

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

« ____ » _____ 20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Перехоплення БПЛА іншим БПЛА в симуляції

**з використанням комп'ютерного зору та моделей прогнозування
траєкторій»**

Виконала:

студентка IV курсу, групи КІ-11

Польнікова Поліна Андріївна _____

Керівник:

асистент кафедри штучного інтелекту, к.т.н.,

Осауленко Вячеслав Миколайович _____

Консультант з економічного розділу:

доцент кафедри ЕК ФММ, к.е.н., доцент,

Рощина Надія Василівна _____

Консультант з нормоконтролю:

фахівець першої категорії кафедри ШІ, к.т.н., доцент,

Комариста Богдана Миколаївна _____

Рецензент:

доцент кафедри ММАД ФТІ, к.т.н, доцент,

Хайдуров Владислав Володимирович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студентка _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«15» січня 2025 р.

ЗАВДАННЯ
на дипломну роботу студентці
Польніковій Поліні Андріївні

1. Тема роботи «Перехоплення БПЛА іншим БПЛА в симуляції з використанням комп'ютерного зору та моделей прогнозування траєкторій», керівник роботи Осауленко В'ячеслав Миколайович, асистент кафедри штучного інтелекту, затверджені наказом по НН ІПСА від «26» травня 2025 р. № 1759-с.
2. Термін подання студентом роботи «10» червня 2025 року.
3. Вихідні дані до роботи: вихідний відеопотік із камери дрона-перехоплювача, програмне середовище: ROS 2 Humble + Ubuntu 22.04 LTS, Unity HDRP 6000.0.24f1, пакети RosSharp 1.1.0; початкові налаштування Unscented Kalman Filter; мережеві адреси host-only та bridge із відкритим портом 9090, конфігурацію SSH-інтерпретера PyCharm і спільні каталоги sf_data/data_shared та sf_ros, у яких зберігаються графіки, відеофайли та файли інтерфейсів ROS 2 (.msg, .srv, .action) для індивідуальних типів даних.
4. Зміст роботи: дослідження методів візуального виявлення та прогнозування траєкторії об'єкта; розробка симуляційного середовища Unity + ROS 2; побудова модуля комп'ютерного зору на основі оператора Sobel і

математичних моментів; реалізація Unscented Kalman Filter для згладжування вимірювань і прогнозу позиції та орієнтації; проектування модульної архітектури з патернами «Стратегія», «Спостерігач», «Посередник»; експериментальне моделювання перехоплення за двома підходами: прямим наведенням на еталонні координати БПЛА та наведенням на поточний центроїд, отриманий через CV-модуль; аналіз точності (MSE, мінімальна відстань); виконання функціонально-вартісного аналізу та оцінка ефективності рішення.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): графіки: порівняння істинних та виміряних координат, MSE залежно від кроку прогнозу, 3-D траєкторії дрона й цілі; фрагменти коду Python.

6. Консультанти розділів роботи

<i>Розділ</i>	<i>Прізвище, ініціали та посада консультанта</i>	<i>Підпис, дата</i>	
		<i>завдання видав</i>	<i>завдання прийняв</i>
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

<i>№ з/п</i>	<i>Назва етапів виконання дипломної роботи</i>	<i>Термін виконання етапів роботи</i>	<i>Примітка</i>
1	Огляд літератури за темою	21.04.2025	Виконано
2	Підготовка першого розділу	28.04.2025	Виконано
3	Підготовка другого розділу	05.05.2025	Виконано
4	Розробка програмного продукту	12.05.2025	Виконано
5	Підготовка третього розділу	19.05.2025	Виконано
6	Підготовка економічної частини	26.05.2025	Виконано
7	Оформлення розділів дипломної роботи	02.06.2025	Виконано
8	Оформлення дипломної роботи	11.06.2025	Виконано
9	Підготовка презентації доповіді	11.06.2025	Виконано

Студент

Поліна ПОЛЬНІКОВА

Керівник

Вячеслав ОСАУЛЕНКО

РЕФЕРАТ

Дипломна робота: 93 с., 12 рис., 5 табл., 32 посилань, 1 додаток.

ПЕРЕХОПЛЕННЯ БПЛА, ДРОН-ПЕРЕХОПЛЮВАЧ, ДЕТЕКЦІЯ, UNSCENTED KALMAN FILTER, ROS 2, UNITY, СИМУЛЯЦІЙНЕ СЕРЕДОВИЩЕ, SOBEL, МОДУЛЬ НАВЕДЕННЯ, ПРЯМИЙ ЗАКОН НАВЕДЕННЯ.

Об'єкт дослідження є процес автономного виявлення, відстеження та перехоплення мало розмірних безпілотних літальних апаратів.

Предмет дослідження є метод візуального вимірювання координат цілі на основі оператора Sobel і математичних моментів, стратегія прямого наведення дрона-перехоплювача на поточний вимірний центроїд у симуляційному стенді Unity + ROS 2 та нелінійна фільтрація Unscented Kalman Filter, що окремо оцінює якість прогнозу, але не входить до контуру наведення.

Мета роботи – розробити й експериментально перевірити програмну систему, яка швидко виявляє БПЛА на відео, згладжує й прогнозує координати та демонструє працездатність прямого наведення дрона-перехоплювача на поточний вимірний комп'ютерним зором центроїд цілі, забезпечуючи успішне зближення в реальному часі в симуляції Unity.

ABSTRACT

Bachelor's thesis: 93 p., 12 figures, 5 tables, 32 references, 1 appendix.

UAV INTERCEPTION, INTERCEPTOR DRONE, DETECTION, UNSCENTED KALMAN FILTER, ROS 2, UNITY, SIMULATION ENVIRONMENT, SOBEL, GUIDANCE MODULE, PURE-PURSUIT GUIDANCE LAW.

Object of research – the process of autonomous detection, tracking and interception of small unmanned aerial vehicles (UAVs).

Subject of research – the visual coordinate-measurement method based on the Sobel operator and geometric moments; the pure-pursuit guidance strategy that steers the interceptor drone to the current centroid measured in a Unity + ROS 2 simulation testbed; and the Unscented Kalman Filter used independently to assess prediction accuracy, remaining outside the real-time guidance loop.

Aim of the work – to develop and experimentally validate a software system that rapidly detects a UAV in video, smooths and predicts its coordinates, and demonstrates the viability of pure-pursuit guidance of the interceptor drone toward the current computer-vision-measured centroid, achieving successful convergence in real time within the Unity simulation.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1 Завдання перехоплення БПЛА	9
1.2 Методи детекції об'єктів	10
1.3 Методи визначення центроїду та контуру	13
1.4 Метод прогнозування траєкторій	15
1.5 Фільтр Калмана і його модифікації	16
1.6 Платформи симуляції: Unity + ROS2	22
1.7 Методи наведення на повітряну ціль	23
Висновки до розділу 1	25
РОЗДІЛ 2 МАТЕМАТИЧНІ ТА МЕТОДИЧНІ ОСНОВИ РОБОТИ	28
2.1 Матеріально-технічна база та середовище розробки	28
2.1.1 Операційна система та встановлення ROS 2	28
2.1.2 Unity: версія, налаштування камери, пакети RosSharp	30
2.1.3 Налаштування зв'язку Ubuntu, PyCharm та Unity, створення спільної папки між Ubuntu та Windows	32
2.2 Розробка підсистеми комп'ютерного зору	35
2.2.1 Архітектура CV-модуля	35
2.2.2 Реалізація Sobel-детектора: алгоритм і код	36
2.2.3 Пошук центроїда цілі та орієнтації	39
2.3 Розробка модуля прогнозування траєкторій	41
2.3.1 Теорія Unscented фільтра Калмана	41
2.3.2 Моделі руху та їхні рівняння	45
2.3.3 Архітектура та реалізація UKF у Python з патернами	48
2.4 Алгоритм наведення камери-дрона	51
Висновки до розділу 2	54

РОЗДІЛ 3 результати дослідження.....	56
3.1 Результати роботи комп'ютерного зору	56
3.2 Результати прогнозування траєкторії	57
3.3 Результати моделювання перехоплення	60
Висновки до розділу 3	62
РОЗДІЛ 4 функціонально-вартісний аналіз	64
4.1 Постановка задачі	64
4.2 Обґрунтування функцій програмного продукту.....	65
4.3 Вибір параметрів для програмного продукту	68
4.4 Експертний аналіз параметрів	70
4.5 Аналіз якості реалізації варіантів функцій.....	73
4.6 Економічний аналіз розробки.....	74
4.7 Вибір оптимального варіанту реалізації ПП	84
Висновки до розділу 4	85
ВИСНОВКИ.....	87
ПЕРЕЛІК ПОСИЛАНЬ.....	89
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ	93

ВСТУП

Останні роки БПЛА¹ з розвідувально-ударними можливостями стали масовим та відносно дешевим інструментом ведення бойових дій. Легкі FPV-дрони, коштуючи кілька сотень доларів, здатні високоточно вражати техніку та живу силу противника, зменшуючи власні втрати. Однак така "демократизація" високоточної зброї має й зворотний бік: ворог так само активно застосовує малорозмірні дрони, а класичні засоби ППО не завжди встигають виявляти й вражати цілі, що маневрують на низьких висотах.

Сучасні системи протидії БПЛА комбінують радіолокаційний, радіочастотний, акустичний та оптичний моніторинг. Кожен із каналів має обмеження, тому для швидкого перехоплення ворожого дрона дедалі частіше пропонують використовувати інший БПЛА-перехоплювач, який знешкоджує ціль сіткою, тараном або боєприпасом.

Ключова інженерна проблема полягає у повністю автономному наведенні. У зоні активних засобів радіоелектронної боротьби оператор може втратити зв'язок, а ціль здатна різко маневрувати. Це вимагає поєднання швидкої комп'ютерної детекції та точного короткострокового прогнозу траєкторії.

Для цього було створено віртуальний стенд Unity + ROS 2, що моделює два дрони – "ворожий" БПЛА та "дрон-перехоплювач". Така конфігурація дозволила безпечно випробувати різні алгоритми детекції та прогнозу в реальному часі, без ризику для техніки та ресурсів. У симуляції перехоплювач отримує відеопотік, який обробляється Python-модулем, що включає базовий модуль комп'ютерного зору, Unscented Kalman Filter для прогнозу траєкторії та підсистему наведення камери-дрона.

¹Тут і нижче використано такий інструмент штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Завдання перехоплення БПЛА

У сучасній реальності широко розповсюджено застосування БПЛА у військових цілях. Зокрема така висока тенденція спостерігається через їх низьку вартість: FPV-дрони для прицільних ударів можуть коштувати всього близько \$300, що значно дешевше за традиційні засоби ураження [1]. Дрони не лише демонструють неймовірну ефективність в точності ураження ворожих об'єктів, але й зберігають життя людей, замінюючи собою наземні розвідувальні групи, виконуючи ризикові завдання на передовій та суттєво знижують втрати серед особового складу. Однак інша сторона медалі полягає в тому, що ворог також застосовує таку зброю на передовій.

Основні методи виявлення ворожих об'єктів включають радіолокаційне сканування, які можуть виявити і класифікувати дрони шляхом аналізу мікро-Doppler зсувів, спричинених обертанням лопатей ротора. Такі сигнатури унікальні для квадрокоптерів і дозволяють відокремлювати їх від птахів чи інших шумів [2]. Крім того, метод RF-сканування може виявляти моделі дронів за допомогою того, що кожен дрон передає свої дані з певними кодами-ідентифікаторами, тим часом антена перехоплює це і в реальному часі розшифровує ці мітки та отримує інформацію про дрон: його швидкість, координати, висоту, тобто телеметричні дані самого дрона. Тим самим фіксує чітке положення дрона в реальному часі, шляхом декодування пакетів ідентифікаторів дронів, однак такий метод доволі чутливий до атмосферних умов та має обмежену дальність, в декілька кілометрів [3, 4]. До того ж, можливе використання акустичних сенсорів, які є фактично потужними мікрофонами. Вони вловлюють звуки, що видають дрони під час польоту. Добре підходять для комбінування з іншими методами, оскільки як самостійно одиниця неефективні [5]. Тож ефективніше використовувати в поєднанні з

машинними навчання [6]. На додачу, виявити ворожий об'єкт можна шляхом використання камери іншого дрона за методами комп'ютерного зору, тобто використовуючи CV методи, алгоритми обробки зображень, нейронні мережі, чи їх комбінації.

Однак після виявлення дрона виникає інше завдання, яке полягає у його знищенні. Одним із запропонованих рішень є перехоплення ворожого БПЛА іншим БПЛА. Найскладнішим же аспектом тут залишається автоматичне наведення без участі оператора. Оскільки в зонах дії радіоелектронної боротьби, особливо, при підльоті дрона до ворожої цілі, де переважно розгортуються засоби РЕБ, оператор втрачає здатність керувати апаратом. Таким чином, не знищуючи ворожий об'єкт. Насамперед інша проблема полягає в модифікації ворожих БПЛА, які здатні при фіксуванні об'єкта, який наближається до них, кардинально міняти положення, тим самим уникаючи ураження. З цієї причини недостатню застосовувати тільки методи комп'ютерної детекції, треба додавати реалізацію прогнозування цілі, її траєкторії.

Тож завдання автоматичного перехоплення ворожих БПЛА іншими безпілотниками є надзвичайно актуальним і критично важливим в умовах боротьби з агресією ворога.

1.2 Методи детекції об'єктів

Завдяки методам комп'ютерного зору, можна виявити об'єкти на зображенні. Оскільки відео складається з послідовних кадрів, тобто оброблюючи кожен кадр, можна на ньому виявити присутність об'єкта. Існують класичні підходи для детекції об'єкта на зображенні.

Наприклад, можна застосувати оператор Sobel або Prewitt. І хоча це дуже прості в реалізації алгоритми та вони обчислювально ефективні, однак є

чутливими до шуму. Водночас прекрасно підходять як фундамент для застосування інших методів детекції зображення. Різниця між ними в тому, що Prewitt застосовує рівномірні коефіцієнти, тому оператор рівномірно реагує на перепади яскравості без додаткового згладжування [7]. Тоді як Sobel підсилює центральний ряд або стовпець, додаючи множник 2, тим самим забезпечуючи одночасно апроксимацію похідної й легке згладжування шуму [8]. Ці оператори добре показують себе на чітких контурах, і є швидкими та простими у реалізації.

Один з найпопулярніших алгоритмів детекції зображення це алгоритм Канні, Canny Edge Detector. Він має високу точність, містячи попередню обробку шуму за допомогою гаусового розмиття. Використовує подвійний поріг для виділення сильних і слабких країв. Застосовує метод non-max suppression для підвищення точності. Та крок, hysteresis, в якому всі пікселі, які з градієнтом вище верхнього порогу беруть одразу, а пікселі між порогами залишають лише коли вони пов'язані з уже прийнятими «сильними» пікселями [9]. Найкращим він є, бо балансує між виявленням релевантних країв і зменшенням кількості хибних спрацювань.

Додатково можна виділити й алгоритм Лапласіан Гаусса, Laplacian of Gaussian, LoG, який комбінує гаусове згладжування з оператором Лапласа, щоб виявити краї. Таке поєднання згладжування та другої похідної дає змогу виявляти як сильні, так і слабкі контури. Це робить його ефективним для знаходження області з різкими змінами інтенсивності. Однак в задачі детекції БПЛА, який може легко зливатися з фоном, є не настільки ефективним. Крім того, при неправильному виборі параметрів згладжування або виборі розміру ядра гауссового фільтра, це може призвести до втрати деталей або, навпаки, до підвищення рівня шуму. Однак потреба обчислювати другу похідну після згладжування уповільнює алгоритм, особливо на великих зображеннях [10].

Окрім цього, можна виокремити, окрему підкатегорію детекції об'єктів на зображенні як методи на основі машинного навчання. Інтуїтивно детекцію зображення за допомогою цих методів можна поділити на 2 підходи. Один з

них це традиційні методи, що базуються на ознаках, обраних людиною. Наприклад, HOG + SVM, де частина HOG, Histograms of Oriented Gradients, яка розбиває зображення на клітинки, в яких рахується гістограма напрямків градієнта, після чого виконується нормалізація по блоках, тим самим формуючи вектор ознак. В подальшому частина SVM, Support Vector Machine, класифікує ці вектори як «ціль» або «фон». Такий метод добре справляється з виявленням одного типу БПЛА, але погано масштабується на широкий спектр безпілотних літальних апаратів через високу розмірність ознак, бо використовуються дуже багато різних характеристик, таких як: кольорова гістограма, текстурні показники, просторові моменти, контури, параметри форми тощо. Особливо цей метод не є стійким до повороту камери [11]. Другий з них це глибоке навчання, тобто згорткові нейронні мережі й Transformer-архітектури. Наприклад, R-CNN, Faster R-CNN, YOLO (You Only Look Once), Transformer-based моделі (DETR). Вони вже не обмежені в узагальненні та видають найвищі показники точності в складних сценах, однак потребують великий набір навчальних даних, пристроїв з потужної обчислювальної здатності й тонкого налаштування. Тому, варто зауважити, що підходи машинного навчання, або потребують колосальних обсягів розмічених даних для навчання, високої обчислювальної та ресурсної місткості, або можуть бути чутливі до маленьких змін в умовах середовища, наприклад, освітлення, фону чи ракурсу. Особливо важливо вказати, що не завжди такі методи здатні працювати у реальному часі, наприклад, YOLO спеціально розроблений, щоб детектувати зображення у поточному часі, але втрачає точність проти двоступеневих методів. Крім того, попри високу результативність, внутрішні представлення ознак у глибоких мережах складно інтерпретувати, що ускладнює пошук причин помилок і може призводити до несподіваних результатів.

Отже, для побудови базового модуля детекції в рамках дипломного проєкту обрано оператор Sobel як простий, швидкий і прозорий метод виділення контурів. Що дасть змогу в подальшому зосередитися на

прогнозуванні траєкторії БПЛА, адже в умовах, коли об'єкт зливається з фоном, через погані погодні умови чи низьку якість відео, складні нейронні моделі, теоретично, можуть виходити з ладу через згадані вище обмеження. До того ж якщо детекція об'єкта зникає з поля зору на один кадр, дрон буде повністю орієнтуватися на прогноз, що більш реалістично з огляду на практичні умови.

1.3 Методи визначення центроїду та контуру

Після детекції об'єкта, треба визначити його центроїд, щоб на основі центроїду прогнозувати подальшу траєкторію об'єкта. Однак спочатку треба визначити контур, щоб на його основі або безпосередньо за контурами, обчислити центроїд. Найпростіший метод це одразу для визначення контуру об'єкта застосувати бінаризацію зображення, а потім використати функцію в бібліотеці в Python "OpenCV", яка витягне замкнені лінії, які розбиті на послідовні пікселі. Після цього визначити область з найбільшою площею, з списку цих послідовних пікселів, отримавши контур об'єкта [12]. Аналогічно, якщо детекція проводилась за алгоритмом Canny спершу перетворюємо зображення на двійкову маску країв, а потім, так само як у випадку з бінаризацією, використовується аналогічна функція в бібліотеці "OpenCV" на Python [13]. Такий підхід є швидким та простий в реалізації. Оскільки в дипломній роботі БПЛА буде літати на фоні неба, тобто однорідного фона, тож такий підхід є більш, ніж достатнім. Однак якщо треба виділяти контури не на однорідному фоні, треба використовувати більш складні методи. Більш просунуті методи, наприклад, активні контури, «Snakes», ґрунтуються на розміщенні початкового сплайна, гладкої замкненої кривої, біля контуру об'єкта. На кожну точку цієї кривої діють внутрішні і зовнішні сили, які і рухають їх. При цьому сам сплайн прагне залишатися

рівномірний, щоб його суміжні точки не надто сильно розходились. Це в теорії, може ігнорувати колії на полі, якщо дрон летить над ціллю. Оскільки колійні борозни мають форму ліній, тому будуть занадто сильно розходитися з об'єктом БПЛА, якщо сам об'єкт має вигнуті форми. Однак метод сильно залежить від вибору початкового положення, якщо сплайн розмістити занадто далеко від справжнього краю, він може не дійти до об'єкта. Тим самим, якщо колії на полі мають яскраво виражений градієнт, то зовнішні сили можуть «зачепити» контур саме за нього, а не за форму БПЛА. Крім того, модель активного контуру не може використовувати таку інформації, як текстура та колір об'єкта. Аналогічно не вирішує проблему, якщо об'єкт зливається з фоном, тобто коли краї об'єкта слабкі, шумні та розмиті [14]. Чи модифікація цього методу як геодезичні активні контури (Geodesic Active Contours, GAC). Вони розширюють ідею класичних “Snakes” через рівнянтове представлення, яке замість явного набору контрольних точок, описує контур як нульовий рівень функції, а еволюція відбувається згідно з рівнянтовим рівнянням, що мінімізує геодезичну довжину кривої в метриці, індукованій градієнтами зображення [15]. Водночас варто не забувати про нейронні мережі сегментації, наприклад, U-Net, Mask R-CNN. Однак їх недоліки такі самі, як і в задачі детекції об'єкта.

