НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних систем, мереж та кібербезпеки

_____ Касаткін Д.Ю., к.пед.н., доц.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

На тему: «Розробка системи виявлення БПЛА»

Спеціальність F7 «Комп’ютерна інженерія»

Гарант освітньої програми

к.фіз.-мат.н., доц. _____ / Нікітенко Є.В. /

(підпис) (ПІБ)

Керівник дипломного проекту: _____ / Волошин С.М. /

(підпис) (ПІБ)

Виконав: _____ / Яловенко В.В/

(підпис) (ПІБ)

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.пед.н., доц. /
(підпис) (ПІБ, вчене звання і ступінь)
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ СТУДЕНТУ

_Яловенко Володимира Вікторовича
(прізвище, ім'я, по батькові)
Спеціальність (напрям підготовки): F7 Комп'ютерна інженерія
Тема кваліфікаційної бакалаврської роботи: Розробка системи виявлення БПЛА
затверджена наказом ректора НУБіП України від "16" _грудня_2024р. №___2250 «С»
Термін подання завершеної роботи на кафедру
Вихідні дані до кваліфікаційної бакалаврської роботи
Перелік питань, що підлягають розробці:
1. Аналіз вимог до системи
2. Проектування системи
3. Тестування системи
Перелік графічного матеріалу (за потреби)

Дата видачі завдання "__16___" ___грудня__2024 р.

Керівник кваліфікаційної роботи _____ Волошин С.М.
(підпис) (прізвище та ініціали)
Завдання прийняв до виконання _____ Яловенко В.В
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	20.12.2024	Виконано
2	Проектування системи	15.01.2025	Виконано
3	Реалізація системи	15.02.2025	Виконано
4	Тестування системи	15.03.2025	Виконано
5	Оформлення пояснювальної записки	15.04.2025	Виконано
6	Оформлення графічного матеріалу	15.05.2025	Виконано

Студент _____ **В.В. Яловенко**
(підпис) (ініціали та прізвище)

Керівник роботи _____ **С.М Волошин**
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 58 сторінок, 14 рисунків, 29 лістингів, 17 джерел.

ВИЯВЛЕННЯ БПЛА, РОЗПІЗНАВАННЯ ОБРАЗІВ, ЗГОРТКОВІ НЕЙРОМЕРЕЖІ, КЛІЄНТ, СЕРВЕР, СЕРВОПРИВІД, БАГАТОПОТОКОВІСТЬ, МЕРЕЖЕВА КОМУНІКАЦІЯ

Об'єкт дослідження – Розробка комп’ютерної системи виявлення БПЛА з використанням технологій машинного навчання. Метою є створення макету прототипу системи, здатного розпізнавати повітряні цілі, та орієнтованого на подальше масштабування.

У першому розділі розглядаються та порівнюються способи виявлення образів. Надано порівняння традиційних методів машинного навчання та методів, що базуються на використанні нейромереж. Розглянути принцип роботи згорткових нейромереж. Розглянуто найбільш розповсюджені архітектури нейромереж, їх принцип роботи. Обрано оптимальну архітектуру для виконання поставлених задач.

У другому розділі увага приділяється архітектурі системи. Розглянуто алгоритм поводження системи до та під час виявлення цілі. Описано апаратну архітектуру макету, та описано характеристики апаратної складової, що використовується у його побудові.

У третьому розділі, описуються кроки побудови макету. Розглянуто основні етапи побудови макету. Описано кроки, необхідні для підготовки сервера та клієнта для роботи. Вказано кроки та метапараметри, необхідні для тренування нейромережі. Зазначено особливості живлення сервоприводів та способи керування ними. Розглянуто програмне забезпечення сервера та клієнта, особливості роботи з мережевою комунікацією.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БПЛА – Безпілотний літальний апарат

МВГ – мобільна вогнева група

ППО – протиповітряна оборона

ПЗРК – переносний зенітно-ракетний комплекс

OpenCV – Open-Source Computer Vision Library

CNN – Convolutional Neural Network

RCNN – Region Based Convolutional Neural Networks

SSD – Single Shot MultiBox Detector

YOLO – You Only Look Once

PWM – Pulse Width Modulation

RPI – Raspberri pi

GPIO – General Purpose Input Output

PIP – Python Package Index (PyPi)

FPGA – Field-Programmable Gate Array

ВСТУП

Наслідком тотальної війни російської федерації проти України є атаки критичної цивільної інфраструктури зокрема за допомогою БПЛА камікадзе. Дешевизна ударних БПЛА не дозволяє використовувати дорогі засоби збиття, тому значна частка оборони повітряного простору покладається на МВГ ППО, які озброєні стрілецькою зброєю та ПЗРК. На сьогодні, більшість МВГ покладаються на власні органи зору для виявлення БПЛА. Натомість, використання автоматизованих систем виявлення дозволить пришвидшити виявлення БПЛА та зменшити навантаження на особовий склад.

Актуальність роботи обумовлена сучасними реаліями військового та цивільного життя, та необхідністю покращення обороноздатності Національного повітряного простору. Автоматизовані системи виявлення є одним з напрямків, які можуть бути покращені.

Наукова новизна полягає у використанні нейромереж для розпізнавання цілі у реальному часі, модульному підході до виконання задачі та поєднанні інформаційних та технічних рішень, що дозволять максимізувати масштабованість та універсальність прототипу.

Практичне значення полягає у створені простого у використанні прототипу системи, яка придатна для масштабування та модифікації в залежності від вимог та матеріальних можливостей.

Мета роботи – створення прототипу системи, що здатна автоматично виявляти БПЛА та інформувати особовий склад про координати цілі.

Предмет дослідження: автоматизовані системи виявлення БПЛА в реальному часі.

Об’єкт дослідження: процес виявлення та ідентифікації БПЛА мобільними вогневими групами ППО.

Завдання дослідження: розробити програмну та апаратну систему виявлення БАЛА.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поставлення мети роботи

Ціллю дослідження та розробки цієї системи доступу було створення моделі розпізнавання та відслідковування БПЛА у автоматичному режимі. Зважаючи на різноманітність можливих середовищ використання та обладнання, створення аналітичного алгоритму не є доцільним. Натомість, є потреба використання нелінійних методів розпізнавання образів, таких, як штучні нейронні мережі.

Також, беручи до уваги, різноманітність матеріально-технічного забезпечення різних підрозділів, є потреба створення системи, яка може працювати з різними конфігураціями обладнання, з мінімальним додатковими налаштуваннями.

Важливим аспектом, також, виступає модульність дизайну, розосередженням вразливих елементів системи, з метою мінімізації матеріальних втрат у небезпечному та агресивному середовищі.

Центральною частиною системи є Raspberry PI Zero 2 W. Цей міні комп’ютер має енергоефективний ARM процесор, GPIO виводи та micro USB порт, які дозволять керувати периферією за допомогою ОС Linux. Малий форм фактор та низька ціна, позитивно вирізняють Raspberry PI Zero 2 W на фоні інших, навіть сучасніших моделей з лінійки Raspberry PI. Мінусом використовуваного комп’ютера є відносно менш продуктивний процесор та менша кількість портів периферії, проте, для задач, які виконуватимуться у цій системі, ці мінуси не позначяться на подальшій роботі.

1.2.Огляд та порівняння методів розпізнавання образів

У цьому розділі розглянемо основні аспекти та технології, що стосуються розробки нашої системи на базі розпізнавання образів**.** Порівняємо методи розпізнавання образів, зважаючи на три основні аспекти: точність, швидкість та масштабованість.

1.2.1. Традиційні методи машинного навчання

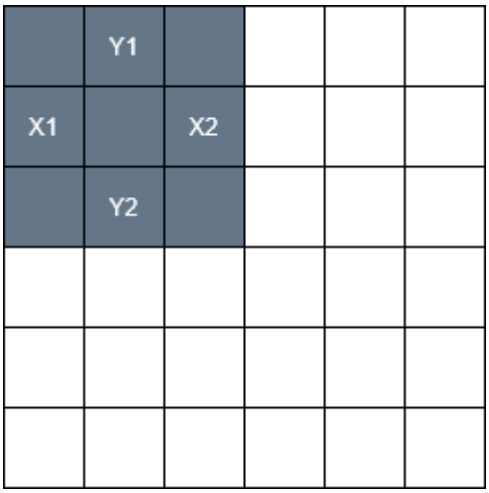
Переважна більшість методів розпізнавання образів покладається на штучний інтелект. Розглянемо підгалузі штучного інтелекту (рис. 1.1).

Машинне навчання – це галузь штучного інтелекту орієнтовано на створення методів виконання задач комп’ютера без прямих інструкцій. Ідея полягає у наданні комп’ютеру даних, на основі яких буде виконуватися навчання та з метою розпізнавання або відтворення даних подібних, до тих, на основі яких виконувалося навчання.

Традиційні алгоритми машинного навчання (які реалізовані без використання нейромереж) покладалися більшою мірою на ручне виокремлення ознак. Вони характеризуються простотою реалізації та, як наслідок, простотою інтерпретації виконання алгоритму людиною, з метою виправлення недоліків моделі. Ці алгоритми доцільно використовувати для виконання простих задач, таких як класифікація тексту – дерево ухвалення рішень (Decision tree);

класифікація – SVM; вилучення ознак – HOG.

Комбінація двох крайніх алгоритмів може використовуватися для створення програми з відстежування образів на зображенні. HOG використовується для обробки зображень та створення гістограми напрямків контурів та яскравості градієнтів на зображенні. Зображення розділяється на поля, з якими ведуть обрахунок (рис. 1.2).



