Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеню вищої освіти – магістр
за освітньо-професійною програмою
«Кіберфізичні системи в промисловості, бізнесі та транспорті»

зі спеціальності
174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка

тема роботи:

«Інтелектуальна система автономної навігації дрону з використанням даних бортової камери та нейрокерування»

Виконав ст. гр. АКІТР-23-1м	_____	Підкоритов В.О.
Керівник	_____	Купін А.І.
Нормоконтроль	_____	Маринич І. А.
Завідувач кафедри	_____	Рубан С. А.

Кривий Ріг – 2024

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

« *5* » *липня* 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи АКІТР-23-1м. *Підкоритову Володимиру Олександровичу*

1. **Тема кваліфікаційної роботи:** *«Інтелектуальна система автономної навігації дрону з використанням даних бортової камери та нейрокерування»*

затверджено наказом по університету № *595с* від *04.07.2024 р.*

2. Термін здачі кваліфікаційної роботи: *01.12.2024 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 75с., додатки, презентація у Microsoft PowerPoint (15 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 проф. Купін А.І.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.07.24*
2	*Розділ 1*	*15.07.24*
3	*Розділ 2*	*18.08.24*
4	*Розділ 3*	*19.09.24*
5	*Висновки*	*15.10.24*
6	*Оформлення кваліфікаційної роботи*	*20.11.24*
7	*Підготовка презентації та графічного матеріалу*	*28.11.24*
8	*Підготовка доповіді до захисту*	*01.12.24*

6. Дата видачі завдання: *28.06.2024р.*

Керівник _____ /Купін А.І./

7. Запевнення: *Я, Підкоритов Володимир Олександрович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ / Підкоритов В.О./

АНОТАЦІЯ

Підкоритов В. О. «Інтелектуальна система автономної навігації дрону з використанням даних бортової камери та нейрокерування».

Кваліфікаційна робота на здобуття ступеню вищої освіти магістр за освітньо-професійною програмою «Кіберфізичні системи в промисловості, бізнесі та транспорті» зі спеціальності 174 – Автоматизація, комп’ютерно – інтегровані технології та робототехніка – Криворізький національний університет, Кривий Ріг, 2024.

Об’єктом дослідження є процес автономної навігації дрону, включаючи інтеграцію відеоаналітики та нейронних мереж для розпізнавання образу та забезпечення самостійного управління навігацією. Конкретно, дослідження зосереджено на:

− Реалізація ПІД - регулятора: вибір і тюнінг параметрів ПІД регулятора для автоматизації системи.

− Алгоритми комп'ютерного зору: Розробка та впровадження алгоритмів для розпізнавання об'єктів, виявлення перешкод і оцінки навколишніх умов.

− Нейронні мережі: Використання нейронних мереж для обробки отриманих даних та прийняття рішень щодо управління дроном, таких як оптимізація маршруту і корекція траєкторії.

У першому розділі представлена основна інформація по підходам і технологіям які будуть використовуватися для створення інтелектуальної системи керування дроном.

В другому розділі представлені математичні моделі та деталі реалізації системи

В третьому розділі зосереджені дані по реалізації готової системи і аналіз її ключових елементів.

АВТОМАТИЗАЦІЯ, ДРОН, НЕЙРОМЕРЕЖА, КОМП’ЮТЕРНИЙ ЗІР, ПІД-РЕГУЛЯТОР, ШТУЧНИЙ ІНТЕЛЕКТ

ANNOTATION

Pidkorytov V.O. «Intelligent autonomous drone navigation system using on-board camera data and neurocontrol».

Graduation master`s work for obtaining an educational degree «Master» for the educational and professional program « Cyber-physical systems in industry, business and transport » in specialty 174 – «Automation, computer-integrated technologies, and robotics». – Kryvyi Rih National University, Kryvyi Rih, 2024

The object of research is the process of autonomous drone navigation, including the integration of video analytics and neural networks to ensure autonomous flight control. Specifically, the research focuses on:

- *Implementation of the PID controller:* selection and adjustment of PID controller parameters for the system automation
- *Computer vision algorithms:* Development and implementation of algorithms for object recognition, obstacle detection and environmental assessment.
- *Neural Networks:* The use of neural networks to process received data and make drone control decisions such as route optimization and trajectory correction.

The first section presents basic information about the approaches and technologies that will be used to create an intelligent drone control system.

The second chapter presents mathematical models and details of system implementation

The third section concentrates data on testing the finished system and analysis of its key elements.

Keywords:

AUTOMATION, DRONE, NEURAL NETWORK, PID-CONTROLLER, ARTIFICIAL INTELLIGENCE

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Скорочення:

БПЛА - безпілотний літальний апарат

ПІД - Пропорційно-інтегрально-диференціальний

ПІДНМ - Пропорційно-інтегрально-диференціальний з нейронною мережею

РЕБ - радіо - електронна боротьба

ШНМ - штучні нейронні мережі

ШІ - штучний інтелект

AI - Artificial intelligence

BSD - Berkeley Software Distribution license

CNN - Convolutional Neural Networks

CPU - Central processing unit

DJI - Dajiang Innovation Technology Co

DRY - don’t repeat yourself

FPV - First Person View

GPS - Global Positioning System

GPU - graphics processing unit

KISS - keep it simple

ML - Machine Learning

SDK - software Development Kit

YAGNI - You Aren’t Gonna Need It

ВСТУП

В наші дні технології штучного інтелекту та робототехніки швидко розвиваються і стають все більш поширеними в різних сферах людської діяльності. Наприклад дрони набули великої популярності не тільки як розважальні аксесуари, але і як професійне обладнання, яке широко застосовується в логістиці, сільському господарстві, будівництві, патрулюванню місцевості, тощо. Також дрони все активніше використовують для вирішення військових задач як в обороні, так і в наступальних діях, що є надзвичайно актуальним в реаліях нашого сьогодення.

Однак, разом із зростанням використання дронів, з'явилися й суттєві виклики. Найбільш актуальним з них є проблема втрати зв'язку між дроном та оператором, особливо на фінальній фазі польоту. Це може значно ускладнити виконання місій та загрожувати успішності їх виконання. Одним із способів подолання цієї проблеми є використання системи інтелектуального керування на основі нейронних мереж (тобто нейрокерування). Цей підхід дозволяє дронам приймати рішення в реальному часі, реагуючи на зміни у навколишньому середовищі. Перевагою нейрокерування є можливість автоматичного захвату обраного об'єкта у "перехресті прицілу" та слідування за ним. Це рішення зменшує залежність від роботи оператора та забезпечує більш стабільну та точну навігацію дрону.

Іншою проблемою є необхідність налаштовувати параметри дрону, що може бути досить трудомістким процесом, особливо коли для різних умов потрібні окремі конфігурації. Завдяки штучному інтелекту можна надати системі здатність автоматично адаптуватися до змін навколишнього середовища під час польоту.

Таким чином, застосування систем інтелектуального управління та нейрокерування в дронах є важливим кроком у вирішенні актуальних проблем їх використання. Ці технології дозволяють значно покращити ефективність дронів у специфічних сферах застосування, забезпечуючи

більш точну та стабільну роботу в умовах, де важлива кожна секунда та кожний метр.

Метою даної кваліфікаційної роботи є створення інтелектуальної системи керування дроном з використанням оптимальних програмних та апаратних засобів необхідних для реалізації машинного зору для автономного виконання завдань.

Для досягнення мети дослідження поставлено і вирішено такі завдання:

- дослідження фізичної моделі дрона типу DJI Tello;
- реалізація сучасного методу розпізнавання образів за допомогою згорткової нейронной мережі;
- реалізація алгоритму та підбір параметрів ПІД - регулятора для автономного керування дроном;
- дослідження і реалізація підсилення ПІД - регулятора нейронною мережею (ПІДНМ)
- написання програмного коду для реалізації алгоритму автономної навігації дрону.
- тестування системи

Об’єкт дослідження – процес інтелектуального керування дроном з використанням бортової камери та нейронних мереж.

Предмет дослідження – сучасні алгоритми нейронних мереж для розпізнавання образів в реальному часі, тюнінг та реалізація ПІД регулятора для автономного керування дроном, порівняння роботи ПІД та ПІДНМ.

Практична цінність. Отримані результати можуть використовуватися для реалізації систем автоматичної навігації дронів які містять мікро комп'ютер і для яких є важливим швидкість прийняття рішень та висока ймовірність виконання завдання в автоматичному режимі в складних умовах. Подібна система може виключити оператора з процесу налаштування і управління дроном.

РОЗДІЛ 1
ВИБІР ПІДХОДІВ ДО РЕАЛІЗАЦІЇ СИСТЕМИ

1.1 Проблематика використання дронів

Українські оператори безпілотників щомісяця втрачають тисячі дронів через роботу російських пристроїв глушіння, йдеться у звіті британського Королівського Об'єднаного інституту оборонних досліджень [1]: ”Російські військові [...] продовжують широко використовувати навігаційні перешкоди в районі бою як різновид радіоелектронного захисту. Це сприяє тому, що рівень втрат українських БПЛА становить приблизно 10 000 одиниць місяць”. За допомогою пристроїв радіоелектронної боротьби (РЕБ) противник може порушити роботу систем навігації та керування дрон, перехопити керування, чи заглушити сигнал зв'язку з оператором дрону. Використання алгоритмів інтелектуального керування дрону за допомогою штучного інтелекту може допомогти підвищити вірогідність успішного виконання завдання, а саме розпізнавання цілі, слідування за ціллю, автономне враження цілі, тощо.

Одним з найбільш поширених методів покращення роботи дронів на сьогодні є підбір коефіцієнтів ПІД - регулятора на основі аналізу даних з попередніх польотів. У відкритих джерелах є багато прикладів таких налаштувань для дронів MAVIC, які широко використовуються в Збройних Силах України. Однак цей процес вимагає значних затрат часу на збір та аналіз даних, а також на налаштування БПЛА під конкретні завдання та різні типи впливів на систему. Розробка інтелектуальної системи, здатної автоматично адаптуватися до змінних умов, значно спростить і покращить цей процес.

1.2 Інтелектуальна система автономної навігації

Інтелектуальна система автономної навігації — це складна система, яка дозволяє об'єктам (наприклад, безпілотним літальним апаратам (дронам), роботам, автомобілям або іншим автономним транспортним засобам) виконувати навігацію без безпосереднього втручання людини. Вона включає в себе набір технологій і алгоритмів, що дозволяють цьому об'єкту ефективно орієнтуватися в просторі, взаємодіяти з навколишнім середовищем і ухвалювати рішення на основі аналізу даних. Ось основні компоненти та принципи роботи такої системи:

1.2.1 Основні компоненти інтелектуальної системи автономної навігації.

Сенсори та сприйняття середовища. Лідар, радари, камери та інші сенсори використовуються для збору інформації про навколишнє середовище (перешкоди, рухомі об'єкти, дорожні знаки тощо). Машинне зір за допомогою комп'ютерного зору (на основі камер) дозволяє виявляти об'єкти і розпізнавати навколишнє середовище, наприклад, шлях, перешкоди або цілі.

Алгоритми обробки та аналізу даних. Фільтри Калмана, інтерфейс сенсорів, методи машинного навчання та нейронні мережі допомагають зібрати, обробити та аналізувати дані, зібрані з сенсорів, щоб отримати точнішу картину оточення і визначити найкращий шлях для руху. Штучні нейронні мережі можуть використовуватися для навчання моделі навігації, зокрема для адаптивної оцінки і прогнозування змін в середовищі.

Планування та ухвалення рішень. Алгоритми планування траєкторії дозволяють вибирати оптимальний шлях на основі даних про перешкоди та навколишнє середовище, з урахуванням таких критеріїв, як безпека, час, енергоефективність і швидкість. Штучний інтелект може бути використаний для прийняття рішень на основі контексту, адаптуючи навігацію до змін в оточенні.

Керування та виконання дій. Після прийняття рішення про оптимальний шлях, система автономної навігації керує рухом об'єкта (дрону, автомобілю або роботу) через систему підсистем керування рухом. Ці підсистеми можуть включати пропорційно-інтегрально-диференціальний ПІД - регулятор, методи безперервного коригування траєкторії і адаптивні алгоритми.

Самонавчання та адаптація. Система здатна вчитися на основі досвіду (методи машинного навчання), удосконалюючи свої алгоритми в реальному часі. Наприклад, вона може коригувати свої стратегії навігації на основі змін у навколишньому середовищі (нові перешкоди, зміна маршруту тощо).

1.2.2 Технології, що використовуються в інтелектуальних системах автономної навігації.

- Машинне навчання та глибоке навчання: Для створення моделей, які можуть адаптуватися до нових ситуацій без необхідності переналаштовувати систему вручну.
- Штучні нейронні мережі: Використовуються для розпізнавання образів і прийняття рішень на основі візуальних і сенсорних даних.
- Фільтри Калмана: Для комбінування різних джерел інформації (наприклад, даних з камери і лідару) з урахуванням шуму і невизначеності.
- Обчислювальні платформи: Інтелектуальні системи часто базуються на обчислювальних платформах, таких як GPU або CPU, для забезпечення швидкої обробки даних в реальному часі.

1.2.3 Приклади застосування інтелектуальних систем навігації

- Безпілотні автомобілі: Індивідуальні автономні автомобілі, які використовують ці системи для навігації по дорогах, визначення об'єктів, що знаходяться на шляху (пішоходи, інші транспортні засоби), та ухвалення рішень щодо руху.

- Безпілотні літальні апарати (дрони): Автономні дрони, які можуть здійснювати польоти за заданим маршрутом, виконувати завдання з моніторингу або доставки, враховуючи перешкоди в небі або на землі.
- Роботи: Роботи для внутрішньої навігації в складних середовищах, наприклад, на складах або в лікарнях, для автоматичної доставки предметів або виконання завдань.
- Роботизовані кораблі та підводні апарати: Навігація в океанах або річках для збору даних або проведення досліджень.

1.2.4 Переваги інтелектуальних систем автономної навігації

- Безпека: Зменшення людських помилок і небезпек, пов'язаних з людським фактором.
- Ефективність: Оптимізація маршрутів, зменшення витрат часу та енергії.
- Адаптивність: Можливість реагувати на зміни в навколишньому середовищі або нові умови без втручання людини.
- Точність: Збільшення точності навігації і виконання завдань, таких як доставка або пошук цілей.

1.3 FPV - дрони

FPV дрон — це безпілотний літальний апарат (БПЛА) з камерою, яка бездротовим способом передає відео на окуляри, гарнітуру, мобільний пристрій або інший дисплей [2]. Користувач має змогу бачити від першої особи (FPV) навколишнє середовище, де літає дрон, і може знімати відео або фотографії.

FPV - дрони можуть керуватися дистанційно або можуть бути запрограмовані на автономний політ за допомогою планів польоту, керованих програмним забезпеченням, які отримують доступ до даних з бортових датчиків і GPS. З точки зору користувача, дрон FPV схожий на літаючого

робота, який забезпечує віртуальну присутність скрізь, де може літати пристрій, часто в середовищах, до яких людина не може фізично отримати безпечний доступ. На відміну від людей, персональні безпілотники можуть отримувати доступ до менших просторів і витримувати суворіші навколишні умови, крім того, що вони здатні літати.

Здатність безпілотних літальних апаратів FPV входити в середовище, небезпечне для людей, робить їх ефективними для пошуково-рятувальних місій. Дронами можна дистанційно керувати, щоб знаходити людей у небезпечних ситуаціях, уникаючи значної частини потреби у фізичній присутності. Подібним чином дрони можна використовувати для перевірки важкодоступної фізичної інфраструктури, наприклад мостів і високих будівель.

FPV-дрони також застосовуються на підприємстві, наприклад, для спостереження за системою безпеки та моніторингу співробітників, чи в агротехнологіях де дрон FPV дозволяє фермеру обстежити посіви та худобу набагато швидше, ніж це можливо за допомогою наземної інспекції, і набагато ближче, ніж це можливо з літака.

1.3.1 Вибір дрону

Найпопулярнішими доступними дронами на українському ринку є дрони від DJI та Mavic, або 7 чи 10 дюймові дрони, які можна самостійно зібрати. Основним параметром для вибору дрону для створення даної інтелектуальної системи є невеликі розміри, підтримка мови програмування чи можливість виконувати команди по бездротовим мережам і невисока вартість. В таблиці 1.1 представлене порівняння основних вимог до системи для доступних дронів.

Таблиця 1.1 – Порівняння доступних дронів

Назва	Ціна	Габарити	Функціонал
DJI Tello	Низька/Середня	маленькі	Підтримка Python
Mavic 2/3/Mini	Середня/Висока	будь-які	SDK для збору і обробки інформац
Дрон 7”/10”	Середня	7 або 10 дюймів	Потрібен додатков мікрокомп'ютер

DJI Tello має невеликі габарити, доступну ціну і підтримує мову програмування Python, що повністю задовольняє вимоги до створюваної системи.

1.3.2 Загальна інформація про дрон DJI Tello

DJI Tello EDU - програмований дрон, що ідеально підходить для навчання [3]. З ним легко вивчити такі мови програмування, як Scratch, Python і Swift. З оновленим SDK 2.0 Tello EDU поставляється з більш просунутими командами і розширеними інтерфейсами даних. Tello EDU, оснащений технологією управління польотом DJI, підтримує електронну стабілізацію зображення. Також є можливість за допомогою коду об'єднати кілька Tello EDU для того щоб створити рій дронів на основі функції штучного інтелекту.

Рисунок 1.1 - Дрон DJI Tello

1.3.3 Основні команди роботи з дроном

DJI Tello — це популярний дрон, який має простий інтерфейс для програмування через Python. Щоб керувати Tello з Python, можна використовувати бібліотеку djitellopy, яка надає зручні методи для роботи з дроном. Основні команди, які можна використовувати для керування дрону і які будуть використовуватися для реалізації інтелектуальної системи:

Імпорт бібліотеки:

```
from djitellopy import Tello
```

Створення об'єкта дрону і підключення до нього:

```
tello = Tello()
tello.connect()
```

Отримання інформації про батарею:

```
battery_level = tello.get_battery()
```

Запуск та зупинка двигунів:

```
tello.takeoff()  # Зліт
tello.land()     # Приземлення
```

Закриття з'єднання з дроном:

```
tello.end()
```

Ці команди охоплюють основні функції дрона, які можна

використовувати для базового керування і автоматизації польотів.

