Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних наук систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол №___ від ___ червня 2026 р.
завідувач кафедри _____ Максим ХРУСЛОВ
«____»_____ *2026 р.*

Кваліфікаційна робота

здобувача першого (бакалаврського) рівня вищої освіти

«МОДЕЛЬ ПЛАНУВАННЯ ТРАЄКТОРІЇ РУХУ БПЛА НА ОСНОВІ ДАНИХ ВІДЕОСПОСТЕРЕЖЕНЬ»

Спеціальність ***123 – Комп'ютерна інженерія***
Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ Владислав БЄЛОУС

Науковий керівник _____ Вікторія СТРІЛЕЦЬ

Харків – 2026

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, переліку використаних джерел та чотирьох додатків. Загальний обсяг роботи становить 80 сторінок, з яких 49 сторінок основної частини, що містять 17 рисунків, 1 таблицю, 26 найменувань у списку використаних джерел та 4 додатки.

Метою кваліфікаційної роботи є покращення адаптивності планування траєкторії руху БПЛА на основі даних відеоспостережень, і забезпечення адаптивного коригування маршруту в реальному часі залежно від виявлених перешкод та умов середовища.

Об'єктом дослідження є процес планування траєкторії руху безпілотного літального апарата в умовах наявності перешкод та частково невизначеного середовища.

Предметом дослідження є моделі та методи автоматизації планування траєкторії руху БПЛА на основі обробки відеоданих, алгоритмів комп'ютерного зору, локальної навігації та адаптивного уникнення перешкод.

У роботі проведено аналіз сучасних методів планування траєкторії руху БПЛА, зокрема алгоритмів Dijkstra, A*, RRT та підходу на основі потенційних полів. Розроблено архітектуру системи автономної навігації, яка поєднує аналіз depth-зображень, побудову локальної карти середовища, алгоритми планування маршруту та механізми реактивного уникнення перешкод. Програмна реалізація виконана мовою Python із використанням симуляційного середовища AirSim.

Створена програмна модель автономної навігації БПЛА, яка може бути використана для дослідження алгоритмів планування траєкторії, уникнення перешкод та адаптивного керування польотом у реальному часі.

Ключові слова: *інформаційні технології, БПЛА, автономна навігація, планування траєкторії, комп'ютерний зір, depth-зображення, алгоритм A*, AirSim, уникнення перешкод, локальна карта середовища, адаптивне керування.*

ABSTRACT

The explanatory note to the qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and four appendices. The total volume of the thesis is 80 pages, including 49 pages of the main part containing 17 figures, 1 table, 26 references, and 4 appendices.

The purpose of the qualification thesis is to enhance the adaptability of UAV trajectory planning using video surveillance data and to provide real-time adaptive route correction in response to detected obstacles and changing environmental conditions.

The object of research is the process of trajectory planning for an unmanned aerial vehicle in the presence of obstacles and a partially uncertain environment.

The subject of research is models and methods for automating UAV trajectory planning based on video data processing, computer vision algorithms, local navigation, and adaptive obstacle avoidance.

The thesis analyzes modern UAV trajectory planning methods, including Dijkstra, A*, RRT algorithms, and the potential field approach. An architecture of an autonomous navigation system combining depth image analysis, local environment mapping, local path planning algorithms, and reactive obstacle avoidance mechanisms has been developed. The software implementation of the system was carried out in Python using the AirSim simulation environment.

The practical significance of the work lies in the creation of a software model for autonomous UAV navigation that can be used for researching trajectory planning algorithms, obstacle avoidance, and adaptive real-time flight control.

Keywords: *information technologies, UAV, autonomous navigation, trajectory planning, computer vision, depth images, A* algorithm, AirSim, obstacle avoidance, local environment map, adaptive control.*

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. МЕТОДИ І АЛГОРИТМИ ПЛАНУВАННЯ ТРАЄКТОРІЇ БПЛА 8

1.1. Графові методи планування траєкторії 8

1.2. Метод випадкового пошуку та підхід на основі потенційних полів 9

1.3. Отримання інформації про навколишнє середовище 11

1.4. Обробка відеоданих та оцінка відстані до перешкод 12

1.5. Локальна карта середовища та побудова сітки 13

1.6. Формування напрямку руху та інтеграція траєкторії 14

Висновки за розділом 1 14

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ ПЛАНУВАННЯ ТРАЄКТОРІЇ РУХУ БПЛА НА ОСНОВІ ВІДЕОСПОСТЕРЕЖЕНЬ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 16

2.1. Архітектура та модель системи планування траєкторії руху БПЛА на основі даних відеоспостережень 17

2.2. Аналіз depth-зображень та виділення області інтересу 19

2.3. Виявлення та класифікація перешкод 21

2.4. Побудова локальної карти середовища 26

Висновки за розділом 2 30

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МЕХАНІЗМІВ АДАПТИВНОГО КЕРУВАННЯ ТА УНИКНЕННЯ ПЕРЕШКОД 32

3.1. Побудова локальної карти середовища 32

3.2. Механізм оцінки ризику зіткнення 33

3.3. Реалізація obstacle memory та згладжування траєкторії 34

3.4. Адаптивне керування висотою польоту 36

3.5. Стабілізація yaw та генерація команд керування 42

3.6. Програмна реалізація системи в середовищі AirSim 43

Висновки за розділом 3 48

ВИСНОВОК 50

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 53

ДОДАТКИ 56

ВСТУП

Сучасний розвиток безпілотних літальних апаратів (БПЛА) супроводжується значним розширенням сфер їх використання. Сьогодні вони активно застосовуються для моніторингу територій, аерозйомки, контролю інфраструктури, у сільському господарстві, а також у рятувальних та пошукових операціях. Разом із цим зростає потреба у підвищенні рівня автономності таких систем, оскільки ручне керування не завжди є ефективним або можливим, особливо в умовах складного або небезпечного середовища [1, 2].

Однією з ключових задач у сфері автономної навігації БПЛА є планування траєкторії руху. У реальних умовах політ може відбуватися серед великої кількості перешкод, які можуть бути як заздалегідь відомими, так і з'являтися динамічно в процесі руху. Через це класичні підходи до навігації, що базуються лише на попередньо відомій карті середовища, не завжди забезпечують необхідний рівень ефективності та безпеки. У зв'язку з цим значного розвитку набули методи комп'ютерного зору, які використовують дані з камер та систем відеоспостереження для аналізу навколишнього середовища в режимі реального часу. Застосування технологій комп'ютерного зору дозволяє безпілотному апарату отримувати інформацію про простір навколо себе, виявляти перешкоди та оперативно змінювати траєкторію руху залежно від ситуації. Це відкриває можливості для створення більш гнучких і адаптивних систем навігації [3, 4]. У даній роботі розглядається підхід до планування траєкторії руху БПЛА, який поєднує методи обробки відеоданих, алгоритми навігації та механізми уникнення перешкод. Особливістю запропонованої моделі є її адаптивність, яка полягає у здатності змінювати маршрут руху залежно від інформації, отриманої з під час польоту. Такий підхід дозволяє підвищити стійкість системи до змін умов навколишнього середовища та покращити безпеку руху [5] .

Метою роботи є покращення адаптивності планування траєкторії руху БПЛА на основі даних відеоспостережень, і забезпечення адаптивного

коригування маршруту в реальному часі залежно від виявлених перешкод та умов середовища.

Об'єктом дослідження є процес планування траєкторії руху безпілотного літального апарата в умовах наявності перешкод та частково невизначеного середовища.

Предметом дослідження є моделі та методи автоматизації планування траєкторії руху БПЛА на основі обробки відеоданих, а також алгоритми комп'ютерного зору та навігації, які забезпечують адаптивну поведінку системи.

Для досягнення мети визначені **задачі** дослідження:

– аналіз сучасних методів планування траєкторії руху БПЛА та постановка задачі;

– аналіз методів комп'ютерного зору та алгоритмів локального планування траєкторії БПЛА;

– розробка моделі планування траєкторії руху БПЛА на основі даних відеоспостережень;

– розробка алгоритмів виявлення та уникнення перешкод у реальному часі;

– програмна реалізація системи автономної навігації БПЛА у середовищі AirSim;

– тестування та аналіз роботи розробленої моделі.

У роботі використовуються методи комп'ютерного зору для аналізу відеопотоку та виявлення об'єктів у навколишньому середовищі. Зокрема застосовуються підходи до обробки глибинних зображень, що дозволяють оцінювати відстань до перешкод і визначати їх розташування відносно БПЛА. Також використовуються методи планування траєкторії руху, зокрема алгоритм A*, який дозволяє знаходити оптимальний маршрут у дискретному просторі. Додатково застосовуються евристичні підходи до уникнення перешкод та методи згладжування траєкторії для забезпечення більш стабільного руху безпілотного апарата. Окремо використовуються методи фільтрації даних сенсорів для зменшення шуму та підвищення точності прийняття рішень.

Практичне значення роботи полягає у створенні програмної моделі, яка дозволяє досліджувати процес автономного планування траєкторії руху БПЛА з використанням даних відеоспостереження. Розроблена система може бути використана для тестування та аналізу алгоритмів навігації, а також як основа для подальших досліджень у сфері автономних безпілотних систем. Реалізація моделі виконана та протестована у симуляційному середовищі AirSim, яке дозволяє відтворювати умови, наближені до реального польоту [6].

РОЗДІЛ 1.
МЕТОДИ І АЛГОРИТМИ ПЛАНУВАННЯ ТРАЄКТОРІЇ БПЛА

Сучасні безпілотні літальні апарати (БПЛА) являють собою складні автономні системи, які об'єднують у собі елементи механіки, електроніки, систем керування та алгоритмів штучного інтелекту. Їх широке застосування в таких сферах, як моніторинг територій, аерозйомка, інспекція інфраструктури та рятувальні операції, зумовлює підвищені вимоги до точності навігації, безпеки польоту та автономності прийняття рішень. Однією з ключових задач у цій області є планування траєкторії руху, яке полягає у побудові оптимального маршруту переміщення БПЛА між заданими точками з урахуванням обмежень середовища та наявних перешкод.

Формально траєкторію руху можна розглядати як неперервну функцію у просторі станів, яка описує положення літального апарата у кожен момент часу. У загальному вигляді вона задається як:

$$\frac{\pi(t)}{t} \to R^n,$$

де $\pi(t)$ визначає координати БПЛА у тривимірному просторі, а R^n відповідає простору можливих станів системи.

У реальних умовах середовище, в якому відбувається рух, часто є частково або повністю невідомим, що ускладнює задачу планування та вимагає використання адаптивних методів, здатних змінювати траєкторію безпосередньо під час польоту.

1.1. Графові методи планування траєкторії

Найбільш поширеними підходами до вирішення задачі планування траєкторії є графові методи, які базуються на представленні простору у вигляді дискретного графа. У таких методах вершини графа відповідають можливим положенням БПЛА, а ребра – допустимим переходам між ними.

Одним із базових алгоритмів є *алгоритм Дейкстри* (Dijkstra), який дозволяє знайти найкоротший шлях у зваженому графі без негативних ваг. Його робота ґрунтується на поступовому накопиченні вартості переходу між вузлами, яка визначається як:

$$g(n) = g(p) + c(p, n),$$

де g(n) є сумарною вартістю до поточного вузла, g(p) − вартістю попереднього вузла, а c(p,n) − вартістю переходу між ними. Основним недоліком даного підходу є відсутність урахування напрямку до цілі, що призводить до надмірного перебору варіантів [5, 7].

Більш ефективним розвитком цього підходу є *алгоритм A**, який поєднує фактичну вартість шляху з евристичною оцінкою відстані до цільової точки. Його функція пріоритету задається як:

$$f(n) = g(n) + h(n), \tag{1.1}$$

де $g(n)$ відповідає фактичній вартості шляху від старту, а $h(n)$ є евристичною оцінкою залишкової відстані до мети. Як правило, для евристики використовується евклідова відстань:

$$h(n) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{1.2}$$

що дозволяє значно зменшити кількість розглянутих станів і прискорити пошук оптимального маршруту [5, 7].

1.2. Метод випадкового пошуку та підхід на основі потенційних полів

Окрім графових методів, у задачах планування траєкторії широко застосовуються алгоритми *випадкового пошуку*, зокрема RRT (Rapidly-exploring Random Tree). Даний метод базується на поступовій побудові дерева станів шляхом випадкового дослідження простору. Новий вузол у такому дереві обчислюється як:

$$q_{new} = q_{near} + \alpha * \frac{q_{rand} - q_{near}}{\|q_{rand} - q_{near}\|},$$

де q_{rand} є випадковою точкою у просторі, q_{near} – найближчою існуючою вершиною дерева, а α визначає крок розширення. Основною перевагою цього підходу є його

ефективність у складних багатовимірних просторах, однак недоліком є відсутність гарантії отримання оптимального маршруту [5].

Ще одним класом методів є підхід *на основі потенційних полів*, який широко використовується для локального уникнення перешкод. У цьому випадку цільова точка розглядається як джерело притягуючої сили, тоді як перешкоди створюють відштовхуючі сили. Сумарний вплив на систему визначається як:

$$F = F_{att} + F_{rep},$$

де притягуюча сила описується виразом:

$$F_{att} = -k(q - q_{goal}),$$

а відштовхуюча сила залежить від відстані до перешкоди і визначається як:

$$F_{rep} = k\left(\frac{1}{d} - \frac{1}{d_0}\right),$$

$\frac{1}{d^2}$ при $d < d_0$, і дорівнює нулю при більшій відстані. Недоліком даного підходу є можливість виникнення локальних мінімумів, у яких система втрачає напрямок руху [8].

Проведений аналіз сучасних методів показує, що кожен з них має як переваги, так і обмеження. Графові алгоритми, такі як A* та Дейкстри (Dijkstra), забезпечують ефективний пошук у відомих середовищах, проте менш придатні для динамічних умов. Методи випадкового пошуку, зокрема RRT, демонструють кращу адаптивність у складних просторах, але не гарантують оптимальність рішення. Підхід на основі потенційних полів забезпечує швидку локальну реакцію, проте може бути нестабільним у складних конфігураціях перешкод [2, 4, 5, 7, 8].

Таким чином, сучасні системи автономної навігації БПЛА базуються на комбінованому підході, який поєднує глобальне планування маршруту та локальне коригування траєкторії в реальному часі. Такий підхід дозволяє досягти балансу між оптимальністю маршруту та здатністю системи адаптуватися до змін умов середовища. Саме ця концепція використовується як основа для подальшої розробки моделі в даній роботі, де планування траєкторії

доповнюється аналізом даних відеоспостереження та динамічним уникненням перешкод.

1.3. Отримання інформації про навколишнє середовище

У задачах автономної навігації безпілотних літальних апаратів ключову роль відіграє здатність системи отримувати інформацію про навколишнє середовище та на її основі формувати рішення щодо подальшого руху. У роботі основним джерелом інформації про середовище є дані відеоспостереження, зокрема глибинні зображення, які дозволяють оцінювати відстань до об'єктів та їх просторове розташування. Такий підхід дозволяє наблизити поведінку системи до реальних умов, де БПЛА приймає рішення на основі візуальних даних, отриманих у режимі реального часу [3, 4, 9].

Для отримання вхідних даних у роботі використовується симуляційне середовище AirSim, яке надає доступ до RGB та depth-камер [6]. Особливу роль відіграють depth-зображення, оскільки кожен піксель такого зображення містить інформацію про відстань до відповідної точки сцени. Формально depth-зображення можна представити як функцію:

$$D(x, y) = z, \tag{1.3}$$

де x та y – координати пікселя зображення, а z – значення відстані від сенсора до об'єкта у відповідній точці простору. Таким чином, кожен піксель кадру містить інформацію про глибину сцени, що дозволяє не лише виявляти перешкоди, а й оцінювати їх просторове положення [10].

У межах системи глибинні дані використовуються для двох основних задач: оцінки відстані до перешкод та побудови локальної карти зайнятості простору. Для цього зображення розбивається на області інтересу (ROI), що дозволяє зосередитися на найбільш важливій частині сцени – центральній зоні перед БПЛА.

1.4. Обробка відеоданих та оцінка відстані до перешкод

Однією з ключових функцій системи є визначення відстані до об'єктів, розташованих попереду БПЛА. Для цього використовується аналіз центральної частини depth-зображення, що дозволяє зменшити вплив шуму та периферійних об'єктів. Оцінка відстані виконується шляхом обчислення медіани значень у центральній області:

$$d_{front} = median\,(D_{ROI}).$$

Використання медіани є більш стійким до шумів порівняно із середнім арифметичним, оскільки дозволяє ігнорувати різкі аномальні значення, які можуть виникати через помилки сенсора або відблиски [10, 11]. Далі застосовується експоненційне згладжування для стабілізації вимірювань:

$$d_t = \alpha d_{t-1} + (1-\alpha) d_t,$$

де α – коефіцієнт згладжування, який визначає вагу попередніх вимірювань.

Такий підхід дозволяє уникнути різких стрибків у значеннях відстані та забезпечує більш плавну реакцію системи керування [11]. Аналогічний підхід використовується для оцінки відстані до землі, що дозволяє контролювати висоту польоту над поверхнею:

$$AGL = median(D_{bottom}), \tag{1.4}$$

де AGL (Above Ground Level) – висота над землею [10].

Окрім оцінки відстані, система виконує аналіз форми та структури об'єктів у кадрі для їх класифікації. Для цього використовується аналіз ROI центральної частини зображення, де визначаються області з низькими значеннями глибини (ближчі об'єкти). Бінарна маска будується за умовою:

$$M(x,y) = \begin{cases} 1, & D(x,y) < T \\ 0, & \text{інакше} \end{cases},$$

де T – порогове значення, яке визначає, чи є об'єкт достатньо близьким для вважання перешкодою [12].

На основі цієї маски обчислюються характеристики об'єктів, такі як щільність, вертикальна та горизонтальна протяжність, а також зміщення об'єкта у нижню частину зображення. Це дозволяє класифікувати перешкоди як

«дерево», «кущ», «камінь» або «земля» [12, 13]. Наприклад, щільність об'єкта визначається як:

$$p = \frac{\sum M(x,y)}{N},$$

де N – кількість пікселів у ROI.

1.5. Локальна карта середовища та побудова сітки

Для задачі локального планування траєкторії використовується дискретизація простору у вигляді сітки розміром N×N. У даній роботі використовується сітка розміру 20×20, яка будується на основі depth-зображення. Кожна комірка сітки позначається як зайнята або вільна відповідно до порогового значення відстані:

$$G(i,j) = \begin{cases} 1, & D(i,j) < T_{obs} \\ 0, & \text{інакше'} \end{cases}$$

де T_{obs} – поріг виявлення перешкоди, а G(i,j) – елемент карти зайнятості простору [14, 15, 16].

