

Article

A Framework for Autonomous UAV Navigation Based on Monocular Depth Estimation

Jonas Gaigalas *, Linas Perkauskas, Henrikas Gričius, Tomas Kanapickas  and Andrius Kriščiūnas 

Department of Applied Informatics, Faculty of Informatics, Kaunas University of Technology, Studentu St. 50, LT-51368 Kaunas, Lithuania

* Correspondence: jonas.gaigalas@ktu.edu

Abstract: UAVs are vastly used in practical applications such as reconnaissance and search and rescue or other missions which typically require experienced operators. Autonomous drone navigation could aid in situations where the environment is unknown, GPS or radio signals are unavailable, and there are no existing 3D models to preplan a trajectory. Traditional navigation methods employ multiple sensors: LiDAR, sonar, inertial measurement units (IMUs), and cameras. This increases the weight and cost of such drones. This work focuses on autonomous drone navigation from point A to point B using visual information obtained from a monocular camera in a simulator. The solution utilizes a depth image estimation model to create an occupancy grid map of the surrounding area and uses an A* path planning algorithm to find optimal paths to end goals while navigating around the obstacles. The simulation is conducted using AirSim in Unreal Engine. With this work, we propose a framework and scenarios in three different open-source virtual environments, varying in complexity, to test and compare autonomous UAV navigation methods based on vision. In this study, fine-tuned models using synthetic RGB and depth image data were used for each environment, demonstrating a noticeable improvement in depth estimation accuracy, with reductions in Mean Absolute Percentage Error (MAPE) from 120.45% to 33.41% in AirSimNH, from 70.09% to 8.04% in Blocks, and from 121.94% to 32.86% in MSBuild2018. While the proposed UAV autonomous navigation framework utilizing depth images directly from AirSim achieves 38.89%, 87.78%, and 13.33% success rates of reaching goals in AirSimNH, Blocks, and MSBuild2018 environments, respectively, the method with pre-trained depth estimation models fails to reach any end points of the scenarios. The fine-tuned depth estimation models enhance performance, increasing the number of reached goals by 3.33% for AirSimNH and 72.22% for Blocks. These findings highlight the benefits of adapting vision-based models to specific environments, improving UAV autonomy in visually guided navigation tasks.



Academic Editor: Ziyang Zhen

Received: 5 February 2025

Revised: 7 March 2025

Accepted: 21 March 2025

Published: 23 March 2025

Citation: Gaigalas, J.; Perkauskas, L.; Gričius, H.; Kanapickas, T.; Kriščiūnas, A. A Framework for Autonomous UAV Navigation Based on Monocular Depth Estimation. *Drones* **2025**, *9*, 236. <https://doi.org/10.3390/drones9040236>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: UAV; control; vision-based navigation; path planning; collision avoidance; simulation; deep learning; monocular depth estimation

1. Introduction

Unmanned aerial vehicles (UAVs) have become an active object of recent research. The quad-rotor drones are especially appealing due to their lightweight frame, sensor modularity, and availability of other cost-effective off-the-shelf parts. Essential topics such as search and rescue, remote sensing, precision agriculture, goods delivery, and other civil applications have seen innovation utilizing UAVs [1]. However, the human operation of such vehicles can become a problem in some scenarios, for example, where the GPS and

radio signals are unavailable, and it is impossible to preplan a trajectory due to non-existing 3D maps or landmark references. Autonomous working drones could be used to perform such missions. This way, the signal interference would not be a threat, as all the computing and control processes would be internal.

The autonomous navigation of a vehicle requires robot perception and localization, motion planning, and the execution of commands with controllers. Usually, for this complicated system to be robust, full-navigation solutions employ the fusion of multiple sensors. The most common of these are light detection and ranging (LiDAR) scanners, multiple cameras, and acoustic or radar sensors accompanied by inertial measurement units (IMUs) [2]. These solutions can be quite expensive in terms of both the price and computational cost. Reliable LiDAR scanners typically range in price from a few thousand to tens of thousands of dollars. In contrast, simple RGB cameras and the necessary system hardware offer a more affordable alternative. Developing a system that achieves results comparable or superior to LiDAR while being cost-effective would provide significant advantages in various systems.

A solution for these problems is to use a monocular camera as the primary sensor for the perception and localization of the drone. RGB cameras are relatively cheap and lightweight, bringing the cost of such a system down. Also, the captured images can provide more semantic information to aid the operations. However, using a simple RGB camera comes with its challenges. The algorithms involved in visual localization are computationally demanding and prone to problems arising from motion blur, variation in illumination, and inherent scale drift [3]. Many researchers opt into RGB-D cameras for their setups to gather additional depth information to counter some of these problems. However, these sensors are relatively expensive in comparison to simple RGB cameras.

In recent years, research has brought attention to various depth estimation methods [4]. By utilizing machine learning models, it is now possible to achieve accurate depth images from a single RGB image. This enables the use of algorithms traditionally only available in setups containing RGB-D cameras, thus expanding the capabilities of monocular RGB camera drone systems, albeit with increased computational cost.

With the rise in UAVs (and robotics in general), new tools for the simulation of such systems have seen the light as software-in-the-loop (SiL) testing is an integral part of building a robust autonomous navigation solution. However, autonomous UAV simulation research mainly consists of indoor environments where robots navigate corridors, areas of sparsely distributed box-shaped objects or smaller, and specific segments of areas [5–7]. Also, most researchers only use a few select tools in their papers, and, most of the time, do not provide the environments or the exact coordinates for the start and end points [8]. All of this does not allow for reasonable comparison between the research or real-world conditions.

This study aims to develop an autonomous navigation system for UAVs that relies on visual information acquired with a monocular RGB camera in simulated environments. By implementing a depth estimation model, an occupancy grid map is generated from point clouds. This allows the drone to dynamically plan a path for obstacle avoidance during the navigation from point A to point B. In this study, the results of simulated flights in different scenarios utilizing estimated depth images are compared with results using depth images from the simulation software AirSim [9]. Different voxel sizes are compared to check their effect on navigation performance during flight. In addition, each flight scenario's exact environments and start/end points are provided for future comparisons between autonomous navigation solutions in AirSim platform.

2. Related Work

Building a UAV navigation system that fully functions without human input is no easy feat, as many components must work in coordination. Therefore, only about 16% of works in the field of UAV navigation—particularly in scenarios without GNSS (Global Navigation Satellite System) availability—have proposed complete navigation solutions, while the rest of the research focused on individual parts of UAV navigation like localization, estimation of various inertial parameters of flight, mapping of surroundings, and path planning [2].

2.1. Localization

Localization is a fundamental aspect of UAV navigation, providing the necessary information by estimating a vehicle's position and orientation in its operating environment. It is essential in other robotics applications and is also the most researched navigation area [2].

Autonomous navigation solutions utilizing vision sensors can be broken down into two main categories: mapless and map-based [8,10,11]. In mapless systems, UAVs do not need prior knowledge of the area they fly in, and no maps are built during the navigation process. Solutions can be categorized as map-based when UAVs are either provided with a map of the area they need to navigate or they simultaneously build one whilst exploring the unknown territory.

Mapless systems rely on real-time sensor data to estimate the motion and position of the vehicle without prior knowledge of the environment. Such vision-based systems extract visible features from the environment to make navigation decisions. The most prominent methods in mapless solutions are optical flow (OF) and feature tracking [11]. In recent years, researchers have also shown systems that use various machine-learning approaches [8].

The optical flow technique compares pixels between consecutive frames and estimates the motion between the camera and the scene. Two main categories of optical flow include global [12] and local [13]. In the case of global OF, the neighboring pixels of a point in the image and the smoothness of their movements are considered. In the local approach, all the pixels in the image are compared, and their differences are measured. The earliest application of optical flow to UAV navigation was based on a stereo vision setup imitating a bee's flight, where the UAV estimates the distance to walls and obstacles from two cameras and traverses through a corridor accordingly [14]. Throughout the years, navigation methods that utilize optical flow have improved. For example, researchers showed that using inverse optical flow techniques with a single forward-facing camera can overcome the previously existing limitations of balance strategy when UAVs face an obstacle directly in front [15]. This enables vehicles to navigate in urban canyon settings.

The feature tracking method extracts points or patterns from images visible in the environment and follows them over time to estimate their motion relative to the camera. This technique can track invariant features such as corners, lines, or other points that could be detected in different images of the same area under changed conditions like rotation, illumination, and scale [16]. Well-known feature point detectors and descriptor extraction algorithms include Harris corners [17], features from accelerated segment test (FAST) [18], scale-invariant feature transform (SIFT) [19], binary robust independent elementary features (BRIEF) [20], speeded up robust features (SURF) [21], and Oriented FAST and Rotated BRIEF (ORB) [22]. In recent years, feature descriptors have become more invariant to illumination and scale changes and more robust than OF methods. However, due to the computational overhead that feature detection and matching brings, the feature-based systems compute sparser environment representation in points. Furthermore, in texture-less (feature-poor) environments, tracking performance can be poor or even lost [3].

Map-based systems rely on sensor data and prior knowledge about the environment, such as known or partially known maps or spaces. It also includes map-building strategies that can reconstruct the map of the area in which drones navigate dynamically [8].

Prior knowledge of surroundings can be specified as landmarks such as waypoints made up of roof positions that can be matched from satellite images during navigation or specialized patterns such as QR or other ground patterns that can be detected with the help of machine learning and feature-based algorithms [8]. These landmarks can serve as access points to guide the drone, show where obstacles are located, or help track targets. Moreover, the knowledge can come in the form of maps of the environment. Two main types of maps are used in UAV navigation: octree and occupancy grid maps [11]. Volumetric 3D environment models can be generated utilizing sensor data and represented as hierarchical tree structures based on octrees. An efficient open-source framework used by many robotic systems is OctoMap [23]. The occupancy grid method represents the environment as a grid of cells where each cell is likely occupied. An example of such a mapping technique is shown in [24]. Information about obstacles and free space is stored in multi-volume occupancy grids, which allows them to incrementally fuse new positive or negative sensor information, enabling correction for errors and a more robust 3D mapping of the environment.

These methods can also be used in a similar map-building approach, which dominates the full-navigation solutions and is called simultaneous localization and mapping (SLAM) [2]. This technique allows UAVs to generate a map and simultaneously estimate their position while navigating an unknown environment [25]. Methods like feature-based ORB-SLAM [26], MonoSLAM [27], or direct LSD-SLAM [28] can be utilized in monocular vision-based setups.

2.2. Monocular Depth Estimation

Monocular depth estimation is the process of assessing depth from singular RGB images and inferring depth information. Researchers try to differentiate the types of depth estimation methods into two categories: single-shot and multi-frame [4].

Single-shot depth estimation drops the frames and estimates the depth from any taken image without a broader context about the environment. These methods often use deep learning models trained on large datasets to extract depth cues from visual patterns such as texture, shading, and perspective. Depth estimation models like Depth Anything [29] provide a way to generalize depth predictions for small and large details by incorporating multi-scale attention mechanisms. Another prominent approach, Monodepth2 [30], achieves accurate depth predictions by combining self-supervised learning with encoder–decoder architectures, allowing it to adapt to various domains and datasets without requiring extensive ground truth depth annotations.

Multi-frame depth estimation leverages temporal information from consecutive frames to infer depth, exploiting motion cues such as parallax and changes in camera position. These methods often integrate principles from structure-from-motion (SfM) or visual-inertial odometry (VIO). By analyzing object shifts across frames, multi-frame methods achieve higher depth accuracy and reconstruct 3D structures more effectively. Recent advancements, such as the Multi-Frame Depth Transformer (MFDT), demonstrate the effectiveness of transformer-based architectures and how depth estimation can be improved by capturing temporal relationships in scene geometry and motion dynamics [31].

2.3. Path Planning

Path planning is a major navigation system component that aims to swiftly and accurately navigate a sufficiently complex environment. One type of path-planning algorithm

that can be used for online UAV navigation is called graph-based path planning. These types of algorithms work by representing the environment as a graph. In such a graph, nodes represent discrete navigable locations in the environment, and edges represent possible paths between these locations. Graph-based search algorithms aim to find the optimal path from the start to the goal node on a pre-defined graph. These methods are highly effective in structured environments with known maps and can efficiently handle static obstacles. Nevertheless, these algorithms can still be used for online path planning. However, such use introduces the need for recurrent replanning. Examples of viable graph-based search algorithms are Dijkstra's Algorithm [32], A* [33], Theta* and Lazy Theta* [34], and Jump Point Search (JPS) [35]. Other graph-based search algorithms are designed to work in dynamic/unknown environments, e.g., D* [36] or D* lite [37].

