

DYNAMIC OBSTACLE AVOIDANCE FOR UAVS USING FUZZY LOGIC CONTROL AND NEURAL NETWORK

Ivan Marynych, Anton Makohonov

Kryvyi Rih National University, Department of Automation, Computer Science and Technology, Kryvyi Rih, Ukraine

Abstract. Unmanned Aerial Vehicles (UAVs) operating in dynamic environments require advanced navigation systems capable of avoiding moving obstacles in real time. In this paper, we introduce a fuzzy logic-based control algorithm designed to enhance UAV autonomy. Although many researches utilize data from different sensors, in this work we tried to operate only with on-board camera. Every frame is handled by a neural network for Object Detection. In this case the obstacle subject is another drone. For this purpose YOLO 10 was additionally trained and used. Detected obstacle is analysed by many parameters: distance, horizontal centre, vertical centre, vertical approaching, horizontal approaching. Following the computation, the derived parameters are input into the Mamdani fuzzy inference algorithm. As it interprets linguistic rules, we define fuzzy sets and apply them to generate an output based on degrees of membership. For the output, horizontal move and vertical move are chosen. The system was tested in a virtual environment. It utilizes the Unity framework, in which a simple scene was created. As an obstacle the model of the drone was used. A camera acts as the unmanned aerial vehicle (UAV). Frames are sent to a python script on a local machine. It helps to simulate the structure of a system on edge device. And output parameters are delivered back to correct the path, if the controlled UAV has to avoid collision.

Keywords: UAV, fuzzy logic, collision avoidance

DYNAMICZNE OMIJANIE PRZESZKÓD PRZEZ BEZPILOTOWE STATKI POWIETRZNE Z WYKORZYSTANIEM STEROWANIA OPARTEGO NA LOGICE ROZMYTEJ I SIECI NEURONOWEJ

Streszczenie. Bezzałogowe statki powietrzne (UAV) działające w dynamicznych środowiskach wymagają zaawansowanych systemów nawigacyjnych zdolnych do omijania ruchomych przeszkód w czasie rzeczywistym. W niniejszym artykule przedstawiamy algorytm sterowania oparty na logice rozmytej, zaprojektowany w celu zwiększenia autonomii UAV. Chociaż w wielu badaniach wykorzystuje się dane z różnych czujników, w niniejszej pracy postanowiliśmy posługiwać się wyłącznie kamerą pokładową. Każda klatka jest przetwarzana przez sieć neuronową w celu wykrycia obiektów. W tym przypadku przeszkodą jest inny dron. W tym celu dodatkowo przeszkolono i wykorzystano model YOLO 10. Wykryta przeszkoda jest analizowana pod kątem wielu parametrów: odległości, środka poziomego, środka pionowego, zbliżania się w pionie oraz zbliżania się w poziomie. Po zakończeniu obliczeń uzyskane parametry są wprowadzane do algorytmu wnioskowania rozmytego Mamdaniego. Ponieważ algorytm ten interpretuje reguły lingwistyczne, definiujemy zbiory rozmyte i stosujemy je w celu wygenerowania wyniku opartego na stopniach przynależności. Jako wyniki wybrano ruch poziomy i ruch pionowy. System został przetestowany w środowisku wirtualnym. Wykorzystuje on framework Unity, w którym utworzono prostą scenę. Jako przeszkodę wykorzystano model drona. Rolę sterowanego bezzałogowego statku powietrznego (UAV) pełni kamera. Klatki są wysyłane do skryptu w języku Python na komputerze lokalnym. Pomaga to w symulacji struktury systemu na urządzeniu brzegowym. Parametry wyjściowe są następnie przekazywane z powrotem w celu skorygowania trajektorii, jeśli sterowany UAV musi uniknąć kolizji.

Słowa kluczowe: UAV, logika rozmyta, unikanie przeszkód

Introduction

As UAVs become increasingly common in civilian and industrial applications, including sectors such as healthcare and health-related services [7], infrastructure [4, 16], precision agriculture [5], delivery [3] and others. The need for flexible and intelligent collision avoidance grows. When obstacle locations are known beforehand, the trajectory can be carefully planned to ensure safe navigation. Nevertheless, in practical applications, you can meet with unexpected objects during flight, which can make the original flight plan potentially misleading or unsafe without dynamic path correction.

An effective path is typically evaluated through key performance indicators (KPIs) such as path length, the number of turns, and main factors like energy consumption and mission completion time [8]. It is not only important to reduce these KPIs in order to improve efficiency, but also to ensure that the UAV's safety remains paramount [15].

Path planning becomes even more challenging when accounting for dynamic environments with obstacles and non-flying zones (NFZs). NFZs refer to restricted airspace where UAVs are not permitted to fly. A primary challenge lies in developing optimal paths that comprehensively cover an area of interest (AoI) while avoiding obstacles and NFZs and minimizing energy costs or operational delays [15].