Отже, після знаходження контуру необхідно знайти центроїд. Точки, яка визначає центр маси фігури. Найпопулярніший та ефективний спосіб знаходити його через математичні моменти. Крім того, найпростішим способом є використання обмежувального прямокутника, bounding box. виділяють максимальні та мінімальні координати з контуру та будують з них прямокутник, де його середина, це і буде центроїд об'єкта. Більш точніший спосіб, ніж bounding box, є побудова еліпса навколо контуру. Цей алгоритм шукає мінімальну еліптичну фігуру, що описує контур, і повертає її параметри – центр, осі й кут повороту. Такий алгоритм може обчислити орієнтацію об'єкта, так само як і математичні моменти. Оскільки в подальшому буде будуватися дві траєкторії руху об'єкта, для більш складної

моделі руху БПЛА, необхідне значення орієнтації. Отже, оскільки важливим завданням в цій дипломній роботі є прогноз, тому обчислення центроїду та орієнтації потребує точних результатів. Тож через ефективність та високу точність буде обчислено центроїд та орієнтація через математичні моменти. Хоча точність може гарантувати й нейронні мережі, наприклад, PoseCNN [16], або особливо в поєднанні з алгоритмом Perspective-n-Point (PnP) [17], однак недоліки нейронних мереж були описані вище.

1.4 Метод прогнозування траєкторій

Треба зауважити, що для ефективного перехоплення ворожого БПЛА недостатньо лише виявити його на зображенні – необхідно ще й передбачити його подальшу траєкторію. Особливо це критично в умовах, коли об'єкт частково зникає з поля зору камери, наприклад, коли БПЛА при певному ракурсі зливається з фоном, або комп'ютерний зір видає нестабільні результати через шум у зображенні. Саме в таких випадках система повинна спиратися на попередню динаміку руху об'єкта, щоб продовжити прогноз навіть за відсутності поточних вимірювань. Існує кілька методів прогнозування траєкторії, так, наприклад, фільтри стану, які використовують математичну модель руху (як CV/CA) як модель стану та поєднують її з реальними вимірюваннями, наприклад, позиція центроїда чи орієнтація [24]. Крім того, для прогнозування траєкторії можна використовувати нейронні мережі. Наприклад, LSTM, яка вирішує проблему зникнення градієнта та «пам'ятає» корисну інформацію довгий час, в дослідженнях показала високу ефективність для прогнозування руху літаків або БПЛА [25]. Вище ефективність в порівнянні з LSTM продемонстрував Transformer, який замість рекурентної обробки використовує self-attention, тим самим не залежить від порядку даних, а аналізує всю послідовність одразу, що робить

його дуже потужним, але ресурсоемним. Таким чином значно пришвидщується навчання і покращується обробка довгих послідовностей. У сфері прогнозування траєкторій трансформери продемонстрували вищу точність в порівнянні з RNN / LSTM [26]. Однак нейронні мережі потребують значні обчислювальні ресурси, великий обсяг даних історичних даних для тренування та мають ризик перенавчання, тобто запам'ятовуючи характерні шаблони руху, тому внаслідок цього погано узагальнюють на нові чи екстремальні траєкторії.

Тож було вирішено не використовувати нейромережеві методи, оскільки вони потребують великих обсягів навчальних даних, мають складний процес навчання, чутливі до зміни контексту, наприклад до змін фону, освітлення чи положення камери, та не завжди працюють у реальному часі без потужної відеокарти. Крім того, такі моделі є менш інтерпретованими й складними в налагодженні. Натомість було обрано Unscented Kalman Filter (UKF) – метод, що базується на теорії фільтрів стану, але водночас здатен обробляти нелінійності без необхідності обчислення похідних, як це потрібно у розширеному фільтрі Калмана (ЕКФ) [24].

1.5 Фільтр Калмана і його модифікації

Після того, як було проведення детекцію об'єкта та обчислено його центроїда, важливо точно прогнозувати його подальшу траєкторію. По-перше, це робиться для того, якщо наступний кадр не знайде об'єкт, але ми впевнені, що в попередньому кадрі він був, то модель буде орієнтуватися на прогнозовану точку, або якщо детекція була нечітка, то треба згладити прогнозоване значення та те, що виявив комп'ютерний зір, так отримається набагато точніше значення, ніж могло бути. Цю задачу буде вирішувати фільтр Калмана. Головна ідея якого: після кожного нового вимірювання

оновлювати найкращу оцінку стану дрона та її невизначеність, а між вимірами – прогнозувати стан наперед за фізичною моделлю руху об’єкта.

Фільтр Калмана ґрунтується на ймовірнісній моделі: припускається, що всі похибки мають нормальний (гауссівський) розподіл. Одна з головних похибок це невизначеність оцінки стану, тобто наскільки відхиляється оцінка стану від реального значення. Зазвичай спочатку задаються великими, щоб фільтр міг «навчитися» на перших спостереженнях і швидко адаптуватися до реальних змін траєкторії. З кожним новим вимірюванням цей показник повинен зменшуватися. Інша головна похибка - це похибка вимірювання, тобто наскільки реальне значення відрізняється від того, що було виміряно за допомогою, наприклад, комп’ютерного зору. Цю похибку треба поставити коректно, оскільки при неправильному налаштуванні похибки фільтр призведе до двох з можливих небажаних наслідків: або він надто довірятиме шумним вимірюванням і видаватиме стрибкоподібні, нестабільні оцінки, або, навпаки, вважатиме спостереження занадто ненадійними й віддаватиме перевагу власному прогнозу, відстаючи від реальних змін траєкторії. З цих двох похибок формується коефіцієнт Калмана або калманівське посилення, який обчислюється як: похибку невизначеності оцінки стану поділити на суму цієї похибки та похибки вимірювання:

$$K = \frac{p_{pred}}{p_{pred}+r}. \quad (1.4.1)$$

Цей коефіцієнт і визначає вагу для прогнозованого значення та вимірювання, на яке з них спиратися. Тож найкраща оцінка стану дрона, а саме в дипломній роботі це його центроїд, це зважена комбінація прогнозованої моделюючою фізикою руху позиції та фактичного вимірювання комп’ютерним зором, який регулюється коефіцієнтом Калмана. В даному випадку, чим ближче цей показник до одиниці, тим більшу вагу має детекція, чим ближче до нуля – тим сильніше розрахунок спирається на

фізичну модель руху. Тобто баланс між прогнозованою моделюючою фізикою руху позиції, x_{pred} , й фактичного вимірювання комп'ютерним зором, z , задає саме калманівське посилення:

$$x_{new} = x_{pred} + K(z - x_{pred}) = (1 - K)x_{pred} + Kz. \quad (1.4.2)$$

Аналогічно калманівське посилення оновлює значення дисперсії оцінки стану, тобто чим більше фільтр довіряє прогнозу, отже, прямує до нуля, то тим значення невизначеності майже не зміниться. Оскільки фільтр буде вважати нове вимірювання майже ненадійним. Інформація з вимірювання практично не потрапляє в оновлення, і фільтр спирається здебільшого на свій прогноз. Тож фільтр залишає невизначеність майже на тому самому рівні, бо вимірювання не переконало його змінити оцінку. Іншими словами, якщо детекція некоректна, то немає різниці зменшувати невизначеність на основі вимірювання. Тому залишається велика дисперсія прогнозного стану, адже нові дані можуть бути ще гіршими:

$$p_{new} = (1 - K)p_{pred}. \quad (1.4.3)$$

Класичний фільтр Калмана можна розділити за розмірністю: одновимірний та багатовимірний випадок. В одновимірному випадку оцінка має в собі тільки одне значення, наприклад, прогнозуємо тільки, на якій висоті знаходиться дрон. В багатовимірному випадку оцінка має в собі кілька значень, наприклад, центроїд для зображення має в собі дві координати. Як раз ці два значення і будуть прогнозуватися фільтром Калмана. З математичної точки зору, різниця полягає в тому, що зі збільшенням кількості вимірювань зростає розмірність вектора оцінки стану.

Крім того, можна розділити за характером руху об'єкта, який прогнозуємо: статичний та динамічний. Іншими словами статичний

одновимірний фільтр Калмана, це коли дрон завис в повітрі, та ми прогнозуємо його висоту. Якщо розглядати статичний багатовимірний фільтр Калмана, це коли дрон завис у повітрі, та ми прогнозуємо його координати по абсцис та ординат. Відповідно динамічний фільтр Калмана, це коли дрон рухається. З математичної точки зору, для статичного фільтра Калмана існують два види похибок: дисперсія вимірювання та невизначеність оцінки стану. Дисперсія вимірювання, тобто наскільки значення висоти отриманого за допомогою лазерного далекоміра, відрізняється від реального висоти, на якій перебуває дрон. Невизначеність оцінки стану, тобто наскільки фінальна оцінка відрізняється від реального значення висоти. У статичному випадку після кожного кроку прогнозування та оновлення фільтра отримане значення стану стає вихідним прогнозом для наступного кроку за формулою (1.4.4):

$$x_{pred} = x_{new}. \quad (1.4.4)$$

У динамічному ж випадку між вимірюваннями фільтр використовує рівняння руху об'єкта для прогнозування наступного стану. Тим самим додаючи похибку процесного шуму. Яка характеризується невизначеністю моделі з реальним значенням, яке прогнозується. Нехай x_{pred} – це найкраща оцінка стану дрона після останнього оновлення, а u – зовнішній вплив або керувальна вхідна команда, наприклад, прискорення або кут повороту. Тоді прогноз буде описуватися моделлю:

$$x_{pred} = f(x_{prev}, u). \quad (1.4.5)$$

Зазначимо, що функція f реалізує модель руху. Наприклад, якщо наводити дуже спрощений приклад, в якому припускаємо, що процесний шум буде додаватися тільки до рівняння швидкості. Крім того, крос-коваріація між положенням та швидкістю дорівнює нулю, тобто $p^{xv} = p^{xv}$.

Отже, оскільки треба прогнозувати положення дрона тільки через швидкість без зовнішнього впливу, тоді його рівняння руху буде:

$$\begin{cases} x_{pred} = x_{prev} + \Delta t \cdot v_{prev}, \\ v_{pred} = v_{prev}. \end{cases} \quad (1.4.6)$$

Оскільки як зазначалось вище, до динамічного випадку додається похибка процесного шуму, q , тобто наскільки модель руху відрізняється від ідеальної моделі руху. Отже, та дисперсія оцінки стану буде обчислюватися:

$$\begin{cases} p_{pred}^x = p_{prev}^x + \Delta t^2 p_{prev}^v, \\ p_{pred}^v = p_{prev}^v + q. \end{cases} \quad (1.4.7)$$

Отже, можна зробити висновок, що невизначеність оцінки стану, при даній моделі руху, формується з дисперсії положення та її швидкості. Зауважимо, що добуток дисперсії швидкості на Δt^2 впливає з властивості дисперсій в теорії ймовірності: коли дисперсія добутка випадкової величини та константи, тоді невизначеність буде дорівнювати константа в квадраті на дисперсію цієї випадкової величини. Або це можна записати у матричній формі, як:

$$P_{pred} = F P_{prev} F^T + Q, \quad (1.4.8)$$

де, в цьому випадку, матриця, F , буде функцією моделі руху, а Q це матриця похибки процесного шуму. Тобто для приклада, що розглядався раніше це буде – див. формулу (1.4.9).

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} 0 & 0 \\ 0 & q \end{bmatrix}. \quad (1.4.9)$$

У динамічних задачах часто розширюють стан, включаючи додаткові параметри (наприклад, швидкість, прискорення), тому отримується багатовимірний стан. В даній дипломній роботі багатовимірний динамічний фільтр Калмана. Однак реалізована модифікація фільтра Калмана - Unscented Kalman Filter (UKF), який уникає грубих апроксимацій нелінійностей через лінеаризацію, як у EKF, розширеного фільтра Калмана, однак натомість генерує набір «сигма-точок», пропускає їх крізь точну модель руху й вимірювання та знову об'єднує в середній коваріації. Водночас розширений фільтр Калмана працює з якобіанами він лінеаризує нелінійні функції переходу та вимірювання навколо поточної оцінки, обчислюючи їхні похідні, Jacobian, але через це може накопичувати значні помилки при сильних нелінійностях. Оскільки навіть для “великої” моделі руху, куди додано орієнтацію, матриця функції стану має розмірність лише 9×9 , а відтак генерація приблизно 19 σ -точок залишається досить швидкою, фільтр зможе працювати в реальному часі без помітних затримок. Тому було прийнято рішення реалізувати Unscented Kalman Filter (UKF), оскільки він забезпечує вищу точність і стійкість при сильних нелінійностях порівняно з розширеним фільтром Калмана (EKF).

Як вже зазначалось вище, фільтр складається в вимірювання, в даному випадку це є центроїд, який був отриманий за допомогою методів комп'ютерного зору та для другої моделі руху, додається ще орієнтація об'єкта. Та його невизначеності, тобто дисперсії, тобто наскільки можливе відхилення центроїду або ще й орієнтації від реального. Наприклад, по осі абсцис відхилення від реального значення десь 2 пікселя, а по осі ординат, наприклад, 3 пікселя. Стосовно орієнтації, можливо, наприклад, 5 градусів. Це значення ставиться власноруч. Тому й важливо, щоб дисперсії вимірювань були налаштовані відповідно до реального розкиду центроїда. Ні низькими, оскільки тоді фільтр буде надто сильно довіряти результатам отриманими детекцією. Ні великими, оскільки тоді фільтр буде вважати, що детекція здійснена комп'ютерним зором є ненадійною, і більш спиратися на

прогноз. Аналогічно матрицю процесного шуму задають власноруч на основі очікуваної невизначеності самої моделі руху дрона. Початкову матрицю невизначеності оцінки стану зазвичай ініціалізують великими значеннями.

Отже, стан системи і вимірювання трактується як випадкові величини, описані середнім значенням, очікуванням, та дисперсією, мірою невизначеності. Алгоритм підтримує оцінку стану, тобто найкраще наближення до істинного значення, та коваріацію оцінки, тобто матрицю, що характеризує похибку прогнозованої оцінки. На кожному кроці прогнозу фільтр видає прогнозовані значення цих параметрів, а на етапі оновлення – коригує їх на основі нового вимірювання і його точності [18].

1.6 Платформи симуляції: Unity + ROS2

У дипломній роботі для створення симуляції, тобто самого об'єкта спостереження, ворожого БПЛА, та сцени, було обрано середовище розробки - Unity. Саме середовище забезпечує реалістичну графіку. Адже використовується High Definition Render Pipeline (HDRP) із налаштуваннями фізично коректного освітлення (Physical Light Units) та Sky and Fog Volume для створення більш реалістичного неба. Створена сама сцена, де ворожий БПЛА, наприклад, БПЛА типу «ZALA», рухається по заданій траєкторії. Водночас дрон, який повинен його знищити, буде спостерігати над БПЛА знизу, щоб не зачіпати горизонт та землю. Зауважимо, що для спрощення сцени, дрон буде динамічною камерою, яка створена за допомогою Cinemachine. Тож оскільки камера це і є дрон, то сцена повинна містити дві камери: одна для публікації кадрів в ROS 2-топік через WebSocket за допомогою пакета RosSharp, а друга для безпосередньо руху камери. Аналогічно для подальшого аналізу треба не тільки публікувати в ROS2-топік кадр, а ще й реальний центроїд об'єкта. Це є спрощеною задачею,

оскільки в дипломній роботі буде виключено детекція об'єкта, який летить на фоні складних ландшафтів: полів, будівель тощо. З огляду на це, оператор Sobel ідеально підійде під задачу. Він створює каркас детекції, тому можна зосередитись у дипломній роботі саме на прогноз траєкторії.

У сцені Unity також реалізовано підписку на ROS2-топік команд наведення камери, який публікує Python-модуль. Завдяки цьому камера у Unity отримує повідомлення й автоматично коригує свій ракурс відповідно до команд із Python, забезпечуючи постійне слідування за ворожим БПЛА.

Зауважимо, що для двонаправленого обміну даними між симуляцією в Unity та Python-модулем, в якому будуть оброблятися дані: реалізований фільтр Калмана, проведення аналізу між реальним і прогнозованим центроїдом та даних для забезпечення наведення дрона на ціль, - використовується пакет RosSharp. Unity-сцена через WebSocket публікує та підписується на ROS2-топіки, що забезпечує передачу відеокадрів, координат реального центроїда об'єкта, позицію та орієнтацію камери та направляє камеру на ціль.

1.7 Методи наведення на повітряну ціль

У літературі з перехоплення маневрових безпілотників описано кілька класичних законів наведення. Найстаріший і найпоширеніший – пропорційна навігація (PN), де кутова швидкість повороту перехоплювача пропорційна швидкості обертання уявного прямого променя, що сполучає центр маси перехоплювача з центром маси цілі у даний момент часу, тобто іншими словами лінії візування (LOS). Саме закон пропорційної навігації лежить в основі ракет «повітря-повітря» завдяки енергетичній оптимальності на великих дистанціях [27]. Для малорозмірних дронів цей закон також ефективний, однак потребує точної оцінки кутової швидкості лінії візування і

помітно деградує при затримках або пропусках кадрів відео потоку [28], хоча ці ефекти частково можна згладити фільтром Калмана. Однак оскільки в симуляції не моделюється повна 3-D-кінематика дрона, спроба далі отримувати лінійну швидкість і прискорення з Unity (через різницеве диференціювання положення кадр-за-кадром) одразу виявилася надто шумною: мала Δt перетворює навіть кілька пікселів помилки на десятки градусів за секунду. Тому оскільки цей закон наведенням вимагає точної кутової швидкості, то без повного моделювання дрона-перехоплювача, результати будуть потенційно нестабільні. Крім того, існують підходи, які складніше за традиційну пропорційну навігацію. Так, наприклад, вектор-польове наведення, *Vector-field guidance*, в якому уся робоча область заповнюється штучним векторним полем, де в кожній точці стрілка поля вказує бажаний напрямок руху дрона так, щоб він зрештою зблизився з ціллю, обходячи перешкоди або тримаючи орбіту навколо неї [29]. Однак недолік полягає в тому, що поле мусить будуватися у глобальних 3-D координатах і регулярно оновлюватися, коли ціль маневрує, що потребує точного положення перехоплювача, цілі та обчислювального ресурсу для перебудови поля. Додатково існує ризик потрапити до «локальних мінімумів», якщо топологія поля обрана невдало [30]. Оскільки камера видає лише 2-D центроїд, а GPS/SLAM не використовується, тому побудувати коректне 3-D поле неможливо. Тому цей метод не буде використовуватися. Інші методи також потребують або повного 3-D стану, або ще й значної обчислювальної потужності. Оскільки камера видає лише 2-D центроїд, а ресурс бортового комп'ютера обмежений, тому з цієї причини для базової реалізації обрано закон прямого наведення (*Pure Pursuit, PP*), де дрон у кожен момент летить уздовж поточного вектора до цілі, LOS. Сучасні дослідження показують, що за умови переваги швидкості дрона, алгоритм гарантує перехоплення. Завдяки цьому PP стійко працює навіть на недорогих одноплатних комп'ютерах і не вимагає оцінювати похідні поточного вектора, що критично, коли цілевказання надходить лише з камери [31].