Таким чином формується локальне представлення середовища, яке використовується для подальшого пошуку маршруту. Цей підхід дозволяє значно спростити задачу планування, оскільки безперервний простір перетворюється у дискретний граф [14, 15].

Для побудови маршруту в межах локальної карти використовується алгоритм A*, який є одним із найефективніших методів пошуку шляху у дискретних просторах. Для оцінки вузла використовуємо формулу (1.1), а задля реалізації використовуємо функцію евклідової або мангеттенської відстані (1.2).

Алгоритм A* дозволяє ефективно знаходити локально оптимальний маршрут у межах обмеженої області, що особливо важливо в умовах динамічного середовища, де карта постійно змінюється на основі нових сенсорних даних [5, 7].

1.6. Формування напрямку руху та інтеграція траєкторії

Після побудови маршруту у вигляді послідовності вузлів сітки здійснюється перетворення цього маршруту у вектор напрямку руху. Це виконується шляхом обчислення нормалізованого вектору між поточною позицією та цільовою точкою:

$$\vec{d} = \frac{p_{target} - p_{current}}{\|p_{target} - p_{current}\|}.$$

Цей вектор $\vec{d}$ використовується як базовий вектор руху БПЛА, який у подальшому коригується з урахуванням даних про перешкоди [17, 18].

Однією з ключових особливостей системи є механізм адаптивного уникнення перешкод, який працює у режимі реального часу. Його основою є оцінка ризику зіткнення, яка залежить від відстані до об'єкта:

$$R(d)\begin{cases} 1, & d < d_{min} \\ \frac{d_{safe} - d}{d_{safe} - d_{min}}, & d_{min} \le d < d_{safe} \\ 0, & d \ge d_{safe} \end{cases}.$$

На основі цієї функції формується додатковий вектор відштовхування, який змінює напрям руху БПЛА. Таким чином, система здатна реагувати на появу перешкод без повного перепланування маршруту, що є важливою перевагою для роботи в динамічних умовах [17, 18, 19]. У межах реалізації також використовується комбінування глобального напрямку руху та локальних коригуючих впливів, що дозволяє отримати більш стабільну та плавну траєкторію [20, 21].

Висновки за розділом 1

У першому розділі було проведено аналіз існуючих підходів до планування траєкторії руху безпілотних літальних апаратів. Розглянуто графові методи планування маршруту, зокрема алгоритми Дейкстри та A*, а також методи випадкового пошуку та підхід на основі потенційних полів. Визначено їх основні переваги, недоліки та особливості використання в умовах динамічного середовища.

Встановлено, що графові алгоритми забезпечують ефективний пошук оптимального маршруту у відомому середовищі, однак мають обмеження при роботі в умовах появи динамічних перешкод. Методи випадкового пошуку демонструють кращу адаптивність у складних просторових конфігураціях, проте не гарантують оптимальності маршруту. Підхід на основі потенційних полів дозволяє швидко реагувати на перешкоди, але може призводити до виникнення локальних мінімумів.

У результаті аналізу встановлено доцільність використання комбінованого підходу, який поєднує глобальне планування маршруту та локальне адаптивне уникнення перешкод. Саме така концепція була обрана як основа для подальшої розробки моделі планування траєкторії руху БПЛА на основі даних відеоспостереження.

Також проведено аналіз методів локального планування траєкторії руху БПЛА на основі відеоданих. Розглянуто принципи обробки depth-зображень, методи оцінки відстані до перешкод, побудови локальної карти середовища та формування напрямку руху безпілотного літального апарата.

Встановлено, що використання depth-камер дозволяє отримувати інформацію про просторове розташування об’єктів у режимі реального часу та забезпечує можливість адаптивної навігації у частково невідомому середовищі. Для оцінки відстані до перешкод доцільним є використання медіанної та процентильної обробки depth-даних, що дозволяє зменшити вплив шуму сенсорів.

Показано ефективність використання локальної карти зайнятості простору та алгоритму A* для побудови маршруту в дискретному середовищі. Також визначено доцільність поєднання глобального напрямку руху та локального механізму уникнення перешкод для забезпечення більш стабільної та адаптивної траєкторії польоту.

Отримані результати стали основою для проєктування власної моделі планування траєкторії руху БПЛА та її подальшої програмної реалізації.

РОЗДІЛ 2.
РОЗРОБКА МОДЕЛІ ПЛАНУВАННЯ ТРАЄКТОРІЇ РУХУ БПЛА НА ОСНОВІ ВІДЕОСПОСТЕРЕЖЕНЬ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

У попередньому розділі було проведено аналіз сучасних методів планування траєкторії руху безпілотних літальних апаратів, розглянуто сучасні підходи комп'ютерного зору, методи обробки відеоданих, а також алгоритми локальної навігації та уникнення перешкод. Проведений аналіз дозволив сформувати вимоги до розробки власної моделі планування траєкторії руху БПЛА на основі даних відеоспостережень. Основною метою даного розділу є розробка моделі автономного планування траєкторії руху БПЛА, яка забезпечує адаптивне переміщення у середовищі з потенційними перешкодами на основі аналізу depth-зображень. Запропонована модель повинна забезпечувати формування безпечної траєкторії польоту, підтримку стабільної висоти, виявлення об'єктів на маршруті, а також зміну напрямку руху відповідно до умов навколишнього середовища в режимі реального часу. На відміну від підходів, що потребують попередньо побудованої карти місцевості або використання складних систем позиціонування, у даній роботі пропонується використання локального аналізу середовища на основі depth-даних, отриманих із камер відеоспостереження. Це дозволяє БПЛА приймати рішення щодо зміни траєкторії безпосередньо під час польоту, адаптуючись до появи нових перешкод та змін просторової конфігурації середовища.

У межах цього розділу буде детально розглянуто архітектуру розробленої системи, логіку функціонування окремих модулів, математичні основи формування траєкторії руху, механізми уникнення перешкод та принципи керування польотом. Окрему увагу буде приділено програмній реалізації моделі, що була виконана мовою Python із використанням симуляційного середовища AirSim, а також аналізу роботи системи під час моделювання польоту.

2.1. Архітектура та модель системи планування траєкторії руху БПЛА на основі даних відеоспостережень

Однією з основних задач сучасних безпілотних літальних апаратів є забезпечення автономного переміщення у середовищі зі змінними умовами та наявністю потенційних перешкод. У даній роботі було розроблено модель планування траєкторії руху БПЛА, яка базується на використанні даних відеоспостереження для оцінки навколишнього середовища, виявлення перешкод та формування безпечного маршруту польоту. Особливістю розробленого підходу є поєднання методів комп'ютерного зору, локального планування маршруту, адаптивного уникнення перешкод та системи керування польотом у межах єдиної програмної моделі. На відміну від традиційних підходів, що базуються виключно на попередньо відомих картах місцевості або GPS-навігації, запропонована система приймає рішення щодо зміни траєкторії безпосередньо в процесі руху БПЛА, аналізуючи поточний стан середовища на основі depth-зображень.

Загальна архітектура розробленої системи планування траєкторії руху БПЛА наведена на рисунку 2.1. Архітектура запропонованої системи побудована за модульним принципом, що дозволяє логічно розділити процес автономної навігації на окремі функціональні компоненти. Такий підхід забезпечує кращу структурованість програмної реалізації, можливість масштабування системи, а також подальшу адаптацію моделі до використання на реальних безпілотних платформах.

Серед усіх компонентів розробленої системи найбільш важливу роль відіграють модуль аналізу середовища, модель планування траєкторії та модуль адаптивного уникнення перешкод. Модуль аналізу середовища здійснює обробку глибинних даних, отриманих із камер БПЛА, та формує інформацію про розташування навколишніх об'єктів і можливих перешкод. На основі цих даних будується локальна карта середовища, яка використовується для подальшої навігації.

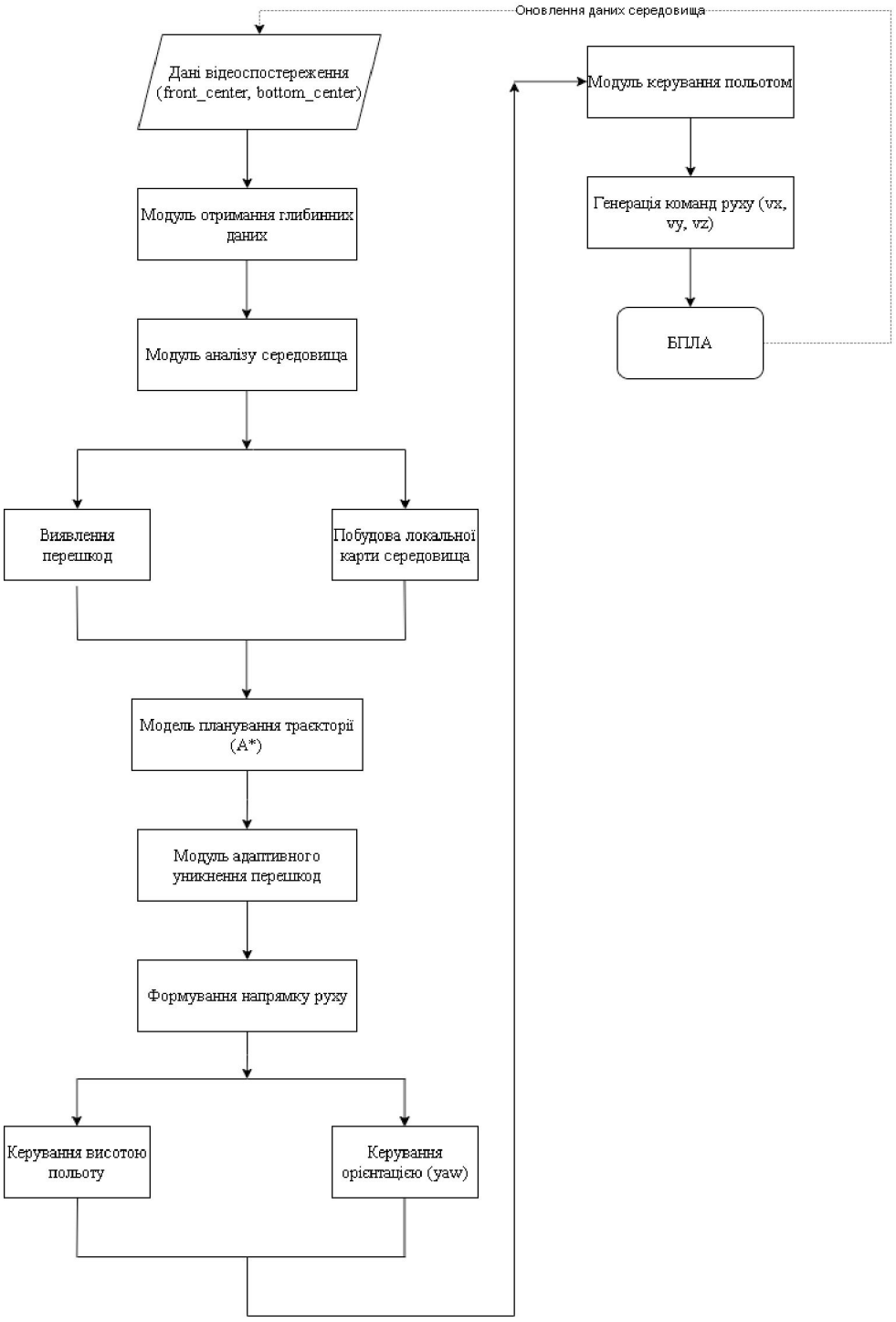


Рисунок 2.1 – Архітектура системи планування траєкторії руху БПЛА на основі даних відеоспостереження.

Ключовим елементом системи є модель планування траєкторії, реалізована на основі алгоритму A*. Даний модуль визначає оптимальний маршрут руху БПЛА до цільової точки з урахуванням наявних перешкод та обмежень

середовища. Отримана траєкторія передається до модулів керування для подальшого виконання.

Важливе значення також має модуль адаптивного уникнення перешкод, який забезпечує коригування маршруту в режимі реального часу. У разі виявлення нових перешкод або змін у навколишньому середовищі модуль оперативно змінює напрямок руху, що дозволяє підтримувати безпечний політ.

Спільна робота цих модулів забезпечує автономне прийняття рішень та підвищує ефективність навігації БПЛА в динамічних умовах.

2.2. Аналіз depth-зображень та виділення області інтересу

Першим етапом роботи системи є отримання даних відеоспостереження. У розробленій моделі використовуються дві камери, доступ до яких реалізується через API симуляційного середовища AirSim. Фронтальна камера (front_center) використовується для аналізу простору перед БПЛА, виявлення потенційних перешкод, оцінки ризику зіткнення та побудови локальної карти середовища. Нижня камера (bottom_center) використовується для визначення висоти польоту відносно поверхні землі та стабілізації вертикального положення БПЛА. У межах розробленої системи використовується не класичний RGB-аналіз відеопотоку, а depth-представлення сцени, що дозволяє отримувати інформацію про відстань до об’єктів [5, 17, 22]. Depth-зображення можна представити у вигляді функції (1.3).

Після отримання depth-зображення система переходить до етапу аналізу навколишнього середовища. На цьому етапі виконується виділення області інтересу (Region of Interest, ROI), яка використовується для оцінки простору безпосередньо перед БПЛА (рис. 2.2). Використання лише центральної області кадру дозволяє зменшити обчислювальне навантаження та мінімізувати вплив другорядних об’єктів, які не впливають на поточну траєкторію польоту[9, 10, 22].



Рисунок 2.2 – Область аналізу depth-зображення (ROI) у фронтальній камері БПЛА.

Вибір ROI, яка використовується для аналізу depth-зображення з фронтальної камери, обумовлений особливостями просторової орієнтації БПЛА під час руху. У реалізованій системі аналізується лише центральна частина кадру, яка відповідає безпосередньому напрямку польоту БПЛА. У програмній реалізації це задається шляхом обрізання зображення за координатами:

```
roi = d[
img.height//4 : 3*img.height//4,
img.width//4  : 3*img.width//4
]
```

Такий вибір меж дозволяє виключити з аналізу периферійні області зображення, які мають нижчу інформативність щодо траєкторії руху та часто містять спотворення перспективи або об’єкти, що не впливають безпосередньо на поточний маршрут. Верхня частина кадру переважно містить небо або віддалені об’єкти, які не становлять критичної загрози для зіткнення на поточному етапі руху, тоді як нижня частина аналізується окремо через використання нижньої камери та функції оцінки висоти польоту ground_distance().

Крім того, обмеження області аналізу дозволяє зменшити обчислювальне навантаження, що є критичним для систем реального часу, оскільки алгоритм виконується з частотою до 20 циклів на секунду. Зменшення розміру даних depth-зображення безпосередньо впливає на швидкодію алгоритмів виявлення перешкод та побудови локальної карти середовища.

Таким чином, обрана область інтересу є компромісом між точністю виявлення перешкод та продуктивністю системи, забезпечуючи стабільну роботу алгоритму планування траєкторії руху БПЛА в умовах обмежених обчислювальних ресурсів.

2.3. Виявлення та класифікація перешкод

Після виділення області інтересу система переходить до етапу оцінки простору перед БПЛА. Для цього у програмній реалізації використовується окремий модуль аналізу depth-зображення, основною задачею якого є визначення мінімальної безпечної відстані до потенційної перешкоди у напрямку руху. У розробленій системі дана логіка реалізована функцією front_depth(), яка виконує аналіз depth-кадру з фронтальної камери та дозволяє оцінити ступінь небезпеки простору перед безпілотним літальним апаратом. На відміну від класичних методів, де використовується усереднення значень глибини по всьому кадру, у роботі застосовано локальний аналіз декількох ключових областей зображення. Зокрема, depth-кадр умовно поділяється на три основні сегменти: центральну область, ліву частину та праву частину. Такий підхід дозволяє не лише визначити наявність перешкоди безпосередньо перед БПЛА, але й оцінити можливість безпечного обходу перешкоди в одну зі сторін [17, 18, 22].

Варто зазначити, що аналіз depth-зображення у розробленій системі здійснюється на двох рівнях (рис. 2.3).



Рисунок 2.3 – Області аналізу depth-зображення для оцінки простору перед БПЛА.

Центральна область інтересу використовується для задач класифікації перешкод та аналізу їх просторового розташування, тоді як оцінка безпечної дистанції до об’єктів виконується шляхом аналізу трьох вертикальних зон по всій висоті кадру. Такий підхід дозволяє зберігати високу чутливість до перешкод, що можуть з’являтися у верхній або нижній частині траєкторії польоту.

Для оцінки відстані до потенційних перешкод у кожній області використовується не середнє арифметичне значення глибини, а п’ятий процентиль вибірки:

$$D_{safe} = P_5(D),$$

де D_{safe} – оцінка мінімальної безпечної відстані, P_5 – п’ятий процентиль, а D – множина depth-значень області аналізу.

Фактично це означає, що система орієнтується не на середню глибину сцени, а на найближчі об’єкти у полі зору. Використання середнього значення могло б призвести до некоректної оцінки ситуації, оскільки декілька далеких об’єктів можуть суттєво збільшити середній показник і приховати небезпечну перешкоду, що знаходиться близько до БПЛА. Для прикладу, якщо у

центральній області depth-зображення більшість значень становить 30–40 м, але присутній окремий об’єкт на відстані 4 м, середнє значення може показувати безпечний простір, хоча ризик зіткнення вже є критичним. Саме тому у реалізованій моделі використовується процентильна оцінка:

$$D_{front} = \min\left(P_5(C), P_5(L), P_5(R)\right),$$

де C – центральна область, L – ліва область, а R – права область

Тобто кінцевою оцінкою простору перед БПЛА є мінімальне значення між центральною, лівою та правою зонами аналізу. Такий підхід дозволяє швидко реагувати навіть на локальні перешкоди, що можуть з’являтися з боків траєкторії руху.

Для зменшення впливу шуму depth-сенсора у системі також реалізовано механізм згладжування значень відстані за допомогою експоненційного фільтра:

$$D_t = \alpha D_{t-1} + (1 - \alpha) D_{new},$$

де D_t – поточне відфільтроване значення, D_{t-1} – попереднє значення, D_{new} – нове значення depth, а α – коефіцієнт згладжування.