Після того, виконують обрахунок градієнта на полі 3x3 та виконують зсув з ліво направо, зверху вниз для кожного пікселя зображення.

Відомо що, яскравість дорівнює:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad , \qquad (1.1)$$

Відомо що, напрямок дорівнює:

$$atan(\frac{|x_1 - x_2|}{|y_1 - y_2|}) \quad , \qquad (1.2)$$

В кінці, надаємо обраховані градієнти SVM класифікатору та отримуємо результат.

Недоліком цього підходу, як і більшості традиційних методів, є погана здатність до масштабування та повільна робота із зображеннями високої роздільної здатності. Також, у порівнянні з методами, які розглянемо далі, HOG + SVM має погану точність та значну кількість хибних спрацювань. [2]

1.2.2. Методи глибокого навчання

Глибоке навчання – це підмножина машинного навчання. У ньому використовується нейромережі з декількома прихованими шарами. Нейромережі були натхненні будовою людського мозку [3], створені на основі найпростіших елементів – нейронів. Будова нейрону штучної нейромережі складається з одного або більше входів та одного виходу. Кожен вхід має свої ваги.

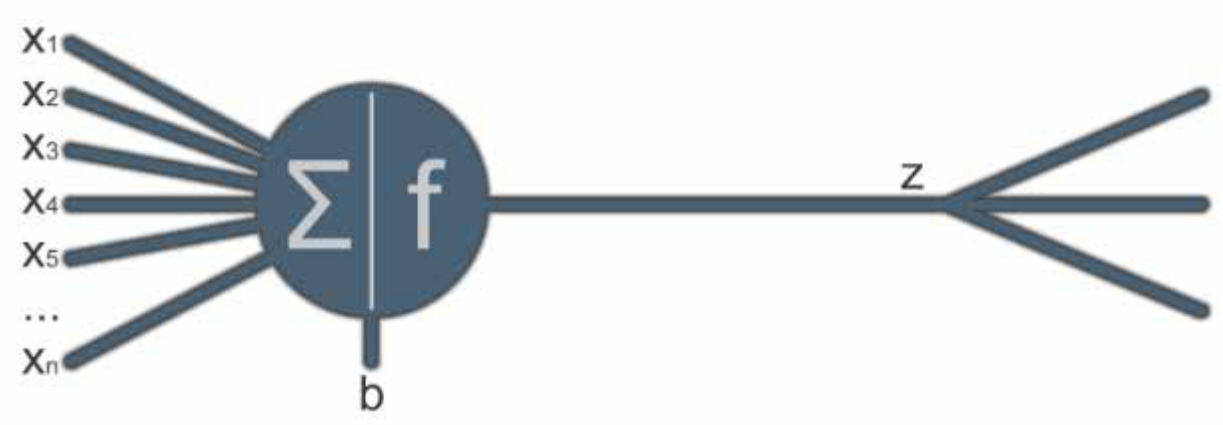


Рис. 1.3 – Будова штучного нейрона [3]

Нейрон додає добутки всіх входів та вагів і зсув.

Відомо, що:

$$z = \left(\sum_{i=1}^{n} x_i \times w_i \right) + b \quad , \qquad (1.3)$$

де z – вихід нейрона;

x – вхідне значення нейрона;

w – ваги;

b – зсув.

Наостанок, обрахована сума добутків та зсуву обраховується функцією активації.

Leaky Rectified Linear Activation (1.4) є найрозповсюдженішою функцією активації у сучасних нейромережах. Це вдосконалена версія звичайної Rectified Linear Activation, яка надає додаткові можливості обрахунку помилки.

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}, \quad (1.4)$$

Глибокі нейромережі складаються з декількох шарів нейронів. Більшість моделей відстежування є згортковими.

1.3. Згорткові нейромережі

Відмінністю згорткових нейромереж від звичайних класифікаторів є наявність згорткових шарів. Їх задачею є виокремлення ознак, для подальшої класифікації зображення. Згорткові нейромережі надають значно кращу точність розпізнавання образів та є найбільш розповсюдженим типом архітектури виявлення образів. Вони складаються з вхідного, вихідного та прихованих шарів. Вхідний шар отримує зображення та складається з кількості нейронів, яка дорівнює роздільній здатності зображення. Вихідний шар призначений для виведення висновку мережі, будь то клас розпізнаного зображення, або його координати. Приховані шари виконують решту задач та складаються з:

- згорткових шарів, які виконують виокремлення ознак
- агрегувальних шарів, які зменшують розмірність шару
- повноз’єднаних шарів, які виконують інтерпритацію ознак з метою виконання задач, які очікуються від нейромережі (класифікація, відстежування об’єктів)

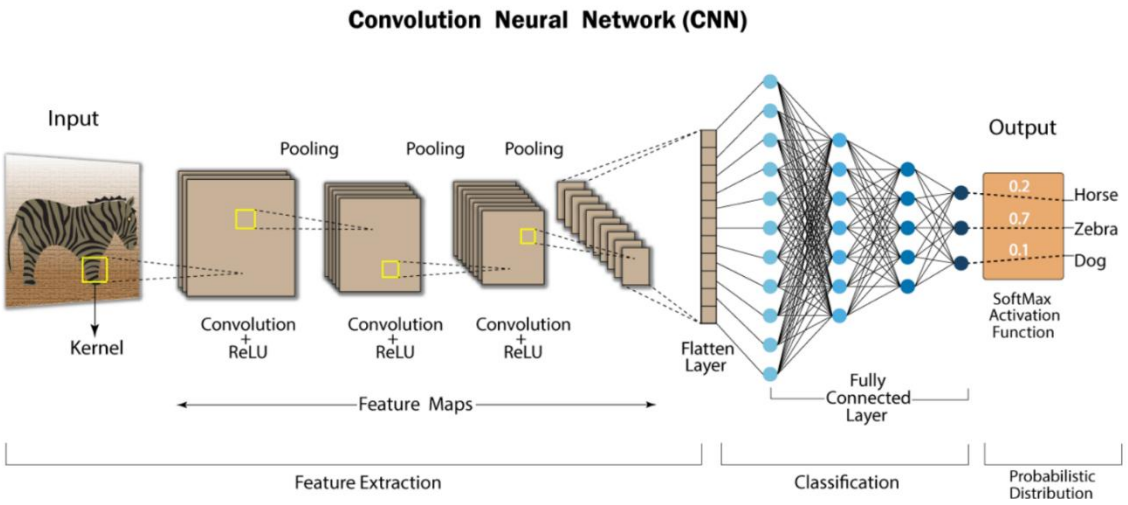


Рис. 1.4 – Приклад архітектури згорткової нейромережі [4]

Розглянемо принцип роботи згорткового шару. У ньому, виконується опрацювання зображення шляхом зсуву ядра (рис 1.4), тільки у цьому випадку, ми шукаємо не градієнти, а виконуємо множення за рахунок розбиття зображення на матриці та перемножування їх з матрицею ядра (рис. 1.6) (рис. 1.7). Це дозволяє виокремлювати ознаки у вхідному зображенні, що спрощує коректне досягнення висновків повнозв'язними шарами.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Роль кожного нейрона у згортковому шарі полягає в тому, щоб виявити

локальну ознаку на своїй позиції в зображенні. Проте, на відміну від класичних

шарів, нейрони згорткового шару не унікальні за вагами, вони поділяють спільні ваги (ядро згортки або фільтр). Тобто, всі нейрони одного фільтра мають однакові параметри, але працюють над різними локальними ділянками вхідного зображення. Під час тренування, у нейромережі підбираються ті згорткові ядра, які надають найкращі результати розпізнавання, будь-то виявлення країв, розмиття зображення тощо.

Хоч основна ідея згорткових нейромереж лишається тою ж, проте їх архітектури можуть суттєво відрізнятись за багатьма параметрами, саме ці відмінності безпосередньо впливають як на точність, так і на продуктивність. Розглянемо найрозповсюдженіші архітектури.

1.3.1. Faster RCNN

Faster R-CNN — це глибока нейронна мережа для виявлення об'єктів, яка поєднує регіональні пропозиції з CNN-класифікацією. Вона була запропонована у 2015 році, як покращення над R-CNN та Fast R-CNN. Faster R-CNN об’єднує два етапи в одну мережу: Генерація регіонів-пропозицій та Класифікація цих регіонів і визначення координат. Faster RCNN Складається з:

1) Backbone, що отримує ознаки зображення за рахунок згорткової мережі.
2) Region proposal Network, що прогнозує область на зображені, на якій може бути об’єкт та впевненість розпізнання
3) Pooling, що перетворює кожен регіон-пропозицію до фіксованого розміру, забезпечуючи сумісність з повнозв’язними шарами.
4) Класифікатор меж, що класифікує об'єкт у кожному регіоні [5]

Процес розпізнавання зображення за допомогою Faster RCNN починається з отримання ознак зображення за допомогою CNN. Ця мапа ознак передається Region proposal Network, що генерує близько 2000 регіонів (якорів). Для кожного шару застосовується Pooling щоб отримати фіксований розмір. Нарешті, мережа класифікує зображення та надає координати меж зображення.

Faster R-CNN — це дуже точна модель для задач, де важлива висока якість

виявлення об'єктів. Ціною цьому є повільна робота та значні обчислювальні витрати. Ця модель досі часто використовується у наукових дослідженнях, медицини, відеоспостереженні та автономних системах, де важлива точність більше, ніж виявлення у реальному часі.