Another widely used path planning algorithm type is sampling-based algorithms. Sampling approaches create a graph representation of the environment by sampling points and connecting them to form a possible path. These methods are especially useful in unknown or partially known, high-dimensionality environments as these algorithms focus on finding any valid solution instead of the globally optimal one. Examples of such algorithms are Rapidly Exploring Random Trees (RRT) [38], Probabilistic Roadmap (PRM) [39], and Voronoi Diagrams [40].

2.4. Autonomous UAV Navigation

Complete solutions for autonomous UAV navigation combine all the parts into one system. Usually, such systems employ numerous different sensors for more robust identification and mapping of the environment to avoid obstacles in the UAV's path [2].

LiDAR scans environments with laser rangefinders to provide accurate distance and position measurements for single points. These sensors are usually used as complimentary devices to provide depth information as they cannot extract texture or color. Techniques like 2D or 3D LiDAR SLAM are utilized in such solutions [5,6,41–44].

RGB-D can also provide depth-sensing capabilities. Devices like Microsoft Kinect, Intel RealSense, or Asus Xtion retrieve red, green, and blue alongside depth data for each pixel. This is usually performed by utilizing infrared projectors and receivers. The depth information combined with SLAM techniques can result in highly detailed 3D maps [7,45–47]. Stereo camera setups using multiple cameras can also generate depth information from the disparity between captured images [48–50]. Numerous approaches also make use of monocular cameras [25,51–53].

3. Materials and Methods

3.1. Simulation Environment

This study uses the AirSim simulator to conduct autonomous drone flights. There are ten default environments built with Unreal Engine and provided by AirSim. Each environment is based on an outdoor setting with varying numbers of objects and complexity. The simulator can extract depth, RGB, and segmentation images from the drone camera's point of view. The AbandonedPark environment has a moving Ferris wheel, the CityEnviron has moving cars and pedestrians implemented, and TrapCam has simulated moving deer and vegetation generation, which can be tuned with different parameters. For experimentation, three of the available environments (AirSimNH, Blocks, and MSBuild2018) were chosen. Overview screenshots from the simulator in these environments can be seen in Figure 1.

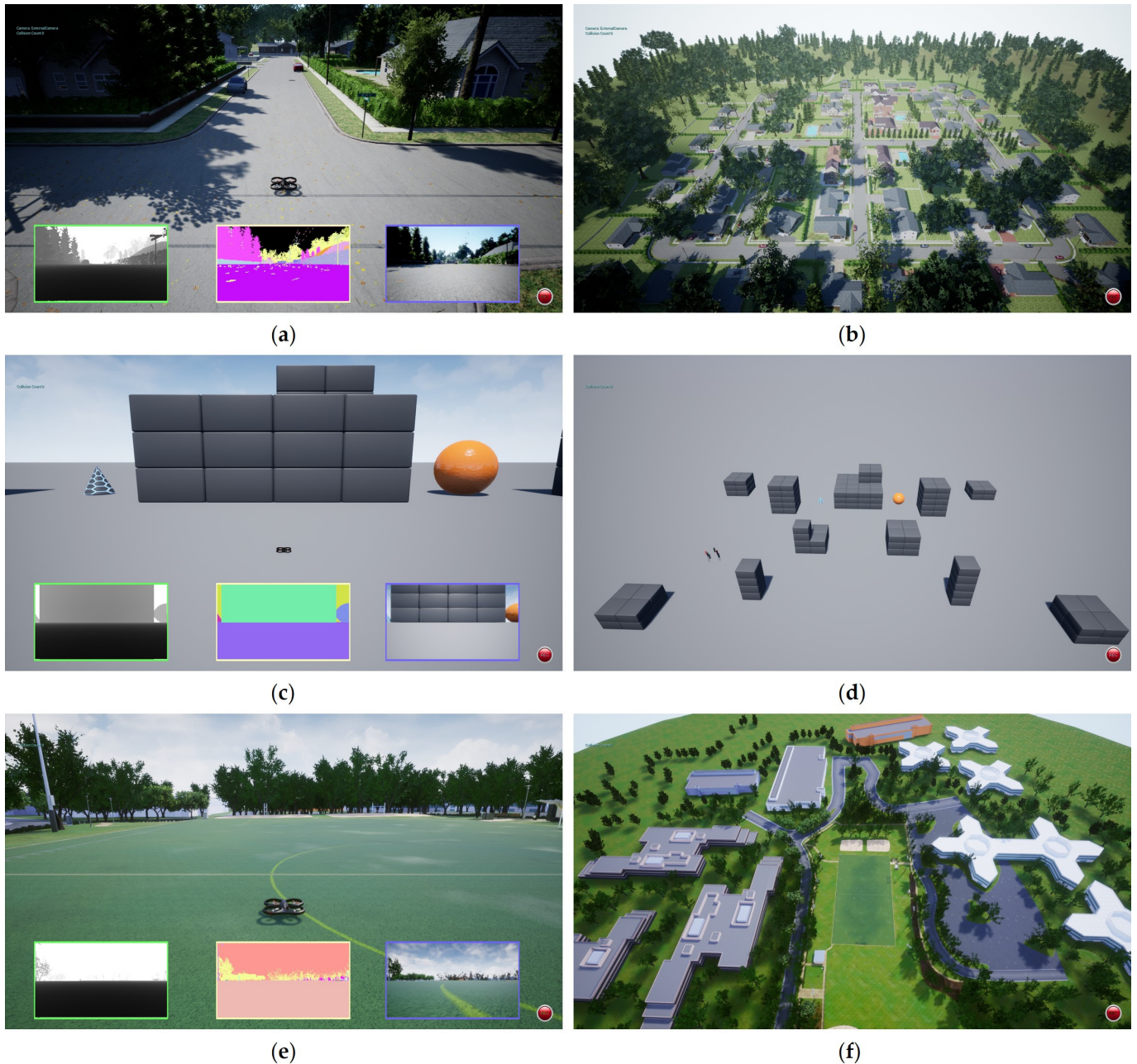


Figure 1. Snapshots from different AirSim environments that show depth, segmentation, and RGB images from the drone’s front-facing camera and the environment from higher up: AirSimNH (a,b); Blocks (c,d); and MSBuild2018 (e,f).

The drone was setup to acquire 518×518 pixel-sized RGB and depth images with a 60-degree field of view (FOV). During navigation, the drone keeps distance from obstacles in a 3×3 voxel grid, as shown as a blue grid in Figure 2. The size of the voxel V can be changed, which in turn scales this grid, allowing the drone to traverse in either narrower or broader spaces. Other drone settings, such as flying speed or other flight parameters, can be configured in the simulator; however, simulations in this study scope were conducted to explore the effects of varying voxel size; therefore, the other parameters were set to default values such as flying velocity of 2 m/s, an occupancy map size of $1800 \times 1800 \times 100$ voxels, a maximum of 200 iterations to find paths, and a maximum of 20 consecutive steps with collisions registered to quit the simulation.

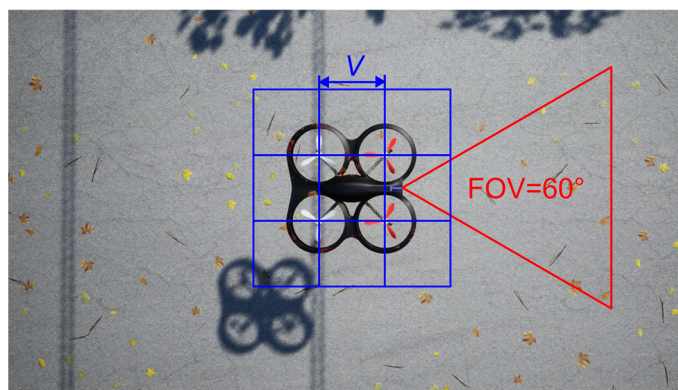


Figure 2. Snapshot of the drone inside the AirSim environment. The blue grid shows the scalable boundary of the drone in voxels of size V . The red area depicts the drone's field of view.

All the simulations were performed on a personal computer. The environments are computationally heavy on graphics; therefore, a dedicated graphics card (GPU) had to be installed in the computer. All the system's specifications are noted in Table 1.

Table 1. Specifications of the personal computer used to run flight simulations in this work.

GPU	CPU	RAM
EVGA GeForce RTX 3070 Ti; 8 GB GDDR6X @ 1575 MHz	AMD Ryzen 7 5800X; 8 cores @ 3.8 GHz	Kingston Fury; 2×32 GB @ 3200 MHz

Three scenarios were chosen for each environment to test the autonomous drone navigation system. Start and end coordinates alongside the Euclidean distance of each proposed scenario are in Table 2. Here, the last column notes where each scenario is depicted in Figure 3 where yellow circles and triangles depict the start and end positions of different scenarios in each environment, respectively. Red lines show the paths that a drone could hypothetically take to navigate through each scenario.

Table 2. Start and end coordinates of different scenarios in chosen environments.

Environment	Scenario	Start (X, Y, Z)	End, (X, Y, Z)	Distance	Snapshot
AirSimNH	1	0, 0, 0	−130, −115, 3	173.59	a
	2	0, 0, 0	130, 130, 3	183.87	b
	3	0, 0, 0	−115, 130, 3	173.59	c
Blocks	1	0, 0, 0	70, 150, 3	165.56	d
	2	0, 0, 0	70, −135, 3	152.10	e
	3	0, 0, 0	−105, −150, 3	183.12	f
MSBuilding2018	1	0, 0, 0	30, 260, 3	261.74	g
	2	0, 0, 0	30, −280, 3	281.62	h
	3	0, 0, 0	−400, 30, 3	401.14	i

As the AirSimNH environment (Figure 3a–c) has houses and their backyards separated by roads, all three proposed flight paths start at the intersection of these roads. The end points are located at the corners of the neighborhood. These paths force the drone to navigate complex environments containing houses, trees, fences, and other objects typically found in such backyards. It is worth noting that deer and raccoons are also moving in this environment. Also, along the outskirts of the environment, there is a densely populated forest, which is well suited for obstacle avoidance experiments.