У реалізованій системі значення коефіцієнта фільтрації становить α = 0.7, що забезпечує компроміс між швидкістю реакції та стабільністю оцінки середовища. Високе значення коефіцієнта дозволяє уникнути різких коливань сигналу, спричинених шумом depth-камери, однак при цьому зберігає достатню швидкість адаптації до появи нових перешкод.

На рис. 2.4 добре видно, як саме використовуваний дрон «бачить» – чим ближче до нього об’єкт, тим він більш темний ніж той, що знаходиться від нього далі і не становить загрози його польоту.

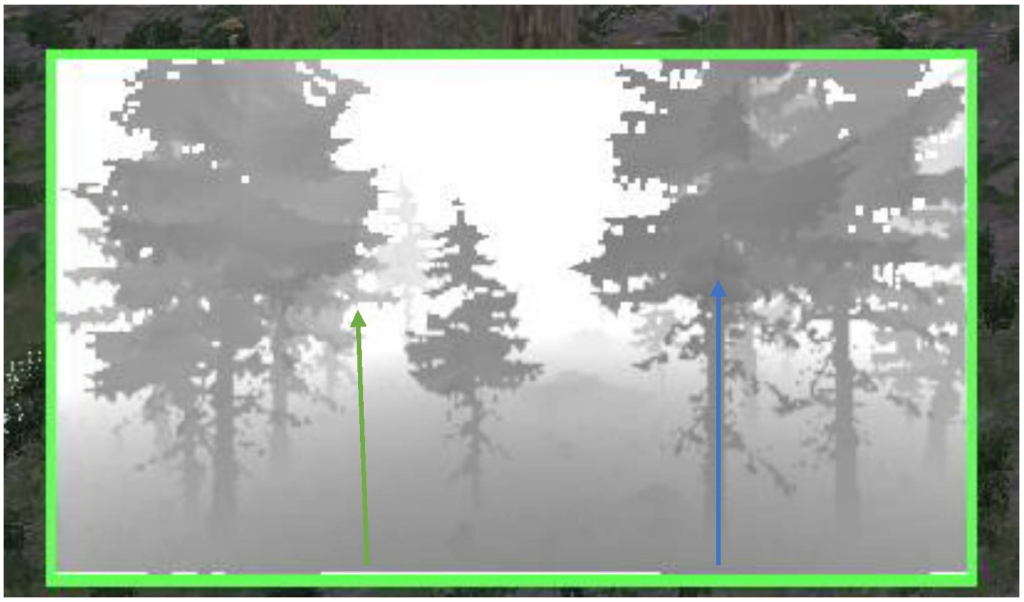


Рисунок 2.4 – Приклад виявлення перешкоди на depth-зображенні.

Недостатньо лише визначити наявність перешкоди перед БПЛА – для забезпечення більш адаптивної зміни траєкторії важливо також оцінювати тип виявленого об’єкта. У реальних умовах польоту різні категорії перешкод можуть потребувати різних стратегій обходу. Наприклад, дерево зазвичай є вертикально орієнтованою перешкодою, яку доцільніше обходити збоку, тоді як низький кущ або нерівність рельєфу можуть вимагати зміни висоти польоту. Саме тому у межах розробленої системи реалізовано модуль евристичної класифікації перешкод, який дозволяє оцінювати геометричні характеристики об’єктів на основі depth-зображення.

На відміну від сучасних підходів, що базуються на використанні згорткових нейронних мереж (CNN), у даній роботі застосовано евристичний метод класифікації [6, 7, 8, 23]. Такий вибір обумовлений необхідністю зменшення обчислювального навантаження, забезпечення роботи в режимі реального часу та відсутністю потреби у попередньому навчанні моделі на великих наборах даних. Крім того, у межах симуляційного середовища AirSim використання евристичного аналізу дозволяє достатньо точно моделювати поведінку системи уникнення перешкод без значного ускладнення архітектури.

Для класифікації перешкод у системі використовується функція classify_obstacle(), яка аналізує центральну область depth-зображення та виділяє області потенційних об'єктів шляхом порогової сегментації. Спочатку з ROI відбираються лише ті точки, що знаходяться ближче за певну межу глибини:

$$D(x, y) < D_{treshold},$$

де $D_{treshold}$ – гранична дистанція виявлення об'єкта.

У реалізованій системі максимальна дистанція аналізу становить приблизно 50 м, що дозволяє відсіювати занадто далекі об'єкти та концентрувати обчислення лише на потенційно небезпечних елементах середовища. Після первинної фільтрації виконується визначення просторових характеристик об'єкта. Для цього обчислюється його вертикальний та горизонтальний розмір відносно області аналізу:

$$H = \frac{y_{max} - y_{min}}{H_{img}}, W = \frac{x_{max} - x_{min}}{W_{img}},$$

де H – нормалізована висота об'єкта, W – нормалізована ширина об'єкта, (W_{img}, H_{img}) – висота та ширина ROI та (x_{min},x_{max},y_{min},y_{max}) – межі виявленої області.

Окрім геометричних розмірів, система оцінює щільність заповнення області перешкоди:

$$p = \frac{N_{obstacle}}{N_{ROI}},$$

де ρ – щільність об'єкта, $N_{obstacle}$ – кількість пікселів, що належать перешкоді та N_{ROI} – загальна кількість пікселів області аналізу.

Додатково оцінюється вертикальне положення об'єкта у кадрі, що дозволяє визначити, чи є перешкода низькою, чи високою. Для цього використовується середня координата об'єкта:

$$Y_{mean} = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad (2.1)$$

де y_i – вертикальні координати пікселів виявленого об'єкта.

На основі комбінації перелічених характеристик система виконує евристичну класифікацію перешкод на декілька категорій: дерево (tree), кущ (bush), камінь (rock), поверхня землі (ground) та невизначений об'єкт (unknown).

Як приклад роботи класифікатора можна розглянути випадок виявлення дерева. Після обробки depth-зображення система формує маску найближчих об'єктів та визначає їх геометричні характеристики. Якщо висота об'єкта займає понад 55% висоти області аналізу (height_span > 0.55), ширина є відносно невеликою (width_span < 0.5), а щільність заповнення маски не перевищує 45% (density < 0.45), об'єкт класифікується як дерево (tree). Такий вибір обумовлений тим, що дерево зазвичай має витягнуту вертикальну форму та займає відносно невелику площу по ширині кадру. Наприклад, для об'єкта з параметрами: height_span = 0.72, width_span = 0.31 та density = 0.28, система віднесе його до категорії tree та призначить коефіцієнт безпеки 1.4. Надалі ця інформація використовується модулем планування траєкторії для коригування маршруту БПЛА та забезпечення безпечного обходу перешкоди.

2.4. Побудова локальної карти середовища

Після класифікації перешкод система переходить до етапу локального планування траєкторії руху. На відміну від глобального планування, яке передбачає використання повної карти місцевості та попередньо відомих координат середовища, у роботі застосовується підхід локального планування маршруту. Його особливістю є формування траєкторії безпосередньо під час польоту на основі актуальних даних відеоспостереження, що дозволяє БПЛА адаптуватися до динамічних змін середовища та появи нових перешкод.

У розробленій моделі локальна навігація базується на побудові дискретної карти простору у вигляді двовимірної сітки перешкод. Формування локальної карти реалізується функцією build_local_grid(), яка перетворює depth-зображення у матрицю зайнятості (occupancy grid). У такій матриці кожна клітинка описує стан певної ділянки простору:

$$G(i,j) = \begin{cases} 1, & \text{якщо виявлено перешкоду} \\ 0, & \text{якщо простір вільний'} \end{cases}$$

де G(i,j) − значення клітинки локальної карти, а (i,j) − координати клітинки у дискретній сітці.

У реалізованій системі використовується локальна карта розміром 20 × 20, що задається параметром `GRID_SIZE = 20`.

Вибір такого розміру обумовлений необхідністю досягнення компромісу між точністю представлення середовища та швидкістю обчислень. Збільшення кількості клітинок дозволило б детальніше описувати перешкоди, однак суттєво збільшило б час побудови маршруту, що негативно впливає на роботу системи в режимі реального часу. Під час формування локальної карти кожна клітинка перевіряється на наявність потенційної перешкоди шляхом порівняння значення глибини з пороговою дистанцією:

$$D(x, y) < D_{obs}$$

де D(x,y) – значення depth, а D_{obs} – гранична відстань виявлення перешкоди.

У розробленій системі використовується значення D_{obs}= 6м, що визначається параметром `OBSTACLE_TH = 6.0`.

Таким чином, якщо відстань до об'єкта менша за встановлений поріг, відповідна область локальної карти позначається як зайнята та надалі враховується при плануванні маршруту.

Після побудови локальної карти система переходить до планування маршруту між поточним положенням БПЛА та локальною цільовою точкою. Для вирішення цієї задачі використовується алгоритм A*, який є одним із найбільш поширених методів пошуку оптимального шляху в робототехнічних системах та навігаційних задачах [24]. Основною перевагою алгоритму A* є можливість знаходження оптимального маршруту при відносно низькій обчислювальній складності. Алгоритм використовує оцінювальну функцію (1.1). У розробленій моделі використовується мангеттенська евристика:

$$h(n) = |x_n - x_g| + |y_n - y_g|,$$

де (x_n,y_n) – координати поточної клітинки, а (x_g,y_g) – координати цілі.

Вибір саме мангеттенської відстані обумовлений її низькою обчислювальною складністю та стабільною роботою в дискретних сітках середовища.

У реалізованій моделі алгоритм допускає переміщення не лише по вертикалі та горизонталі, а й по діагоналі. Вартість переходу між клітинками задається таким чином:

$$c = \begin{cases} 1, \text{горизонтальний/вертикальний рух} \\ 1.4, \text{діагональний рух} \end{cases}.$$

Число 1.4 є наближенням $-\sqrt{2} \approx 1.414$, що відповідає евклідовій довжині діагонального переміщення між сусідніми клітинками.

На рис. 2.5 представлено UML-діаграму діяльності алгоритму A*, що використовується у розробленій системі локального планування траєкторії. Алгоритм здійснює пошук оптимального маршруту між поточним положенням БПЛА та локальною цільовою точкою з урахуванням перешкод, виявлених на основі аналізу depth-зображення.

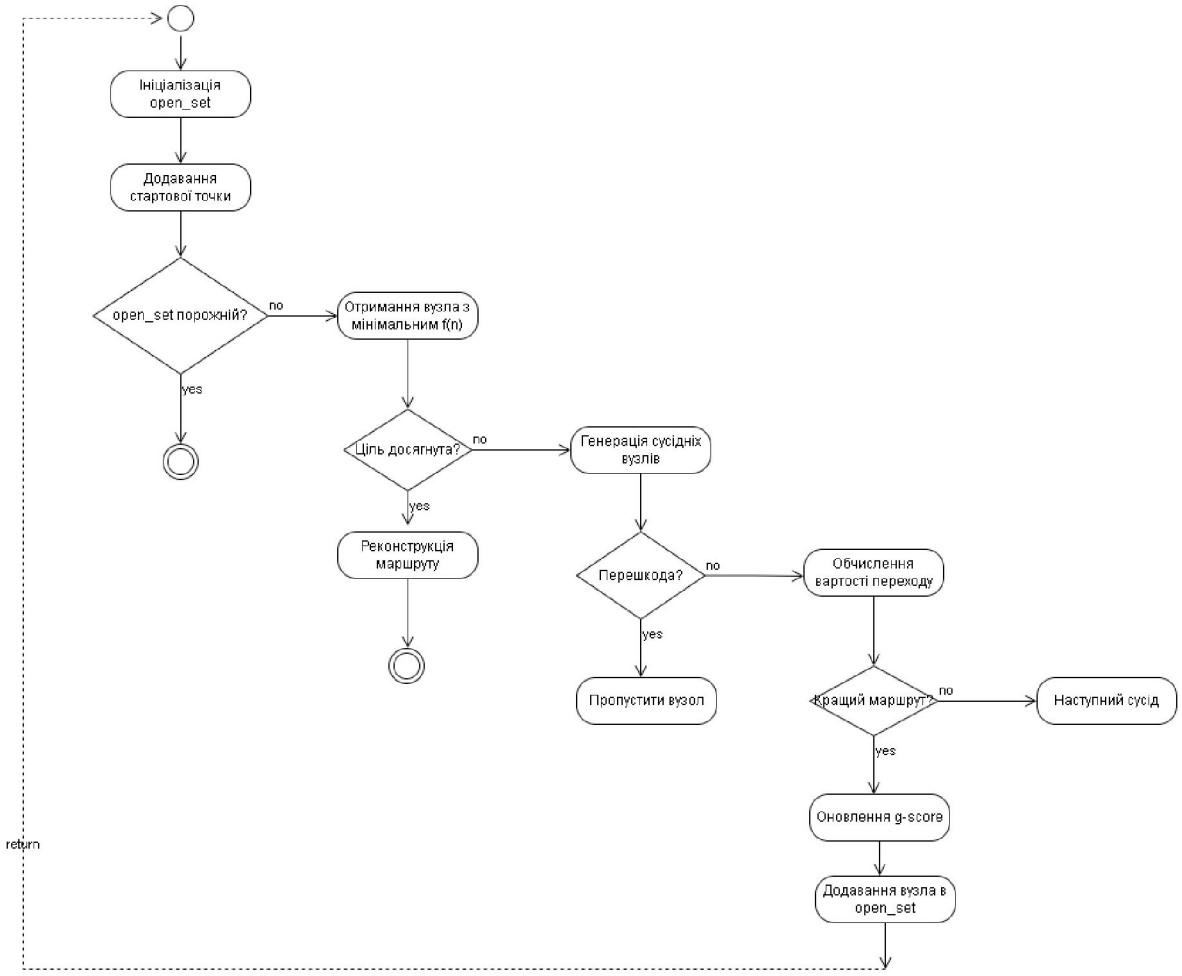


Рисунок 2.5 – UML діаграма активностей роботи алгоритму A*.

На рис. 2.6 представлено приклад фактичної траєкторії руху безпілотного літального апарата під час виконання польотного завдання в середовищі AirSim.

На графіку відображено початкову позицію БПЛА, кінцеву цільову точку (waypoint), побудовану траєкторію руху, а також точки виявлених перешкод, що були отримані під час аналізу depth-зображень.

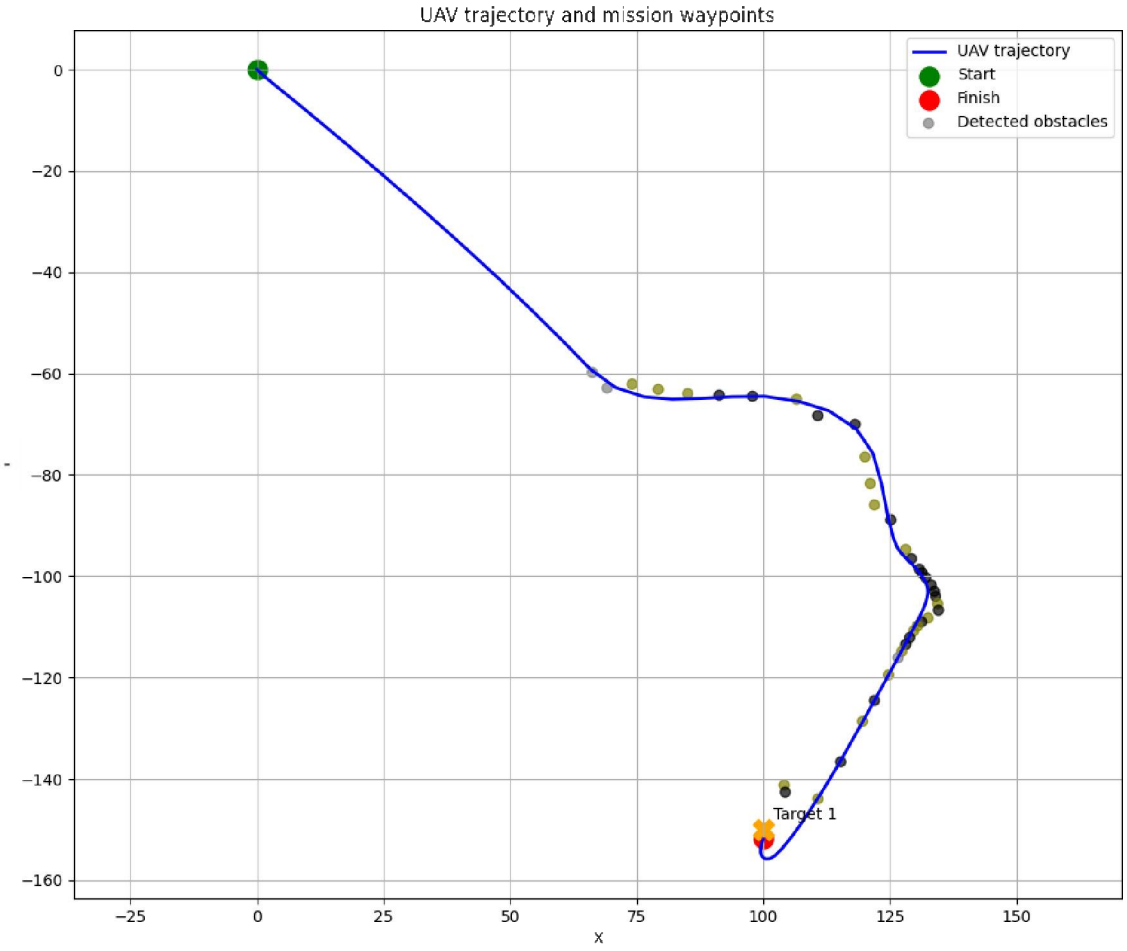


Рисунок 2.6 – Приклад побудованої траєкторії алгоритмом A* на локальній карті.

Можна помітити, що на початковій ділянці маршруту рух БПЛА відбувається майже по прямолінійній траєкторії, оскільки у полі зору системи відсутні критичні перешкоди. Однак у центральній частині маршруту спостерігається поступова зміна напрямку руху, викликана виявленням групи

об’єктів перед БПЛА. У даному випадку система адаптивного планування траєкторії виконує локальну корекцію маршруту та формує безпечний обхід потенційно небезпечної області.

