Real-time collision-free landing path planning for drone deliveries in urban environments

Hanseob Lee\textsuperscript{1} | Sungwook Cho\textsuperscript{2} | Hoon Jung\textsuperscript{1}

\textsuperscript{1}Digital Convergence Research Laboratory, Postal & Logistics Technology Research Center, Electronics Telecommunications Research Institute, Daejeon, Republic of Korea
\textsuperscript{2}Department of Aeronautical and Mechanical Engineering, Cheongju University, Cheongju, Republic of Korea

Correspondence
Sungwook Cho, Department of Aeronautical and Mechanical Engineering, Cheongju University, Cheongju, Republic of Korea.
Email: swcho84@cju.ac.kr

Funding information
Ministry of Science and ICT, South Korea, Grant/Award Number: NRF-2020R1G1A10048001; Ministry of Trade, Industry and Energy (MOTIE), Republic of Korea, Grant/Award Number: RS-2023-00256794

Abstract
This study presents a novel safe landing algorithm for urban drone deliveries. The rapid advancement of drone technology has given rise to various delivery services for everyday necessities and emergency relief efforts. However, the reliability of drone delivery technology is still insufficient for application in urban environments. The proposed approach uses the “landing angle control” method to allow the drone to land vertically and a rapidly exploring random tree-based collision avoidance algorithm to generate safe and efficient vertical landing paths for drones while avoiding common urban obstacles like trees, street lights, utility poles, and wires; these methods allow for precise and reliable urban drone delivery. We verified the approach within a Gazebo simulation operated through ROS using a six-degree-of-freedom drone model and sensors with similar specifications to actual models. The performance of the algorithms was tested in various scenarios by comparing it with that of state-of-the-art 3D path planning algorithms.

KEYWORDS
autonomous drone, landing angle control guidance, obstacle avoidance, system integration, urban drone delivery

1 INTRODUCTION

Recently, sensor-enabled unmanned aerial vehicles (UAVs) have been widely deployed for civilian applications. Sensor-based environmental recognition technologies with small and powerful graphics processing unit-enabled embedded systems are at the core of these advancements. Notably, autonomous landing is one of the biggest problem areas in this domain today. Not only must a UAV land precisely at a given waypoint, it must do so while recognizing a designated landing pad and safely navigating natural and artificial obstacles in complicated urban environments.

Contemporary UAV landing methods rely on the global positioning system (GPS), which can be used as long as the landing zone is properly mapped [1]. However, commercial GPS has an error range of several meters, making it unsuitable for precision landings. Hence, GPS-enabled adaptive inertial navigation system controls have been studied [2, 3]. More recently, increasingly powerful vision processing hardware has allowed real-time image processing, which is highly beneficial for automatic visual targeting to achieve precision automatic landing [4–6].

Collision avoidance is a critical aspect of drone delivery, as it ensures the safety of people and property, as well as the integrity of the drone and its payload. To prevent collisions, drones are equipped with sensors and other technologies that allow them to detect and avoid...
obstacles in their path. Prior to real-time imaging, drones relied on recognizing specific high-contrast patterns such as augmented-reality markers using traditional image processing algorithms [7]. However, recent deep-learning techniques have been improved object detection algorithms such as the single-shot multibox detector and the “you only look once” algorithms [8, 9]. For operating drones in urban environments, we need to consider the restricted airspace around the landing pad. Some research proposed that a conical airspace is effective in an urban area [10–14].

This paper proposes a safe and automatic landing algorithm for drones in urban environments that generates an optimal landing path using the landing angle control (LAC) guidance law and includes a rapidly exploring random tree (RRT)-based collision avoidance algorithm to avoid obstacles encountered during landing. We evaluated the performance of the proposed algorithm through simulations and compared it with that of state-of-the-art algorithms using various performance metrics.

2 | RELATED WORKS

2.1 | Obstacle avoidance

Obstacle avoidance is a crucial aspect of drone operations in urban environments. Pioneering research was performed by Choi and others [15], but they focused only on building avoidance.