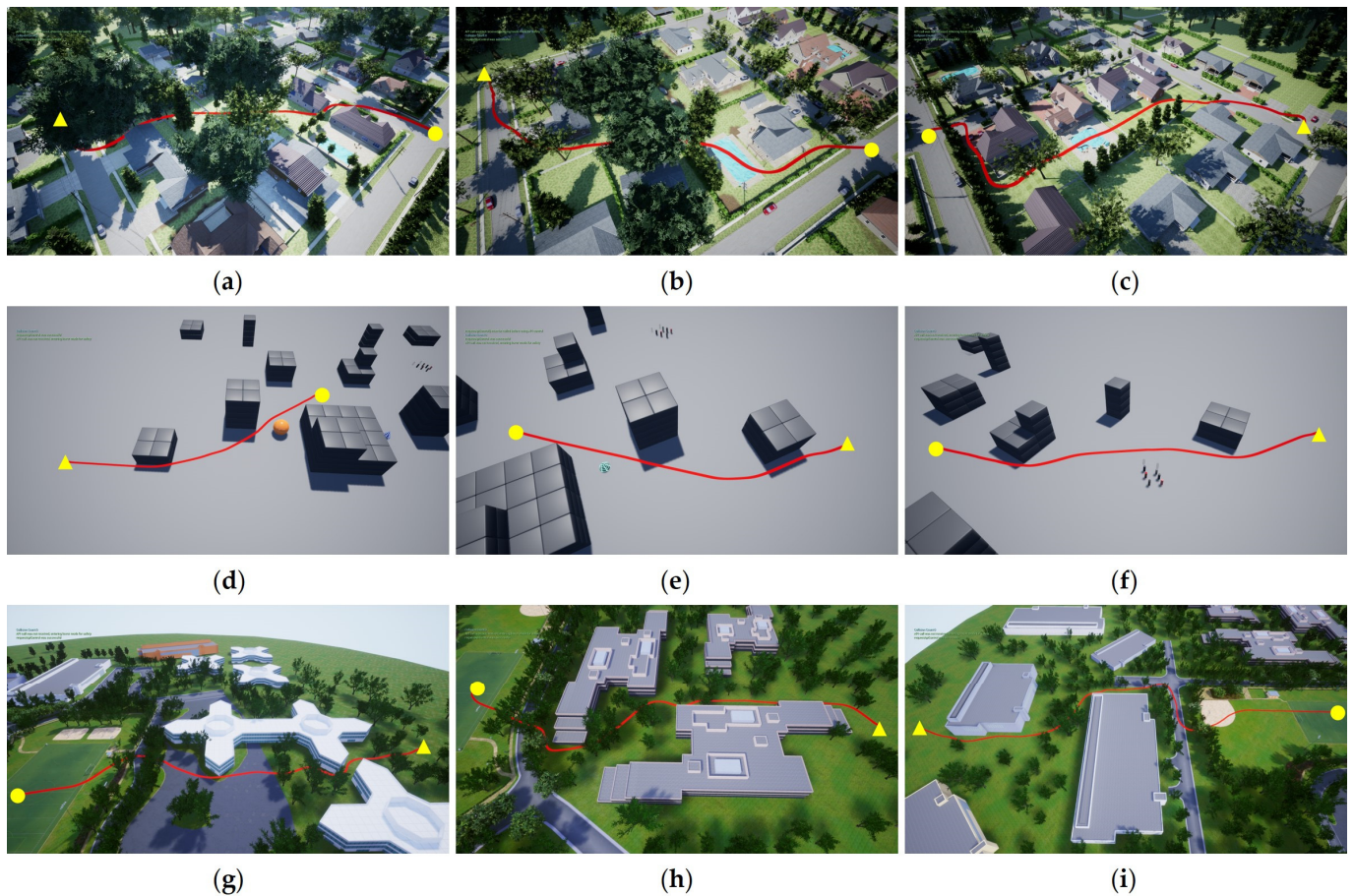


Figure 3. Snapshots of different scenarios in different environments. Yellow circles show start positions, triangles—set goal coordinates. Red lines depict hypothetical paths for simulated flights. Scenarios no.1–3 in different AirSim environments: (a–c) AirSimNH; (d–f) Blocks; and (g–i) MSBuild2018.

For tests in less complex settings, the Blocks environment (Figure 3d–f) was chosen. Relatively big-sized blocks are scattered sparsely in this environment alongside some other basic shapes like balls, cones, or cylinders. The end point coordinates are located at the corners of the map. To reach denoted positions, the drone would need to evade simple-shaped boxes and other objects.

MSBuild2018 is one of the AirSim provided environments (Figure 3g–i) that is composed of an open field, roads, and building complexes densely surrounded by trees and other objects. In proposed scenarios, the drone starts in the open field. To reach the goals, the drone must navigate through thick vegetation along the roads and between the buildings, making the task more complex.

3.2. System Architecture

The proposed architecture for an autonomous monocular drone navigation system is depicted in Figure 4. The system is divided into a simulation environment and a drone navigation system called a “Monocular drone”. For the simulation environment, Airsim v1.8.1 was used. The navigation system runs in a containerized Docker environment running Ubuntu 20.04 guest. The ROS noetic v1.17.0 framework was used for communication between these two parts.

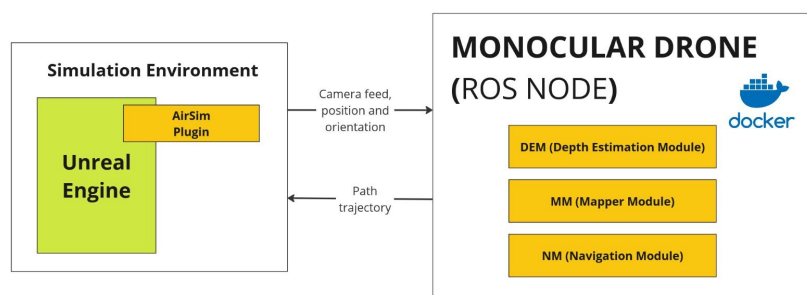


Figure 4. Diagram of the autonomous monocular drone navigation system.

The navigation system was further divided into three modules:

- Depth estimation module (DEM). This module is responsible for estimating depth images from the RGB camera feed provided by the simulation environment. For this task, the Depth Anything V2 model was utilized.
- Mapper module (MM). The purpose of MM is to build and iteratively update the occupancy map-based 3D environment using depth images supplied by DEM and camera position and orientation retrieved from the simulation environment.
- Navigation module (NM). This module finds viable path trajectories to specified points in an everchanging mapped 3D environment. Path-finding logic is based on the A* algorithm. The output of this module is fed back to the simulation environment.

3.3. Depth Estimation

3.3.1. Preparing Synthetic Data for Depth Estimation Models

The inherent scale-drift problem occurred using the pretrained single-shot monocular depth estimation model. However, a hypothesis was proposed that the errors in depth estimation come from insufficient synthetic images and that the results could be improved by finetuning the model.

The models were finetuned with the simulation environment data to obtain more accurate depth images from RGB images. A dataset was generated for each environment to improve the depth estimation accuracy, as the pre-trained model struggled to adjust to environment lighting conditions, textures, objects, and other scene features. A script was developed to generate datasets to fine-tune the model. The script in Algorithm 1 used parameters such as the following:

- Number of coordinates that define how many points will be generated;
- Map boundaries specifying the range in which the drone can be placed;
- Height list specifying predefined height (1, 3, 5, 7, and 9 m) for generated coordinates along the z-axis;
- Camera IDs, representing cameras used to capture depth and RGB images. Cameras were positioned at 0, 45, 90, 135, 180, 225, 270, and 315 degrees, allowing images to be captured from all sides for a specific coordinate.

For each different height, a set of coordinates was generated, and at these coordinates, all cameras captured depth and RGB images, ensuring that various viewpoints and heights would be captured. The visualization of all the angles of cameras and examples of acquired images are depicted in Figure 5.

Algorithm 1. Data generation script

Input: Number of coordinates num_coord , map boundaries $x_start, x_end, y_start, y_end$, Height list $z_heights$,

Constants: Camera id list $camera_ids$

Output: Saved images.

```

1:   Generate random coordinates:
2:   Initialize coordinate list  $coord\_list$ 
3:   For  $z$  to  $z\_heights$ :
4:     For  $i$  to  $num\_coord$ :
5:       add the generated coordinate to coordinate list
 $coord\_list.add(random(x\_start, x\_end), random(y\_start, y\_end), z)$ 
6:     End For
7:   End For
8:   Start and stabilize the drone in environment
9:   For  $coordinate$  in  $coord\_list$ :
10:    Spawn and hover drone in  $coordinate$ 
11:    For a  $camera$  in  $camera\_ids$ :
12:      Save the Depth and RGB image of the  $camera$ 
13:    End For
14: End For

```

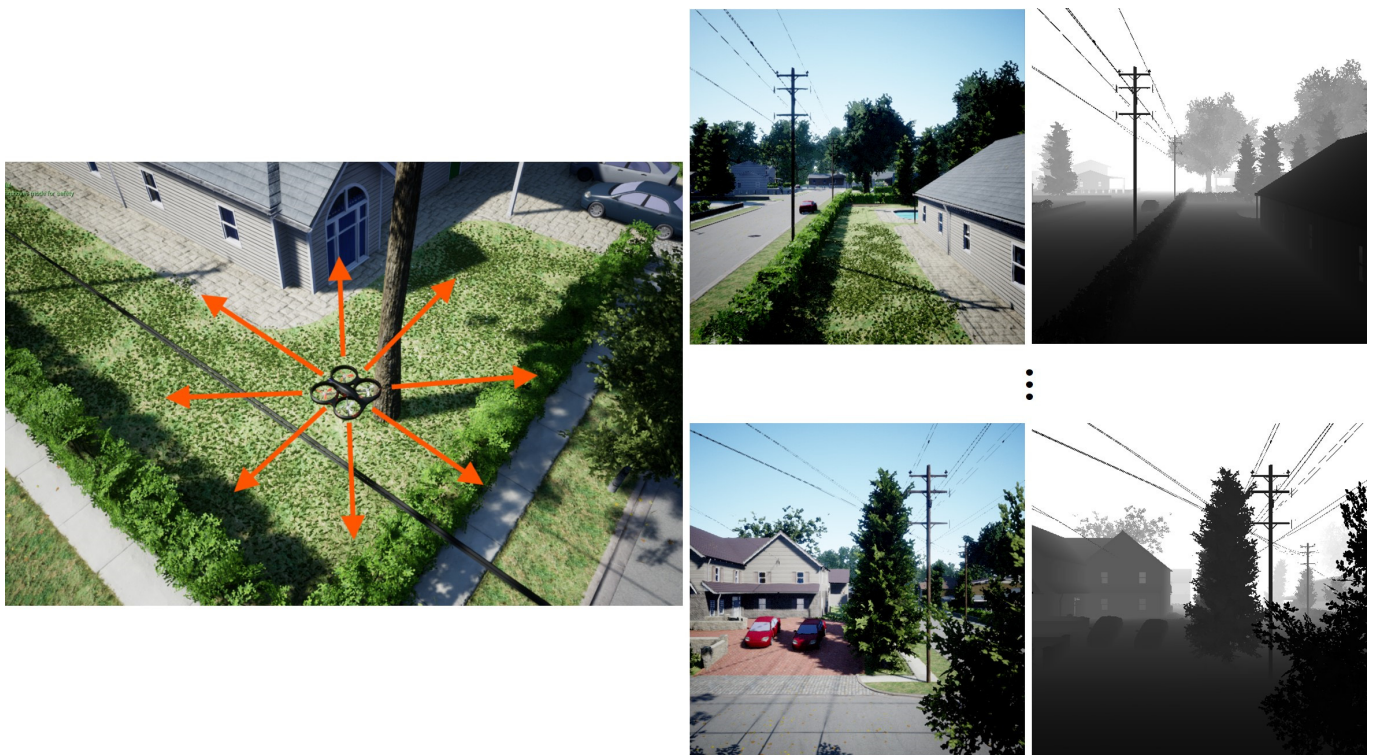


Figure 5. Generation of RGB and depth images from AirSim for finetuning dataset in AirSimNH environment. On the (right), a snapshot of a drone setup in randomly generated coordinates, where red arrows depict the angles at which the cameras are pointing. On the (left) are examples of acquired pairs of RGB and depth images from front and back-facing cameras.

3.3.2. Process for Fine-Tuning Depth Estimation Model

Specialized depth estimation models were fine-tuned for each environment. Training images had a resolution of 518×518 pixels due to the chosen model's architectural design.

The prepared dataset was split into training and validation sets (approximately 80% for training and 20% for validation). Training images included absolute depth annotations allowing the model to infer statistically accurate metric depths without requiring an explicit scale correction factor. Unlike event cameras, which rely on dynamic scene changes to estimate depth, our method infers depth purely from static image inputs, making it more suitable for structured scene analysis. The chosen architecture is the Depth Anything V2 [29] model with a Vision Transformer (ViT-S) encoder. The model using the (ViT-S) encoder is the smallest but one of the fastest and least resource-intensive among released models. The fine-tuning processes involved adapting the network to the specific domain. The dataset was preprocessed by normalizing pixel values and adjusting them to fit the model architecture. The training was performed using the BerHu loss function and AdamW optimizer with a learning rate scheduler which helped reduce the learning rate reduction over time and improve convergence.

Additionally, models were trained without references to objects in a simulated environment such as object coordinates, object classes, or object mapping. The fine-tuned model only relies on the RGB, and the depth images extracted from the AirSim simulation environment, which means that the model only learns from the visuals of the environment.

The BerHu loss function has been chosen to evaluate performance because it combines $L1$ and $L2$ terms. The loss function behaves similarly to the standard $L1$ measure for minor prediction errors. It is more robust to outliers due to its constant gradient and avoidance of massive updates. As the mistakes grow, they transition smoothly to the $L2$ penalty that prevents gradients from exploding while assigning proportionately higher penalties to substantial discrepancies. Such a loss definition is particularly effective in handling large-depth errors while remaining sensitive to more minor errors. This approach is motivated by prior work on depth estimation, where the BerHu loss has been shown to improve the robustness of deep learning models [54].

$$l_i = \begin{cases} \delta_i, & \text{if } \delta_i \leq \tau, \\ \frac{\delta_i^2 + \tau^1}{2\tau}, & \text{if } \delta_i > \tau. \end{cases} \quad (1)$$

The overall loss is then the average over all the pixels:

$$L_{BerHu}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N l_i \quad (2)$$

This can be expressed mathematically as a piecewise function where y is the ground-truth depth, $\hat{y}$ is the predicted depth, τ is the threshold (in this work, the threshold $\tau = 0.2$), and N is the total number of pixels in the batch. Define $\delta_i = |y_i - \hat{y}_i|$.