Особливістю реалізованої моделі є відсутність різких змін напрямку польоту. Замість миттєвого повороту використовується плавна корекція вектора руху, що дозволяє уникнути нестабільності керування та різких маневрів. Це добре помітно на заключній ділянці маршруту, де БПЛА здійснює поступове повернення до цільової точки після обходу перешкоди. Важливо зазначити, що побудована траєкторія не є прямим результатом роботи лише алгоритму A*. У межах реалізованої моделі кінцевий напрямок руху формується як комбінація декількох компонентів: напрямку до цілі, рекомендацій локального планування маршруту, сформованих алгоритмом A*, а також модуля реактивного уникнення перешкод. Це можна представити у вигляді:

$$\vec{D} = \alpha\overrightarrow{D_{target}} + \beta\overrightarrow{D_{A*}} + \gamma\overrightarrow{D_{avoid}},$$

де $\overrightarrow{D_{target}}$ – базовий напрямок до цілі, $\overrightarrow{D_{A*}}$ – локальна корекція маршруту, сформована алгоритмом A*, $\overrightarrow{D_{avoid}}$ – вектор уникнення перешкод, а (α,β,γ) – вагові коефіцієнти впливу.

У реалізованій системі домінуючим компонентом є напрямок до цілі, тоді як A* та avoidance-механізм виконують локальне коригування траєкторії лише у випадку появи потенційно небезпечних об’єктів у зоні польоту. Саме це забезпечує більш природну та адаптивну поведінку БПЛА у середовищі з перешкодами.

Висновки за розділом 2

У другому розділі було виконано проєктування моделі планування траєкторії руху БПЛА на основі даних відеоспостереження. Розроблено архітектуру системи, яка об’єднує модулі аналізу depth-зображень, виявлення та класифікації перешкод, побудови локальної карти середовища та локального планування маршруту.

У межах роботи реалізовано підхід до аналізу області інтересу depth-зображення, що дозволяє зменшити обчислювальне навантаження та зосередити обробку на найбільш важливій частині сцени. Запропоновано механізм оцінки безпечної дистанції до перешкод із використанням процентильної оцінки depth-даних, що забезпечує підвищення надійності виявлення небезпечних об'єктів.

Також реалізовано евристичний механізм класифікації перешкод, який дозволяє враховувати геометричні характеристики об'єктів під час вибору траєкторії обходу. Для локального планування маршруту використано алгоритм A*, що забезпечує побудову безпечного шляху в межах локальної карти середовища.

У результаті було сформовано структуровану модель системи автономної навігації БПЛА, яка забезпечує адаптивне планування траєкторії на основі аналізу відеоданих у режимі реального часу.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ МЕХАНІЗМІВ АДАПТИВНОГО КЕРУВАННЯ ТА УНИКНЕННЯ ПЕРЕШКОД

Однією з ключових задач автономної навігації безпілотного літального апарата є забезпечення безпечного руху в умовах невизначеного середовища. У реальних сценаріях польоту БПЛА постійно стикається зі зміною просторової конфігурації середовища, появою нових об'єктів та частковою невизначеністю сенсорних даних. Використання лише глобального планування маршруту у таких умовах є недостатнім, оскільки заздалегідь побудована траєкторія не враховує динамічні зміни простору. Саме тому у даному розділі кваліфікаційної роботи реалізовано адаптивний механізм планування траєкторії, який дозволяє коригувати напрямок руху БПЛА у режимі реального часу [18, 19, 24, 25].

3.1. Побудова локальної карти середовища

Основною особливістю розробленої моделі є комбінований принцип формування траєкторії (рис. 3.1), у якому кінцевий напрямок руху визначається не одним алгоритмом, а сукупністю декількох компонентів. Зокрема, у системі одночасно враховуються напрямок руху до цільової точки, локальний маршрут, побудований алгоритмом A*, реактивний механізм уникнення перешкод, система стабілізації висоти польоту та механізм згладжування руху. Такий підхід дозволяє уникнути характерної проблеми багатьох класичних алгоритмів планування маршруту, коли БПЛА здійснює різкі повороти або нестабільно реагує на перешкоди.

Базовий напрямок польоту визначається шляхом побудови вектора між поточним положенням БПЛА та цільовою точкою маршруту:

$$\overrightarrow{D_{target}} = \frac{P_{target} - P_{uav}}{\|P_{target} - P_{uav}\|},$$

де P_{target} – координати цілі, а P_{uav} – поточна позиція БПЛА.

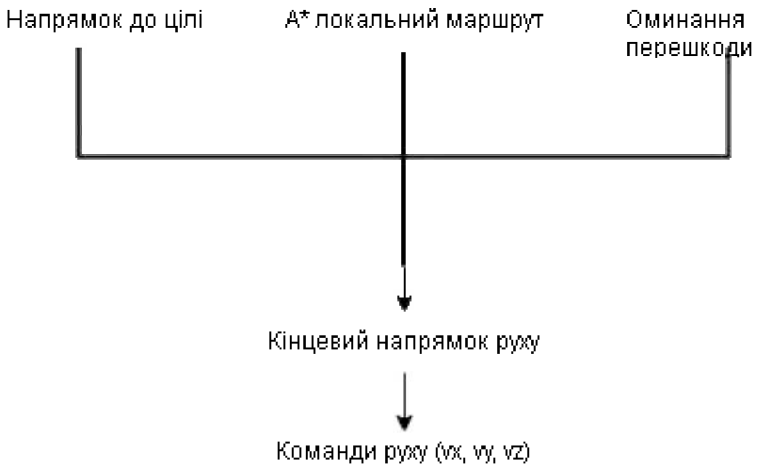


Рисунок 3.1 – Схема формування результуючого напрямку руху БПЛА .

Нормалізація вектору необхідна для отримання лише напрямку руху без урахування абсолютної відстані до цілі. Однак використання лише такого підходу робить систему надто чутливою до перешкод, оскільки БПЛА намагатиметься рухатися напряму до цілі навіть у потенційно небезпечних умовах [21, 25, 26].

3.2. Механізм оцінки ризику зіткнення

Для вирішення цієї проблеми використовується адаптивний механізм оцінки ризику зіткнення [19]. У розробленій системі рівень небезпеки визначається на основі відстані до найближчої перешкоди перед БПЛА:

$$R(d) = \begin{cases} 0, & d > 18 \\ 1, & d < 6 \\ \frac{18-d}{12}, & 6 \le d \le 18 \end{cases},$$

де R(d) – коефіцієнт ризику, а d – дистанція до перешкоди.

Фактично дана функція виконує плавну адаптацію поведінки БПЛА залежно від відстані до перешкоди. При великій дистанції система не активує уникнення перешкод, а при критичному зближенні сила корекції значно збільшується (рис. 3.2).

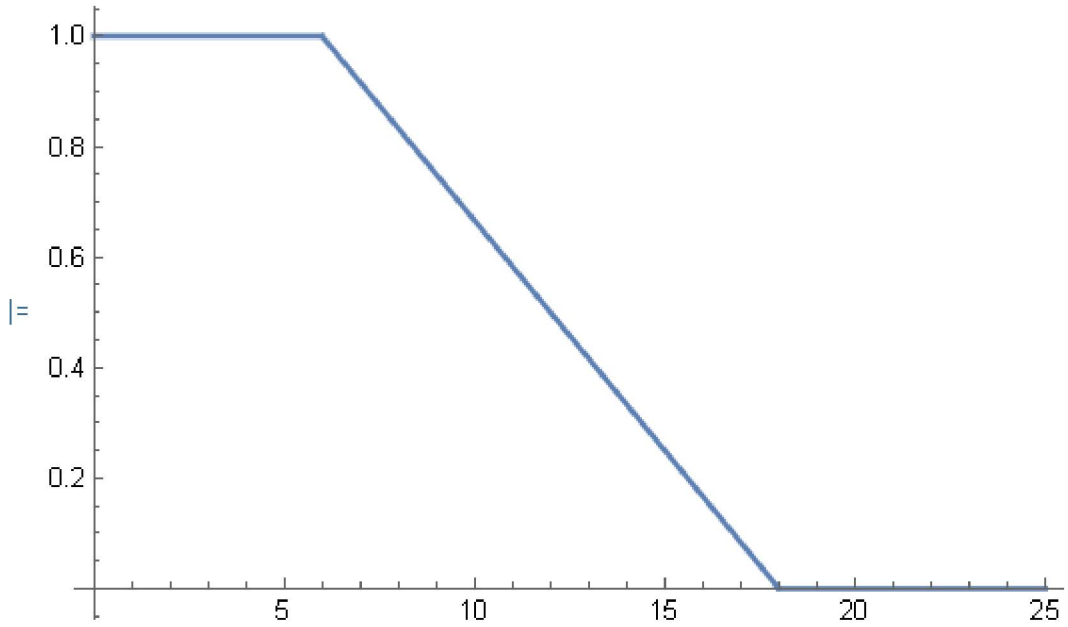


Рисунок 3.2 − Графік функції оцінки ризику зіткнення.

Однією з ключових проблем реактивних систем уникнення перешкод є нестабільність поведінки при швидкій зміні сенсорних даних. У випадку, якщо система приймає рішення лише на основі поточного кадру depth-зображення, навіть короткочасне зникнення перешкоди з поля зору може призвести до різкої зміни напрямку руху або передчасного повернення на небезпечну траєкторію. Подібна поведінка створює ефект нестабільності руху та може викликати коливання курсу БПЛА.

3.3. Реалізація obstacle memory та згладжування траєкторії

Для вирішення цієї проблеми у межах розробленої моделі реалізовано механізм адаптивної пам’яті перешкод (OBSTACLE_MEMORY). Його основною задачею є накопичення інформації про нещодавно виявлені перешкоди та поступове зменшення їх впливу на траєкторію руху навіть після тимчасового зникнення з області спостереження. У програмній реалізації механізм адаптивної пам’яті базується на використанні експоненційного згасання:

$$M_t = \lambda M_{t-1},$$

де M_t – поточний рівень obstacle memory, M_{t-1} – попереднє значення, а λ – коефіцієнт згасання. У системі використовується значення λ=0.85, що забезпечує плавне зменшення впливу перешкоди у часі.

Крім механізму згасання, obstacle memory також враховує поточний рівень небезпеки, що формується на основі оцінки ризику та типу виявленої перешкоди:

$$S_{obs} = R(d) * W_{obj},$$

де S_{obs} – сигнал небезпеки, R(d) – ризик зіткнення, а W_{obj} – ваговий коефіцієнт типу obstacle.

У роззробленій моделі різні категорії об'єктів мають різний вплив на траєкторію руху. Наприклад, вертикальні перешкоди типу tree отримують більшу вагу, оскільки потенційно становлять вищий ризик для польоту, ніж низькорозташовані об'єкти. Оновлення obstacle memory здійснюється за правилом:

$$M_t = \max(\lambda M_{t-1}, S_{obs}).$$

Тобто система або зберігає поступово згасаючу пам'ять про obstacle, або підсилює її, якщо нова перешкода створює більший рівень ризику. Подібний підхід дозволяє уникнути явища «мерехтіння» траєкторії, коли БПЛА постійно змінює курс через незначні коливання depth-зображення.

Після оцінки рівня небезпеки система переходить до побудови вектора уникнення перешкод. На відміну від класичних підходів, де obstacle avoidance реалізується лише шляхом відштовхування від об'єкта, у роботі використовується напрямок, перпендикулярний до базового руху БПЛА. Для формування бокового обходу використовується ортогональний вектор:

$$\overrightarrow{D_{side}} = (-D_y, D_x),$$

де D_y, D_x – компоненти напрямку польоту, а $\overrightarrow{D_{side}}$ – боковий вектор обходу перешкоди.

Такий підхід дозволяє не виконувати повне гальмування або різкий розворот, а плавно зміщувати траєкторію вбік. Вектор уникнення визначається як:

$$\overrightarrow{D_{avoid}} = k_a * M_t * \overrightarrow{D_{side}},$$

де k_a – коефіцієнт сили обходу, M_t – obstacle memory, а $\overrightarrow{D_{side}}$ – боковий напрямок.

У системі значення коефіцієнта обходу становить k_a=4.5, що дозволяє забезпечити достатню силу корекції траєкторії навіть при критичному наближенні до obstacle.

3.4. Адаптивне керування висотою польоту

Однією з важливих особливостей автономної навігації безпілотного літального апарата є необхідність врахування вертикального простору під час планування маршруту. Використання лише двовимірного обходу перешкод може бути недостатнім у випадках, коли об'єкти знаходяться на різних висотах або займають лише частину простору польоту. Саме тому у розробленій моделі реалізовано механізм адаптивного керування висотою, який дозволяє БПЛА змінювати траєкторію не лише у горизонтальній площині, але й по вертикалі [17, 26].

На відміну від класичних підходів, де висота польоту підтримується сталою, у даній роботі використовується адаптивна логіка зміни altitude залежно від положення перешкоди у полі зору фронтальної depth-камери. Основною ідеєю є аналіз вертикального розташування перешкоди на зображенні та визначення найбільш безпечної стратегії обходу. Для оцінки вертикального положення перешкоди використовується середня координата області перешкод (2.1).

Після нормалізації відносно висоти кадру отримується відносний рівень obstacle:

$$H_{level} = \frac{Y_{mean}}{H_{img}},$$

де H_{img} – висота ROI.

Отримане значення дозволяє визначити приблизне положення перешкоди відносно траєкторії польоту БПЛА. Якщо перешкода розташовується у верхній частині кадру, система інтерпретує це як об'єкт, що знаходиться вище рівня

польоту, а тому доцільним є маневр проходження під ним. Водночас перешкода у нижній частині depth-зображення інтерпретується як низька перешкода або елемент рельєфу, що потребує набору висоти (рис 3.3). У загальному вигляді логіка вибору траєкторії може бути представлена таким правилом:

$$H_{target} = \begin{cases} H_0 - \Delta H, & H_{level} < 0.35 \\ H_0 + \Delta H, & H_{level} > 0.60, \\ H_0, & \text{інакше} \end{cases}$$

де H_0 – базова висота польоту, ΔH – корекція висоти, а H_{level} – цільова висота

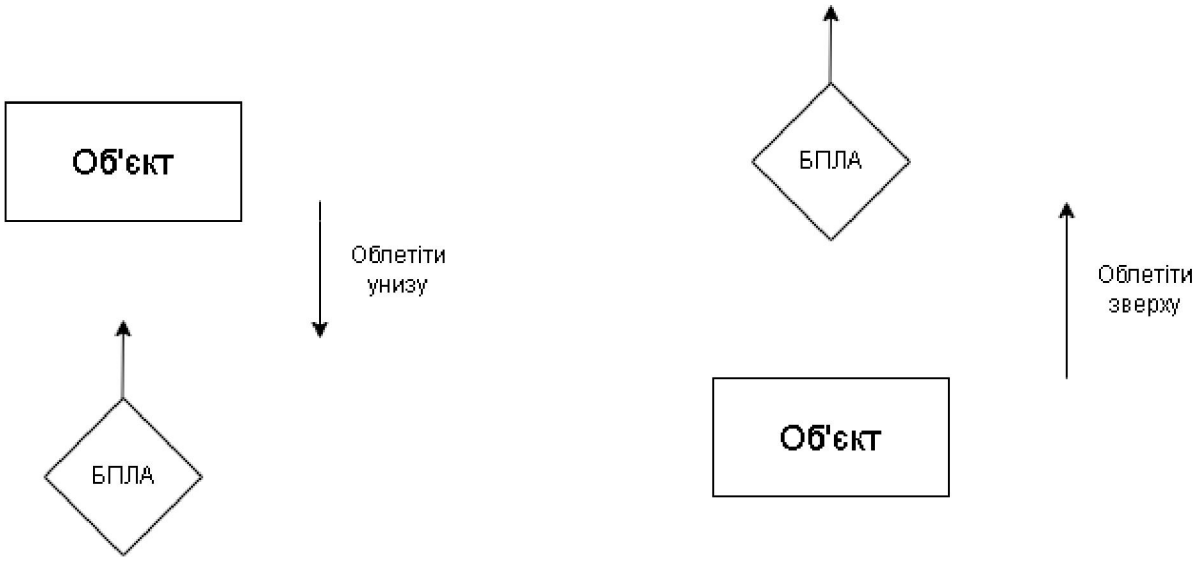


Рисунок 3.3 – Принцип адаптивної зміни висоти польоту БПЛА.

У реалізованій моделі використовується базова висота H_0=3.5 м. При цьому система може динамічно зменшувати висоту приблизно на 1.5 м або збільшувати її до 2 м залежно від просторового положення obstacle.

Для підтримання стабільної висоти польоту використовується оцінка відстані до землі на основі depth-зображення нижньої камери [5, 17]. У системі реалізовано функцію ground_distance(), яка визначає актуальну висоту БПЛА над поверхнею. Оцінка висоти здійснюється на основі медіанного значення центральної області нижнього depth-зображення (1.4).

Схема стабілізація зображена на рисунку 3.4.

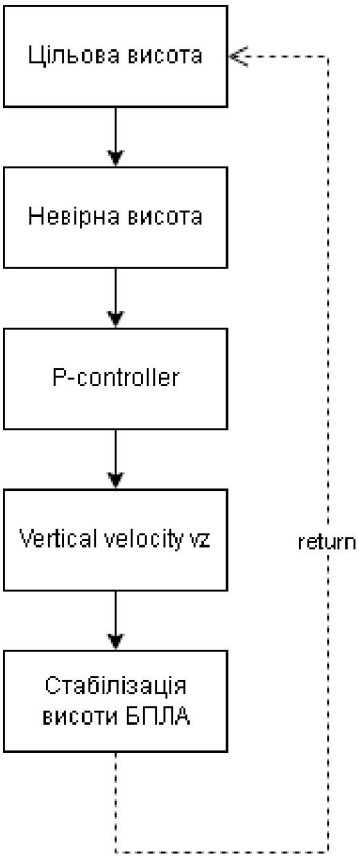


Рисунок 3.4 – Схема стабілізації висоти польоту БПЛА.

Використання медіани замість середнього значення дозволяє зменшити вплив аномальних depth-вимірювань та шуму сенсора. Для стабілізації висоти застосовується пропорційний регулятор (P-controller):

$$V_z = K_p(AGL - AGL_{target}),$$

де V_z – вертикальна швидкість, K_p – коефіцієнт регулятора, а AGL_{target} – бажана висота.

У розробленій системі використовується $K_p = 0.6$, а максимальна швидкість вертикального переміщення обмежується $|V_z| \leq 2.0$ м/с, що дозволяє уникнути надто агресивних вертикальних маневрів та забезпечує стабільність польоту. Після формування результуючого напрямку руху система переходить до етапу стабілізації траєкторії. У випадку прямого використання миттєво обчисленого напрямку польоту виникає проблема різких змін траєкторії, особливо у середовищі з великою кількістю перешкод або нестабільними depth-

вимірюваннями. Це може призвести до коливань курсу, частих змін yaw та нестійкого руху БПЛА.