Отже, для базового прототипу дрона-перехоплювача у дипломній роботі обрано саме Pure Pursuit. Таке рішення диктується кількома чинниками. По-перше, алгоритм потребує лише координат поточного центроїда і тому сумісний із візуальним датчиком без кутової швидкості. По-друге, PP майже не чутливий до пропусків кадрів: якщо вимірювання тимчасово зникає, дрон інерційно продовжує рух у тому ж напрямку, а після відновлення детекції миттєво корегує курс. По-третє, низька обчислювальна вартість дає змогу запускати наведення разом із Unscented Kalman Filter на одному одноплатнику без GPU. Нарешті, PP легко поєднується з UKF-прогнозом: коли ціль втрачається з кадру, LOS перенаправляється на прогнозовану точку, що зменшує ризик зриву переслідування. У перспективі, після додавання IMU чи радіолокаційного датчика швидкості обертання LOS, система може бути розширена до гібридного PP + PN або до енергооптимального MPC, але для стартової версії PP забезпечує необхідний баланс між простотою, робастністю й обчислювальною доступністю.

Висновки до розділу 1

Стрімке здешевлення та масове застосування малорозмірних БПЛА зробили їх одним із найнебезпечніших засобів сучасного поля бою. Існуючі канали виявлення – радарні, RF-сканування, акустичні та оптичні – поодиноці мають суттєві обмеження за дальністю, завадостійкістю або швидкодією, тому потребують комплексного використання. Навіть гарантоване виявлення цілі не розв'язує головної задачі – її фізичного знищення. Тому практичне завдання перехоплення БПЛА виходить за межі однієї «детекції» й вимагає інтегрованої системи: швидкого комп'ютерного зору, алгоритмів прогнозування траєкторії та високоточних стратегій наведення. Саме така

комбінація спроможна забезпечити надійне знешкодження ворожого безпілотної в умовах сучасних бойових дій.

Перше завдання для розв'язання цього аспекту, полягає в обранні методів комп'ютерного зору. Хоча глибокі CNN/Transformer-детектори (YOLO, DETR) дають найкращу точність, але вони потребують великих датасетів, потужного GPU і чутливі до деградації зображення. В умовах обмежених ресурсів базова детекція на контрастному фоні може бути реалізована простими градієнтними операторами із подальшим виділенням найбільшого контуру. Такий вибір виправданий для стартового прототипу: він мінімізує обчислювальні витрати й полегшує аналіз помилок на етапі налагодження.

Для локалізації цілі доцільно використовувати геометричні моменти – вони швидко дають координати центроїда й оцінку орієнтації без дорогих процедур оптимізації. У разі складного фону чи неоднозначного силуету техніку можна розширити еліптичним апроксимаційним контуром або комбінувати з легкою сегментаційною мережею.

Нарешті, порівняння схем фільтрації траєкторій засвідчило, що класичний та розширений фільтри Калмана накопичують похибку при суттєвих нелінійностях, властивих активному маневру БПЛА. Unscented Kalman Filter із набором сигма-точок забезпечує другий порядок точності без числення Якобіанів. Таким чином, UKF обґрунтовано обрано ядром прогнозувальної підсистеми.

Останнім етапом було проаналізовано сучасні закони наведення дрона-перехоплювача. Розглянуто пропорційну навігацію та вектор-польове керування. Показано, що всі перелічені підходи забезпечують високу точність перехоплення, проте потребують або точної кутової швидкості лінії візування, або потужного процесора та детальної моделі динаміки. За умов, коли датчиком є лише камера, а бортовий комп'ютер обмежений, обґрунтовано вибрано найпростішу та найробастнішу схему – чисте пряме наведення, яке оперує одним вектором LOS, стійкий до пропусків кадрів,

легко комбінується з UKF-прогнозом і створює базу для подальшого переходу до складніших законів після інтеграції додаткових сенсорів.

Підсумовуючи, дослідження предметної області дозволило сформулювати вимоги до прототипу: легка й прозора детекція та контур-аналіз для початкової версії, використання Unscented Kalman Filter для стабільного передбачення координат і кутів, віртуальне симуляційне середовище Unity + ROS 2 для безпечного й швидкого тестування алгоритмів.

РОЗДІЛ 2 МАТЕМАТИЧНІ ТА МЕТОДИЧНІ ОСНОВИ РОБОТИ

2.1 Матеріально-технічна база та середовище розробки

2.1.1 Операційна система та встановлення ROS 2

Оскільки Unity для передачі даних взаємодіє з ROS2. Натомість й хоча ROS2 має інструкції для встановлення на Windows, найстабільніші пакети й найновіші оновлення спершу виходять під Ubuntu, особливо версії LTS. Крім того, Linux-ядро дає кращий контроль над плануванням процесів і реальним часом, що важливо для даної дипломної роботи. Тому було прийнято рішення встановити ROS2 на Ubuntu 22.04.05 LTS, який знаходиться на віртуальній машині, VirtualBox. Операційній системі виділено 4096 МБ операційної пам'яті та чотири ядра процесора..

Отже, спочатку необхідно додати репозиторій “universe”. Командою “sudo apt install software-properties-common” - встановляться утиліти для зручного керування списком репозиторію “universe”, який треба додати. Та “sudo add-apt-repository universe”, який безпосередньо додасть до списку репозиторіїв компонент Universe, де містяться пакунки з відкритим кодом, необхідні для деяких залежностей ROS 2.

Особливо важливо перевірити налаштування UTF-8 локалі, командою: “locale”. Якщо їх немає, то виправити це командами: “sudo locale-gen en_US.UTF-8”, який генерує підтримку заданої локалі “en_US.UTF-8” у системній базі локалей, “sudo update-locale LANG=en_US.UTF-8”, який створює або оновлює файл “/etc/default/locale”, задаючи основну системну локаль, якої ім'я знаходиться під змінної “LANG” у “en_US.UTF-8”, та “source /etc/default/locale”, який завантажить змінні середовища з файлу “/etc/default/locale” у поточну інтерактивну оболонку, Bash, без перезавантаження системи чи повторного входу користувача.

Особливо важливо потім оновити список пакетів, тобто локальну базу доступних пакунків із усіх увімкнених репозиторіїв, щоб система «побачила»

нові пакунки, командою: “sudo apt update”. Після встановити curl, це утиліта, яка дасть можливість завантажити файли по HTTP/HTTPS з командного рядка. Та імпортувати GPG-ключ ROS 2. Тобто виконати такі команди:

```
“sudo apt install curl -y” та “sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg”
```

В подальшому, необхідно додати репозиторій ROS 2 до списку джерел командою:

```
“echo “deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-
archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release &&
echo $UBUNTU_CODENAME) main” \ | sudo tee /etc/apt/sources.list.d/ros2.list
> /dev/null”.
```

Пізніше, знову оновити список пакетів, щоб apt побачив пакунки з нового репозиторію ROS 2. Опціонально можна оновити існуючі пакети, командою: “sudo apt upgrade -y”. Щоб підвищити версії вже встановлених пакетів до найновіших з усіх репозиторіїв, аби уникнути конфлікти залежностей.

Нарешті було встановлено пакети ROS 2. Командою: “sudo apt install ros-humble-desktop -y”, - встановлено ROS, Rviz, демонстрації, приклади та всі залежності для повноцінної роботи. Командою: “sudo apt install ros-humble-ros-base -y”, - було забезпечено мінімальну істаляцію, тобто встановлено лише ядро ROS 2 та CLI, без графічних пакетів. Командою: “sudo apt install ros-dev-tools -y”, - встановлено додаткові інструменти розробника необхідні для збірки та управління вихідним кодом.

Одразу було встановлено автоматичне підвантаження ROS 2 при старті оболонки, аби не вводити, щоразу команду: “source /opt/ros/humble/setup.bash”. Було додано команду: “echo “source /opt/ros/humble/setup.bash” >> ~/.bashrc”, - до конфігураційного файлу ініціалізації інтерактивної оболонки Bash користувача. Це гарантує, що тільки при відкритті терміналу змінні ROS 2 завантажаться автоматично.

Тепер, щоб підключитися до Unity було встановлено та запущено “rosbridge_server” командою: “sudo apt install ros-humble-rosbridge-server -y” [19]. У Unity достатньо вказати IP-адресу та порт ROS-сервера. Якщо використовуєте статичну мережу, доцільніше підключатися через мережевий міст, Bridge, в іншому разі – через інтерфейс Host-only. Проте варто пам’ятати, що Bridge іноді може втрачати зв’язок, а Host-only без додаткової конфігурації NAT не забезпечує доступу до Інтернету. Щоб поєднати переваги обох режимів, необхідно тонко налаштувати функцію переадресації портів через NAT.

2.1.2 Unity: версія, налаштування камери, пакети RosSharp

Версія Unity “Unity Editor 6000.0.24f1”, яка встановлена через Unity Hub і використовується для розробки та тестування проєкту. Проєкт підключає офіційний пакет RosSharp (версія 1.1.0), який реалізує зв’язок Unity з ROS2 через протокол WebSocket.

Створена сама сцена, де ворожий БПЛА, наприклад, БПЛА типу “ZALA”, який імпортується у вигляді FBX-префабу (Zala_421_16E2) із компонентами Mesh Filter, який містить саму геометрію об’єкта, набір вершинок і трикутників, які описують форму дрона, і Mesh Renderer, який працює з Mesh Filter, щоб взяти цю геометрію і відобразити її на екрані, та фізичний контур об’єкта Mesh Collider, який Unity використає для зіткнення. Щоб при перехопленні повідомлення про це в коді, завершити симуляцію. Mesh Collider налаштований таким чином, щоб лише відстежувати перетин об’єктів без фізичного зіткнення. Рух дрона не керується фізикою Unity (PhysX). Rigidbody додають лише для відлову тригерних подій (HitDetector), а саме переміщення здійснюється власним скриптом. Динаміка польоту реалізована в компоненті ZalaTrajectory. У методі Start() скрипт фіксує

початкову позицію як центр руху та додатково вимірюється розмір об'єкта (для налагодження), а в Update() щокадру Змінна angle зменшується пропорційно швидкості й дельті часу. Нові координати розраховуються за коловою траєкторією. Такий підхід дає точний контроль над параметрами радіуса й швидкості, а також зручний механізм візуалізації попередньої та прогнозованої траєкторії.

Сцена містить дві камери: Main Camera, для руху камери, й ROSCamera, для публікації зображень у ROS2-топік camera_image_status та RosConnector, по якому підключається Unity до ROS 2, щоб надіслати або прийняти повідомлення. Водночас дрон, який повинен його знищити, буде спостерігати над БПЛА знизу, щоб не зачіпати горизонт та землю. Зауважимо, що для спрощення сцени, дрон буде динамічною камерою, яка створена за допомогою Cinemachine. Підлога моделюється широким Plane (100×100), щоб усунути детекцію землі, а небо й освітлення задаються через HDRP-компоненти Sun і Sky and Fog Volume. Додаткові GameObject-и: ros2, який надсилає кадр, pixel_pose, який надсилає реальний центроїд об'єкта, де Python-скрипт аналізує його з тим, що видав фільтр Калмана та HitDetector, який відловлює тригерні події і завершує симуляцію. Така архітектура поєднує гнучкість Unity-середовища з потужністю ROS2-занурення і робить проект готовим до подальшого масштабування й експериментів.

Це є спрощеною задачею, оскільки в дипломній роботі буде виключено детекція об'єкта, який летить на фоні полів. З огляду на це, оператор Sobel ідеально підійде під задачу, і можна зосередитись у дипломній роботі саме на прогноз траєкторії.

2.1.3 Налаштування зв'язку Ubuntu, PyCharm та Unity, створення спільної папки між Ubuntu та Windows

У цій дипломній роботі, створюється віртуальне середовище, в якому буде проводитись обчислення. Отже, треба знову спочатку оновити пакети командою “`sudo apt update`” та завантажити пакет для створення віртуальних середовищ командою “`sudo apt install python3-venv`”. Після цього створити віртуальне середовище з ім'ям “`venv`” командою “`python3 -m venv venv`”. Та щоб працювати у ньому, його необхідно кожного разу активувати командою: “`source venv/bin/activate`”, після чого в терміналі на початку рядка з'явиться позначка (`venv`), що означає активоване середовище.

Натомість код пишеться в PyCharm на Windows, а обробляється на Ubuntu. Тому треба налаштувати SSH з'єднання. Є два варіанта їх налаштування або через мережевий міст, Bridge, або через інтерфейс Host-only. Якщо використовується статична мережа, тобто її IP не змінюється з часом, то краще підключатися за допомогою Bridge. Однак на базі, де була пройдена практика був нестабільний IP адрес, тому було вирішено використовувати Host-only. Проте кроки налаштувати SSH з'єднання однакові. Тож, спочатку треба в Ubuntu налаштувати SSH-підключення, оновлюючи спочатку пакети командою “`sudo apt update`”, а потім встановлюючи OpenSSH, командою “`sudo apt install -y openssh-server`”. Згодом треба увімкнути, командою “`sudo systemctl enable ssh`” та запустити сервіс `sshd`, командою “`sudo systemctl start ssh`”. Тепер треба перевірити що `sshd` слухає порт 22, саме по цьому порту будемо і підключатися. Командою “`sudo ss -tnlp | grep sshd`” повинен вивести очікуваний результат “`LISTEN 0 128 0.0.0.0:22 0.0.0.0:* users:(("sshd",pid=...,fd=...))`”. Опційно, якщо “`ufw`” блокує цей порт, тоді треба виконати команду “`sudo ufw allow ssh`”. Та перевірити дозвіл для порта “22” командою “`sudo ufw status`” або, якщо неактивний “`ufw`”, то активувати його командою “`sudo ufw enable`”.

Далі треба створити SSH Interpreter у PyCharm. В самій IDE перейти у “File → Settings (Ctrl+Alt+S) → Project: <ім'я> → Python Interpreter”. У вікні вести порт “22”, вказати IP-адресу гостьової системи Ubuntu, до якого підключається Unity, ім'я користувача та ключ чи пароль. Далі повинно успішно пройти з'єднання, якщо цього немає, або не завантажені бібліотеки, або блокується порт, або неправильно вказано IP-адресу. Після чого на кроці "Project directory and Python runtime configuration" треба оберіти в Virtualenv Environment кнопку Existing. Це робиться, оскільки не треба створювати віртуальне середовище, бо воно вже було створено на попередніх кроках з ім'ям “venv” раніше. Далі в полі “Interpreter” треба вказати шлях до цього віртуального середовища, venv-інтерпретатора, наприклад “/home/polina/venv/bin/python3.10”. В полі “Sync folders” треба переконатися, що з папки проекту в PyCharm будуть синхронізуватися файли в папці проекту на віртуальній машині, тобто, наприклад, “<Project root> → /home/polina/Diploma”. Якщо на віртуальній машині немає такої папки її треба створити командою “mkdir”. Далі можна налаштувати Deployment у PyCharm, щоб або виключити папки, які не треба для обчислення, для цього треба перейти у “File → Settings → Build, Execution, Deployment → Deployment” і налаштувати папки. Обов'язково, якщо декілька налаштовано SSH з'єднань, треба зробити коректне з'єднання дефолтним. Далі щоби скопіювати всю теку проекту одразу, у вікні Project правий-клік на корені з назвою проекту далі перейти в Deployment, далі обрати з'єднання, наприклад, “Upload to polina@192.168.54.100” по якому будемо завантажити проєкт. І далі за вкладкою File Transfer має пройти завантаження всіх файлів. Зауважимо, щоб кожного разу так не робити при змінах в проєкті, краще в “File → Settings → Build, Execution, Deployment → Options” обрати в вкладці “Upload changed files automatically to the default server” флаг “Always”.

Оскільки в файл “requirements.txt” містить бібліотеки, які треба для роботи, то їх відразу треба завантажити на віртуальне середовище “venv”. Де на в терміналі VM треба перейти у папку проєкту, де є файл з потрібними

бібліотеками, які треба встановити на віртуальне середовище: “cd ~/Diploma/Game*”. Активувати це середовище командою “source ~/venv/bin/activate” та встановити бібліотеки командою “pip install -r requirements.txt”.

Зауважимо, що IP-адресу можна перевірити в терміналі за командою “ip a”, і обрати або Host-only або мережевий міст, Bridge. Оскільки спочатку необхідно завантажити бібліотеки краще вибрати IP-адресу мережевого міста.

Останнім етапом треба створити спільну папку, в якій буде зберігатися дані для подальшого аналізу та результати. В VirtualBox, VB, треба перейти на вкладку “Settings → Shared Folders” та створити спільну папку, де вказати шлях на Windows та назву папки. Наприклад, шлях у Windows: “Folder Path: D:\\Diploma\\GameOfDronesDev\\game_of_drones\\data”, та назва спільної папки: “Folder Name: data” та поставити опції Auto-mount, тобто автоматичне підключення спільної папки при запуску VM, та Make Permanent, тобто збереження налаштувань між перезавантаженнями для постійного доступу. Після чого треба запустити VB та створити точку монтування в проєкті командою “sudo mount -t vboxsf data ~/Diploma/GameOfDronesDev/game_of_drones/data_shared” де ім’я спільної папки на Ubuntu буде з ім’ям “data_shared”, а ім’я, яке ми задали в “Folder Name” це є “data”. У подальшому, попередня команда буде автоматично виконуватися в fstab. Отже, треба відкрити цей файл командою “sudo nano /etc/fstab” та написати в кінці рядок “data /home/polina/Diploma/GameOfDronesDev/game_of_drones/data_shared vboxsf defaults 0 0”. Після чого, зберегти файл та перевірити, що ніяких помилок не видасть командою “sudo mount -a”. Після чого перезавантажити VM командою “sudo reboot”. Після чого папка буде автоматично оновлюватися. Однак, щоб користувач міг вносити зміни в папку “data_shared” треба, щоб він мав права на запис. Для цього треба додати його до групи “vboxsf” командою “sudo usermod -aG vboxsf \$USER”. Після перезавантаження VM

командою “sudo reboot”, за допомогою команди “ls -ld /media/sf_data” можна перевірити права доступу, наприклад, вони повинні бути “drwxrwx---”.

Таким чином, було організоване повністю ізольоване та відтворюване середовище розробки для дипломного проєкту. Спершу у гостьовій Ubuntu 22.04 LTS на VirtualBox налаштовано віртуальне середовище Python, інстальовано ROS 2 Humble та OpenSSH-доступ для віддаленої роботи з PyCharm. Далі в IDE створено SSH Interpreter із підключенням до Ubuntu через обраний мережевий інтерфейс або Host-only, або Bridge, синхронізовано директорії проєкту та налаштовано автоматичне перенесення змін у код на віртуальну машину. Наприкінці реалізовано спільну папку між Windows і Ubuntu та додано запис у “fstab” для автоматичного підключення. Така конфігурація гарантує, що весь код, бібліотеки й дані завжди будуть доступні обом ОС без повторних ручних налаштувань, що значно підвищує продуктивність розробки й спрощує відтворюваність результатів.

2.2 Розробка підсистеми комп’ютерного зору

2.2.1 Архітектура CV-модуля

Підсистема комп’ютерного зору виконує три основні функції. Перша функція виконує попередню обробку кадру та виділяє характерні ознаки об’єкта. Друга функція виділяє найімовірніший контур, тобто шукає суцільну замкнуту лінію, що відповідає БПЛА. Остання функція обчислює параметри вимірювання, щоб передати їх в фільтр Калмана.

Гнучкість модулю забезпечує патерн «Стратегія», де створено абстрактні інтерфейси `DetectionStrategy` та `ContourStrategy`. Інтерфейси містять єдиний абстрактний метод: клас `DetectionStrategy` містить метод `process(image)`, клас `ContourStrategy` містить метод `find(image)`, в кожному з яких треба реалізувати конкретні можливі стратегії. Ці методи будуть

відповідати саме за те, щоб детектувати об'єкт на зображенні. Натомість, за допомогою декоратору `@abstractmethod` Python гарантує, що кожен підклас повинен реалізувати метод, інакше об'єкт створити не вдасться.