1.3.2. Single Shot MultiBox Detector

Single Shot MultiBox Detector (SSD) — це одноетапна нейронна мережа для виявлення об’єктів на зображеннях. Вона була представлена в 2016 році компанією Google і є швидшою альтернативою двоетапним моделям, таким як Faster R-CNN, при збереженні високої точності. SSD складається з:

1. Backbone, що отримує координати зображення за рахунок згорткової мережі (наприклад VGG16, ResNet, MobileNet).
2. Додаткові згорткові шари (Extra Feature Layers), додають нові шари з меншою роздільною здатністю. Це дозволяє знаходити об’єкти різного розміру.
3. Predictor heads (вихідні шари), виконують регресію координат (x, y, w, h) та виводять впевненість у кожному класі.

Принцип роботи SSD полягає в тому, що модель виконує виявлення об'єктів за один прохід (single shot), на відміну від двоетапних моделей (типу Faster R-CNN), які спочатку знаходять регіони, а потім класифікують їх. [6]

SSD — це швидка і компактна архітектура виявлення, яка добре підходить для реального часу. Хоч ця модель і не настільки точна як Faster RCNN, проте перевага у швидкості дозволила цій моделі стати основою для багатьох оптимізованих рішень на мобільних і вбудованих пристроях.

1.3.3. You Only Look Once (YOLO)

YOLO — це сімейство нейромереж для виявлення об'єктів у реальному часі. Головна ідея — обробка зображення в один прохід нейромережею, на відміну від двоетапних методів, таких як Faster R-CNN. Замість окремої генерації

Region proposal (як у Faster R-CNN), YOLO прямо передбачає координати та класи всіх об’єктів в один етап. Архітектура YOLO складається з:

1. Backbone, що отримує координати зображення за рахунок згорткової мережі (наприклад C2f, Conv, Bottleneck).
2. Neck, що збирає інформацію з різних рівнів абстракції (різні масштаби).
3. Head (вихідна частина), що для кожної клітинки сітки передбачає координати меж (x, y, w, h), впевненість наявності об’єкта та ймовірності визначених класів. [7]

Принцип роботи Neck у YOLOv8 — а саме, компонента FPN + PAN — полягає в ефективному об'єднанні ознак з різних рівнів глибини мережі, щоб модель краще виявляла об’єкти різного розміру. Ознаки з глибших шарів (менші за розміром, але, пройшовши попередні шари, багатші на семантику) — краще "розуміють", що є на зображенні. Ознаки з ранніх шарів (великі за розміром, але з низьким рівнем абстракції) — краще зберігають положення об’єкта.

Head — це остання частина нейронної мережі, яка виконує передбачення об'єктів. Вона бере оброблені ознаки від Neck і перетворює їх у конкретні передбачення: координати рамок, класи об’єктів, і вірогідність. YOLOv8 використовує детектор з трьома масштабами, тобто має окремі "голови" для ознак різних розмірів (наприклад, 80×80, 40×40, 20×20). Це дозволяє виявляти об’єкти різного розміру. Тож, Head перетворює абстрактні ознаки в конкретні передбачення працює паралельно на кількох масштабах видає координати, класи, і впевненість.

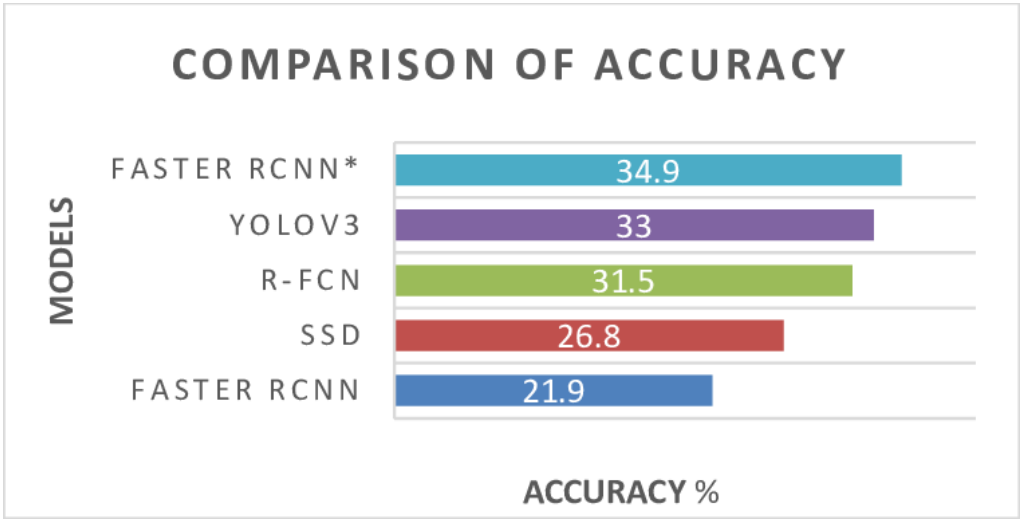
YOLOv8 — це модульна, без якірна архітектура, яка працює швидко і точно навіть на слабкому залізі. Вона оптимізована для практичного використання — мобільних, embedded та edge-пристроїв.

1.3.4. Обрання оптимальної мережі

Під час порівняння мереж для задач, які потребують виконання у реальному часі, швидкість моделі не поступається у важливості точності.

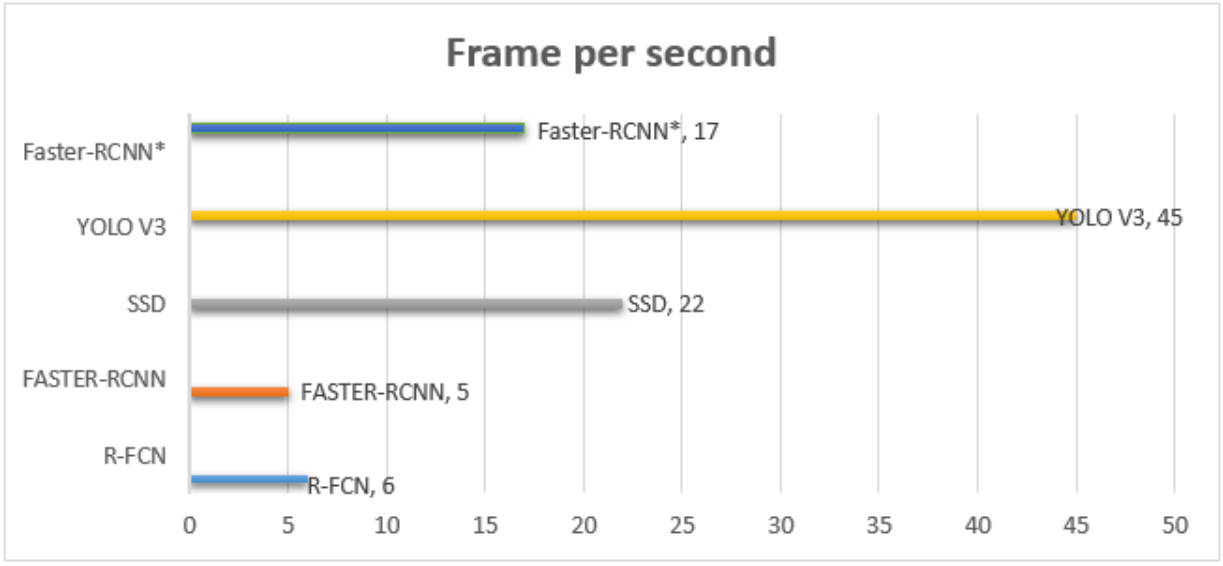
Порівнявши архітектури, переваги та недоліки трьох моделей, можемо зробити висновок щодо оптимальної моделі. Бачимо, що Faster RCNN є найточнішою з розглянутих моделей (рис. 1.7).

Двоетапна архітектура дозволяє визначити ділянки, де можуть бути об’єкти (Region Proposal), щоб після того, класифікатор міг визначити, що це за



об’єкт. Проте, сфера у якій Faster RCNN поступається іншим розглянутим мережам, це швидкість (рис 1.9).

У плані швидкості, беззаперечним переможцем є YOLO. YOLO не



використовує повільних шарів, як Region Proposal та ROI. Крім того, YOLO не надто поступається Faster RCNN у точності.

Таблиця 1.1

Порівняння розглянутих моделей

Критерій	Faster R-	YOLO	SSD	Коментарі

	CNN			
Тип архітектури	Двоетапна	Одноетапна	Одноетапна	YOLO таSSD швидші завдяки меншій складності
Швидкість	Низька	Висока	Вища за Faster R-CNN, нижча за YOLO	YOLO найшвидший для реального часу
Точність	Висока	Висока	Середня/висока	Faster R-CNN кращий для дрібних або перекритих об'єктів
Складність реалізації	Складна	Середня	Середня	Faster R-CNN вимагає більше обчислень
Підтримка дрібних об'єктів	Хороша	Посередня	Посередня	Двоетапність допомагає краще локалізувати
Використання у реальному часі	Погано підходить	Добре підходить	Добре підходить	YOLO створений для швидких застосувань

Беручи все до уваги, було прийнято рішення використовувати модель YOLO, через її виняткову швидкість та значну точність. Недоліки YOLO у плані гіршої роботи на складному фоні, будуть нівельовані специфікою застосування.