This article focused on the creation of a system that can recognize new dynamic objects in AoI, estimate their path, and if it can lead to collision, the system has to form commands for avoidance. While many researchers use data from a lot of sensors, we think that it increases the cost of manufacturing and energy consumption. It is especially important for small aerial vehicles. That's why study utilises just information from on-board camera and the second physical component is computational

resource. It can be microcomputers, such as Raspberry PI with additional blocks, NVIDIA Jetson, etc. Because every intelligent system has to be served by some instrument. We can not use cloud GPU, because connection between drone and server can be absent in critical moments of decision making. The second reason is ping latency. That's why it has to be an edge device. A camera is also a necessary thing. It is necessary for different tasks. For example for logging moving, to improve algorithms in future or debug bad actions. Therefore, in such a system model we do not use any additional details.

1. Literature overview

To solve this task there are different ways. In this section we introduce a short brief of approaches used by researchers.

Often researchers improve path planning algorithms, such as A* [6, 11, 12], RRT [22], RRT* [9, 10] or use them as the first step [20]. Despite their advantages, each of these methods has notable limitations. When applied to dynamic environments, grid-based methods can be computationally intensive, for example, A* requires recalculating the full path for local obstacles. Sampling-based methods (e. g. RRT, RRT*), although efficient in the open environment, may struggle with low success rates and maintaining path optimality in complex environments. In particular, in the narrow channel and trap obstacle environments, the number of nodes expanded by Informed RRT* and BIT* increases significantly, reflecting the challenges these algorithms face in finding feasible paths while ensuring optimality within confined or intricate spaces [24].

Another group of methods is velocity fields, which guide the UAV by assigning preferred movement directions throughout the environment. For instance, article [17] introduces the approach of an enhanced velocity field that incorporates hydrodynamic

theory tools. Initially designed for the 2D case, the method is subsequently extended to the 3D scenario by introducing vector field extensions and rotations. To validate the proposed scheme, experiments are conducted using a UAV model tasked with tracking a circular contour. The control system employs two PD controllers for regulating the vertical position and yaw angle, while the roll and pitch angles are controlled using a nested saturation method.

In research [13], deep reinforcement learning strategies that combine the Yolov8-StrongSort model with a new DQN data storage method were presented. Considering the negative impact of dynamic obstacles, the reward function is set up using an artificial potential field to circumvent obstacles and approach the target so that the UAV can obtain the optimal path. Also a data storage mechanism that alters the distribution of the experience types in the replay memory, thereby accelerating the convergence of the proposed method was applied. Based on the scenarios, the Yolov8-StrongSort model considers dynamic obstacle trajectories and the background of the image output from the model, thereby reducing the difficulty of training the UAV. For efficient target tracking, OSNet was chosen, a lightweight yet powerful network architecture with exceptional target reidentification capabilities, ensuring stable performance even in complex environments [13]. However, reinforcement-learning approaches suffer from unstable training and are sensitive to reward functions [17].

In [23] an interesting approach that utilizes an RGB-D camera was introduced. The proposed system uses a 3D hybrid map that adopts an occupancy voxel map for static environment representation and tracks dynamic obstacles using their bounding boxes with velocity information. The method applies the depth map-based detector to obtain 3D region proposals for dynamic obstacles and refine the bounding boxes' sizes by our occupancy map refinement approach. Then, Kalman filter and continuity filter to identify and track dynamic obstacles is applied. Next, the proposed dynamic-region cleaning method is used to free the moving obstacle areas in the static map. Besides, the Markov chain-based trajectory prediction method is developed for dynamic obstacles, considering the interaction between dynamic obstacles and static environments [23].

Another promising direction in UAV path planning and obstacle avoidance is the use of fuzzy logic systems, which have attracted increasing attention due to their ability to handle imprecise information and nonlinear dynamics. For example, [1] introduces a Probability Fuzzy Logic (PFL) system that utilises probability-based impact assessment and fuzzy logic rules to deal with unknowns and environmental changes. The fuzzy logic system takes in input about the distance of objects from the drone's front, left, and right sides, as well as the probability of collision based on the drone's speed and how close it is to the obstacles. The set of thirty fuzzy rules based on the distance of the obstacle from front left and right are defined to decide the output, i.e. speed of the drone and heading angle. The simulation environment is MATLAB, with grid-based motion planning that accounts for static and dynamic obstacles. The system's performance is validated through simulations and real-world experiments, comparing path length and travel time. On comparing the simulation and experimental results, the proposed PFL-based controller has been proven to be efficient, accurate, and robust for both static and dynamic and simple to complex environments. The drones can plan the shortest and most collision-free path across all the scenarios, as depicted in the simulation and experimentation results. However, due to communication delay, inaccuracy of sensor response, environmental impact and motor delay, there are slight deviations between the simulation and experimentation values. Upon performing the error analysis, it is found that the error between the simulation and experimental value is within the range of 6.66% in all the studied scenarios [1].

Also fuzzy logic can be combined with other algorithms, such as Deep Reinforcement Learning [21] or velocity [2].

2. Main research

The UAV manoeuvre control system consists of 3 main elements:

- 1) Neural networks for object detection.
- 2) System for determining parameters necessary for decision-making.
- 3) Decision-making system.

YOLO10n, which was additionally trained [14], was chosen as a neural network for testing. Also, to speed up the work on the local machine, tests were conducted using the ONNX [19] and OpenVINO [10] frameworks. The results are given in Table 1, based on processing frames of an already saved video on an Apple M2 processor and 8 GB of RAM.