Отже, реалізовано дві конкретні стратегії: `SobelDetection`, який за оператором Соболя детектує об'єкт, наслідуючи клас `DetectionStrategy`, та `ContourStrategy`, який виділяє контур об'єкта, наслідуючи клас `ContourStrategy`.

Нарешті, у класі `ComputerVision` об'єкти `SobelDetection` та `LargestContour` передаються через конструктор. Сам CV-модуль не здогадується про деталі алгоритмів, а просто викликає `detection_strategy.process(frame)`, отримуючи відповідну маску, яку передає в `contour_strategy.find(mask)`, де отримується відповідний контур, після чого за моментами обчислює центроїд та орієнтацію. Завдяки цьому в будь-який момент можна підмінити стратегії, наприклад, замість Sobel підставити Canny або згорткову мережу, а замість LargestContour – алгоритм з відсіканням «тонких» об'єктів, не змінюючи решту коду, а просто додаючи відповідні класи, який успадкує відповідний абстрактний інтерфейс. Така ізоляція алгоритмів і їхня взаємозамінність і є суттю патерна «Стратегія», що лежить в основі всієї архітектури.

2.2.2 Реалізація Sobel-детектора: алгоритм і код

Для реалізації Sobel-детектора спочатку треба перевести отриманий кадр у відтінки сірого, це робиться для того, бо класичні ядра очікують один канал, тобто якщо залишити кольорове зображення (BGR/RGB), то прийдеться запускати Sobel-детектор тричі, для функції окремо для R-, G- та B-функцій яскравості. Тим самим реалізувати функцію, яка врешті решт об'єднує ці три функції, що є не тільки необов'язковою, а й може викликати

похибки. Адже і природі часто буває різкий контраст лише в одній компоненті.

Після цього окремо для кожного пікселя треба обчислити модуль градієнта, що складається з вертикальної та горизонтальної складової. Кожна з яких має власне ядро Соболя з розмірністю 3×3 . Так, горизонтальна складова ядра Соболя – див. формулу (2.2.1):

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}. \quad (2.2.1)$$

Натомість вертикальна складова ядра Соболя – див. формулу (2.2.2):

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}. \quad (2.2.2)$$

Так, для конкретного пікселя кожні складову обчислюють, розмістивши ядро Sobel так, щоб його центральний елемент збігався саме з координатами цього пікселя, після чого виконують згортку над дев'ятьма значеннями вікна 3×3 – самим пікселем і вісьмома його сусідами. Після чого отримують значення вертикальної та горизонтальної складової градієнта. Якщо горизонтальна складова є додатнім числом, то інтенсивність зростає зліва направо, тобто ліва сторона темніша, права світліша. Якщо є від'ємним, то навпаки інтенсивність зменшується зліва направо, тобто ліва світліша, права темніша. Якщо це значення дорівнює нулю, то по горизонталі яскравість майже не змінюється. Аналогічне пояснення і для вертикальної складової. Якщо вона є додатнім числом, то інтенсивність зростає згори донизу, отже, верх темніший, низ світліший. Якщо є від'ємним, то навпаки інтенсивність зменшується згори донизу, тобто верх світліший, низ темніший. Якщо це значення нульове, то по вертикалі зміна мінімальна.

Отже, нульове значення означає, що у цьому напрямку різкого переходу яскравості немає.

Після цього обчислюється модуль градієнта обчислюють за формулою (2.2.3):

$$G(x, y) = \sqrt{G_x^2(x, y) + G_y^2(x, y)}. \quad (2.2.3)$$

Тож чим більший градієнт, тим різкіше змінюється яскравість, отже лінія контрастніша. Після цього утворюється карта модулів градієнтів. Її перетворюємо на бінарну, порівнюючи кожен піксель із фіксованим порогом – у роботі реалізації це 5: усі значення $G(x, y) \geq 5$ оголошуємо краями, решту – фоном [20]. В коді це реалізовано в класі SobelDetection, який наслідує клас DetectionStrategy, та має вигляд, який зображений на рисунку 2.2.1.

```

from abc import ABC, abstractmethod
import cv2
import numpy as np

class DetectionStrategy(ABC):
    @abstractmethod
    def process(self, image: np.ndarray) -> np.ndarray:
        pass

class SobelDetection(DetectionStrategy):
    def process(self, image: np.ndarray) -> np.ndarray:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        grad_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
        grad_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
        gradient = cv2.magnitude(grad_x, grad_y)
        gradient = cv2.convertScaleAbs(gradient)
        _, binary = cv2.threshold(gradient, 5, 255, cv2.THRESH_BINARY)
        return binary

```

Рисункок 2.2.1 – Виявлення країв на бінарному зображенні

Тож після того як зображення перетворено на бінарну карту країв, наступний крок це виділення силуету дрона. Для цього у стратегії LargestContour використано функцію find(image), яка переглядає білу маску й

повертає всі замкнені лінії. Режим `RETR_EXTERNAL` змушує алгоритм залишити лише зовнішні контури, відкидаючи внутрішні порожнини, а `CHAIN_APPROX_SIMPLE` зберігає лише ключові вершини, що зменшує обсяг даних без втрати геометрії. Отримавши список контурів, алгоритм обирає релевантний за площею. Виклик методу `max` з ключем `cv2.contourArea` повертає контур із найбільшим значенням. Іншими словами найбільшу світлу пляму на кадрі [21]. Якщо ж `find(image)` не знаходить жодної фігури, перехоплюється виняток і повертається `None`, який є сигналом, що на поточному кадрі ціль не детектовано, і фільтр Калмана пропускає фазу оновлення та виконує лише прогноз.

2.2.3 Пошук центроїда цілі та орієнтації

Повернутий масив точок контура одразу використовується для обчислення геометричних моментів, з яких виводяться координати центроїда та орієнтація об'єкта. Для обчислення моментів будується індикаторна функція:

$$f(x, y) = \begin{cases} 1, & \text{якщо піксель білий,} \\ 0, & \text{якщо піксель чорний.} \end{cases} \quad (2.2.4)$$

Тоді прості моменти це звичайні дискретні суми, які знаходяться формулою (2.2.5):

$$m_{pq} = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} x^p y^q f(x, y) \quad (2.2.5)$$

де W – ширина кадру; H – висота кадру.

Тож, m_{00} це кількість білих пікселів, тобто площа об'єкта, m_{10} це перший просторовий момент уздовж осі x , тобто це сума всіх горизонтальних координат білих пікселів силуету, іншими словами «горизонтальний важіль» маси фігури. Натомість m_{01} навпаки просторовий момент уздовж осі y , тобто це сума всіх вертикальних координат білих пікселів, іншими словами «вертикальний важіль» маси. Зауважимо, що x може набувати значень від 0 до $W-1$, а координата y від 0 до $H-1$. Отже, тому центральні моменти знаходяться за формулою (2.2.6):

$$\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}} \quad (2.2.6)$$

Щоб знайти орієнтацію, треба обчислити другі моменти, додаючи до формулу зсув по центроїду, тобто за формулою (2.2.7) буде:

$$\mu_{pq} = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} (x - \bar{x})^p (y - \bar{y})^q f(x, y) \quad (2.2.7)$$

Тоді орієнтацію можна обчислити за формулою (2.2.8):

$$\theta = \frac{1}{2} \tan^{-1} \frac{2\mu_{11}}{\mu_{20} + \mu_{02}} \quad (2.2.8)$$

Застосувавши формули (2.2.5) – (2.2.8) до бінарної маски, було отримано центроїд та орієнтацію об'єкта [22, 23]. Обидва параметри надалі слугують вектором спостереження для Unscented Kalman Filter, забезпечуючи стабільне оновлення стану дрона навіть за часткової втрати контуру.

2.3 Розробка модуля прогнозування траєкторій

2.3.1 Теорія Unscented фільтра Калмана

Ворожа ціль рухається з непередбаченою траєкторією, тож для адекватного прогнозування необхідно оцінювати не лише положення, а й швидкості та прискорення, можливо, додати орієнтацію з кутової швидкістю та прискорення. Тож, задача дипломної роботи передбачає використання динамічного та багатовимірного фільтра Калмана. Проте на практиці моделі руху й вимірювальні функції часто виявляються нелінійними, а розподіли шумів – не ідеально гаусівськими. Тому було реалізовано Unscented фільтр Калмана (UKF), який замість лінеаризації використовує набір «сигма-точок» для точнішого наближення розподілу стану через нелінійності.

Коротко кажучи, UKF генерує навколо поточної оцінки стану набір «сигма-точок» шляхом додавання і віднімання відхилень, що розраховуються за коваріаційною матрицею та спеціальними коефіцієнтами. Так створюється масив «сигма точок», який потім окремо пропускається через функцію стану й вимірювання, після чого результат усереднюють за вагами, де початкова оцінка стану отримує найбільше значення ваг, а інші точки, де конкретний елемент відхилився, отримують однакові ваги.

Отже, згідно формул, то це відхилення, тобто параметр розрахунку «сигма-точок», отримують за формулою (2.3.1):

$$\lambda = \alpha^2(N + k) - N \quad (2.3.1)$$

де N – розмір вектора стану;

α – визначає «розмір» розсіювання сигма-точок навколо середнього, зазвичай ставиться дуже малим, наприклад, $\alpha = 0,001$ в загальному коливається від $\alpha \in (0, 1]$;

k – допоміжний параметр масштабування, часто встановлюється нулем.

Нехай поточна оцінка вектора стану позначимо як x_n та її коваріацію як P_n . Тоді генерація «сигма-точок» обчислюється за формулою (2.3.2):

$$\begin{aligned} \chi_n^{(0)} &= x_n \\ \chi_n^{(i)} &= x_n + \sqrt{(N + \lambda)P_n}, \quad i = 1, \dots, N \\ \chi_n^{(i)} &= x_n - \sqrt{(N + \lambda)P_n}, \quad i = N + 1, \dots, 2N \end{aligned} \quad (2.3.2)$$

Ваги «сигма-точок» для прогнозу середнього обчислюються за формулою (2.3.3):

$$\begin{cases} \omega_0^{(m)} = \frac{\lambda}{N + k} \\ \omega_i^{(m)} = \frac{1}{2(N + \lambda)} \end{cases} \quad (2.3.3)$$

Ваги «сигма-точок» для коваріації обчислюються за формулою (2.3.4):

$$\begin{cases} \omega_0^{(c)} = \frac{\lambda}{N + \lambda} + (1 - \alpha^2) + \beta \\ \omega_i^{(c)} = \frac{1}{2(N + \lambda)} \end{cases} \quad (2.3.4)$$

де β – який враховує апріорні відомості про розподіл шумів.

Причому додається коефіцієнт β , який враховує апріорні відомості про розподіл шумів, тобто якщо похибки мають гаусівський розподіл, то ставиться $\beta = 2$. Однак, якщо розподіл має більш «важкий хвіст» варто взяти $\beta > 2$, щоб збільшити вагу центральної «сигма-точки» в коваріаційних обчисленнях і точніше передає хвостову поведінку розподілу.

Зауважимо, що кожен «сигма-точку» $\chi_n^{(i)}$ окремо пропускається через функцію стану – див. формулу (2.3.5):

$$\chi_{n+1} = f(\chi_n) \quad (2.3.5)$$

Отже, оновлені середнє обчислюється за формулою (2.3.6):

$$x_{n+1} = \sum_{i=0}^{2N} \omega_i^{(m)} \chi_{n+1}^{(i)} \quad (2.3.6)$$

Зауважимо, що кожен «сигма-точку» $\chi_n^{(i)}$ окремо пропускається через функцію стану.

Натомість коваріація обчислюється за формулою (2.3.7):

$$P_{n+1} = \sum_{i=0}^{2N} \omega_i^{(c)} \left(\chi_{n+1}^{(i)} - x_{n+1} \right) \left(\chi_{n+1}^{(i)} - x_{n+1} \right)^T + Q \quad (2.3.7)$$

Після цього χ_{n+1} стає χ_{n-1} , x_{n+1} стає x_{n-1} та P_{n+1} стає P_{n-1} . Тепер можна оновити оцінку стану, але спочатку треба обчислити усереднене вимірювання, використовуючи ваги для прогнозованої точки, тобто $\omega_i^{(m)}$. Тож, аналогічно кожен «сигма-точку», але тепер $\chi_{n-1}^{(i)}$ окремо пропускається вже через функцію вимірювання. Таким чином, формула виглядає так – див. формулу (2.3.8):

$$Z_n = h(\chi_{n-1}) \quad (2.3.8)$$

Далі обчислюємо зважене середнє прогнозу вимірювання – див. формулу (2.3.9):

$$\bar{z}_n = \sum_{i=0}^{2N} \omega_i^{(m)} z_n^{(i)} \quad (2.3.9)$$

Аналогічно змінюється обчислення коефіцієнт Калмана, хоча основа формування коефіцієнт відповідна. Отже, важливим етапом обчислити крос-коваріацію між просторами стану та вимірювання й коваріацію вимірювання – див. формулу (2.3.10):

$$\begin{cases} P_{xz_n} = \sum_{i=0}^{2N} \omega_i^{(c)} (\chi_{n-1}^{(i)} - x_{n-1}) (Z_n^{(i)} - \bar{z}_n)^T \\ P_{z_n} = \sum_{i=0}^{2N} \omega_i^{(c)} (Z_n^{(i)} - \bar{z}_n) (Z_n^{(i)} - \bar{z}_n)^T + R_n \end{cases} \quad (2.3.10)$$

Після цього, обчислюється коефіцієнт Калмана, за значенням якого оновлюється оцінка стану та її дисперсія за формулою (2.3.11):

$$\begin{cases} K_n = P_{xz_n} (P_{z_n})^{-1} \\ x_n = x_{n-1} + K_n (z_n - \bar{z}_n) \\ P_n = P_{n-1} - K_n P_{z_n} K_n^T \end{cases} \quad (2.3.11)$$

Отже, Unscented фільтр Калмана дозволяє точно відтворювати середнє та коваріацію крізь будь-які гладкі нелінійності до другого порядку без необхідності в обчисленні Якобіанів, як це є в розширеному фільтрі Калмана. Однак із ростом розмірності кількість сигма-точок зростають як $O(N^3)$. Оскільки в дипломній роботі розмірність вектора стану є відносно невеликою, UKF залишається ефективним і забезпечує високу точність оцінки без значних обчислювальних витрат.

Таким чином, UKF поєднує простоту реалізації з високою точністю для задач прогнозування траєкторій рухомих цілей із шумними та нелінійними спостереженнями.

2.3.2 Моделі руху та їхні рівняння

У дипломній роботі було реалізовано дві моделі руху цілі. Перша - спрощена, яка враховує положення в площині координати x та y , їх швидкості та прискорення. Друга - розширена, де додається орієнтація, обчислена за другими моментами, її швидкість та прискорення.

В спрощеній моделі вектор стану x , матриця стану F та матриця вимірювання H зображено на рисунку 2.1:

$$x = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix}, F = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2} \Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2} \Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Рисунок 2.1 – Модель векторів стану та матриць стану та вимірювання для спрощеної моделі руху

Зауважимо, що x та y це координати центру цілі в нормованих одиницях кадру; v_x , v_y це лінійні швидкості по осях абсцис та ординат; a_x , a_y це лінійні прискорення по осях; Δt це тривалість кроку прогнозу тобто часова різниця між кадрами.

Зазначимо, що у матриці переходу F блоки відповідають рівнянням, які зображені на рисунку 2.2:

$$\begin{cases} x_{k+1} = x_k + v_k^x \Delta t + \frac{1}{2} a_k^x \Delta t^2 \\ y_{k+1} = y_k + v_k^y \Delta t + \frac{1}{2} a_k^y \Delta t^2 \\ v_{k+1}^x = v_k^x + a_k^x \Delta t \\ v_{k+1}^y = v_k^y + a_k^y \Delta t \\ a_{k+1}^x = a_k^x \\ a_{k+1}^y = a_k^y \end{cases}$$

Рисунок 2.2 – Формули переходу для координат, їх швидкостей та прискорень для спрощеної моделі руху з часовим кроком

Матриця вимірювання H відбирає ті компоненти стану, які реально вимірюються, тобто у спрощеній моделі тільки координати x та y , які отримуються за допомогою комп'ютерного зору.

У розширеній моделі вектор стану x , матриця стану F та матриця вимірювання H зображено на рисунку 2.3:

$$x = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \\ \Theta \\ \omega \\ \alpha \end{bmatrix}, F = \begin{bmatrix} 1 & 0 & \cos(\Theta)\Delta t & 0 & \frac{1}{2}\cos(\Theta)\Delta t^2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \sin(\Theta)\Delta t & 0 & \frac{1}{2}\sin(\Theta)\Delta t^2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Рисунок 2.3 – Модель векторів стану та матриць стану та вимірювання для розширеної моделі руху

Зауважимо, що θ це орієнтація об'єкта (кут повороту) в радіанах; ω це кутова швидкість; α це кутове прискорення.

Зазначимо, що у матриці переходу F блоки відповідають рівнянням, які зображені на рисунку 2.4:

$$\left\{ \begin{array}{l} x_{k+1} = x_k + \cos(\Theta_k)v_k^x \Delta t + \frac{1}{2} \cos(\Theta_k)a_k^x \Delta t^2 \\ y_{k+1} = y_k + \sin(\Theta_k)v_k^y \Delta t + \frac{1}{2} \sin(\Theta_k)a_k^y \Delta t^2 \\ v_{k+1}^x = v_k^x + a_k^x \Delta t \\ v_{k+1}^y = v_k^y + a_k^y \Delta t \\ a_{k+1}^x = a_k^x \\ a_{k+1}^y = a_k^y \\ \Theta_{k+1} = \Theta_k + \omega_k \Delta t + \frac{1}{2} \alpha_k \Delta t^2 \\ \omega_{k+1} = \omega_k + \alpha_k \Delta t \\ \alpha_{k+1} = \alpha_k \end{array} \right.$$

Рисунок 2.4 – Формули переходу для координат та орієнтації, їх швидкостей та прискорень для розширеної моделі руху з часовим кроком

Матриця вимірювання H в розширеній моделі аналогічно попередній відбирає ті компоненти стану, які реально вимірюються, тобто в цій моделі обчислюється ще й орієнтація об'єкта за допомогою комп'ютерного зору. Однак, зауважимо, що кут θ обчислюється за допомогою других моментів, однак може давати неоднозначність, наприклад, вісь крила трактуватиметься як ніс. У таких випадках фільтр вважатиме, що об'єкт “застиг” або обертається на місці, хоча насправді його орієнтація не змінюється. У разі нечіткої сегментації контуру або симетрії форми оберт обчислюється в іншому напрямку, хоча реальна орієнтація може не змінюватися. У рамках цієї дипломної роботи ця двозначність виявлена, але не усувається. Проте в подальших дослідженнях можливе поєднання оцінки через моменти з нейромережею (наприклад, PoseCNN [16]) чи зі схемою Perspective-n-Point (PnP) [17], що забезпечить більш стійке та точне визначення напрямку «носу» об'єкта.

2.3.3 Архітектура та реалізація UKF у Python з патернами

У роботі для реалізації фільтра Калмана застосовано три основні патерни проектування.