Recent advances have focused on real-time trajectory optimization using Euclidean signed distance fields [16]. However, these methods are computationally prohibitive, particularly with drones, especially in complex environments. There is a use case to avoid obstacles in various environments using a drone with a two-dimensional (2D) path planning method [17, 18]. To address this issue, this research uses gradient information from obstacles to improve three-dimensional (3D) path planning [19].

Extant studies have examined local path planning obstacle avoidance methods in which candidate paths are generated in advance, and the path of maximum cost is selected [20]. Other methods have applied RRT schemes [21, 22].

2.2 | Autonomous landing

Autonomous landing research has focused largely on reaching a stationary target with markers [23–25]. These models rely on onboard vision systems that use simple tracking algorithms. Other researchers [26, 27] have applied guidance laws traditionally used for missile systems. For moving markers, one researcher [28] used a downward-facing camera, whereas another [29] used a forward-facing camera to track targets at low speeds. An indoor low-speed moving-platform algorithm was designed using $H_{ac}$ controls [30], and a method for landing drones on moving vehicles with markers was later developed [31]. Notably, a study by another researcher [32] presented a method of landing drones on vehicles moving at speeds of up to 3 m/s.

3 | METHOD

In this section, we fully describe our vertical landing collision avoidance technique: the LAC-based receding horizon (RH) RRT star (RRT*) model (LAC-RHRRT*).

We first explain the generation of a vertical landing reference path from the drone’s current position to a landing marker. We then provide our novel algorithm for obstacle avoidance along the reference landing path, which ensures safe obstacle avoidance and landing. We finally discuss the process of filtering point clouds and registering obstacles for efficient calculations.

Figure 1 illustrates the software architecture of the proposed algorithm. This figure explains that the Ouster light detection and ranging (LiDAR) recognizes surrounding objects, and then, the point cloud data are filtered and registered as obstacles. LAC-RHRRT* generates the landing path while avoiding the registered obstacles.

3.1 | Landing path generation

To generate an optimal vertical landing path, we leveraged the impact angle control (IAC) guidance law used
in missile guidance laws. The IAC guidance law algorithm causes a missile to collide at a specific angle at its terminal stage. In this study, the drone can be adjusted to land vertically at the terminal stage. First, to create a vertical landing path, the geometry between the drone and the landing position can be expressed as shown in Figure 2A.

IAC guidance kinematics must be changed to the rotated coordinate system at a desirable angle to land in the fixed coordinate system when defining the lateral error $z$ in the original missile guidance law. Figure 2B presents the modified kinematics for vertical landing. The velocity command, $V_{cmd}$, is divided into downward velocity $w_{cmd}$ and horizontal velocity $V_{H cmd}$; the horizontal velocity is divided into $u_{cmd}$ and $v_{cmd}$. The optimal solution is calculated by obtaining the linearized equation through the line-of-sight (LOS) and flight path angles defined in the kinematics.

We define the vehicle flight path angle, $\gamma$, and LOS angles, $\lambda$, in a new frame, $XY_u-Z_u$:

$$\gamma = \gamma_f - \lambda_f.$$  

(1)

By applying this to the nonlinear equations of motion in the $XY_u-Z_u$ frame, we obtain

$$\dot{z} = v = V_{cmd} \sin \gamma \quad \dot{\gamma} = a_{cmd}/V_{cmd}.$$  

(2)

If (2) is linearized, it may be expressed as (3):

$$\dot{z} = v \quad \dot{v} = a_{cmd} \quad (\gamma \text{ is small}).$$  

(3)

To solve the optimization problem, the system in (3) is represented as the state equation in (4):

$$\begin{bmatrix} \dot{z} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} z \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a_{cmd}. \quad (4)$$

$$X = [z \ v] A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (4)$$

In (4), because the terminal condition of the optimal path algorithm entails hitting the target at the specified angle, the constraints of position and velocity must be determined as shown in (5).

$$v(t_f) = z(t_f) = 0 \quad D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \quad (5)$$

