the route map for ground guards. (c) Visualized results of the route map containing only aerial guards. (d) Connection diagram of corresponding nodes between aerial and ground. (e) Connection diagram of corresponding nodes for terrestrial. (f) Connection diagram of corresponding nodes for aerial.

checks if the step flag is true and evaluates whether the current state is out of range or has collided (lines 21 and 22). If a violation is detected, the control is added to the out-of-range list; if no violation occurs, the control remains valid (lines 23–26).

5. Simulation

5.1. *Simulation setup*

To validate the proposed method, we construct a simulation environment using MATLAB R2021b, C++, and ROS. Obstacles are abstracted as geometric objects with varying shapes or configurations, such as cuboids, cylinders and spheres. All simulations were done on a PC with an Intel Core i5-9400H processor and Nvidia Quadro T100 graphics card.

5.2. *Terrestrial-aerial geometric path planning*

The linked representation of the feasible space obtained through the proposed geometric path search is depicted in Fig. 9. From Fig. 9(a), it's evident that after acquiring feasible spaces through random sampling, the utilization of bridges effectively establishes connections between these feasible areas occupied by guards. Figure 9(d) presents a clearer depiction of the connectivity between different regions. Here, red nodes represent aerial nodes, blue nodes denote nodes on the plane or ground, and green nodes indicate nodes situated on feasible areas of the obstacle Obs_1.

Based on the aforementioned connectivity, we can conveniently derive multi-modal paths from Start 1 to Goal 2. For instance, in the roadmap shown in Fig. 9(d), this includes terrestrial rolling paths: $1-10-2$, $1-9-2$; mixed paths involving mode switching between rolling and flying: $1-3-7-2$; and hybrid paths utilizing feasible areas on the surface of

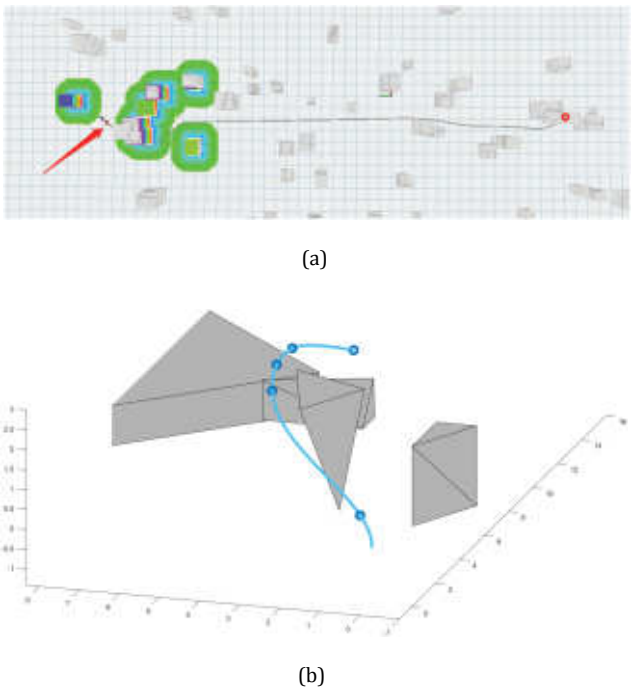


Fig. 10. The polynomial curve that satisfies dynamic constraints obtained from the aerial trajectory generation module, (a) simulated in C++ and ROS, and (b) simulated in MATLAB.

Algorithm 10. Random Graph Search with Pruning.

Input: $(\mathbb{X}, \mathcal{U}, x_\circ, N)$

initialization, G = $\mathbb{V} \leftarrow \{x_\circ\}$;
for N iterations **do**
  if size of AbleList > MaxSize **then**
    ScaleFlag = true;
  end
  if size of AbleList ≤ SetRange and ScaleFlag **then**
    ResetTabu($CtrlTabu, CtrlOutRange$);
    MergeList($AbleList, AbleListOutRange$);
  end
  u_{new} ←**ResetTabu**($GuidingPath, \mathcal{U}$);
  StepFlag = true;
  if ctrlU in **CtrlTabu**(index_now) **then**
    $u_{new} \leftarrow$ **RandSelect**($GuidingPath, \mathcal{U}$);
    if u_{new} is **CtrlTabu**(index_now) **then**
      StepFlag = false;
    else
      Update($u_{new}, CtrlTabu$);
      Delete($AbleList(u_{new})$);
    end
  end
  if StepFlag = true **then**
    if the current state is out of range or collides **then**
      Add($AbleListOutRange(u_{new})$);
    else
      Delete($AbleList(u_{new})$);
      Add($AbleListOutRange(index\_now)$);
    end
  end
end