Стратегія (Strategy), де інкапсулюється в окремі класи різні моделі руху, щоб фільтр можна було перевантажувати без зміни його внутрішньої логіки. Реалізується інтерфейс «MotionModelStrategy» із методами `f_state`, який є нелінійною функцією переходу стану, на основі якого UKF прогнозується новий вектор стану, `h_state`, який є нелінійною функцією вимірювання, що перетворює стан у простір вимірювань, `initialize_filter`, метод який повертає початкові матриці коваріації стану, процесного шуму й шуму вимірювань для UKF, `count_element_measurement`, метод який повертає вимірювання, що були обчислені за допомогою комп'ютерного зору, та `predict_steps`, метод, який прогнозує вектор на динамічну кількість кроків вперед, однак без використання «сигма-точок», водночас метод `predict_steps_generate_sigma_points` навпаки генерує прогноз за допомогою «сигма-точок». Зауважимо, що метод `predict_steps_generate_sigma_points`, єдиний, який був реалізован у класі, а не у моделях-нащадках. Конкретні моделі-нащадки, `FullMotionModel` та `SimpleMotionModel`, реалізують ці методи і передаються в конструктор `UnscentedFilterKalman`. Так, наприклад, на рисунку 2.5 зображено UKF інтегрується з патерном «Стратегія», як саме змінювати моделі руху:

```
from unscented_filter_kalman import UnscentedFilterKalman
from motion_model import FullMotionModel, SimpleMotionModel
# Для моделі з орієнтацією:
ukf_full = UnscentedFilterKalman(motion_model=FullMotionModel())
# Для спрощеної моделі:
ukf_simple = UnscentedFilterKalman(motion_model=SimpleMotionModel())
```

Рисунок 2.5 – Інтеграція UKF з патерном «Стратегія»

Крім, того було реалізовано патерн проектування як спостерігач (Observer), він, який організує підписку на ROS2-топіки та передає отримані повідомлення в Mediator без прямої залежності від деталей ROS2 у самому модулі прогнозування.

Окрім цього було реалізовано патерн посередник (Mediator), який координує взаємодію між усіма компонентами, а саме: отриманням кадрів і статусу камери через клас Observer, обробкою їх у класі ComputerVision, передачею результатів у клас UnscentedFilterKalman, синхронізацією з реальними даними та виведенням результатів у класах AnalysisResults, який виводить графіки похибок та зберігає їх у спільній папці, VisualResults, який малює відео, які після цього зберігає клас Mediator, у спільній папці.

Застосування цих патернів дозволяє зберегти гнучкість і розширюваність системи: легко міняти модель руху, додавати нові канали отримання чи обробки даних, а також структурувати код та винести його в окремі класи.

Щодо реалізації UKF у Python, то перед першим кроком прогнозу, треба обчислити початкову оцінку стану, часовий діапазон між двома найпершими точками, які комп'ютерний зір знайшов.

Після того, як це було обчислено задано початкові матриці невизначеності стану, похибки процесного шуму та похибки вимірювання. Зауважимо, що початкова коваріація стану ініціалізується як одинична матриця помножена на велику константу, в даній реалізації це значення 100. Таке вилеке значення повинно відобразити слабку впевненість у першій оцінці стану й дозволити фільтру «розбігтися» на початку. Крім того, процесний шум задається формується діагонально з параметрами. Коваріація шуму вимірювань формується теж діагональна, зі значеннями, що відповідають очікуваній точності. Наприклад, для координат похибка в 1 піксель, якщо є орієнтація то похибка в 5 градусів.

Звернемо увагу на те, що для генерації точок для коефіцієнтів було обрано такі значення як: $\alpha = 0,001$, $\beta = 2$, $k = 0$. Потім за формулами, які

були описані вище, обчислюється оцінка вектора стану. Однак варто зауважити, що якщо комп'ютерний зір не знайшов об'єкт, а початкова ініціалізація вже відбулася, то модель буде оновлюватися виключно за прогнозом моделі руху.

У Python-реалізації UKF матриці коваріації шумів процесу та вимірювань не є статичними – вони коригуються після оновлення, щоб в теорії допомагає зменшити раптові стрибки, тим самим допомагає краще адаптувати фільтр до реальних умов. Відтак підвищить стабільність фільтра в умовах, коли вимірювання різної якості, а рух об'єкта – непередбачуваний. Для обох оновлень використовується фільтр експоненційного згладжування зі згладжувальними коефіцієнтами, обидва яких дорівнюють 0,95. Оскільки значення коефіцієнтів близьке до одиниці, це повинно забезпечити, що значення матриць Q та R змінювалися повільно, аби забезпечити більшу стабільність поведінки фільтра, коли вимірювання різної якості або непередбаченому русі об'єкта. Адже жорстко зафіксовані коваріації Q та R можуть не відповідати реальній динаміці об'єкта чи стану камери, наприклад, при раптовому пориву вітра або несподіваному падінні якості сегментації. Адаптивне оновлення, теоретично, дозволить фільтру «підлаштуватися» під нові умови. За відсутності згладжування коваріаційної матриці її елементи можуть раптово змінюватися між кадрами, що викликає коливання невизначеності оцінки стану – або надто завищеної, або навпаки, заниженої. Використання значень близько до 0,95 дозволяє балансувати між адаптивністю та стабільністю. В коді це виглядає як на рисунку 2.6:

Отже, було реалізовано патерни «Стратегія», «Спостерігач» та «Посередник», які побудували гнучку архітектуру модуля прогнозування на Python. Конкретна кінематика руху інкапсулюється в класах-нащадках інтерфейсу `MotionModelStrategy`, прийом даних із ROS2-топиків винесено в `Observer`, а узгодження всіх компонентів здійснює `Mediator`. Було реалізовано `Unscented Kalman Filter`. Додатково матриці процесного та вимірювального шуму динамічно оновлюються за допомогою експоненційного

згладжуванням, що забезпечує плавність змін і стійкість фільтра в умовах шумних або нерівномірних даних.

```
self.alpha_R = 0.95
self.beta_Q = 0.95

# Оновлення R (коваріація шуму вимірювань)
measurement_residual = self.z.reshape(-1, 1) - z_pred
self.R = self.alpha_R * self.R + (1 - self.alpha_R) * (measurement_residual @ measurement_residual.T)

# Оновлення Q (коваріація процесного шуму)
state_residual = self.x.reshape(-1, 1) - x_pred.reshape(-1, 1)
self.Q = self.beta_Q * self.Q + (1 - self.beta_Q) * (state_residual @ state_residual.T)
```

Рисунок 2.6 – Реалізація динамічних матриць коваріацій

2.4 Алгоритм наведення камери-дрона

На практиці виявилось, що під час першої спроби реалізації наведення в Unity значення лінійної швидкості й прискорення камери обчислювалися шляхом прямого диференціювання координат кадр-за-кадром. Однак результати виявилися надто шумними: навіть незначна похибка у кілька пікселів, поділена на дуже короткий інтервал часу – близько тридцяти трьох мілісекунд – давала швидкість у десятки метрів за секунду та прискорення у сотні метрів за секунду на секунду. Тому було зроблено усереднення за декілька кроків, інтервальне вимірювання швидкості, згладжування швидкості та прискорення за допомогою експоненціального згладжування, комбінування методів згладжування, однак це не дало покращення. Через це величини швидкості та прискорення «стрибали» на порядки вище реальних, що призводило до надмірного відхилення точки наведення й в результаті втрати цілі. Тож було прийнято рішення перенести розрахунки у Python-вузол та зробити наведення не на прогнозовану точку, а виключно на

значення, що показав комп'ютерний зір. В результаті це значно покращило стабільність системи.

Отже, Python-вузол маж назву CameraController. Unity же надсилає у ROS 2 стислий кадр, оптичні параметри камери та її поточну позу у форматі PoseStamped та також для побудови графіка перехоплення цілі й поточну позицію самої цілі. У відповідь отримує нову позу тієї ж камери, згенеровану згаданим вузлом. У Python кадр одразу переводиться у відтінки сірого, після чого оператор Sobel дає карту градієнтів. Найконтрастніша пляма визначає положення цілі, її нормалізований центроїд множиться на розмір кадру й перетворюється на піксельні координати. Однак для перетворення пікселів у кути, за якими треба буде повертати камери і рухати дрон-перехоплювач на ціль, необхідно знати фокусну відстань - див. (2.4.1)

$$f_x = \frac{W}{2 \tan(\frac{FOV_h}{2})}, f_y = \frac{H}{2 \tan(\frac{FOV_v}{2})} \quad (2.4.1)$$

де W – горизонтальна роздільна здатність кадру, пікс;

H – вертикальна роздільна здатність кадру, пікс;

FOV_h – горизонтальний кут огляду камери, рад;

FOV_v – вертикальний кут огляду, рад.

Далі конвертуються пікселі у кути за формулою (2.4.2):

$$\theta = \arctg\left(\frac{y - H/2}{f_y}\right), \psi = \arctg\left(\frac{x - W/2}{f_x}\right) \quad (2.4.2)$$

де W – горизонтальна роздільна здатність кадру, пікс;

H – вертикальна роздільна здатність кадру, пікс;

x, y – координати центроїда цілі на кадрі.

Після того, як було переведено піксельні координати в кути [32], то вузол `CameraController` виконує дві різні дії залежно від обраного режиму. У режимі стеження камера лише обертається навколо власної осі, виконуючи плавну корекцію кутів без зміни позиції, щоб утримувати ціль у центрі кадру. Натомість нові орієнтації надсилаються в Unity кожні два кадри. Чи режим зближення, який зміщує дрон-перехоплювач вздовж своєї фронтальної осі з фіксованою швидкістю. Однак щоб не перевантажувати мережу, нова позиція камери публікується лише раз на три кадри. При цьому в повідомленні `PoseStamped` передаються і оновлені координати, і кватерніон із новими кутами. Скрипт `CameraTargetSubscriber` у Unity приймає пакет та безпосередньо присвоює позицію й обертання об'єкта-камери. У результаті сцена демонструє швидке зближення: камера різко фіксує ціль у центрі кадру й починає прямолінійно скорочувати відстань.

Зауважимо, аби уникнути мерехтіння, вузол відсікає центроїди, що опинилися ближче ніж 20 пікселів до краю кадру, та згладжує кути експоненціальним фільтром.

У межах дипломного проєкту розроблено підсистему, що автоматично утримує ворожий БПЛА у центрі кадру й, за потреби, виконує швидке зближення для ураження. Вона замикає відеоконтур між віртуальною сценою Unity та Python-модулем: Unity передає стислий кадр і поточну позицію камери через ROS 2, а у відповідь отримує команду з новими координатами й орієнтацією камери. Усе спілкування здійснюється повідомленнями `CompressedImage` і `PoseStamped`. Таким чином реалізовано два пов'язані процеси: детекція центроїда та наведення камери за законом прямого наведення в піксельному просторі. Механізм працює незалежно від прогнозу UKF, однак залишається сумісним із ним: якщо в майбутньому фільтр продукуватиме прогнозований центроїд, `CameraController` зможе спрямувати камеру й за цією точкою. Таким чином, підсистема наведення поєднує просту, але робастну візуальну детекцію з мінімалістичним законом керування, працює на бюджетному одноплатнику й підтримує оновлення

позиції камери з частотою відеопотоку без помітних ривків чи втрачених кадрів. Подальші роботи планується зосередити на інтеграції фільтра Калмана, щоб поєднати детекцію з прогнозом траєкторії, а також на випробуванні алгоритму на реальному БПЛА.

Висновки до розділу 2

У цьому розділі було сформовано й докладно описано повний технологічний ланцюжок – від інструментальної бази до алгоритмічних рішень, – що забезпечує відтворювані експерименти з автоматичного перехоплення БПЛА. Вибір Ubuntu 22.04 LTS у поєднанні з ROS 2 Humble та запуском у VirtualBox дав змогу ізолювати середовище, гарантувати сумісність пакетів і водночас спростити розгортання на інших машинах. Автоматизація встановлення (репозиторії, GPG-ключі, ros-dev-tools) та інтеграція з PyCharm через SSH-інтерпретер показали, що навіть складну гібридну конфігурацію Windows ↔ Linux можна налаштувати «в один клік» і підтримувати без ручних копіювань коду завдяки автоматичному деплою та спільній папці vboxsf.

Графічна складова симуляції реалізована на Unity HDRP 6000.0 із пакетом RosSharp як транспортом WebSocket. Це дало дві важливі переваги: реалістичне відтворення освітлення та геометрії, необхідне для тестування CV-алгоритмів, і нативну роботу з повідомленнями ROS 2 типів CompressedImage та PoseStamped. Архітектура «дві камери + Cinemachine» дозволила розділити рендеринг відеопотоку й кінематику динамічної камери-дрона, що спростило експерименти зі швидкими маневрами.

Алгоритмічна частина побудована на трьох патернах. «Стратегія» в CV-модулі ізолює оператор Sobel та алгоритми побудови контуру, даючи змогу підміняти їх без зміни загального коду. «Спостерігач» інкапсулює усі

підписки на ROS 2-топіки, а «Посередник» координує обмін даними між вузлами, що істотно зменшило зв'язність компонентів і спростило налагодження. У модулі прогнозу введено Unscented Kalman Filter: адаптивне оновлення матриць Q та R (експоненційне згладжування з коефіцієнтами 0,95) дало стійку роботу при флуктуаціях якості зображення й різких змінах траєкторії.

При переході до підсистеми наведення виявлено, що первинне диференціювання координат у Unity спричиняло «стрибки» швидкостей та прискорення. Тож перенесення усіх розрахунків у Python-вузол, а саму камеру керування переведено з прогнозу на «пряму» корекцію за центроїдом, обчисленим CV-модулем, значно покращило стабільність системи.

Таким чином, побудовано масштабовану, модульну та повністю відтворювану платформу, яка підтвердила спроможність інтегрованої зв'язки «детекція - UKF-прогноз - донаведення» у складних динамічних сценаріях. До подальших кроків належать: перенесення стеку на апаратну плату реального БПЛА, інтеграція даних IMU у стан-вектор UKF і розширення CV-модуля нейромережею сегментації для роботи на фоні складних ландшафтів.

РОЗДІЛ 3 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Результати роботи комп'ютерного зору

У симуляцію об'єкт виявлявся на зображеннях за допомогою оператора Соболя. Подальше обчислення центроїда об'єкта відбувалося за допомогою математичних моментів. Для оцінки точності виявлення, істинне положення центроїда було отримано з середовища Unity, трансформовано у піксельні координати та передано разом із часовою міткою через ROS2. У Python коді було виконано синхронізацію за часом і зіставлено розрахований та реальний центроїди. Отримані результати наведено на рисунку 3.1, де по горизонтальній осі відкладається номер кадру, що відповідає порядку надходження зображень із симуляції Unity через ROS2. По вертикальній осі зображено нормалізовані координати центроїда об'єкта на зображенні: верхній графік відображає координату абсцис, натомість нижній – координату ординат.

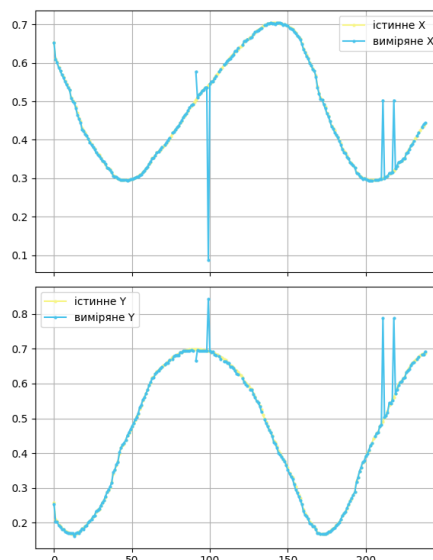


Рисунок 3.1 – Порівняння істинних та вимірних координат центроїда об'єкта

Аналізуючи графік, можна помітити, що навіть за умови однорідного фону виміряні координати центроїда місять викиди. Це спричинено через малий розмір об'єкта на зображенні, що ускладнює детекцію, тому для її покращення треба зменшити кут огляду. Також спостерігаються випадки, коли оператор Соболя взагалі не виявляє БПЛА, що проявляється у вигляді розривів на графіку. Тож в таких випадках, доречно, або згладити результати, коли виникають викиди, або продовжити логічну траєкторію об'єкта, аналізуючи його минулу траєкторію. Тобто необхідно прогнозувати положення об'єкта на основі його попередньої траєкторії, тим самим згладжуючи можливі викиди та зменшуючи похибки вимірювання.

3.2 Результати прогнозування траєкторії

Застосування Unscented Kalman Filter (UKF) дозволило згладити шуми та забезпечити стійке відстеження об'єкта без розривів. Отримані результати, продемонстрували покращення точності відстеження об'єкта. Вони згладили шум та відновили положення, де спостерігалися розриви в даних. Результати прогнозування положення об'єкта на кілька кроків уперед можуть бути використані у подальших дослідженнях для реалізації наведення на передбачувану траєкторію. З цією метою було побудовано графіки, на яких розраховано середньоквадратичну похибку (MSE) між прогнозованими та реальними координатами для кожного конкретного кроку вперед. Тим самим, дозволивши проаналізувати надійність прогнозу та його стабільність.

Отже, для оцінки прогнозу траєкторій, було протестовано дві моделі руху. Графік спрощеної моделі руху зображений на рисунку 3.2.

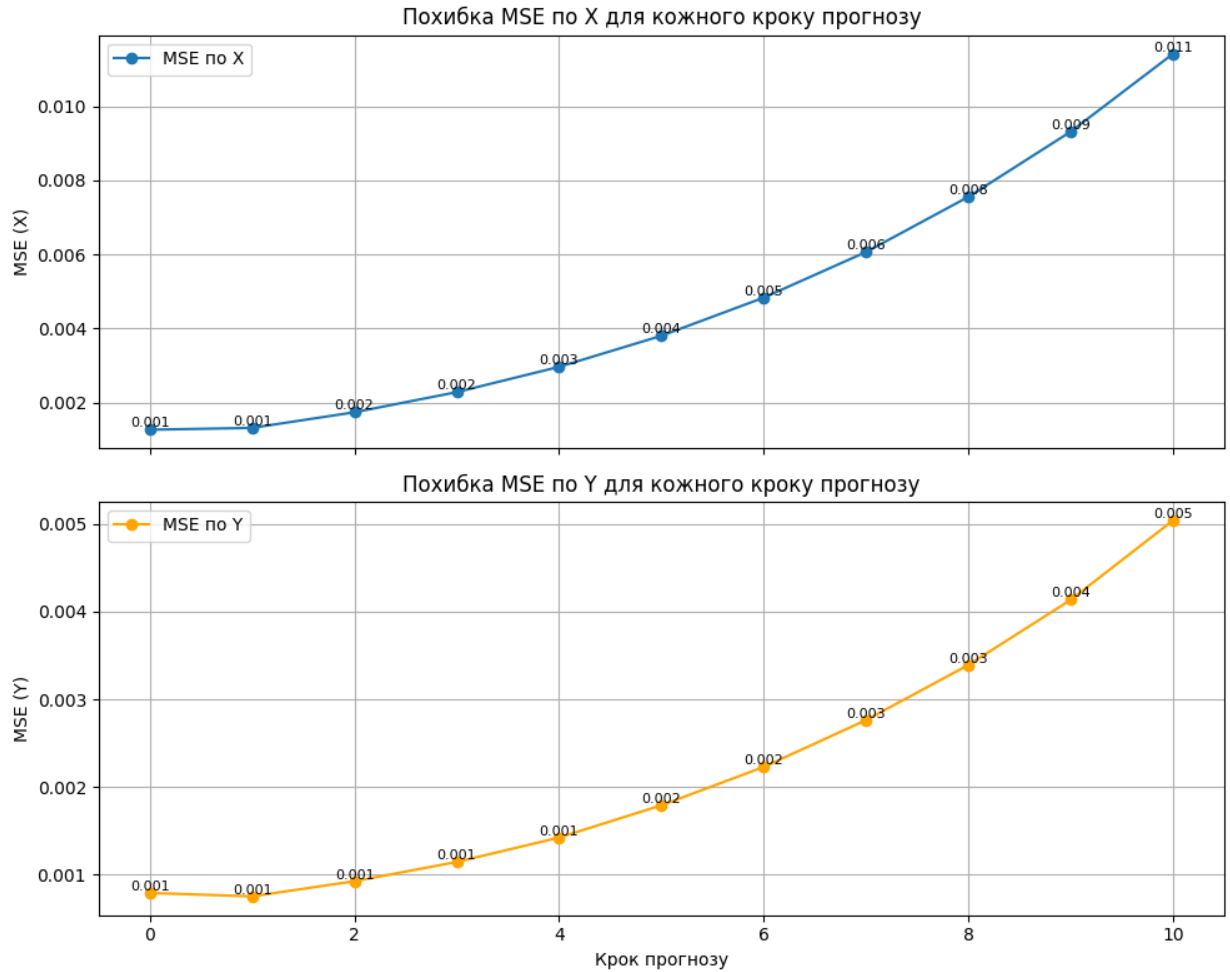


Рисунок 3.2 – Залежність середньоквадратичної похибки координат центроїда від кроку прогнозу для моделі спрощеного руху

Отже, можна спостерігати, що похибка зростає з кожним наступним кроком прогнозу. Водночас, якщо врахувати, що зображення має роздільну здатність 1280×720 пікселів, то навіть на десятому кроці похибка за віссю абсцис не перевищує 15 пікселів, а за віссю ординат – 4 пікселів, що свідчить про достатньо високу точність прогнозування.