1.3.4 Переваги і недоліки дрону DJI Tello

До основних переваг можна віднести:

- відносно невелика ціна (150-200$ в залежності від комплектації)
- підтримка Python та інших мов програмування
- невеликі розміри, які дозволяють тестувати систему навіть в квартирі

До основних недоліків можна віднести:

- аккумулятор працює 12 хвилин до повного розрядження
- чим нижче заряд батареї, тим нижче обчислювальна потужність пристрою
- маленька вага, через що складно протистояти вітру

1.4 Машинний зір

У 2023 році інтеграція штучного інтелекту та комп’ютерного зору в технологію дронів стала каталізатором монументального зрушення в автономних можливостях таких пристроїв [4]. Цей прогрес базується на складних алгоритмах машинного навчання, які дозволяють дронам інтерпретувати середовище та реагувати на нього з безпрецедентним рівнем точності. Синергія між штучним інтелектом і платформою зору, відіграла ключову роль у розширенні функціональних можливостей дронів, вивівши їх далеко за рамки традиційних ролей.

Роль програмного забезпечення для комп’ютерного зору в дронах була визначною, дозволяючи виявляти в реальному часі та відображати об’єкти, що має вирішальне значення для різних програм. Зараз дрони обробляють величезну кількість даних зі своїх датчиків, що сприяє більш обґрунтованому та автономному прийняттю рішень. У пошуково-рятувальних операціях, наприклад, це вдосконалення дозволяє безпілотникам, таким як DJI Drone

Dock, самостійно орієнтуватися в складних умовах, швидко визначати місцезнаходження та надавати допомогу в надзвичайних ситуаціях.

Крім того, у 2023 році БПЛА використовували штучний інтелект для покращення аеронавігації, підвищуючи їх ефективність у таких завданнях, як картографування та моніторинг. Впровадження штучного інтелекту в безпілотних літальних апаратах також передбачає використання складних алгоритмів навчання, що додатково вдосконалює їх можливості виявлення та відстеження. Ця еволюція являє собою значний стрибок в технологіях передачі даних безпілотників, знаменуючи нову еру автономних та інтелектуальних рішень.

У світі технологій безпілотних літальних апаратів, що швидко розвивається, Python став ключовим гравцем у розробці програм на базі ШІ для БПЛА. Python, відомий своєю простотою та можливістю адаптації, є найкращою мовою програмування для створення алгоритмів навчання, які забезпечують ШІ в дронах. У 2023 році роль Python у розробці штучного інтелекту безпілотників є більш важливою, ніж будь-коли, що дозволяє створювати складні алгоритми, які дозволяють безпілотникам розумно інтерпретувати та взаємодіяти з навколишнім середовищем.

Великі бібліотеки та фреймворки Python, зокрема у сфері машинного навчання та комп’ютерного зору, як-от TensorFlow і OpenCV, оптимізували впровадження складних алгоритмів у дронах, щоб виконувати більше роботи. Ці інструменти сприяють ефективній обробці великих обсягів даних, зібраних датчиками дронів, роблячи можливим аналіз даних та прийняття рішень у реальному часі. Ця можливість є важливою в сценаріях, що вимагають негайного реагування, наприклад, у надзвичайних ситуаціях, коли своєчасні дії є вирішальними.

Крім того, сумісність Python з різними структурами ШІ та машинного навчання зробила його ідеальним вибором для розробки автономних навігаційних систем у дронах. Ці системи значною мірою покладаються на

алгоритми, які обробляють дані датчиків для виявлення перешкод і визначення оптимальних траєкторій польоту, забезпечуючи безпечну та ефективну роботу. Інтеграція Python у штучний інтелект безпілотників не лише покращує їхні навігаційні можливості, але й розширює їхні потенційні застосування в різних сферах, від аерофотозйомки до міського планування.

Підсумовуючи, можна сказати, що робота Python з ШІ та алгоритмами машинного навчання має фундаментальне значення для вдосконалення можливостей дронів.

1.5 Штучні нейронні мережі і нейрокерування

Дрони, якими можна керувати дистанційно або за допомогою внутрішніх комп'ютерних систем, відомі як безпілотні літальні апарати (БПЛА) [5]. Вони мають багато різних застосувань. Коли ці пристрої були створені вперше, їх дистанційне керування було ручним. Безпілотники зі штучним інтелектом, які автоматизують деякі або всі їхні завдання, стають все більш поширеними. Поєднуючи штучний інтелект (ШІ) з інформацією, отриманою від датчиків, камер і вбудованого обладнання дронів, дрони тепер можуть збирати та використовувати візуальні та атмосферні дані.

Одним з ефективних методів реалізації ШІ є використання штучних нейронних мереж. Нейронні мережі [6] — це моделі машинного навчання, які імітують складні функції людського мозку. Ці моделі складаються з взаємопов'язаних вузлів або нейронів, які обробляють дані, вивчають шаблони та дозволяють виконувати такі завдання, як розпізнавання шаблонів і прийняття рішень.

Нейронні мережі здатні вивчати та ідентифікувати шаблони безпосередньо з даних без попередньо визначених правил. Ці мережі складаються з кількох ключових компонентів:

Нейрони: основні одиниці, які отримують вхідні дані, кожен нейрон керується порогом і функцією активації.

Зв’язки: зв’язки між нейронами, які передають інформацію, регульовані вагами та упередженнями.

Ваги та зміщення: ці параметри визначають силу та вплив зв’язків.

Функції розповсюдження: механізми, які допомагають обробляти та передавати дані між шарами нейронів.

Правило навчання: метод, який коригує ваги та зміщення з часом для підвищення точності.

Навчання в нейронних мережах відбувається за структурованим багатоетапним процесом:

Вхідні обчислення: дані подаються в мережу.

Генерація виходу: на основі поточних параметрів мережа генерує вихід.

Ітеративне уточнення: мережа вдосконалює свій вихід, регулюючи вагові коефіцієнти та зміщення, поступово покращуючи свою продуктивність у різноманітних завданнях.

В адаптивному навчальному середовищі:

Нейронна мережа піддається впливу змодельованого сценарію або набору даних.

Такі параметри, як ваги та зміщення, оновлюються відповідно до нових даних або умов.

З кожним налаштуванням реакція мережі розвивається, дозволяючи їй ефективно адаптуватися до різних завдань або середовищ.

Для створення інтелектуальної системи управління дроном має бути використано дві штучні нейронні мережі:

- Згорткова нейронна мережа (CNN) для розпізнавання образу цілі
- Повнозвязна нейронна мережа для підсилення ПІД - регулятора

1.5.1 Загальна інформація по алгоритму CNN

Так як сам дрон, зазвичай, не має потужних обчислювальних можливостей, постає важливе питання вибору такого алгоритму навчання штучної нейронної мережі, який має достатню швидкодію для вирішення

задач розпізнавання і реалізації машинного зору. Одним з таких алгоритмів є згорткові нейронні мережі (Convolutional Neural Network, CNN). CNN працюють, імітуючи людський мозок, та використовують набори правил, які допомагають комп’ютерній системі знаходити особливості у зображеннях, розуміти та інтерпретувати інформацію. Даний алгоритм використанйи за основу в алгоритмі розпізнавання образів бібліотеки Mediapipe для Python.

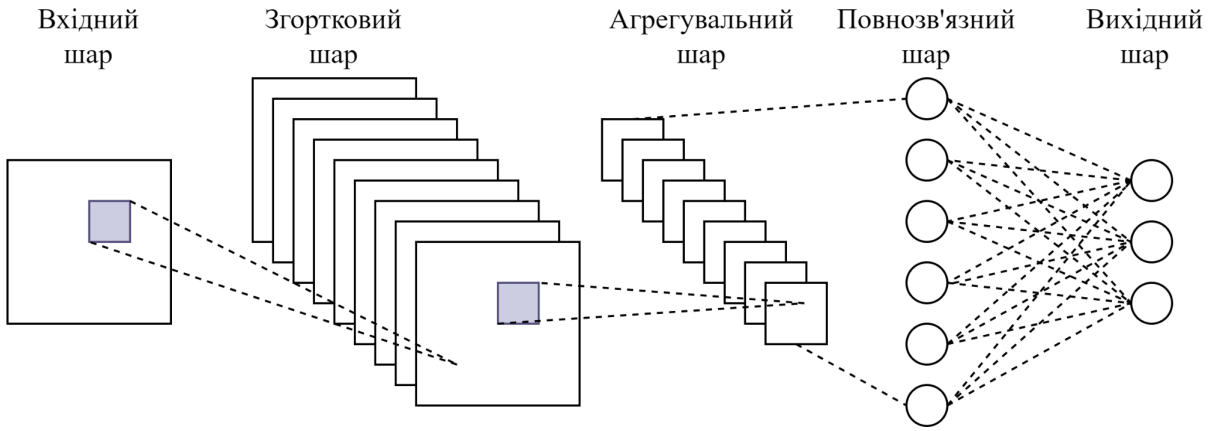

Рисунок 1.2 - Структурна схема згорткової нейронної мережі

На рис.1.2 зображено приклад простої схеми CNN. Зазвичай така схема складається з вхідного та вихідного шарів, а також з групи трьох прихованих шарів [7]. Кожен шар згорткової нейронної мережі обробляє дані та передає виявлені особливості іншому шару для наступного етапу обробки. Перші три шари виконують задачу визначення специфічних ознак на зображеннях. Останні два шари виконують задачу класифікації.

Вхідний шар приймає інформацію і визначає фіксований розмір для вхідних зображень, який можна змінювати за потреби. Наступним йде згортковий шар, який застосовує фільтри для виявлення текстур і меж об'єктів на зображенні. Далі рівень об’єднання зменшує розмір зображення, одночасно намагаючись зберегти отриману інформацію. Повнозв’язний шар являє собою класичну нейромережу багатошарового перцептрону, у якого кожен нейрон одного шару зв'язаний з кожним нейроном іншого шару. Останній вихідний шар видає інформацію по класифікації об'єкта.

1.5.2 Загальна інформація по глибоким повнозв’язним нейронним мережам

Глибока повнозв’язна нейронна мережа (або глибокий повнозв’язний персептрон) – це один із типів штучних нейронних мереж, що складається з декількох шарів штучних нейронів, де кожен нейрон на одному шарі пов'язаний із усіма нейронами наступного шару.

Основні компоненти глибокої повнозв’язної нейронної мережі:

1. Вхідний шар:

- Приймає вхідні дані (наприклад, числа, пікселі зображення, текстові вектори тощо).

- Кількість нейронів у вхідному шарі дорівнює кількості ознак вхідного сигналу.

2. Приховані шари складаються з нейронів, які виконують обчислення та передають інформацію далі.

3. Вихідний шар генерує кінцевий результат (наприклад, класифікацію, регресію або ймовірності).

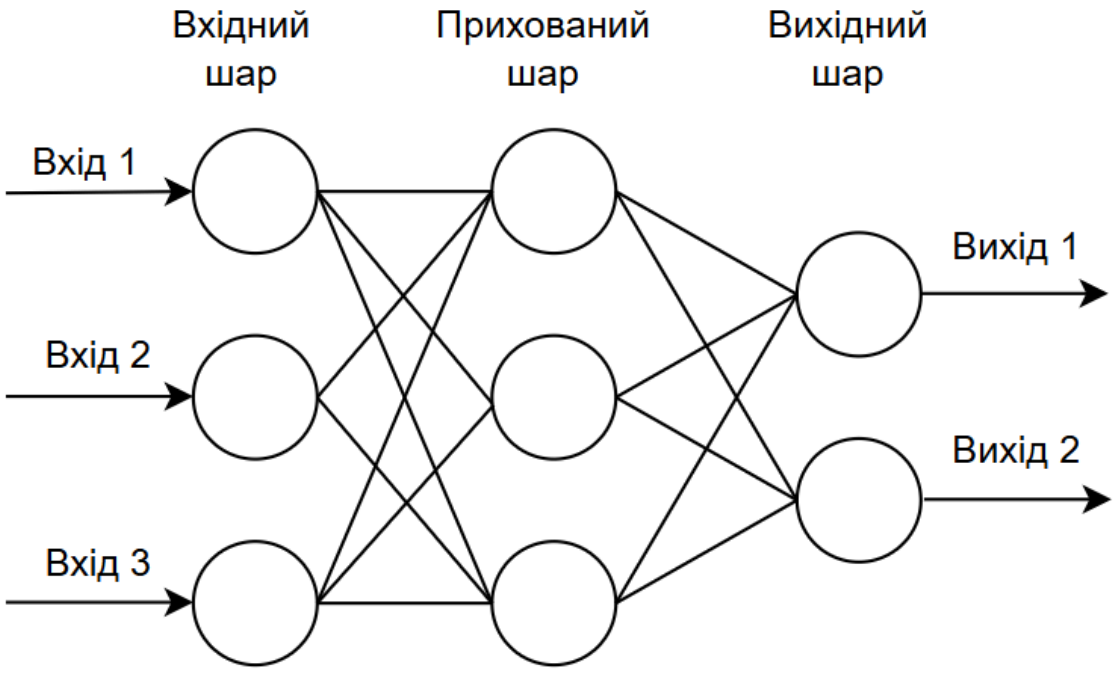

Рисунок 1.3 - Приклад трьохшарової повнозв’язної нейронної мережі

На рис. 1.3 нейрон зображений у вигляді кола. Структура нейрону зображена на рис. 1.4:

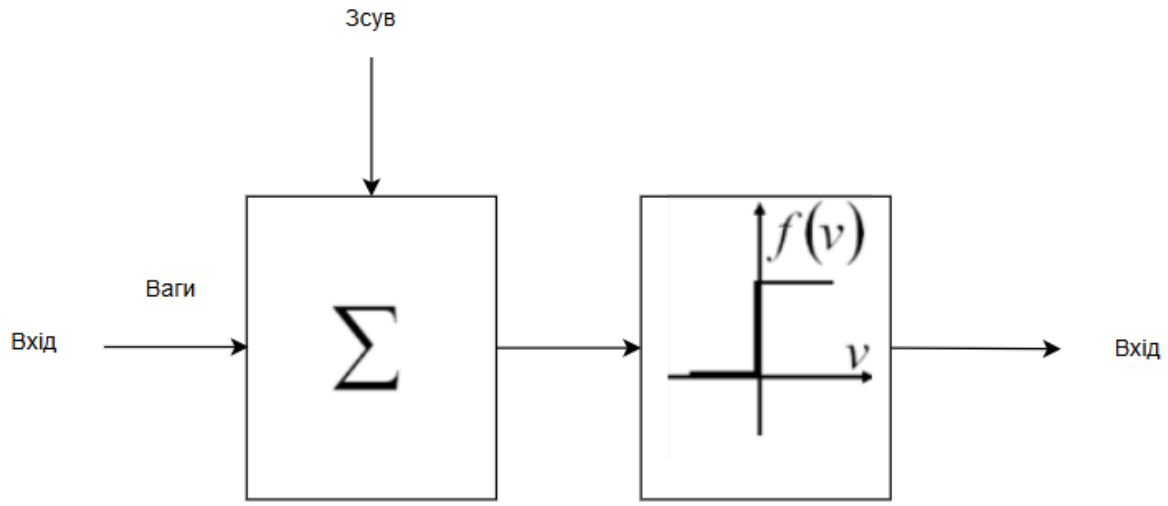

Рисунок 1.4 - Базовий штучний нейрон

Штучний нейрон працює шляхом прийому вхідних значень, далі оновлюються ваги, додається зсув, а потім дані проходять через активаційну функцію для отримання результату. Це дозволяє нейрону навчатися на прикладах і виконувати складні завдання, такі як класифікація, регресія, розпізнавання образів та інші.

Вхідні дані. Кожен штучний нейрон приймає один або кілька вхідних сигналів, які можуть бути представлені у вигляді чисел або векторів.

Вага. Кожен вхід має свою вагу. Вага визначає важливість конкретного вхідного сигналу для нейрона. Ваги коригуються в процесі навчання нейронної мережі.

Зсув. Для забезпечення гнучкості моделі додається ще один параметр — зсув. Зсув дозволяє коригувати активацію нейрона без залежності від вхідних даних. Він дозволяє змінювати поріг активації нейрона і допомагає йому бути більш гнучким у моделюванні складних функцій.

Активаційна функція. Дані проходять через активаційну функцію, яка визначає, чи буде нейрон активований, а також форму вихідного сигналу.

Активаційна функція додає нелінійність до моделі, що дозволяє нейронній мережі навчатися складним патернам. ReLU (Rectified Linear Unit) - є простою і популярною активаційною функцією, саме вона буде використана для створення даної інтелектуальної системи.

1.6 ПІД - регулятор

ПІД-регулятори являють собою складний механізм зворотного зв'язку, важливий для керування динамічними системами [8]. За своєю суттю вони працюють за допомогою трьох основних елементів: пропорційний *(P)*, інтегральний *(I)* і диференційний *(D)*. Кожна складова унікальним чином модулює вихідний сигнал на основі різниці між бажаним заданим значенням і фактичним виміряним значенням, відомим як похибка.

