Photorealistic simulation platform for autonomous landing of fixed-wing aircraft

1st Weili Xue
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
u201613306@sjtu.edu.cn

2nd Zhen Sun
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
zhensun@sjtu.edu.cn

3rd Kehui Ma
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
khma@stu.xidian.edu.cn

4th Ling Pei✉
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China
ling.pei@sjtu.edu.cn

***Abstract*—To investigate the vision-based autonomous landing of fixed-wing aircraft, we propose a photorealistic simulation platform that leverages ROS, Unreal Engine, and Pixhawk 4. This platform adopts a software-in-the-loop model consisting of rendering, control, communication, and sensor modules. To achieve realistic rendering control, the seawater hydrodynamics and fixed-wing aircraft dynamics are modeled. Based on the platform, we create simulation scenarios under different weather and disturbance conditions for autonomous landing of the aircraft carrier. The platform solves the problem that datasets of related scenes are difficult to collect and experiments are difficult to carry out. To verify the developability of the platform, we design and implement a runway line feature point extraction method (vision-based pose estimation algorithm), and evaluate the performance of the method under various conditions. Experiments show that our software-in-the-loop platform enables vision-based algorithm verification and supports simulation research for the autonomous landing of fixed-wing aircraft.**

***Keywords*—Photorealistic, Simulation platform, Fixed-wing aircraft, Landing**

I. Introduction

The evolution of aircraft landing on ships has undergone various stages, starting with the Landing Signals Officer (LSO) method, followed by the implementation of optical landing aid systems and finally, the advent of all-weather landing systems. In the early days of naval aviation, the LSO method was utilized to guide aircraft during landing, however, this approach was limited in its applicability due to the high speed of aircraft during the landing process. The 1960s saw the introduction of the Fresnel Lens Optical Landing System (FLOLS), which continues to be in use today, however, its shortcoming lies in the limited penetration and visibility of the light during cloudy weather conditions. To address this issue, long-range optical landing aid system [1] [2] was developed in the 1990s that utilized the superior penetration and straightness of laser beams to provide the pilot with landing information from a greater distance. Simultaneously, the All Weather Carrier Landing System (AWCLS) with a higher degree of autonomy has also matured, with a precision tracking radar on the carrier deck serving as its core component. The precision tracking radar measures the aircraft's motion and uses predictive estimation methods, such as Kalman filtering and neural networks, to compensate for interferences signals, such as deck motion, and assist the pilot in completing the landing operation. Recently, to address the challenges posed by unknown or GNSS-rejected environments, and strong electronic interference, the landing scheme has evolved from relying on single sensors to utilizing a multi-sensor approach. Among these, vision-based landing schemes have garnered increased attention from researchers [3]. Vision-based autonomous landing involves the use of an airborne camera to acquire image information of the target and then, through image processing techniques, calculates the relative attitude of the aircraft and the deck. The calculated attitude information is then transmitted to the flight control system to complete the autonomous landing process. Vision-based autonomous landing offers advantages such as low cost [4], high autonomy [5], and rich information [6].

Several institutions have undertaken research on vision-based landing and achieved noteworthy results. McCarthy et al. [5] explored the feasibility of a vision-based landing scheme for autonomous control and feedback. Zhang et al. [6] proposed the use of airborne forward-looking camera-acquired images to extract various navigation information, such as attitude and position, through a significance analysis method based on spectral residuals. Long et al. [4] analyzed the benefits of vision-based landing from the perspectives of low cost and anti-jamming capabilities. Additionally, they also discussed the design considerations for integrating vision-based schemes with radar-based and other schemes for landing. There have also been numerous studies dedicated to enhancing the performance of algorithms for feature extraction and pose estimation during landing [7] [8]. However, due to the absence of the high-fidelity simulation platform, many of the current algorithms for autonomous landing on aircraft carriers have not been tested in a relatively realistic simulation environment. The landing datasets utilized for testing purposes often have limited views or lack images of the moving ocean, and are usually of a single type, precluding experimentation under varying weather and disturbance conditions. Thus, the development of a high-fidelity simulation platform would greatly benefit research into vision-based carrier landing. Based on the above considerations, the simulation platform for autonoumous landing must have the following requirements: (1) It should reflect the dynamic characteristics and environmental characteristics of a real aircraft carrier as much as possible. (2)It should support algorithm verification and be able to generate datasets in different environments, e.g, different weather and different disturbance conditions.

Based on the above motivation, we build a simulation platform for autonomous landing of fixed-wing aircraft that has near-realistic rendering and dynamic models. To achieve photorealism, we model the dynamics of seawater and fixed-wing aircraft. This dynamic modeling simplifies the calculation process and reduces the resource consumption of rendering scenes on the premise of ensuring simulation accuracy. The platform implements software-in-the-loop from sensor, communication, control, and rendering modules. Finally, we use the platform to make various datasets. Based on different trajectories and weather conditions, we carried out the runway line reconstruction method to achieve pose estimation.

One of the ideal trajectories of aircraft landing is shown in Figure 1.

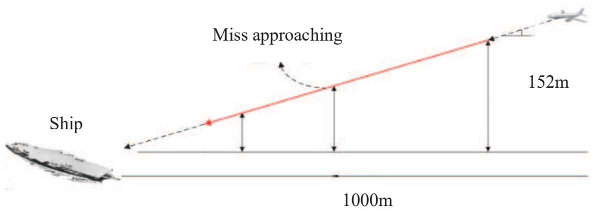


Fig. 1. Fixed-wing aircraft landing process diagram

The main contributions of our work are summarized as follows.

- We develop a photorealistic simulation platform for the autonomous landing of fixed-wing aircraft, which enables near-realistic rendering and dynamic models. This platform implements software-in-the-loop from sensor, communication, control, and rendering modules. We generate autonomous landing datasets under various weather(rain, fog) and disturbance conditions(random noise, horizontal noise lines) based on this platform.
- To model a photorealistic simulation scenario, we implement dynamic modeling for seawater and fixed-wing aircraft in the proposed platform. Meanwhile, our modeling considers the balance between photorealism and operational efficiency.
- With various experiments on our proposed platform, we demonstrate that the platform can be used to collect aircraft carrier landing datasets under various external conditions and support advanced algorithm development. This solves difficulties associated with datasets creation and simulation experiments in the field of fixed-wing landing on aircraft carrier.