Графік розширеної моделі руху зображений на рисунку 3.3.

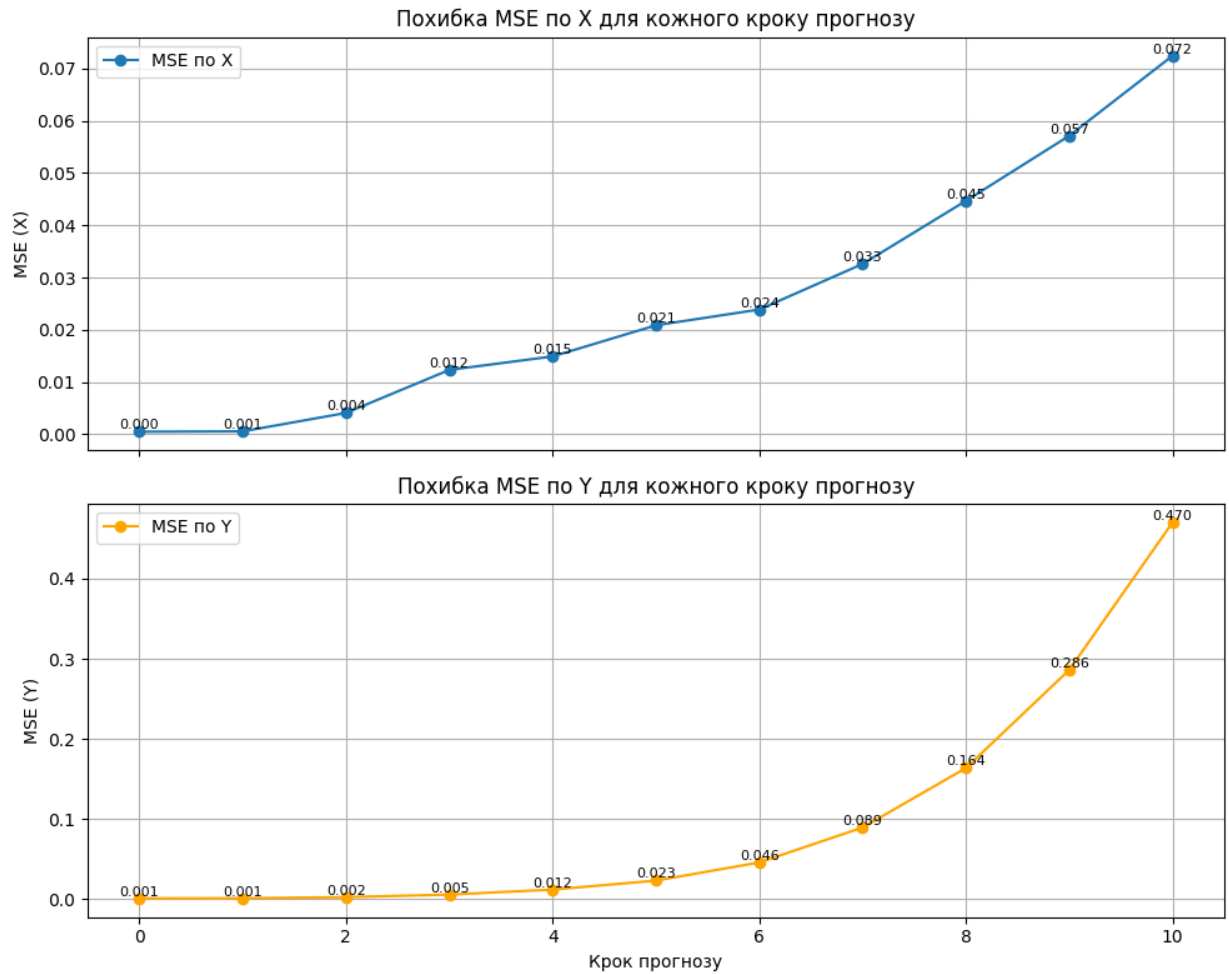


Рисунок 3.3 – Залежність середньоквадратичної похибки координат центроїда від кроку прогнозу для моделі з урахуванням орієнтації

Результати розширеної моделі при зростанні кроку прогнозування є менш точними у порівнянні зі спрощеною моделлю. Так, на десятому кроці середньоквадратична похибка за віссю абсцис досягає понад 92 пікселів, а за віссю ординат – перевищує 338 пікселів, що становить майже половину висоти зображення. Хоча й на перших кроках, тобто у короткостроковому прогнозуванні, точність обох моделей є співставною. Однак у розширеній моделі похибка зростає значно швидше зі збільшенням кроку прогнозування. Це зумовлено тим, що, попри теоретичну перевагу використання орієнтації об'єкта як додаткової ознаки, її обчислення на основі моментів зображення виявилось недостатньо надійним. Зокрема, в окремих випадках напрям

орієнтації було визначено некоректно – наприклад, при плутанні носової частини БПЛА з його крилом, що суттєво вплинуло на якість прогнозу.

3.3 Результати моделювання перехоплення

Для демонстрації працездатності всієї системи було змодельовано базову ситуацію перехоплення з використанням стратегії прямого наведення. Суть якої полягає в тому, що дрон-перехоплювач у кожен момент часу орієнтується безпосередньо на поточну позицію цілі та рухається в її напрямку з фіксованою швидкістю.

На рисунку 3.4 наведено тривимірну траєкторію цілі та дрона-перехоплювача у системі координат. Ціль рухалася по коловій траєкторії з постійною швидкістю. Дрон-перехоплювач, швидкість якого була вищою, поступово коригував курс і наближався до цілі.

Цей приклад демонструє базову ефективність стратегії навіть без прогнозу. Однак варто зазначити, що якщо швидкість перехоплювача перевищує швидкість цілі, тоді перехоплення буде гарантовано.

Крім того, проведено експеримент із наведенням виключно за допомогою комп'ютерного зору, тобто дрон-перехоплювач орієнтувався не на ідеальні координати з Unity, а винятково на центроїд, обчислений оператором Sobel. На рис. 3.4 червона пунктирна лінія відображає саме цю «vision-only» траєкторію, хоча й попри шумні детекції й пропуски кадрів, дрон стабільно втримував ціль у центрі та успішно зближувався.

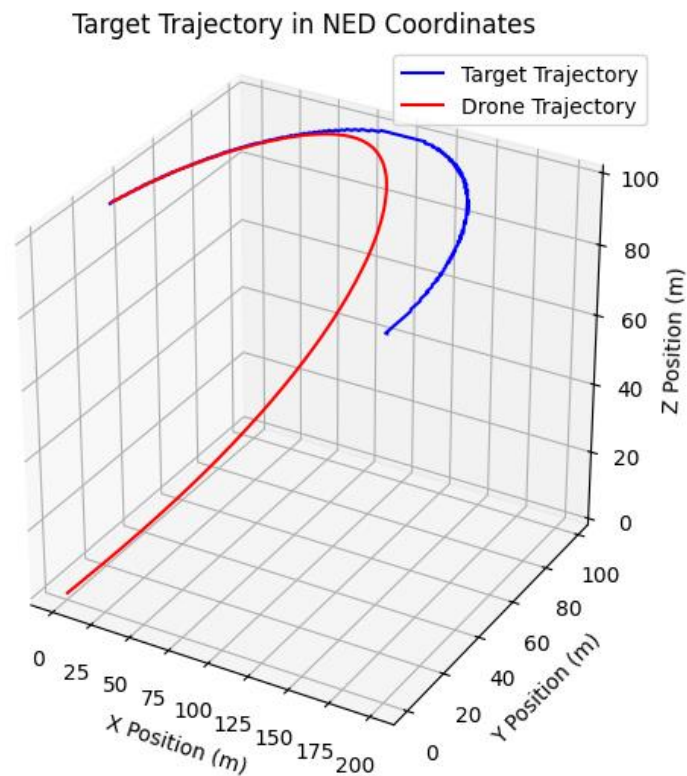


Рисунок 3.4 – Тривимірна траєкторія цілі та дрона-перехоплювача в системі координат NED за еталонними даними Unity та за стратегією прямого наведення

Така апроксимація демонструє, що навіть за відсутності прогнозу Unscented Kalman Filter чистий pure-pursuit на основі CV-даних спроможний привести перехоплювач у задану зону ураження, однак з набагато більшою швидкістю, ніж швидкість цілі.

У подальших дослідженнях доцільно провести порівняльний аналіз ефективності стратегії наведення на прогнозовану точку, визначену за допомогою Unscented Kalman Filter. Такий підхід дозволить оцінити, наскільки використання прогнозу покращує здатність до перехоплення у складніших сценаріях, зокрема за умов часткової втрати візуального контакту або наявності завад.

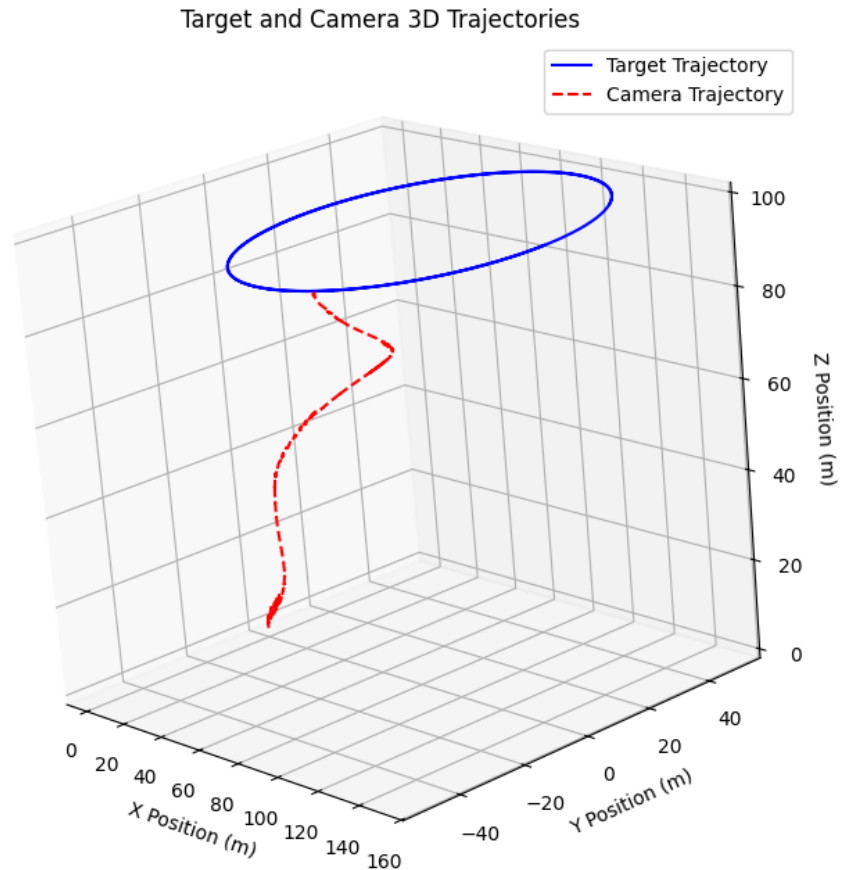


Рисунок 3.5 – Тривимірна траєкторія цілі та дрона-перехоплювача в системі координат NED, отримана виключно на основі даних комп'ютерного зору та прямого закону наведення

Висновки до розділу 3

У цьому розділі було: проведено аналіз точності вимірювання координат центроїда за допомогою комп'ютерного зору, здійснено порівняльний аналіз ефективності реалізованих моделей прогнозування на декілька кроків уперед та також досліджено базову ефективність стратегії перехоплення навіть без прогнозу.

Зокрема, модуль комп'ютерного зору на основі градієнтного оператора Собеля продемонстрував високу стабільність у простих умовах – ціль виявлялася у більшості кадрів, що забезпечило регулярне оновлення стану фільтра. Проблеми виникали лише за наявності викидів або втрати чіткого контуру. Саме тому в подібних випадках фільтр UKF автоматично переходив у режим автономного прогнозування, компенсуючи відсутність вимірювань. А у разі наявності викидів згладжував вимірювання, поєднуючи інформацію з поточного вимірювання та власного прогнозу.

Порівняння спрощеної та розширеної моделей руху продемонструвало, що врахування орієнтації не завжди покращує точність через похибку в обчисленні орієнтації. Натомість спрощена модель, попри свою простоту, забезпечувала стабільніше прогнозування.

Змодельоване перехоплення на основі стратегії прямого наведення підтвердило її ефективність за умови переваги в швидкості. Крім того, реалізована підсистема прямого наведення камери-дрона на основі даних комп'ютерного зору забезпечує успішне зближення з ціллю. Однак для складніших сценаріїв, наприклад, коли ціль ухиляється від дрона або ціль рухається не з постійною швидкістю, або дрон-перехоплювач не може набирати достатню швидкість і треба направлятися не на поточку точку, тоді необхідна інтеграція з фільтром Калмана для врахування майбутньої позиції цілі.

Таким чином, реалізована система продемонструвала працездатність усіх ключових компонентів – від виявлення об'єкта та прогнозування до моделювання перехоплення – та підтвердила потенціал для подальшого розвитку й ускладнення сценаріїв.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

4.1 Постановка задачі

У рамках функціонально-вартісного аналізу (ФВА) перед нами стоїть завдання оптимізації реалізації програмного рішення для моделювання процесу перехоплення ворожого БПЛА іншим БПЛА в симульованому середовищі. Розроблювана система має забезпечити виявлення цілі, прогнозування її траєкторії та реалізацію маневру перехоплення в умовах, максимально наближених до реального часу.

Дана система повинна інтегрувати декілька модулів, які взаємодіють між собою у межах єдиної архітектури: комп'ютерний зір для обробки зображення з камери, фільтр Калмана для прогнозу координат, моделі наведення для розрахунку оптимальної траєкторії, а також симуляційне середовище, реалізоване в Unity, для тестування візуального контролю та поведінки агента-перехоплювача.

Технічні вимоги до програмного продукту:

- стабільне виявлення цілі за допомогою класичних або глибинних методів комп'ютерного зору в симульованому середовищі;
- реалізація фільтра прогнозування траєкторії цілі (Unscented Kalman Filter), який враховує шум у вимірюваннях і дозволяє працювати з неповними або втраченими даними;
- підтримка базових та розширених моделей руху цілі (із змінною швидкістю, кутовим прискоренням, можливими різкими маневрами);
- використання стратегій наведення (Pure Pursuit, а також розширюваність до PN або MPC);
- інтеграція візуального контролю камери у Unity для слідкування за ціллю в реальному часі;

- синхронізація між ROS2 (модулі аналізу та прогнозування) та Unity (візуалізація, взаємодія) через websocket-зв'язок;
- забезпечення логування результатів, збереження відео, графіків та аналітики для подальшого аналізу.

Для вирішення поставленої задачі необхідно виконати наступні кроки:

- провести аналіз методів виявлення та відстеження об'єктів на зображеннях в режимі реального часу;
- реалізувати модуль прогнозування траєкторії цілі з урахуванням можливих втрат даних;
- впровадити механізм керування камерою в Unity з урахуванням обчислених координат цілі;
- забезпечити обчислення орієнтації та швидкості дрона-перехоплювача на основі вхідних координат;
- оцінити ефективність варіантів реалізації програмного рішення на основі технічних параметрів (затримка, пам'ять, обсяг коду) та повної вартості розробки.

Таким чином, задача полягає не лише у створенні працездатної моделі перехоплення цілі, а й у виборі оптимального варіанту реалізації з точки зору співвідношення «якість / витрати». Саме для цього у подальших підрозділах буде проведено техніко-економічний аналіз з обчисленням показників ефективності.

4.2 Обґрунтування функцій програмного продукту

У цьому підрозділі розглядається обґрунтування основних функцій інформаційної системи, що була реалізована в межах дипломної роботи. Метою системи є створення симуляційного середовища для виявлення,

супроводу та перехоплення ворожого БПЛА іншим БПЛА з використанням комп'ютерного зору, прогнозування траєкторії та засобів моделювання у Unity з підтримкою двостороннього обміну даними через ROS2.

Для виконання зазначених задач програмний продукт має забезпечувати виконання таких основних функцій:

- F1 – виявлення об'єкта на зображенні;
- F2 – прогнозування траєкторії цілі;
- F3 – візуалізація та інтеграція з симуляцією в Unity;
- F4 – наведення дрона-перехоплювача на ціль.

Кожна з функцій може бути реалізована кількома альтернативними способами. Нижче наведено варіанти реалізації для кожної функції, а також їх порівняння за перевагами та недоліками.

F1 – виявлення об'єкта на зображенні:

- А — оператор Собеля та математичні моменти (реалізовано). Такий спосіб простий у реалізації, дозволяє обчислити центроїд і орієнтацію об'єкта на основі градієнтів;
- В — глибинна модель (YOLO/SSD) (альтернатива). Такий спосіб має вищу точність, стійкість до шумів, але потребує навчання та GPU-ресурсів.

F2 – прогнозування траєкторії:

- А — Unscented Kalman Filter (UKF) (реалізовано). Такий спосіб забезпечує точне згладжування та прогноз за умов нелінійної динаміки й зашумлених вимірювань.;
- В — Extended Kalman Filter (EKF). Такий спосіб є простішою у реалізації, менш точний при різких змінах траєкторії.

F3 – візуалізація та інтеграція з симуляцією в Unity:

- А — Unity + ROS2 (реалізовано). Це потужна візуалізація сцени та двостороння передача даних між Unity та ROS2 через RosSharp;

- В — TCP/UDP з'єднання (альтернатива). Такий спосіб є простим прямим з'єднанням, з меншою затримкою, але без структури та масштабованості.

F4 – наведення дрона-перехоплювача на ціль:

- А — алгоритм прямого наведення за вектором напрямку (реалізовано). Реалізований у Python, адаптивний до зміни положення цілі, протестований у моделі кругового руху;
- В — Rule-based логіка (альтернатива). Це просте логічне сценарне управління, що не адаптується до змін траєкторії.

На основі порівняльного аналізу було обрано реалізацію варіантів А для всіх функцій (F1–F4), оскільки вони забезпечують баланс між продуктивністю, точністю та практичною досяжністю реалізації у межах дипломного проєкту.

В результаті функціонально-вартісного аналізу сформовано дві допустимі комбінації реалізації програмної системи:

- Комбінація 1: F_{1a} – F_{2a} – F_{3a};
- Комбінація 1: F_{1b} – F_{2b} – F_{3b}.

Перша конфігурація орієнтована на швидкість, стабільність та простоту реалізації в умовах обмежених обчислювальних ресурсів. Вона базується на операторі Собеля, Unscented Kalman Filter, інтеграції Unity з ROS2 та прямому алгоритмі наведенні — і повністю реалізована в межах даної дипломної роботи.

Друга конфігурація передбачає використання сучасніших та потенційно точніших підходів — глибинної моделі виявлення, EKF, TCP/UDP-інтеграції та rule-based логіки. Проте ця комбінація вимагає значно більших витрат на реалізацію, навчання моделей та підтримку інфраструктури. У межах дипломної роботи було реалізовано низку функціональних компонентів, які забезпечують взаємодію між модулями

комп'ютерного зору, прогнозування траєкторії та симуляції у середовищі Unity.