Для вирішення цієї проблеми у розробленій системі реалізовано механізм згладжування траєкторії (vector smoothing). Його основна задача полягає у поступовій зміні напрямку польоту замість миттєвої перебудови маршруту. У програмній реалізації використовується експоненційне згладжування вектору руху:

$$\overrightarrow{V_t} = \alpha\overrightarrow{V_{t-1}} + (1-\alpha)\overrightarrow{D_t},$$

де $\overrightarrow{V_t}$ – згладжений напрямок руху, $\overrightarrow{V_{t-1}}$ – попередній вектор, а $\overrightarrow{D_t}$ – новий обчислений напрямок.

У моделі використовується α=0.58. Таке значення забезпечує баланс між адаптивністю та стабільністю руху. При меншому значенні БПЛА реагуватиме занадто різко, а при більшому – надмірно повільно змінюватиме курс. Після згладжування результуючий напрямок нормалізується:

$$\overrightarrow{D_{norm}} = \frac{\overrightarrow{V_t}}{\|\overrightarrow{V_t}\|},$$

що дозволяє використовувати його виключно як напрямок руху незалежно від абсолютної довжини вектору.

Крім напрямку руху, важливим елементом стабільного польоту є коректна орієнтація БПЛА у просторі. У розробленій моделі використовується адаптивне керування yaw (рис. 3.5), яке дозволяє апарату постійно орієнтуватися у напрямку руху. Кут yaw визначається на основі результуючого вектору польоту:

$$Yaw = arctan\left(\frac{D_y}{D_x}\right),$$

де D_x і D_y – компоненти напрямку руху.

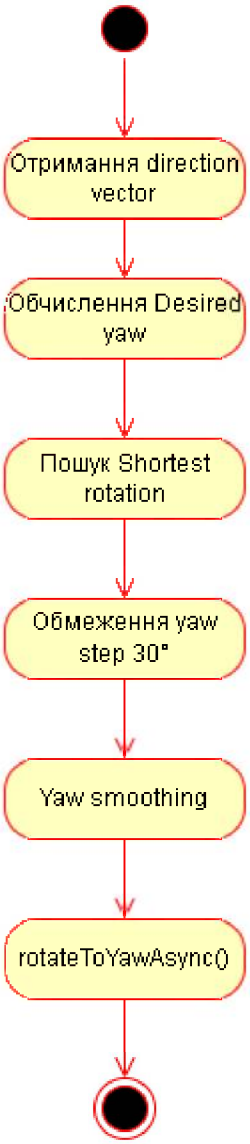


Рисунок 3.5 – UML–діаграма активностей механізму керування yaw.

У програмній реалізації використовується функція:

```
math.atan2(direction[1], direction[0])
```

оскільки вона дозволяє правильно визначати кут в усіх квадрантах координатної площини. Для уникнення різких поворотів реалізовано обмеження максимальної зміни yaw за один цикл керування:

$$\Delta yaw_{max} = 30.$$

Тобто навіть якщо перешкода вимагає різкої зміни напрямку, БПЛА виконує плавний розворот, що позитивно впливає на стабільність польоту. Крім цього, використовується згладжування yaw:

$$Yaw_t = Yaw_{t-1} + \alpha \Delta Yaw,$$

де α – коефіцієнт стабілізації yaw.

У системі використовується α=0.99, що забезпечує майже безперервний поворот у напрямку польоту.

Після визначення напрямку руху та орієнтації система переходить до генерації команд керування швидкістю БПЛА (рис. 3.6). Горизонтальні компоненти швидкості визначаються множенням нормалізованого напрямку руху на бажану швидкість польоту:

$$V_x = D_x * V, \;\; V_y = D_y * V,$$

де V – швидкість руху.

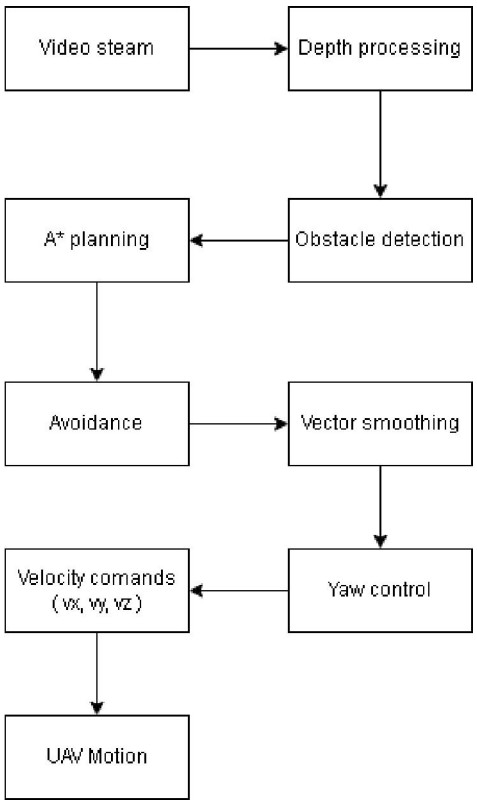


Рисунок 3.6 – Загальна схема формування команд керування БПЛА.

Вертикальна швидкість формується як сума:

$$V_z = V_{PID} + V_{avoid},$$

де V_{PID} – стабілізація altitude, а V_{avoid} – вертикальний компонент обходу перешкоди.

Отримані команди передаються у систему керування AirSim:

```
client.moveByVelocityAsync(
    vx,
    vy,
    vz + vz_cmd,
    duration
)
```

що забезпечує безперервне оновлення траєкторії у режимі реального часу.

3.5. Стабілізація yaw та генерація команд керування

Програмна реалізація системи планування траєкторії руху БПЛА була виконана мовою Python із використанням симуляційного середовища AirSim, що працює на базі Unreal Engine. Використання симуляції дозволило безпечно протестувати алгоритм у різних сценаріях польоту, змінювати конфігурацію перешкод та аналізувати поведінку системи без ризику пошкодження реального обладнання.

Оскільки метою роботи було дослідження адаптивного планування траєкторії на основі відеоспостережень, реалізована модель (рис 3.7) орієнтована саме на обробку depth-зображень, побудову локальної карти середовища та формування команд руху у режимі реального часу.

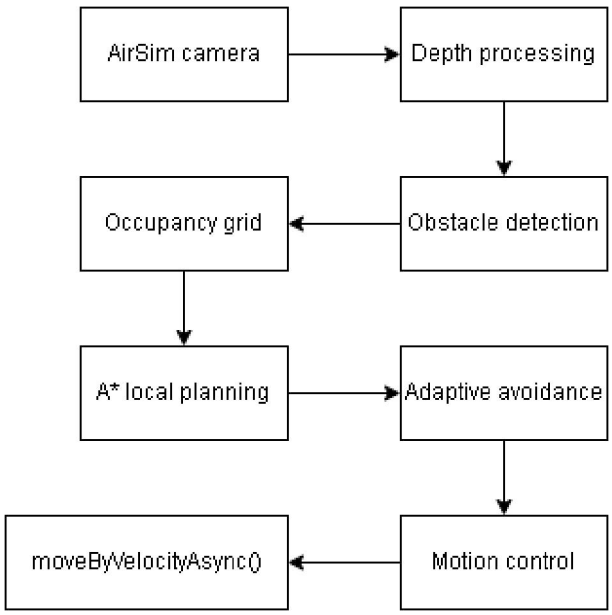


Рисунок 3.7 – Архітектура програмної реалізації моделі планування траєкторії БПЛА.

У програмній реалізації система складається з декількох взаємопов'язаних модулів, а саме отримання depth-зображень, аналізу перешкод, локального планування маршруту, адаптивного уникнення obstacle, стабілізації руху, формування команд керування БПЛА.

3.6. Програмна реалізація системи в середовищі AirSim

Отримання сенсорних даних реалізовано за допомогою API AirSim. Для аналізу навколишнього середовища використовуються фронтальна та нижня depth-камери. Фронтальна камера забезпечує виявлення перешкоди перед БПЛА, тоді як нижня використовується для оцінки висоти над поверхнею. Основна логіка отримання depth-зображення реалізована через функцію:

```
simGetImages()
```

Подальша обробка depth-даних здійснюється відповідно до підходу, описаного у підрозділі 2.2, де формується область інтересу (ROI) та локальна карта зайнятості.

Приклад depth-зображення у середовищі AirSim можна побачити на рис. 3.8.



Рисунок 3.8 – Приклад depth-зображення у середовищі AirSim.

Для локального планування маршруту алгоритм A*, який працює на occupancy grid, сформованій із depth-зображення. Програмна реалізація алгоритму представлена функцією:

`def astar(grid, start, goal)`

На рисунку 3.9 можна побачити, що БПЛА коригує свою траєкторію враховуючи перешкоди, що знаходяться на його шляху. Після того як оминає небезпеку – вирівнює свою траєкторію аби вдало досягти заданої йому цілі, в кінці, після сповільнення швидкості досягає цільової точки і вилітає на наступну координату, як і до цього, оминаючи всі перешкоди.

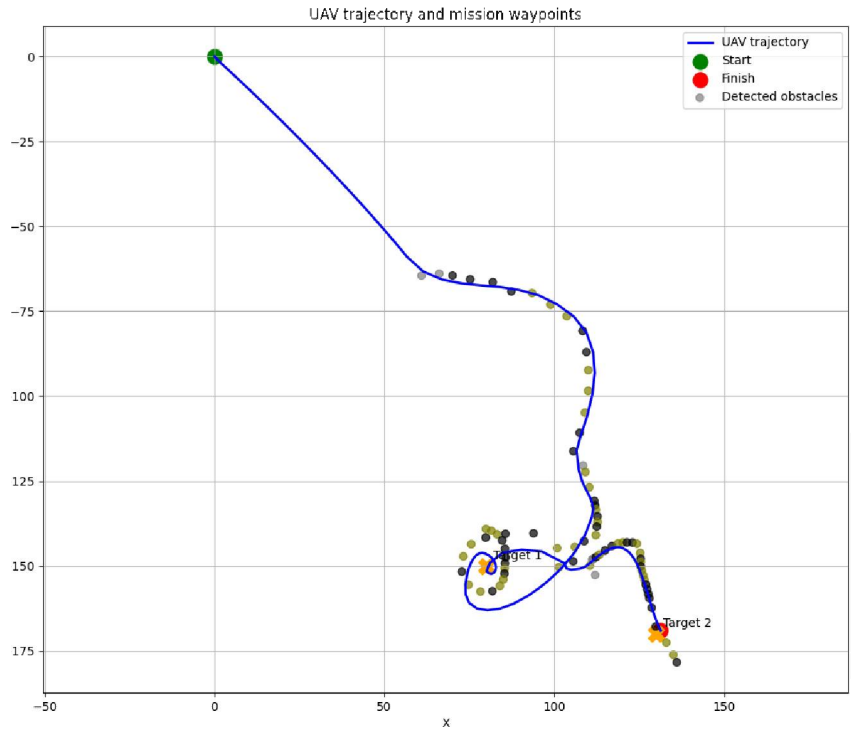


Рисунок 3.9 – Приклад локального маршруту, побудованого алгоритмом A*.

Алгоритм виконує пошук найменш затратного маршруту між поточною позицією БПЛА та локальною ціллю, використовуючи евристичну оцінку відстані (див. підрозділ 3.1). У програмній реалізації для ефективного вибору вузла використовується пріоритетна черга:

`heapq.heappush()`,

а сформований маршрут перетворюється у локальний вектор руху через:

`path_to_direction(path)`.

Окремим компонентом системи є модуль адаптивного уникнення перешкод, реалізований на основі механізму obstacle memory та векторної корекції напрямку руху. Теоретичні принципи його роботи були наведені у підрозділі 3.3,

тому у межах цього підрозділу акцент зроблено на практичній реалізації. Ключовою особливістю моделі є те, що obstacle avoidance не виконує повної перебудови маршруту, а лише локально коригує напрямок руху БПЛА. Завдяки цьому вдається уникнути різких змін траєкторії та нестабільної поведінки.

Формування avoidance-вектора, UML-діаграму якого представлена на рисунку 3.10, здійснюється у функції:

`fly()`,

де кінцевий напрямок польоту формується шляхом комбінування – target direction, A* route. obstacle avoidance та altitude correction.



Рисунок 3.10 – UML–діаграма роботи алгоритму керування польотом (fly()).

Для перевірки працездатності моделі було проведено серію експериментів у симуляційному середовищі AirSim. Тестування проводилось у декількох сценаріях, які відрізнялись кількістю та конфігурацією obstacle.

Було досліджено сценарії:

- рух без перешкод;
- рух при наявності одиночної перешкоди;
- рух у середовищі з групою перешкод;
- перешкоди що вимагають вертикального обходу (вгорі, внизу);
- змішані сценарії з комбінованим обходом.

Під час тестування сценаріїв з таблиці 3.1 оцінювались здатність БПЛА до уникнення перешкод, плавність траєкторії, стабільність польоту, точність досягнення до цілі та адаптація до зміни висоти. Для оцінки ефективності моделі враховані успішність досягнення до цілі, відсутність зіткнень, стабільність траєкторії та швидкість реакції на перешкоду.

Таблиця 3.1

Сценарії тестування моделі і результати

№	Сценарій	Результат
1	Без перешкод	Прямолінійний рух
2	Одна перешкода	Локальний обхід
3	Декілька перешкод	Корекція траєкторії
4	Верхня перешкода	Зниження висоти
5	Нижня перешкода	Набір висоти

Приклад траєкторії руху БПЛА під час обходу перешкоди показано на рисунку 3.11.

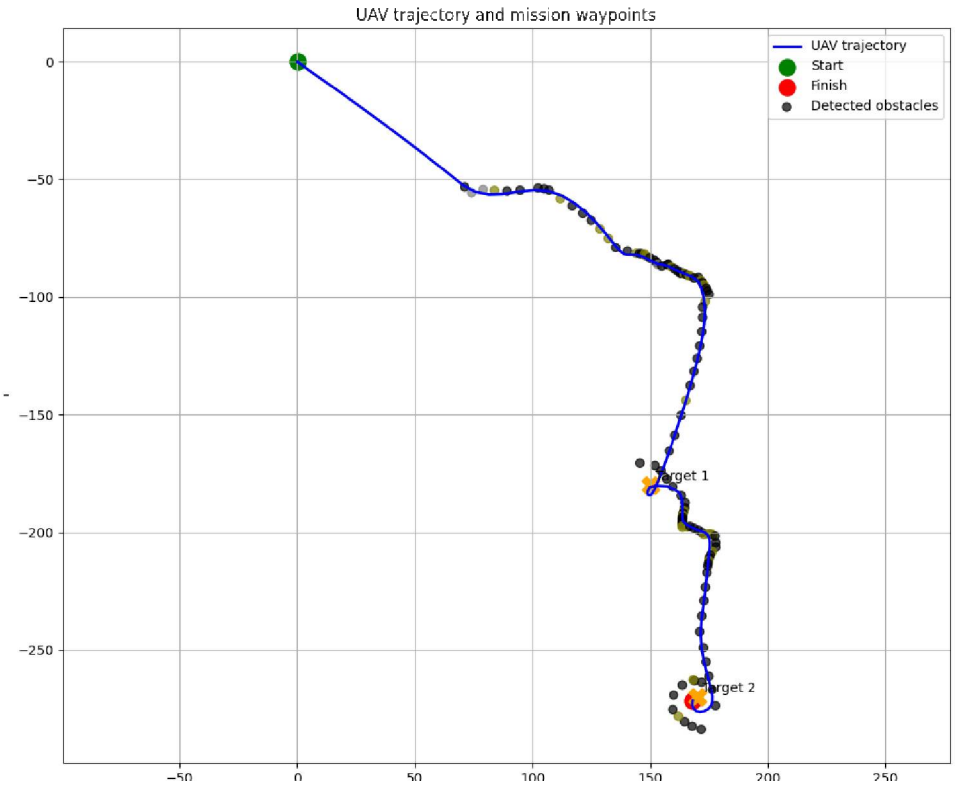


Рисунок 3.11 – Приклад траєкторії БПЛА під час обходу obstacle.

У результаті проведених експериментів встановлено, що розроблена модель здатна забезпечувати адаптивне коригування траєкторії БПЛА у середовищі з obstacle. Під час тестування БПЛА успішно здійснював обхід перешкод без повної перебудови маршруту, використовуючи локальну корекцію напрямку. Особливо важливим результатом є плавність сформованої траєкторії, що була досягнута завдяки використанню механізму obstacle memory, згладжування векторів руху та обмеження швидкості зміни yaw. У більшості тестових сценаріїв БПЛА успішно досягав цілей без критичних відхилень маршруту.

Водночас проведене тестування також показало певні обмеження моделі. Зокрема, робота алгоритму досліджувалась лише у симуляційному середовищі AirSim, тому питання адаптації до реального БПЛА потребує додаткового дослідження. Крім того, точність обходу перешкоди може залежати від якості depth-зображень та характеристик середовища.

Висновки за розділом 3

У третьому розділі було реалізовано механізми адаптивного керування рухом БПЛА та уникнення перешкод у реальному часі. Розроблено комбінований підхід до формування результуючого напрямку руху, який поєднує глобальний напрямок польоту, локальне планування маршруту та реактивний механізм обходу перешкод.

Запропоновано механізм оцінки ризику зіткнення на основі відстані до об'єктів, що дозволяє динамічно змінювати силу впливу системи уникнення перешкод залежно від рівня небезпеки. Для підвищення стабільності роботи системи реалізовано механізм obstacle memory, який забезпечує поступове згасання впливу нещодавно виявлених перешкод та зменшує нестабільність траєкторії.

Також реалізовано адаптивне керування висотою польоту на основі аналізу вертикального положення перешкод у depth-зображенні. Для забезпечення плавності руху використано механізми згладжування траєкторії та стабілізації yaw, що дозволило уникнути різких маневрів та покращити стабільність польоту.

У результаті було створено адаптивну систему керування рухом БПЛА, здатну ефективно функціонувати в умовах динамічного середовища та змінної конфігурації перешкод.

Також було виконано програмну реалізацію моделі планування траєкторії руху БПЛА та проведено дослідження її роботи у симуляційному середовищі AirSim. Реалізацію системи виконано мовою Python із використанням API AirSim для отримання depth-зображень, формування команд керування та моделювання процесу польоту.

Під час тестування було перевірено роботу модулів аналізу depth-даних, побудови локальної карти середовища, алгоритму A*, механізмів уникнення перешкод та стабілізації руху. Результати моделювання показали, що запропонована система здатна адаптивно змінювати траєкторію руху залежно від конфігурації середовища та забезпечувати безпечний обхід перешкод.