The remainder of this paper is organized as follows: Section II introduces related work in the field of simulation platforms, seawater dynamics modeling and pose estimation. Section III introduces the architecture of the proposed simulation platform. Section IV introduces the dynamic modeling and implementation of seawater and fixed-wing aircraft. Section V introduces the runway line reconstruction method. Section VI describes the experimental portion of the study. Section VII summarizes our work and proposes future work.

II. Related Work

Aircraft simulation platform. Due to their excellent developability, most simulation platforms are based on ROS using Linux systems [9] [10] [11]. Tim et al. [12]developed a fixed-wing simulation platform. It is a classic project often used to teach novice engineers how to develop drone simulation platforms. However, it lacks the implementation of high-level algorithms. Xiao et al. [13]implemented a powerful UAV simulation platform XTDrone which is based on ROS and PX4. This platform supports SLAM algorithms such as VIO, as well as multi-UAV path planning. However, Gazebo's rendering falls short of photorealism. Shital Shah [14] et al. completed a great simulation platform AirSim, which combines the excellent visualization capability of Unreal Engine with its own dynamics and control system. However, AirSim is not suitable for all scenarios due to its limited vehicle and drone models. For example, it only supports quadrotor aircraft and does not implement a fixed-wing dynamic model. Integrating the previous work, we complete a photorealistic simulation platform for the autonomous landing of fixed-wing aircraft.

Dynamics modeling. In the field of sea surface dynamics modeling, modeling and animating the ocean's surface, mainly rely on two approaches: 1) the approximation of ocean dynamics with parametric, spectral or hybrid models and the use of empirical laws from oceanographic research. and 2) physics-based methods that use Navier–Stokes equations to represent breaking waves and, more generally, ocean surfaces near the shore. For example, Mark et al. [15] proposes different models for global ocean modeling at different resolution requirements. The model can simulate climate change and eddy current activity. However, it cannot be used for rendering pipelines and simulation scenes due to their computational complexity. Considering the balance between photorealism and operational efficiency, we decided to use the sea surface height function [16] to simulate the ocean surface and ship motion. In the

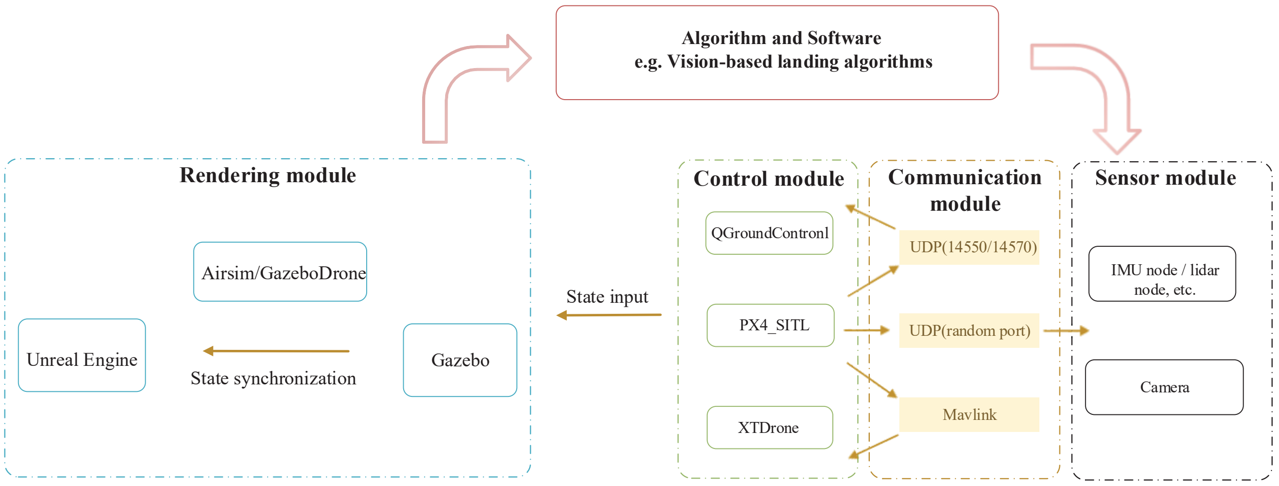


Fig. 2. Platform architecture

TABLE I
DISPLAY FORM OF AIRCRAFT CARRIER LANDING PLATFORM SUPPORT FUNCTIONS

Weather simulation	Interference simulation	Customizable dynamics model	Track control	Supported sensors
rain, fog, snow	random, horizontal	fixed-wing, quadrotor	real-time, pre-planning	camera, IMU

1 Random is Gaussian random noise. The pixels in a unit area share the same noise value that satisfies the Gaussian random distribution.
2 Horizontal is the horizontal noise line. The horizontal noise lines increase the noise area on the horizontal line.
3 The fixed-wing dynamic model has now implemented the Cessna C-172.
4 Real-time track control relies on the keyboard. Meanwhile, the pre-planning of trajectories relies on QGroundContronl.

field of fixed-wing dynamics modeling, there are many mature theories and models. We chose the most classic one, Cessna C-172 [17]

III. SIMULATION PLATFORM

In this section, we first describe the structure of the platform. Then, we introduce the modules that make up the platform in detail, such as the rendering module, control and communication module, and sensor module.

A. Platform architecture

Table I shows the functions currently supported by our platform. Figure 2 shows the detailed architecture of the simulation platform. The platform contains rendering, communication, control, and sensor modules. Among them, the communication module mainly uses the Mavlink protocol and topic communication. The bottom layer motor of the control module is simulated by PX4_SITL, and the upper QGroundControl is used to plan the flight path of the aircraft. XTDrone is used for SLAM algorithm development. The aircraft flight status is sent to Gazebo for display. The rendering module reads the aircraft states from Gazebo and synchronously displays them in the scene simulated in Unreal Engine. The sensor module mainly includes the camera. The platform can feed back the pose information of the sensor module to the rendering module through the control and communication module. The above modules construct a software-in-the-loop pipeline.

B. Rendering module

1) Rendering in Unreal Engine: Modern visual engines, such as Unreal Engine, exhibit far better visual performance than Gazebo, which is used for robot simulation display on the Linux platform. An important reason is that the former realizes ray tracing and physics-based rendering. We are able to sculpt realistic models in professional modeling software such as Blender. We can provide normal maps, specular maps, and noise maps to render models and obtain near-realistic materials [18].