This can then be expressed as the following linear quadratic problem:

$$J = \frac{1}{2} \int_{t_0}^{t_f} (x^T F x + u^T G u) dt,$$

$$\dot{X} = AX + Bu \quad \Psi = DX_f.$$

(6)

In (6), $F$ is a zero matrix, $[0]_{2 \times 2}$, and $G$ is a unit vector, $[1]$. $\Psi$ is a terminal condition of the optimal problem. $u$ minimizes the cost function, $J$. By substituting the value of the state equation obtained above, $u^*$ is obtained as in (7):

$$u^* = \frac{V_{cmd}}{t_{go}} \left( -6 \lambda + 4 \gamma + 2 \gamma_f \right). \quad (7)$$

Hence, when we apply the command acceleration in (7) to the drone, it will perform the desired landing at the desired angle on the target.

Using the optimal command defined above, a vertical landing path can be obtained by using the virtual state

![Figure 2](image-url)

**Figure 2** Landing path generation: (A) geometry of and landing profile for optimal path and (B) modified kinematics for vertical landing.
value when initiating the vertical landing. Acceleration is obtained, velocity is calculated through Euler integration, and the virtual position is updated by integrating the updated velocity. The position obtained while proceeding until the virtual position reaches the marker becomes the landing path.

In Algorithm 1, $Q$ is the reference landing path set, and $X_{uav}$ and $X_{tar}$ are the initial positions of the UAV and marker, respectively. $X_v$ and $V_v$ are the virtual states for calculating the landing path, and $n$ is a normal vector from the velocity vector. $\Delta R$ is a relative distance to divide with $\eta$ segment. The virtual position is saved in $Q$ for each $\eta$ segment on the path during Euler integration. While integrating, if the virtual position $X_v$ enters within $\eta$ distance of the marker, the while loop is terminated, and the target position $X_{tar}$ is stored in $Q$.

**Algorithm 1**

```plaintext
$Q \leftarrow \text{CalculateLandingPath}(X_{uav}, X_{tar})$

**Require:** set zero $Q$

1. $\Delta R \leftarrow 0$
2. $X_v \leftarrow X_{uav}$
3. $V_v \leftarrow V_{init}$
4. while $\|X_{tar} - X_v\| < \eta$ do
5.   $u^* \leftarrow \text{LandingAngleControl}(X_{uav}, X_{tar})$
6.   $V_v \leftarrow V_v + u^* \Delta T n$
7.   $X_v \leftarrow X_v + V_v \Delta T$
8.   $\Delta R \leftarrow \Delta R + \|\Delta X_v\|$   
9.   if $\Delta R > \eta$ then
10.     $Q_{push}(X_v)$
11.     $\Delta R \leftarrow 0$
12. end if
13. end while
14. $Q_{push}(X_{tar})$
15. return $Q$
```

3.2 | Collision-free planning

In this section, we determine how the drone follows the optimal landing path while avoiding obstacles along the way. We apply the RRT-based avoidance method because there are many irregular obstacles in urban environments.

The referenced paper introduces the real-time path planner using RRT* in 2D space [33]. The RHRRT* algorithm is developed to use the RRT* path planning method in real time. A random sample is generated to optimize the path in the RH area. In this case, we have adapted this algorithm in a 3D space. We already have a trajectory toward the landing pad; thus, the avoidance algorithm follows it while the drone avoids it.

3.2.1 | LAC RHRRT*

Before examining the full procedure in detail, we first define the key variables. $Q$ is an $n$-element vertical landing path set, $\{q_1, q_2, ..., q_n\}$, and $q_i \in \mathbb{R}^3$. From $Q$, we can extract the set $P$, which is the tracking path. $P$ can be expressed as follows:

$$P = \{p_i | p_i \in Q, p_i = q_{i+t}, t = 1 to n_p\}.$$  (8)