На рисунку 1.5 зображено схему роботи ПІД - регулятора:

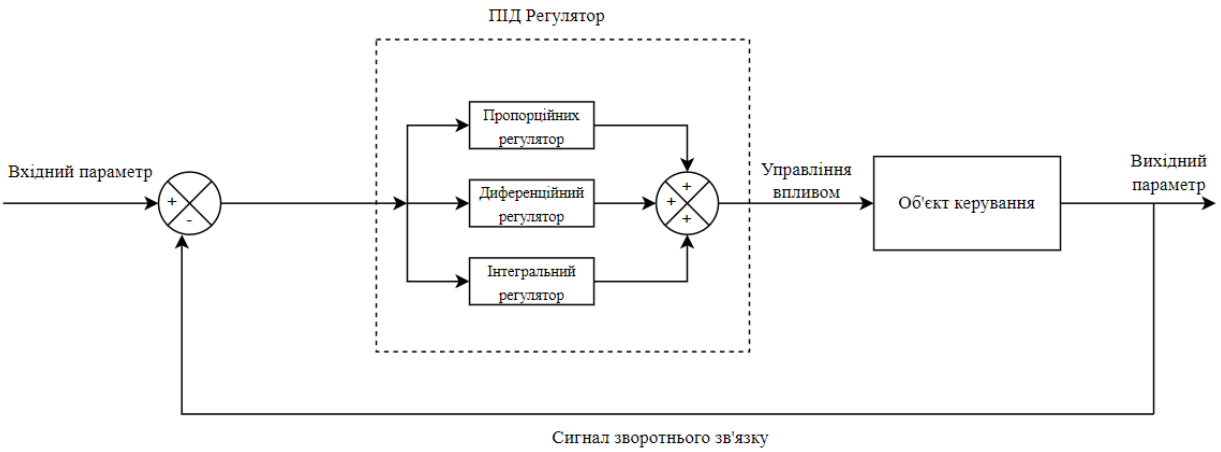

Рисунок 1.5 - Схема ПІД-регулятора

ПІД-регулятор безперервно обчислює сигнал помилки як різницю між бажаним заданим значенням (цільовим значенням) і поточною змінною процесу (виміряним значенням). На основі цього сигналу помилки контролер регулює вхідний сигнал керування системою, щоб мінімізувати помилку та підтримувати змінну процесу близько до заданого значення.

Ось як це працює докладніше:

- Розрахунок помилки: ПІД-регулятор постійно обчислює сигнал помилки як різницю між бажаним заданим значенням і поточною змінною процесу.

- Пропорційне керування: пропорційна *(P)* складова реагує на поточну помилку, виробляючи вихід, пропорційний величині помилки.

- Інтегральний контроль: інтегральна *(I)* складова відповідає кумулятивній сумі минулих помилок і спрямований на усунення будь-якої стаціонарної помилки. Він розраховується як інтеграл похибки за часом.

- Контроль похідної: похідна *(D)* реагує на швидкість зміни помилки та допомагає пом’якшити швидкі зміни в системі. Він розраховується як похідна похибки за часом.

- Контрольний вихід: контрольний вихід є сумою пропорційних, інтегральних і похідних членів.

- Налаштування керуючого входу: контрольний вихід застосовується як вхід до системи, що керується. Він регулює такі параметри системи, як положення клапанів, швидкість двигуна або нагрівальні елементи, щоб наблизити змінну процесу до заданого значення.

Контур зворотного зв'язку: змінна процесу безперервно вимірюється та подається назад до контролера, замикаючи контур керування. Контролер регулює вхід керування на основі зворотного зв’язку, прагнучи мінімізувати помилку та підтримувати змінну процесу на заданому значенні.

1.7 Оптимізація роботи ПІД - регулятора з допомогою нейронних мереж

Нейронні мережі можуть значно покращити роботу традиційних ПІД-регуляторів за допомогою адаптації параметрів в реальному часі, компенсації нелінійностей, зниження часу налаштування і покращення стабільності. Вони дають змогу системам швидко реагувати на зміни в

умовах навколишнього середовища, що робить систему керування більш точнішою та надійною.

1.7.1 Адаптивне налаштування параметрів регулятора

Нейронні мережі можуть використовуватися для автоматичного налаштування параметрів ПІД-регулятора в реальному часі, що дозволяє покращити його ефективність при змінних умовах. Наприклад:

- Під час зміни зовнішніх факторів, таких як швидкість вітру або зміни температури, ПІД-регулятор може недостатньо ефективно коригувати параметри без допомоги адаптації. Нейронні мережі можуть навчатися на історичних даних та коригувати параметри K_p, K_i, K_d таким чином, щоб забезпечити оптимальну продуктивність.
- Покращення стійкості системи: Нейронні мережі можуть бути навчені на даних про стрибки або зміни в зовнішніх умовах, що дозволяє більш точно передбачити та регулювати реакцію системи.

1.7.2 Корекція за допомогою передбачення майбутніх значень

Нейронні мережі, можуть бути використані для передбачення майбутніх значень помилки або відхилень, що дозволяє ПІД-регулятору не лише реагувати на поточну помилку, але й враховувати майбутні зміни. Наприклад, замість того, щоб просто коригувати помилку в реальному часі, система може передбачити, як ця помилка буде змінюватися на основі минулих даних, і підготувати регулятор до коригування параметрів заздалегідь.

1.8 ПІД - регулятор з нейронною мережею

Існує два популярних методи створення ПІД регулятора з нейронною мережею:

- Нейронна мережа з П-, І-, Д- нейронами
- Підсилення ПІД - регулятора нейронною мережею

Перший варіант є більш складним для реалізації, так як вимагає створення пропорційного, інтегрального, диференціального типу нейронів, і на їх базі створення тришарової нейронної мережі [9]:

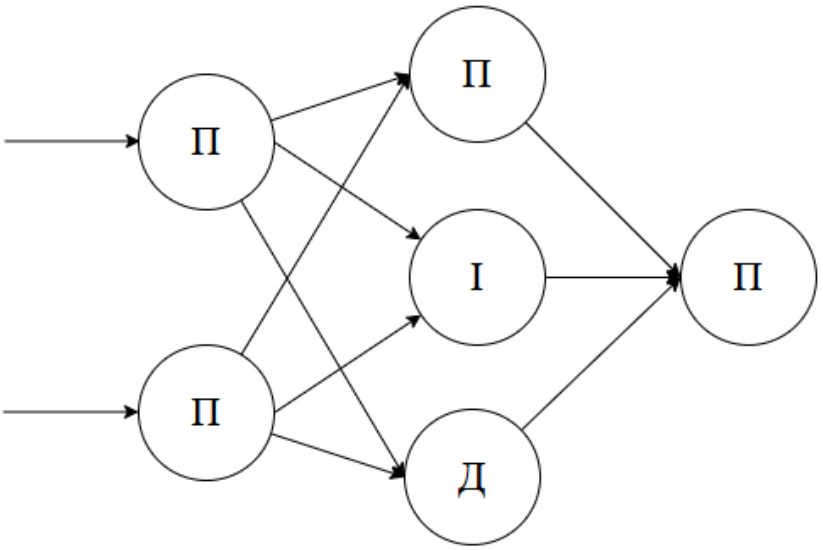

Рисунок 1.6 - Схема ШНМ з ПІД нейронами

Другий варіант більш простий в реалізації, так як являє собою класичний ПІД - регулятор вихід якого є входом в нейронну мережу, яка в свою чергу прогнозує майбутнє значенні ПІД - регулятора:

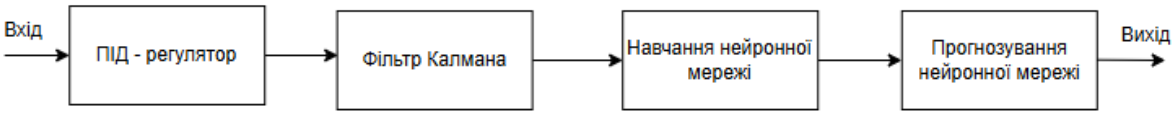

Рисунок 1.7 - Схема ПІД регулятора з підсиленням ШНМ

Даний підхід дозволяє порівняти як працює система з ПІД - регулятором і ПІДНМ - регулятором. Через це для реалізації даної інтелектуально системи навігації обрано другий варіант.

Висновки до розділу:

У цьому розділі було розглянуто основні підходи та технології, які можуть бути використані для реалізації системи автономної навігації для дронів. Окрему увагу було приділено FPV-дронам, зокрема дрону DJI Tello,

розглянуто його основні характеристики, можливості програмування та головним перевагам і недолікам. Важливим елементом цього розділу є розгляд технології машинного зору та застосування штучних нейронних мереж для навігації, що дозволяють підвищити ефективність роботи дронів. Також було розглянуто роль ПІД-регулятора в управлінні автономними системами, а також його оптимізацію з використанням нейронних мереж, що дозволяє адаптувати параметри регулятора та покращити передбачення майбутніх значень. Використання таких підходів відкриває нові перспективи в управлінні дронами та їх інтеграції в складні автономні системи.

Таким чином, результати даного розділу демонструють важливість інтеграції сучасних технологій у створення високопродуктивних автономних систем для дронів, зокрема за допомогою адаптивного управління, машинного зору та нейронних мереж.

РОЗДІЛ 2
ІДЕНТИФІКАЦІЯ МАТЕМАТИЧНОЇ МОДЕЛІ ТА СИНТЕЗ КЕРУВАННЯ ПРОЦЕСОМ

2.1 Підбір параметрів ПІД - регулятора

Параметри налаштування ПІД - регулятора повинні бути обрані так, щоб у замкненій системі автоматичного регулювання забезпечувався необхідний рівень робастності. При цьому показники якості регулювання повинні відповідати заданим вимогам або досягати оптимальних значень. Оскільки в теорії автоматичного регулювання існують різні способи оцінки запасу стійкості та застосовуються різні критерії якості, в інженерних розрахунках використовують кілька методів для визначення оптимальних параметрів налаштування регуляторів.

2.1.1 Порівняння методів підбору параметрів ПІД - регулятора

В таблиці 2.1 представлені основні переваги та недоліки методів визначення параметрів ПІД регулятора [10]:

Таблиця 2.1 – Порівняння методів підбору параметрів ПІД - регулятора

Назва методу	Переваги	Недоліки
Циглера - Нікольса	Простий у реалізації та розумінні, широко використовуваний.	Може спричинити коливання або перевищення.
Реле зворотного зв'язку	Може ефективно визначити критичне посилення та період коливань.	Потрібна стабільна система, може працювати не для всіх процесів.

Продовження табл. 2.1

Внутрішній контроль моделі	Стійкість до неточностей моделі.	Більш складний і вимагає точної моделі процесу.
Меток Коен - Кун	Добре підходить для процесів із значним запізненням.	Не ефективний для дуже нелінійних систем.
Модель еталонного адаптивного керування	Автоматично пристосовується до змін у динаміці системи.	Комплекс для реалізації та налаштування.

Для підбору параметрів ПІД - регулятора було обрано метод Циглера - Нікольса, так як він є достатньо ефективним і простим в реалізації.

2.2 Метод Циглера - Нікольса

Евристичний метод налаштування відомий як метод Циглера–Ніколса [11], представлений Джоном Г. Ціглером і Натаніелем Б. Ніколсом у 1940-х роках. Як і в методі вище, підсилення K_i і K_d спочатку встановлюються на нуль. Пропорційне посилення збільшується, поки не досягне кінцевого коефіцієнта посилення K_u, при якому вихідний сигнал петлі починає постійно коливатися. K_u та період коливань T_u використовуються для встановлення коефіцієнтів підсилення наступним чином:

Таблиця 2.2 – Розрахунок коефіцієнтів ПІД - регулятора

Тип регулятора	Пропорційний коефіцієнт, K_p	Інтеграційний коефіцієнт, K_i	Диференційний коефіцієнт, K_d
ПІД	$0.60*K_u$	$1.2*K_u/T_u$	$3*K_u*T_u/40$

Замість цього часто вимірюється частота коливань, і зворотні величини кожного множення дають той самий результат. Ці переваги застосовуються до

ідеальної паралельної форми ПІД-регулятора. У застосуванні до стандартної форми ПІД тільки інтегральний і похідний підсилення K_i і K_d залежать від періоду коливань T_u.

2.3 Фільтр Калмана

Багато сучасних систем використовують кілька датчиків для оцінки прихованих (невідомих) станів за допомогою серії вимірювань [12]. Наприклад, GPS - приймач може оцінювати місцезнаходження та швидкість, де місцезнаходження та швидкість представляють приховані стани, тоді як диференціальний час надходження сигналів від супутників служить вимірюванням.

Однією з найбільших проблем систем відстеження та контролю є забезпечення точної оцінки прихованих станів за наявності невизначеності. Наприклад, GPS-приймачі схильні до невизначеності вимірювань, що залежить від зовнішніх факторів, таких як тепловий шум, атмосферні впливи, незначні зміни в позиціях супутника, точність годинника приймача тощо.

Фільтр Калмана — широко використовуваний алгоритм оцінки, який відіграє вирішальну роль у багатьох сферах. Він призначений для оцінки прихованих станів системи, навіть якщо вимірювання є неточними та невизначеними. Крім того, фільтр Калмана передбачає майбутній стан системи на основі попередніх оцінок.

Фільтр названо на честь Рудольфа Е. Калмана (19 травня 1930 – 2 липня 2016). У 1960 році Калман опублікував свою знамениту статтю, в якій описував рекурсивне рішення проблеми лінійної фільтрації дискретних даних.

2.4 Стек технологій

Для реалізації інтелектуальної системи обрано мову програмування Python, яка має велику кількість необхідних бібліотек для роботи з машинним

зором, машинним навчанням, нейронними мережами, роботою з графікою, фільтри, багатопотоковість, тощо

Середовищем розробки обрано Visual Studio Code через її простоту і доступність.

Всі файли проекту зберігаються в репозиторій Github.

2.5 Огляд бібліотек необхідних для реалізації системи

2.5.1 Бібліотека Mediapipe

MediaPipe Solutions [13] пропонує набір бібліотек та інструментів для швидкого впровадження методів штучного інтелекту (AI) та машинного навчання (ML) у ваші програми. Досить просто інтегрувати ці рішення в проект, налаштувати їх відповідно до конкретних вимог та використовувати на різних платформах розробки. MediaPipe Solutions є частиною проекту з відкритим вихідним кодом MediaPipe, що дозволяє додатково налаштувати код під специфічні потреби вашої програми. Набір рішень MediaPipe включає такі компоненти. Ці бібліотеки та ресурси забезпечують основні функції для кожного рішення MediaPipe:

- MediaPipe Tasks: міжплатформені API та бібліотеки для розгортання рішень.
- Моделі MediaPipe: попередньо навчені, готові до запуску моделі для використання з кожним рішенням.

Ці інструменти дозволяють налаштувати та оцінити рішення:

- MediaPipe Model Maker: налаштування моделі для заданих даних.
- MediaPipe Studio: візуалізація, оцінка та порівняння рішення у своєму браузері.

Завдання MediaPipe Pose Landmarker [14] дозволяє виявляти орієнтири людських тіл на зображенні чи відео. Досить зручно використовувати цю функціональність, щоб визначити розташування тіла, проаналізувати поставу та класифікувати рухи. Це завдання використовує моделі машинного

навчання (ML), які працюють з окремими зображеннями або відео. Завдання виводить орієнтири пози тіла в координатах зображення та в координатах 3-вимірного світу. Основою даної бібліотеки є згорткова нейронна мережа CNN.

2.5.2 Бібліотека djitellopy

Інтерфейс python для дронів DJI Tello з використанням офіційних Tello SDK і Tello EDU SDK. Ця бібліотека має такі функції:

- виконання команд tello
- легкий доступ до відеопотоку
- отримання та аналіз параметрів стану дрону
- керування роєм дронів
- підтримка python версії вище ніж 3.6

Основні можливості бібліотеки djitellopy:

- Управління дроном: djitellopy можна використовувати для виконання різних команд, таких як зліт, посадка, політ у певному напрямку, та інші функції.
- Отримання відео: Бібліотека дозволяє отримувати відеопотік з камери дрона в реальному часі.
- Обробка зображень: можливість захоплювати та обробляти зображення, зроблені дроном.
- Автоматизація: З djitellopy можна створювати автоматизовані польоти та сценарії для виконання різних завдань.

2.5.3 Бібліотека pygame

Pygame — набір кросплатформенних модулів для мови програмування Python, створений для розробки відеоігор. Включає в себе бібліотеки комп'ютерної графіки і звуку на базі SDL.[15]

Функції ядра бібліотеки написані на C та Assembly. Це робить pygame дедалі швидшим, через те, що C код зазвичай в 10-20 разів швидший за Python, а Assembly в свою чергу в 100 раз швидший, ніж Python.

Також в залежностях pygame немає OpenGL, що забезпечує простішу розробку та портативність.

2.5.4 Бібліотека cv2

OpenCV [16], скорочення від Open Source Computer Vision Library, — це бібліотека комп'ютерного зору та машинного навчання з відкритим кодом. Спочатку був розроблений компанією Intel, тепер підтримується спільнотою розробників OpenCV Foundation.

Opencv — це величезна бібліотека з відкритим кодом для комп'ютерного зору, машинного навчання та обробки зображень. Тепер він відіграє важливу роль у роботі в режимі реального часу, що дуже важливо в сучасних системах. Використовуючи його, можна обробляти зображення та відео, щоб ідентифікувати предмети, обличчя або навіть почерк людини.

2.5.5 Бібліотека torch

Torch — відкрита бібліотека для машинного навчання [17], система для наукових обчислень та мова сценаріїв на основі мови програмування Lua. Пропонує широкий спектр алгоритмів для глибокого навчання і використовує мову сценаріїв LuaJIT та реалізацію мовою C в основі. Станом на 2018 рік, Torch більше не перебуває в активній розробці. Проте, станом на серпень 2019 року активно розроблюють PyTorch.

2.5.6 Бібліотека kalman

Проста і гнучка бібліотека для фільтра Калмана, яка дозволяє працювати з одно- і багатовимірними даними.

2.5.7 Бібліотека unittest

Фреймворк модульного тестування unittest [18] спочатку був натхненний JUnit і має схожий підхід, як і основні фреймворки модульного тестування іншими мовами. Він підтримує автоматизацію тестування, спільне використання коду налаштування та завершення тестів, об'єднання тестів у колекції та незалежність тестів від структури звіту.

Щоб досягти цього, unittest підтримує деякі важливі концепції в об'єктно-орієнтований спосіб:

Тестовий блок (test fixture) являє собою підготовку, необхідну для виконання одного або кількох тестів і будь-яких пов'язаних дій з очищенням даних. Це може включати, наприклад, створення тимчасових або проксі-баз даних, каталогів або запуск серверного процесу.

Тестовий приклад (test case) — це окрема одиниця тестування. Він перевіряє конкретну відповідь на певний набір вхідних даних. unittest надає базовий клас TestCase, який можна використовувати для створення нових тестів.

Набір тестів (test suite) — це набір тестів, наборів тестів або обох. Він використовується для агрегування тестів, які слід виконувати разом.

Програма запуску тестування (test runner) — це компонент, який керує виконанням тестів і надає результат користувачеві. Виконувач може використовувати графічний інтерфейс, текстовий інтерфейс або повертати спеціальне значення для вказівки результатів виконання тестів.