C. Control and communication module

The underlying flight control and dynamics model are implemented based on PX4. The upper-layer communication scripts, control algorithms, and state estimation are implemented by XTdrone. The communication between the aircraft and the simulation scene in Gazebo is realized through the Mavros module of ROS. Topic communication is used between nodes, and the specific message format can refer to [13]. At the same time, we realize point-to-point flight control and modification of flight parameters through QGroundControl.

D. Sensor module

A real camera has many parameters, such as external parameters, internal parameters, and distortion. Therefore, the parameters of the simulated camera used by the platform are similar. The internal parameters of the camera are:

$$f_x = f_y = \frac{Width/2}{tan(fov/2)} \tag{1}$$

$$K = \begin{bmatrix} f_x & 0 & \frac{Width}{2} \\ 0 & f_y & \frac{High}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

In the formula, *width* and *high* are the length and width of the photo, and *fov* is the camera field of view.

The camera distortion parameters default to 0, which can be customized in [14].

IV. Dynamics Modeling

In this section, we introduce the principles and implementation of seawater dynamics modeling and fixed-wing dynamics modeling. The state variables used in the derivation process can be found in Table II.

TABLE II
THE VARIABLES USED IN THE DERIVATION PROCESS

Variables	Description
ρ, u, p	Fluid density, fluid velocity and pressure
μ, ϕ, g	Fluid viscosity, velocity potential, gravitational acceleration
U	$U = gh$ is the gravitational potential of the fluid h is the sea level
k	The wave vector, which points in the direction of the motion of the wave, $k = \frac{2\pi}{\lambda}$, λ is wavelength
ι	$\iota = \frac{v^2}{g}$,v is the wind speed
ω, w, A	Angular frequency, wind direction, seawater amplitude
θ	θ is theangle between wave vector and wind direction
F^i	Inertial coordinate system(North-East-Down)
F^v	Aircraft coordinate system as Figure 3(a)
F^{v1}	Aircraft-1 coordinate system as Figure 3(b). The F^{v1} coordinate system is obtained by rotating F^v coordinate system around the K^v-axis by an angle ψ
F^{v2}	Aircraft-2 coordinate system as Figure 3(c). The F^{v2} coordinate system is obtained by rotating F^{v1} coordinate system around the j^{v1}-axis by an angle θ
F^b	Body coordinate system as Figure 3(d)
P_n, P_e, P_d	The inertial north, east, down of the MAV along the i^i, j^i, k^i axis in the F^i coordinate system
u, v, w	velocity along i^b, j^b, k^b axis in F^bcoordinate system
ϕ, θ, φ	Roll, Pitch, Yaw angle
p, q, r	Roll, Pitch, Yaw velocity along the i^b, j^b,k^b axis in the F^bcoordinate system

A. Seawater hydrodynamic modeling

In the selected scene, the datasets images are mostly images of the ocean, so it is important to render high-fidelity seawater. We make the following assumptions:(1) Seawater is an incompressible fluid. (2) The current of seawater does not cross. (3) The density of seawater is constant. (4) Turbulence is ignored. From the Navier-Stokes equation and the fluid continuity equation, under the above assumptions, the mass conservation and momentum conservation constraints of seawater can be obtained:

$$\begin{cases} \frac{\partial \phi}{\partial t} + \frac{1}{2}(\bigtriangledown \phi)^2 = -U - p \\ \bigtriangledown^2 \phi = 0 \end{cases} \tag{3}$$

According to plane wave theory and substituting into the directional wave spectrum (Philippe spectrum), combined with the empirical formula of ocean statistics [19], the approximate solution of the sea surface height can be obtained:

$$h(x,t) = \sqrt{\frac{A}{2}}(\varepsilon_r + i\varepsilon_i) \sum_k \frac{|k \cdot w|}{k^2} e^{\frac{i}{k\iota}} \tag{4}$$

where x are sea level coordinates and $\varepsilon_i, \varepsilon_r \sim N(0,1)$ are normally distributed random numbers. For a more detailed proof of the seawater height function, refer to [16]. Then, we use the height function in Unreal Engine to simulate fluctuating sea surface.

B. Fixed-wing aircraft dynamics modeling

The coordinate systems we used are shown in Figure 3, in which Figure 3(e) shows the definition of the aircraft motion axis. For fixed-wing aircraft, the dynamic system is highly relative to the control system.

There is a relationship of :

$R_v^b(\varphi,\theta,\psi) = R_{v1}^b(\varphi) R_{v1}^{v2}(\theta) R_b^{v1}(\psi)$, and the transformation matrix from the aircraft coordinate system F^v to the body coordinate system F^b is:

$$R_v^b(\varphi,\theta,\psi) = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\varphi s_\theta c_\psi - c_\varphi s_\psi & s_\varphi s_\theta s_\psi - c_\theta c_\psi & s_\varphi c_\theta \\ c_\theta s_\theta c_\psi + s_\varphi s_\psi & c_\varphi s_\theta s_\psi - s_\varphi c_\psi & c_\varphi c_\theta \end{bmatrix} \tag{5}$$

By linking the translation velocity and position, the constraints of position can be obtained:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} P_n \\ P_e \\ P_d \end{bmatrix} = (R_v^b)^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{6}$$

When the plane is performing translational motion, combined with Newton's second law, the force and velocity in body coordinates, as shown in Figure 3(d) can be expressed as:

$$m(\frac{\mathrm{dV_g^b}}{\mathrm{d}t_b} + \omega_{b/i}^b \times V_g^b) = f^b \tag{7}$$

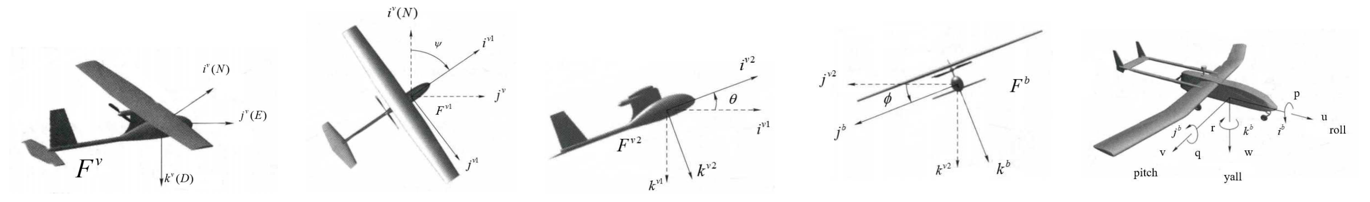
where $V_g^b = (u,v,w)^T$,$\omega_{b/i}^b = (p,q,r)^T$.