$P$ is an $n_p$-element set, and $p_i \in \mathbb{R}^3$. $q_i$ is the current location of the drone. The last point on the tracking path, $p_{n_p}$, is the local goal point, $G$, which is the temporary goal of RHRRT*. If the goal point is assigned, we can determine the goal vertex from the set of vertices, as follows:

$$D = \begin{cases} 
  d_1 = 0.1 & \text{if } \min \|p_{n_p} - O\| > \eta, \\
  d_2 = 1.2 \min \|p_{n_p} - V\| & \text{otherwise}.
\end{cases}$$  (9)

$$H = \arg\min_{v_i} \{v_i | d_i = \|v_i - p_{n_p}\|, d_i < D, i = 1 to m\}.$$  (10)
\( V \) is an \( m \)-element set of LAC-RHRRT* vertices, \( \{ v_1, v_2, ..., v_m \} \), and \( v_i \in \mathbb{R}^3 \). \( O \) is the set of registered obstacles. The goal vertex \( H \) is chosen from among the vertices as the nearest from \( p_{np} \). During this process, \( D \) is set according to one of two conditions. If \( p_{np} \) is far from the obstacle points, \( D \) is set to the small value, such as 0.1 m in (9). However, if \( p_{np} \) is near an obstacle point, \( D \) is set to the value of the minimum distance between \( V \) and \( p_{np} \). According to (10), the goal vertex \( H \) is always determined until \( p_{np} \) approaches \( q_n \), which means that the local path avoiding the obstacles always exists in all processes. \( L \) is a set of local path points.

To efficiently extend the random tree, the random samples are biased toward \( G \). Biased random sampling was originally designed for 2D spaces; however, as mentioned before, we extend this to create a tree by making a random sample in a 3D space. The whole process to generate the avoidance path is in Figure 3. This figure shows the step process of the path generation such as how the goal vertex is selected and how the reference points near the obstacles are excluded. When the goal vertex is selected, the local path can be decided to avoid obstacles.

### 3.2.2 Node removal

The RRT algorithm provides optimized solutions, even in a non-convex environments; however, the computational burden increases with the number of nodes. To overcome this problem, unnecessary nodes must be periodically removed. Our node removal algorithm consists of two stages. The first removes past nodes along the local path, and the second removes nodes close to obstacles.

As the drone flies along the local path, and even after passing path points, the nodes created in the RRT remain in the memory space. Because the path to the local goal is attached to the front of the local

---

**Figure 3** The process of obstacle avoidance using landing angle control-based RHRRT*: (A) the drone is located at time \( i \), (B) the local goal point is in the obstacle region at time \( i + 3 \), (C) the avoidance path is generated at time \( i + 5 \), and (D) the local goal point is out from the obstacle at time \( i + 7 \).

**Figure 4** Removal of nodes connected to the previous waypoint. (A) The nodes are connected to the current waypoint; (B) after changing the target waypoint, they are all removed.
path, nodes attached to the path point that has already been passed do not significantly affect it. Therefore, the nodes attached to the previous path point are removed through tree searching. Figure 4 illustrates the node removal process. This figure shows that the nodes connected to the past waypoint, are removed after the removal process. The removal criterion is in Algorithm 3, which is keeping the minimum nodes in the RRT. The pseudo-code of the removal node is listed in Algorithm 4. When the function of the removal criteria is active, it finds the entire child nodes attached to the past waypoint and then removes all from the RRT.

**Algorithm 3** Flag, cnt← RemoveCriteria \( (X_{\text{nav}}, L, T, \text{cnt}) \)

1. FlagDist ← false
2. \( d = ||L[\text{cnt}] - X_{\text{nav}}|| \)
3. if \( d < \eta \) then
   4. FlagDist ← true
   5. cnt ← cnt+1
4. end if
5. if FlagDist is true then
   6. if len(T) > \( \zeta \) then
      7. Flag ← true
      8. FlagDist ← false
   9. end if
10. end if
11. return Flag, cnt