2.5.8 Бібліотека matplot

Matplotlib [19] — це комплексна бібліотека для створення статичних, анімованих та інтерактивних візуалізацій на Python. Matplotlib робить легкі речі легкими, а складні – можливими.

2.6 Алгоритм реалізації інтелектуальної системи навігації дрону

Розглянемо алгоритм реалізації системи розпізнавання та захвату цілі, а також керування рухом дрону за ціллю який представлено на рисунку 2.1:

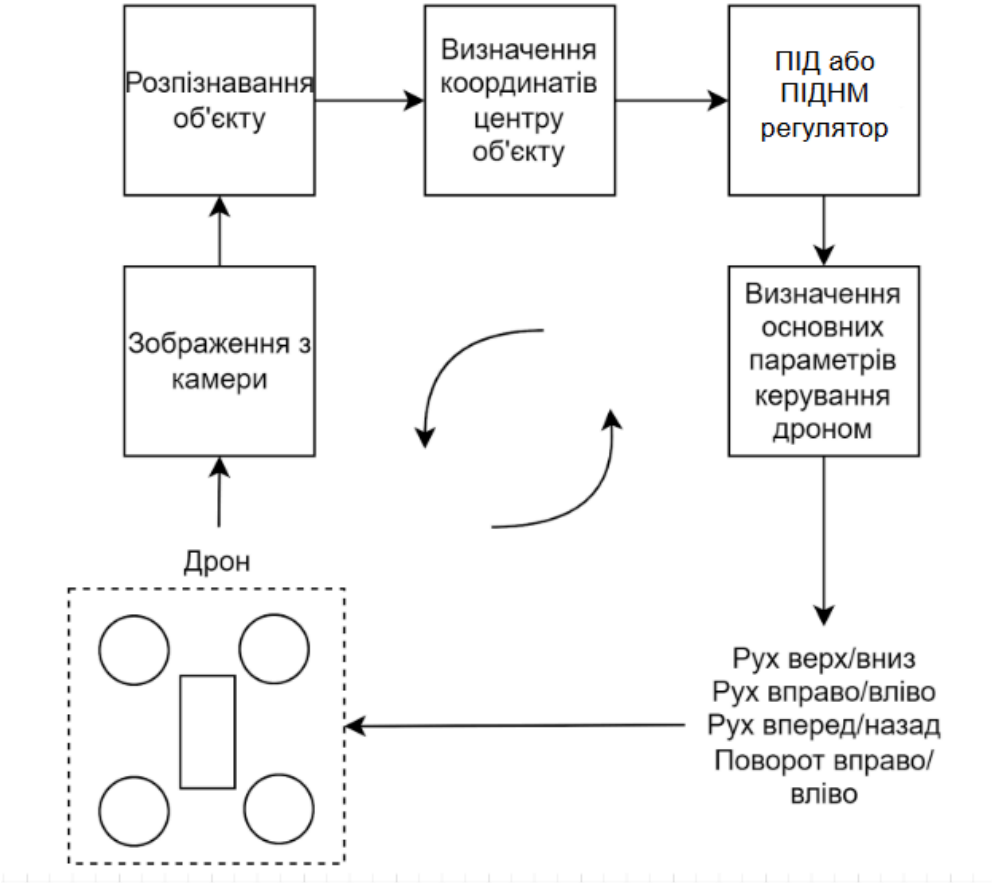

Рисунок 2.1 - Алгоритм реалізації машинного зору системи

За допомогою вбудованої камери робиться захват відео, кожен кадр якого можна обробити алгоритмом розпізнавання образів відповідно до задачі що вирішеється. Одним із потужних сучасних інструментів розпізнавання є бібліотека MediaPipe Solutions від Google. За допомогою неї можна розпізнати об’єкт і визначити його координати. Шкала координат визначається кількістю пікселів зображення по горизонталі і вертикалі.

Наступним етапом є робота ПІД-Регулятора. Для цього необхідно визначити пропорційний коефіцієнт регуляції та обраховувати помилку положення дрону. Коефіцієнти ПІД - регулятора визначаються за допомогою методу Циглера-Нікольса. Для визначення помилки обчислюємо різниці між

центром зображення і поточними координатами цілі. Далі отримані величини передаються дрону як команди повороту дрону вправо або вліво і його рух вверх - вниз.

Нехай величина зображення 960x720 пікселів. Центр зображення буде мати координати [480; 360]. Якщо координати цілі, наприклад [720;540], то величина помилки буде обчислена наступним чином:

delta_X = 720-480

delta_Y = 540-360

Далі ці величини переводяться у шкалу яка є зрозумілою фреймворку дрону, від -100 до 100 і передаються як команди повороту та руху по горизонтальній осі і руху по вертикальній осі. Схематично даний алгоритм зображено на рисунку 2.2:

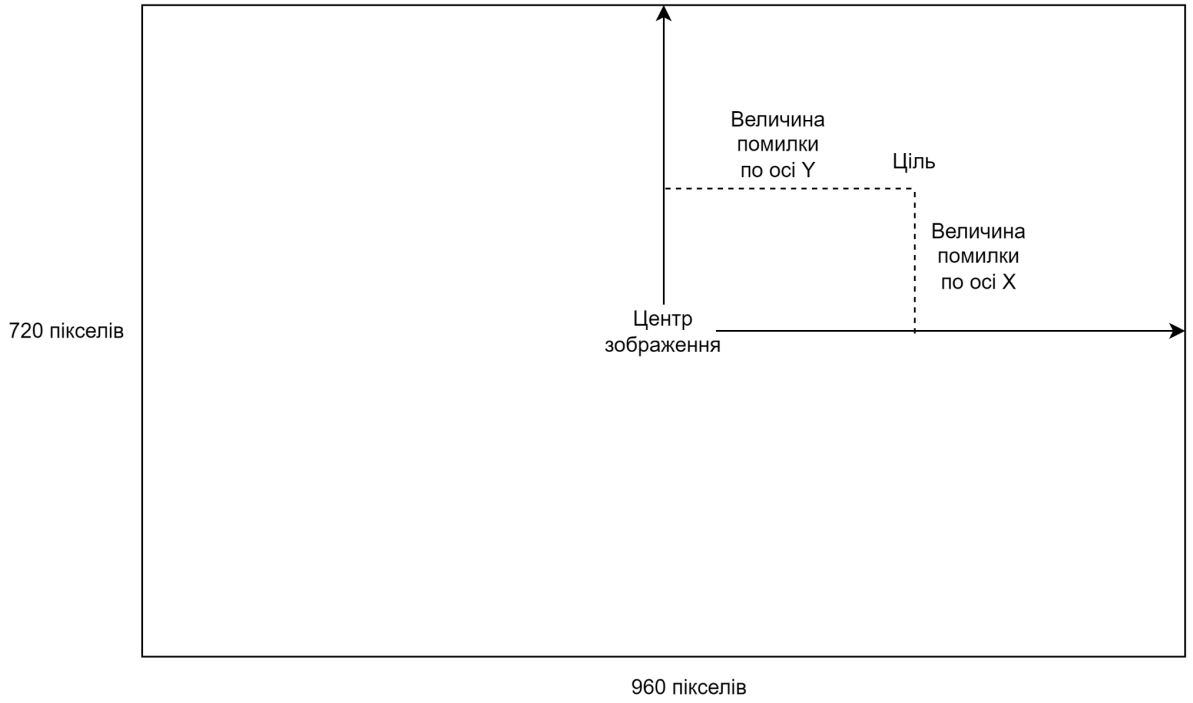

Рисунок 2.2 - Обчислення помилки для ПІД-регулятора

Бібліотека MediaPipe також має можливість визначати координати відстані до розпізнаної цілі по осі аплікат.

2.7 Тестування системи

Основні підходи тестування системи:

- Юніт тести для методів, які роблять калькуляції. Основний модуль: Core.
- Тестування і порівняння роботи системи з ПІД- та ПІДНМ-регулятором
- Тестування системи в умовах без збурень, зі збуреннями в вигляді вітру і поганої видимості.

Висновки до розділу:

У даному розділі було розглянуто методи підбору параметрів ПІД-регулятора. Для налаштування параметрів регулятора було обрано метод Циглера-Нікольса, який є простим і достатньо ефективним.

Додатково було описано стек необхідних технологій, що включає програмні бібліотеки, необхідні для реалізації системи, такі як Mediapipe для обробки зображень, djitellopy для роботи з дроном, pygame для графічного інтерфейсу, а також бібліотеки cv2, torch, kalman, unittest і matplot для різних аспектів системи, включаючи машинне навчання.

РОЗДІЛ 3
ПРОГРАМНО-ТЕХНІЧНА РЕАЛІЗАЦІЯ СИСТЕМИ КЕРУВАННЯ ПРОЦЕСОМ

Для даної інтелектуальної системи використано мову Python, яка є досить простою, та містить велику кількість потужних інструментів, такі як:

- OpenCV для реалізації комп’ютерного зору
- djitellopy для управління дроном DJI, чи інша бібліотека відповідно до моделі дрону.
- Mediapipe рішення, яке надає набір бібліотек та інструментів для швидкого застосування методів штучного інтелекту і машинного навчання
- torch для створення трьохшарової нейронної мережі
- kalman для застосування фільтра Калмана
- unittest для тестування
- matplot для створення графіку

3.1 Архітектура системи

Програмний додаток має модульну структуру [20], яка складається з трьох основних шарів:

- GUI
- Device
- Core

Код кожного шару зберігається в окремому файлі. На рисунку 3.1 зображена схема шарів програмного додатку і зв'язки між ними. Також вказані основні методи кожного шару:

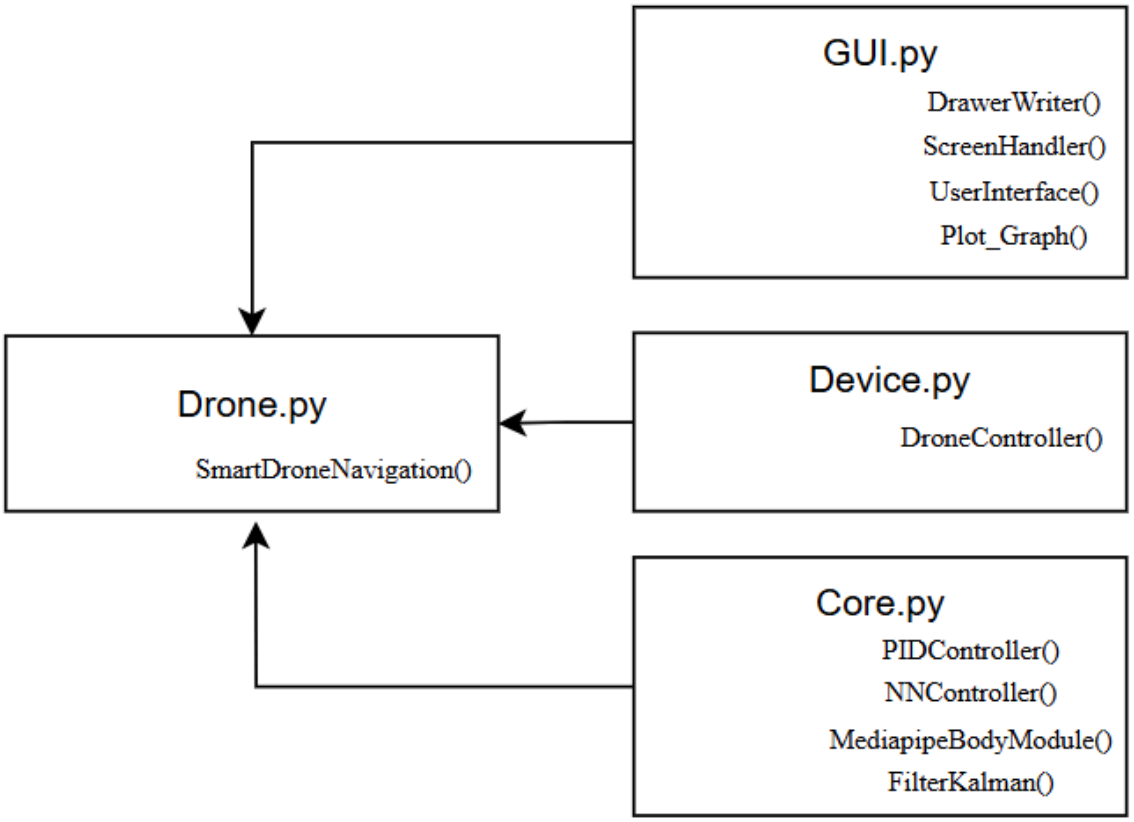

Рисунок 3.1 - Схема модулів програмного додатку

Основною перевагою використання модульної системи додатку є незалежність частин коду одного від іншого. Це дає, в перспективі, можливість переходу на інший дрон без суттєвих змін програми, лише Device шар потрібно буде переписати, вся інша логіка буде працювати без змін.

3.1.1 GUI шар

Даний шар містить наступні класи:

Writer - використовується для виводу тексту на екран і малювання графічних елементів

DrawerWriter - додавання даних на екран. Малювання графічних елементів

ScreenHandler - отримання даних розміру кадру, FPS, тощо

UserInterface - обробка вводу з клавіатури. В системі реалізовано керування системою за допомогою наступних клавіш:

- R (Recording) - увімкнути/вимкнути запис відео з камери пристрою
- T (Take off) - зліт дрону
- L (Land) - приземлення дрону

Plot_Graph - створення графіка

3.1.2 Шар Device

Даний шар містить наступний клас:

DroneController - використовується для з'єднання, керування дроном.

3.1.3 Шар Core

Даний шар містить наступні класи:

PIDController - реалізація логіки роботи ПІД - регулятора

NNController - реалізація нейронної мережі

MediapipeBodyModule - розпізнавання цілі

FilterKalman - фільтр Калмана

3.1.4 Файл Drone.py

Даний файл має доступ до усіх трьох вищеописаних шарів, і викликає один головний метод SmartDroneNavigation, в якому реалізована основна логіка даної інтелектуальної системи навігації.

3.1.5 Файл завдання MediaPipe Pose Landmarker

Файл pose_landmarker_lite.task (завдання MediaPipe Pose Landmarker) зберігається в каталозі /Lib.

Завдання MediaPipe Pose Landmarker [21] дозволяє виявляти орієнтири людських тіл на зображенні чи відео. Це завдання може використовуватись, щоб визначити ключове розташування тіла, проаналізувати поставу та класифікувати рухи. Також воно використовує моделі машинного навчання (ML), які працюють з окремими зображеннями або відео. Завдання виводить

орієнтири пози тіла в координатах зображення та в трьох-вимірних координатах.

3.1.6 Файл .gitignore

В даному файлі зберігаються назви файлів або маски по яким можна визначити файли, які не будуть зберігатись в репозиторій.

3.2 Структура нейронної мережі

Для того, щоб додати в систему можливість адаптуватися і підлаштовуватися під навколишні умови використовується глибока повнозв'язна нейронна мережа . Мережа не потребує пре-тренування, навчання відбувається під час роботи системи.

Дана нейронна мережа використовується для підсилення ПІД - регулятора і має наступну трьох - шарову архітектуру:

- Вхідний шар, який приймає 3 входи та має 64 нейрони.
- Проміжний (прихований) шар з 64 нейронами.
- Вихідний шар, що видає 3 значення на виході.

Оптимізатор: Використовується оптимізатор Adam з коефіцієнтом навчання 0.001 для налаштування ваг нейронної мережі.

Оптимізатор Adam (Adaptive Moment Estimation) [22] є одним з найбільш популярних методів оптимізації в машинному навчанні для нейронних мереж. Він поєднує переваги двох інших методів оптимізації — AdaGrad і RMSProp — що дозволяє ефективно адаптувати швидкість навчання для кожного параметра окремо. Adam використовує оцінки першого (середнє значення градієнтів) та другого (квадрати градієнтів) моментів, що допомагає зменшити коливання під час оптимізації і забезпечує більш стабільне і швидке навчання. У нашому випадку використовується коефіцієнт навчання 0.001, що є оптимальним значенням для багатьох задач.

Функція втрат: Використовується MSELoss [23] (середньоквадратична помилка) для обчислення втрат, що визначають, наскільки добре мережа передбачає цільові значення.

Функція втрат MSELoss (Mean Squared Error Loss) є стандартною метрикою для регресійних задач. Вона вимірює середнє значення квадрату різниці між передбаченими значеннями та реальними цільовими значеннями. MSELoss застосовується для оцінки того, наскільки добре модель передбачає значення, і чим менша величина втрат, тим точніше модель. Ця функція втрат широко використовується в задачах, де потрібно мінімізувати відхилення передбачень від реальних результатів.

3.3 Домовленості про стиль коду і процесу розробки

1. Слідувати принципам KISS, DRY, YAGNI
2. Використовувати snake_case, це коли всі слова написані маленькими літерами, і між ними ставиться підкреслення
3. Трьохшарова модульна архітектура
4. Код, що самодокументується. Назви змінних, класів і методів повинні відображати їх суть. Використання коментарів.
5. Всі комміти мають робитись в Github
6. Юніт тести мають бути швидкі. Комміт коду робиться після перевірки коду юніт тестами
7. Не використовувати анти-патерни програмування

3.4 Патерни програмування

При створення системи були використані архітектурні патерни проектування [24]. Розглянемо основні методи та патерни, які були для них використані:

Фасад (для спрощеного доступу до складних систем, таких як Mediapipe та Kalman Filter). Патерн проектування Фасад (Facade) є структурним патерном, який надає спрощений інтерфейс до складної системи або набору підсистем. Він дозволяє знизити складність використання системи, приховуючи від користувача внутрішні деталі реалізації, та забезпечує єдиний точку доступу для взаємодії з системою.

Адаптер (для адаптації між різними підсистемами, наприклад, ПІД-регулятор та нейронна мережа). Патерн проектування Адаптер (Adapter) є структурним патерном, який дозволяє об'єктам з несумісними інтерфейсами працювати разом. Він виступає як "перехідний" елемент, що адаптує інтерфейс одного класу до інтерфейсу, який очікує інший клас або система. Таким чином, адаптер дозволяє використовувати об'єкти, які не підходять за формою або способом взаємодії, без необхідності змінювати їхній код.