3.4. Occupancy Grid Map

A probability-based occupancy map was used for environment mapping. The mapping process consists of iteration, where each iteration is divided into three stages: in the first stage, the mapping subsystem takes $W \times H$ sized normalized depth images *dimg* received from the depth estimation model as input. Here, *dimg* is a single-channel image, where pixel intensity is directly linearly proportional to the distance from the point of observation; the input image is then downsampled by filtering pixels with a horizontal and vertical stride of 2, i.e. each second pixel in each second row is used; the resulting image is then converted to a point cloud by applying Algorithm 2.

Algorithm 2. Depth Image to Point Cloud Conversion

Input: Depth image $dimg$, Field of view fov , Image dimensions (H, W) , Stride $stride$, Minimum depth $dmin$, Maximum depth $dmax$.

Output: Point cloud pcd in ENU coordinate system.

```

1: Initialize empty point cloud  $pcd$ 
2: Compute horizontal FOV in radians  $hfov_{rad} = fov \cdot \pi/180.0$ 
3: Calculate focal lengths  $fx = fy = (W/2.0)/\tan(hfov_{rad}/2.0)$ 
4: Set principal point coordinates  $cx = W/2.0, cy = H/2.0$ 
5: For  $v = 0$  to  $H$  step  $stride$ :
6:   For  $u = 0$  to  $W$  step  $stride$ :
7:     Retrieve depth value  $z = dimg[v,u]$ 
8:     If  $dmin \leq z < dmax$  then:
9:       Compute  $x = (u - cx) \cdot z/fx$ 
10:      Compute  $y = (v - cy) \cdot z/fy$ 
11:      Rotate and add point  $[z, -x, -y]$  to  $pcd$  ENU point
12:    End If
13:  End For
14: End For
15: Return point cloud  $pcd$ 

```

This conversion discards points that are too close or too far away, then rotates 90 degrees along the pitch axis and converts them to use the ENU (East–North–Up) world reference system. This results in a point cloud from the drone’s perspective. This point cloud has to be converted to a global perspective. It is performed by rotating the whole point vector by camera orientation quaternion, which was already converted from NED (North–East–Down) to the ENU world reference system. Then, it follows the translation to the camera position. The position was also converted from NED to ENU world reference system. The resulting point cloud must then be discretized. The point cloud is scaled by performing an element-wise division by the selected V value and rounding each element to the nearest integer. This results in the voxel indices array O of currently visible obstacles.

The second stage of mapping is known as empty space detection. This is crucial for cleaning up obstacle data that is no longer valid due to introduced depth estimation positioning errors and environmental changes. The known empty space voxel indices array E is collected by applying Bresenham [55] 3D ray-casting algorithm from camera voxel to each obstacle voxel from O .

The final stage combines previously collected obstacle data O and empty space data E into the global map. In our case, the global map is a continuous 3D byte array with default values set to the chosen $occup_unkn$ value representing unknown space. The known space voxel values fall into the occupancy intensity interval $[occup_min; occup_max]$. A lower value means lower obstacle certainty and higher values mean vice versa. In our study, intensity values are capped in $[0; 12]$ intervals. One specific value in this interval is chosen as the occupancy threshold value. Reaching it would mean that the voxel can be treated as occupied. Modification of occupancy intensity values starts by changing the voxel value from $occup_unkn$ to $occup_min$ when the voxel index first occurs in either O or E . Then, values at E are decreased by ray_miss constant (1 was used in this work). Finally, values at O are increased by the ray_hit constant (8 was used in this work).

A series of these iterations results in one global dynamic probability-based occupancy map that can be used for online navigation. The result of such a mapping process is visualized in Figure 6.



Figure 6. Snapshot of AirSimNH environment mapping result using depth images acquired from AirSim. (a) RGB image; (b) occupancy grid map visualized in rviz. Different color blocks represent known occupied space: blue means higher occupancy certainty; red means lower.

3.5. Path Planning and Collision Avoidance

The path planning and collision avoidance in the occupancy map are implemented using the A* search algorithm [56] limited to N_{max} number of iterations. In order for A* to be able to find a path point, set p to a given destination point G in an unknown and/or dynamic environment 3D grid $\mathbb{Z}^3$, perform an optimistic search in the unobserved space, and replanning actions, when certain criteria are met, have to be added. However, these additions do not come without challenges.

As any point in space, including the drone's exact position, is abstracted by a single voxel, it is hard to estimate the precise distance to obstacles, i.e., 2 points approximated by neighboring voxels could be vanishingly close to each other or as far as two V values apart. This increases the probability of a collision with an environment. To counteract that, path planning is only allowed in voxels that are unknown or not occupied and have 26 neighboring voxels that are also unknown or not occupied. In other words, path p is valid if set of voxels E does not contain known occupied voxels. This increases the distance to obstacles by a single V value and reduces the number of collision scenarios.

$$E = P \cup \left\{ (x', y', z') \in \mathbb{Z}^3 \mid \exists (x, y, z) \in P, |x - x'| \leq 1, |y - y'| \leq 1, |z - z'| \leq 1 \right\} \quad (3)$$

As we allow path planning in known space, even static obstacles will inevitably appear in the middle of the planned path, requiring path adjustments. As mentioned, these adjustments, in our case, are made in the form of replanning, which itself is a repeated path planning action from current point C to previous goal point G .

Replanning is triggered after one of these conditions are met:

1. A new obstacle is detected inside set E ;
2. Replanning is also performed if the drone drifts away from the planned path by a specified distance d , i.e., if $|p \in P \mid \|p - C\| \leq d| = 0$;
3. Intermediate goal point I is reached. Point I is a voxel with best known distance heuristic to goal G after failure to find a complete path to G in N_{max} iterations.

4. Results

4.1. Fine-Tuning of Depth Estimation Models

Three datasets were generated for the Blocks, MSBuild2018, and AirsimNH environments using the previously provided algorithm. Each dataset consisted of 50 thousand pairs of RGB and depth images. Additionally, the dataset was generated in subsets of

10 thousand image pairs. Each time a subset generation was completed, a new depth estimation model was trained alongside the previously generated images and the new subset. In simulations, it was observed that as the number of images increased, the performance of the fine-tuned depth estimation model improved. The drone demonstrated improved capabilities in planning the path towards the goal coordinate as more data were introduced to the depth estimation model. Moreover, while inspecting the dataset, some outlier coordinates were observed, such as locations inside buildings that lacked a complete interior and were below the environment's ground. Furthermore, some coordinates were generated inside buildings with complete interiors. However, these coordinates could also be considered outliers because the drone is not expected to navigate inside buildings. These outliers likely harmed the quality of the depth estimation model.

Nevertheless, despite probable outliers, these images were used to finetune models. After each training epoch, the models were evaluated and compared against collected AirSim depth images to quantify performance (Figure 7). The depth estimation models were trained using the following hyperparameters: the learning rate of 1×10^{-4} , a batch size of 8, the AdamW optimizer, and 15 epochs. All three datasets show a steady loss reduction across early epochs, but no noticeable improvements are seen after the 11th epoch. The model trained on the Blocks environment retained the lowest loss compared to other environments. This is likely due to fewer depth ambiguities and more consistent scene structure in Blocks, while AirSimNH and MSBuild2018 had more complex scenes.

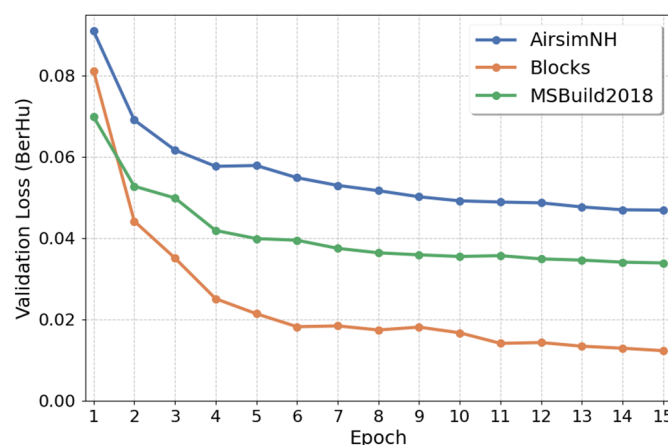


Figure 7. BerHu loss was experienced during training on the depth estimation model. Different lines for different environments: blue for AirsimNH, orange for Blocks, and green for MSBuild2018.

After conducting qualitative inspections and examining whether the predicted depth maps accurately captured major scene structures (e.g., walls, objects, and terrain features), it can be seen that the results are quite noticeable. From the evaluation in Figure 8, it is visually noticeable that the base pretrained model on the AirSimNH environment tends to estimate objects a lot further in comparison to ground truth images from simulation. In contrast, the fine-tuned model captures objects better with up-close details, higher mapping accuracy, and a more diverse depth range. A high difference in MAPE can be observed in Table 3 between models. Depth image discrepancies between pretrained and fine-tuned models can lead to the belief that models should be specialized for each environment. Since higher error discrepancies become more apparent in mismatched domains, the likelihood of crashes or collisions increases significantly, as the model's inaccurate depth predictions may fail to capture obstacles and distances correctly.

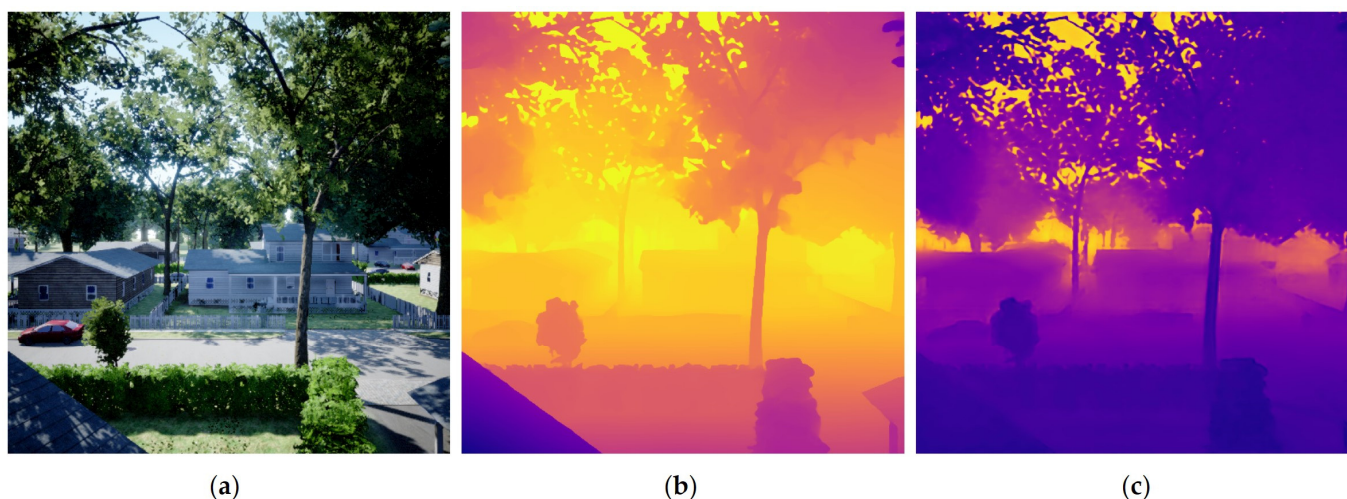


Figure 8. Visualizations comparing: (a) RGB input; (b) pre-trained Depth Anything V2 small model's depth image output; and (c) fine-tuned version's depth image. The image was captured in the AirSimNH environment.

Table 3. Mean Absolute Percentage Error (MAPE) comparison between pre-trained and fine-tuned models across different environments.

Environment	Pre-Trained Model	Finetuned Model
AirSimNH	120.45%	33.41%
Blocks	70.09%	8.04%
MSBuild2018	121.94%	32.86%

4.2. Autonomous Navigation Simulation Results

4.2.1. General Experimentation Results

During the experimentation phase of this work, the navigation system was tested in three different environments. In total, 810 flights were conducted: 10 for each scenario and voxel size V [0.5, 1.0, 2.0] pair for each environment. These autonomous flights were performed using depth images acquired directly from AirSim, depth images estimated with the pre-trained Depth Anything V2 model, and depth images estimated by models finetuned for each environment.