obstacle Obs_1: $1-3-8-5-4-2$. This topological path search method can also be applied to robots that only have terrestrial mobility or only have aerial mobility. Figures 9(b) and 9(e) show the roadmap and corresponding spatial connectivity obtained when only terrestrial mobility is available, while Figs. 9(c) and 9(f) show the roadmap and corresponding spatial connectivity obtained when only aerial mobility is available.

5.3. *Octocopter aerial trajectory generation*

The L-BFGS method is employed to solve the optimization problem. Here, the gray blocks represent obstacles, while the colored elevation reflects the octocopter's mapping perception of the surrounding environment during simulation. The gradient-colored map denotes one layer of the generated Euclidean Signed Distance Field (ESDF) map.

The aerial trajectory generation shown in Fig. 10(a) is demonstrated in rviz. The aerial trajectory generation shown in Fig. 10(b) is implemented, and the collision gradient calculation method described in [23] is employed to replace the gradient generated by ESDF for obstacle avoidance trajectory optimization. This section also employs L-BFGS for solving the optimization problem.

From the simulation results, it can be seen that the aerial trajectory generation algorithm used in this study can effectively meet the requirements of trajectory smoothness and feasibility for the tensegrone.

5.4. *Terrestrial rolling trajectory generation*

In the simulation of the tensegrity terrestrial rolling algorithm, for the convenience of simulation experiments, we used the primitive formed by the tensegrity in a balanced state, which forms a dodecahedron when rolling on the ground, as the retrieved primitive in the motion primitive library. Figure 11 shows the simulation experiment of the

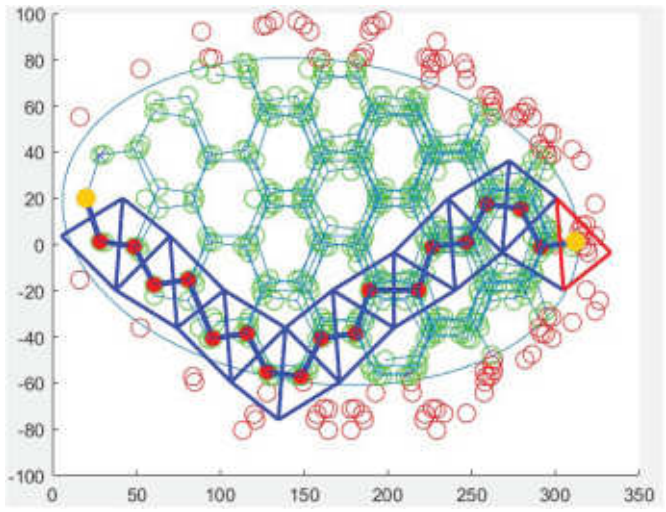
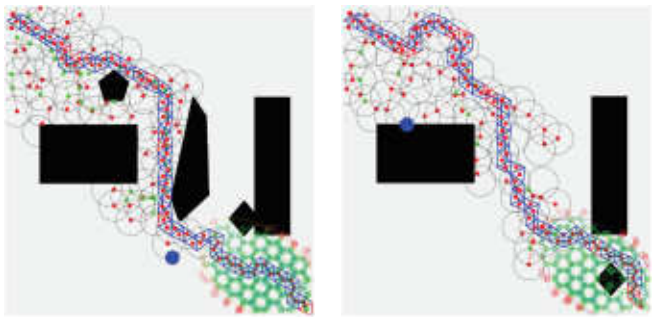


Fig. 11. Illustration of the second stage of tensegrity terrestrial rolling algorithm, with yellow nodes representing the start and end points.