РОЗДІЛ 2

СТРУКТУРА ТА ФУНКЦІОНАЛ СИСТЕМИ

2.1. Етапи розробки системи

Розробку можна поділити на декілька основних етапів, що є важливими для подальшого проектування системи:

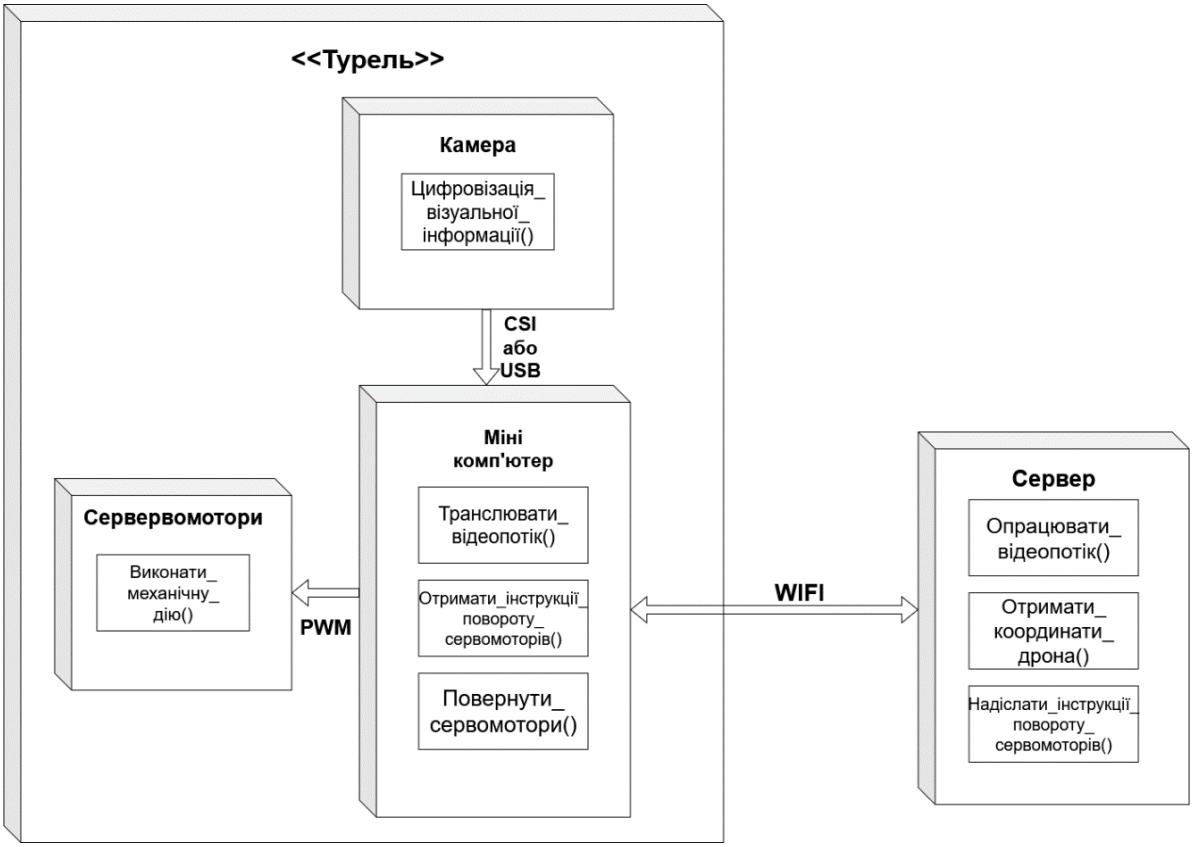
1) Обрання та тренування моделі виявлення зображень:
   - порівняння існуючих моделей розпізнавання образів.
   - пошук, обрання та доповнення набору даних, на основі яких виконуватиметься тренування.
   - задання метапараметрів та тренування моделі
2) Налаштування та підготовка міні комп’ютера:
   - пошук методу передачі відеопотоку
   - пошук методу керування сервоприводом
3) Проектування фізичної моделі:
   - під’єднання елементів турелі для автономної роботи
   - створення 3D моделі опорно рухової частини та корпусу макету
4) Розробка програмного забезпечення міні комп’ютера:
   - розробка програми отримання та інструкцій повороту сервоприводів.
   - програмування алгоритму пошуку цілі.
5) Тестування та оцінка продуктивності:
   - проведення тестів у різних умовах для оцінки точності та надійності системи.
   - аналіз результатів та оптимізація системи для покращення її роботи.

2.2. Архітектура системи

Основною задумкою системи є взаємодія декількох модулів, які контролюють периферію та опрацьовують зображення за допомогою нейромережі. Для цього треба використовувати конфігурацію з двох комп’ютерів:

- міні комп’ютера, який працює з камерою, сервомоторами та надсилає відеопотік за допомогою WIFI;
- сервера з більшими обчислювальними потужностями, який буде опрацьовувати відеопотік, та надсилати інструкції повороту сервомоторів.

Всі елементи, окрім сервера, є структурною частиною турелі (рис. 2.1). Потреба у потужнішому сервері поставатиме разом зі збільшенням роздільної здатності камери, і, як наслідок, потребуватиме нейромережу з більшою роздільною здатністю, для того, щоб використати оптичну перевагу. У практичному використанні, до турелі буде прикріплено прожектор, який буде зорієнтований таким чином, що його промінь світитиме у напрямку центра поля зору камери.



2.2.1. Пошук цілі

Тоді, коли модель не розпізнає жодної цілі, камера виконує пошук сектору ведення вогню шляхом повороту у заданому азимуті. Важливо, щоб під час цього, турель зберігала структурну цілісність та достатній рівень заряду батареї. Тому, швидкість повороту повинна бути меншою за максимальну. Під час цього, міні комп’ютер продовжує відсилати відеопотік на сервер, який у свою чергу, виконує розпізнавання зображення. Також, необхідно бути впевненим, що зв’язок з сервером не перервався під час годин очікування. Якщо це трапилося, турель зупиняє рух, сигналізуючи, про несправність.

2.2.2. Відслідковування цілі

Коли сервер виявляє БПЛА, він переймає контроль над рухами камери міні комп’ютера та надсилає інструкції щодо повороту камери таким чином, щоб

БПЛА зображення знаходився у центрі поля зору камери. Разом з тим, це забезпечує відслідковування цілі бійцями, за рахунок прожектору, який спрямовано у центр поля зору камери. При припиненні виявлення цілі (через вихід з поля зору або помилку розпізнавання моделі) турель продовжить обертання у попередньому напрямку. Це забезпечить можливість продовжити виявлення втраченої цілі. Через декількох секунд, після втрати цілі, турель продовжить патрулювання доки не виявить черговий БПЛА.

2.3. Матеріально-технічна складова макету

У цьому розділі розглянемо технічні рішення, пов’язані зі створенням макету відстежування БПЛА. Ця робота виконувалася з чітким розумінням того, що для того, щоб прототип був придатний до використання у бойових умовах, потрібно витратити набагато більше матеріального та часового ресурсу, ніж є у наявності в рамках бакалаврської роботи. Тому, технічні рішення приймалися з розрахунком на створення прототипу, придатного для подальшої розробки, масштабування та експериментування. Тож, зважаючи на це, порівняємо можливі способи досягнення цілей.

2.3.1. Міні комп’ютер

У ролі мінікомп’ютера використовуватимемо Raspberry Pi Zero 2 W. Цей модуль, попри те, що він слабший за новіші моделі, виграє у плані розміру та енергоефективності. Завдяки своїм компактним габаритам (всього 65×30 мм), він ідеально підходить для проектів з обмеженим простором, таких як вбудовані системи, переносні пристрої чи роботи (Рис. 2.2).

Zero 2 W оснащено чотириядерним процесором Broadcom BCM2710A1 Cortex-A53 з тактовою частотою до 1 ГГц, що забезпечує суттєве зростання продуктивності в порівнянні з оригінальним Zero. У пристрої доступно 512 МБ оперативної пам’яті, що є достатнім для виконання базових обчислювальних

задач, таких як обробка сигналів, керування сенсорами чи обмін даними через мережу.

Raspberry Pi Zero 2 W має вбудований Wi-Fi 802.11 b/g/n та Bluetooth 4.2, що дозволяє створювати бездротові системи моніторингу, керування або обміну даними без потреби у зовнішніх модулях. Живлення здійснюється через micro-USB, що робить його сумісним із більшістю стандартних блоків живлення або мобільних акумуляторів.

Ще однією перевагою є сумісність із операційною системою Raspberry Pi OS, яка базується на Debian і має велику кількість доступних бібліотек та інструментів для розробки. Завдяки цьому пристрій легко інтегрується в системи на базі Python, C/C++, Node.js та інших мов програмування. Також доступна повна підтримка GPIO-портів, що дозволяє підключати датчики, приводи, екрани та інші компоненти.

Таким чином, Raspberry Pi Zero 2 W є збалансованим рішенням для проектів, де ключову роль відіграють компактність, низьке енергоспоживання та достатній рівень обчислювальної потужності.

Основні характеристики RPI Zero 2 W:

- процесор (CPU): Broadcom BCM2710A1, 64-бітний, 4-ядерний ARM Cortex-A53; Частота: 1.0 ГГц

- оперативна пам’ять (RAM): 512 МБ LPDDR2 SDRAM
- графіка (GPU): Broadcom VideoCore IV
- бездротові інтерфейси: Wi-Fi 802.11 b/g/n (2.4 GHz)
- bluetooth 4.2, Bluetooth Low Energy (BLE)
- порти: 1 × micro-USB для живлення; 1 × micro-USB OTG для периферії; 1 × mini-HDMI (1080p60); 1 × слот для microSD (система зберігається на карті пам’яті); 1 × CSI-камера роз’єм (вузький FPC, сумісний з камерами для Pi Zero)
- GPIO: 40 контактів GPIO (не розпаяні, можна припаяти)
- розміри: 65 мм × 30 мм × ~5 мм
- маса: приблизно 11 г
- живлення: 5 В через micro-USB (рекомендовано блок живлення на 5 В / 2.5 А)
- сумісність: Сумісна з багатьма аксесуарами Raspberry Pi Zero; Підтримка Raspberry Pi OS та інших дистрибутивів Linux (наприклад, Raspbian, Ubuntu, DietPi)

У макеті використовуються порти: 1 × micro-USB для живлення; 1 × micro-USB для камери; 1 × слот для microSD для операційної системи та програми; 2 х контакта GPIO в режимі PWM для контролю сервомоторів. Передача відеопотоку та отримання інструкцій повороту сервомоторів виконуватиметься за допомогою бездротового інтерфейсу WIFI.