The resulting metrics of the flights conducted during the experimentation are shown in Table 4. Here, the results from all the scenarios and voxel sizes used in each environment are added for each depth image acquisition method. The third column shows the number of flights that reached the goal position successfully within 5 m. The fourth column shows how close to the goal all the failed flights ended on average.

As can be seen, when depth images were estimated by a pre-trained model, not a single simulation successfully reached the goal. The paths flown during these simulations are depicted in Appendix A, Figure A1. Most of the flights ended prematurely when the drone would become stuck in the ground or other obstacles and could not move further because of the simulator's physics. This is mainly because the used model does not generalize well for these digital environments and also suffers from scale-drift issues or problems where closer objects occupying more of the view are estimated further away or vice versa.

Table 4. Results from autonomous flights simulated in different AirSim environments with varying methods of depth image acquisition.

Environment	Method	Reached Goal	Distance to Goal
AirSimNH	Depth Images from AirSim	35 of 90	117.40 (55 unsuccessful)
	Estimated Depth Images (finetuned)	3 of 90	135.28 (87 unsuccessful)
	Estimated Depth Images (pre-trained)	0 of 90	171.20 (90 unsuccessful)
Blocks	Depth Images from AirSim	79 of 90	67.75 (11 unsuccessful)
	Estimated Depth Images (finetuned)	65 of 90	95.95 (25 unsuccessful)
	Estimated Depth Images (pre-trained)	0 of 90	152.34 (90 unsuccessful)
MSBuild2018	Depth Images from AirSim	12 of 90	187.80 (78 unsuccessful)
	Estimated Depth Images (finetuned)	0 of 90	230.50 (90 unsuccessful)
	Estimated Depth Images (pre-trained)	0 of 90	311.25 (90 unsuccessful)

The finetuning of the models improved the number of successful navigations by 72.22% and 3.33% in Blocks and AirSimNH environments, respectively. However, using this method, the drone still did not manage to reach a single goal in MSBuild2018 simulations. Utilization of finetuned models for depth estimation also helped to get the drone closer to the goal in flights where the simulation ended prematurely. The average distance to goals improved by 36, 56, and 81 m in AirSimNH, Blocks, and MSBuild2018 environments.

Across all environments, the processed frames per second (FPS) range from approximately 2.7 to 4.35, depending on the voxel size and complexity of the environment. The full list of averaged frames per second can be seen in Appendix A, Table A1. All of the values indicate that the drone travels ~0.46 m per frame in the highest FPS scenario, while in the lowest FPS scenario, it travels ~0.74 m per frame. These measurements show that the drone replans its path after covering these distances.

Increasing FPS is critical when solving problems like this because if the drone does not receive new depth images that indicate an obstruction before a collision, it cannot replan its path before the collision with an object. For this reason, a higher FPS indicates improved drone performance, faster responses to obstacles, and more frequent optimal path re-evaluations.

Furthermore, FPS measurements show that the lowest performance was observed in the AirsimNH environment, while the best was in the Blocks environment. Visually, Blocks is the least detailed environment, and AirsimNH is the most thorough. It indicates that either environment complexity or graphical demand impacts the image processing rate. Due to limited data, it is not possible to determine which factor plays a more significant role.

When voxel size is increased, the image processing rate improves due to reduced computational requirements for path planning as larger voxel sizes simplify the complexity of the environment—additionally, safety margins increase, leading to a reduced risk of collision. However, when V is decreased, smaller objects and narrower paths may go unnoticed or overestimated, leading to suboptimal paths, blocked narrow passages, or even collisions. While lower voxel sizes may provide a more precise path, it also increases execution costs, reduces image processing rate, and decreases reaction times, leading to more challenging path planning.

Lastly, testing cases utilizing AirSim depth images demonstrate the lowest FPS values, primarily due to additional preprocessing steps such as depth image normalizations and format conversion. Additionally, the finetuned depth estimation model displays lower FPS than a pre-trained model, which is expected, as fine-tuning increases the model complexity of the hidden layers, leading to increased processing times.

4.2.2. Acquired AirSim Depth Images vs. Finetuned Depth Estimation

The averaged results for each pair of environment and depth image acquisition methods are in Tables 5–7. Here, the columns denote the voxel size in meters discretized from point cloud data, the scenario number specifying which coordinates in Table 2 were used for the end goal, the number of flights that reached the goal within 5 m, the average number of steps that collided with objects in the area, the average calculated Euclidean flight distance in meters, the average time of flight in seconds, and average calculated distance in meters to goal coordinates from the last step before premature termination of simulation such as collision for 20 consecutive steps (the drone is stuck in an obstacle and cannot move) or failure of 200 consecutive iterations to find a new path. These constraints ensure that the drone does not keep hovering in place for an infinite time during experiments.

The snapshots in Figures 9–11 depict all the paths the drone took to reach the end positions from the start in different environments for the AirSim acquired and estimated depth image usage. Here, red lines are the flights with voxel size V of 0.5 m; green lines show paths using 1.0 m; blue lines are 2.0 m.

Table 5. Autonomous flight results comparison between AirSim and estimated depth image usage in AirSimNH environment. The results are calculated from ten flights for each voxel size and scenario pair. The best results for each scenario are bolded.

Voxel Size	Scenario	AirSim Depth Images					Estimated Depth Images				
		Reached Goal	Collisions	Distance	Time	Distance to Goal	Reached Goal	Collisions	Distance	Time	Distance to Goal
0.5	1	3	12.00	226.75	136.42	129.56	0	-	-	-	120.26
	2	3	0.67	228.47	132.29	125.93	3	5.00	282.79	176.08	126.85
	3	0	-	-	-	142.63	0	-	-	-	128.47
1.0	1	4	3.00	345.22	212.07	41.13	0	-	-	-	124.08
	2	7	0.43	288.95	174.66	78.63	0	-	-	-	133.49
	3	3	4.67	439.83	271.87	131.08	0	-	-	-	147.58
2.0	1	9	0.22	346.49	225.01	72.87	0	-	-	-	130.44
	2	6	0	348.58	224.92	172.17	0	-	-	-	140.68
	3	0	-	-	-	162.58	0	-	-	-	165.69

Table 6. Autonomous flight results comparison between AirSim and estimated depth image usage in the Blocks environment. The results are calculated from ten flights for each voxel size and scenario pair. The best results for each scenario are bolded.

Voxel Size	Scenario	AirSim Depth Images					Estimated Depth Images				
		Reached Goal	Collisions	Distance	Time	Distance to Goal	Reached Goal	Collisions	Distance	Time	Distance to Goal
0.5	1	7	0.71	178.51	95.62	122.58	10	0	196.28	112.91	-
	2	10	0	160.09	83.25	-	9	0	160.16	88.51	106.87
	3	8	2.63	221.09	122.61	129.52	8	0.63	222.76	128.11	90.84
1.0	1	9	0	179.57	97.30	124.76	7	0	207.58	118.11	129.32
	2	10	0	160.64	85.98	-	7	0	161.32	92.70	102.68
	3	7	0.71	226.13	125.64	109.51	5	0	226.93	133.94	58.17
2.0	1	8	0.25	188.67	118.40	123.39	4	0	350.40	217.18	119.89
	2	10	0	166.80	106.18	-	9	0	186.23	119.91	96.14
	3	10	0	209.12	131.73	-	9	0	235.37	153.67	159.64

Table 7. The autonomous flight result comparison between AirSim and estimated depth image usage in the MSBuild2018 environment. The results are calculated from ten flights for each voxel size and scenario pair. The best results for each scenario are bolded.

Voxel Size	Scenario	AirSim Depth Images				Estimated Depth Images					
		Reached Goal	Collisions	Distance	Time	Distance to Goal	Reached Goal	Collisions	Distance	Time	Distance to Goal
0.5	1	6	2.33	376.99	217.90	53.96	0	-	-	-	104.55
	2	4	25.00	472.27	283.29	176.02	0	-	-	-	222.09
	3	0	-	-	-	218.00	0	-	-	-	254.27
1.0	1	1	43.00	376.29	224.41	174.06	0	-	-	-	216.62
	2	0	-	-	-	160.91	0	-	-	-	246.09
	3	0	-	-	-	234.73	0	-	-	-	268.21
2.0	1	4	14.25	375.22	220.60	158.99	0	-	-	-	227.49
	2	0	-	-	-	236.57	0	-	-	-	247.53
	3	0	-	-	-	276.91	0	-	-	-	287.67

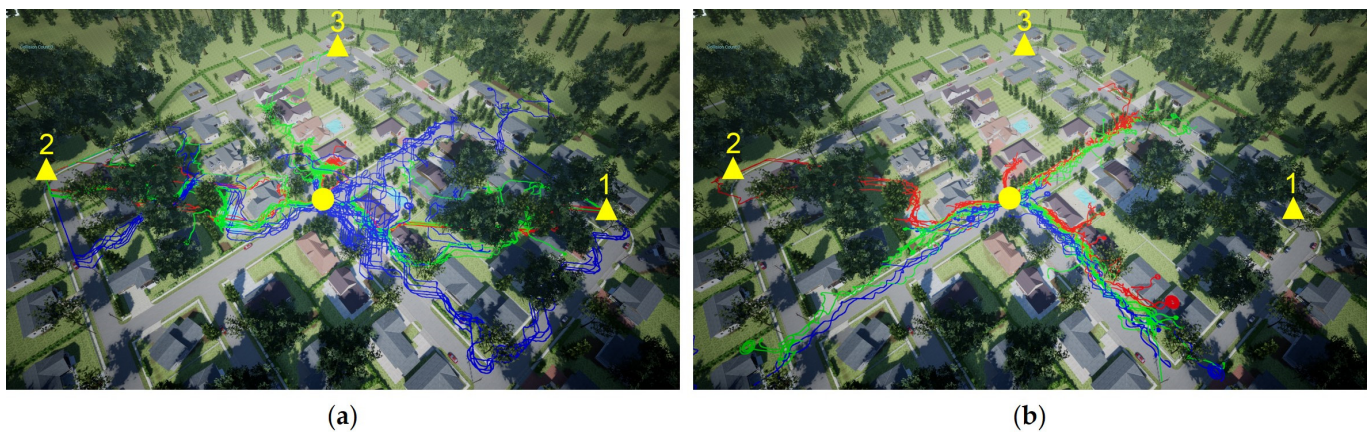


Figure 9. Comparison of flight paths during experimentation in the AirSimNH environment between (a) AirSim depth image and (b) estimated depth image usage. Different path colors mean different occupancy grid voxel size: red lines 0.5, green lines 1.0, blue lines 2.0. Yellow circles show start positions, triangles—goals. Scenarios are numbered accordingly.

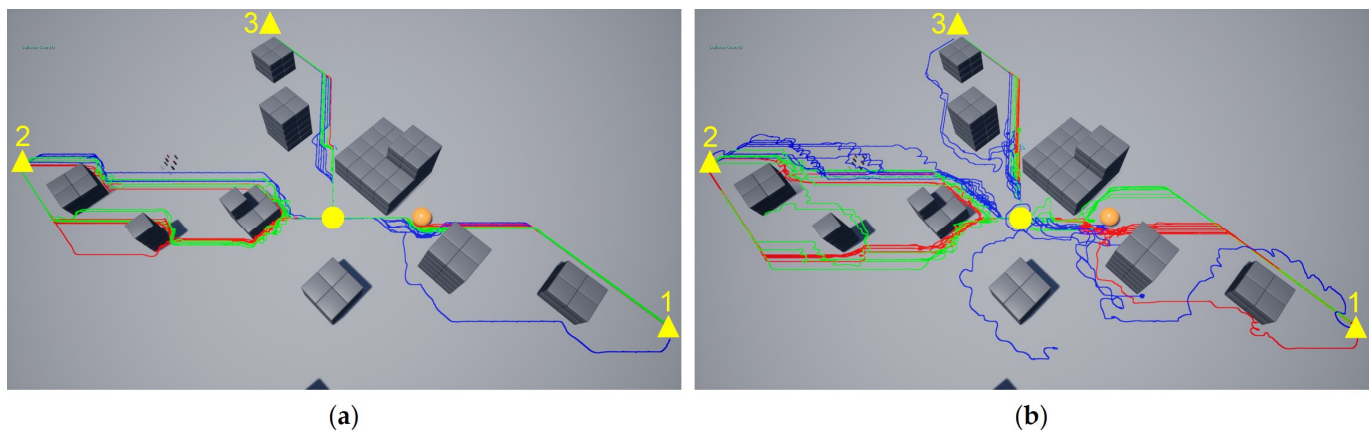


Figure 10. Comparison of flight paths during experimentation in Blocks environment between (a) AirSim depth image and (b) estimated depth image usage. Different path colors mean different occupancy grid voxel sizes: red lines 0.5, green lines 1.0, blue lines 2.0. Yellow circles show start positions, triangles—goals. Scenarios are numbered accordingly.