Table 1. Frame rate speed comparison

Variations of the model	Without framework	ONNX	OpenVINO
Time(ms)	80	40	30

The following parameters for input were chosen:

- 'distance' – relative distance to the object,
- 'horizontal_offset' – horizontal distance,
- 'vertical_offset' – vertical distance,
- 'vertical_approaching' – a relative indicator that gives an understanding of whether another object is approaching or receding vertically,
- 'horizontal_approaching' – a relative indicator that gives an understanding of whether another object is approaching or receding horizontally,
- 'altitude' – flight altitude above the surface. Obtained from on-board sensors. In the context of this work, it is constantly defined.

For distance, we use a heuristic approach – we estimate how large the object (bounding box) is compared to the frame size, and based on this, we roughly calculate the distance to the object – the smaller it is on the screen, the further it is (and vice versa):

$$bbox_ratios = \sum_1^{len(bbox_history)} \frac{bbox[width] \times bbox[height]}{width \times height} \quad (1)$$

where *bbox_history* is an array with last detection produced by a neural network, limited to 10 values. *bbox['width']* and *bbox['height']* are sizes of each bounding box (object frame). Width and height are sizes of a given image.

And we calculate the average value of the *bbox* area for noise reduction:

$$average_ratio = \frac{1}{n} \sum_{i=1}^n bbox_ratios_i \quad (2)$$

We add constraints to normalize the values to the range [0,1]:

$$normalized_ratio = \frac{\min(average_ratio, max_ratio)}{max_ratio} \quad (3)$$

where *max_ratio* is 0.1 for preventing the impact of outliers.

The last step is to scale it to 100 units. The closer the value is to 100, the further away the object is. The smaller the value is, the closer the object is:

$$current_distance = (1 - normalized_ratio) \times 100 \quad (4)$$

To provide spatial interpretation of the object position in the field of view of the computer vision system, the *calculate_signed_centering_offset* function was implemented, which calculates the normalized offset of the object centre (bounding box) relative to the frame centre. By passing different sets of parameters, the *horizontal_offset* and *vertical_offset* values are obtained. The function accepts the following parameters:

- 1) *bbox_center* – coordinate of the object centre (horizontally or vertically),
- 2) *frame_center* – frame centre coordinate,
- 3) *frame_dimension* – the corresponding frame size (width or height).

The displacement value is calculated as:

$$distance = bbox_center - frame_center \quad (5)$$

$$max_distance = \frac{frame_dimension}{2} \quad (6)$$

$$\text{normalized_offset} = \frac{\text{max_distance}}{\text{distance}} \quad (7)$$

To maintain stability and limit the value to a predictable range, the normalized bias is kept within:

$$\text{normalized_offset} \in [-1.0, 1.0] \quad (8)$$

This allows us to interpret the offset as a percentage deviation from the centre of the frame:

- a value of 0 corresponds to a perfectly centred object,
- values of -1 and 1 correspond to the extreme left/right (or top/bottom) positions within the frame.

Determining the tendency of an unmanned aerial vehicle (drone) approach to a given centre of the frame along the vertical axis, is based on the analysis of the history of its positions. For this purpose the *is_vertical_approaching* function was developed, which accepts the following parameters:

- 1) *center_history* – list of drone centre coordinates for previous frames,
- 2) *frame_center_y* – vertical coordinate of the frame centre,
- 3) *sensitivity* – algorithm sensitivity coefficient (default 0.1).

The function implements a statistical approach to analysing the drone's movement trend. If the number of records in the history is less than 5, a neutral value of 0.5 is returned, indicating insufficient data for analysis. The analysis uses the last 5 positions of the drone from the history, which ensures the relevance of the assessment and reduces the impact of outdated data.

For each of the last positions, the absolute vertical distance from the centre of the frame is calculated using the formula:

$$\text{vertical_distance} = |\text{center_y} - \text{frame_center_y}| \quad (9)$$

Next, the least squares method is used to determine the slope of the trend line of vertical distances over time. To do this, we form an array of *x* using the code:

$$n = \text{len}(\text{vertical_distances}) \quad (10)$$

The arithmetic mean of the coordinates is obtained using the formulas:

$$x_mean = \frac{\sum(x)}{n} \quad (11)$$

$$y_mean = \frac{\sum(\text{vertical_distances})}{n} \quad (12)$$

And we directly calculate the slope of the regression line:

$$\text{slope} = \frac{\sum_{i=1}^n (x_i - x_mean) \times (y_i - y_mean)}{\sum_{i=1}^n (x_i - x_mean)^2} \quad (13)$$

In the case where the denominator is zero (all points have the same *y*-coordinate), the function returns the neutral value 0.5. The resulting slope is transformed using the sigmoid function:

$$\text{sigmoid} = \frac{1}{1 + e^{\text{slope} \times \text{sensitivity}}} \quad (14)$$

Using the sigmoid function provides:

- smooth display of a linear trend in a limited range [0, 1],
- nonlinear sensitivity to slope changes,
- stability of operation at extreme slope values.

The sensitivity parameter allows you to adjust the steepness of the sigmoidal function, affecting the sensitivity of the system to changes in the drone's movement tendency. The function returns a normalized value in the range [0, 1]:

- close to 0 – the drone is moving away from the vertical centre (positive slope),
- close to 0.5 – there is no clear trend or not enough data,
- close to 1 – the drone is moving towards the vertical centre (negative slope).