2.3.2. Сервопривід Сервомотор

SG90 — це один з найпопулярніших і недорогих сервоприводів, який часто використовується в навчальних і хобі-проєктах, зокрема з **Raspberry Pi**. Завдяки своїй компактності, легкій вазі (близько 9 г) та простому керуванню, він ідеально підходить для реалізації механізмів повороту, маніпуляторів, шасі з поворотними колесами, камери на платформі тощо.

Цей сервомотор має робочий кут обертання приблизно 180°, а управління

ним здійснюється за допомогою PWM-сигналу (широтно-імпульсної модуляції), що робить його сумісним з GPIO-портами Raspberry Pi або Arduino. Напруга живлення становить від 4.8 до 6 В, а типовий струм споживання — близько 100–250 мА при навантаженні, що дозволяє живити його навіть від невеликого джерела живлення.

Усередині SG90 вбудований контролер положення, тому сервопривод автоматично підтримує заданий кут повороту без потреби в зовнішньому зворотному зв’язку. Це значно спрощує програмування.

Ще однією перевагою є доступність та універсальність кріплень, що постачаються в комплекті — зокрема, різні типи важелів і гвинтів, які можна легко інтегрувати в корпуси 3D-друку.

Основні характеристики SG90:

- напруга живлення: 4.8–6.0 В
- максимальний крутний момент: ~1.8 кг·см при 4.8 В
- кут обертання: Приблизно 0–180° (може бути трохи менше)
- швидкість обертання: ~0.1 с/60° при 4.8 В
- тип управління: ШІМ (PWM)
- вага: 9 г
- матеріал шестерень: Пластик

Сервопривід керується за допомогою PWM-сигналу. Ширина імпульсу визначає кут повороту валу:

- 1 мс → приблизно 0°
- 1.5 мс → приблизно 90°
- 2 мс → приблизно 180°

Сигнал повторюється з частотою 50 Гц.

У SG90 зазвичай три дроти:

- помаранчевий — керуючий сигнал (PWM)
- червоний — живлення (VCC, 5 В)
- коричневий — земля (GND)

2.3.3. Камера

У цьому макеті використовуватиметься звичайна вебкамера TR18679 (рис. 2.4) Хоч вона і не придатна для нічної зйомки через відсутність інфрачервоних фільтрів та підсвітки, проте для демонстрації концепту її базових характеристик буде цілком достатньо. Ця модель забезпечує роздільну здатність до 720p та підключається через стандартний USB-інтерфейс, що дозволяє легко інтегрувати її з Raspberry Pi або будь-яким іншим комп’ютером.

Камера підтримується більшістю сучасних операційних систем і сумісна з такими бібліотеками, як OpenCV, що дає змогу проводити обробку відеопотоку в реальному часі, виявлення об’єктів, а також реалізацію базових систем машинного зору.

Завдяки низькому енергоспоживанню та доступності, TR18679 підходить для прототипування і початкових стадій розробки, де ключовим є тестування логіки системи, а не якість зображення. У разі потреби модель легко замінюється на більш потужні камери з підтримкою нічного бачення чи вищою роздільною здатністю, без необхідності змінювати архітектуру системи.

Основні характеристики TR18679:

- роздільна здатність відео: HD 720p (1280×720 пікселів) при 30 кадрах/с

- фотографії: до 8 мегапікселів (інтерполяція)
- сенсор: CMOS, 1.3 МП
- фокус: фіксований
- кут огляду: 52°
- мікрофон: вбудований, з автоматичним балансом білого
- кнопка знімання: є, для швидкого створення фотографій
- кріплення: універсальна прищіпка або настільне встановлення
- інтерфейс: USB 2.0, довжина кабелю — 1.43 м
- розміри: 70×62×81 мм
- вага: 88 г

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Інсталяція залежностей

У цьому проекті використовується низка програм та бібліотек, які можуть потребувати компіляції з вихідного коду. І сервер, і міні комп'ютер (у подальшому клієнт) працюють на Linux дистрибютиві на основі Debian (Raspberry pi працює з Rapberry pi OS. Це важливо, тому що у коплекті з цією ОС ідуть бібліотеки для контролю GPIO). На сервері буде виконано інсталяцію бібліотек мови Python: ultralitics та open cv 2. Також, було встановлено програми CUDA для прискорення роботи за допомогою відеокарти. Для коректної роботи PWM сигналу буде встановлено програму pigpio. Для трансляції відеопотоку використовується програма mjpegstreamer.

3.1.1. Підготовка серверу

У цьому проєкті використовується pyenv — утиліта для контролю версій Python. Вона особливо корисна у випадках, коли потрібно працювати з різними версіями Python для різних проєктів, що часто трапляється у складних середовищах розробки. Pyenv дозволяє легко встановлювати, перемикатися та використовувати потрібну версію Python без зміни системного Python, що мінімізує ризик конфліктів із системними пакетами та оновленнями.

Однією з переваг pyenv є можливість використання системи керування пакунками pip без обов'язкового створення віртуального середовища, хоча воно також підтримується через pyenv-virtualenv. Це забезпечує більшу гнучкість у розгортанні та обслуговуванні Python-середовищ, особливо в автоматизованих або скриптових системах.

- Pyenv має зручну CLI-інтерфейсну команду для:

- встановлення нових версій Python (pyenv install)
- вибору глобальної або локальної версії для конкретного проєкту (pyenv global, pyenv local)
- перегляду наявних та активних версій (pyenv versions, pyenv which)

Для інсталяції pyenv можна скористатися автоматичним інсталятором. [9]

```
curl -fsSL https://pyenv.run | bash
```

Після цього, треба асоціювати утиліту з оболонкою операційної системи (у подальшому bash)

```
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
echo '[[ -d $PYENV_ROOT/bin ]] && export
PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(pyenv init - bash)"' >> ~/.bashrc
Інсталюємо та переключимось на потрібну версію python
pyenv install 3.12.8
pyenv global 3.12.8
```

Далі, можна приступити до інсталяції бібліотек.

Бібліотека ultralytics — це офіційна Python-реалізація новітніх моделей сімейства YOLO від компанії Ultralytics. Вона надає зручний високорівневий інтерфейс для тренування власних моделей, запуску інференсу з попередньо натренованими мережами, а також підтримує такі завдання комп’ютерного зору, як:

- виявлення об’єктів (object detection)
- семантична та інстанс-сегментація (segmentation)
- класифікація зображень (image classification)
- відстеження об’єктів (object tracking)

Однією з ключових переваг бібліотеки є її простота у використанні та активна підтримка з боку розробників. Для запуску моделей достатньо лише кількох рядків коду, а також є можливість виводу результатів на зображеннях чи відео, з автоматичним відображенням bounding box'ів, міток та ймовірностей.

Бібліотека також підтримує експорт моделей у різні формати для подальшого використання на мобільних пристроях, вбудованих системах або в edge-комп'ютерах. Підтримувані формати включають:

Інсталяція виконується за допомогою pip

```
pip install ultralytics
```

CUDA (Compute Unified Device Architecture) — це паралельна обчислювальна платформа та програмна модель, створена компанією NVIDIA, яка дозволяє розробникам використовувати GPU (графічний процесор) не лише для обробки графіки, але й для загального призначення обчислень (GPGPU — General-Purpose computing on Graphics Processing Units). Іншими словами, це спосіб програмувати відеокарту для складних обчислювальних задач, таких як машинне навчання, симуляції, обробка сигналів, рендеринг чи обробка великих масивів даних.

CUDA надає високопродуктивне середовище для масово-паралельних обчислень, де сотні або тисячі потоків можуть працювати одночасно. Розробка відбувається за допомогою розширення мов C, C++ та Fortran, а також через API для Python (наприклад, Numba, PyCUDA, TensorFlow, PyTorch із підтримкою CUDA).

Серед основних переваг CUDA:

- Доступ до повної потужності GPU, з прямим керуванням пам'яттю, потоками та блоками обчислень.
- Прискорення програм у десятки або сотні разів у порівнянні з виконанням на CPU.
- Широка екосистема інструментів та бібліотек, таких як cuDNN, cuBLAS, cuFFT, TensorRT тощо.
- Підтримка популярних фреймворків глибокого навчання, включно з PyTorch.

CUDA є критично важливою технологією для високопродуктивних обчислень у сферах наукових досліджень, штучного інтелекту, обробки відео,

біоінформатики, фінансового моделювання та багатьох інших.