(a) (b)

Fig. 12. The terrestrial rolling trajectory of tensegrone obtained through the two-stage algorithm. (a) and (b) show the performance of the algorithm with different obstacle placements.

second stage algorithm, where the blue faces represent non-takeoff surfaces, the red faces represent takeoff surfaces, and the elliptical region constitutes the information set used to restrict the search direction. The green circles indicate the feasibility of the current primitive, while the red circles indicate the infeasibility. The solid blue lines connecting the solid red circles form the feasible trajectory found by the search. From the figure, it can be observed that the proposed algorithm can guide the tensegrity from its initial position to the endpoint satisfying posture constraints and certain position constraints.

We conduct algorithm tests in simulation environments. The black area in the figure represents the obstacle region expanded according to the robot radius. The main difference between Figs. 12(a) and 12(b) lies in the presence of obstacles near the endpoint, which can test the adaptability of the second-stage algorithm to obstacles. The first stage, based on the SST algorithm of motion primitives, guides the trajectory to the vicinity of the endpoint and triggers the second stage. The second stage, based on a pruning-based random graph search algorithm, searches for trajectories that simultaneously satisfy the endpoint posture and position constraints, ensuring safety throughout the process. It can be seen from Fig. 12 that the two-stage algorithm can handle scenarios with complex obstacle environments.

6. Experiment for Rolling Planning

The retroreflective balls are placed on the rods of the tensegrity structure, and a rigid body model is established for each rod using the motion capture system Optitrack. The position and orientation of each rod's rigid body are calculated using Optitrack at any given time, and the pose data of each rod are recorded through the ROS interface provided by the motion capture system. Due to the ongoing development of the physical prototype of the tensegrone, specific energy consumption models for flight and rolling cannot be provided at this time. However, based on completed tests, the tensegrone has a hovering flight time of approximately 3 min and a rolling motion time of around half an hour.

In the simulation, when the acute angle of the given tensegrity surface is 45, the planned trajectory obtained is

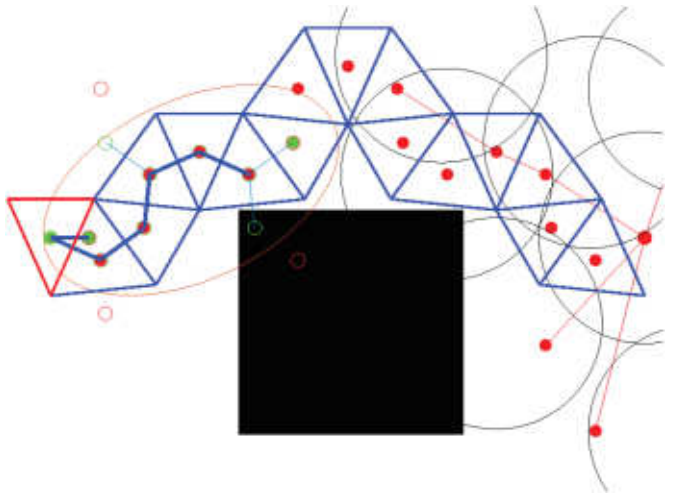


Fig. 13. Schematic representation of simulated trajectory planning results.