The *horizontal_approaching* parameter is calculated using a similar algorithm, except that the *vertical_distance* parameter is calculated in the same way. The formula (9) is as follows:

$$\text{vertical_distance} = |\text{center_x} - \text{frame_center_x}| \quad (15)$$

where *center_x* is the centre of the moving object horizontally.

Interpretation of the results is as follows:

- close to 0 – the drone is moving away from the centre horizontally (positive slope),
- close to 0.5 – there is no clear trend or not enough data,
- close to 1 – the drone is moving towards the centre horizontally (negative slope).

To determine the need to change the movement along the *X* or *Y* axis based on the data obtained from the previous subsystem, the Mamdani fuzzy logic algorithm was used. To start using it, it is necessary to distribute the terms over the input parameters according to the functions. In our case, we use triangular and trapezoidal membership functions.

The distribution term by parameters is as follows:

1. *distance*:
 - 'far': 93, 96, 100, 100 (trapezoidal),
 - 'close': 85, 93, 95 (triangular),
 - 'very_close': 60, 75, 90 (triangular),
 - 'critically_close': 0, 0, 60, 65 (trapezoidal),
2. *horizontal_centering*:
 - 'left': -1, -1, -0.3, 0 (trapezoidal),
 - 'center': -0.3, 0, 0.3 (triangular),
 - 'right': 0, 0.3, 1, 1 (trapezoidal),
3. *vertical_centering*:
 - 'low': -1, -1, -0.3, 0 (trapezoidal),
 - 'center': -0.3, 0, 0.3 (triangular),
 - 'high': 0, 0.3, 1, 1 (trapezoidal),
4. *vertical_approaching*:
 - 'moving_away': 0, 0, 0.35, 0.45 (trapezoidal),
 - 'stable': 0.43, 0.505, 0.58 (triangular),
 - 'approaching_weak': 0.55, 0.625, 0.7 (triangular),
 - 'approaching_strong': 0.7, 0.85, 1, 1 (trapezoidal),
5. *horizontal_approaching*:
 - 'moving_away': 0, 0, 0.35, 0.45 (trapezoidal),
 - 'stable': 0.43, 0.505, 0.58 (triangular),
 - 'approaching_weak': 0.55, 0.625, 0.7 (triangular),
 - 'approaching_strong': 0.7, 0.85, 1, 1 (trapezoidal).

The following group of variable terms are formed for the output parameters:

1. *horizontal_move*:
 - 'left': -1, -1, -0.5, 0 (trapezoidal),
 - 'no_change': -0.3, 0, 0.3 (triangular),
 - 'right': 0, 0.5, 1, 1 (trapezoidal),
2. *vertical_move*:
 - 'down': -1, -1, -0.5, 0 (trapezoidal),
 - 'no_change': -0.3, 0, 0.3 (triangular),
 - 'up': 0, 0.5, 1, 1 (trapezoidal).

The trapezoidal shape provides a plateau-like region of maximum membership ($\mu = 1$), for example for distances from 96 to 100. This means that all distances above 96 units are considered equally 'far', i.e. when the drone is very far away, the exact distance value is less critical for decision-making, which achieves additional stability of the system.

Figures 1 and 2 show how it looks graphically.

At the same time Triangular functions provide a gradient transition of membership and more accurately reflect the transition states between different levels of proximity.

After defining all the terms, the next step is to define the rules by which they will operate. Since there are so many variations, only a few of them will be listed:

- if *distance* is *far*, *horizontal_approaching* is *stable* and *horizontal_centering* is *left*, then *horizontal_move* is *no_change* and *vertical_move* is *no_change*,
- if *distance* is *close*, *horizontal_approaching* is *stable*, *vertical_centering* is *center* and *horizontal_centering* is *center*, then *horizontal_move* is *left* and *vertical_move* is *no_change*
- if *distance* is *close* and *horizontal_approaching* is *moving_away*, then *horizontal_move* is *no_change* and *vertical_move* is *no_change*,
- if *distance* is *far*, *horizontal_approaching* is *approaching_weak*, *vertical_centering* is *center* and *horizontal_centering* is *left*, then *horizontal_move* is *no_change* and *vertical_move* is *up*,
- if *distance* is *far*, *horizontal_approaching* is *approaching_weak*, *vertical_centering* is *center*

and *horizontal_centering* is *right*, than *horizontal_move* is *no_change* and *vertical_move* is *up*,

- if *distance* is *very_close*, *vertical_approaching* is *approaching_strong*, *vertical_centering* is *low* and *horizontal_centering* is *left*, than *horizontal_move* is *left* and *vertical_move* is *no_change*,
- if *distance* is *far*, *vertical_approaching* is *stable*, *vertical_centering* is *center* and *horizontal_centering* is *center*, than *horizontal_move* is *no_change* and *vertical_move* is *up*,
- if *distance* is *very_close*, *vertical_approaching* is *approaching_weak*, *vertical_centering* is *low* and *horizontal_centering* is *center*, than *horizontal_move* is *left* and *vertical_move* is *no_change*,
- if *distance* is *very_close*, *vertical_approaching* is *approaching_weak*, *vertical_centering* is *high* and *horizontal_centering* is *center*, than *horizontal_move* is *left* and *vertical_move* is *no_change*,
- if *distance* is *very_close*, *vertical_approaching* is *approaching_strong*, *vertical_centering* is *high* and *horizontal_centering* is *center*, than *horizontal_move* is *left* and *vertical_move* is *no_change*,
- if *distance* is *far*, *horizontal_approaching* is *stable* and *horizontal_centering* is *right*, than *horizontal_move* is *left* and *vertical_move* is *no_change*.