and $f^b = (f_x, f_y, f_z)^T$represents the sum of all external forces in the body coordinate system. After substituting Formula (7) for simplification, we can obtain the velocity constraints:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} \tag{8}$$

In the same way, combined with the angular momentum and angular velocity projection, the constraint of the aircraft's rotational motion can be obtained as:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \Gamma_1 pq - \Gamma_2 qr + \Gamma_3 l + \Gamma_4 n \\ \Gamma_5 pr - \Gamma_6 (p^2 - r^2) + \frac{m}{J_y} \\ \Gamma_7 pq - \Gamma_1 qr + \Gamma_4 l + \Gamma_8 n \end{bmatrix} \tag{9}$$



(a) aircraft coordinate (b) aircraft-1 coordinate (c) aircraft-2 coordinate (d) body coordinate (e) motion axis

Fig. 3. Definition of the coordinate systems

where:

$$\begin{cases} \Gamma_1 = \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma}, \Gamma_2 = \frac{J_z(J_z - J_y) + J_{xz}^2}{\Gamma} \\ \Gamma_3 = \frac{J_z}{\Gamma}, \Gamma_4 = \frac{J_{xz}}{\Gamma}, \Gamma_5 = \frac{J_z - J_x}{J_y} \\ \Gamma_6 = \frac{J_{xz}}{J_y}, \Gamma_7 = \frac{(J_x - J_y)J_x + Jxz^2}{\Gamma}, \Gamma_8 = \frac{J_x}{\Gamma} \\ \Gamma = J_x J_z - J_{xz}^2 \end{cases} \tag{10}$$

and:

$$J = \begin{bmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{bmatrix} \tag{11}$$

where J is the rotational inertia matrix of the aircraft, in which the diagonal elements J_x, J_y, J_z are inertial momentum, and the off-diagonal elements are inertial products. The above equations (6)(8)(9) give the fixed-wing aircraft's kinematic and dynamic constraints of 6-DOF and 12-states, respectively. The detailed derivation process can be found in [12]. Numerical solvers that simulate rigid body dynamics use the open dynamics engine(ODE) [20].

V. Vision-Based Pose Estimation Algorithm for Platform Verification

In this section, we first introduce the algorithm pipline for estimating the aircraft attitude in the fixed-wing landing scenario. Next, we briefly describe the implementation details and code of the algorithm.

A. Method and algorithm pipeline

The pipeline design of our method is shown in Figure 4. First we need a target detection algorithm to process the image frames of the datasets to obtain the region of interest (ROI) of the aircraft carrier. In view of the requirement for real-time operation during the landing process, the target detection of the first step of the algorithm must not drag down performance. The simulation environment utilized in this study has a rendering pipeline established in Unreal Engine, yielding a maximum output frame rate of 60 frames per second (FPS). As a result, it is necessary to select a target detection algorithm with sufficient operational efficiency, specifically, a rate greater than 60 FPS. After conducting a thorough evaluation, YOLOv5 is deemed an appropriate choice, as it has a maximum frame rate of 140 FPS and demonstrate satisfactory performance in detecting small targets. Next, we binarize the ROI and perform edge detection and corner extraction on the binary image. Then, we use a priori infomation to obtain the priori region of the runway line and use this region to filter out the corner points. Finally, we use the corner points to reconstruct the new runway line, find the endpoints, and run the Perspective-n-Point algorithm [21] to determine the pose estimation. The algorithm pipeline is shown in Algorithm 1.

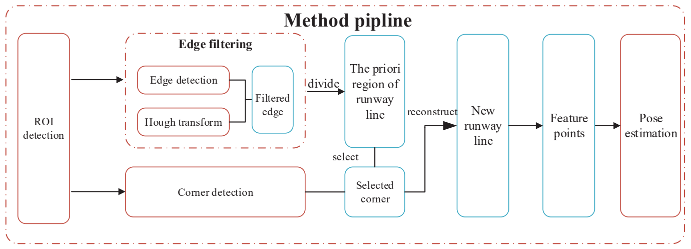


Fig. 4. Pipeline of the proposed method

Algorithm 1: Method pipeline

```
Data: datasets collected based on the platform
Result: Pose of Euler Angles
while 1 in all frames do
    // In this example we use YOLOv5
    Object Detection;
    threshold() and medianBlur();
    Canny() HoughLinesP();
    SelectEdge() and SetArea();
    img = cornerHarris();
    Normalize();
    for pixel in img do
        if (pixel.color > threshold and is_inarea(pixel)
         == True) then
            add pixel to Points[];
        end
    end
    // New runway line fitting
    Reconstruction::RANSAC(Points[]);
    // Pose estimation
    SolvePnP();
end
```

B. Details of the algorithm

After we obtain the region of interest(ROI) of the airport runway, we also get the pixel height and width of the aircraft carrier in the picture, which are recorded as *Height* and *Width*.

After preliminary experiments, we find that due to blurry pictures in the ROI area and many interference lines, the edge detection has the following problems: (1) The detected runway edge is not a complete straight line but is instead composed of multiple small line segments oriented in slightly different directions; (2) the edges of other parts of the carrier can also be detected, which introduces noise to the runway line edge detection. Based on the above facts, we decided to jointly use Canny [22] operator edge detection, Hough Transform [23], and Harris corner detection [24].

The constraints we use to filter runway lines are:

$$\begin{cases} \theta \epsilon (\theta_{min}, \theta_{max}) \\ L \epsilon (L_{min}, L_{max}) \end{cases} \tag{12}$$

As shown in Formula (12), θ is the horizontal angle of the line, where θ_{min} and θ_{max} are the angle ranges of the aircraft carrier runway line under the current trajectory. L is the length of the line, where L_{max} is the maximum length. According to the runway line length of the real model of the aircraft carrier, $L_{max} = 0.6 * Width$ (*Width* is the width of the ROI), and L_{min} is the minimum length of the edge line. We take a empirical value for the minimum length. For each edge of the runway line chosen, we establish a rectangle where the rectangle is oriented along the direction of the edge. The ideal runway area can be obtained by superimposing all rectangles. This step is shown in Figure 5.