Проведене дослідження підтвердило ефективність використання depth-зображень для задач автономної навігації БПЛА та доцільність застосування комбінованого підходу до планування траєкторії. Запропонована модель забезпечує стабільний рух, плавне коригування маршруту та можливість функціонування у режимі реального часу.

Отримані результати можуть бути використані як основа для подальшого розвитку систем автономної навігації безпілотних літальних апаратів та інтеграції розроблених алгоритмів у реальні робототехнічні платформи.

ВИСНОВОК

У кваліфікаційній роботі було розглянуто проблему планування траєкторії руху безпілотного літального апарата на основі даних відеоспостережень. Актуальність теми зумовлена необхідністю підвищення рівня автономності БПЛА під час руху в умовах наявності перешкод та змін навколишнього середовища. Одним із важливих завдань сучасних безпілотних систем є здатність самостійно аналізувати простір перед собою, визначати потенційно небезпечні об'єкти та коригувати маршрут руху без прямого втручання оператора. У межах виконання дипломної роботи було проведено аналіз існуючих підходів до планування траєкторії руху БПЛА. Розглянуто класичні алгоритми пошуку маршруту, зокрема алгоритм Дейкстри, алгоритм A*, а також методи локального уникнення перешкод і сучасні підходи до використання комп'ютерного зору в задачах автономної навігації. Проведений аналіз показав, що для задач локального планування маршруту в умовах обмежених обчислювальних ресурсів ефективним є використання комбінованих підходів, які поєднують алгоритми маршрутизації та методи аналізу візуальних даних.

У роботі було досліджено методи комп'ютерного зору та обробки depth-зображень для навігації БПЛА. Розглянуто підходи до отримання depth-даних, формування області інтересу, виявлення obstacle та побудови локальної карти середовища. Встановлено, що використання depth-зображень дозволяє отримувати інформацію про відстань до перешкоди та використовувати її для адаптивного коригування маршруту руху БПЛА. Основним результатом дипломної роботи стала розробка моделі планування траєкторії руху БПЛА на основі depth-даних, отриманих із відеосенсорів у симуляційному середовищі. Розроблена модель базується на комбінованому підході, у межах якого використовується локальне планування маршруту алгоритмом A*, механізм адаптивного уникнення перешкод та формування команд керування на основі аналізу depth-зображень.

У межах реалізованої моделі було розроблено механізм побудови локальної карти середовища (occupancy grid) на основі depth-зображень фронтальної камери. Для локального обходу перешкоди використано алгоритм A*, який застосовується для побудови безпечного напрямку руху в межах локальної області перед БПЛА. На відміну від повної перебудови маршруту, запропонований підхід здійснює локальну корекцію траєкторії, що дозволяє забезпечити більш плавний рух та швидшу реакцію на перешкоду. Особливістю розробленої моделі є використання адаптивного механізму корекції траєкторії, який враховує відстань до перешкоди, їх розташування у кадрі та ризик потенційного зіткнення. Для підвищення стабільності польоту використано механізм obstacle memory, що дозволяє уникнути різких змін траєкторії у випадку короткочасної втрати перешкоди у полі зору камери. Крім горизонтального обходу перешкоди, у роботі реалізовано механізм адаптивного керування висотою польоту, який дозволяє змінювати висоту БПЛА залежно від вертикального положення перешкоди у depth-зображенні. Це дало можливість реалізувати не лише горизонтальний, але й просторовий обхід перешкоди. Практична реалізація моделі виконана мовою Python із використанням бібліотек NumPy, Matplotlib, heapq та API середовища AirSim. Для перевірки працездатності алгоритму використано симуляційне середовище AirSim, у якому реалізовано тестові сценарії з різною конфігурацією перешкод. Під час тестування досліджено роботу алгоритму у сценаріях із відсутністю перешкоди, одиночними перешкодами та групами перешкод. Отримані результати показали, що запропонована модель здатна виконувати локальне коригування траєкторії, уникати перешкоди та забезпечувати досягнення заданих цілей у більшості тестових сценаріїв. Також встановлено, що використання механізмів згладжування траєкторії та плавної зміни yaw позитивно впливає на стабільність польоту.

Поставлена у роботі мета – покращення адаптивності планування траєкторії руху БПЛА на основі даних відеоспостережень – була досягнута. Під час виконання роботи проведено аналіз існуючих підходів до навігації БПЛА,

досліджено методи комп'ютерного зору для аналізу depth-даних, розроблено модель планування траєкторії руху БПЛА, реалізовано програмну систему та проведено її тестування у симуляційному середовищі. Водночас проведене дослідження має певні обмеження, оскільки тестування моделі здійснювалось виключно у середовищі AirSim, а тому питання її ефективності при використанні на реальному БПЛА потребує додаткового дослідження. Зокрема, необхідною є адаптація алгоритму до роботи з фізичними сенсорами, врахування шумів вимірювання, обмежень бортових обчислювальних ресурсів та впливу реальних умов польоту. Перспективами подальшого розвитку роботи є тестування моделі на реальному БПЛА, удосконалення механізмів виявлення перешкод, оптимізація алгоритму локального планування маршруту та розширення можливостей адаптивного керування у складніших середовищах.

Таким чином, у кваліфікаційній роботи було розроблено та досліджено модель планування траєкторії руху БПЛА на основі даних відеоспостережень, яка забезпечує адаптивне коригування маршруту в умовах наявності перешкоди та демонструє працездатність у симуляційному середовищі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Li S., Xu Y., Meng M.Q.-H. Vision-Based UAV Navigation in Dynamic Environments // Sensors. – 2021. – Vol. 21, № 4. – P. 1224.

2. Wu Q., Su Y., Tan W., Zhan R., Liu J., Jiang L. UAV Path Planning Trends from 2000 to 2024: A Bibliometric Analysis and Visualization // Drones. – 2025. – Vol. 9, № 2. – P. 128.

3. Jiang Y., Xu X.-X., Zheng M.-Y., Zhan Z.-H. Evolutionary Computation for Unmanned Aerial Vehicle Path Planning: A Survey // Artificial Intelligence Review. – 2024. – Vol. 57. – P. 267.

4. Sikora T., Papić V. Survey of Path Planning for Aerial Drone Inspection of Multiple Moving Objects // Drones. – 2024. – Vol. 8, № 12. – P. 705.

5. Zhang J., Wang L. Real-Time Depth Estimation and Obstacle Avoidance for UAV Navigation // Sensors. – 2022. – Vol. 22, № 9. – P. 3411.

6. Wang A., Chen H., Liu L., Chen K., Lin Z., Han J., Ding G. YOLOv10: Real-Time End-to-End Object Detection. – 2024.

7. Mei J., Zhu W. BGF-YOLOv10: Small Object Detection Algorithm from Unmanned Aerial Vehicle Perspective Based on Improved YOLOv10 // Sensors. – 2024. – Vol. 24, № 21. – P. 6911.

8. LD-YOLOv10: A Lightweight Target Detection Algorithm for Drone Scenarios Based on YOLOv10 // Electronics. – 2024. – Vol. 13, № 16. – P. 3269.

9. Tahir M., Böling J., Haghbayan M. Computer Vision-Based Approaches for UAV Navigation and Obstacle Avoidance: A Review // Sensors. – 2023. – Vol. 23, № 7. – P. 3564.

10. Wang Y., Zhao Z., Li H. Image Processing Techniques for Autonomous Navigation Systems: Recent Advances and Applications // Applied Sciences. – 2023. – Vol. 13, № 4. – P. 2198.

11. Masek P., Krejci J., Hosek J. Software Architectures for Autonomous Robotic Systems: A Survey // Electronics. – 2022. – Vol. 11, № 18. – P. 2920.

12. Shah S., Dey D., Lovett C., Kapoor A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles // Field and Service Robotics. – 2021 Edition. – Springer. – P. 621–635.

13. Macenski S., Foote T., Gerkey B., Lalancette C., Woodall W. Robot Operating System 2: Design, Architecture, and Uses in the Wild // Science Robotics. – 2022. – Vol. 7, № 66.

14. Racinskis P., Arents J., Greitans M. Constructing Maps for Autonomous Robotics: An Introductory Conceptual Overview // Electronics. – 2023. – Vol. 12, № 13. – Art. 2925. – DOI: 10.3390/electronics12132925.

15. Li Y., Wang L., Ren Y., Chen F., Zhu W. FIImap: Fast Incremental Inflate Mapping for Autonomous MAV Navigation // Electronics. – 2023. – Vol. 12, № 3. – Art. 534. – DOI: 10.3390/electronics12030534.

16. Thrun S., Burgard W., Fox D. Probabilistic Robotics. – Cambridge, MA : MIT Press, 2005. – 647 p. – ISBN 9780262201629.

17. Müller H., Niculescu V., Polonelli T., Magno M., Benini L. Robust and Efficient Depth-based Obstacle Avoidance for Autonomous Miniaturized UAVs. – 2022. – DOI: 10.48550/arXiv.2208.12624.

18. Ruf B., Monka S., Kollmann M., Grinberg M. Real-time On-board Obstacle Avoidance for UAVs Based on Embedded Stereo Vision. – 2018. – DOI: 10.48550/arXiv.1807.06271.

19. Houshangi N., Asghari M., Tavoosi V. Collision Risk Assessment and Safe Navigation for Autonomous UAV Systems // Aerospace. – 2024. – Vol. 11, № 5. – Art. 371.

20. Wang J., Zhao Z., Qu J., Chen X. APPA-3D: An Autonomous 3D Path Planning Algorithm for UAVs in Unknown Complex Environments // Scientific Reports. – 2024. – Vol. 14. – Art. 51286. – DOI: 10.1038/s41598-024-51286-2.

21. Gómez-Arnaldo C., Zamarreño-Suárez M., Pérez-Moreno F., Delgado-Aguilera R. Path Planning for Unmanned Aerial Vehicles in Complex Environments // Drones. – 2024. – Vol. 8, № 7. – Art. 288. – DOI: 10.3390/drones8070288.

22. Tahir M., Böling J., Haghbayan M. Computer Vision-Based Approaches for UAV Navigation and Obstacle Avoidance: A Review // Sensors. – 2023. – Vol. 23, № 7. – Art. 3564. – DOI: 10.3390/s23073564.

23. Zhao F., Li D., Wang Z., Mao J., Wang N. Autonomous Localized Path Planning Algorithm for UAVs Based on TD3 Strategy // Scientific Reports. – 2024. – Vol. 14. – Art. 51349. – DOI: 10.1038/s41598-024-51349-4.

24. He Y., Hou T., Wang M. A New Method for Unmanned Aerial Vehicle Path Planning in Complex Environments // Scientific Reports. – 2024. – Vol. 14. – Art. 60051. – DOI: 10.1038/s41598-024-60051-4.

25. Yang Z., Yang Y., He X., Qi W. Incremental Coverage Path Planning Method for UAV Ground Mapping in Unknown Areas // International Journal of Advanced Robotic Systems. – 2024. – Vol. 21. – DOI: 10.1177/17568293241262323.

26. Coombes M., Fletcher T., Chen W.-H., Liu C. Fixed-Wing UAV Adaptive Flight Control Using Incremental Dynamic Inversion // Control Engineering Practice. – 2024. – Vol. 145. – Art. 106498. – DOI: 10.1016/j.conengprac.2024.106498.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача <u>першого (бакалаврського)</u> рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

<u>БЄЛОУСА ВЛАДИСЛАВА ВІТАЛІЙОВИЧА</u>
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **<u>МОДЕЛЬ ПЛАНУВАННЯ ТРАЄКТОРІЇ РУХУ БПЛА НА ОСНОВІ ДАНИХ ВІДЕОСПОСТЕРЕЖЕНЬ</u>**
керівник роботи **<u>Стрілець Вікторія Євгенівна, доцент зво кафедри комп'ютерних систем і робототехніки, к. т. н.</u>**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101−5/1245 від 28.05.2026 року***.
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
1) вимоги до програмної реалізації моделі автоматизації процесу планування траєкторії руху БПЛА;
2) дані відеоспостережень (зображення) для тестування і аналізу роботи моделі.
4. Перелік питань, які потрібно розробити:
1) аналіз існуючих методів планування траєкторії руху БПЛА та постановка задачі;
2) аналіз методів комп'ютерного зору та алгоритмів локального планування траєкторії БПЛА;
3) розробка моделі планування траєкторії руху БПЛА на основі даних відеоспостережень;
4) розробка алгоритмів виявлення та уникнення перешкод у реальному часі
5) програмна реалізація системи автономної навігації БПЛА у середовищі AirSim.
6) тестування та аналіз роботи розробленої моделі.

5. План роботи

№ з/п	Назви етапів роботи	Строк виконання етапів роботи
1.	Аналіз предметної області та постановка задачі	13.10.2025 – 15.11.2025
2.	Аналіз існуючих методів планування траєкторії руху БПЛА та постановка задачі	16.11.2025 – 14.01.2026
3.	Аналіз методів комп'ютерного зору та алгоритмів локального планування траєкторії БПЛА.	15.01.2026 – 31.01.2026
4.	Розробка моделі планування траєкторії руху БПЛА на основі даних відеоспостережень.	01.02.2026 – 14.02.2026
5.	Розробка алгоритмів виявлення та уникнення перешкод у реальному часі.	15.02.2026 – 10.03.2026
6.	Програмна реалізація системи автономної навігації БПЛА у середовищі AirSim.	11.03.2026 – 24.03.2026
7.	Тестування та аналіз роботи розробленої моделі.	25.03.2026 – 14.04.2026
8.	Підготовка рекомендацій та вдосконалення моделі	15.04.2026 – 30.04.2026
9.	Оформлення пояснювальної записки	01.05.2026 – 20.05.2026
10.	Представлення роботи керівнику та рецензенту	20.05.2026 – 30.05.2026

6. Дата видачі завдання ***13 жовтня 2025 року***

Здобувач вищої освіти _____ Владислав БЄЛОУС

Керівник роботи _____ Вікторія СТРІЛЕЦЬ

Додаток Б

Затверджую

_____

«_____»_____2026 р.

Технічне завдання
на розробку програмного виробу
«Модель планування траєкторії руху БПЛА на основі даних відеоспостережень»

Назва розділу	Назва і зміст підрозділу
1. Вступ	1.1. Назва програмного виробу – Модель планування траєкторії руху БПЛА на основі даних відеоспостережень 1.2. Галузь застосування – Інформаційні технології, системи автономної навігації, робототехніка, комп'ютерний зір, безпілотні літальні апарати та інтелектуальні системи керування рухом.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101−5/1245 від «28» травня 2026 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3. Призначення розробки	3.1. Мета розробки: створення моделі планування траєкторії руху безпілотного літального апарата на основі даних відеоспостережень, яка забезпечує адаптивне коригування маршруту польоту, виявлення та уникнення перешкод у реальному часі, а також підвищення автономності навігації БПЛА в умовах змінного середовища. 3.2. Призначення розробки: полягає у створенні програмної моделі автономної навігації БПЛА, яка на основі depth-даних відеоспостережень дозволяє аналізувати навколишнє середовище, будувати локальну траєкторію руху, виконувати обхід перешкод та забезпечувати досягнення заданих цілей у симуляційному середовищ. 3.3. Початкові дані для розробки: методи та моделі планування траєкторії руху БПЛА, алгоритми локального планування маршруту та уникнення перешкод, методи комп'ютерного зору для аналізу відеоданих і depth-зображень, дані depth-камер для оцінювання відстані до перешкод, вимоги до програмної реалізації систем автономної навігації БПЛА, середовище моделювання AirSim для тестування алгоритмів польоту.

4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: • Отримання depth-даних із фронтальної та нижньої камери БПЛА. • Аналіз відеоданих для визначення перешкод у середовищі. • Побудова локальної карти середовища на основі depth-зображення. • Планування траєкторії руху БПЛА із використанням алгоритму A*. • Автоматичне уникнення перешкод під час руху. • Адаптивне коригування напрямку руху залежно від навколишнього середовища. • Адаптивне керування висотою польоту БПЛА. • Автоматичне проходження заданих цілей. • Візуалізація траєкторії руху та перешкод. • Тестування моделі у середовищі AirSim 4.2. Вимоги до надійності: Забезпечення стабільної роботи системи планування траєкторії БПЛА при зміні перешкоди, шумів depth-даних та короткочасній втраті сенсорної інформації. 4.3. Вимоги до умов експлуатації: Програмний виріб повинен функціонувати у симуляційному середовищі AirSim на персональному комп'ютер 4.4. Вимоги до складу і параметрів технічних засобів: Персональний комп'ютер із встановленими: середовищем AirSim, мовою програмування Python 3.x, бібліотеками NumPy, Matplotlib, AirSim API 4.5. Вимоги до інформаційної та програмної сумісності: Забезпечення сумісності з Python 3.x, AirSim API, а також підтримка роботи із depth-даними відеоспостережень. 4.6. Вимоги до маркування та упаковки: відсутні. 4.7. Вимоги до транспортування і зберігання: відсутні. 4.8. Спеціальні вимоги: Можливість подальшого вдосконалення алгоритмів планування траєкторії та адаптації моделі до реального БПЛА.
5. Вимоги до програмної документації .	Програмною документацією до виробу «Модель планування траєкторії руху БПЛА на основі даних відеоспостережень» вважати: • Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). • Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).

<table>
<tr><td></td><td colspan="2">• Опис програмного виробу (представити у розділі 3 пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td>6. Техніко–економічні показники</td><td colspan="2">В даному розділі можуть бути представлені:
• Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи).
• Результати тестування та оцінки ефективності моделі планування траєкторії руху БПЛА (у вигляді підрозділу 3.6 пояснювальної записки до кваліфікаційної роботи).
• Опис програмного виробу (представити у розділі 3 пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td rowspan="11">7. Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>13.10.2025 – 15.11.2025</td><td>Аналіз предметної області та постановка задачі</td></tr>
<tr><td>16.11.2025 – 14.01.2026</td><td>Аналіз підходів, методів і моделей планування траєкторії руху БПЛА</td></tr>
<tr><td>15.01.2026 – 31.01.2026</td><td>Огляд мов програмування, засобів моделювання та методів комп’ютерного зору</td></tr>
<tr><td>01.02.2026 – 14.02.2026</td><td>Розробка моделі планування траєкторії руху БПЛА на основі даних відеоспостережень</td></tr>
<tr><td>15.02.2026 – 10.03.2026</td><td>Програмна реалізація моделі планування траєкторії руху БПЛА</td></tr>
<tr><td>11.03.2026 – 24.03.2026</td><td>Тестування моделі планування траєкторії руху БПЛА</td></tr>
<tr><td>25.03.2026 – 14.04.2026</td><td>Дослідження якості роботи моделі та аналіз результатів</td></tr>
<tr><td>15.04.2026 – 30.04.2026</td><td>Формування рекомендацій щодо вдосконалення моделі</td></tr>
<tr><td>01.05.2026 – 20.05.2026</td><td>Оформлення пояснювальної записки</td></tr>
<tr><td>20.05.2026 – 30.05.2026</td><td>Представлення роботи керівнику та рецензенту</td></tr>
</table>

8. Порядок контролю і приймання	• Перевірку ходу розробки програмного виробу керівнику робіт виконувати раз на 3 тижні. • Випробування програмного виробу провести відповідно до програми та методики випробувань у симуляційному середовищі AirSim. • Захист розробленої моделі провести на засіданні Атестаційної комісії. • Пояснювальну записку подати в паперовому та електронному вигляді.