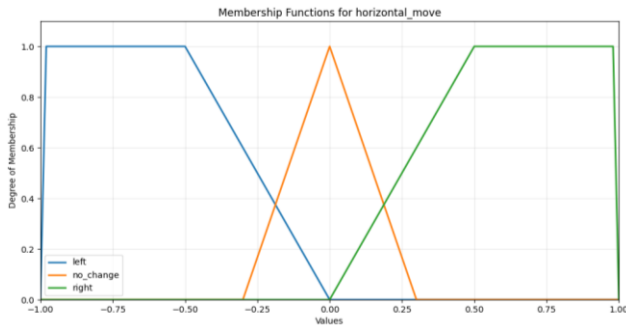


Fig. 1. Membership function for *horizontal_move*

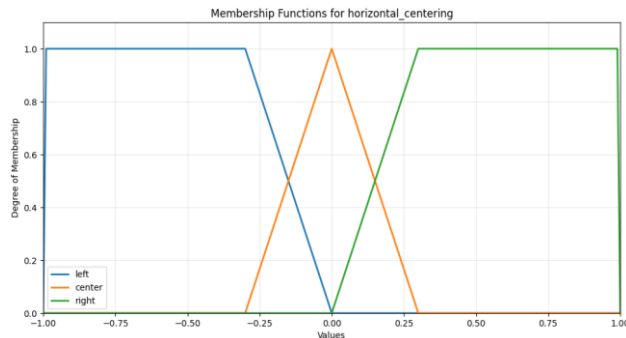


Fig. 2. Membership function for *horizontal_centering*

3. Simulation environment

To test the system, the multiplatform game engine Unity was chosen. An environment was created in the system (Figure 3) and a drone model was placed (Figure 4).

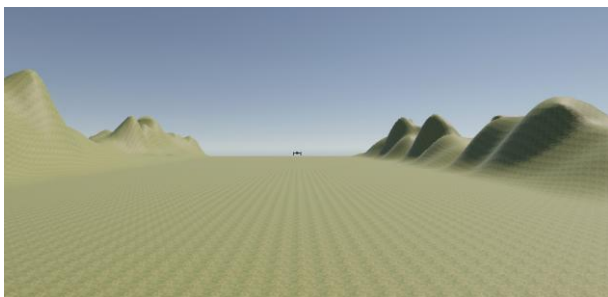


Fig. 3. Environment for testing



Fig. 4. Drone model for testing

A camera object was also defined, simulating a drone controlled by the system. Each frame is sent to a python server, which is also located on the local machine. In order for the server to be able to process the data in time (testing was carried out on insufficient capacities), a deceleration factor of 0.5 was introduced. After processing according to the specified algorithms, the server returns the values of *vertical_move* and *horizontal_move*. In unity, these values are used to correct the drone's movement.

4. Simulation results

We tested different situations in which one drone is under our control and the second is flying on some trajectory. For example, forward to ours, approaching from left and right, parallel moving on the right side and up. For a detailed view we present the test scenario in which the drone moves in a straight line without changing the positions of *X* and *Y*. The second drone should move in the opposite direction, without changing the positions along the *X* and *Y* axes. The location of the other object is the centre of the screen. Thus, a direct collision is simulated. The results of frame processing were obtained (only a part is given), which are shown in Table 2.

Table 2. Data obtained as a result of frame processing

Frame number	Distance	Horizontal offset	Vertical offset	Vertical approaching	Horizontal approaching
12	99.5	0.008	0.035	0.495	0.488
13	99.5	0.016	0.035	0.493	0.483
14	99.5	0.016	0.035	0.493	0.48
15	99.5	0.016	0.035	0.495	0.485
16	99.5	0.02	0.035	0.5	0.485
17	99.5	0.023	0.035	0.5	0.488
18	99.5	0.027	0.035	0.5	0.48
19	99.5	0.027	0.035	0.5	0.48

The data obtained after handling parameters from Table 2 by the Mamdani algorithm are given in Table 3.

Table 3. Output parameters

Frame number	Vertical move	Horizontal move
12	9.519823953531259e-18	-0.415255843354391
13	1.2415012218529244e-17	-0.4069044414535665
14	1.5352302762394745e-17	-0.40191713483146063
15	1.120515163908464e-17	-0.41003603013429407
16	1.120515163908464e-17	-0.41003603013429407
17	9.519823953531259e-18	-0.415255843354391
18	1.5352302762394745e-17	-0.40191713483146063
19	1.5352302762394745e-17	-0.40191713483146063
20	1.2415012218529244e-17	-0.4069044414535665
21	1.2415012218529244e-17	-0.4069044414535665
22	1.5352302762394745e-17	-0.40191713483146063
23	1.5352302762394745e-17	-0.40191713483146063
24	1.56976510862932e-17	-0.39894927536231883
25	1.5695117163599686e-17	-0.39420952380952384
26	1.56976510862932e-17	-0.39894927536231883

As can be seen from the data in Table 3, a very small movement vertically upwards is detected, which is ignored. And there is a clear signal for movement to the left horizontally, which decreases with the change of frames, because the drone obstacle moves in the centre, without changing its position.