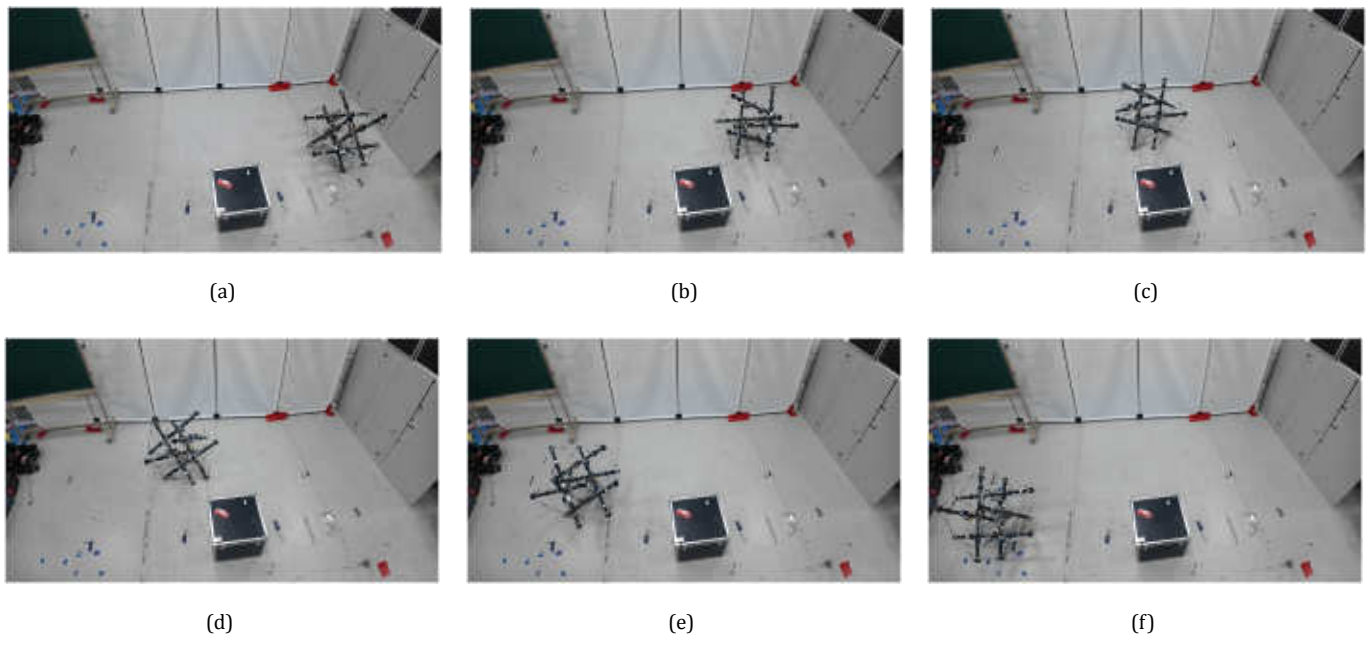


Fig. 14. Schematic representation of experiment for tensegrity rolling planning.

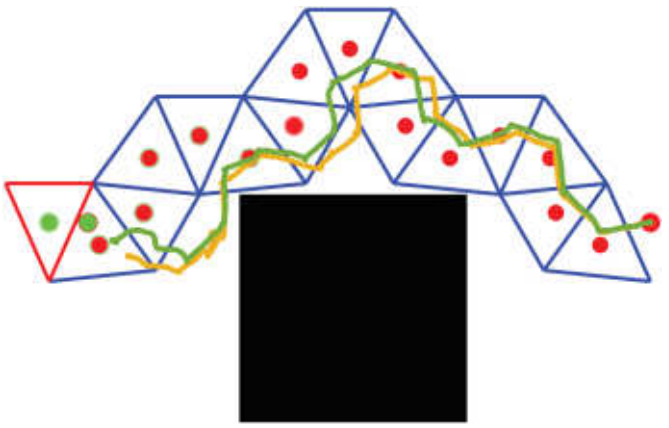


Fig. 15. Schematic representation of simulated trajectory planning results.

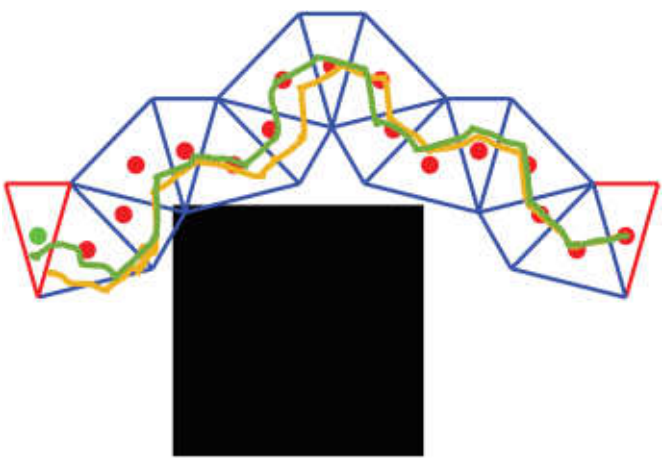


Fig. 16. Schematic representation of simulated trajectory planning results.

shown in Fig. 13 , with the starting point located on the right side of the image, and the red triangular face representing the takeoff surface.