Виконавець

студент групи К1–41

Бєлоус В.В.

Замовник

доцент зво кафедри

комп'ютерних систем та робототехніки

Стрілець В.Є.

Додаток В

Програма і методика випробувань програмного виробу «Модель комп'ютеризованої системи контролю доступу до офісного приміщення через Інтернет»

1. Об'єкт випробувань

Назва програмного виробу : «Модель планування траєкторії руху БПЛА на основі даних відеоспостережень».
Галузь застосування : Інформаційні технології: системи автономної навігації, робототехніка, комп'ютерний зір та безпілотні літальні апарати.
Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації моделі заявленим функціональним можливостям, наведеним у технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

- **Підстава для проведення випробувань:** наказ по університету про створення атестаційної комісії з проведення захисту кваліфікаційних робіт бакалаврів від **28 травня 2026 року № 4101−5/1245.**
- **Місце і тривалість випробувань:** комп'ютерний клас кафедри комп'ютерних систем та робототехніки, **10−20 травня 2026 р.**
- **Обсяг випробувань:** випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.
- **Організації, які беруть участь у випробуваннях:** атестаційна комісія за участю розробника та керівника кваліфікаційної роботи.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

- Отримання та обробка depth-даних відеоспостережень;
- Аналіз перешкод у середовищі;
- Побудова локальної карти перешкод;
- Планування траєкторії руху БПЛА із використанням алгоритму A*;
- Автоматичне уникнення перешкод;
- Адаптивне керування напрямком руху та висотою польоту;
- Проходження заданих цілей;
- Сумісність із Python 3.x, AirSim API, NumPy, Matplotlib;
- Забезпечення стабільної роботи алгоритму;
- Вимоги до складу і параметрів технічних засобів;
- Вимоги до маркування та упаковки (не висуваються);
- Вимоги до транспортування і зберігання (не висуваються).

Спеціальні вимоги (не пред'являються).

5. Вимоги до програмної документації

Програмною документацією до виробу «Модель планування траєкторії руху БПЛА на основі даних відеоспостережень» вважати:

- Справжнє технічне завдання на розробку програмного виробу (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);
- Програму і методику випробувань розробленого програмного виробу (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);
- Опис програмного виробу (представити у розділі 3 пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1. Засоби випробувань

Для проведення випробувань необхідний:

- персональний комп'ютер із встановленим середовищем моделювання AirSim;
- Python 3.x;
- бібліотеки AirSim API, NumPy, Matplotlib;
- середовище моделювання польоту БПЛА Microsoft AirSim.

6.2. Порядок проведення випробувань

Випробування проводяться у два етапи:

- ознайомчий (1–й етап);
- випробування програмного виробу (2–й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає:

Перевірку комплектності програмної документації

Перевірка комплектності складу програмної документації здійснюється за критерієм наявності документації, зазначеної в технічному завданні.

Перевірку комплектності складу технічних і програмних засобів

Перевіряється наявність:

- середовища AirSim;
- інтерпретатора Python 3.x;
- необхідних бібліотек;
- коректного підключення API до середовища моделювання.

Методику проведення перевірок на 1 етапі випробувань

Якість програмної документації перевіряється на відповідність вимогам ДСТУ.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає:

- перевірку відповідності технічних характеристик моделі вимогам технічного завдання;
- перевірку ступеня виконання функціональних вимог до програмного виробу;

- перевірку роботи алгоритмів планування траєкторії та уникнення перешкод;
- перевірку стабільності проходження цілі;
- перевірку роботи системи адаптивного керування висотою.

Програма працює відповідно до умов експлуатації персонального комп'ютера у симуляційному середовищі AirSim.
Для роботи необхідний персональний комп'ютер із встановленим середовищем AirSim та Python 3.x.

Порядок проведення випробувань

Тест 1

Перевірка руху БПЛА до цілі без перешкод.
БПЛА виконує автономний політ до заданої точки маршруту.
Тест вважається пройденим, якщо:

- БПЛА успішно досягає цільової точки;
- траєкторія руху є стабільною;
- не виникає помилок виконання.



Рисунок В.1. – Тест 1: стартова позиція БПЛА.



Рисунок В.2. – Тест 1: кінцева позиція БПЛА.

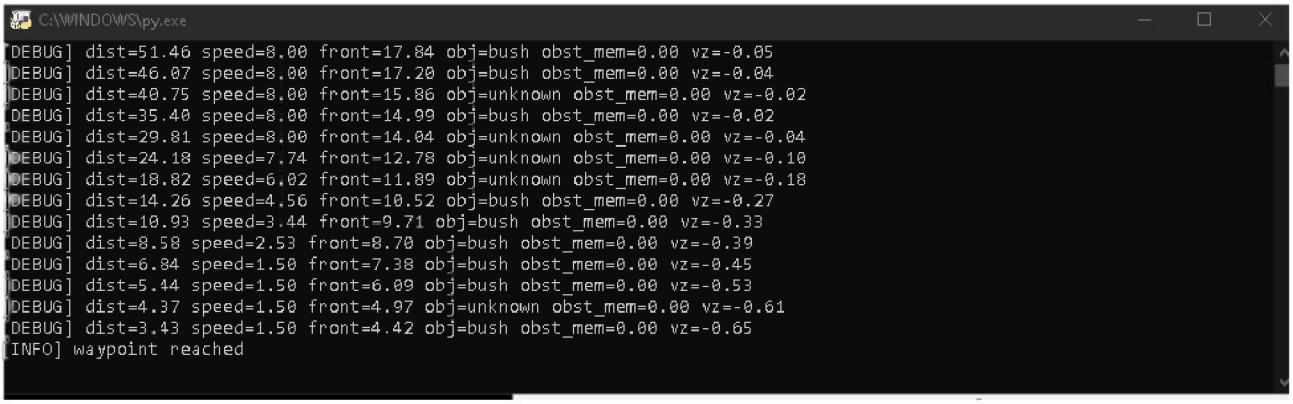


Рисунок В.3. – Тест 1: вікно налаштування польоту БПЛА.

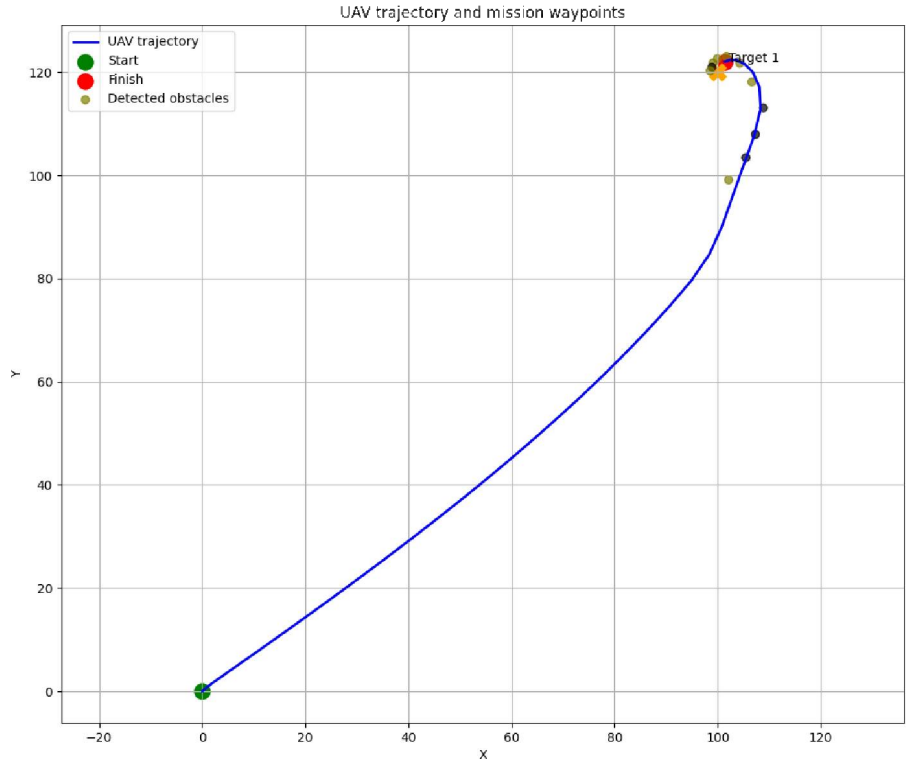


Рисунок В.4. – Тест 1: траєкторія польоту БПЛА у середовищі.

Тест 2

Перевірка уникнення перешкод під час руху БПЛА.

На траєкторії польоту розміщується об’єкти.

Під час польоту модель повинна:

- виявити перешкоду за depth-даними;
- скоригувати напрямок руху;
- побудувати локальний маршрут обходу;
- продовжити рух до цілі.

Тест вважається пройденим, якщо:

- зіткнення з об’єктом не відбулося;
- маршрут успішно перебудовано;
- цільова точка досягнута.

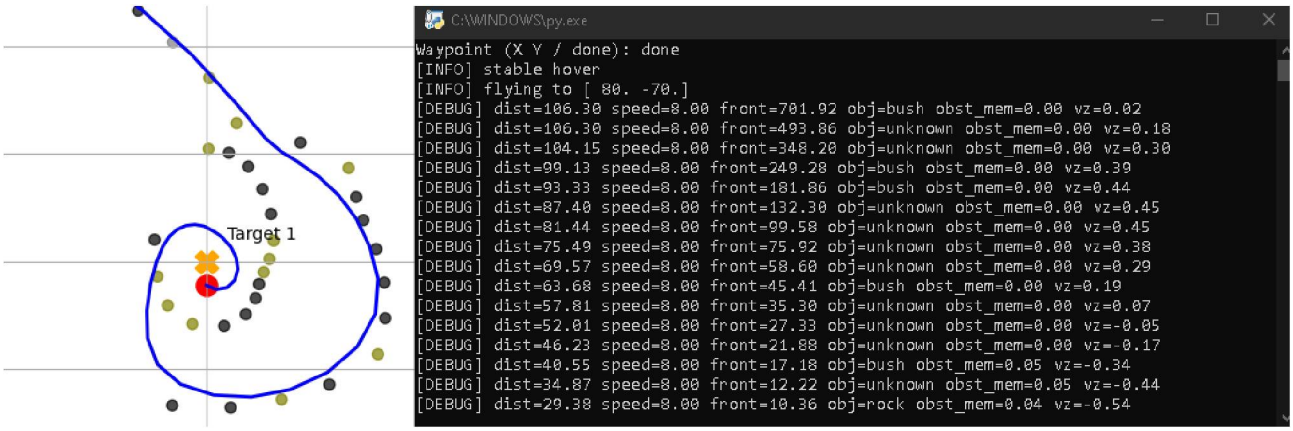


Рисунок В.5 – Тест 2: успішне досягнення цілі.

Тест 3

Під час польоту перевіряється здатність системи коригувати висоту БПЛА залежно від об’єкту.

Тест вважається пройденим, якщо:

- зміна висоти виконується автоматично;
- перешкода успішно обходиться;
- рух БПЛА залишається стабільним.

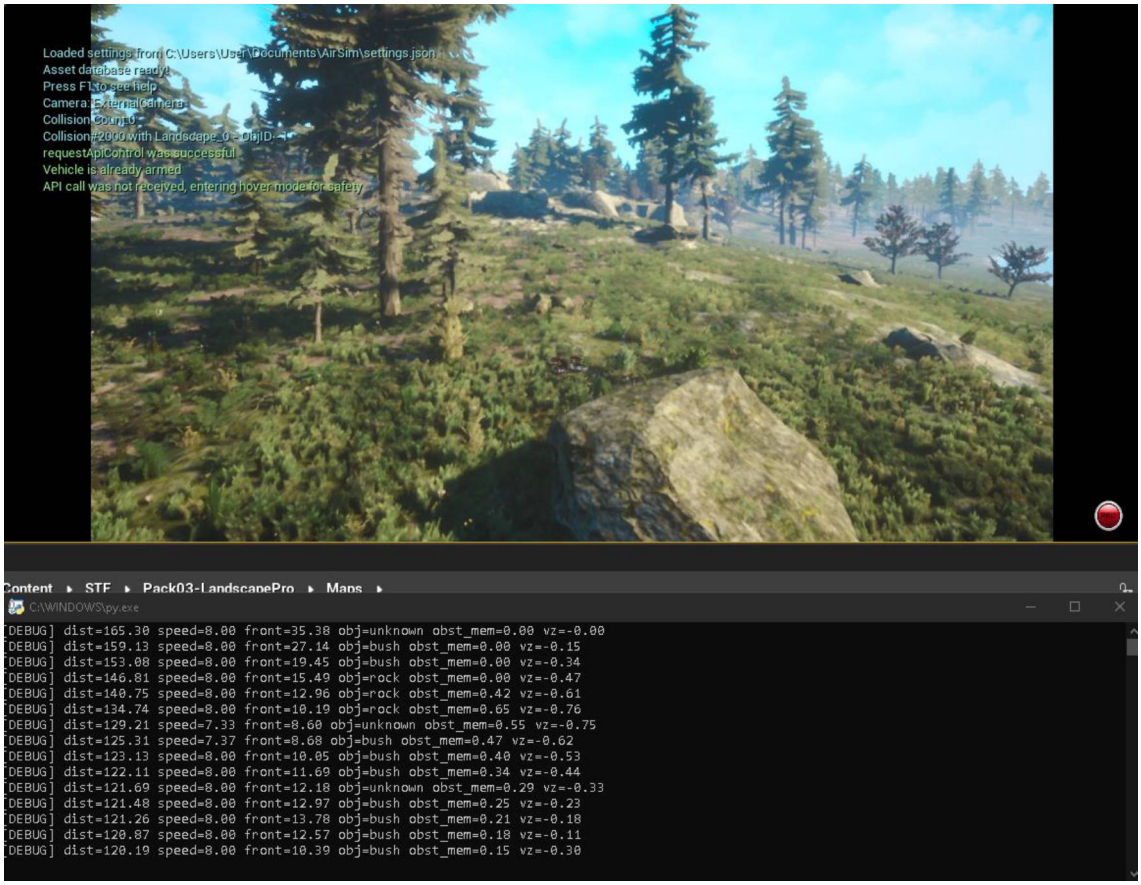


Рисунок В.6 – Тест 3: успішний проліт над каменем.

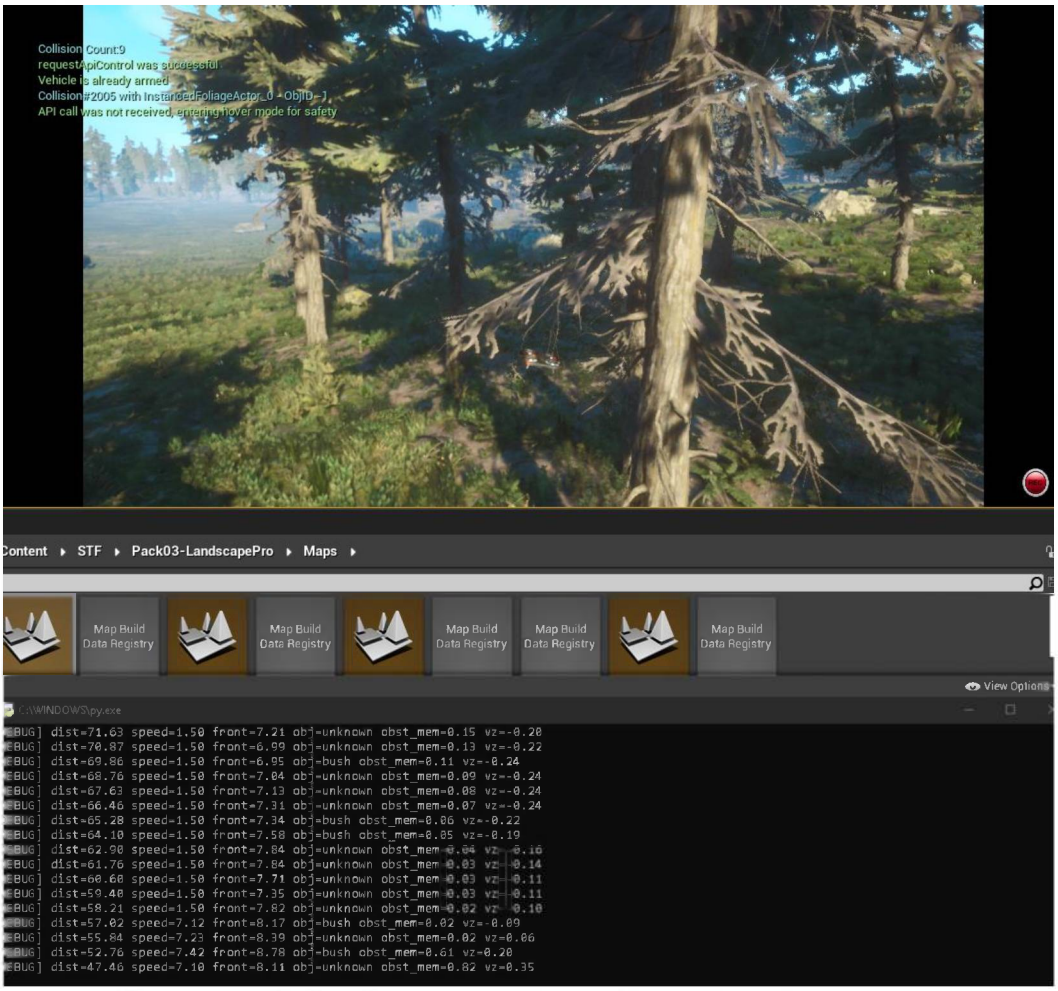


Рисунок В.7 – Тест 3: успішний проліт під гілкою.