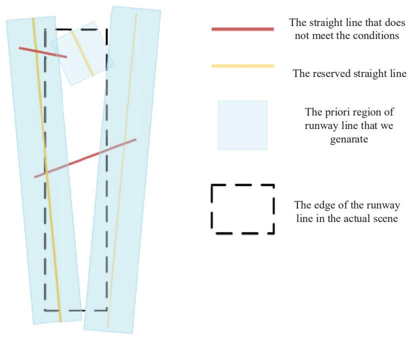

The straight line that does not meet the conditions
The reserved straight line
The priori region of runway line that we genarate
The edge of the runway line in the actual scene

Fig. 5. Edge selection and runway division

C. Runway line reconstruction

We use a priori region, which is shown in Figure 5, to filter corners and group them into sets $\{S_a\}, \{S_b\}$. Then we use the RANSAC algorithm to mitigate the outliers.As long as the new runway line is constructed, we also obtain the endpoints of the runway line. We denote these endpoints as P_0, P_1, P_2, P_3. Because the camera intrinsic matrix K is known, the real coordinates and the accurate pixel coordinates of the four points P_0, P_1, P_2, P_3 are also known, and we apply the Perspective-n-Point algorithm to estimate the camera pose.

VI. Experiments

In this section, we first introduce the software and hardware configuration. Then, we show the details of the simulation environment construction. Next, we describe the experimental steps of the algorithm. Furthermore, we conduct experiments under different weather conditions and trajectories and evaluate the performance of the algorithm in different situations. This demonstrates that our platform can support experiments under a variety of conditions and further supports algorithm development and verification.

A. Configuration

1) Software: The software environment is Ubuntu 18.04, Python version 3.7.2, AirSim version 1.7.0 [14], Unreal Engine 4.25 [25], ROS-neotic [26],PX4, Gazebo9.1, QGroundControl, and XTDrone [13].

2) Hardware: The hardware equipment used in this experiment includes an Nvidia RTX2060 GPU and an Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz.

B. Construction details of the aircraft carrier fixed-wing landing environment

In this section, we briefly describe the simulation environment components.

1) Construction of a photorealistic sea: Using the height function in formula 4, we can generate the height spectrum in Unreal Engine and then obtain the offset spectrum. Knowing the offset spectrum, we implement IFFT to obtain the (x, y, z) offset. The offset is superimposed to obtain the offset texture and then the normal texture can be calculated through the offset texture. Finally, we use the normal texture for rendering. This process is shown in Figure 6. Considering the aircraft carrier as a rigid body with a center of mass, and assigning the offset in the Z direction of one point to the center of mass, the vibration of the aircraft carrier deck at sea can be roughly simulated. In addition, to get the photorealistic seawater in Unreal Engine, we also need to set the water depth, caustics, and foam. The final built sea surface is ideal (without shoals, edges, and rapids), as shown in Figure 7.

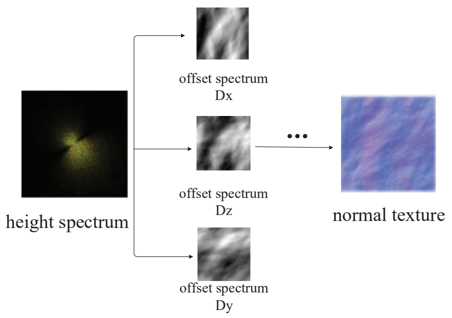


Fig. 6. Normal texture of the generated seawater

2) Other details of the aircraft carrier fixed-wing landing environment: The prototype of the aircraft carrier used in the simulation is the Nimitz Class Aircraft Carrier, as shown in figure 8(a). The fighter prototype used for landing is the Lockheed Martin F-35 Lightning II, as shown in Figure 8(b). In addition, to simplify the experiment, for the fixed-wing dynamic model we temporarily use the Cessna C-172.

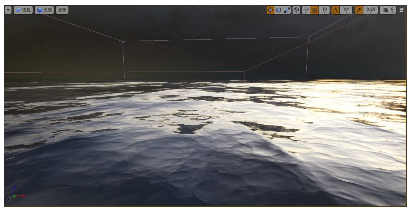
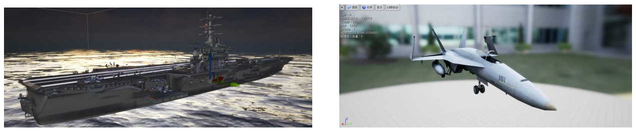


Fig. 7. Ideal sea rendering in Unreal Engine



(a) Nimitz Class Aircraft Carrier (b) Lockheed Martin F-35 Lightning

Fig. 8. Aircraft carrier and fixed-wing aircraft prototype display

3) The trade-off between photorealism and operational efficiency: Datasets rendering supports various resolutions (360P, 480P, 1080P, 2K, 4K). The higher the resolution is, the higher the computing power required for rendering. The relatively simple dynamic modeling of the sea surface and fixed wings can significantly reduce the computing power requirements for rendering and improve the frame rate performance, but it will lose the realism of the picture. Table III shows the relationship between the frame rate and resolution under the current experimental environment. In this table, we compare our platform's fixed-wing landing scenario, the default scenario in Gazebo, and the Airsim scenario based on our platform with the fixed-wing dynamics removed. Obviously, compared to Gazebo, we sacrifice operating efficiency for the improvement of photorealism. At the same time, the frame rate of our platform is not much different from that of native AirSim at high resolution. This means that the platform achieves the need for a customizable fixed-wing dynamic model without sacrificing operational efficiency. In addition, the frame rate performance of the current environment at high resolution (1080P) can meet most of the experimental requirements.

TABLE III
THE FRAME RATE PERFORMANCE OF OUR ENVIRONMENT UNDER DIFFERENT RESOLUTION REQUIREMENTS

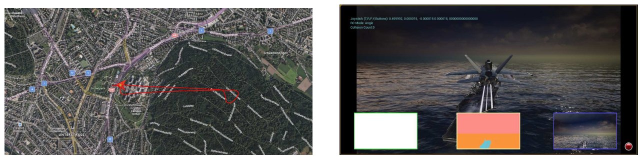
Resolution	480P	720P	1080P	2K	No-re	Fixed-wing
Ours	39	34	15	7	20	✓
Gazebo	100+	84	67	NULL	60	✓
Airsim	41	36	16	7	23	✘

1 The unit of frame rate is FPS. When rendering the datasets, the environment is not displayed in real time.
2 No-re refers to the frame rate at which the platform is running without rendering the datasets.
3 NULL indicates that the current scene cannot perfectly support the condition.