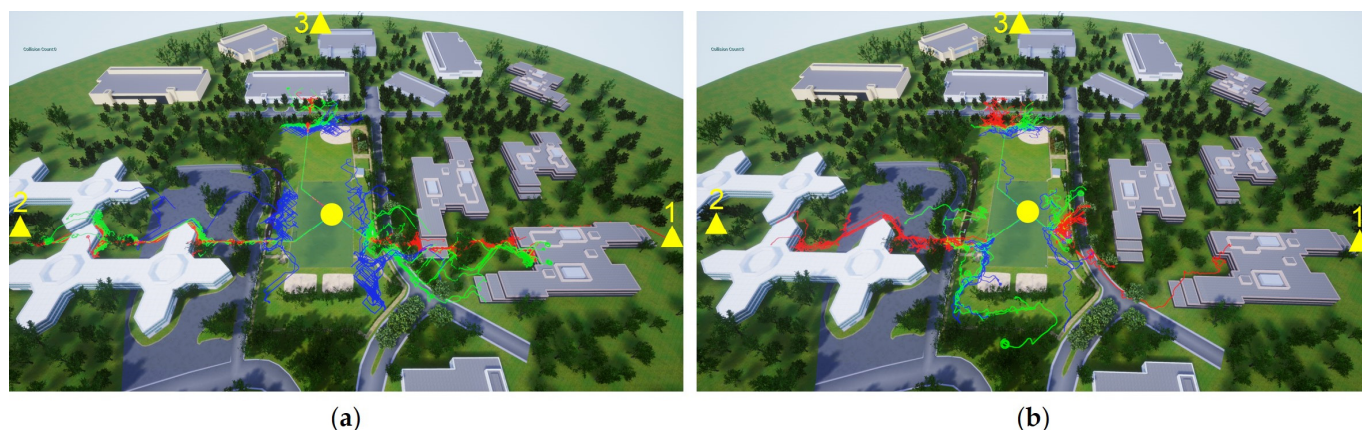


Figure 11. Comparison of flight paths during experimentation in MSBuild2018 environment between (a) AirSim depth image and (b) estimated depth image usage. Different path colors mean different occupancy grid voxel sizes: red lines 0.5, green lines 1.0, blue lines 2.0. Yellow circles show start positions, triangles—goals. Scenarios are numbered accordingly.

A clear difference in navigation routes can be seen between AirSim depth image and estimated depth image use when comparing flights in complex AirSimNH environment (Figure 9). Even with bigger voxel sizes, the drone can navigate between houses, trees, and other objects using depth acquired from AirSim (a). However, the drone would often fly too close to obstacles and become stuck when the voxel sizes are smaller. On the other hand, the estimated depth images are still too noisy and allow errors to accumulate in the occupancy grid map of the environment as can be seen in Appendix B, where Figure A2 depicts differences in occupancy grids (a for this environment, scenario no.2). This leaves the drone frantically looking around as the path planning algorithm recalculates to navigate in some small opening, only to close that opening with more voxels discretized from noisy depth data.

The best results for each scenario for the AirSimNH environment are bolded in Table 5. The third scenario was the toughest to navigate, even for a drone acquiring depth images directly from AirSim. Only three flights ended successfully in goal coordinates, all with 1.0 m for voxel size. Scenarios no.1 and no.2 both had sixteen flights reaching the set goals when using AirSim depth images. The most successful flights were recorded using $V = 2.0$ for the first scenario and $V = 1.0$ for the second. However, for the successful flights, the lowest distance and time traveled were reached when $V = 0.5$.

The simpler Blocks environment proved that the depth estimation methodology can be successfully utilized in such conditions. However, the drone navigates more chaotically (Figure 10b) and travels a greater distance. Unlike when using depth data from AirSim, the errors accumulated in estimated depth images steered the drone further from obstacles. This helped the former to outperform the latter in the first scenario with the orange ball, as noted in Table 6. In the case of successfully finished flights, the collision count was generally lower for estimation methodology. As previously stated, the occupancy grid map examples for this environment scenario no.1 can be seen in Appendix B, Figure A2b.

All the flights reached goals set in the second scenario when utilizing AirSim depth images while the least distance and time was traveled with $V = 0.5$. The most successful simulations and least average distance traveled in scenario no.3 were recorded with the largest voxel size. All the flights with a $V = 0.5$ setting reached the goal in the first scenario when depth images were estimated with the finetuned model; however, the distance and time traveled were still better with depth images from AirSim. Out of the failed attempts, the lower distance to the goal was calculated using the estimated depth image method.

The most difficult environment proved to be MSBuild2018. It starts in an open field surrounded by dense tree lines from all directions. There are some openings to the roads and parking lots, but because the path planner starts by giving the most optimal routes, the drone tries to steer between the narrow openings of tree lines when $V = 0.5$ (red lines in Figure 11). As the voxel size increases, the drone can no longer see the gaps in the occupancy map, so it tries to go higher—above the trees and buildings. As the depression angle is low from step to step, the drone is not able to build an occupancy grid map of further area, the planner fails to find the next optimal route, and the simulation finishes prematurely. This happens in flights with AirSim acquired and estimated depth images, as seen from comparison snapshots.

Using the depth images from AirSim, the first scenario was successfully navigated only 11 times out of 30, and only one other time in scenario no. 2 (Table 7). The depth image estimation technique failed to reach any goal. It also did not help to reach fewer distance to goals even in failed attempts. All of this shows that the navigation algorithm needs some improvements to navigate such complex environments.

5. Discussion

One of the primary hurdles that were faced in the implementation of the navigation system was poor obstacle avoidance due to the drone constantly following the walls of the obstacles while the drone's inertia, i.e., external forces, was not being accounted for. This behavior was observed not only with depth estimation but also when utilizing depth images acquired with AirSim. To improve this, instead of an algorithm that searches in neighboring nodes, i.e., A^* , we could use path-finding algorithms that search in all directly observable nodes. A good potential candidate could be Theta* or JPS. The latter was used in [57] to achieve safe and fast collision-free flights. Another potential solution could be smoothing paths by interpolating them using splines along with gradient-based obstacle avoidance. Such a solution was used in [58] to achieve collision-free flights in unknown environments.

Currently, path searching is performed using the A^* algorithm. Replanning is performed by repeating the path search from the current position. This is very inefficient. In future research, we will investigate path planning and navigation efficiency improvements using algorithms that do not have to recalculate the whole path after obstacle encounters, e.g., [36,37]. Also, for better planning efficiency in high-complexity environments, we will investigate sampling-based algorithms, as it was performed in [38–40].

Image generation also came with its challenges as some outliers ended up in the dataset. For example, interiors of the buildings and objects with transparent walls captured from inside or underground. These outliers were included due to limitations of the tools that AirSim provides to verify if the coordinates of the teleported drone are not inside an object or outside the map. The simulator only indicates collisions if the drone becomes stuck inside thinner objects or walls. A more sophisticated data generation script could be developed to detect and exclude mentioned outliers. This could be performed by either filtering the outliers from the dataset or preventing them from being generated in the first place by obtaining the correct sizes and coordinates of objects inside the environment and excluding generated coordinates in those areas. Another option would be generating the dataset by manually flying the drone and capturing the images in specific time intervals. Furthermore, all the images were generated under similar weather conditions which presents limitations if different weather conditions were introduced into the simulation environment. To improve the robustness of depth estimation and train models that generalize well, images with varying illumination (weather conditions) could be introduced for the datasets. Also,

in further research, the performance of a single model trained on a larger combined dataset needs to be investigated.

The monocular single-shot depth estimation model Depth Anything V2, despite being among the most accurate ones, failed to keep consistent depth estimations across multiple frames. This could be improved by integrating more accurately known distances to certain key points obtained by SLAM (or other feature-based approaches) into the depth estimation model or utilizing another depth estimation approach better suited for UAV navigation (multi-frame estimation, taking video sequences as input). However, this would require more computational resources that could not be spared in an autonomous standalone system. Also, the monocular single-shot depth estimation model introduced problems when facing or getting too close to a wall. Since depth information could not be extracted from such images due to a lack of details, the model would highly underestimate depth and tend to crash.

The frame processing rate observed in the simulation was low, indicating slow response times and an increased risk that the drone will not be able to replan its path in time before colliding with objects that are yet to be identified. However, the Unreal Engine environment itself used a lot of resources during these simulations, and the FPS is lower than it would be in a real-world environment when the processing power is used only on gathering RGB images, depth estimation, and path planning. To better assess the frame processing rate using the current equipment or to determine system requirements that would satisfy the desired FPS, future tests should be designed to reduce resource usage from external sources. Rendering of the Unreal Engine environment should be minimized only to the camera's field of view to reduce the usage of the view that is used to navigate the drone. Modifications to the testing environment could include the utilization of two GPUs: one dedicated to rendering the environment and another responsible for image retrieval, depth estimation, and path planning. This solution would reduce the computational load of the rendering process, making performance closer to real-world conditions. Image processing FPS will increase in real-world applications when the resources are not occupied by the simulation environment, which in conducted tests used about 25–50% of the GPU resources. Furthermore, image extraction from the AirSim environment had some latency due to the API, where using an onboard processing unit on a drone with a monocular camera would be a more direct, faster approach to extract images. All in all, the main problem influencing FPS in the conducted research is the environment itself and its framework, which would not influence the FPS in real-world applications and the FPS could improve by two or more times.

Furthermore, the current implementation of the system is synchronized between image generation and navigation. Concurrency and better performance could be achieved if mapping, navigation, and depth processing calculations were parallelized.

Additionally, three simulation environments were used in the simulations with varying complexities. One environment was relatively simple, while the other two were highly complex. Further research will focus on improving navigation in complex environments. With more optimal navigation solutions, additional challenges such as moving objects like cars and animals, and more challenging weather effects like clouds, fog, or other weather conditions, will be explored. These additional environmental conditions will increase the complexity of the environment and assess how well the proposed solution adapts to dynamically moving objects, brightness, visibility, etc.

Lastly, all tests were performed using simulations on a desktop computer. For real-life applications, all calculations must be performed on a miniature onboard computer with CUDA capabilities. NVIDIA Jetson TX2 with 256 CUDA cores, 1.3 GHz dual-core CPU, and 8GB of memory should be up for this task. However, this assumption must be evaluated practically.

6. Conclusions

The proposed UAV navigation framework was developed for autonomous flight simulation. It utilizes the Depth Anything V2 deep learning model to estimate depth images from single RGB images captured by a monocular front-facing camera. These images are used to dynamically build a probability-based occupancy grid map. The A* algorithm is then used to plan optimal paths in the built map for the drone to navigate towards the coordinates of a target.

The navigation solution was tested across three different open-source environments in an AirSim simulator with proposed scenarios. Results showed successful autonomous drone navigation in low-complexity environments, while highly complex settings presented challenges such as inconsistent depth estimations and drone clipping into objects with trickier geometries.

Finetuning depth estimation models for specific environments improved the accuracy and reliability of the system, reducing the likelihood of navigation failures caused by mismatched depth predictions. The results demonstrate a substantial decrease in MAPE across different environments after the finetuning process. The MAPE was reduced from 120.45% to 33.41% in AirSimNH, from 70.09% to 8.04% in Blocks, and from 121.94% to 32.86% in MSBuild2018.

Navigation performance improvements were also observed when utilizing fine-tuned models instead of pretrained. The number of goals reached in all scenarios increased by 3.33% in AirSimNH and 72.22% in Blocks. Furthermore, the distance to goal positions in unsuccessful flights decreased by 36 m in AirSimNH, 56 m in Blocks, and 81 m in MSBuild2018. These findings emphasize the importance of domain-specific model adaptation, demonstrating that finetuning with synthetic data that is tailored to a given environment allows for more accurate depth estimation and better overall system performance in autonomous UAV navigation.

However, the frame processing rate (ranging from 2.7 to 4.4 FPS) indicates slow response times, which could lead to collisions with objects along previously planned and outdated paths. The simulation conditions suggest that these results, which appear worse than they would be in real-life conditions, could be misleading. Despite this limitation, modifications of the simulation environment setup to provide more system resources to the algorithm would improve the performance and response time of the proposed solution.