And the drone, which is under the control of the system, moves to the left, led by signals. This example shows that the system is workable and ready for further development.

To compare we take into account 4 methods: A*, improved A* from [11], artificial potential field and RRT*. We used the following approach:

- 1) to calculate the path from start to goal,
- 2) to take the next point,
- 3) to calculate the path from this point to goal, taking into account the new position of obstacle.

Points 2 and 3 repeated in cycle, until the goal is reached. For the test, we simulate the same situation, where the drone moves in a straight line without changing the positions of X and Y. The results are demonstrated in Table 4. Trajectory deviation shows maximum distance between the drone and the straight trajectory during the manoeuvre, and additional path means how much longer the trajectory is in comparison with a straight line.

Table 4. The results

	Our method	A*	A* improved	APF	RRT*
Additional path length	21.22	0	0	-	9.02
Trajectory deviation	22.15	0	0	-	3.74
Goal reached	Yes	Yes	Yes	No	Yes
Start maneuver at point	15.58	-	-	-	104.38
Obstacle avoided	Yes	No	No	-	Yes

The A* algorithm cannot avoid moving obstacle. In our example, the intersection point is at position 113. In this case, if the obstacle is at (115, 150), then the path is generated as follows [(111, 150), (112, 150), (113, 150), (114, 150), **(114, 149)**, **(115, 149)**, **(116, 149)**, (116, 150), (117, 150), (118, 150), (119, 150), (120, 150), (121, 150), (122, 150), (123, 150), (124, 150), (125, 150), (126, 150), (127, 150), (128, 150), (129, 150), (130, 150), (131, 150)]. If at (114, 150), then the path is [(112, 150), (113, 150), **(113, 149)**, **(114, 149)**, **(115, 149)**, (115, 150), (116, 150), (117, 150), (118, 150), (119, 150), (120, 150), (121, 150), (122, 150), (123, 150), (124, 150), (125, 150), (126, 150), (127, 150), (128, 150), (129, 150), (130, 150), (131, 150), (132, 150)]. And accordingly, at (113, 150), 113 is the starting point, because the path is [(**113, 150**), (114, 150), (115, 150), (116, 150), (117, 150), (118, 150), (119, 150), (120, 150), (121, 150), (122, 150), (123, 150), (124, 150), (125, 150), (126, 150), (127, 150), (128, 150), (129, 150), (130, 150), (131, 150), (132, 150)]. Thus, the manoeuvre for avoiding the collision either starts too late or does not start at all.

Improved A* algorithm shows the same. If the obstacle is at (115, 150), then the path is generated as follows [(111, 150), (112, 150), (113, 150), (114, 150), **(114, 149)**, **(115, 149)**, **(116, 149)**, (116, 150), (117, 150), (118, 150), (119, 150), (120, 150), (121, 150), (122, 150), (123, 150), (124, 150), (125, 150), (126, 150), (127, 150), (128, 150), (129, 150), (130, 150), (131, 150), (132, 150)]. If at (114, 150), then the path is [(112, 150), (113, 150), **(113, 149)**, **(114, 149)**, **(115, 149)**, (115, 150), (116, 150), (117, 150), (118, 150), (119, 150), (120, 150), (121, 150), (122, 150), (123, 150), (124, 150), (125, 150), (126, 150), (127, 150), (128, 150), (129, 150), (130, 150), (131, 150), (132, 150)]. And accordingly at (113, 150), 113 is the starting point, because the path is [(**113, 150**), (114, 150), (115, 150), (116, 150), (117, 150), (118, 150), (119, 150), (120, 150), (121, 150), (122, 150), (123, 150), (124, 150), (125, 150), (126, 150), (127, 150), (128, 150), (129, 150), (130, 150), (131, 150), (132, 150), (133, 150)]. So in this case collision occurs.

Artificial potential did not achieve the goal, because when the obstacle is closer, it generates strong repulsion force, and APF chooses the easier way – to go back.

RRT* avoids the obstacle and reaches the goal. But it does not have a very smooth path (Figure 5).

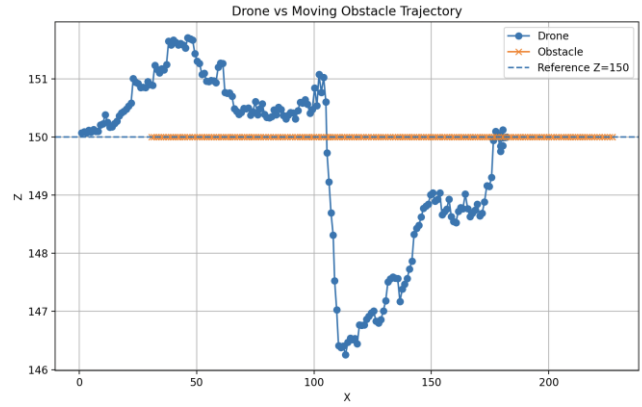


Fig. 5. Drone path generated with the help of RRT* algorithm

For drones it is hard to fly with such a trajectory. The trajectory produced by our method looks more natural (Figure 6).