**Algorithm 4** \( T(V, E) \) ← RemovalNodes \( (T, L, \text{cnt}) \)

Require: set zero \( R \)
1. FlagRemove ← true
2. for \( i = 0 \) to cnt+1 do
   3. \( R \).push(\( L[i] \))
   4. \( C \) ← FindChildNode(\( L[i], T \))
   5. for \( j = 0 \) to \( C \).size do
      6. if \( T(C[j]) \) is not \( L[i + 1] \) then
         7. \( R \).push(\( C[j] \))
   8. end if
   9. end for
10. end for
11. \( T \) ← IndexRemovalNodes(\( R, T \))
12. while (FlagRemove) do
   13. \( R' \) ← FindChildNode(\( R, T \))
   14. \( T \) ← IndexRemovalNodes(\( R', T \))
   15. \( R \) ← \( R' \)
   16. if \( R \) is empty then
      17. FlagRemove ← false
   18. end if
19. end while
20. return \( T(V, E) \)

When creating a path using the RHRRRT* algorithm, obstacles detected when reaching the sensor distance in an unknown environment may overlap with the current path. Simply removing nodes adjacent to obstacle locations from the vertex set will result in a truncated graph, which causes problems when calculating the cost later. Therefore, to avoid the additional computational costs that this would incur, if we find and delete the child node of the node deleted above, and also find and delete the child node of the deleted node, all paths caught in the obstacles can be removed.

### 3.3 Obstacle perception

Image sensors used alone are limited in the number and fidelity of obstacles that can be identified and tracked in urban environments. There may be obstacles in the unlearned class, and even if this class is included in the dataset, there are several disadvantages; when an entirely new type of product comes out, it is necessary to build a learning dataset again. However, when using 3D LiDAR, any obstacle in the city center can be used for obstacle avoidance because the relative distance to the drone is measured.

If all 3D LiDAR data are used, the onboard processing units would be heavily overburdened. Hence, preliminary filtering is necessary based on point cloud distances. Furthermore, down-sampling is performed using voxel grid filtering on adjacent points to further reduce the burden of unnecessary calculations. The obtained obstacle candidates thus require a much smaller number of point clouds. Figure 5 illustrates a point cloud of raw data and the change in the distribution of points when each filter is applied.

![Figure 5](image-url)
In the raw data result in Figure 5A, you can easily and precisely distinguish the shape of the street lamp and tree. In the down-sampling result in Figure 5B, points located far away do not change much as there is a large gap between the points from the beginning. However, when observing the street lights near the drone in the case of the distance filter in Figure 5C, the points farther away are filtered out but are still dense enough to distinguish the shape of the street light. Finally, when comparing the down-sampling filter and distance filter, one can see that the number of points is significantly reduced while detecting the important street lights, which are obstacles to be avoided right away, around the drone. Table 1 compares the number of point clouds measured in each case.

When the obstacle candidate group is found as above, it must be registered in the obstacle queue to be used for real obstacle avoidance. To avoid data instability and noise, if data such as obstacle candidates are directly used in the algorithm, the sensor data are filtered directly, so the data itself may be unstable or contain a lot of noise. The pseudo-code to implement this is in Algorithm 5.

Table 1 presents the number of point clouds measured in each case.

<table>
<thead>
<tr>
<th></th>
<th>Raw data</th>
<th>Down-sampling</th>
<th>Distance filter</th>
<th>Mixed filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trees</td>
<td>66670</td>
<td>7822</td>
<td>5606</td>
<td>313</td>
</tr>
<tr>
<td>Street lights</td>
<td>62888</td>
<td>7635</td>
<td>5660</td>
<td>321</td>
</tr>
<tr>
<td>Utility poles</td>
<td>73426</td>
<td>8229</td>
<td>5800</td>
<td>335</td>
</tr>
</tbody>
</table>

4 | RESULTS

This section reports the performance results of the proposed algorithm in various urban scenarios comprising obstacles such as trees, street lights, utility poles, and wires that are found mainly in urban environments. Each obstacle has its own set of characteristics that factor into the model’s obstacle avoidance algorithms.