Стратегія (для вибору між різними методами вирішення задачі). Патерн проектування Стратегія (Strategy) є поведінковим патерном, який дозволяє визначити набір алгоритмів, інкапсулювати кожен з них і зробити їх взаємозамінними. Це дозволяє змінювати поведінку об'єкта в залежності від контексту, не змінюючи його код.

Фабрика (для ініціалізації складних об'єктів, як KalmanFilter). Патерн проектування Фабрика (Factory) є підходом для створення об'єктів, що дозволяє делегувати процес створення об'єктів певному класу чи методу. Основна мета цього патерну — спростити створення об'єктів, забезпечивши більшу гнучкість у виборі типу створюваного об'єкта без безпосереднього

виклику конструктора. Це дозволяє приховати деталі створення об'єкта від клієнтського коду і зробити систему більш масштабованою.

Спостерігач (для спостереження та обробки результатів зображень). Патерн проектування Спостерігач (Observer) є поведінковим патерном, який визначає залежність типу один-до-багатьох між об'єктами. Це означає, що коли один об'єкт змінює свій стан, всі його залежні об'єкти (спостерігачі) автоматично отримують оновлення про ці зміни. Патерн спостерігача дозволяє об'єктам спілкуватися без необхідності жорстко зв'язувати їх між собою, що робить систему більш гнучкою.

Клас UserInterface використовує Командний паттерн для обробки вводу від користувача. Кожна клавіша, яку натискає користувач, виконує певну команду. Наприклад:

- натискання “T” ініціює команду "takeoff",
- натискання “L” ініціює команду "land",
- натискання “R” перемикає режим запису відео

Патерн проектування Команда (Command) є поведінковим патерном, який перетворює запит на дію в об'єкт. Це дозволяє інкапсулювати всі параметри, необхідні для виконання операції, у вигляді об'єкта, що забезпечує більш гнучке управління виконанням операцій. Патерн Команда розділяє виклик операції від її виконання, що дозволяє зберігати відокремленість клієнта від конкретних реалізацій операцій.

Використання патернів допомагає зробити код більш гнучким, розширюваним та підтримуваним, дозволяючи легко змінювати окремі частини системи без необхідності змінювати інші компоненти.

3.5 Інтерфейс користувача

В головне вікно системи виводиться відео з дрону.

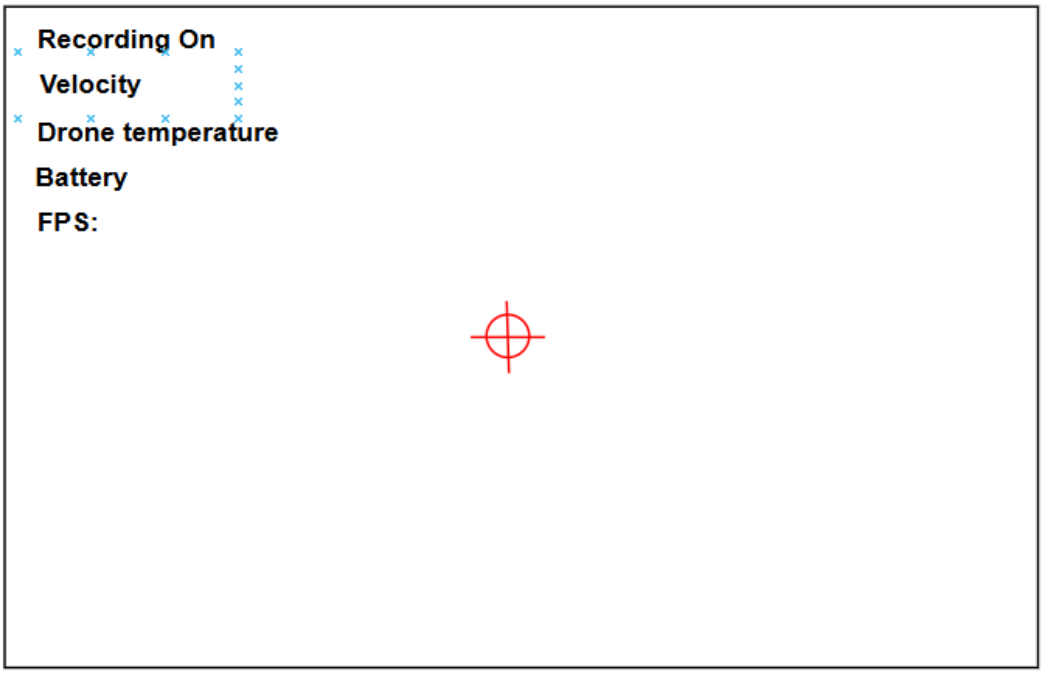

Рисунок 3.2 - Схема інтерфейсу користувача системи

В лівому верхньому кутку екрана виведено додаткову інформацію:

- Recording On - ведеться запис відео
- Velocity - кут повороту дрону до розпізнаної цілі
- Drone temperature - температура дрону в Цельсіях
- Battery - відсоток заряду батареї дрону
- FPS - кількість кадрів в секунду

Приклад інтерфейсу користувача зображено на рис. 3.2:

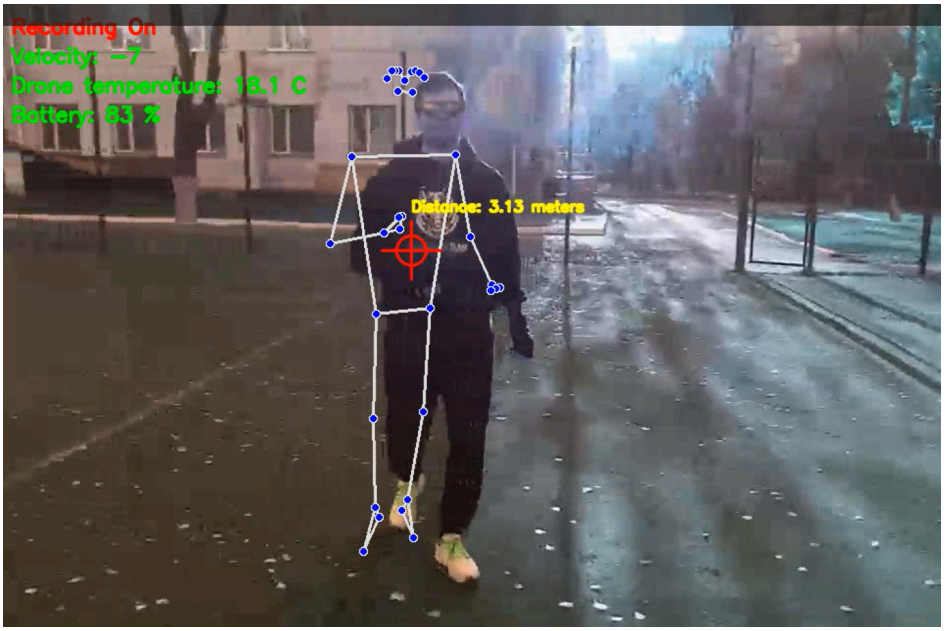

Рисунок 3.3 - Приклад інтерфейсу користувача створеної системи

Для порівняння роботи ПІД та ПІДНМ регулятора створено наступний графік:

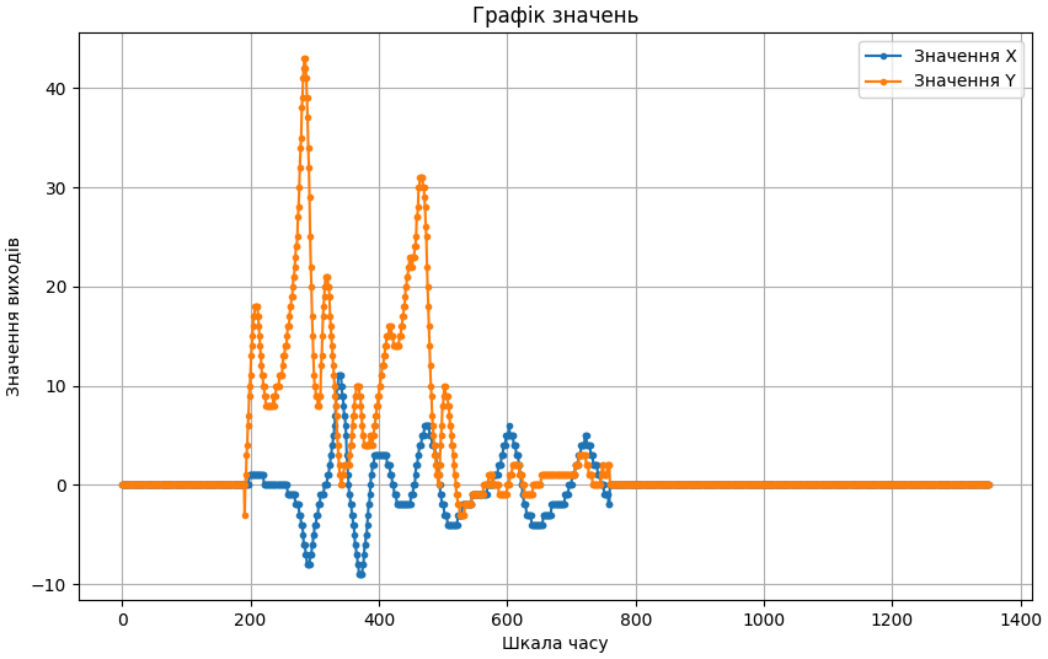

Рисунок 3.4 - Графік виходів ПІД або ПІДНМ регулятора

По осі X відображається шкала ітерації. На кожній ітерації відбувається аналіз одного кадру зображення. Камера дрону працює з FPS рівним 30 кадрів в секунду. Дані одиниці можна перевести в секунди, поділивши значення на 30.

По осі Y відображається значення виходу яке вимірюється в сантиметрах руху дрону. Значення які передаються дрону для руху по осі X позначені синім кольором, значення які передаються дрону для руху по осі Y позначені помаранчевим кольором.

3.6 Схема алгоритму роботи системи

Дана система має змогу працювати з камерою дрону або камерою ноутбука. Камера ноутбука може використовуватись для спрощення виконання технічних задач, таких як рефакторинг, реінжиніринг та написання чи тестування частин коду, які безпосередньо не пов’язані з процесом автоматизації. Перемикання режиму роботи камери задається через глобальний параметр DRONE_STREAM (True - відео береться з дрону, False - з камери ноутбуку) в файлі Device.py

Загальний алгоритм роботи програми зображено на рисунку 3.5 у вигляді блок - схеми:

Рисунок 3.5 - Загальна блок схема алгоритму системи

Алгоритм блоку інтелектуальної системи навігації зображено на рис. 3.6:

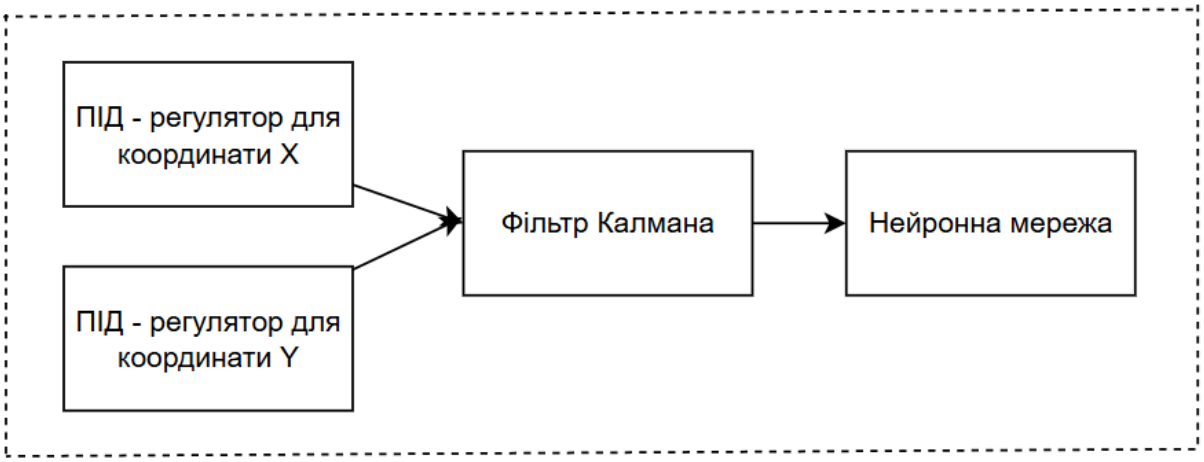

Рисунок 3.6 - Схема блоку інтелектуальна система навігації

Як бачимо з рис. 3.6 система використовує ПІД - регулятор для визначення вихідних даних окремо для осі X та Y. Далі ці дані проходять через фільтр Калмана. Вихідні дані з фільтру передаються в нейронну мережу, яка навчається на цих даних і прогнозує дані для наступної ітерації. Для навчання використовуються реальні дані отримані з ПІД - регулятора і прогнозовані дані з попередньої ітерації. Дану нейронну мережу можна легко відключити змінивши всього один параметр в коді фалу drone.py[102]:

outputs = neural_network.apply_NN(neural_network, outputZ, filtered_outputX, filtered_outputY, outputs, **False**)

Якщо значення параметра True - система працює в режимі ПІДНМ, якщо False - ПІД.

3.7 Опис логіки роботи програмного додатку.

1. В __main__ створює екземпляр класу SmartDroneNavigation(). Це є стартовою точкою запуску програми.
2. Ініціалізація даних. Створюються об'єкти усіх необхідних класів.
3. Визначається тип пристрою: дрон чи камера з ноутбуку. Камера з ноутбуку може використовуватись для тестування роботи системи

розпізнавання та виводу даних на екран. Відбувається приєднання до пристрою.

4. Відбувається розпізнавання образу на кожному кадрі, який отримується з камери пристрою з частотою 30 кадрів в секунду.

5. Якщо об'єкт розпізнано, визначаються координати X та Y центру об'єкта

6. Визначається швидкість руху дрону. Чим менша помилка, тим вище швидкість руху і навпаки. Без обмежень для останніх метрів до цілі.

7. Дані координати відправляються в ПІД регулятор, який намагається визначити помилку як відстань в пікселях від центру об'єкта до центру екрану

8. Визначаються П, І, Д складові регулятора для двох (X, Y) осей.

9. Вихідні дані ПІД - регулятора передаються в фільтр Калмана, який прибирає з системи керування шуми і випадкові дані.

10. Фільтровані дані передаються в нейронну мережу, яка навчається на цих даних і адаптує алгоритм роботи системи під них.

11. Вихід з нейронної мережі для кожної осі перетворюється на зрозумілі для дрону команди:

- вправо/вліво по осі X
- вверх/вниз по осі Y
- швидкість руху вперех по осі Z
- кут повороту по осі X

всі чотири величини мають діапазон значень від -100 до 100

12. Ці операції виконуються в циклі. В цей час система очікує натиснення клавіш керування R, T, L які описані в розділі 3.1.1

13. Також відбувається вивід наступної інформації на екран:

- Якщо нажати кнопку R, на екрані з'являється напис червоним кольором Recording, що свідчить про те, що йде запис відео. Якщо повторно натиснути клавішу R, даний запис зникне і запис зупиниться

- Температура дрону
- Заряд дрону
- FPS

14. Якщо закрити основне вікно програми, робота системи зупиниться.

3.7.1 Опис роботи класу FilterKalman

Клас FilterKalman реалізує фільтр Калмана для обробки та згладжування вимірювань. Фільтр Калмана є рекурсивним методом, який використовується для оцінки невизначених параметрів системи (наприклад, координат) на основі шумних і неповних вимірювань. В даному випадку клас має два основні методи: ініціалізацію фільтра та застосування фільтра до нових даних.

Основна логіка

Ініціалізація фільтра Калмана. Метод kalman_init() ініціалізує фільтр Калмана з певними початковими параметрами:

- kf.x — початковий стан системи, у даному випадку це масив з двох значень: [позиція X, позиція Y], ініціалізований як [0, 0].
- kf.F — матриця переходу стану, яка описує, як змінюється стан системи між двома часовими кроками. У цьому випадку вона є одиничною матрицею 2x2, що означає, що система не змінюється за кожен крок.
- kf.H — матриця вимірювань, що перетворює вимірювання в оцінки стану. Також одинична матриця, що вказує на те, що вимірювання прямо відповідають стану.
- kf.P — матриця коваріації невизначеності стану. Ініціалізована значенням 1000 для того, щоб показати високу невизначеність на початку.
- kf.R — матриця коваріації шуму вимірювань. Це показник точності вимірювань. У даному випадку шум вимірювань має одиничне значення.

- kf.Q — матриця коваріації шуму процесу. Вона описує невизначеність, пов'язану з самою динамікою системи. У цьому випадку її значення є малим (0.1).

Фільтр ініціалізується і повертається з цією конфігурацією для подальшого використання в обробці вимірювань.

Застосування фільтра Калмана:

Метод apply_kalman_filter() застосовує фільтр до нових вимірювань (положення X та Y), щоб отримати фільтровані значення.

measurements — масив вимірювань, що містить два значення: положення X та Y.

Далі виконуються два основні етапи фільтрації:

Прогнозування (prediction): Використовується метод kf.predict(), щоб передбачити наступний стан системи на основі попередніх даних.

Оновлення (update): Використовується метод kf.update(measurements) для коригування передбачення з урахуванням нових вимірювань.

Після цих етапів фільтр оновлює свій стан, і в результаті отримуються фільтровані значення:

- filtered_data — новий оцінений стан, де kf.x[0] і kf.x[1] — це нові фільтровані координати X і Y.
- filtered_outputX і filtered_outputY — це округлені до цілих чисел значення координат X і Y, які повертаються як результат роботи методу.

3.7.2 Опис роботи класу PIDController

Клас PIDController реалізує ПІД-регулятор.

Логіка роботи класу:

Ініціалізація параметрів та змінних:

У конструкторі класу __init__() задаються три основні параметри ПІД-регулятора:

- Kp — коефіцієнт пропорційної частини.
- Ki — коефіцієнт інтегральної частини.

- Kd — коефіцієнт диференційної частини.

Ці параметри можна налаштувати при створенні об'єкта класу, або ж використовуються значення за замовчуванням.