Тест вважається пройденим, якщо відбуваються вказані операції і їх відображення у програмному продукті.

Висновки: Тест 1, Тест 2 та Тест 3 успішно пройшли випробування. Розроблена модель планування траєкторії руху БПЛА функціонує відповідно до вимог технічного завдання.

Виконавець: студент групи КІ−41, Бєлоус В.В. _____

Додаток Г

Програмний код реалізації моделі

```
import airsim
import numpy as np
import time
import math
import heapq
import matplotlib.pyplot as plt

# ========================
# CONFIG
# ========================
TARGET_RADIUS = 2.5
SPEED = 8.0

CONTROL_HZ = 20
DT = 80 / CONTROL_HZ

DESIRED_AGL = 3.5
Z_KP = 0.6
MAX_VZ = 2.0

AVOID_DIST = 12.0
SLOW_DIST = 25.0
CRITICAL_DIST = 8.0
AVOID_GAIN = 4.5

SIDE_SAFE_DIST = 3.0
SIDE_GAIN = 1.0

SMOOTH_ALPHA = 0.58
VEC_MEMORY = np.array([0.0, 0.0, 0.0])

GRID_SIZE = 20
OBSTACLE_TH = 6.0

FINAL_APPROACH_DIST = 6.0

OBSTACLE_MEMORY = 0.0
OBSTACLE_DECAY = 0.85
OBSTACLE_TRIGGER = 7.0
OBSTACLE_CLEAR = 12.0

FRONT_FILTER_ALPHA = 0.7
front_filtered = 999

YAW_MEMORY = 0.0
YAW_ALPHA = 0.99

trajectory = []
obstacle_points = []
obstacle_types = []
mission_points = []

UP_AVOID_GAIN = 1.5
DOWN_LIMIT = 1.0
UP_LIMIT = 6.0

agl_filtered = DESIRED_AGL
```

```
# =========================
client = airsim.MultirotorClient()
client.confirmConnection()
client.enableApiControl(True)
client.armDisarm(True)

home_position = None

# =========================
def pos():
    p = client.getMultirotorState().kinematics_estimated.position
    return np.array([p.x_val, p.y_val, p.z_val])


# =========================
def classify_obstacle():
    try:
        img = client.simGetImages([
            airsim.ImageRequest("front_center",
airsim.ImageType.DepthPerspective, True, False)
        ])[0]

        if len(img.image_data_float) == 0:
            return "unknown", 1.0

        d = np.array(img.image_data_float, dtype=np.float32).reshape(img.height,
img.width)

        roi = d[
            img.height//4: 3*img.height//4,
            img.width//4: 3*img.width//4
        ]

        valid = roi[roi < 50]
        if len(valid) < 50:
            return "none", 1.0

        mask = roi < np.percentile(roi, 35)
        ys, xs = np.where(mask)

        if len(xs) < 30:
            return "none", 1.0

        h_img, w_img = roi.shape

        height_span = (np.max(ys) - np.min(ys)) / h_img
        width_span = (np.max(xs) - np.min(xs)) / w_img
        density = np.mean(mask)
        depth_mean = np.mean(valid)

        bottom_bias = np.mean(ys > h_img * 0.6)

        center_col = roi[:, w_img // 2]
        grad = np.std(center_col)

        if density > 0.55 and grad < 1.5 and bottom_bias > 0.6:
            return "ground", 0.1

        if height_span > 0.55 and width_span < 0.5 and density < 0.45:
            return "tree", 1.4

        if width_span > 0.45 and density > 0.35:
            return "bush", 0.8
```

```
        if density > 0.2 and depth_mean < 12:
            return "rock", 1.0

        return "unknown", 1.0

    except:
        return "unknown", 1.0


# =========================
def ground_distance():
    global agl_filtered

    try:
        img = client.simGetImages([
            airsim.ImageRequest("bottom_center",
airsim.ImageType.DepthPerspective, True, False)
        ])[0]

        if len(img.image_data_float) == 0:
            return agl_filtered

        d = np.array(img.image_data_float, dtype=np.float32)
        d = d.reshape(img.height, img.width)

        center = d[
            img.height//2 - 5: img.height//2 + 5,
            img.width//2 - 5: img.width//2 + 5
        ]

        val = float(np.median(center))

        if np.isnan(val) or val < 0.2 or val > 50:
            val = agl_filtered

        agl_filtered = 0.8 * agl_filtered + 0.2 * val
        return agl_filtered

    except:
        return agl_filtered


# =========================
def front_depth():
    global front_filtered

    try:
        img = client.simGetImages([
            airsim.ImageRequest("front_center",
airsim.ImageType.DepthPerspective, True, False)
        ])[0]

        if len(img.image_data_float) == 0:
            return front_filtered

        d = np.array(img.image_data_float, dtype=np.float32).reshape(img.height,
img.width)

        w = img.width

        center = d[:, w//2 - 20:w//2 + 20]
        left   = d[:, w//4 - 15:w//4 + 15]
```

```
        right  = d[:, 3*w//4 - 15:3*w//4 + 15]

        c = float(np.percentile(center, 5))
        l = float(np.percentile(left, 5))
        r = float(np.percentile(right, 5))

        val = min(c, l, r)

        if np.isnan(val) or val < 0.2 or val > 80:
            val = front_filtered

        front_filtered = FRONT_FILTER_ALPHA * front_filtered + (1 -
FRONT_FILTER_ALPHA) * val
        return front_filtered

    except:
        return front_filtered


# =========================
def obstacle_risk(d_front):
    if d_front > 18:
        return 0.0
    if d_front < 6:
        return 1.0
    return (18 - d_front) / 12.0


# =========================
def side_depth():
    try:
        img = client.simGetImages([
            airsim.ImageRequest("front_center",
airsim.ImageType.DepthPerspective, True, False)
        ])[0]

        if len(img.image_data_float) == 0:
            return 999, 999

        d = np.array(img.image_data_float, dtype=np.float32).reshape(img.height,
img.width)

        w = d.shape[1]
        left = d[:, w//6:w//3]
        right = d[:, 2*w//3:5*w//6]

        return float(np.median(left)), float(np.median(right))

    except:
        return 999, 999


# =========================
def build_local_grid():
    img = client.simGetImages([
        airsim.ImageRequest("front_center", airsim.ImageType.DepthPerspective,
True, False)
    ])[0]

    d = np.array(img.image_data_float, dtype=np.float32)
    d = d.reshape(img.height, img.width)

    grid = np.zeros((GRID_SIZE, GRID_SIZE))
```

```
    for i in range(GRID_SIZE):
        for j in range(GRID_SIZE):
            x = int(i * img.height / GRID_SIZE)
            y = int(j * img.width / GRID_SIZE)
            if d[x, y] < OBSTACLE_TH:
                grid[i, j] = 1

    return grid


# =========================
def astar(grid, start, goal):
    def h(a, b):
        return abs(a[0] - b[0]) + abs(a[1] - b[1])

    open_set = []
    heapq.heappush(open_set, (0, start))

    came_from = {}
    g = {start: 0}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            break

        x, y = current

        for dx, dy, cost in [
            (-1,0,1),(1,0,1),(0,-1,1),(0,1,1),
            (-1,-1,1.4),(-1,1,1.4),(1,-1,1.4),(1,1,1.4)
        ]:
            nx, ny = x + dx, y + dy

            if 0 <= nx < GRID_SIZE and 0 <= ny < GRID_SIZE:
                if grid[nx][ny] == 1:
                    continue

                ng = g[current] + cost

                if (nx, ny) not in g or ng < g[(nx, ny)]:
                    g[(nx, ny)] = ng
                    came_from[(nx, ny)] = current
                    heapq.heappush(open_set, (ng + h((nx, ny), goal), (nx, ny)))

    path = []
    node = goal

    while node in came_from:
        path.append(node)
        node = came_from[node]

    path.reverse()
    return path


# =========================
def path_to_direction(path):
    if len(path) < 3:
        return np.array([1.0, 0.0])
```

```
    center = np.array([GRID_SIZE//2, GRID_SIZE//2])
    p = np.array(path[min(4, len(path)-1)])

    d = p - center
    return d / (np.linalg.norm(d) + 1e-6)


# ========================
def yaw_to(direction):
    global YAW_MEMORY

    n = np.linalg.norm(direction)
    if n < 1e-6:
        return

    # normalize
    direction = direction / n

    # yaw from velocity vector
    desired_yaw = math.degrees(
        math.atan2(direction[1], direction[0])
    )

    # shortest rotation path
    diff = desired_yaw - YAW_MEMORY
    diff = (diff + 180) % 360 - 180

    # MUCH faster yaw tracking
    max_yaw_step = 30.0   # degrees per control cycle

    diff = np.clip(
        diff,
        -max_yaw_step,
        max_yaw_step
    )

    YAW_MEMORY += diff * YAW_ALPHA

    # normalize
    YAW_MEMORY = (
        YAW_MEMORY + 180
    ) % 360 - 180

    # FAST yaw update
    client.rotateToYawAsync(
        YAW_MEMORY,
        timeout_sec=0.08
    )


# ========================
def takeoff():
    client.takeoffAsync().join()
    time.sleep(1)

    p = pos()
    client.moveToPositionAsync(p[0], p[1], -DESIRED_AGL, 2).join()
    time.sleep(1)

    print("[INFO] stable hover")


# ========================
```

```
def fly(target):
    local_agl_target = DESIRED_AGL
    global VEC_MEMORY, VEL_MEMORY, OBSTACLE_MEMORY

    print(f"[INFO] flying to {target}")

    while True:

        trajectory.append(pos()[:2].copy())
        p = pos()

        to_target = target[:2] - p[:2]
        dist = np.linalg.norm(to_target)

        if dist < TARGET_RADIUS:
            print("[INFO] waypoint reached")
            break

        direction_xy = to_target / (np.linalg.norm(to_target) + 1e-6)
        direction = np.array([direction_xy[0], direction_xy[1], 0.0])

        # =========================
        # SPEED CONTROL
        # =========================
        speed = SPEED

        if dist < SLOW_DIST:
            speed = max(SPEED * (dist / SLOW_DIST), 2.5)

        if dist < FINAL_APPROACH_DIST:
            speed *= 0.5

        # =========================
        # SENSORS
        # =========================
        d_front = front_depth()
        brake = 0.0

        if d_front < 10.0:
            brake = (10.0 - d_front) / 10.0
            speed *= (1.0 - 0.6 * brake)
        obstacle_type, obstacle_weight = classify_obstacle()
        save_obstacle_point(p, obstacle_type, d_front)

        if d_front < CRITICAL_DIST:
            speed = 1.5
            vx = 1.5
            vy = 1.5
        if d_front < 3.0:
            # жорсткий відхід назад + вверх
            move_dir = np.array([-direction[0], -direction[1], 0.8])
            speed = 3.0
        elif d_front < AVOID_DIST:
            speed *= 1.0

        grid = build_local_grid()
        start = (GRID_SIZE // 2, GRID_SIZE // 2)

        goal = (
            GRID_SIZE // 2 - 1,
            GRID_SIZE // 2 + int(np.clip(to_target[0], -GRID_SIZE // 2, GRID_SIZE
// 2))
        )
```

```
        path = astar(grid, start, goal)
        astar_dir_2d = path_to_direction(path)
        astar_dir = np.array([astar_dir_2d[0], astar_dir_2d[1], 0.0])

        direction = 0.7 * direction + 0.3 * astar_dir
        direction /= (np.linalg.norm(direction) + 1e-6)

        left_dist, right_dist = side_depth()
        up_down_bias = 0.0

        # ========================
        # HEIGHT FILTER (FIXED ROI)
        # ========================
        img = client.simGetImages([
            airsim.ImageRequest("front_center",
airsim.ImageType.DepthPerspective, True, False)
        ])[0]

        d = np.array(img.image_data_float, dtype=np.float32).reshape(img.height,
img.width)

        roi = d[
            img.height // 4: 3 * img.height // 4,
            img.width // 4: 3 * img.width // 4
        ]

        h_img, w_img = roi.shape

        mask = roi < 8.0
        ys, xs = np.where(mask)

        obstacle_signal = 0.0

        if len(xs) > 30:

            mean_y = np.mean(ys)
            height_level = mean_y / h_img

            is_ground_clutter = height_level > 0.72
            small_low_objects = (np.mean(mask) < 0.18 and height_level > 0.6)

            if not (is_ground_clutter or small_low_objects):
                obstacle_signal = obstacle_risk(d_front) * obstacle_weight

        OBSTACLE_MEMORY *= OBSTACLE_DECAY
        OBSTACLE_MEMORY = max(OBSTACLE_MEMORY, obstacle_signal)

        avoid = np.zeros(3)
        avoid_z = 0.0  # FIX: завжди існує

        if OBSTACLE_MEMORY > 0.2:

            side_2d = np.array([-direction[1], direction[0]])

            avoid_xy = side_2d * (AVOID_GAIN * 1.0) * OBSTACLE_MEMORY

            if d_front < 8.0:
                avoid_z = UP_AVOID_GAIN * OBSTACLE_MEMORY

            avoid = np.array([avoid_xy[0], avoid_xy[1], avoid_z])

        raw_move = direction + avoid
```

```
raw_move /= (np.linalg.norm(raw_move) + 1e-6)

VEC_MEMORY = SMOOTH_ALPHA * VEC_MEMORY + (1 - SMOOTH_ALPHA) * raw_move
move_dir = VEC_MEMORY / (np.linalg.norm(VEC_MEMORY) + 1e-6)

# ========================
# YAW FIX (INDENTATION FIXED)
# ========================
future_dir = move_dir
future_dir /= (np.linalg.norm(future_dir) + 1e-6)

yaw_to(future_dir)

# ========================
# VELOCITY
# ========================
vx = move_dir[0] * speed
vy = move_dir[1] * speed
vz_cmd = move_dir[2] * speed

ground = ground_distance()

# ========================
# AGL TARGET (NOT UE4 Z)
# ========================
desired_agl = DESIRED_AGL

if OBSTACLE_MEMORY > 0.2:

    # obstacle in front
    if d_front < 8.0:

        # detect obstacle vertical position
        mean_y = np.mean(ys) / h_img if len(xs) > 20 else 0.5

        # object upper in image -> go UNDER
        if mean_y < 0.35:

            target_agl = max(
                DOWN_LIMIT,
                DESIRED_AGL - 1.5
            )

        # object lower in image -> go ABOVE
        elif mean_y > 0.60:

            target_agl = min(
                UP_LIMIT,
                DESIRED_AGL + 2.0
            )

        else:
            target_agl = DESIRED_AGL

        desired_agl = target_agl

# ========================
# ALTITUDE PID FROM GROUND
# ========================
base_vz = np.clip(
    Z_KP * (ground - desired_agl),
    -MAX_VZ,
    MAX_VZ
```

```
        )

        MIN_AGL = 0.8

        if ground < MIN_AGL:
            base_vz = abs(base_vz)  # тільки вгору

        vz = base_vz

        vz = np.clip(vz, -MAX_VZ, MAX_VZ)

        print(
            f"[DEBUG] dist={dist:.2f} "
            f"speed={speed:.2f} "
            f"front={d_front:.2f} "
            f"obj={obstacle_type} "
            f"obst_mem={OBSTACLE_MEMORY:.2f} "
            f"vz={vz:.2f}"
        )

        client.moveByVelocityAsync(vx, vy, vz + vz_cmd, 40 / CONTROL_HZ)
        time.sleep(0.001 / CONTROL_HZ)

# =========================
def input_waypoints():
    waypoints = []
    while True:
        raw = input("Waypoint (X Y / done): ").strip()
        if raw.lower() == "done":
            break
        try:
            x, y = map(float, raw.split())
            waypoints.append(np.array([x, y]))
        except:
            print("[ERROR] format: X Y")
    return waypoints

# =========================
def save_obstacle_point(drone_pos, obstacle_type, d_front):

    if d_front > 15:
        return

    yaw_rad = math.radians(YAW_MEMORY)

    obstacle_x = drone_pos[0] + math.cos(yaw_rad) * d_front
    obstacle_y = drone_pos[1] + math.sin(yaw_rad) * d_front

    obstacle_points.append([obstacle_x, obstacle_y])
    obstacle_types.append(obstacle_type)


# =========================
def plot_trajectory():

    plt.figure(figsize=(12, 10))

    # =========================
    # UAV TRAJECTORY
    # =========================
    if len(trajectory) > 0:

        traj = np.array(trajectory)
```

```
    plt.plot(
        traj[:, 0],
        traj[:, 1],
        'b-',
        linewidth=2,
        label='UAV trajectory'
    )

    # start
    plt.scatter(
        traj[0, 0],
        traj[0, 1],
        c='green',
        s=150,
        label='Start'
    )

    # finish
    plt.scatter(
        traj[-1, 0],
        traj[-1, 1],
        c='red',
        s=150,
        label='Finish'
    )

# =========================
# WAYPOINTS / TARGETS
# =========================
if len(mission_points) > 0:

    for i, wp in enumerate(mission_points):

        plt.scatter(
            wp[0],
            wp[1],
            c='orange',
            s=180,
            marker='X'
        )

        plt.text(
            wp[0] + 2,
            wp[1] + 2,
            f'Target {i+1}',
            fontsize=10
        )

# =========================
# OBSTACLES
# =========================
if len(obstacle_points) > 0:

    obstacles = np.array(obstacle_points)

    colors = []

    for obj_type in obstacle_types:

        if obj_type == "tree":
            colors.append("green")
```

```
            elif obj_type == "rock":
                colors.append("gray")

            elif obj_type == "bush":
                colors.append("olive")

            else:
                colors.append("black")

        plt.scatter(
            obstacles[:, 0],
            obstacles[:, 1],
            c=colors,
            s=40,
            alpha=0.7,
            label='Detected obstacles'
        )

    # =========================
    # STYLE
    # =========================
    plt.grid(True)
    plt.axis("equal")

    plt.xlabel("X")
    plt.ylabel("Y")

    plt.title("UAV trajectory and mission waypoints")
    plt.legend()

    plt.show()

# =========================
if __name__ == "__main__":

    client.enableApiControl(True)
    client.armDisarm(True)

    mission = input_waypoints()
    mission_points = mission.copy()
    if len(mission) == 0:
        exit(0)

    takeoff()

    for wp in mission:
        fly(wp)

    client.hoverAsync().join()
    time.sleep(0.5)

    client.landAsync().join()
    client.armDisarm(False)
    client.enableApiControl(False)
    plot_trajectory()
```