In this section, we first analyze the performance of our algorithm in a Gazebo simulation. Then, we compare its performance metrics to those of a state-of-the-art method in 3D path planning, EGO-Planner [19]. Lastly, we test the algorithm’s efficacy in a real-world application.

4.1 | Simulation environment

Quantitative comparisons and analyses were performed in a simulation environment before validating our model’s real-world performance. All algorithms were developed in the C++-based ROS Melodic, which was built in Linux 18.04 LTS. An urban environment was designed using the Gazebo simulator, and street lights and trees were installed in front of the landing pad as obstacles to be used in the avoidance scenario. The appearance of the constructed simulation is shown in Figure 6.

The comparison test path began at an altitude of 8 m at a distance of 20 m from the landing site. The path was followed at a speed of 0.6 m/s, and obstacle clearance was set to 1.4 m.

4.2 | Vertical landing path

To perform obstacle avoidance, it was necessary to check whether the path for performing vertical landing was feasible. The vertical landing path creates a path that allows the drone to land vertically at the last landing point no matter where the drone is located, so the path for each altitude must be confirmed.

Figure 7 shows the results of routes created at 2, 4, and 8 m. Figure 7A displays the path created at 2 m. According to the trajectory, the drone was simulated to land at 0.4 m/s by initially increasing its altitude followed by landing vertically past the obstacles. Approaching
without raising the altitude and landing vertically may not be the optimal route because it requires a lot of acceleration. In Figure 7B, the flight path began at an altitude of 4 m; hence, the drone necessarily climbed less than during the case in which it began at a 2-m altitude. Nevertheless, it successfully landed vertically. In Figure 7C, the flight path began at an altitude of 8 m; in this case, it can be seen that the route was created to land vertically without climbing much.

4.3 Collision-free vertical landing

Here, we compare the proposed algorithm’s performance with the state-of-the-art 3D planning method, EGO-Planner in three different scenarios. Because EGO-Planner is a general path planning method, we used some vertical landing path samples, $Q$, as waypoints for performance comparisons under equivalent conditions. To compare the vertical landing performance of the two algorithms, we measured and compared numerical values such as the travel distance, travel time, and landing angle and set the initial point for each trial to begin at 8 m above the surface. The same landing waypoint was used for both models. For each trial, we recorded the travel distance and time from the initial point to the landing waypoint, and we calculated the average landing angle. We conducted 10 trials for each algorithm to ensure the statistical significance of our results.
Figure 8 visualizes the trajectories of each scenario. Figure 8A–C reflects the trajectories using the proposed algorithm, in which all paths generated by the algorithms in our experiments achieved vertical landings while maintaining a minimum distance from all obstacles. Figure 8D–F visualizes the trajectories provided by the EGO-Planner. Notably, the efficiency of the generated paths varied depending on the model’s optimization level and the presence of obstacles. Some paths may be more efficient if highly optimized, while others may result in longer travel times due to detours around obstacles. Notably, an issue arose with the green trajectory shown in Figure 8F. Here, a thin electric wire was detected relatively late, causing the drone to take a long detour to avoid it. This result suggests that the effective use of EGO-Planner may require excessive hyperparameter tunings, such as obstacle detection thresholds and path weighting factors, to suit the specific environmental conditions and

![Figure 8](image)

**Figure 8** Trajectory visualizations per case: (A–C) the results from the landing angle control-based receding horizon rapidly exploring random tree star (LAC-RHRRT*) model for each scenario and (D–F) the results from the Euclidean signed distance field-free gradient-based local (EGO) planner in each scenario.

![Figure 9](image)

**Figure 9** Performance comparison graphs showing (A) travel distance, (B) travel time, and (C) approach angle. EGO, Euclidean signed distance field-free gradient-based local; LAC-RHRRT*, landing angle control-based receding horizon rapidly exploring random tree star.
obstacles encountered in the field. This would, of course, not be suitable for practical applications.