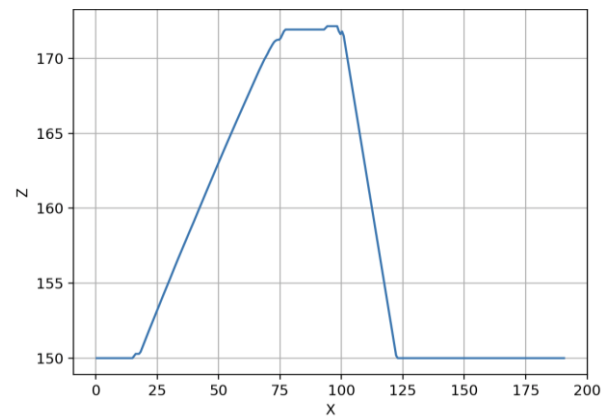


Fig. 6. Drone path generated with the help of our method algorithm

But the most significant disadvantage of these methods is that we have to know the exact point of the obstacle for the manoeuvre, which is impossible without additional equipment. Another important parameter is the radius of avoidance. In most real-world scenarios, selecting an optimal radius is non-trivial, as it depends on multiple uncertain factors such as obstacle motion, sensor accuracy, and system dynamics. An improperly chosen radius may either lead to unsafe paths or unnecessary maneuvers. In our system avoidance radius is not applied directly. The apparent size of the object in the image is used as an indicator of its relative distance and physical dimensions. Based on this data, a conclusion is made about the beginning and end of the manoeuvre. This allows the system to dynamically adjust the avoidance radius in real time, making the path planning process more flexible and context-aware.

5. Conclusion

The article introduces a system that consists of 3 main parts: neural network for object detection, system for determining parameters necessary for decision-making and decision-making system. This structure allows not to use additional equipment, such as LiDAR. And only use necessary components for every intelligent control system, namely a camera and computational boards. It reduces manufacturing costs and energy consumption, which is critically important for small and medium unmanned aerial vehicles.

As a neural network we took YOLO10n, additionally trained. Tests show that it can be significantly improved with applying ONNX or OpenVino frameworks. And we hope, modern microcomputers with GPU or NTU blocks can provide real-time work. The list of input parameters and approaches to calculate them,

fuzzy terms and rules were developed. In the virtual testing environment the system shows good results in avoiding collision with another drone.

Comparison with some other path-planning algorithms shows advantages of our approach, despite the long distance necessary for avoiding an obstacle. The path of the manoeuvre is smooth and natural, which is particularly important for drones, because their motion is constrained by inertia and dynamic limitations. As a result, drones cannot accurately follow trajectories with sharp turns or abrupt changes in direction.

Of course, the list of parameters has to be grown. It has to include altitude control, battery level control. And for example, if the battery charge is low, we have to think about energy saving manoeuvres. The next very important parameter in the real world is wind direction. Including this parameter can help make a decision about manoeuvre to the right or left side, decreasing general energy consumption.