C. Vision-based pose estimation algorithm implementation

In this part, we control the fixed-wing aircraft to fly a fixed trajectory in QGroundControl and generate datasets. Subsequently, we implement our algorithms based on the above datasets and analyze the results of the experiment.

1) Flight demonstration: We show the entire process of a fixed-wing aircraft from take-off to landing along a fixed trajectory in a demo video. The flight trajectory is controlled by QGroundControl. The coordinates of the ideal flight landing point in the Unreal Engine world scene are (0, 0, 24.3) (the above coordinate units have been converted to meters). Figure 9(a) shows the trajectory of the aircraft. The landing algorithm uses the PX4 default method AUTO_LAND. The dynamic of the fixed-wing aircraft uses the classic Cessna C-172 [27]. Figure 9(b) shows the flight demonstration with the depth image, camera image and semantic segmentation image.



(a) QGroundControl (b) Unreal Engine

Fig. 9. The trajectory shown in QGroundControl and the corresponding flight process in Unreal Engine

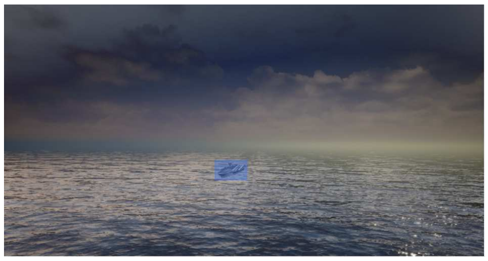
2) Datasets generation: For the multiconditional experiments in the next section, we generate datasets of fixed-wing aircraft landings on the deck of an aircraft carrier under different trajectories, weather(rain, fog) and interference conditions (random noise, horizon noise lines), as shown in Table IV.

TABLE IV
THE FORMAT OF THE DATASETS CONFIGURATION

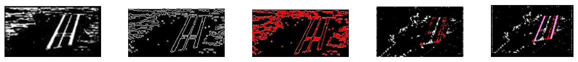
Size	TimeStamp	POS_X	POS_Y	POS_Z
210	168988397609	5.27	-1.945	-9.99
Q_W	Q_X	Q_Y	Q_Z	Condition
-0.02542	0.02811	0.001381	-0.999	rain, fog, noise

3) Vision-based pose estimation algorithm: Figure 10 shows the complete pipeline for runway line feature extraction. Considering the layout space, the pictures, except for Figure 10(a) , only show the region of interest. Among them, Figure 10(a) is the YOLO recognition result, Figure 10(b) is the binarized image, Figure 10(c) is the result of the edge detection and Figure 10(d) is the line detection based on the edge detection image. Figure 10(e) is the corner detection result, the red dots are the points used for RANSAC. Figure 10(f) is the runway line fitting result. The algorithms effectively extract the features of the runway line in the distance.

4) Pose estimation: Based on the extracted feature points with known world coordinates and pixel coordinates, we run the Perspective-n-Point algorithm. We compare the pose estimation and the ground truth in pitch, roll, and yaw in Figure 11. When the aircraft is diving and descending towards the aircraft carrier, the error in the pitch and roll directions is between 5 degrees, but the error in the yaw direction is higher than 10 degrees. At the same time, as time progresses, as the aircraft



(a) region of interest



(b) (c) (d) (e) (f)

Fig. 10. The whole process of runway line feature extraction using the vision-based pose estimation algorithm

approaches the aircraft carrier, the error of pose estimation tends to decrease. Obviously, it can be seen that the trend of the curve is consistent with the ground truth, which indicates that the feature points we extracted are valid. However, our method for pose estimation has better performance in the near space, but the accuracy decreases in the distance. This is because the Perspective-n-Point algorithm is not sufficiently accurate with fewer points, and we do not utilize multisensor fusion to design feedback for the elimination of errors.

D. Vision-based pose estimation algorithm under various conditions

We design two different flight trajectories. The landing points of both trajectories are (0, 0, 24.3). The first trajectory lands directly on the aircraft carrier, and the start recording point is (0,1000, 153), while the second trajectory lands on the side of the aircraft carrier, and the start recording point is (200, 1000, 153). Table V shows the performance of the algorithm under different weather conditions(normal, rain, fog), different interference conditions (normal, random noise, horizontal noise lines), and different trajectories (trajectory I, trajectory II). The criteria P_s, Δ_{error} for evaluating the quality of feature point extraction follow the formula:

$$\begin{cases} P_s = N_s/N \\ \Delta_{error} = \frac{1}{n}\sum_i^n \sqrt{(x_i - x)^2 + (y_i - y)^2} \end{cases} \tag{13}$$

where N_s is the number of datasets from which feature points are successfully extracted, and N is the size of the datasets. Meanwhile, (x_i, y_i) are the pixel coordinates of the extracted feature points, and (x, y) are their corresponding real pixel coordinates. n is the set of extracted feature points. Furthermore, we present a set of representative recognition images in Fig 12 to show the influence of different external conditions on the algorithm.

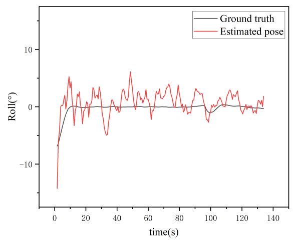
The results in Table V and Fig 12 show that the trajectory, simulated weather, and interference noise all affect the extraction of runway line feature points for the algorithm. From the perspective of weather, in rainy conditions, raindrops block the target, which reduces the success rate of feature point extraction, but the accuracy of point extraction is not affected. However, in foggy weather, the visibility in the environment is reduced, which has a great impact on both target detection and feature point extraction. Next, from the perspective of noise, under random noise conditions, the picture distortion is serious, and the success rate of feature point extraction remains unchanged, but the accuracy is greatly reduced. Compared with random noise, the horizontal noise lines perform better because the noise only exists in the horizontal direction. In addition, due to the threshold setting inside the algorithm, the different trajectories selected will also affect the performance of the algorithm.