Для використання CUDA потрібна відеокарта NVIDIA з підтримкою цієї технології, а також встановлення драйверів NVIDIA, CUDA Toolkit та, за потреби, сумісних бібліотек чи SDK. Для інсталяції CUDA на Debian дистрибютивах, потрібно виконати інструкції інсталяції [10]:

```
wget https://developer.download.nvidia.com/compute/cuda/12.9.0/local_installers/cuda-repo-debian12-12-9-local_12.9.0-575.51.03-1_amd64.deb
sudo dpkg -i cuda-repo-debian12-12-9-local_12.9.0-575.51.03-1_amd64.deb
sudo cp /var/cuda-repo-debian12-12-9-local/cuda-*-keyring.gpg /usr/share/keyrings/
sudo apt-get update
sudo apt-get -y install cuda-toolkit-12-9
```

3.1.2. Підготовка клієнта

Raspberry pi працює з операційною системою Linux. Це надає низку переваг та дозволяє використовувати її у ролі повноцінного комп’ютера. Проте, це створює і декілька недоліків.

Linux не є системою реального часу. У більшості випадків це не є недоліком, а навпаки дозволяє використовувати ресурси системи не переймаючись низькорівневими процесами. Проблема постає тоді, коли потрібно керувати часово залежними процесами, такими як PWM. Сервопривід SG90 працює з PWM частотою 50Hz, повертаючи вал у залежності зі зміною прогальності. Критично важливим є те, щоб частота лишалась сталою. Будь-які шуми будуть позначатися на точності та стабільності руху валу, призводячи до випадкових хаотичних рухів. Цю проблему можна подолати за допомогою pigpio. Ця бібліотека, використовує DMA (Direct Memory Access) [11] для надзвичайно точного керування GPIO сигналами, зокрема, для програмного

PWM, що дає набагато кращу точність, ніж звичайні цикли в програмі.

Pigpio формує список інструкцій, які визначають, коли вмикати та вимикати GPIO-піни. Ці інструкції створюють хвильову форму, яка описує бажаний сигнал PWM. Замість того, щоб CPU постійно керував GPIO, pigpio передає ці інструкції DMA-контролеру, який самостійно записує значення в регістри GPIO. Це дозволяє досягти високої точності та знизити навантаження на процесор.

Для інсталяції pigpio, необхідно завантажити вихідний код та скомпілювати його:

```
wget https://github.com/joan2937/pigpio/archive/master.zip\
unzip master.zip\
cd pigpio-master
make -j4
```

Після того, активуємо бібліотеку, шляхом запуску демона від імені адміністратора.

```
sudo ./pigpiod
```

Сама бібліотека написана переважно на мові C, через її пристосованість для контролю пам’яті, але вона може без проблем взаємодіяти з Python скриптами.

Окрім керування сервомоторами, задача клієнту надсилати зображення на сервер. Для цього скористалися програмою mjpegstreamer. Це легка програма для потокової трансляції відео з камер (наприклад, USB камери або камери з CSI конектором) у форматі MJPEG (Motion JPEG) через HTTP. Завдяки своїй простоті, зображення які передаються на сервер не потребують додаткової обробки кодеками, та є одразу готовими до обробки нейромережею. Mjpegstreamer потребує встановлення залежностей та компіляції вихідного коду перед використанням [12]:

```
https://github.com/jacksonliam/mjpg-streamer
git clone https://github.com/jacksonliam/mjpg-streamer.git
sudo apt-get install cmake libjpeg8-dev
```

```
cd mjpg-streamer-experimental
make -j4
```

Після виконання цих кроків, сервер та клієнт готові до подальшої роботи

3.2. Тренування моделі.

Тренування моделі YOLO передбачає кілька ключових етапів: підготовку датасету, анотацій (міток), конфігураційного YAML-файлу, запуск тренування, та перевірку результатів[13]. YOLO використовує зображення та відповідні до них анотації. Кожне зображення має відповідний .txt файл з такою ж назвою. У цьому .txt файлі кожен рядок — це один об’єкт:

```
<class_id> <x_center> <y_center> <width> <height>
```

Всі координати нормалізовані до діапазону [0, 1]. x_center, y_center, width, height — відносні до ширини і висоти зображення.

YOLO бере інформацію про класи та директорії зображень та анотацій з .yaml файлу.

Лістинг 1 – Yolo .yaml файл

```
path: /path/to/dataset  # коренева папка з images/ і labels/
train: images/train      # шлях до тренувальних зображень
val: images/val          # шлях до валідаційних зображень
nc: 1                    # кількість класів
names: ['UAV']    # список класів
```

Метапараметри тренування моделі задаються у python файлі, який використовується для тренування. Спочатку імпортуємо бібліотеку ultralytics та обираємо моделі нейромережі, на основі якої відбуватиметься тренування.

Лістинг 2 – Імпорт бібліотеки та задання моделі

```
from ultralytics import YOLO
model = YOLO("yolov8n.yaml")
```

Далі, задаємо метапараметри (data – шлях до YAML-файлу, що описує датасет, epochs – кількість епох, imgsz – розмір зображень, які будуть

використовуватися під час тренування, device – використовується GPU з індексом 0) та запускаємо процес тренування.

Лістинг 3 – тренування та метапарпметри

```
train_results = model.train(
    data="drone_detect_v1.yaml",
    epochs=100,
    imgsz=400,
    device=0,
)
```

На останок, виконується оцінка якості моделі на валідаційному наборі даних, що вказаний у YAML-файлі.

Лістинг 4 – Валідація моделі

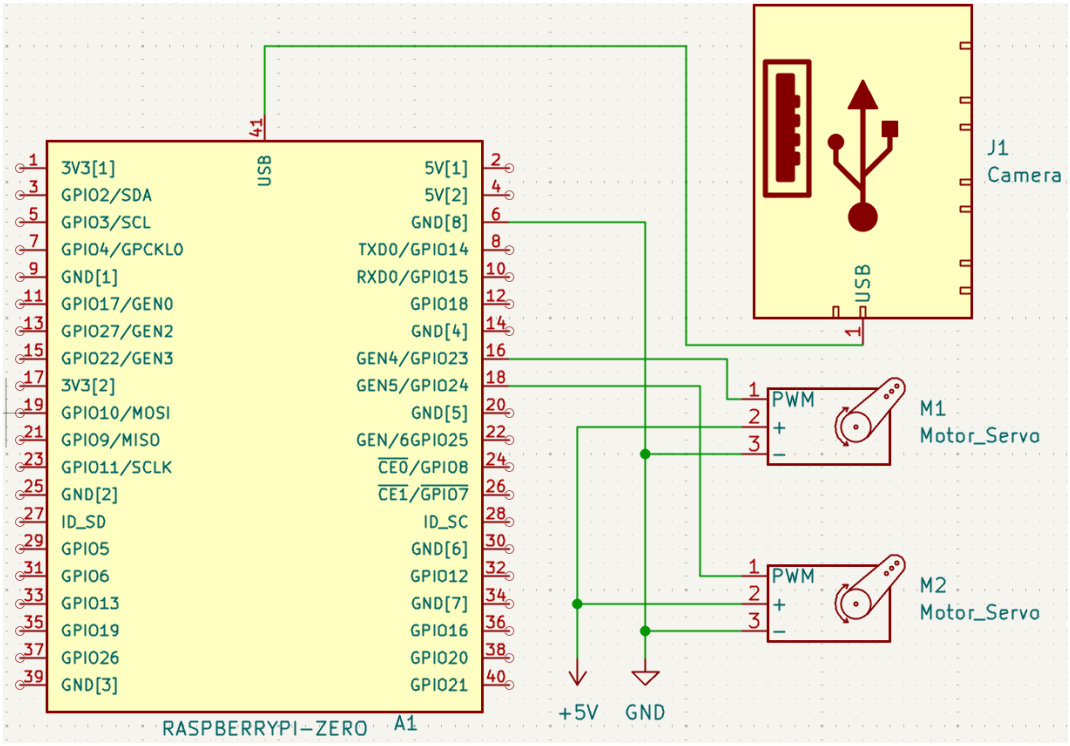
```
metrics = model.val()
```

3.3. Створення схеми та під’єднання елементів турелі.

Оскільки обмін інформацією між сервером та клієнтом здійснюватиметься за допомогою бездротового зв’язку, розглянемо схему під’єднання елементів турелі. Саме клієнт здійснюватиме контроль над сервомоторми. Бібліотека pigpio дозволяє використовувати будь-які піни у ролі PWM виводів, тож ми можемо обирати ті піни, які нам зручно. Важливим питанням є живлення сервопривода та клієнта. Електродвигуни (до яких належать сервоприводи) створюють індуктивне навантаження або «індуктивність» на джерелі живлення, через фізику роботи їхніх електродвигунів і пов'язану з цим електромагнітну інерцію. Наприклад, коли серво різко стартує або зупиняється струм через обмотку швидко змінюється, а індуктивність намагається зберегти струм незмінним. В результаті виникають піки напруги або «зворотні імпульси» (індуктивні перенапруги). Це навантажує джерело живлення і може створювати електромагнітні завади або падіння напруги. Ці явища призводять до перезавантаження клієнта, та можуть призвести до його поломки. Існують різні

способи протидії цим ефектам, такі як використання конденсатора, діода Шотткі, або RC фільтри. Проте, найбільш надійним способом є використання різних джерел напруги. Проте, треба мати на увазі необхідність об'єднати землю сервопривода та клієнта, інакше отримаємо хаотичні рухи валу.

Камера під'єднується за допомогою USB (рис. 3.1). Також, можна використовувати CSI інтерфейс, який, скоріше за все, знайде використання при застосуванні більш якісної камери. Більша версія схеми у ДОДАТКУ А.