Further refinement is required for depth estimation methods, navigation algorithms, and overall system architecture to achieve more reliable navigation in broader and higher-complexity environments. Further research is also needed on the navigation solution's robustness against different illumination conditions and environment feature levels. Despite this, the authors hope that the proposed framework and simulation scenarios will be useful for researchers to compare and improve methodologies used in monocular vision-based autonomous navigation.

Author Contributions: Conceptualization, J.G., H.G., T.K., L.P. and A.K.; methodology, J.G., H.G., T.K., L.P. and A.K.; software, J.G., T.K. and L.P.; validation, J.G., H.G., T.K., L.P. and A.K.; formal analysis, J.G., H.G., T.K., L.P. and A.K.; investigation, J.G., H.G., T.K., L.P. and A.K.; resources, J.G., T.K. and L.P.; data curation, H.G., J.G., T.K. and L.P.; writing—original draft preparation, H.G.; writing—review and editing, J.G., H.G., T.K., L.P. and A.K.; visualization, H.G., J.G., T.K. and L.P.; supervision, A.K.; project administration, H.G., J.G., T.K., L.P. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original data presented in the study are openly available at <https://github.com/jonasctrl/monocular-slam-drone> (accessed on 22 March 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

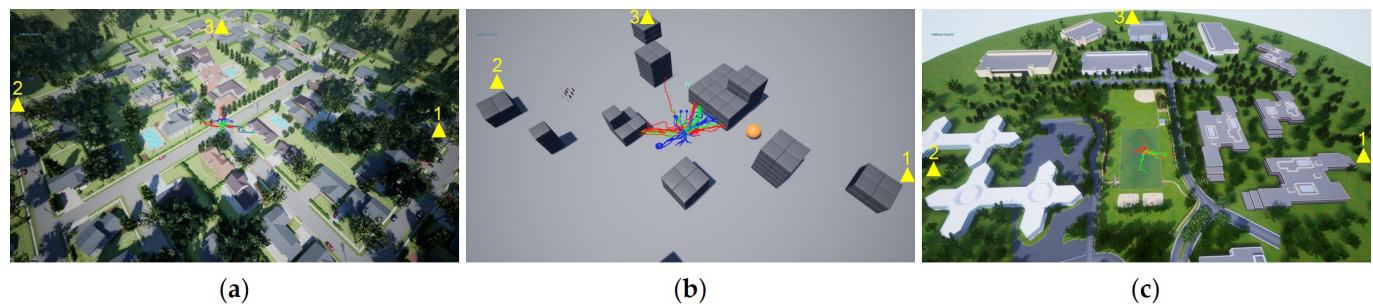


Figure A1. Paths of flight during experimentation using pre-trained Depth Anything V2 model for depth image estimation. Environments: (a) AirSimNH; (b) Blocks; and (c) MSBuild2018. Different path colors mean different voxel sizes: red lines 0.5, green lines 1.0, blue lines 2.0. Yellow triangles show end points for different scenarios. Scenarios are numbered accordingly.

Table A1. Image processing rates (FPS) for AirsimNH, Blocks, and MSBuild2018 environments across voxel sizes of 0.5, 1, and 2, as well as an additional overall FPS measure. Comparison of image processing rate with different depth retrieval methods: true depth image from AirSim, fine-tuned, and pretrained depth estimation models.

Environment	Voxel Size	AirSim Depth Images	Fine-Tuned Model	Pretrained Model
AirsimNH	0.5	2.72	2.77	3.10
	1	2.99	3.03	3.15
	2	3.32	3.15	3.39
	Overall	3.01	2.98	3.21
Blocks	0.5	3.88	3.59	4.00
	1	4.32	3.86	4.30
	2	4.35	4.00	4.38
	Overall	4.19	3.82	4.22
MSBuild2018	0.5	3.03	3.57	3.67
	1	3.64	3.79	3.91
	2	4.08	3.91	4.17
	Overall	3.58	3.76	3.92

Appendix B

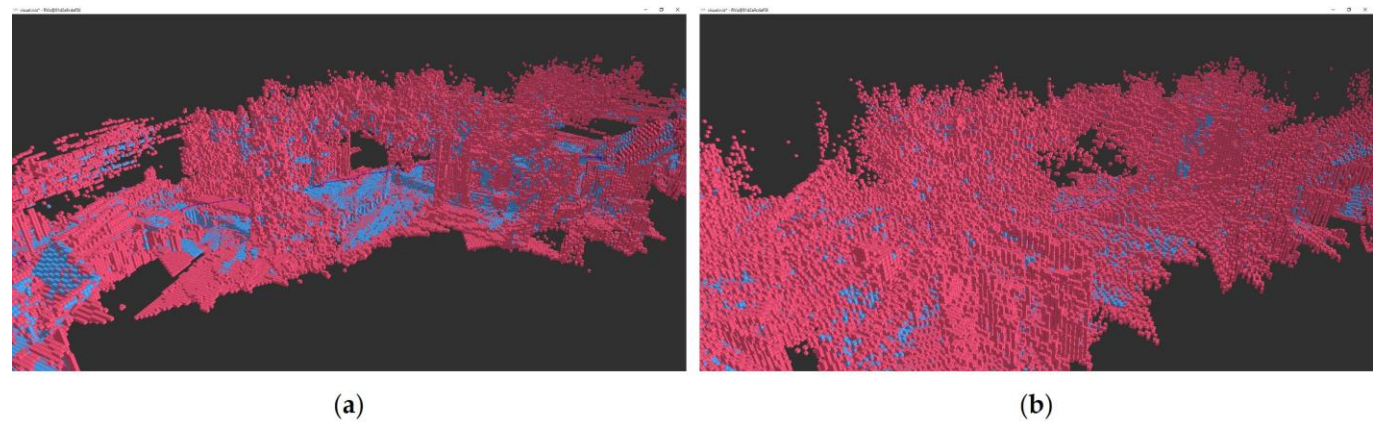


Figure A2. Cont.

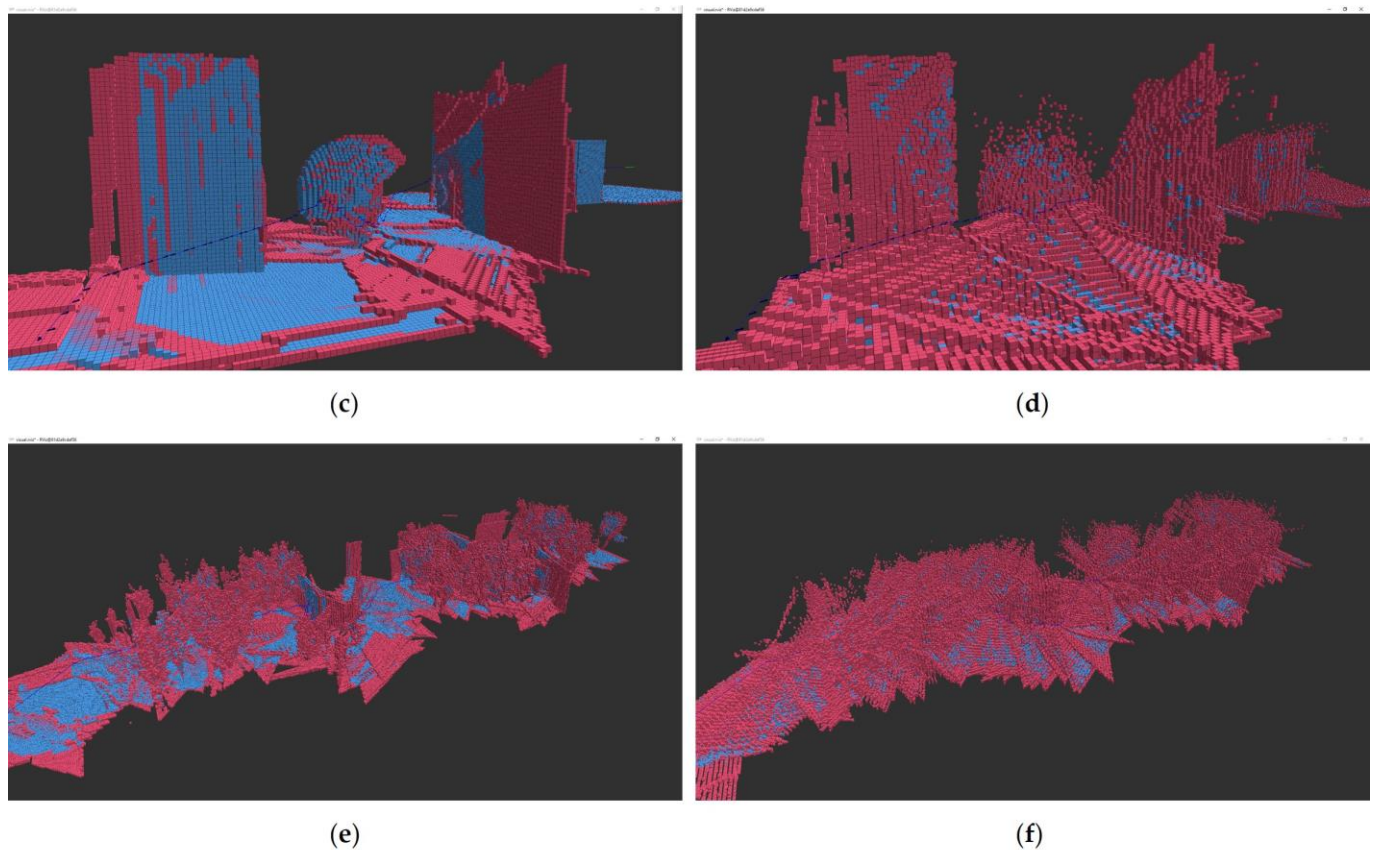


Figure A2. Snapshots of occupancy grid maps built during autonomous navigation simulations and visualized in rviz. Depth images acquired from AirSim and estimated with finetuned models were used for different environments and scenarios, respectively: AirSimNH scenario no.2 (a,b); Blocks scenario no.1 (c,d); and MSBuild2018 scenario no.1 (e,f). Voxels of different colors depict different certainty of known occupied space: blue means higher certainty of occupancy; red means lower certainty. Map size is 800×800 (except for e and f its 1200×1200) and voxel size is 0.5.

References

- Shakhatreh, H.; Sawalmeh, A.H.; Al-Fuqaha, A.; Dou, Z.; Almaita, E.; Khalil, I.; Othman, N.S.; Khreishah, A.; Guizani, M. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access* **2019**, *7*, 48572–48634. [\[CrossRef\]](#)
- Gyagenda, N.; Hatilima, J.V.; Roth, H.; Zhmud, V. A Review of GNSS-Independent UAV Navigation Techniques. *Rob. Auton. Syst.* **2022**, *152*, 104069. [\[CrossRef\]](#)
- Yang, N.; Wang, R.; Gao, X.; Cremers, D. Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias and Rolling Shutter Effect. *arXiv* **2018**, arXiv:1705.04300. [\[CrossRef\]](#)
- Arampatzakis, V.; Pavlidis, G.; Mitianoudis, N.; Papamarkos, N. Monocular Depth Estimation: A Thorough Review. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 2396–2414. [\[CrossRef\]](#)
- Wang, C.; Ma, H.; Chen, W.; Liu, L.; Meng, M.Q.-H. Efficient Autonomous Exploration with Incrementally Built Topological Map in 3-D Environments. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 9853–9865. [\[CrossRef\]](#)
- Faria, M.; Ferreira, A.S.; Pérez-Leon, H.; Maza, I.; Viguria, A. Autonomous 3D Exploration of Large Structures Using an UAV Equipped with a 2D LIDAR. *Sensors* **2019**, *19*, 4849. [\[CrossRef\]](#) [\[PubMed\]](#)
- Vanegas, F.; Gaston, K.J.; Roberts, J.; Gonzalez, F. A Framework for UAV Navigation and Exploration in GPS-Denied Environments. In Proceedings of the 2019 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2019; pp. 1–6.
- Chang, Y.; Cheng, Y.; Manzoor, U.; Murray, J. A Review of UAV Autonomous Navigation in GPS-Denied Environments. *Rob. Auton. Syst.* **2023**, *170*, 104533. [\[CrossRef\]](#)
- Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In Proceedings of the Field and Service Robotics, Zürich, Switzerland, 13–15 September 2017.
- Lu, Y.; Xue, Z.; Xia, G.-S.; Zhang, L. A Survey on Vision-Based UAV Navigation. *Geo-Spat. Inf. Sci.* **2018**, *21*, 21–32. [\[CrossRef\]](#)
- Arafat, M.Y.; Alam, M.M.; Moh, S. Vision-Based Navigation Techniques for Unmanned Aerial Vehicles: Review and Challenges. *Drones* **2023**, *7*, 89. [\[CrossRef\]](#)