TABLE V
PERFORMANCE OF THE ALGORITHM UNDER DIFFERENT CONDITIONS

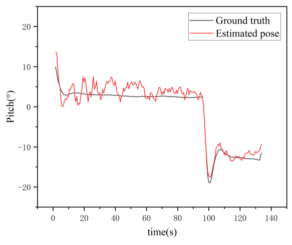
Trajectory	*Weather*	*Interference*	P_s	Δ_{error}	Degree
I(facing)	normal	normal	0.829	5.23	0
I(facing)	normal	**random**	0.814	35.9	3
I(facing)	normal	**horizontal**	0.819	24.2	2
I(facing)	**rain**	normal	0.543	7.93	2
I(facing)	**fog**	normal	0.362	17.7	4
II(oblique)	normal	normal	0.733	12.4	1
II(oblique)	normal	**random**	0.700	44.7	4
II(oblique)	normal	**horizontal**	0.719	39.9	3
II(oblique)	**rain**	normal	0.476	18.6	3
II(oblique)	**fog**	normal	0.333	22.5	5

1 In Trajectory I, the aircraft is facing the aircraft carrier. While, the aircraft in trajectory II is on the oblique side of the aircraft carrier.
2 P_s represents the runway feature points extraction success rate.
3 Δ_{error} represents the feature point average pixel error.
4 normal is the ideal situation with no weather disturbances or noise.
5 Random is Gaussian random noise. Random noise is Gaussian random noise. The pixels in a unit area share the same noise value that satisfies the Gaussian random distribution.
6 Horizontal is the horizontal noise line. The horizontal noise lines increase the noise area on the horizontal line.
7 Degree is a qualitative reference compared to ideal, describing the effect of trajectory, weather, noise, etc. on algorithm performance. Degree is divided into 5 levels from 0 to 5.

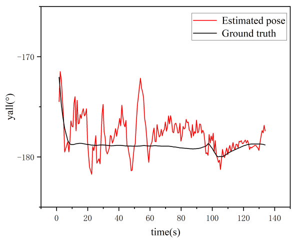
The figure 13 demonstrates the aircraft carrier landing pose estimation results in trajectory II (To facilitate analysis, we apply mean filtering to the pose estimation results). The pose estimation results are compared under different interference scenarios, including normal, foggy, rainy, horizontal noise, and random noise. It is observed that the accuracy of feature point extraction significantly impacts the accuracy of the pose estimation results. Under normal conditions, where the feature point extraction accuracy is high, the pose estimation results are in good agreement with the ground truth value. Conversely, in foggy weather conditions with the lowest extraction accuracy of feature points, the pose estimation results deviate the most. Furthermore,Table VI displays the absolute error of the pose estimation results in each direction. When the accuracy of feature point extraction is low, the results of pose estimation will lose their reference significance. Hence, to improve the accuracy of pose estimation, it is recommended to use more feature points to solve the pose and select a better pose estimation algorithm.



(a) Ground truth *vs* estimated pose in roll

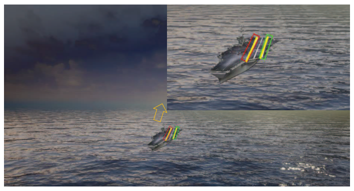


(b) Ground truth *vs* estimated pose in pitch

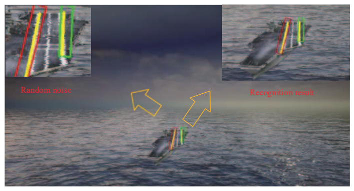


(c) Ground truth *vs* estimated pose in yaw

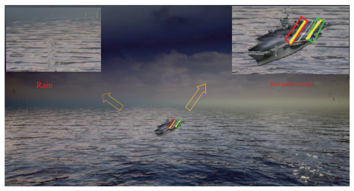
Fig. 11. Comparison of the pose estimation results under three rotation angles



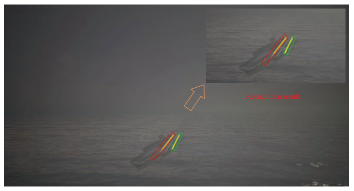
(a) Normal conditions



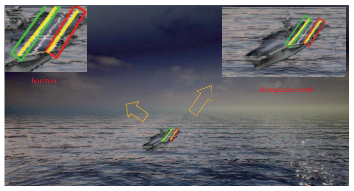
(b) Random noise



(c) Rain



(d) Fog



(e) Horizon noise lines

Fig. 12. The performance of the algorithm under different conditions

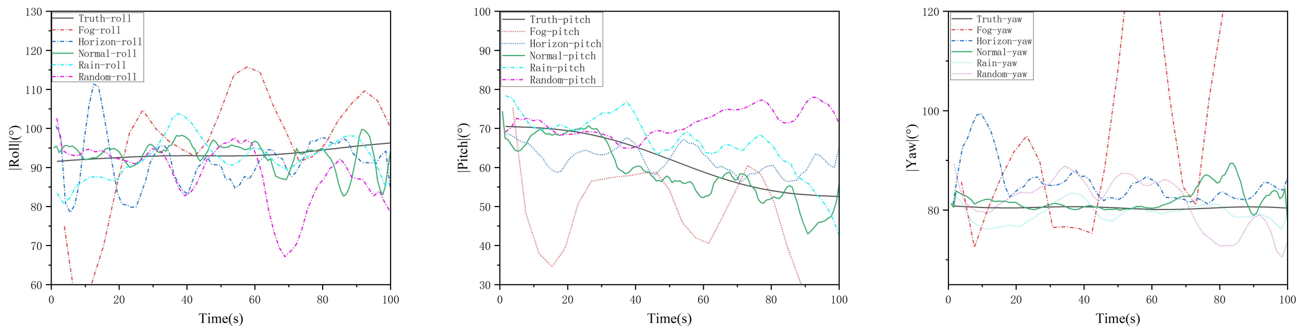


Fig. 13. Pose estimation results for various situations in trajectory II

Thus far, we have completed the design, implementation, and evaluation of the vision-based pose estimation algorithm on the photorealistic simulation platform.