Метод init_parameters() ініціалізує початкові значення для різних змінних:

- current_errorX та current_errorY — поточні помилки по осях X та Y (відстань між поточною позицією і бажаною).
- outputX, outputY та outputZ — вихідні значення для коригування руху по осях X, Y та Z.

Обчислення виходів ПІД - регулятора:

Метод PID_controller() реалізує сам алгоритм ПІД-регулятора:

- current_error — обчислюється за допомогою методу get_error(). Це різниця між поточною позицією (значення center_pixel) та бажаною позицією (центр екрану).
- Після цього обчислюються три складові ПІД -регулятора
- Потім усі три компоненти сумуються, і обчислюється вихід регулятора output, який визначає коригування.

Обчислення помилки

Метод get_error() обчислює помилку між поточною позицією (виміряною на сенсорі) та бажаною позицією (центром екрану).Помилка обчислюється як різниця між центром екрану (який є бажаним положенням) та поточною позицією. Якщо довжина не дорівнює нулю, то помилка обчислюється як різниця між половиною довжини та поточною позицією.

Визначення дій дрону

Метод define_actions() приймає виходи нейронної мережі (NN_outputs) і визначає дії для управління дроном по осях:

- outputX — коригує рух по осі X
- outputY — коригує рух по осі Y
- outputZ — коригує швидкість по осі Z.

- yaw_velocity — визначає кутову швидкість (обертання) на основі outputX.
- Всі значення коригуються за допомогою мінімальних обмежень (максимум 100).

3.7.3 Опис роботи класу NNController

Клас NNController реалізує нейронну мережу для керування, яка складається з трьох шарів, і використовується для прогнозування та коригування значень, які можуть бути застосовані до системи. Цей клас поєднує традиційні методи регулювання з можливістю використання нейронної мережі для покращення навігації.

Ініціалізація:

У методі __init__() ініціалізується структура нейронної мережі:

- fc1 — перший вхідний шар мережі з 3 входами та 64 виходами.
- fc2 — другий прихований шар з 64 входами та 64 виходами.
- fc3 — вихідний шар з 64 входами та 3 виходами (ймовірно, для керування трьома параметрами: X, Y, Z).
- optimizer — оптимізатор Adam, який використовує параметри мережі та вчить її на основі функції втрат.
- criterion — функція втрат MSELoss (Mean Squared Error), яка оцінює точність прогнозу нейронної мережі порівняно з реальними значеннями.

Тренування нейронної мережі

Метод train_network(target, outputs) здійснює тренування мережі:

- Використовує функцію втрат MSELoss, щоб оцінити відмінність між прогнозами мережі (виходами) та цільовими значеннями.
- Обчислює градієнти за допомогою backpropagation та оновлює ваги мережі за допомогою оптимізатора Adam.

Прогнозування

Метод predict_values(outputX, outputY, outputZ) створює тензор на основі вхідних значень outputX, outputY та outputZ, які є вхідними параметрами для нейронної мережі.

Застосування нейронної мережі

Метод apply_NN(neural_network, outputZ, filtered_outputX, filtered_outputY, outputs, NN) використовується для прогнозування виходів за допомогою нейронної мережі:

- Якщо NN == True, то виходи від ПІД-регулятора використовуються як ціль для тренування нейронної мережі.
- Використовує метод get_outputs() для прогнозування нових значень.
- Тренує нейронну мережу, використовуючи отримані виходи ПІД-регулятора як вхід і виходи нейронної мережі.
- Після тренування нейронна мережа надає нові коригування, які використовуються для управління системою.

Основні компоненти нейронної мережі в класі:

Клас використовує три повнозв'язних шари:

- Вхідний шар (fc1) з 3 входами.
- Прихований шар (fc2) з 64 виходами.
- Вихідний шар (fc3) з 3 виходами.

Для обробки даних використовується функція активації ReLU між шарами.

3.7.4 Опис роботи класу MediapipeBodyModule

У методі __init__() ініціалізуються важливі компоненти:

- mp_drawing — для малювання на зображенні розпізнаних ключових точок.
- mp_pose — використовує MediaPipe для розпізнавання пози людини на зображеннях.
- results — змінна для збереження результатів розпізнавання (типу PoseLandmarkerResult).

Метод draw_landmarks_on_image(self, rgb_image, detection_result) обробляє зображення, на якому розпізнаються ключові точки людського тіла (наприклад, лікті, коліна, плечі та інші).

Кроки роботи методу:

- Отримання координат точок: За допомогою detection_result.pose_landmarks отримуються координати точок, які визначає MediaPipe.
- Обчислення середнього центру тіла: Визначаються середні координати (по X і Y) для всіх ключових точок тіла. Ці координати потім перетворюються в пікселі відповідно до розміру зображення.
- Малювання точок на зображенні: Використовується mp_drawing.draw_landmarks(), щоб накласти розпізнані ключові точки на зображення. Це дозволяє візуалізувати результати розпізнавання.

Результатом виконання методу є зображення з накладеними точками та координатами центральної точки тіла (центр тіла у пікселях).

- target_x_pixel і target_y_pixel — це координати центральної точки тіла в пікселях на зображенні. Вони обчислюються як середні значення координат усіх ключових точок на зображенні.

Метод print_result(self, result: PoseLandmarkerResult, output_image: mp.Image, timestamp_ms: int) зберігає результат розпізнавання у змінній results. Це дозволяє зберігати результати обробки для подальшого використання чи аналізу.

3.7.5 Опис роботи класу DroneController

Клас DroneController є частиною Device layer (шару пристрою) і відповідає за ініціалізацію, отримання інформації та керування дроном DJI Tello через відповідні API та методи.

Метод drone_init

Цей метод ініціалізує підключення до дрону. Спочатку створюється об'єкт Tello(), який представляє дрон, після чого встановлюється з'єднання через метод connect(), що забезпечує підключення до дрону через Wi-Fi. Потім запускається потік відео за допомогою методу streamon(), щоб дрон почав передавати відео в реальному часі. Метод повертає підключений об'єкт дрону, щоб з ним можна було працювати далі.

Метод drone_get_info

Цей метод отримує важливу інформацію про дрон, зокрема температуру і рівень заряду батареї.

- temperature_f — отримує температуру дрону в градусах Фаренгейта через метод get_temperature(). Потім ця температура конвертується в градуси Цельсія за допомогою формули та округляється до одного знака після коми.
- battery — отримує рівень заряду батареї через метод get_battery(). Метод повертає значення температури в градусах Цельсія та рівень заряду батареї.

Метод drone_control Цей метод відповідає за управління дроном, приймаючи параметри drone (об'єкт дрону) та action (дія, яку потрібно виконати).

- У методі визначено словник actions, в якому ключі — це назви можливих дій (наприклад, 'takeoff', 'land', 'streamoff'), а значення — це відповідні методи для виконання цих дій.
- Якщо вхідна дія знаходиться в словнику actions, то відповідний метод виконується в окремому потоці, щоб не блокувати основний процес.
- Якщо дія є 'land' або 'takeoff', після виконання цих дій дрону відправляється команда send_rc_control(0, 0, 0, 0), щоб зупинити рухи дрону та стабілізувати його.

Таким чином, клас DroneController дозволяє налаштувати, контролювати та отримувати інформацію від дрону, а також виконувати основні дії, такі як зліт, посадка та вимкнення відео потоку.

3.7.6 Опис роботи класу CreatePlot

Клас CreatePlot є частиною GUI layer (шару графічного інтерфейсу користувача) і відповідає за побудову графіка на основі двох масивів значень — x_values та y_values.

Метод plot_graph створює та відображає графік на основі вхідних масивів значень x_values та y_values.

Метод виконує наступні дії:

- Перевірка на порожні масиви: Метод спочатку перевіряє, чи не є масиви x_values або y_values порожніми. Якщо хоча б один з масивів порожній, виводиться повідомлення про помилку, і графік не будується.
- Створення індексів для осі X: Оскільки для побудови графіка потрібно визначити точки на осі X, створюється список індексів indices, який представляє числові індекси елементів масивів x_values та y_values. Ці індекси використовуються як значення на осі X.

Використовується бібліотека matplotlib.pyplot для побудови графіка. Викликається метод plt.figure(), щоб налаштувати розмір графіка. Далі, за допомогою методу plt.plot(), на графіку малюються дві лінії:

- Одна для x_values (значення по осі X).
- Інша для y_values (значення по осі Y).
- Для кожної лінії використовуються маркери 'o' для позначення точок та налаштовується їх розмір за допомогою markersize.

Налаштування осей та заголовка:

- Метод plt.title() задає заголовок графіка.
- plt.xlabel() і plt.ylabel() задають підписи до осей X та Y відповідно.

- Використовується plt.legend() для відображення легенди, що позначає лінії графіка (для x_values та y_values).
- За допомогою plt.grid(True) додається сітка на графік, щоб полегшити читання даних.

Відображення графіка: Наприкінці викликається метод plt.show(block=True), що відображає графік у вікні користувача.

3.7.7 Опис роботи класу UserInterface

Клас UserInterface є частиною GUI layer (шару графічного інтерфейсу користувача) і призначений для взаємодії з користувачем через клавіатуру, обробки подій та управління діями дрону. Основною метою цього класу є забезпечення можливості користувачу контролювати політ дрону та налаштування запису відео через інтерфейс.

Конструктор init

Конструктор ініціалізує два атрибути:

is_running — булева змінна, яка вказує, чи працює програма (за замовчуванням True).

is_flying — булева змінна, що вказує, чи знаходиться дрон у повітрі (за замовчуванням False).

Метод key_action

Цей метод обробляє події клавіатури та змінює стан системи на основі натиснутих клавіш. Параметри, що передаються в метод:

is_flying — стан польоту дрону (чи в повітрі).

is_running — стан роботи програми.

use_live_stream — булева змінна, що вказує, чи активовано потокову трансляцію відео.

save_video — булева змінна, що вказує, чи активовано запис відео.

video_writer — об'єкт для запису відео.

fps — кількість кадрів на секунду для відеозапису.

width і height — розміри відео.

Алгоритм роботи методу:

Обробка події закриття вікна (QUIT):

Якщо натиснута кнопка закриття вікна (вікно програми закривається), то:

- Якщо активна пряма трансляція та дрон у повітрі, зупиняється зйомка та вимикається відео потік (action = 'land', 'streamoff').
- В іншому випадку просто зупиняється відео потік (action = 'streamoff').
- Програма припиняє свою роботу (is_running = False).

Обробка натискання клавіш (KEYDOWN):

Якщо натиснута клавіша R:

- починається або припиняється запис відео.
- Якщо запис активовано (save_video = True), створюється об'єкт cv2.VideoWriter для збереження відео у файл 'video.mp4'.
- Якщо запис вимкнуто, відео припиняється, і файл запису закривається.
- дрон злітає, якщо він ще не в повітрі (is_flying = False), викликається дія зліту 'takeoff'.
- дрон приземляється, якщо він уже в повітрі (is_flying = True), викликається дія посадки 'land'.

Повернення значень: Метод повертає актуалізовані значення:

is_flying — чи знаходиться дрон у повітрі.

is_running — чи працює програма.

save_video — чи ведеться запис відео.

video_writer — об'єкт відео запису.

action — виконана дія (зліт, посадка, зупинка потоку відео, початок чи зупинка запису відео).

3.8 Розрахунок коефіцієнтів ПІД - регулятора

Для визначення коефіцієнтів ПІД-регулятора для дрона DJI Tello використаємо метод Циглера - Нікольса. Для цього запустимо дрон в режимі П - регулятора і підберемо коефіцієнт посилення K_u та період коливання T_u, при якому вихідний сигнал починає постійно коливатися:

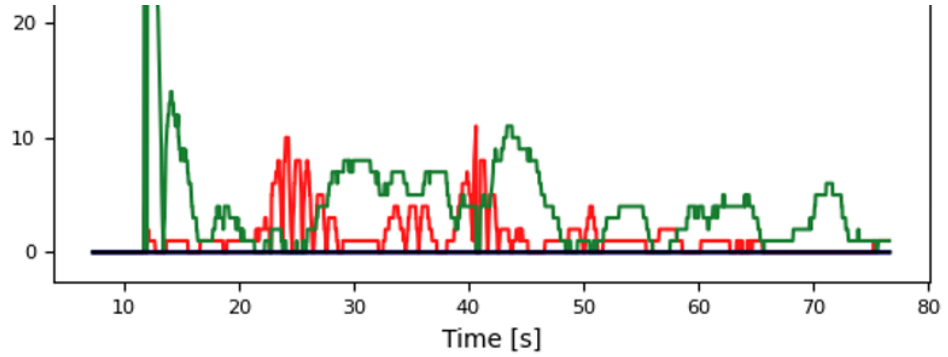

Рисунок 3.6 - Графік помилки по осям X та Y(розмірність: вісь X, ітерація; вісь Y, величина помилки у пікселях)

$K_u = 0.272$

$T_u = 7$

Розрахуємо коефіцієнти ПІД - регулятора за формулами:

$K_p = 0.60 * 0.272 = 0.16$

$K_i = 1.2 * 0.272 / 7 = 0.05$

$K_d = 3 * 0.272 * 7 / 40 = 0.14$

3.9 Порівняння роботи системи з ПІД - регулятором та ПІДНМ регулятором

Порівняємо наступні тест сценарії:

- тестування злету дрону, порівняння значень виходів ПІД- та ПІДНМ- регуляторів

- тестування повороту дрону по горизонталі
- тестування розпізнавання цілі в умовах поганого освітлення

На графіках вісь X має синій колір, вісь Y має помаранчевий колір.

3.9.1 Тестування злету дрону і руху до цілі

Порівняємо роботу системи для завдання зльоту і розпізнавання цілі на відстані двох метрів.

На рисунку 3.5 зображено виходи ПІД - регулятора для злету дрону (основний рух по осі Y):

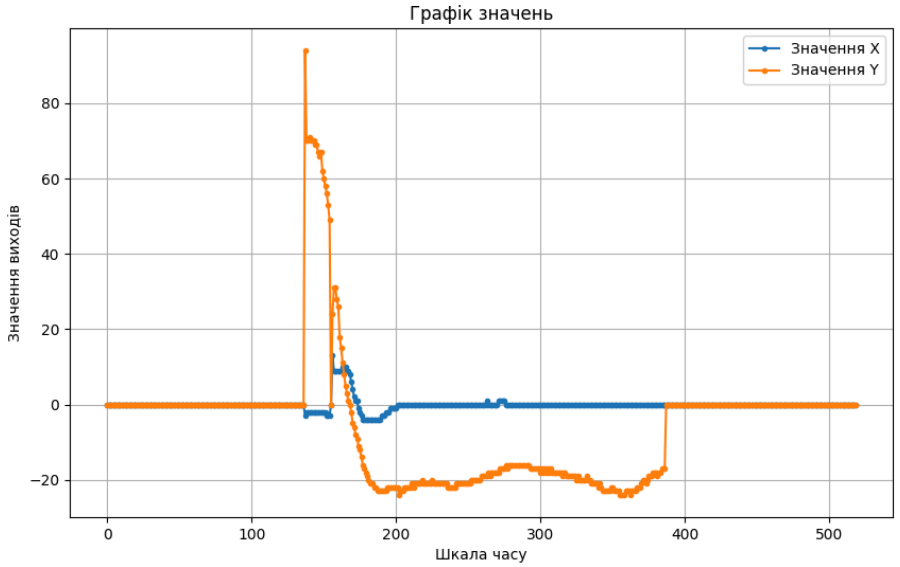

Рисунок 3.7 - Вихід в режимі ПІД - регулятора (розмірність: вісь X, с; вісь Y, см)

Як бачимо з графіка, значення виходу стрибає до величини близької 100, потім спускається до -20, і використовує це значення виходу Y для руху до цілі.

На рисунку 3.6 зображено виходи ПІДНМ - регулятора для злету дрону (основний рух по осі Y):

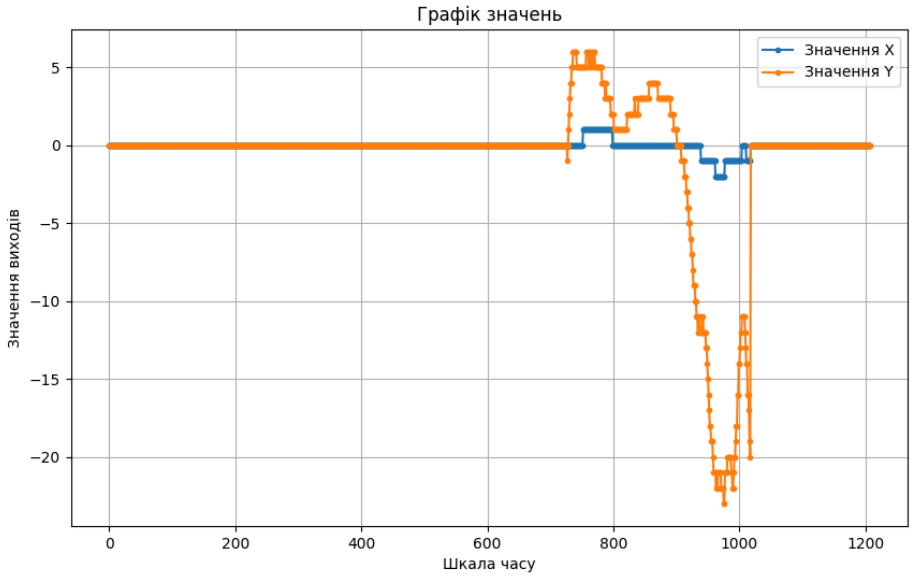

Рисунок 3.8 - Вихід в режимі ПІДНМ - регулятора (розмірність: вісь X, с; вісь Y, см)

Як бачимо з даного графіка, значення виходу регулятора стрибає до величини близької 6, потім спускається до -25, і використовує це значення виходу Y для руху до цілі.