Для створення опорно-рухової частини використовувалася параметрична CAD-програма FreeCAD. Це вільна програма з відкритим вихідним кодом, яка дозволяє створювати тривимірні моделі з детальним контролем геометричних розмірів, відношень та залежностей між елементами. Однією з головних переваг FreeCAD є підтримка параметричного моделювання — завдяки цьому користувач може легко змінювати вихідні параметри (такі як розміри, кути, положення елементів) і програма автоматично перебудовує модель відповідно до внесених змін. Це значно спрощує процес внесення правок та дозволяє оперативно адаптувати конструкцію під нові вимоги.

FreeCAD має модульну архітектуру, що дає змогу розширювати

функціональність за допомогою додаткових робочих середовищ (Workbench). Зокрема, доступні середовища для роботи з твердотільним моделюванням, поверхнями, кінематикою, аналізом навантажень методом скінчених елементів (FEM), створення креслень, трасування плат (PCB) та навіть для моделювання архітектурних об'єктів. Це робить FreeCAD універсальним інструментом, придатним для широкого спектра інженерних та технічних задач.

Іншою важливою перевагою є підтримка численних форматів обміну даними, таких як STEP, IGES, STL, OBJ, DXF, SVG та інші. Зокрема, після завершення проєктування деталі її можна експортувати у STL-формат, який є стандартом для 3D-друку. Це дозволяє безпосередньо передати модель до слайсера та виготовити фізичний прототип на 3D-принтері.

FreeCAD також підтримує інтеграцію з мовою програмування Python, що відкриває можливості для автоматизації рутинних процесів, створення власних макросів, генерації параметричних моделей за заданими алгоритмами та виконання складних обчислень у середовищі моделювання. Завдяки відкритому коду, спільнота розробників постійно оновлює програму, додає нові можливості та виправляє помилки, що гарантує її стабільний розвиток і відповідність сучасним вимогам.

Таким чином, використання FreeCAD при розробці опорно-рухової частини дало змогу швидко, гнучко та точно створити тривимірну модель, адаптувати її до вимог конструкції та підготувати до виготовлення на 3D-принтері без необхідності використання комерційного ПЗ.

За допомогою FreeCAD було створено наступні деталі:

- контейнер для комплектуючих RPI та живлення
- кришку контейнера
- з’єднувач між сервоприводами
- кріплення камери

Контейнер та кришка, створювалися з активним використанням параметричного дизайну. Основною причиною цьому було те, що на момент

створення дизайну, не було відомо розміри живлення, яке буде сховано у контейнері. Було створено таблицю, у якій зазначено всі параметри деталі (рис.

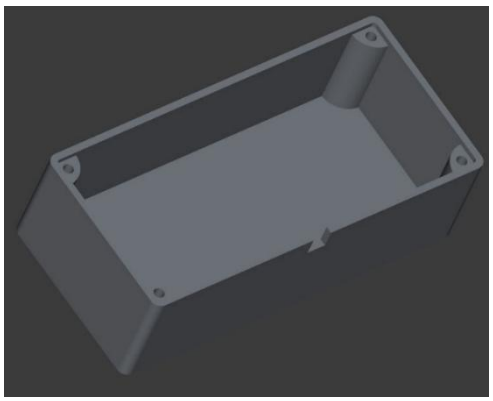
Content:

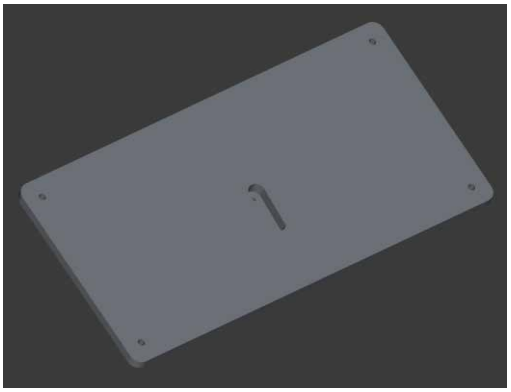
	A	B	C	D
1	BOX BODY	STATS		
2	box_width	190		
3	box_length	90		
4				
5	box_height	56		
6	inside_radius	2	CALCED	
7	outside_radius	5	CALCED	
8				
9	wall_thickness	3		
10	hole_dia	5,10		
11	lug_height	50	CALCED	
12	lug_radius	12,75	CALCED	
13	hole_center_length	36,90	CALCED	
14	hole_center_width	86,90	CALCED	
15	box_inside_length	84	CALCED	
16	box_inside_width	184	CALCED	
17				
18	clearance_sides	0,50		
19	clearance_hole	3		

Start | Turret_with_spreadsheet : 1 | Spreadsheet

3.2).

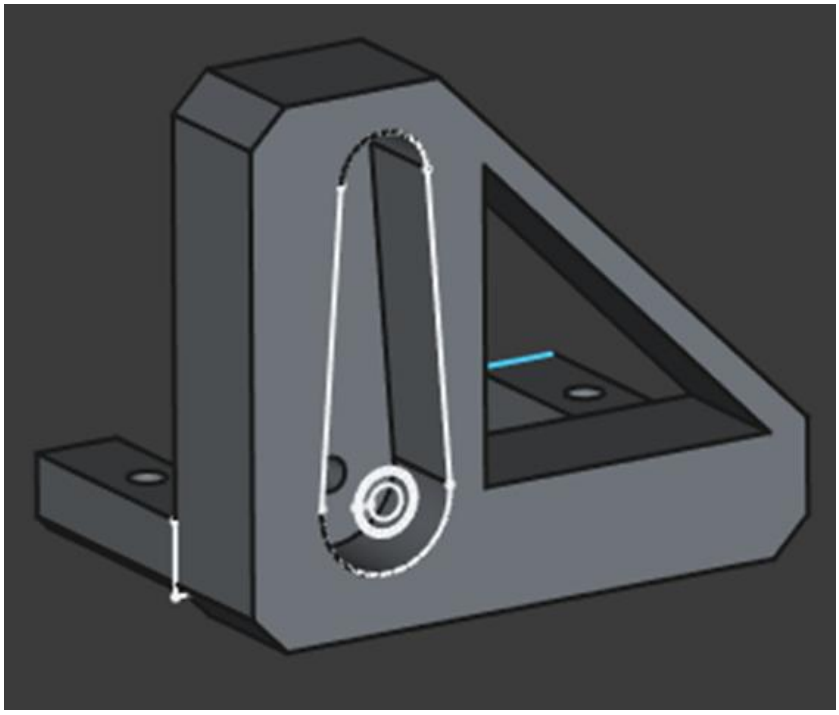
Кришка та контейнер мають закеруглені сторони та отвори для гвинтів, що утримуватимуть їх разом. Кришка має паз для валу сервомотора (рис. 3.3) (рис 3.4).



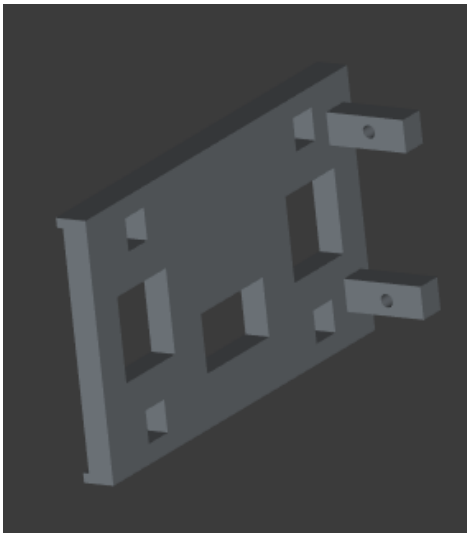


З’єднання між сервоприводами використовується для об’єднання

вертикального та горизонтального сервопривода. Воно має отвір для валу та кріплення до корпусу сервоприводу (рис. 3.5).



Остання деталь використовується для кріплення камери до решти турелі.



Вона має кріплення до корпусу сервомотору, отвори для стяжок та отвір для USB кабелю (рис. 3.6).

Після зборки, турель виглядає наступним чином (рис. 3.7):