4.3 Вибір параметрів для програмного продукту

Для об'єктивної оцінки ефективності реалізації системи перехоплення БПЛА було сформовано систему технічних параметрів, які дозволяють порівняти альтернативні комбінації реалізації за продуктивністю, обчислювальними витратами та складністю підтримки.

Оцінювання здійснюється за чотирма ключовими параметрами, які мають найбільший вплив на загальну якість, швидкодію та масштабованість програмного продукту.

Тобто X_1 відповідає за швидкість виконання, тобто кількість оброблених кадрів на секунду або затримка між виявленням і реакцією. Це критичний показник для роботи в реальному часі. Якщо затримка між обробкою кадру і реакцією дрона перевищує 1 секунду — це критично знижує ефективність. X_2 відповідає за обсяг оперативної пам'яті, тобто максимальний обсяг RAM, необхідний для стабільної роботи системи. Цей параметр важливий для підтримки стабільної роботи системи на обмежених апаратних платформах або в умовах масштабування. X_3 відповідає за час попередньої обробки, тобто час, який необхідний для підготовки системи до запуску або між модулями. Він враховує затрати часу на ініціалізацію, підключення ROS2, запуск Unity-сцени, що впливає на час реакції системи на зміну завдань. X_4 відповідає за обсяг програмного коду, тобто кількість рядків коду, що відповідають за реалізацію всіх функцій. Він опосередковано відображає складність підтримки, супроводу, ризик помилок, масштабованість. Тож, для кожного з параметрів було встановлено три рівні оцінки — «гірше», «середнє» та «краще», що подано в таблиці 4.1.

Таблиця 4.1 – Граничні значення параметрів

<i>Параметр</i>	<i>Позначення</i>	<i>Гірше (1)</i>	<i>Середнє (2)</i>	<i>Краще (3)</i>
Швидкість виконання	X1	< 10 FPS або > 1 с затримки	10–20 FPS або ~0.5 с	> 30 FPS або < 0.2 с
Обсяг пам'яті	X2	> 4 ГБ	2–4 ГБ	< 2 ГБ
Час попередньої обробки	X3	> 20 с	2–20 с	< 2 с
Обсяг коду	X4	> 5000 рядків	1000–5000 рядків	< 1000 рядків

У цьому розділі було обґрунтовано вибір технічних параметрів, які найбільшою мірою впливають на ефективність реалізації програмного продукту для системи перехоплення БПЛА. Обрані чотири показники — швидкість виконання, обсяг оперативної пам'яті, час попередньої обробки та обсяг програмного коду — дозволяють комплексно оцінити продуктивність, масштабованість та зручність супроводу системи.

Запропонована система оцінювання за трирівневою шкалою («гірше», «середнє», «краще») забезпечує формалізований підхід до порівняння альтернативних варіантів реалізації. Це дає змогу не лише визначити оптимальне технічне рішення з точки зору продуктивності, але й врахувати обмеження апаратного забезпечення та складність майбутньої підтримки програмного продукту.

Таким чином, розроблена система критеріїв слугує основою для подальшої багатофакторної техніко-економічної оцінки, що дозволяє здійснити обґрунтований вибір найефективнішого варіанту реалізації системи.

4.4 Експертний аналіз параметрів

У цьому підрозділі проводиться експертна оцінка технічних параметрів, визначених у попередньому підрозділі з метою встановлення їхньої вагомості для загальної ефективності програмного продукту. Аналіз базується на методі попарного порівняння, який дозволяє визначити ступінь важливості кожного параметра порівняно з іншими.

Експертна комісія у складі 5 осіб (розробники, дослідники та аналітики) провела незалежне ранжування чотирьох параметрів, де кожен експерт проставив ранги від 1 (найважливіший параметр) до 4 (найменш важливий). Також, обчислимо загальну суму рангів, використовуючи формулу (4.4.1), яка дозволяє знайти суму рангів для всіх параметрів разом.

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2}. \quad (4.4.1)$$

Обчислимо середню суму рангів за формулою (4.4.2):

$$T = \frac{1}{n} R_{ij}. \quad (4.4.2)$$

Обчислимо відхилення суми рангів кожного параметра від середньої суми рангів за формулою (4.4.3):

$$\Delta_i = R_i - T. \quad (4.4.3)$$

Обчислимо загальну суму квадратів відхилень за формулою (4.4.4):

$$S = \sum_{i=1}^N \Delta_i^2. \quad (4.4.4)$$

Результати ранжування подано в таблиці 4.2.

Таблиця 4.2 – Ранжування параметрів за експертною оцінкою

Позначення	Ранг експертів (1–7)							Сума рангів, R_i	Відхилення, Δ_i	Δ_i^2
X1	1	1	2	1	2	1	1	9	-8.5	72.25
X2	2	2	1	2	1	2	3	13	-4.5	20.25
X3	3	3	4	4	3	4	4	25	7.5	56.25
X4	4	4	3	3	4	3	2	23	5.5	30.25

Порахуємо коефіцієнт узгодженості за формулою (4.4.5):

$$W = \frac{12S}{N^2(n^3-n)}. \quad (4.4.5)$$

Знайдений коефіцієнт узгодженості 0.73 перевищує нормативне значення 0.67, тому ранжування можна вважати достовірним. З використанням результатів ранжування проведемо попарне порівняння всіх параметрів, а результати занесемо у таблицю 4.3.

Таблиця 4.3 – Попарне порівняння параметрів

Позначення	Ранг експертів (1–7)							Оцінка	Значення
X1 i X2	<	<	<	<	<	<	<	<	0.5
X1 i X3	<	<	<	<	<	<	<	<	0.5
X1 i X4	<	<	<	<	<	<	<	<	0.5
X2 i X3	<	<	<	<	<	<	<	<	0.5
X2 i X4	<	<	<	<	<	<	<	<	0.5
X3 i X4	>	>	>	>	>	>	>	>	1.5

Для визначення ступеня переваги одного параметра значення a_{ij} визначається згідно з формулою (4.4.6):

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.4.6)$$

З отриманих числових оцінок переваги складається матриця. Для кожного параметра розраховується вагомість згідно з наступною формулою: (4.4.7)

$$K_{\text{Ві}} = \frac{b_i}{\sum_{i=1}^n b_i}, \quad b_i = \sum_{i=1}^N a_{ij} \quad (4.4.7)$$

Зауважимо, що відносні оцінки розраховуються декілька разів, поки наступні значення не відрізняються від попередніх менше ніж на 2. Тож, результати обчислення вагомостей параметрів за методом попарного порівняння зведено в таблицю 4.4, де наведено значення коефіцієнтів для кожного параметра на трьох ітераціях, що підтверджують стабільність розрахунків і дозволяють перейти до подальшої багатокритеріальної оцінки варіантів реалізації.

Таблиця 4.4 – Розрахунок вагомості параметрів

Параметри, X_i	Параметри, X_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{Ві}}$	b_i^1	$K_{\text{Ві}}^1$	b_i^2	$K_{\text{Ві}}^2$
X1	1	0.5	0.5	0.5	2.5	0.156	9.25	0.156	34.125	0.157
X2	1.5	1	0.5	0.5	3.5	0.218	12.25	0.207	44.875	0.207
X3	1.5	1.5	1	1.5	5.5	0.343	21.25	0.36	77.875	0.36
X4	1.5	1.5	0.5	1	4.5	0.281	16.75	0.275	59.125	0.273
Всього:					16	1	98	1	216	1

З таблиці 4.4 бачимо, що різниця між значеннями коефіцієнтів вагомості не перевищує 2%, тому додаткові ітерації не потрібні.

4.5 Аналіз якості реалізації варіантів функцій

Для оцінки ефективності виконання основних функцій програмного продукту розглядаються абсолютні значення технічних параметрів, отриманих у результаті реалізації кожного з варіантів функцій F1–F4. Ці значення дозволяють проаналізувати вплив кожного з варіантів на загальну якість, продуктивність та зручність підтримки системи.

З метою кількісного аналізу використовується коефіцієнт технічного рівня, який обчислюється за формулою (4.5.1):

$$K_K(j) = \sum_{i=1}^n K_{b_{i,j}} B_{i,j} . \quad (4.5.1)$$

Кожен показник якості визначається за формулою (4.5.2), яка множить вагу параметра на загальну суму балів параметра.

$$K_K = K[F_{1k}] + K[F_{2k}] + \dots + K[F_{zk}] . \quad (4.5.2)$$

Результати цих розрахунків містить таблиця 4.5, яка надає оцінку якості кількох можливостей реалізації ключових можливостей програмного продукту.

Таблиця 4.5 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

<i>Основні функції</i>	<i>Варіант</i>	<i>Параметри, що беруть участь у реалізації функції</i>	<i>Абсолютне значення</i>	$V_{i,j}$	$K_{b_{i,j}}$	$K[F_{zk}]$
F1(X2)	A	X2	300 МБ	10	0.207	2.7
	B	X2	2500 МБ	3	0.207	0.621
F2(X3)	A	X3	0.05 с	8	0.36	2.88
	B	X3	0.09 с	5	0.36	1.8
F3(X3, X4)	A	X3	0.04 с	8	0.36	2.88
		X4	9000 рядків	5	0.273	1.365
	B	X3	0.01 с	9	0.36	3.24
		X4	4500 рядків	8	0.273	2.184
F(X4)	A	X4	2500 рядків	9	0.273	2.457
	B	X4	1800 рядків	10	0.273	2.73

На основі розрахунків технічного рівня видно, що перший варіант реалізації (А) має вищу сумарну оцінку якості (12,275 проти 10,575), що свідчить про його загальну перевагу з точки зору сукупності обраних технічних параметрів. Це означає, що реалізація А забезпечує кращий баланс між точністю, швидкістю, обсягом коду та споживанням ресурсів, навіть попри те, що в окремих функціях альтернативний варіант В може мати локальні переваги.

Таким чином, варіант А доцільно вважати кращим кандидатом для впровадження в програмному продукті, особливо враховуючи його реалізацію в межах дипломної роботи та підтримку основних вимог до системи.

4.6 Економічний аналіз розробки

Для оцінки трудомісткості реалізації системи перехоплення БПЛА було визначено чотири ключові завдання, що охоплюють етапи створення

функціональної системи в симульованому середовищі Unity + ROS2. Кожне завдання оцінено за критеріями: ступінь новизни, складність алгоритмів, тип вхідної інформації, мова програмування, використання стандартних модулів та математичного забезпечення. Розрахунок базується на загальній формулі (4.6.1):

$$T_0 = T_P * K_{\Pi} * K_{СК} * K_M * K_{СТ} * K_{СТ.М}. \quad (4.6.1)$$

У першому варіанту для реалізації модуля виявлення об'єкта за допомогою оператора Собеля та математичних моментів було обрано базову трудомісткість у 80 людино-днів. Оскільки застосовуються поширені алгоритми, новизна рішення оцінена коефіцієнтом 1.3. Складність вхідної інформації вважається типовою, тому відповідний коефіцієнт дорівнює 1. Рівень мови програмування (Python) не вносить додаткової складності, тому коефіцієнт також становить 1. Водночас передбачається часткове використання стандартних бібліотек комп'ютерного зору (наприклад, OpenCV), тож коефіцієнт використання готових модулів дорівнює 0.6. Крім того, математичне забезпечення, яке використовується, є достатньо стандартним, що відображено у значенні коефіцієнта 0.7. У результаті розрахунку отримано загальну трудомісткість реалізації даного модуля — 43.68 людино-днів.

У другому варіанту, коли використовується глибинна нейронна модель (наприклад, YOLO або SSD) для виявлення об'єкта. На відміну від класичних операторів градієнтів, такі моделі здатні знаходити об'єкти навіть у складних умовах, при частковому перекритті, зміні освітлення або присутності шумів. Базова трудомісткість цього завдання зростає до 120 людино-днів, оскільки реалізація включає навчання, підбір гіперпараметрів, а також підготовку датасету. Коефіцієнт новизни оцінюється як 1.5 — модель складна та сучасна. Вхідна інформація (зображення) потребує попередньої обробки, але залишається стандартною — коефіцієнт 1. Мова програмування — Python з

PyTorch, коефіцієнт 1. Оскільки використовуються бібліотеки типу PyTorch, коефіцієнт використання готових модулів — 0.5, а математичне забезпечення є високорівневим, тож коефіцієнт — 0.8. Загальна трудомісткість — 72 людино-днів.

У першому варіанту розробка модуля прогнозування траєкторії з використанням Unscented Kalman Filter (UKF) потребувала більшої базової трудомісткості, яка становить 100 людино-днів. Через використання методу, здатного працювати з нелінійними динамічними системами, що характеризуються шумами, коефіцієнт новизни оцінено як 1.4. Як і у попередньому випадку, вхідна інформація є типовою, що обґрунтовує значення коефіцієнта 1. Мова реалізації (Python) також не ускладнює процес, отже, коефіцієнт дорівнює 1. Ступінь використання готових модулів помірний (коефіцієнт 0.6), а математичне забезпечення більш складне, ніж у F1, тому коефіцієнт дорівнює 0.9. Сукупно це дає загальну трудомісткість реалізації фільтра UKF — 75.6 людино-днів.

другому варіанті застосовується розширений фільтр Калмана (ЕКФ), що є спрощеною версією UKF. ЕКФ добре працює у лінійно-нелінійних задачах, але менш точний за UKF, особливо у випадках з високим шумом або сильними нелінійностями. Базова трудомісткість — 80 людино-днів. Новизна оцінюється коефіцієнтом 1.2 — метод добре вивчений, з великою кількістю прикладів. Складність вхідної інформації — типова (коефіцієнт 1), як і мова реалізації (1). Для ЕКФ часто використовують готові бібліотеки, тому коефіцієнт використання стандартних модулів — 0.7, математичне забезпечення — 0.9. Отримуємо загальну трудомісткість — 60.48 людино-днів.

У першому варіанту завдання з інтеграції програмного продукту з візуалізаційним середовищем Unity та ROS2 передбачає базову трудомісткість у 90 людино-днів. Новизна проекту оцінена коефіцієнтом 1.2, оскільки поєднання Unity з ROS2 через RosSharp вимагає налаштування з урахуванням форматів повідомлень та обміну даними в реальному часі.

Інформація, що обробляється (позиції, зображення, повідомлення ROS), є типовою, як і мова програмування. Значення відповідних коефіцієнтів дорівнюють 1. Активне використання готових пакетів Unity та RosSharp знижує трудомісткість — коефіцієнт становить 0.5. Математичне забезпечення візуалізації оцінено коефіцієнтом 0.9. Сумарна трудомісткість інтеграції з Unity та ROS2 становить 48.6 людино-днів.

У другому варіанті замість повноцінної інтеграції з ROS2 та Unity застосовується пряме з'єднання на базі TCP/UDP протоколів. Це рішення не передбачає складної структури повідомлень або масштабованості, але дозволяє зменшити затримки та спростити реалізацію. Базова трудомісткість — 70 людино-днів. Новизна рішення — 1.1, бо з'єднання за TCP/UDP реалізується досить часто. Вхідна інформація — типова (коефіцієнт 1), мова — Python (1), використання готових бібліотек сокетів — 0.8, математичне забезпечення — мінімальне (0.6). Загальна трудомісткість — 37.03 людино-днів.

У першому варіанті розробка алгоритму прямого наведення дрона на ціль передбачала базову трудомісткість у 60 людино-днів. Оскільки реалізація є адаптацією відомих підходів навігації в реальному часі, новизна завдання оцінена коефіцієнтом 1.1. Інформація, яка обробляється (координати, орієнтація), є типовою, як і середовище реалізації. Таким чином, коефіцієнти складності вхідної інформації та рівня мови залишаються на рівні 1. Водночас часткове використання готових геометричних функцій Python дає коефіцієнт 0.7 для стандартних модулів. Враховуючи використання стандартної математики (векторні обчислення), коефіцієнт становить 0.9. Отже, повна трудомісткість реалізації модуля наведення дорівнює 41.58 людино-днів.

У другому варіанті, коли використовується набір умовних правил для управління рухом дрона. Такий підхід простий, швидкий у реалізації, але менш гнучкий до змін у траєкторії цілі. Базова трудомісткість — 50 людино-днів. Новизна — 1.0, складність вхідної інформації — типова (1), мова —

Python (1), використання готових умовних конструкцій — 0.8, математичне забезпечення — 0.6. Отже, повна трудомісткість — 24 людино-дні.

Отже, для першого варіанту загальна трудомісткість становить 209.46 людино-днів, що відповідає 1675.68 годин. Натомість для другого загальна трудомісткість — 193.44 людино-днів, або 1547.52 годин.

Розрахуємо погодинну ставку оплати праці, для цього використаємо формулу 4.6.2:

$$C = \frac{M}{T_m * t} \text{ грн.} \quad (4.6.2)$$

У розробці програмного продукту задіяно два програмісти з місячним окладом по 70 000 грн кожен та один спеціаліст з обробки даних із місячним окладом 40 000 грн. Тож, погодинна ставка для розрахунку трудової частини витрат у межах цього проєкту становить 1 071.43 грн/год.

Тоді, розрахуємо заробітну плату за формулою 4.6.3:

$$C = C * T_i * K. \quad (4.6.3)$$

Для варіанту А сумарна трудомісткість виконання усіх завдань становить 209.46 людино-днів, що еквівалентно 1675.68 годин (з урахуванням 8-годинного робочого дня). Погодинна ставка, розрахована на основі місячного фонду оплати праці трьох спеціалістів, становить 1071.43 грн/год. Також враховується норматив, що враховує додаткову заробітну плату, який дорівнює 1.2. Тоді отримаємо 2154448,59 грн.

Для варіанту В загальна трудомісткість становить 193.44 людино-днів, що відповідає 1547.52 годин. Знову ж таки, використовуючи погодинну ставку 1071.43 грн/год та той самий норматив 1.2, отримаємо 1989671,22 грн.

Згідно з чинним законодавством, нарахування на фонд оплати праці включають єдиний соціальний внесок (ЄСВ), який становить 22% від суми

заробітної плати. Відповідно, після розрахунку основної заробітної плати для кожного варіанту, необхідно додатково врахувати цей внесок. Тож, для варіанту А, де заробітна плата складає 2 154 448,59 гривень, розмір ЄСВ дорівнює 473 978,69 гривень. У випадку з варіантом В, при основній заробітній платі 1 989 671,22 гривень, внесок ЄСВ становить 437 727,67 гривень.

Таким чином, повна сума витрат на оплату праці з урахуванням ЄСВ у варіанті А становить 2 628 427,28 гривень, тоді як у варіанті В — 2 427 398,89 гривень. Це свідчить про те, що варіант А вимагає більших фінансових витрат на оплату праці — різниця становить 201 028,39 гривень. Такий приріст витрат пояснюється вищою загальною трудомісткістю реалізації, що, однак, може бути виправдано вищою якістю, точністю та надійністю обраних технічних рішень.

Тепер розглянемо розрахунок витрат на оплату однієї машино-години, яка використовується під час розробки проєкту системи перехоплення БПЛА. Припустимо, що одна обчислювальна машина обслуговує одного програміста з місячним окладом 70 000 грн. З урахуванням коефіцієнта зайнятості, який дорівнює 0,9, річний фонд заробітної плати для однієї машини розраховується як:

$$C_{\Gamma} = 12 * K = 12 * 70000 * 0,9 = 756000 \text{ грн.}$$

Навіть якщо в деяких варіантах не виконується навчання глибинних моделей, навантаження на обчислювальні ресурси залишається високим через активне використання візуального середовища Unity, яке вимагає значної графічної обробки в реальному часі. Додатково, реалізовано обробку відеопотоку, модулі візуалізації, передача повідомлень через ROS2, а також прогнозування траєкторії, що разом створює високу інтенсивність використання центрального та графічного процесорів. Усе це обґрунтовує

використання коефіцієнта зайнятості 0.9, навіть без безпосереднього навчання нейронних мереж.