12. Horn, B.K.P.; Schunck, B.G. Determining Optical Flow. *Artif. Intell.* **1981**, *17*, 185–203. [\[CrossRef\]](#)
13. Lucas, B.D.; Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. In Proceedings of the IJCAI'81: 7th International Joint Conference on Artificial Intelligence, Vancouver, BC, Canada, 24–28 August 1981; Volume 2, pp. 674–679.
14. Santos-Victor, J.; Sandini, G.; Curotto, F.; Garibaldi, S. Divergent Stereo for Robot Navigation: Learning from Bees. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, New York, NY, USA, 15–17 June 1993; pp. 434–439.
15. Agrawal, P.; Ratnoo, A.; Ghose, D. Inverse Optical Flow Based Guidance for UAV Navigation through Urban Canyons. *Aerosp. Sci. Technol.* **2017**, *68*, 163–178. [\[CrossRef\]](#)
16. Couturier, A.; Akhloufi, M.A. A Review on Absolute Visual Localization for UAV. *Rob. Auton. Syst.* **2021**, *135*, 103666. [\[CrossRef\]](#)
17. Harris, C.; Stephens, M. A Combined Corner and Edge Detector. In Proceedings of the Proc. AVC, Manchester, UK, 15–17 September 1988; pp. 23.1–23.6.
18. Edward, R.; Drummond, T. Machine Learning for High-Speed Corner Detection. In Proceedings of the Computer Vision—ECCV, Graz, Austria, 7–13 May 2006; Leonardis, A., Bischof, H., Pinz, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 430–443.
19. Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vis.* **2004**, *60*, 91–110. [\[CrossRef\]](#)
20. Calonder, M.; Lepetit, V.; Strecha, C.; Fua, P. BRIEF: Binary Robust Independent Elementary Features. In Proceedings of the Computer Vision—ECCV, Heraklion, Crete, 5–11 September 2010; Daniilidis, K., Maragos, P., Paragios, N., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 778–792.
21. Bay, H.; Tuytelaars, T.; Van Gool, L. SURF: Speeded Up Robust Features. In Proceedings of the Computer Vision—ECCV, Graz, Austria, 7–13 May 2006; Leonardis, A., Bischof, H., Pinz, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 404–417.
22. Rublee, E.; Rabaud, V.; Konolige, K.; Bradski, G. ORB: An Efficient Alternative to SIFT or SURF. In Proceedings of the 2011 International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 2564–2571.
23. Hornung, A.; Wurm, K.M.; Bennewitz, M.; Stachniss, C.; Burgard, W. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Auton. Robot.* **2013**, *34*, 189–206. [\[CrossRef\]](#)
24. Dryanovski, I.; Morris, W.; Xiao, J. Multi-Volume Occupancy Grids: An Efficient Probabilistic 3D Mapping Model for Micro Aerial Vehicles. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010; pp. 1553–1559.
25. Herrera-Granda, E.P.; Torres-Cantero, J.C.; Peluffo-Ordóñez, D.H. Monocular Visual SLAM, Visual Odometry, and Structure from Motion Methods Applied to 3D Reconstruction: A Comprehensive Survey. *Heliyon* **2024**, *10*, e37356. [\[CrossRef\]](#)
26. Campos, C.; Elvira, R.; Rodríguez, J.J.G.; Montiel, J.M.; Tardós, J.D. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multimap SLAM. *IEEE Trans. Robot.* **2021**, *37*, 1874–1890. [\[CrossRef\]](#)
27. Davison, A.J.; Reid, I.D.; Molton, N.D.; Stasse, O. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Trans. Pattern Anal. Mach. Intell.* **2007**, *29*, 1052–1067. [\[CrossRef\]](#)
28. Engel, J.; Schöps, T.; Cremers, D. LSD-SLAM: Large-Scale Direct Monocular SLAM. In Proceedings of the Computer Vision—ECCV, Zurich, Switzerland, 6–12 September 2014; Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 834–849.
29. Yang, L.; Kang, B.; Huang, Z.; Zhao, Z.; Xu, X.; Feng, J.; Zhao, H. Depth Anything V2. *arXiv* **2024**, arXiv:2406.09414.
30. Godard, C.; Mac Aodha, O.; Firman, M.; Brostow, G.J. Digging into Self-Supervised Monocular Depth Prediction. In Proceedings of the International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019.
31. Guizilini, V.; Ambrus, R.; Pillai, S.; Raventos, A.; Gaidon, A. 3D Packing for Self-Supervised Monocular Depth Estimation. *arXiv* **2020**, arXiv:1905.02693.
32. Dhulkefl, E.; Durdu, A.; Terzioğlu, H. Dijkstra algorithm using uav path planning. *Konya J. Eng. Sci.* **2020**, *8*, 92–105. [\[CrossRef\]](#)
33. Zhibo, Z. A Review of Unmanned Aerial Vehicle Path Planning Techniques. *Appl. Comput. Eng.* **2024**, *33*, 234–241. [\[CrossRef\]](#)
34. Mandloi, D.; Arya, R.; Verma, A.K. Unmanned Aerial Vehicle Path Planning Based on A* Algorithm and Its Variants in 3d Environment. *Int. J. Syst. Assur. Eng. Manag.* **2021**, *12*, 990–1000. [\[CrossRef\]](#)
35. Luo, Y.; Lu, J.; Zhang, Y.; Qin, Q.; Liu, Y. 3D JPS Path Optimization Algorithm and Dynamic-Obstacle Avoidance Design Based on Near-Ground Search Drone. *Appl. Sci.* **2022**, *12*, 7333. [\[CrossRef\]](#)
36. Suanpang, P.; Jamjunr, P. Optimizing Autonomous UAV Navigation with D* Algorithm for Sustainable Development. *Sustainability* **2024**, *16*, 7867. [\[CrossRef\]](#)
37. Zhu, X.; Yan, B.; Yue, Y. Path Planning and Collision Avoidance in Unknown Environments for USVs Based on an Improved D* Lite. *Appl. Sci.* **2021**, *11*, 7863. [\[CrossRef\]](#)
38. Yang, K.; Gan, S.K.; Sukkarieh, S. A Gaussian Process-Based RRT Planner for the Exploration of an Unknown and Cluttered Environment with a UAV. *Adv. Robot.* **2013**, *27*, 431–443. [\[CrossRef\]](#)
39. Yan, F.; Zhuang, Y.; Xiao, J. 3D PRM Based Real-Time Path Planning for UAV in Complex Environment. In Proceedings of the 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), Guangzhou, China, 11–14 December 2012; pp. 1135–1140.

40. Candeloro, M.; Lekkas, A.M.; Hegde, J.; Sørensen, A.J. A 3D Dynamic Voronoi Diagram-Based Path-Planning System for UUVs. In Proceedings of the OCEANS 2016 MTS/IEEE Monterey, Monterey, CA, USA, 19–23 September 2016; pp. 1–8.
41. Bi, Y.; Lan, M.; Li, J.; Zhang, K.; Qin, H.; Lai, S.; Chen, B.M. Robust Autonomous Flight and Mission Management for MAVs in GPS-Denied Environments. In Proceedings of the 2017 11th Asian Control Conference (ASCC), Gold Coast, Australia, 17–20 December 2017; pp. 67–72.
42. Sampedro, C.; Rodriguez-Ramos, A.; Bavle, H.; Carrio, A.; de la Puente, P.; Campoy, P. A Fully-Autonomous Aerial Robot for Search and Rescue Applications in Indoor Environments Using Learning-Based Techniques. *J. Intell. Robot. Syst.* **2019**, *95*, 601–627. [\[CrossRef\]](#)
43. Mohta, K.; Watterson, M.; Mulgaonkar, Y.; Liu, S.; Qu, C.; Makineni, A.; Saulnier, K.; Sun, K.; Zhu, A.; Delmerico, J.; et al. Fast, Autonomous Flight in GPS-Denied and Cluttered Environments. *J. Field Robot.* **2018**, *35*, 101–120. [\[CrossRef\]](#)
44. Bachrach, A.; Prentice, S.; He, R.; Roy, N. RANGE—Robust Autonomous Navigation in GPS-Denied Environments. *J. Field Robot.* **2011**, *28*, 644–666. [\[CrossRef\]](#)
45. Leishman, R.C.; McLain, T.W.; Beard, R.W. Relative Navigation Approach for Vision-Based Aerial GPS-Denied Navigation. *J. Intell. Robot. Syst.* **2014**, *74*, 97–111. [\[CrossRef\]](#)
46. Li, Q.; Li, D.-C.; Wu, Q.; Tang, L.; Huo, Y.; Zhang, Y.; Cheng, N. Autonomous Navigation and Environment Modeling for MAVs in 3-D Enclosed Industrial Environments. *Comput. Ind.* **2013**, *64*, 1161–1177. [\[CrossRef\]](#)
47. Perez-Grau, F.J.; Ragel, R.; Caballero, F.; Viguria, A.; Ollero, A. An Architecture for Robust UAV Navigation in GPS-Denied Areas. *J. Field Robot.* **2018**, *35*, 121–145. [\[CrossRef\]](#)
48. Valenti, F.; Giaquinto, D.; Musto, L.; Zinelli, A.; Bertozzi, M.; Broggi, A. Enabling Computer Vision-Based Autonomous Navigation for Unmanned Aerial Vehicles in Cluttered GPS-Denied Environments. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 3886–3891.
49. Schmid, K.; Lutz, P.; Tomić, T.; Mair, E.; Hirschmüller, H. Autonomous Vision-Based Micro Air Vehicle for Indoor and Outdoor Navigation. *J. Field Robot.* **2014**, *31*, 537–570. [\[CrossRef\]](#)
50. Lutz, P.; Müller, M.G.; Maier, M.; Stoneman, S.; Tomić, T.; von Bargen, I.; Schuster, M.J.; Steidle, F.; Wedler, A.; Stürzl, W.; et al. ARDEA—An MAV with Skills for Future Planetary Missions. *J. Field Robot.* **2020**, *37*, 515–551. [\[CrossRef\]](#)
51. Lin, Y.; Gao, F.; Qin, T.; Gao, W.; Liu, T.; Wu, W.; Yang, Z.; Shen, S. Autonomous Aerial Navigation Using Monocular Visual-Inertial Fusion. *J. Field Robot.* **2018**, *35*, 23–51. [\[CrossRef\]](#)
52. Esrafilian, O.; Taghirad, H.D. Autonomous Flight and Obstacle Avoidance of a Quadrotor by Monocular SLAM. In Proceedings of the 2016 4th International Conference on Robotics and Mechatronics (ICROM), Tehran, Iran, 26–28 October 2016; pp. 240–245.
53. von Stumberg, L.; Usenko, V.; Engel, J.; Stücker, J.; Cremers, D. From Monocular SLAM to Autonomous Drone Exploration. In Proceedings of the 2017 European Conference on Mobile Robots (ECMR), Paris, France, 6–8 September 2017; pp. 1–8.
54. Laina, I.; Rupprecht, C.; Belagiannis, V.; Tombari, F.; Navab, N. Deeper Depth Prediction with Fully Convolutional Residual Networks. In Proceedings of the 2016 Fourth International Conference on 3D Vision (3DV), Stanford, CA, USA, 25–28 October 2016; pp. 239–248.
55. Bresenham, J.E. Algorithm for Computer Control of a Digital Plotter. *IBM Syst. J.* **1965**, *4*, 25–30. [\[CrossRef\]](#)
56. Hart, P.E.; Nilsson, N.J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [\[CrossRef\]](#)
57. Tordesillas, J.; Lopez, B.T.; Everett, M.; How, J.P. FASTER: Fast and Safe Trajectory Planner for Navigation in Unknown Environments. *IEEE Trans. Robot.* **2022**, *38*, 922–938. [\[CrossRef\]](#)
58. Sun, W.; Sun, P.; Ding, W.; Zhao, J.; Li, Y. Gradient-Based Autonomous Obstacle Avoidance Trajectory Planning for B-Spline UAVs. *Sci. Rep.* **2024**, *14*, 14458. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.