References

- [1] Agrawal, S., Patle, B. K., & Sanap, S. (2025). Navigation control of unmanned aerial vehicles in dynamic collaborative indoor environment using probability fuzzy logic approach. *Cognitive Robotics*, 5, 86–113. <https://doi.org/10.1016/j.cogr.2025.02.002>
- [2] Candan, F., Beke, A., Mahfouf, M., & Mihaylova, L. (2024). A Real-time Fuzzy Interacting Multiple-Model Velocity Obstacle Avoidance Approach for Unmanned Aerial Vehicles. *Journal of Intelligent & Robotic Systems*, 110(2), 61. <https://doi.org/10.1007/s10846-024-02075-6>
- [3] Chi, N. T. K., Phong, L. T., & Hanh, N. T. (2023). The drone delivery services: An innovative application in an emerging economy. *The Asian Journal of Shipping and Logistics*, 39(2), 39–45. <https://doi.org/10.1016/j.ajsl.2023.01.002>
- [4] Fan, J. & M. Ala Saadeghvaziri. (2019). *Applications of Drones in Infrastructures: Challenges and Opportunities*. <https://doi.org/10.5281/ZENODO.3566281>
- [5] Guebsi, R., Mami, S., & Chokmani, K. (2024). Drones in Precision Agriculture: A Comprehensive Review of Applications, Technologies, and Challenges. *Drones*, 8(11), 686. <https://doi.org/10.3390/drones8110686>
- [6] Haidar Ahmad, A., Zahwe, O., Nasser, A., & Clement, B. (2024). Path Planning for Unmanned Aerial Vehicles in Dynamic Environments: A Novel Approach Using Improved A* and Grey Wolf Optimizer. *World Electric Vehicle Journal*, 15(11), 531. <https://doi.org/10.3390/wevj15110531>
- [7] Hiebert, B., Nouvet, E., Jeyabalan, V., & Donelle, L. (2020). The Application of Drones in Healthcare and Health-Related Services in North America: A Scoping Review. *Drones*, 4(3), 30. <https://doi.org/10.3390/drones4030030>
- [8] Huang, H., & Savkin, A. V. (2018). Towards the Internet of Flying Robots: A Survey. *Sensors*, 18(11), 4038. <https://doi.org/10.3390/s18114038>
- [9] Huang, Z., Jiang, C., Shen, C., Liu, B., Huang, T., & Zhang, M. (2025). A Hybrid Dynamic Path-Planning Method for Obstacle Avoidance in Unmanned Aerial Vehicle-Based Power Inspection. *World Electric Vehicle Journal*, 16(1), 22. <https://doi.org/10.3390/wevj16010022>
- [10] Jain, K., & Kashyap, A. (2026). Integrated RRT* and Dynamic Window Approach for UAV Obstacle Control. *Aerotecnica Missili & Spazio*, 105(2), 183–196. <https://doi.org/10.1007/s42496-025-00273-5>
- [11] Ju, C., Luo, Q., & Yan, X. (2020). Path Planning Using an Improved A-star Algorithm. *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*, 23–26. <https://doi.org/10.1109/PHM-Jinan48558.2020.00012>
- [12] Li, J., Kang, F., Chen, C., Tong, S., Jia, Y., Zhang, C., & Wang, Y. (2023). The Improved A* Algorithm for Quadrotor UAVs under Forest Obstacle Avoidance Path Planning. *Applied Sciences*, 13(7), 4290. <https://doi.org/10.3390/app13074290>
- [13] Liu, J., Luo, W., Zhang, G., & Li, R. (2025). Unmanned Aerial Vehicle Path Planning in Complex Dynamic Environments Based on Deep Reinforcement Learning. *Machines*, 13(2), 162. <https://doi.org/10.3390/machines13020162>
- [14] Marynych, I. A., & Makohonov, A. V. (2024). Detection of UAVs using machine vision technologies. *Scientific Notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences*, 1(5), 188–196. <https://doi.org/10.32782/2663-5941/2024.5.1/28>
- [15] Merei, A., Mcheick, H., Ghaddar, A., & Rebaine, D. (2025). A Survey on Obstacle Detection and Avoidance Methods for UAVs. *Drones*, 9(3), 203. <https://doi.org/10.3390/drones9030203>
- [16] Nwaogu, J. M., Yang, Y., Chan, A. P. C., & Chi, H. (2023). Application of drones in the architecture, engineering, and construction (AEC) industry. *Automation in Construction*, 150, 104827. <https://doi.org/10.1016/j.autcon.2023.104827>
- [17] Rascón, J., Toledo, B., Di Gennaro, S., & García, L. A. (2024). An Algorithm for Dynamic Obstacle Avoidance Applied to Uavs. *SSRN*. <https://doi.org/10.2139/ssrn.4690925>
- [18] Ultralytics. (2023). *Intel OpenVINO export-Ultralytics YOLO docs*. <https://docs.ultralytics.com/integrations/openvino/>
- [19] Ultralytics. (2024). *ONNX export for YOLO11 models-Ultralytics YOLO docs*. <https://docs.ultralytics.com/integrations/onnx/>
- [20] Wei, J. (2025). *Real-Time Obstacle Avoidance Algorithms for Unmanned Aerial and Ground Vehicles (Version 1)*. arXiv. <https://doi.org/10.48550/ARXIV.2506.20311>
- [21] Xia, B., Mantegh, I., & Xie, W.-F. (2024). Hybrid Framework for UAV Motion Planning and Obstacle Avoidance: Integrating Deep Reinforcement Learning with Fuzzy Logic. *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2662–2669. <https://doi.org/10.1109/CoDIT62066.2024.10708494>
- [22] Xiong, J., & Duan, X. (2021). Path planning for UAV based on improved dynamic step RRT algorithm. *Journal of Physics: Conference Series*, 1983(1), 012034. <https://doi.org/10.1088/1742-6596/1983/1/012034>
- [23] Xu, Z., Zhan, X., Chen, B., Xiu, Y., Yang, C., & Shimada, K. (2022). A real-time dynamic obstacle tracking and mapping system for UAV navigation and collision avoidance with an RGB-D camera. <https://doi.org/10.48550/ARXIV.2209.08258>
- [24] Zhang, J., Xian, Y., Zhu, X., & Deng, H. (2025). A Hybrid Deep Learning Model for UAV Path Planning in Dynamic Environments. *IEEE Access*, 13, 67459–67475. <https://doi.org/10.1109/ACCESS.2025.3557394>

Ph.D. Ivan Marynych

e-mail: marynych@knu.edu.ua

Associate professor of the Department of Automation, Computer Science and Technology of Kryvyi Rih National University, Kryvyi Rih, Ukraine.
Research interest: modelling of technological processes and control systems; modern methods of control theory; optimal control of technological processes and computer-integrated production.

<https://orcid.org/0000-0002-9036-8532>

M.Sc. Anton Makohonov

e-mail: antonmakogonov5@gmail.com

Ph.D. student of the Department of Automation, Computer Science and Technology of Kryvyi Rih National University, Kryvyi Rih, Ukraine.
Research interest: UAV detection, path planning and collision avoidance.

<https://orcid.org/0009-0008-1761-5593>