In terms of rolling control, the motion primitive library based on open-loop control strategy is employed. To achieve continuous and rapid rolling, we utilize motion primitives of $\Delta \leftarrow \Lambda \leftarrow \Delta$ to simplify the control strategy. The terrestrial rolling planning experiment of the tensegrity is shown in Fig. 14. Figure 14(a) depicts the initial posture of the tensegrity, positioned on the takeoff surface of the tensegrity robot. Subsequently, the tensegrity continuously rolls to the left side. Figures 14(b)–14(e) illustrate the continuous rolling process, where the rolling trajectory bypasses obstacles, finally reaching the endpoint position depicted in Fig. 14(f), where the tensegrity returns to the takeoff surface.

The trajectories of the rolling process collected from multiple experiments using the motion capture system are shown in Fig. 15, with the mean position and orientation of

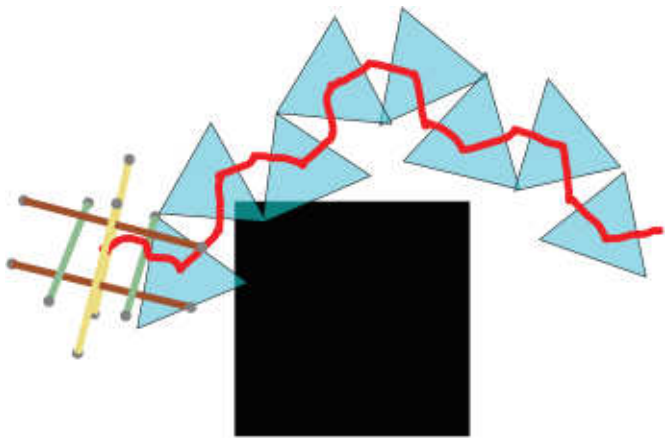


Fig. 17. Schematic representation of simulated trajectory planning results.

all rods used as the center of the tensegrity. Due to the significant difference between the acute angle of the Λ surface used in simulation planning and the actual robot, the prototype exhibits sliding and deformation movements during rolling, resulting in deviation of the trajectory from the endpoint. After adjusting the acute angle of the Λ surface to a value closer to the actual situation, which is 31.89, the rolling trajectory is shown in Fig. 16, while the trajectory obtained when the actual prototype lands on the ground is depicted in Fig. 17. It can be observed from the figures that the corrected simulation trajectory is closer to the actual central trajectory, with a smaller deviation distance from the endpoint. Hence, the accuracy of the motion primitive library significantly influences the effectiveness of the rolling trajectory planning proposed in this paper.

Furthermore, a video demonstrating the rolling test, rolling to flying transition, and flying test on the real tensegrone robot can be accessed at the following link: [https://youtu.be/QuiLQrpRhs8].

7. Conclusion

This paper proposes a joint terrestrial-aerial geometric path planning method that can be applied to tensegrone robots. A trajectory generation method is developed based on motion primitives and sampling to address the dual constraints of terrestrial rolling endpoint pose and position for tensegrone. This approach enables the tensegrone to autonomously roll to the vicinity of the endpoint while meeting takeoff requirements. The effectiveness of the algorithm proposed in this paper has been verified through simulations and experiments. In addition, this paper explores an optimization-based aerial trajectory generation method, which can serve as the foundation for subsequent tensegrone aerial deformation.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Nos. 2022YFB4702000 and 2022YFA1004703, in part by the NSFC under Grants Nos. 62373048, 62133002, U1913602, 62088101, 61720106011, in part by the Fundamental Research Funds for the Central Universities, and in part by the Shanghai Municipal Science and Technology Major Project (2021SHZDZX0100).

References

[1] Z. Zhang, C. Zhang and W. Li, Semantic segmentation of urban buildings from VHR remotely sensed imagery using attention-based CNN, in *IGARSS 2020-2020 IEEE Int. Geoscience and Remote Sensing Symp.* (IEEE, 2020), pp. 1833–1836.