З урахуванням додаткової заробітної плати, отримуємо загальну заробітну плату:

$$C_{ЗП} = C_{Г} * (1 + K) = 756000 * (1 + 0.2) = 907\ 200 \text{ грн}$$

Далі розраховується відрахування на соціальний внесок (СВІД), яке складає 22%:

$$C_{ВІД} = C_{ЗП} * 0,22 = 199\ 584 \text{ грн}$$

Для розрахунку амортизаційних відрахувань використовується коефіцієнт, який враховує витрати на транспортування та монтаж обладнання безпосередньо у користувача. Цей коефіцієнт приймається рівним 1.4, оскільки передбачається доставка та налаштування системи на місці експлуатації. Річна норма амортизації відображає частку вартості приладу, яка списується щороку внаслідок зносу та старіння. Для сучасної обчислювальної техніки ця норма зазвичай становить 25%. Крім того, параметр, що використовується для розрахунку амортизаційних витрат, також включає договірну вартість приладу, тобто ту суму, за якою комп'ютерне обладнання було придбане або оцінене для обліку. У цьому випадку вартість становить 150 000 гривень. Саме ця сума є основою для подальших обчислень — вона визначає загальний обсяг коштів, які щороку підлягають списанню у вигляді амортизації. Отже, амортизаційні відрахування розраховуємо:

$$C_{А} = K_{ТМ} * K_{А} * ПР = 1.4 * 0.25 * 150000 = 52\ 500 \text{ грн}$$

Витрати на ремонт та профілактику з одним важливим параметром, яким є відсоток витрат на поточні ремонти, який відображає частку від вартості обладнання, що зазвичай витрачається на технічне обслуговування, розраховуємо як:

$$C_P = K_{TM} * PR * K_P = 1.4 * 150000 * 0.08 = 16800 \text{ грн}$$

Ефективний годинний фонд часу персонального комп'ютера за рік показує, скільки годин техніка реально може використовуватись для виконання робочих завдань. Цей показник враховує не лише календарну кількість днів у році, але й усі дні, коли обладнання не працює через вихідні, святкові дні або планові ремонти. Базовим орієнтиром є загальна кількість календарних днів у році — 365. Із цієї кількості віднімаються 104 вихідних дні (дві суботи-неділі на тиждень), 12 офіційних святкових днів, а також 16 днів, передбачених для планового технічного обслуговування і ремонту обладнання. Таким чином, розраховується кількість робочих днів на рік, коли ПК реально може використовуватись. Кожен робочий день включає 8 годин роботи. Але не весь цей час обладнання використовується з максимальною ефективністю. Тому враховується коефіцієнт використання пристрою протягом зміни, який у нашому випадку становить 0.9. Він відображає реальний відсоток часу, коли ПК активно задіяний у виконанні завдань, з урахуванням перерв, очікування даних або простою. Тож, ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{EF} = (k - v - c - p) * t * K_B = 1677.6 \text{ грн}$$

Витрати на електроенергію, що споживається комп'ютерною технікою протягом року, розраховуються з урахуванням реального часу роботи обладнання та його потужності. В основі розрахунку лежить ефективний годинний фонд — 1677.6 годин, що визначає загальну кількість годин,

протягом яких комп'ютер працює в активному режимі за рік. Середньо-споживча потужність комп'ютера встановлена на рівні 0.6 кВт — це усереднене значення, яке враховує як пікові навантаження (наприклад, при обробці відео чи тренуванні моделей), так і фонове споживання. Додатково враховується коефіцієнт зайнятості приладу — 0.9, який показує, що не весь ефективний час ПК споживає енергію на повну потужність. Це може бути пов'язано з режимами очікування, перервами в обробці або неповним завантаженням ресурсів. Тариф на електроенергію береться відповідно до чинних комерційних розцінок і становить 9.23 грн за 1 кВт-год. Тож витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} * c * K_3 * \text{ЕН} = 8361.49 \text{ грн}$$

Для повної оцінки річних експлуатаційних витрат також необхідно врахувати накладні витрати, $C_{\text{Н}}$. Ці витрати охоплюють адміністративне управління, обслуговування приміщення, комунальні послуги, логістичну підтримку, а також інші непрямі витрати, що супроводжують експлуатацію обчислювальної техніки. У типовому розрахунку приймається, що накладні витрати становлять 67% від балансової вартості приладу. Оскільки ціна обладнання становить 150 000 грн, то накладні витрати дорівнюють 100 500 грн.

Після врахування всіх складових — заробітної плати (907 200 грн), єдиного соціального внеску (199 584 грн), амортизаційних відрахувань (52 500 грн), витрат на ремонт і профілактику (16 800 грн), електроенергії (8361.49 грн) та накладних витрат (100 500 грн) — можна підсумувати загальні річні експлуатаційні витрати, $C_{\text{ЕКС}}$. У результаті маємо 1 284 945.49 грн.

Отже, собівартість однієї машино-години ЕОМ обчислюється за формулою (4.6.4):

$$C_{M-Г} = \frac{C_{EKС}}{T_{EF}} \quad (4.6.4)$$

Отже, це становить 765.94 грн/год. Оскільки всі етапи створення програмного продукту виконуються на ЕОМ, важливо враховувати витрати на машинний час, необхідний для реалізації кожного варіанту проекту. Ці витрати визначаються шляхом множення собівартості однієї машино-години на загальну трудомісткість у годинах, яка була визначена раніше для кожного варіанту. Тобто за формулою (4.6.5):

$$C_M = C_{M-Г} * T \quad (4.6.5)$$

Таким чином, для варіанта А, при трудомісткості 1675.68 годин, витрати на машинний час становлять 1 283 331.40 грн. А для варіанта В, при трудомісткості 1547.52 годин, витрати становлять 1 185 610.26 грн.

Для визначення накладних витрат у межах розробки програмного забезпечення використовується частка від загальної заробітної плати. Прийнято, що накладні витрати становлять 67% від суми оплати праці, що є типовим показником для ІТ-проектів із урахуванням адміністративних витрат, комунальних послуг, оренди приміщень, офісного забезпечення тощо. У цьому випадку загальна заробітна плата становить 907 200 гривень. Відповідно, накладні витрати визначаються як 67% від цієї суми. Таким чином, розмір накладних витрат становить 607 824 гривень.

Остаточна вартість розробки програмного продукту визначається шляхом підсумовування всіх ключових складових витрат. До них належать основна заробітна плата з урахуванням надбавок, відрахування на соціальні потреби (зокрема, єдиний соціальний внесок), витрати на машинний час, а також накладні витрати, пов'язані з організаційним та технічним забезпеченням проекту. У варіанті А загальна сума витрат формується з

наступних величин: 907 200 гривень — основна заробітна плата, 199 584 гривень — відрахування на соціальні потреби, 1 283 331.40 гривень — витрати на машинний час, і 607 824 гривень — накладні витрати. У сумі це становить 2 997 939.40 гривень. У варіанті В: 907 200 гривень — заробітна плата, 199 584 гривень — соціальні відрахування, 1 185 610.26 гривень — машинний час, 607 824 гривень — накладні витрати. Разом — 2 900 218.26 гривень.

У результаті проведених розрахунків встановлено, що загальна вартість розробки програмного забезпечення залежить не лише від обсягу трудових ресурсів, а й від експлуатаційних витрат на використання комп'ютерної техніки та супутніх накладних витрат. Для варіанта А, який передбачає використання глибинного фільтра UKF, інтеграцію з Unity+ROS2 та пряме наведення дрона, загальна вартість розробки склала 2 997 939.40 грн. Для варіанта В, де реалізовано спрощену архітектуру на базі EKF з підключенням через TCP/UDP та простішою логікою керування, витрати становлять 2 900 218.26 грн. Різниця між варіантами становить 97 721.14 грн, що зумовлено вищою загальною трудомісткістю у варіанті А та відповідно — більшими витратами на машинний час. Водночас варіант А передбачає застосування складніших та більш точних алгоритмів, що можуть забезпечити вищу точність прогнозування траєкторії та стабільність під час перехоплення БПЛА. Таким чином, попри вищу вартість реалізації, варіант А може бути доцільнішим для критичних застосувань, де якість та надійність системи мають пріоритет над зменшенням витрат.

4.7 Вибір оптимального варіанту реалізації ПП

У цьому розділі виконується узагальнене порівняння двох варіантів реалізації програмного продукту з урахуванням як технічної якості, так і

витрат на його розробку. Для цього використовується коефіцієнт техніко-економічної рентабельності (КТЕР), який дозволяє оцінити співвідношення між отриманими перевагами (у вигляді технічних показників) і понесеними витратами. Такий підхід забезпечує комплексну оцінку ефективності варіантів реалізації. Розрахунок виконується за формулою:

$$K_{\text{ТЕР}j} = \frac{K_{kj}}{C_{\phi j}} \quad (4.7.1)$$

Для варіанта А інтегральна технічна оцінка становить 12,275, а повна вартість розробки — 2 997 939,40 грн. Тому коефіцієнт техніко-економічної рентабельності становить 0,000004096. Для варіанта В відповідні значення становлять 10,575 та 2 900 218,26 грн, що дає 0,000003646.

Отримані результати свідчать про те, що варіант А має вищий коефіцієнт техніко-економічної рентабельності, незважаючи на більші витрати на реалізацію. Це означає, що варіант А забезпечує краще співвідношення між якістю системи та витратами на її створення. Тому з точки зору комплексної ефективності саме варіант А є доцільнішим для впровадження, особливо у випадках, коли пріоритетом є точність, стабільність та надійність програмного продукту.

Висновки до розділу 4

У цьому розділі проведено узагальнене порівняння двох варіантів реалізації програмного продукту з точки зору їхньої техніко-економічної ефективності. Для комплексної оцінки було використано коефіцієнт техніко-економічної рентабельності (КТЕР), що дозволяє врахувати як технічну перевагу, так і повну вартість реалізації кожного з варіантів.

Розрахунки показали, що варіант А має інтегральну технічну оцінку 12,275 та повну вартість розробки 2 997 939,40 грн, що дає КТЕР ≈ 0.00409 . Натомість варіант В має нижчу технічну оцінку — 10,575, але й менші витрати — 2 900 218,26 грн, що призводить до КТЕР ≈ 0.00365 .

Таким чином, незважаючи на дещо більші витрати, варіант А демонструє вищу техніко-економічну рентабельність. Це свідчить про доцільність вибору саме цього варіанта, адже він забезпечує кращу якість реалізації за прийняттого рівня витрат.

ВИСНОВКИ

В межах дипломної роботи на ізольованій VM Ubuntu 22.04 LTS встановлено та налаштовано ROS 2 Humble, організовано взаємодію з Windows-оточенням через SSH-інтерпретер PyCharm, а також налаштовано автоматичний деплой коду й синхронізацію результатів у спільній папці vboxsf. У Unity HDRP побудовано сцену та запущено двосторонній обмін повідомленнями за допомогою RosSharp.

Реалізовано детекцію цілі на відеопотоці за допомогою оператора Собеля для виділення контурів та знаходження координат об'єкта у кадрі. З метою підвищення стійкості до шумів координати, отримані з комп'ютерного зору, підлягали фільтрації та прогнозуванню за допомогою Unscented Kalman Filter (UKF), що дозволило згладити траєкторію та оцінити майбутнє положення цілі.

У процесі моделювання перехоплення цілі в середовищі Unity було реалізовано систему, в якій керування перехоплювачем здійснювалося виключно на основі даних, отриманих з модуля комп'ютерного зору. Алгоритм обробки зображень дозволяв в реальному часі визначати координати цілі на кадрах відеопотоку та передавати цю інформацію системі керування. Таким чином, траєкторія руху перехоплювача формувалася відповідно до актуальних координат цілі, які надавав комп'ютерний зір.

Крім того, використання патернів програмування «Стратегія», «Спостерігач» та «Посередник» підвищило гнучкість, масштабованість і підтримуваність програмної архітектури.

У ході дослідження було проведено порівняльний аналіз розширеної та спрощеної моделей руху цілі, а також побудовано графіки похибок прогнозу на різних кроках та результати моделювання траєкторії перехоплювача у середовищі Unity. Це дозволило комплексно оцінити точність прогнозування

та роботу системи наведення в умовах ідеалізованих і більш реалістичних моделей.

Наукова цінність роботи полягає у розробці та апробації комплексної програмної архітектури для моделювання задачі перехоплення БПЛА із залученням сучасних технологій комп'ютерного зору, фільтрації стану та гнучкого програмного керування. Робота створює підґрунтя для подальших досліджень у галузі інтелектуального керування безпілотними системами, зокрема для задач автономного переслідування, супроводу або перехоплення рухомих об'єктів у реальних умовах.

У подальших дослідженнях перспективним напрямом є реалізація стратегії перехоплення цілі з урахуванням не лише поточних, а й прогнозованих координат, які обчислює Unscented Kalman Filter (UKF). Це дозволить порівняти ефективність підходу, заснованого на прогнозі, з реалізованою у роботі стратегією прямого наведення на актуальні координати цілі. Подібне порівняння дозволить оцінити вплив затримок та шумів у даних комп'ютерного зору на якість та швидкість перехоплення, а також визначити, наскільки використання прогнозу покращує результат у складних сценаріях.

ПЕРЕЛІК ПОСИЛАНЬ

1. The Role of FPV Drones in the Military Market. 2024. URL: <https://droneii.com/military-fpv-drones> (дата звернення: 17.05.2025)
2. Exploring Radar Micro-Doppler Signatures for Recognition of Drone Types. 2023. URL: <https://www.mdpi.com/2504-446X/7/4/280> (дата звернення: 17.05.2025)
3. Drone Detection and Tracking Using RF Identification Signals. 2023. URL: <https://www.mdpi.com/1424-8220/23/17/7650> (дата звернення: 17.05.2025)
4. The Pros and Cons of Radio Frequency Analysers in Drone Detection. 15.11.2022. URL: <https://www.robinradar.com/blog/radio-frequency-analysers-drone-detection> (дата звернення: 17.05.2025)
5. The Pros and Cons of Using an Acoustic Detection System Against Drones. 29.11.2022. URL: <https://www.robinradar.com/blog/acoustic-sensors-drone-detection> (дата звернення: 17.05.2025)
6. Analysis of Distance and Environmental Impact on UAV Acoustic Detectio. 2024. URL: <https://www.mdpi.com/2079-9292/13/3/643> (дата звернення: 17.05.2025)
7. Prewitt operator. *Wikipedia*, остання редакція – 04.12.2024. URL: https://en.wikipedia.org/wiki/Prewitt_operator (дата звернення: 14.04.2025)
8. Sobel operator. *Wikipedia*, остання редакція – 05.03.2025. URL: https://en.wikipedia.org/wiki/Sobel_operator (дата звернення: 14.04.2025)
9. Canny edge detector, *Wikipedia*, остання редакція – 21.05.2025. URL: https://en.wikipedia.org/wiki/Canny_edge_detector (дата звернення: 14.04.2025)
10. Fisher R., Perkins S., Walker A. and Wolfart E.. Spatial Filters – Laplacian/Laplacian of Gaussian. URL:

- <https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm> (дата звернення: 16.04.2025)
11. Dalal N., Triggs B. Histograms of Oriented Gradients for Human Detection. 2005. URL: <https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf> (дата звернення: 16.04.2025)
 12. Contours : Getting Started. OpenCV-Python. Версія 3.4. 2025-05-22. URL: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html (дата звернення: 15.04.2025)
 13. Canny Edge Detection. OpenCV-Python Tutorials, версія 4.x, 2025-05-22. URL: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html (дата звернення: 14.04.2025)
 14. Ivins J., Porrill J.. Everything You Always Wanted to Know about Snakes (But Were Afraid to Ask). 2000. URL: <https://web.mat.upc.edu/toni.susin/files/SnakesAivru86с.pdf> (дата звернення: 15.04.2025)
 15. Caselles V., Kimmel R., Sapiro G. Geodesic Active Contours. *International Journal of Computer Vision*. 1997. URL: https://www.researchgate.net/publication/220660519_Geodesic_Active_Contours (дата звернення: 15.04.2025)
 16. Xiang Y. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. 2018. URL: <https://rse-lab.cs.washington.edu/projects/posecnn/> (дата звернення: 01.05.2025)
 17. Perspective-n-Point. *Wikipedia*, остання редакція – 22.04.2025. URL: <https://en.wikipedia.org/wiki/Perspective-n-Point> (дата звернення: 01.05.2025)
 18. Alex B. Kalman Filter from the Ground Up. 1-ше вид., 2023. 436 с.
 19. Installation Guide (ROS 2 Humble Hawksbill), офіційна документація ROS 2, 2025. URL: <https://docs.ros.org/en/humble/Installation.html> (дата звернення: 05.04.2025)

20. Edge detection using Prewitt, Scharr and Sobel Operator. 24.11.2022. URL: <https://www.geeksforgeeks.org/edge-detection-using-prewitt-scharr-and-sobel-operator/> (дата звернення: 10.04.2025)
21. Shahid Akhtar Khan. How to compute the area and perimeter of an image contour using OpenCV-Python. 2022-09-28. URL: <https://www.tutorialspoint.com/how-to-compute-the-area-and-perimeter-of-an-image-contour-using-opencv-python> (дата звернення: 20.04.2025)
22. Moments Class Reference, OpenCV. URL: https://docs.opencv.org/3.4/d8/d23/classcv_1_1Moments.html (дата звернення: 20.04.2025)
23. Image rotation angle calculation in OpenCV. 07.11.2014 (оновлено 20.04.2023). URL: <https://stackoverflow.com/questions/26793658/image-rotation-angle-calculation-in-opencv> (дата звернення: 20.04.2025)
24. Mingjun Deng, Shuhang Li, Xueqing Jiang, Xiang Li. Vehicle Trajectory Prediction Method Based on “Current” Statistical Model and Cubature Kalman Filter. 2023. URL: <https://www.mdpi.com/2079-9292/12/11/2464> (дата звернення: 20.04.2025)
25. Yifan Zhang, Ziyi Jia, Chao Dong, Yuntian Liu, Lei Zhang, Qihui Wu. Recurrent LSTM-based UAV Trajectory Prediction with ADS-B Information URL: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/2209.00436> (дата звернення: 20.04.2025)
26. Francesco Juliari, Irtiza Hasan, Marco Cristani, Fabio Galasso. Transformer Networks for Trajectory Forecasting URL: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/2003.08111> (дата звернення: 20.04.2025)
27. Xiaoru Cai, Chao Yang, Zhiming Guo, Zonghua Sun, Liaoni Wu, Fuqiang Bing, Guoqiang Su. Research and Flight Test on the Terminal Guidance Control Technology for Cruising Unmanned Aerial Vehicles. 2024. URL: <https://www.mdpi.com/2226-4310/11/12/975> (дата звернення: 20.04.2025)

28. Towards Safe Mid-Air Drone Interception: Strategies for Tracking & Capture. URL: <https://arxiv.org/html/2405.13542v1> (дата звернення: 20.04.2025)
29. Derek R. Nelson, D. Blake Barber, Timothy W. McLain, Randal W. Beard. Vector Field Path Following for Small Unmanned Air Vehicles. URL: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2543&context=facpub> (дата звернення: 20.04.2025)
30. Yuxie Luo, Jia Song, Kai Zhao, Yang Liu. UAV-Cooperative Penetration Dynamic-Tracking Interceptor Method Based on DDPG. 2022. URL: <https://www.mdpi.com/2076-3417/12/3/1618> (дата звернення: 20.04.2025)
31. Jasper Thomas Arneberg. Guidance Laws for Partially-Observable UAV Interception Based on Linear Covariance Analysis. 2018. URL: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://dspace.mit.edu/bitstream/handle/1721.1/119025/1057725527-MIT.pdf?isAllowed=y&sequence=1> (дата звернення: 20.04.2025)
32. Gregory Hollows, Nicholas James. Understanding Focal Length and Field of View. URL: https://www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/?srsltid=AfmBOoqJJ4dLucOwamwYVKhmEZrC8v1H9nlbstHxTMJ8eUHmyA0SS_cK (дата звернення: 18.05.2025)

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Повний лістинг коду розміщено за посиланням

URL:<https://github.com/PolnikovaPolina/Diploma>