Figure 9 displays the performance index comparison graphs for the proposed algorithms. Overall, the results show that the proposed algorithms achieved significantly shorter travel distances and times than the EGO-Planner for all tested scenarios. Furthermore, the proposed algorithm produced landing angles that were almost three times smaller than those produced by the EGO-Planner. These findings suggest that the proposed algorithms outperform the state-of-the-art EGO-Planner in terms of both efficiency and accuracy in the tasks of obstacle avoidance and vertical landing.

5 | CONCLUSIONS

In this paper, we developed an algorithm that enables the vertical landing of drones at designated landing points in urban areas while effectively avoiding collisions with obstacles. The algorithm was designed to address the unique challenges associated with drone delivery services in urban environments.

To ensure a safe landing, the drone must land vertically during its final stage. However, landing directly from a high altitude is inefficient; hence, an optimal path for horizontal and vertical movements must be created. In this study, we developed an algorithm that generates the optimal path for vertical landing by applying the path planning rules leveraged by missile guidance systems. Additionally, we incorporated a 3D LiDAR sensor to detect obstacles in real time while filtering them for avoidance during the landing process in urban environments. To provide obstacle avoidance capabilities for the proposed algorithm, the existing 2D RRT-based algorithm was extended to 3D, and further research was conducted to enhance it as per the requirements of this study. Specifically, the algorithm was designed to ensure that a viable avoidance path could be generated throughout the entire landing process.

The algorithm developed through this study was verified in a real-time simulation environment through a sensor with the same specifications as the real environment and a six-degree-of-freedom drone model. Furthermore, we compared the proposed algorithm with EGO-Planner, a state-of-the-art 3D planning algorithm, and the results show that the proposed algorithm achieved excellent performance in terms of efficiency avoidance when landing vertically.

Safe landing in urban areas at a normal flight state can be achieved by setting a conical airspace in the altitude axis direction based on the center point of the landing pad, and the safest landing trajectory under this condition has known to be a vertically descending trajectory toward the center point of the landing pad. The proposed contribution in this paper has highly effectiveness and efficiently applicable to real-world flight performance. However, because the proposed contribution does not consider the conical safety airspace nearby the landing pad, explicit safe landing trajectory performance cannot always be achieved. In order to overcome these limitations related to explicit and statistical deterministic landing performance, large-scale Monte Carlo simulations and flight tests under various conditions will be planned and conducted.

CONFLICT OF INTEREST STATEMENT

The authors declare that there are no conflicts of interest.

ORCID
Hanseob Lee © https://orcid.org/0000-0003-4186-9418

REFERENCES

AUTHOR BIOGRAPHIES

Hanseob Lee received his BS degree in Aerospace Engineering from Korea Aerospace University, Goyang, Republic of Korea, in 2015, and his MS and PhD degrees in Aerospace Engineering from KAIST, Daejeon, Republic of Korea, in 2017 and 2021, respectively. Since 2022, he has been working for the Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea, where he is now a researcher. His research interests are autonomous applications in aerial robotics, continuous reinforcement learning, and vision-based tracking.

Sungwook Cho received his BS degree in Aerospace Engineering from Korea Aerospace University, in 2010, and his MS and PhD degrees in Aerospace Engineering from KAIST, Daejeon, Republic of Korea, in 2012 and 2017, respectively. Since 2019, he has been with the Department of Aeronautical and Mechanical Engineering, Cheongju University, Republic of Korea, where he is now an associate professor. His research interests are vision-enabled autonomous system development and its applications with sensor-fusion in the field robotics.

Hoon Jung received his MS and PhD degrees in Industrial Engineering from Iowa State University, IA, USA, and the University of Missouri, MO, USA, in 1997 and 2001, respectively. Since 2002, he has been working for the Electronics and
Telecommunications Research Institute, Daejeon, Republic of Korea, where he is now a director. His main research interests are supply chain management, logistics automation and information systems, drone and robot delivery system, and prognostic health management (PHM).

How to cite this article: H. Lee, S. Cho, and H. Jung, Real-time collision-free landing path planning for drone deliveries in urban environments, ETRI Journal 45 (2023), 746–757. DOI 10.4218/etrij.2023-0129