[2] A. Coluccia, A. Fascista and G. Ricci, Cfar feature plane: A novel framework for the analysis and design of radar detectors, *IEEE Trans. Signal Process.* **68** (2020) 3903–3916.

[3] G. Pramatarov, D. De Martini, M. Gadd and P. Newman, Boxgraph: Semantic place recognition and pose estimation from 3d lidar, in *2022 IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (*IROS*) (IEEE, 2022), pp. 7004–7011.

[4] S. Spiegel, J. Sun and J. Zhao, A shape-changing wheeling and jumping robot using tensegrity wheels and bistable mechanism, *IEEE/ASME Trans. Mechatron.* **28** (2023) 2073–2082.

[5] T. Wu, Y. Zhu, L. Zhang, J. Yang and Y. Ding, Unified terrestrial/aerial motion planning for HyTAQs via NMPC, *IEEE Robot. Autom. Lett.* **8**(2) (2023) 1085–1092.

[6] R. Zhang, Y. Wu, L. Zhang, C. Xu and F. Gao, Autonomous and adaptive navigation for terrestrial-aerial bimodal vehicles, *IEEE Robot. Autom. Lett.* **7**(2) (2022) 3008–3015.

[7] K. Shi *et al.*, Mtabot: An efficient morphable terrestrial-aerial robot with two transformable wheels, *IEEE Robot. Autom. Lett.* **9** (2024) 1875–1882.

[8] P. Zheng, F. Xiao, P. H. Nguyen, A. Farinha and M. Kovac, Metamorphic aerial robot capable of mid-air shape morphing for rapid perching, *Sci. Rep.* **13**(1) (2023) 1297.

[9] J. Mo, C. Gao, H. Fang and Q. Yang, Design and locomotion characteristic analysis of a novel tensegrity hopping robot, in *2023 IEEE Int. Conf. Robotics and Biomimetics* (*ROBIO*) (IEEE, 2023), pp. 1–8.

[10] D. D. Fan, R. Thakker, T. Bartlett, M. B. Miled, L. Kim, E. Theodorou and A.-a. Agha-mohammadi, Autonomous hybrid ground/aerial mobility in unknown environments, in *2019 IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (*IROS*) (IEEE, 2019), pp. 3070–3077.

[11] R. Zhang, J. Lin, Y. Wu, Y. Gao, C. Wang, C. Xu, Y. Cao and F. Gao, Model-based planning and control for terrestrial-aerial bimodal vehicles with passive wheels, in *2023 IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (*IROS*) (IEEE, 2023), pp. 1070–1077.

[12] B. Zhou, J. Pan, F. Gao and S. Shen, Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight, *IEEE Trans. Robot.* **37**(6) (2021) 1992–2009.

[13] J. Chang, B. Li, W. Liu and W. Du, The path planning method of tensegrity robot based on a algorithm, in *2018 IEEE 8th Annual Int. Conf. CYBER Technology in Automation*, *Control*, *and Intelligent Systems* (*CYBER*) (IEEE, 2018), pp. 1502–1507.

[14] Y. Lu, X. Xu and Y. Luo, Path planning for rolling locomotion of polyhedral tensegrity robots based on dijkstra algorithm, *J. Int. Assoc. Shell Spat. Struct.* **60**(4) (2019) 273–286.

[15] Y. Chitour *et al.*, Rolling polyhedra on a plane, analysis of the reachable set, in *Algorithms for Robotic Motion and Manipulation* (CRC Press, 1997), pp. 277–285.

[16] N. T. Lam, I. Howard and L. Cui, Path planning for the platonic solids on prescribed grids by edge-rolling, *PLoS One* **16**(6) (2021) e0252613.

[17] Z. Littlefield, D. Surovik, M. Vespignani, J. Bruce, W. Wang and K. E. Bekris, Kinodynamic planning for spherical tensegrity locomotion with effective gait primitives, *Int. J. Robot. Res.* **38**(12–13) (2019) 1442–1462.