3.9.2 Тестування злету дрону і поворотів в горизонтальній площині

На рисунку 3.7 зображено виходи ПІДНМ - регулятора для злету дрону і поворотів в горизонтальній площині (основний рух по осі X):

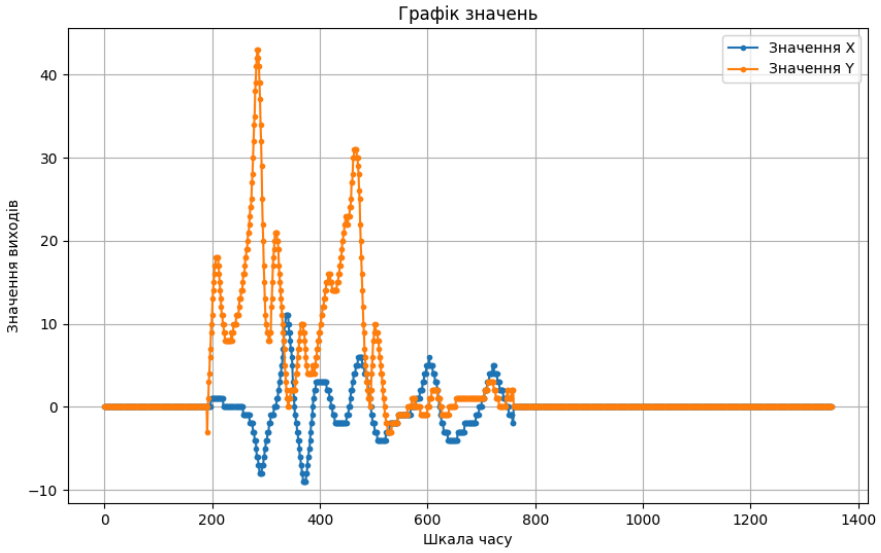

Рисунок 3.9 - Горизонтальний рух в режимі ПІДНМ - регулятора (розмірність: вісь X, с; вісь Y, см)

Як бачимо з графіка, значення X коливаються в межах від 10 до -10 на початку руху, яке зменшується до діапазону від 6 до -4.

На рисунку 3.8 зображено виходи ПІД - регулятора для злету дрону і поворотів в горизонтальній площині (основний рух по осі X):

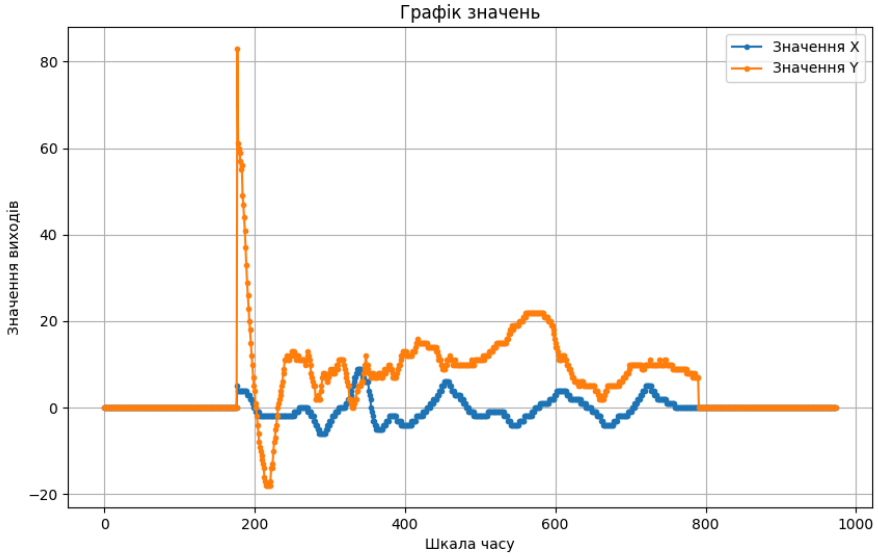

Рисунок 3.10 - Горизонтальний рух в режимі ПІД - регулятора (розмірність: вісь X, с; вісь Y, см)

Як бачимо з графіка, значення X коливаються в межах від 10 до -8 на початку руху, яке зменшується до діапазону від 6 до -4.

3.9.3 Тестування розпізнавання цілі в умовах поганого освітлення

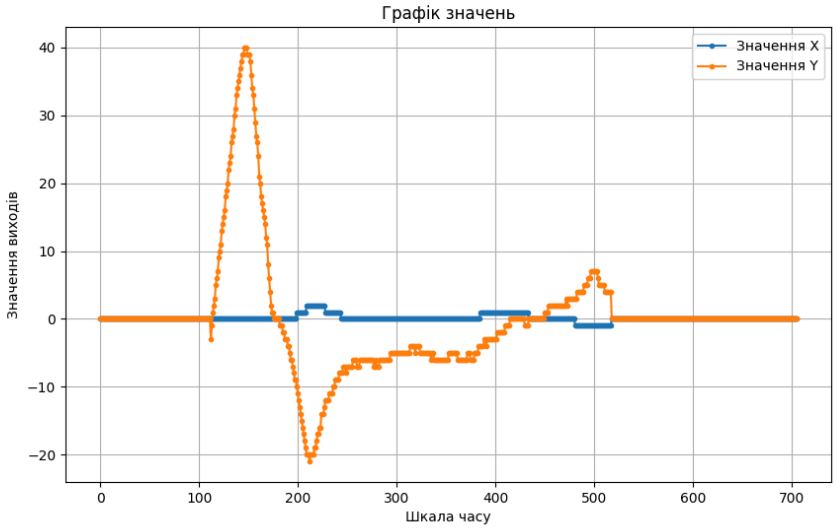

Рисунок 3.11 - Розпізнавання образу в режимі ПІДНМ - регулятора (розмірність: вісь X, с; вісь Y, см)

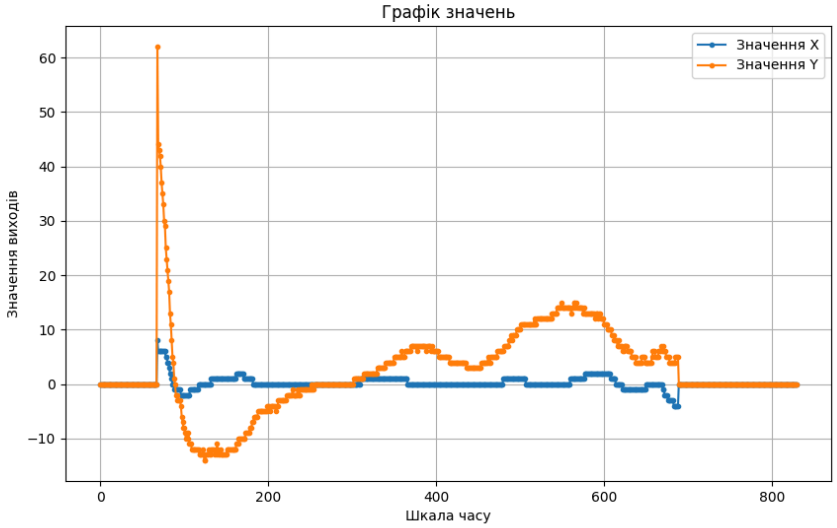

Рисунок 3.12 - Розпізнавання образу в режимі ПІД - регулятора (розмірність: вісь X, с; вісь Y, см)

3.9.4 Зведені дані

В таблиці 3.1 порівняємо результати попередніх досліджень:

Таблиця 3.1 – порівняння результатів досліджень

Сценарій	Діапазон значень виходу см	Кількість ітерацій
Тестовий сценарій 1 (вісь Y)	ПІД - від 100 до - 20 ПІДНМ - від 6 до - 25	ПІД - 250 ПІДНМ - 250
Тестовий сценарій 2 (вісь X)	ПІД - від 10 до -10 ПІДНМ - від 10 до - 10	ПІД - 620 ПІДНМ - 580
Тестовий сценарій 3 (недостатнє освітлення)	ПІД - від 63 до -15 ПІДНМ - від 40 до -20	ПІД - 600 ПІДНМ - 400

З даного порівняння можна зробити висновок, що системі під управлінням ПІДНМ необхідно менше ітерацій для стабілізації роботи системи і амплітуда значень графіків виходів регулятора є меншою у ПІДНМ регулятора ніж у ПІД - регулятора

3.9.5 Юніт тести

У системі написані юніт тести для методів які виконують калькуляцію:

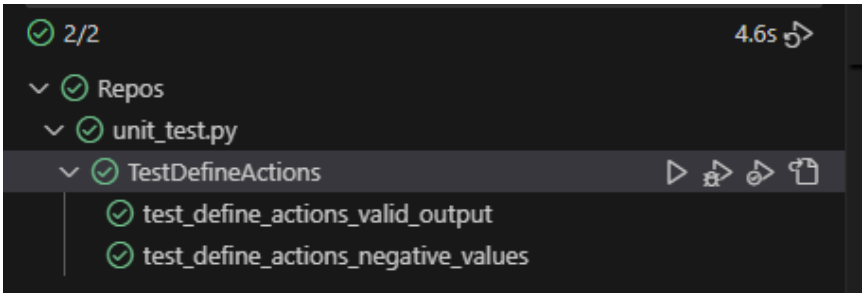

Рисунок 3.13 - Юніт тести

Для тестування використовується бібліотека unittest

Основною вимогою до юніт тестів були наступні критерії:

- Один юніт тест має покривати один метод чи частину одного метода
- Висока швидкодія має бути пріоритетом

- Юніт тест не має працювати з файлами, а лише перевіряти логіку роботи коду

3.9.6 Тестування розпізнавання образу

Тестування розпізнавання людини виконувалось з камери ноутбука і камери дрона. В обох випадках розпізнавання проходить майже відразу і без затримок. Приклад розпізнаного образу зображено на рисунку 3.12:

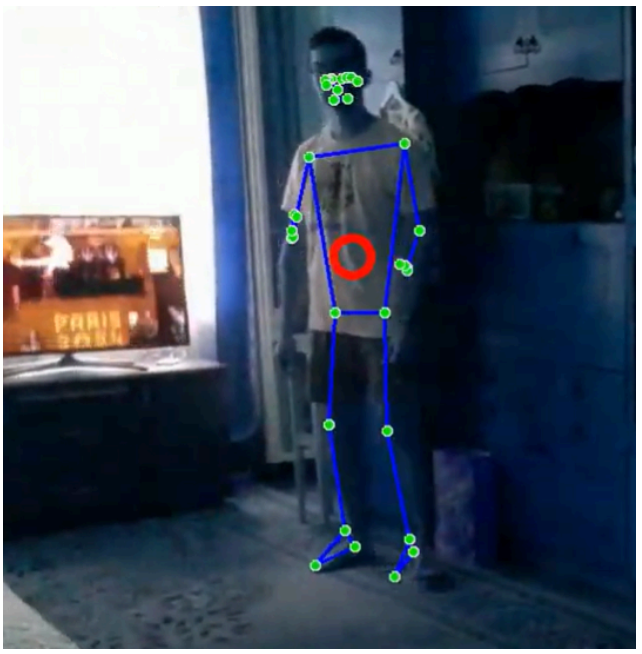

Рисунок 3.14 - Розпізнаний за допомогою алгоритму образ

Як можна побачити на рисунку 3.12, синім та зеленим кольором позначені частини розпізнаного образу тіла людини. Червоним колом позначено центр образу, який буде використовуватися системою як ціль.

Висновки по розділу:

У цьому розділі була представлена програмно-технічна реалізація системи керування процесом для автономної навігації дрону, що включає детальну розробку архітектури системи, її компонентів та алгоритмів. Архітектура системи включає кілька важливих шарів, таких як Device, що відповідає за апаратну частину, та Core, де реалізуються основні функції

системи. Також було розглянуто інтерфейс користувача, що дозволяє взаємодіяти з системою, і схему алгоритму роботи, що описує основні етапи роботи програмного забезпечення. Логіка роботи основних класів, таких як FilterKalman, PIDController, NNController, та MediapipeBodyModule, була детально описана, що дозволяє зрозуміти, як кожен компонент взаємодіє в загальній системі.

Тестування системи було важливим етапом для оцінки її ефективності. Було проведено порівняння роботи системи з ПІД-регулятором та ПІДНМ-регулятором, що показало покращення результатів в задачах злету дрону, руху до цілі та виконання поворотів. Крім того, було виконано юніт тестування для перевірки коректності роботи окремих компонентів, а також тестування розпізнавання образу, що дозволило переконатися в ефективності застосованих технологій.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

У результаті проведеного дослідження було спроектовано і створено систему інтелектуального керування дроном на основі сучасних технологій машинного зору та нейронних мереж. Зокрема, було використано фізичну модель дрона типу DJI Tello, вивчено методи розпізнавання образів за допомогою бібліотек mediapipe та реалізовано ПІД регулятор з використанням нейронних мереж. Зокрема, було зроблено підбір параметрів ПІД-регулятора, що дозволяє досягти стабільності та високої точності керування. Використання методу Циглера-Нікольса для налаштування регулятора та застосування фільтра Калмана для покращення точності оцінки стану системи в умовах шуму і невизначеності дозволяє підвищити надійність і ефективність системи керування. Було розроблено простий і зручний інтерфейс користувача. Тестування різних варіантів регуляторів (з нейронною мережею та без) показало, що інтеграція нейронних мереж у систему керування покращує її адаптивність і дозволяє дрону виконувати завдання з високою швидкістю та точністю, що є важливим для успішного виконання завдань без втручання оператора. Це дозволяє дрону адаптуватися до змін у навколишньому середовищі та автоматично приймати оптимальні рішення в реальному часі. Практична цінність розробленої системи полягає в можливості її впровадження для автономних дронів, здатних виконувати складні місії в умовах, де критично важлива швидкість реакції, точність та висока ймовірність виконання завдання.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Russian Tactics in the Second Year of Its Invasion of Ukraine URL: https://static.rusi.org/403-SR-Russian-Tactics-web-final.pdf (дата звернення: 07.12.2024)

2. FPV. techtarget.com. URL: https://www.techtarget.com/whatis/definition/FPV-drone-first-person-view-drone (дата звернення: 07.12.2024)

3. Квадрокоптер DJI Tello. quadro.ua. URL: https://quadro.ua (дата звернення: 07.12.2024)

4. Машинний зір. URL: https://visionplatform.ai/computer-vision-for-drones-and-uav-in-2024/ (дата звернення: 07.12.2024).

5. Штучний інтелект у дронах. hashdork.com. URL: https://hashdork.com/uk/artificial-intelligence-in-drones/ (дата звернення: 07.12.2024)(

6. Штучна нейронна мережа. Wikipedia.org. URL: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/ (дата звернення: 07.12.2024)

7. Електронний ресурс. A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. URL: https://www.researchgate.net/publication/336805909_A_High-Accuracy_Model_Average_Ensemble_of_Convolutional_Neural_Networks_for_Classification_of_Cloud_Image_Patches_on_Small_Datasets#pf3 (дата звернення: 07.12.2024)

8. ПІД-регулятор. URL: https://dewesoft.com/blog/what-is-pid-controller (дата звернення: 07.12.2024)

9. PIDNN. URL: https://molefrog.com/etc/pid-neural-network/#d-neuron (дата звернення: 07.12.2024)

10. Ziegler–Nichols method. URL: https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method (дата звернення: 07.12.2024)

11. Порівняння методів підбору параметрів ПІД - регулятора. URL: https://www.wevolver.com/article/mastering-pid-tuning-the-comprehensive-guide (дата звернення: 07.12.2024)

12. Фільтр Калмана. URL: https://www.kalmanfilter.net/default.aspx (дата звернення: 07.12.2024)

13. Бібліотека MediaPipe solutions guide. URL: https://ai.google.dev/edge/mediapipe/solutions/guide (дата звернення: 07.12.2024)

14. Бібліотека MediaPipe vision: pose landmarker. URL: https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker (дата звернення: 07.12.2024)

15. Бібліотека Pygame. Wikipedia.org. URL: https://uk.wikipedia.org/wiki/Pygame (дата звернення: 07.12.2024)

16. Бібліотека OpenCV. Wikipedia.org. URL: https://uk.wikipedia.org/wiki/OpenCV (дата звернення: 07.12.2024)

17. Бібліотека Torch. Wikipedia.org. URL: https://uk.wikipedia.org/wiki/Torch (дата звернення: 07.12.2024)

18. Бібліотека unittest. URL: https://docs.python.org/3/library/unittest.html (дата звернення: 07.12.2024)

19. Бібліотека matplot. URL: https://matplotlib.org/ (дата звернення: 07.12.2024)

20. Модульне програмування. URL: https://en.wikipedia.org/wiki/Modular_programming (дата звернення: 07.12.2024)

21. MediaPipe Pose Landmarker. URL: https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker/android (дата звернення: 07.12.2024)

22. ADAM. URL: https://medium.com/@nerdjock/deep-learning-course-lesson-7-4-adam-adaptive-moment-estimation-e23434850bfc (дата звернення: 07.12.2024)

23. MSELoss. URL: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html (дата звернення: 07.12.2024)

24. Архітектурні патерни проектування. URL: https://refactoring.guru/uk/design-patterns/catalog (дата звернення: 07.12.2024)

25. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.

26. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

27. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

28. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові.

29. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

30. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).