VII. Conclusion and Future Work

In this paper, we develop a photorealistic simulation platform for the autonomous landing of fixed-wing aircraft, which has near-realistic rendering and dynamic models. The platform solves the problems that the aircraft carrier deck landing datasets is difficult to produce and advanced experiments are difficult to carry out. At the same time, by testing algorithms under various conditions, we show that our software-in-the-loop platform can be used for algorithm verification and advanced development. We share demo videos and datasets datasets at https://github.com/U201613306/flight-demonstration.git

In future work, we plan to further enhance the platform by incorporating additional sensor simulations onto the existing infrastructure. Additionally, we intend to design a multi-sensor fusion experiment [28] in order to evaluate the functionality of the simulation module. At present, the simulation of the sea surface on the platform does not take into account the impact

TABLE VI
POSE ESTIMATION ERROR FOR VARIOUS SITUATION

Trajectory	*Weather*	*Interference*	Δin Roll	Δin Pitch	Δin Yaw
II	normal	normal	4.23	6.91	2.64
II	normal	**random**	12.78	17.23	7.33
II	normal	**horizontal**	8.44	9.14	9.61
II	**rain**	normal	4.55	7.44	3.95
II	**fog**	normal	23.11	34.61	45.89

1 This table provides a complementary error analysis of the pose estimation results presented in Figure 13.
2 The dataset used for test comprises 150 frames.
3 Δ denotes the average of the absolute error between the estimated and true attitude values.

of sea conditions. As a result, we aim to simulate typical sea conditions by incorporating noise maps into the simulation.

ACKNOWLEDGMENT

This work is supported by the Equipment Pre-Research Field Foundation under Grant 80913010303.

REFERENCES

[1] William Lee Wellons. *A shipboard global positioning system carrier phase interferometric aircraft flight reference system*. PhD thesis, Ohio University, 1994.

[2] Alison Brown and B Matthews. A robust gps/ins kinematic integrity algorithm for aircraft landing. In *Proceedings of the 19th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2006)*, pages 715–725, 2006.

[3] Dengqing Tang, Tianjiang Hu, Lincheng Shen, Daibing Zhang, Weiwei Kong, and Kin Huat Low. Ground stereo vision-based navigation for autonomous take-off and landing of uavs: a chan-vese model approach. *International Journal of Advanced Robotic Systems*, 13(2):67, 2016.

[4] Long Xin, Zimu Tang, Weiqi Gai, and Haobo Liu. Vision-based autonomous landing for the uav: A review. *Aerospace*, 9(11):634, 2022.

[5] Tyler B McCarthy. Feasibility study of a vision-based landing system for unmanned fixed-wing aircraft. Technical report, Naval Postgraduate School Monterey United States, 2017.

[6] Zhouyu Zhang, Yunfeng Cao, Meng Ding, Likui Zhuang, and Jiang Tao. Vision-based guidance for fixed-wing unmanned aerial vehicle autonomous carrier landing. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 233(8):2894–2913, 2019.

[7] XiaoBin Xu, Zhao Wang, and YiMin Deng. A software platform for vision-based uav autonomous landing guidance based on markers estimation. *Science China Technological Sciences*, 62:1825–1836, 2019.

[8] Zhuo Zhang, Qing Shi, Hui Huang, and Xiaoliang Sun. Application of centernet technology in the process of shipborne aircraft autonomous landing. In *2022 5th International Symposium on Autonomous Systems (ISAS)*, pages 1–6. IEEE, 2022.

[9] Yifan Zhu, Ling Pei, Danping Zou, Wenxian Yu, Tao Li, Qi Wu, and Songpengcheng Xia. A geometric consistency model of virtual camera for vision-based slam simulation. In *International Conference on Spatial Data and Intelligence*, pages 271–280. Springer, 2020.

[10] Xinglong Yang, Danping Zou, Ling Pei, Daniele Sartori, and Wenxian Yu. An efficient simulation platform for testing and validating autonomous navigation algorithms for multi-rotor uavs based on unreal engine. In *China Satellite Navigation Conference*, pages 527–539. Springer, 2019.

[11] Rongzhi Wang, Danping Zou, Changqing Xu, Ling Pei, Peilin Liu, and Wenxian Yu. An aerodynamic model-aided state estimator for multirotor uavs. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2164–2170. IEEE, 2017.

[12] Gary Ellingson and Tim McLain. Rosplane: Fixed-wing autopilot for education and research. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1503–1507. IEEE, 2017.

[13] Kun Xiao, Shaochang Tan, Guohui Wang, Xueyan An, Xiang Wang, and Xiangke Wang. Xtdrone: A customizable multi-rotor uavs simulation platform. In *2020 4th International Conference on Robotics and Automation Sciences (ICRAS)*, pages 55–61. IEEE, 2020.

[14] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[15] Todd Ringler, Mark Petersen, Robert L Higdon, Doug Jacobsen, Philip W Jones, and Mathew Maltrud. A multi-resolution approach to global ocean modeling. *Ocean Modelling*, 69:211–232, 2013.

[16] Xudong Yang, Xuexian Pi, Liang Zeng, and Sikun Li. Gpu-based realtime simulation and rendering of unbounded ocean surface. In *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, pages 6–pp. IEEE, 2005.

[17] Hajar Righi. *Hybrid electric aircraft*. Mississippi State University, 2016.

[18] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[19] Kyung-Duck Suh, Hyuk-Dong Kwon, and Dong-Young Lee. Some statistical characteristics of large deepwater waves around the korean peninsula. *Coastal Engineering*, 57(4):375–384, 2010.

[20] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[21] Radu Horaud, Bernard Conio, Olivier Leboulleux, and Bernard Lacolle. An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics, and Image Processing*, 47(1):33–44, 1989.

[22] Lijun Ding and Ardeshir Goshtasby. On the canny edge detector. *Pattern recognition*, 34(3):721–725, 2001.

[23] Nahum Kiryati, Yuval Eldar, and Alfred M Bruckstein. A probabilistic hough transform. *Pattern recognition*, 24(4):303–316, 1991.

[24] Zhiyong Ye, Yijian Pei, and Jihong Shi. An improved algorithm for harris corner detection. In *2009 2nd International Congress on Image and Signal Processing*, pages 1–4. IEEE, 2009.

[25] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4(3):1, 2013.

[26] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[27] Hwankee Cho. A study on the configuration modeling and aerodynamic analysis of small airplanes for flight training. *Journal of the Korean Society for Aviation and Aeronautics*, 28(1):59–65, 2020.

[28] Xin Chen, Di He, and Ling Pei. Bds b1i multipath channel statistical model comparison between static and dynamic scenarios in dense urban canyon environment. *Satellite Navigation*, 1(1):1–16, 2020.