[18] Z. Wang, X. Zhou, C. Xu and F. Gao, Geometrically constrained trajectory optimization for multicopters, *IEEE Trans. Robot.* **38**(5) (2022) 3259–3278.

[19] R. Li, J. Lyu, A. Wang, R. Yu, D. Wu and B. Xin, Flagdroneracing: An autonomous drone racing system, *Unmanned Syst.* **12** (2023).

[20] J. Bresenham, Algorithm for computer control of a digital plotte, *IBM Syst. J.* **4** (1965) 25–30.

[21] Z. Littlefield, Efficient and asymptotically optimal kinodynamic motion planning, PhD Thesis, Rutgers The State University of New Jersey, School of Graduate Studies (2020).

[22] E. T. Alotaibi, S. S. Alqefari and A. Koubaa, Lsar: Multi-UAV collaboration for search and rescue missions, *IEEE Access* **7** (2019) 55817–55832.

[23] Q. Wang, Z. Wang, L. Pei, C. Xu and F. Gao, A linear and exact algorithm for whole-body collision evaluation via scale optimization, in *2023 IEEE Int. Conf. Robotics and Automation* (*ICRA*) (IEEE, 2023), pp. 3621–3627.

Songyuan Liu received his B.S. degree in automation from the Beijing Institute of Technology, China, in 2017, and his M.S. degree in Aeronautical Engineering from Nanjing University of Aeronautics and Astronautics, in 2020. He is currently working toward his Ph.D. degree in control science and engineering with the Beijing Institute of Technology, China.

His current research interests include tensegrity robotics system design, modeling, and rolling control.

Zhe Jing received her B.Eng. degree in control science and engineering from the School of Automation and Electrical Engineering, the University of Science and Technology Beijing, Beijing, China, in 2016, where she received the Ph.D. degree in control science and engineering with the School of Intelligence Science and Technology, in 2023.

She is currently Postdoctoral Researcher at the School of Automation, Beijing Institute of Technology, Beijing, China.

Her research interests include distributed parameter systems and intelligent control.

Siyuan Hao received his B.S. degree in mechatronics engineering from the Beijing Institute of Technology, Beijing, China, in 2022, and is currently pursuing his M.S. degree in Control Science and Engineering from the Beijing Institute of Technology, Beijing, China.

His research interests include motion planning, tensegrity, and autonomous aerial robots.

Jingshuo Lyu received his B.S. degree in automation from the Beijing Institute of Technology, Beijing, China, in 2021, and is currently pursuing M.S. degree in control science and engineering from the Beijing Institute of Technology, Beijing, China.

His research interests include motion planning, tensegrity robots, and UAV.

Zichen Tao received his B.S. degree in automation from Beijing Jiaotong University, China, in 2024. He is current working toward Ph.D. in Control Science and Engineering with the Beijing Institute of Technology, China.

His current research interests include motion control and simulation of tensegrity based multi-domain robots.

Yun Gui received his B.S. degree in electrical engineering and automation from Beijing Institute of Technology, China in 2024. He is currently working toward his Ph.D. degree in artificial intelligence at the Beijing Institute of Technology.

His current research interests include tensegrity robotics locomotion control and mode switching by deep reinforcement learning.

Hao Fang (Member, IEEE) received his B.S. degree from the Xi'an University of Technology, Shaanxi, China, in 1995, and the M.S. and Ph.D. degrees from the Xi'an Jiaotong University, Shaanxi, in 1998 and 2002, respectively. Since 2011, he has been a Professor with the Beijing Institute of Technology, Beijing, China.

He held two postdoctoral appointments with the INRIA/France Research Group of COPRIN and the LASMEA (UNR6602 CNRS/Blaise Pascal University, Clermont-Ferrand, France). His research interests include all-terrain mobile robots, robotic control, and multi-agent systems.

Qingkai Yang (Member, IEEE) received the first Ph.D. degree in control science and engineering from the Beijing Institute of Technology, Beijing, China, in 2018, and the second Ph.D. degree in system control from the University of Groningen, Groningen, The Netherlands, in 2018.

He is currently a Professor with the School of Automation, Beijing Institute of Technology. His research interest is in cooperative motion control of multi-agent systems and autonomous robots.