ДОДАТОК А

Основний файл drone.py

```
import pygame
import cv2
import mediapipe as mp
from GUI import Drawer, ScreenHandler, UserInterface, CreatePlot
from device import DroneController
from core import PIDController, NNController, FilterKalman,
MediapipeBodyModule

DRONE_STREAM = True
MODEL_PATH = 'Lib/pose_landmarker_lite.task'

BaseOptions = mp.tasks.BaseOptions
PoseLandmarker = mp.tasks.vision.PoseLandmarker
PoseLandmarkerOptions = mp.tasks.vision.PoseLandmarkerOptions
PoseLandmarkerResult = mp.tasks.vision.PoseLandmarkerResult
VisionRunningMode = mp.tasks.vision.RunningMode

class SmartDroneNavigation():

    def __init__(self):
        self.actionX = 0
        self.actionY = 0
        self.actionZ = 0
        self.timestamp = 0
        self.temperature_c = 0
        self.battery = 0
        self.fps = 0
        self.is_running = True
        self.is_flying = False
        self.save_video = False

    def get_result(self, mediapipe_body, BaseOptions, PoseLandmarkerOptions,
VisionRunningMode):
        options = PoseLandmarkerOptions(
            base_options = BaseOptions(model_asset_path = MODEL_PATH),
```

```
            running_mode = VisionRunningMode.LIVE_STREAM,
            result_callback = mediapipe_body.print_result
        )

        return options

    def main(self):

        draw_plot = CreatePlot()
        pid_controller = PIDController()
        neural_network = NNController()
        drone_controller = DroneController()
        user_interface = UserInterface()
        screen_handler = ScreenHandler()
        mediapipe_body = MediapipeBodyModule()
        drawer_writer = Drawer()
        kalman_filter = FilterKalman()

        options = PoseLandmarkerOptions(
            base_options = BaseOptions(model_asset_path = MODEL_PATH),
            running_mode = VisionRunningMode.LIVE_STREAM,
            result_callback = mediapipe_body.print_result
        )

        if DRONE_STREAM:
            drone = drone_controller.drone_init()
            frame_read = drone.get_frame_read()
        else:
            frame_read = None

        pygame.init()
        kf = kalman_filter.kalman_init()
        video_source, screen_width, screen_height, fps =
screen_handler.get_screen_parameters(DRONE_STREAM)
        scr = pygame.display.set_mode((screen_width, screen_height))
        clock = pygame.time.Clock()
        current_errorX, current_errorY, outputX, outputY, outputZ =
pid_controller.init_parameters()
```

```
        video_writer = None
        array_X = []
        array_Y = []

        with PoseLandmarker.create_from_options(options) as landmarker:
            while self.is_running:
                if DRONE_STREAM:
                    self.is_flying, self.is_running, self.save_video, video_writer, action
= user_interface.key_action(self.is_flying, self.is_running, DRONE_STREAM,
self.save_video, video_writer, fps, screen_width, screen_height)
                    drone_controller.drone_control(drone, action)
                frame, ok = screen_handler.get_frame(frame_read, video_source,
DRONE_STREAM)

                if not ok:
                    break

                mp_image = mp.Image(image_format=mp.ImageFormat.SRGB,
data=frame)
                landmarker.detect_async(mp_image, self.timestamp)

                if mediapipe_body.results is not None:
                    annotated_image, target_x_pixel, target_y_pixel =
mediapipe_body.draw_landmarks_on_image(mp_image.numpy_view(),
mediapipe_body.results)
                    if target_x_pixel is not None and target_y_pixel is not None and
self.is_flying:

                        # Отримуємо поточні значення ПІД регулятора
                        current_errorX, outputX =
pid_controller.PID_controller(target_x_pixel, screen_width, current_errorX)
                        current_errorY, outputY =
pid_controller.PID_controller(target_y_pixel, screen_height, current_errorY)

                        # Розраховуємо швидкість руху вперед по осі Z
                        outputZ, outputs = pid_controller.calculate_speed(outputX,
outputY, target_x_pixel, target_y_pixel)
```

```
                    # Застосуємо фільтр Калмана
                    filtered_outputX, filtered_outputY =
kalman_filter.apply_kalman_filter(kf, outputX, outputY)

                    # Застосуємо нейронну мережу: True - PIDNN, False - PID
                    outputs = neural_network.apply_NN(neural_network, outputZ,
filtered_outputX, filtered_outputY, outputs, True)

                    # Отримання параметрів керування дроном
                    self.actionZ, self.actionY, self.actionX, self.yaw_velocity =
pid_controller.define_actions(outputs)

                    drawer_writer.DrawTarget(outputZ, annotated_image,
target_x_pixel, target_y_pixel)
                else:
                    self.actionZ = 0
                    self.actionY = 0
                    self.actionX = 0
                    self.yaw_velocity = 0
            else:
                annotated_image = frame
                self.yaw_velocity = 0
                self.outputY = 0
                self.outputX = 0

            if DRONE_STREAM:
                if self.is_flying:
                    drone.send_rc_control(self.actionX, self.actionZ, self.actionY,
self.yaw_velocity)

                if self.timestamp % fps == 0:
                    self.temperature_c, self.battery =
drone_controller.drone_get_info(drone)
                    self.fps = clock.get_fps()

                array_X.append(self.actionX)
                array_Y.append(self.actionY)
```

```
            annotated_image =
drawer_writer.display_drone_info(annotated_image, self.temperature_c,
self.battery, self.yaw_velocity, self.save_video, self.fps, DRONE_STREAM)
# Анотація на зображенні
            screen_handler.display_result(scr, annotated_image)

            pygame.display.flip()
            clock.tick(fps)

            self.timestamp += 1

            if self.save_video and video_writer is not None:
                video_frame = cv2.cvtColor(annotated_image,
cv2.COLOR_RGB2BGR)
                video_writer.write(video_frame)

        draw_plot.plot_graph(array_X, array_Y)      # Графік вихідних величин
        pygame.quit()

        if not DRONE_STREAM:
            video_source.release()

if __name__ == "__main__":
    body_module = SmartDroneNavigation()
    body_module.main()
```

ДОДАТОК Б

Файл core.py

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import mediapipe as mp
from filterpy.kalman import KalmanFilter
from mediapipe.framework.formats import landmark_pb2

class PIDController():

    def __init__(self, Kp=0.16, Ki=0.05, Kd=0.14):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd

    def init_parameters(self):
        current_errorX = 0  # поточна помилка X
        current_errorY = 0  # поточна помилка Y
        outputX = 0
        outputY = 0
        outputZ = 0
        return current_errorX, current_errorY,  outputX, outputY, outputZ

    def PID_controller(self, center_pixel, length, error):

        current_error = self.get_error(length, center_pixel)

        P = self.Kp * current_error                # Пропорційна частина регулятора
        I = self.Ki * (current_error + error)      # Інтегральна частина регулятора
        D = self.Kd * (current_error - error)      # Диференційна частина
регулятора

        output = int(P + I + D)                # Обчислюємо вихід ПІД регулятора
        return current_error, output

    def get_error(self, length, center_pixel):
```

```
        current_error = (length / 2 - center_pixel) if length != 0 else 5  # 10

        return current_error

    def define_actions(self, NN_outputs):

        outputX = int(round(NN_outputs[0].item()))
        outputY = int(round(NN_outputs[1].item()))
        outputZ = int(round(NN_outputs[2].item()))

        yaw_velocity = - outputX

        actionZ = outputZ * 10 if outputZ > 0 else 0

        actionY = outputY
        actionX = int(np.round(outputX/10,0))

        outputX = min(outputX, 100)
        outputY = min(outputY, 100)
        outputZ = min(outputZ, 100)
        yaw_velocity = min(yaw_velocity, 100)

        return actionZ, actionY, actionX, yaw_velocity

    def calculate_speed(self, outputX, outputY, target_x_pixel, target_y_pixel):
        outputZ = np.round((np.abs(target_x_pixel)+np.abs(target_y_pixel))/400,0)
# зменшуємо максимальну швикість в N разів
        outputs = torch.tensor([outputX, outputY, outputZ], dtype=torch.float32)
        return outputZ, outputs

class NNController(nn.Module):
    def __init__(self):
        super(NNController, self).__init__()
        # Просту повнозв'язну нейронну мережу з одним шаром
        self.fc1 = nn.Linear(3, 64)  # Вхідний шар
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 3)  # Вихідний шар
```

```
        self.optimizer = optim.Adam(self.parameters(), lr=0.001) # Оптимізатор для тренування
        self.criterion = nn.MSELoss()  # Функція втрат для тренування

    def forward(self, x):
        x = torch.relu(self.fc1(x))  # Використовуємо ReLU як активаційну функцію
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)  # Вихід
        return x

    def train_network(self, target, outputs):
        loss = self.criterion(outputs, target) # Ми намагаємося зменшити різницю між прогнозами та реальними помилками
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

    def predict_values(self, outputX, outputY, outputZ):
        outputs = torch.tensor([outputX, outputY, outputZ], dtype=torch.float32)
        return outputs

    def get_outputs(self, outputX, outputY, outputZ, NN):
        if NN == True:
            NN_outputs = self.predict_values(outputX, outputY, outputZ)
            outputs = self.forward(NN_outputs)  # Нейронна мережа прогнозує значення
        else:
            outputs = torch.tensor([outputX, outputY, outputZ], dtype=torch.float32)
        return outputs

    def apply_NN(self, neural_network, outputZ, filtered_outputX, filtered_outputY, outputs, NN):
        if NN == True:
            outputs_from_PID = outputs

            # Використовуємо нейронну мережу для прогнозування виходів
```

```
            outputs_from_NN = neural_network.get_outputs(filtered_outputX,
filtered_outputY, outputZ, NN)

            # Тренування нейронної мережі в процесі роботи
            neural_network.train_network(outputs_from_PID, outputs_from_NN)

            outputs = outputs_from_NN
        return outputs

class MediapipeBodyModule():

    PoseLandmarkerResult = mp.tasks.vision.PoseLandmarkerResult

    def __init__(self):
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_pose = mp.solutions.pose
        self.results = None

    def draw_landmarks_on_image(self, rgb_image, detection_result):
        pose_landmarks_list = detection_result.pose_landmarks
        annotated_image = np.copy(rgb_image)

        # Ініціалізація значень для центрів
        target_x_pixel = None
        target_y_pixel = None

        for idx in range(len(pose_landmarks_list)):
            pose_landmarks = pose_landmarks_list[idx]
            landmarks = [(landmark.x, landmark.y, landmark.z) for landmark in
pose_landmarks]

            if not landmarks:
                continue
            x_coords = [landmark[0] for landmark in landmarks]
            y_coords = [landmark[1] for landmark in landmarks]
            #z_coords = [landmark[2] for landmark in landmarks]

            target_x = np.mean(x_coords)
```

```
            target_y = np.mean(y_coords)

            image_height, image_width, _ = rgb_image.shape
            target_x_pixel = int(target_x * image_width)
            target_y_pixel = int(target_y * image_height)

            pose_landmarks_proto = landmark_pb2.NormalizedLandmarkList()

pose_landmarks_proto.landmark.extend([landmark_pb2.NormalizedLandmark(
x=landmark.x, y=landmark.y, z=landmark.z) for landmark in pose_landmarks])
            self.mp_drawing.draw_landmarks(
               annotated_image,
               pose_landmarks_proto,
               self.mp_pose.POSE_CONNECTIONS
            )

        return annotated_image, target_x_pixel, target_y_pixel

    def print_result(self, result: PoseLandmarkerResult, output_image: mp.Image,
timestamp_ms: int): # type: ignore
        self.results = result

class FilterKalman():

    def kalman_init(self):

        # Ініціалізація фільтра Калмана
        kf = KalmanFilter(dim_x=2, dim_z=2)
        kf.x = np.array([0, 0])  # Початковий стан: [позиція X, позиція Y]
        kf.F = np.array([[1, 0], [0, 1]])  # Матриця переходу стану (простий
випадок)
        kf.H = np.array([[1, 0], [0, 1]])  # Матриця вимірювань
        kf.P *= 1000  # Початкова невизначеність
        kf.R = np.array([[1, 0], [0, 1]])  # Коваріація шуму вимірювань
        kf.Q = np.array([[0.1, 0], [0, 0.1]])  # Шум процесу
        return kf

    def apply_kalman_filter(self, kf, outputX, outputY):
```

```
        # Створюємо вектор вимірювань
        measurements = np.array([outputX, outputY])

        # 1. Прогноз
        kf.predict()

        # 2. Оновлення
        kf.update(measurements)

        # Фільтровані значення
        filtered_data = kf.x

        filtered_outputX = int(np.round(filtered_data[0],0))
        filtered_outputY = int(np.round(filtered_data[1],0))
        #print(f"Фільтровані значення: X = {filtered_outputX}, Y =
{filtered_outputY}")
        return filtered_outputX, filtered_outputY
```

ДОДАТОК В

Файл device.py

```
import threading
from djitellopy import Tello

class DroneController():    # Device layer

    def drone_init(self):
        drone = Tello()
        drone.connect()
        drone.streamon()
        return drone

    def drone_get_info(self, drone):
        temperature_f = drone.get_temperature()
        temperature_c = round((temperature_f - 32) * 5 / 9, 1)
        battery = drone.get_battery()
        return temperature_c, battery

    def drone_control(self, drone, action):
        actions = {
            'takeoff': drone.takeoff,
            'land': drone.land,
            'streamoff': drone.streamoff
        }

        if action in actions:
            threading.Thread(target=actions[action]).start()

        if action == 'land' or action == 'takeoff':
            drone.send_rc_control(0, 0, 0, 0)
```

ДОДАТОК Г

Файл GUI.py

```
import cv2
import pygame
import numpy as np
import matplotlib.pyplot as plt
import numpy as np

SMALL_SIZE = 8
MEDIUM_SIZE = 10
BIGGER_SIZE = 12

RED_COLOR = (255, 0, 0)
GREEN_COLOR = (0, 255, 0)
YELLOW_COLOR = (255, 255, 0)
BLUE_COLOR = (0, 0, 255)
FONT = cv2.FONT_HERSHEY_SIMPLEX

class Writer:
    def __init__(self):
        pass

    @staticmethod
    def save_video(image):
        cv2.putText(image, f'Recording On', (10, 30), FONT, 0.7, RED_COLOR,
2, cv2.LINE_AA)

    @staticmethod
    def velocity(image, velocity):
        cv2.putText(image, f'Velocity: {velocity}', (10, 60), FONT, 0.7,
GREEN_COLOR, 2, cv2.LINE_AA)

    @staticmethod
    def temperature(image, temperature):
        cv2.putText(image, f'Drone temperature: {temperature} C', (10, 90), FONT,
0.7, GREEN_COLOR, 2, cv2.LINE_AA)

    @staticmethod
```

```
    def battery(image, battery):
        cv2.putText(image, f'Battery: {battery} %', (10, 120), FONT, 0.7,
GREEN_COLOR, 2, cv2.LINE_AA)

    @staticmethod
    def fps(image, fps):
        cv2.putText(image, f'FPS: {fps:.2f}', (10, 150), FONT, 0.7,
GREEN_COLOR, 2, cv2.LINE_AA)

    @staticmethod
    def distance(annotated_image, distance, center_x_pixel, center_y_pixel):
        distance_text = f"Distance: {distance:.2f} meters"  # Відстань в метрах
        cv2.putText(annotated_image, distance_text, (center_x_pixel,
center_y_pixel - 40), FONT, 0.5, YELLOW_COLOR, 2, cv2.LINE_AA)

    @staticmethod
    def circle(annotated_image, center_x_pixel, center_y_pixel):
        # Параметри для знака плюс
        line_length = 30
        thickness = 2

        # Малювання горизонтальної лінії знака прицілa
        cv2.line(annotated_image,
                 (center_x_pixel - line_length, center_y_pixel),
                 (center_x_pixel + line_length, center_y_pixel),
                 RED_COLOR, thickness)

        # Малювання вертикальної лінії знака прицілa
        cv2.line(annotated_image,
                 (center_x_pixel, center_y_pixel - line_length),
                 (center_x_pixel, center_y_pixel + line_length),
                 RED_COLOR, thickness)

        cv2.circle(annotated_image, (center_x_pixel, center_y_pixel), 15,
RED_COLOR, thickness)

writer = Writer()
```

```
class Drawer():  # GUI layer

    def display_drone_info(self, image, temperature, battery, velocity,
save_video, fps, use_live_stream):

        if save_video:
            writer.save_video(image)

        if use_live_stream:
            writer.velocity(image, velocity)
            writer.temperature(image, temperature)
            writer.battery(image, battery)
            writer.fps(image, fps)

        return image

    def DrawTarget(self, lenght, annotated_image, center_x_pixel,
center_y_pixel):
        writer.distance(annotated_image, lenght, center_x_pixel, center_y_pixel) #
Виведення відстані на зображення
        writer.circle(annotated_image, center_x_pixel, center_y_pixel)

class ScreenHandler():   # GUI layer

    def get_screen_parameters(self, use_live_stream):
        if use_live_stream:
            video_source = None  # Для LIVE STREAM
            width = 960
            height = 720
            fps = 30
        else:
            video_source = cv2.VideoCapture(0)  # Відео з камери ноутбука
            width = int(video_source.get(cv2.CAP_PROP_FRAME_WIDTH))
# Отримуємо ширину кадру
            height = int(video_source.get(cv2.CAP_PROP_FRAME_HEIGHT))
# Отримуємо висоту кадру
            fps = video_source.get(cv2.CAP_PROP_FPS)
        return video_source,width,height,fps
```

```
    def display_result(self, screen, annotated_image):
        frame = np.rot90(annotated_image)
        frame = np.flipud(frame)
        frame = pygame.surfarray.make_surface(frame)
        screen.blit(frame, (0, 0))

    def get_frame(self, frame_read, video_source, use_live_stream):
        if use_live_stream:
            frame = frame_read.frame
            ok = True
        else:
            ret, frame = video_source.read()
            ok = bool(ret)

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        return frame,ok

class UserInterface():  # GUI layer
    def __init__(self):
        self.is_running = True
        self.is_flying = False

    def key_action(self, is_flying, is_running, use_live_stream, save_video,
video_writer, fps, width, height):

        action = 'none'
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                if use_live_stream and is_flying:
                    action = 'land','streamoff'
                    is_flying = False
                else:
                    action = 'streamoff'
                is_running = False
            if event.type == pygame.KEYDOWN:
                if use_live_stream:
                    if event.key == pygame.K_r:
```

```
                    save_video = not save_video
                    if save_video:
                        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                        video_writer = cv2.VideoWriter('video.mp4', fourcc, fps,
(width, height))
                        print("Запис відео розпочато...")
                    else:
                        if video_writer is not None:
                            video_writer.release()
                            print("Запис відео завершено...")
                if event.key == pygame.K_t and not (is_flying):
                    is_flying = True
                    action = 'takeoff'
                if event.key == pygame.K_l and is_flying:
                    action = 'land'
                    is_flying = False
        return is_flying, is_running, save_video, video_writer, action

class CreatePlot():  # GUI layer

    def plot_graph(self, x_values, y_values):
        if not x_values or not y_values:
            print("Масиви порожні, графік не може бути побудований.")
            return

        # Використовуємо індекси елементів як шкалу по осі X
        indices = list(range(len(x_values)))

        plt.figure(figsize=(10, 6))
        plt.plot(indices, x_values, marker='o', markersize=3, label="Значення X")
        plt.plot(indices, y_values, marker='o', markersize=3, label="Значення Y")

        plt.title("Графік значень")
        plt.xlabel("Шкала часу")
        plt.ylabel("Значення виходів")
        plt.legend()
        plt.grid(True)
        plt.show(block=True)  # Відображення графіка
```