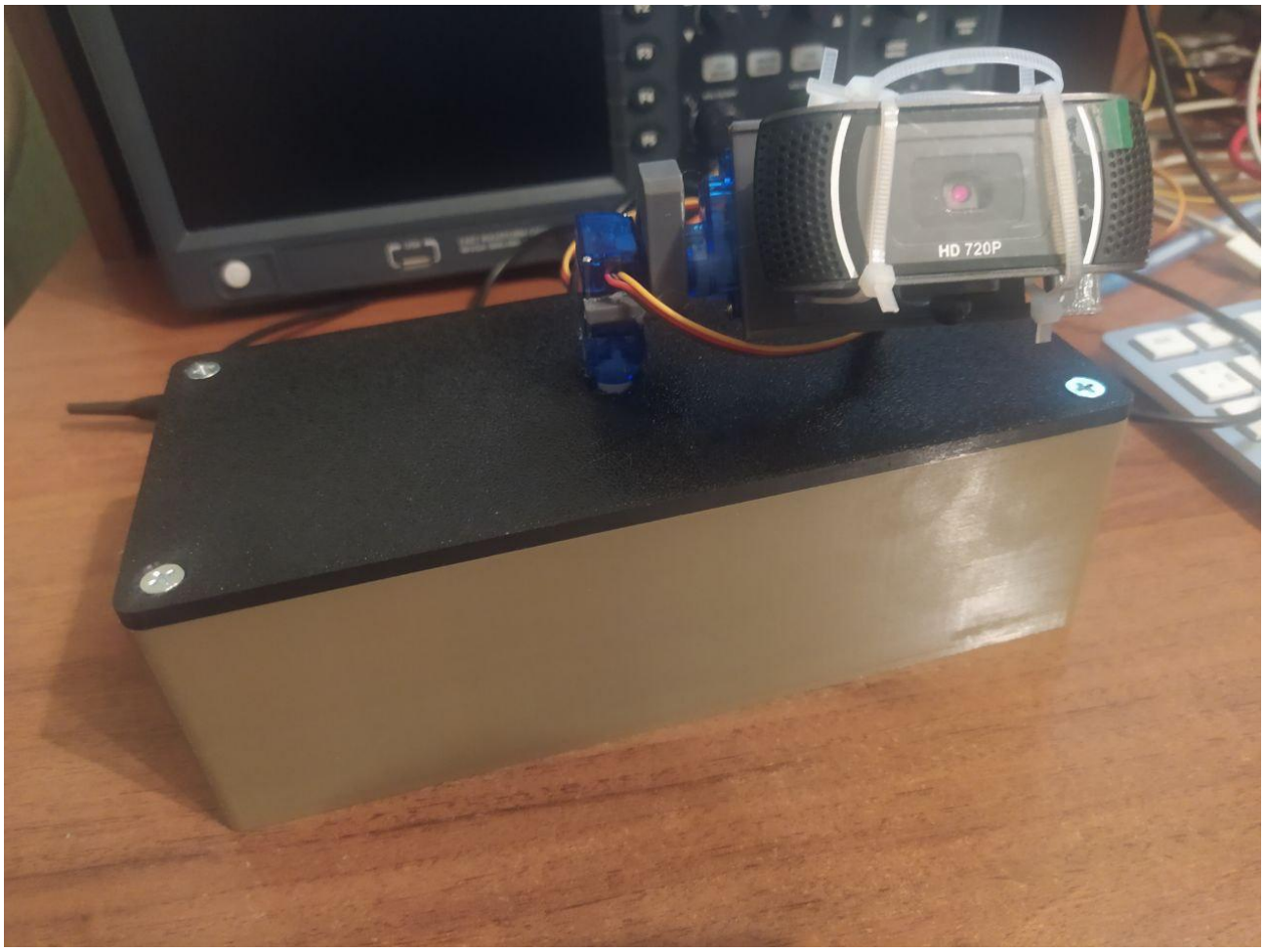


Рис. 3.7 – Зібраний макет

3.4. Програмування макету

Архітектура системи покладається на використання клієнту для керування сервоприводами та передачі відеопотоку, та сервера для корегування рухом базуючись на вмісті обробленого зображення. Для цього використовується вбудована бібліотека socket. Обробка та надсилання пакетів, які ми раніше розглянули, є не єдиною задачею системи. Для того, щоб обидві складові системи робили більше, ніж просто чекати на пакети, необхідно використовувати багатопотоковість. Вбудована бібліотека threading, дозволяє це імплементувати. Хоч і не дійсна багатопотоковість (не використовує декілька ядер), вона тим не менш дозволяє виконувати інші частини програми, коли, наприклад, інша програма очікує виконання блокуючої операції.

3.4.1. Програмування сервера

Перш за все необхідно, налагодити socket комунікацію. Імпортуємо бібліотеки та задаємо глобальні змінні.

Лістинг 5 – Імпорт бібліотек серверу

```
import time
import socket
import threading
from time import sleep
import cv2
```

Лістинг 6 – Глобальні змінні.

```
HEADER = 64 # Розмір першого повідомлення у байтах
PORT = 5050 # Порт
SERVER = "0.0.0.0"  # IP сервера. Слухає всі IP-інтерфейси
ADDR = (SERVER, PORT)
DISCONNECT_MESSAGE = "-=DISCONNECT=-" # Команда від’єднання
FORMAT = "utf8" # формат кодування повідомлення
```

Створемо новий сервер та прив’яжемо до нього адресу та порт

Лістинг 7 – Ініціалізація серверу

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)
```

Далі, створили функцію старту сервера. Метод socket.accept() є блокуючим [14]. Тоді, коли з’являється нове з’єднання, створюється потік handle_client та виводиться кількість з’єднань на основі кількості активних потоків.

Лістинг 8 – Функція старту сервера

```
def start_server():
    server.listen()
    print(f"[LISTENING] Sever is listening on {SERVER}")
    while True:
```

```
        conn, addr = server.accept()
        threading.Thread(target=handle_client, args=(conn,
addr)).start()
        print(f"[ACTIVE CONNECTIONS] {threading.active_count() -
1}")
```

Для коректного та стабільного отримання інформації, пристрої мають оброблювати певну кількість байтів. Метод socket.recv(n) не гарантує, що одразу поверне n байтів. Він може повернути менше, наприклад, через затримки в мережі або фрагментацію TCP-пакетів. Тому, якщо треба надіслати точну кількість байтів (наприклад, довжину повідомлення або саме повідомлення) — потрібно викликати recv() кілька разів.

Лістинг 9 – Функція отримання точної кількості байтів

```
def recv_all(conn, length):
    data = b''
    while len(data) < length:
        packet = conn.recv(length - len(data))
        if not packet:
            return None  # connection closed
        data += packet
    return data
```

Функція handle_client(conn, addr) — це обробник клієнтських підключень. Вона запускається в окремому потоці для кожного нового клієнта, який з'єднується з сервером. Також, стартуємо окремий фоновий потік для керування туреллю. Переводимо аргумент daemon у True. Це дозволить закривати процес, навіть якщо turret_control виконується

Лістинг 10 – Створення потоку turret_control у функції handle_client()

```
def handle_client(conn, addr):
     threading.Thread(target=turret_control, daemon=True,
args=(conn, )).start()
```

Входимо у цикл отримання повідомлень. Чекаємо, поки клієнт надішле заголовок (довжину повідомлення). Якщо клієнт розірвав з'єднання, то відбудеться вихід із циклу. Перетворюємо заголовок на число. Якщо це не вдається, то сервер завершує обробку клієнта. Далі, зчитується саме повідомлення довжини msg_length, декодується, перевіряється на DISCONNECT_MESSAGE, виводиться в консоль.

Лістинг 11 – виведення та обробка повідомлень від клієнта

```
    connected = True
    while connected:
        msg_length_raw = recv_all(conn, HEADER)
        if not msg_length_raw:
            break  # Connection closed

        try:
            msg_length =
int(msg_length_raw.decode(FORMAT).strip())
        except ValueError:
            print(f"Failed to parse message length:
{msg_length_raw}")
            break

        msg = recv_all(conn, msg_length)
        if not msg:
            break  # Connection closed

        msg = msg.decode(FORMAT)
        if msg == DISCONNECT_MESSAGE:
            connected = False
        print(f"[{addr}] {msg}")
    conn.close()
```

Функція send_message(conn, msg) відповідає за відправку повідомлення по

TCP-сокету з використанням спеціального протоколу, який починається з заголовка (header), що вказує довжину повідомлення. Далі, текстове повідомлення msg перетворюється у байти за допомогою кодування UTF-8. Потім, визначається кількість байтів у закодованому повідомленні (тобто довжину повідомлення). Число msg_length перетворюється у рядок, а потім кодується у байти. Далі, додаємо пробіли, щоб заголовок займав точно HEADER байтів (64 байти за замовчуванням). Надсилає заголовок з довжиною повідомлення після заголовка, та надсилається саме повідомлення (в байтах).

Лістинг 12 – Функція надсилання повідомлення

```
def send_message(conn, msg):
    message = msg.encode(FORMAT)
    msg_length = len(message)
    send_length = str(msg_length).encode(FORMAT)
    send_length += b' ' * (HEADER - len(send_length))
    conn.send(send_length)
    conn.send(message)
```

Тепер, перейдемо до контролю турелі. Функція compute_errors() використовується для обчислення нормалізованої помилки між центром кадру та координатами об'єкта (наприклад, обличчя), яке потрібно відслідковувати.

Лістинг 11 – функція обрахунку помилки турелі

```
def compute_errors(x_obj, y_obj, frame_width, frame_height):
    error_x = (frame_width / 2 - x_obj) / (frame_width / 2)  #
-1 to 1
    error_y = - (frame_height / 2 - y_obj) / (frame_height /
2)  # -1 to 1
    return error_x, error_y
```

update_servo_position() реалізує пропорційне регулювання для керування положенням сервопривода на основі помилки позиціонування. Можна змінювати значення аргументу gain для контролю швидкості повертання.

Лістинг 12 – Функція обрахунку нового положення

```
def update_servo_position(current_value, error, gain=0.0001):
    # Proportional control
    delta = error * gain
    new_value = current_value + delta
    # Clamp to [-1.0, 1.0]
    new_value = max(-1.0, min(1.0, new_value))
    return new_value
```

yolo_detect() — це генератор, який захоплює відеопотік, обробляє кожен кадр за допомогою YOLOv8-моделі, визначає об’єкти (наприклад, обличчя або людей), і повертає координати виявлених об'єктів.

```
from ultralytics import YOLO

def yolo_detect():
    model = YOLO("best.pt")
    capture =
cv2.VideoCapture("http://192.168.0.104:8080/?action=stream")

    if not capture.isOpened():
        print("[ERROR] Cannot open video stream")
        return

    while True:
        ret, frame = capture.read()
        if not ret:
            print("[WARNING] Failed to read frame")
            yield []
            continue

        results = model(frame)[0]
        UAVs = []

        for box in results.boxes:
```

```
                cls = int(box.cls[0])

                x1, y1, x2, y2 = map(int, box.xyxy[0])
                w = x2 - x1
                h = y2 - y1
                cx = x1 + w // 2
                cy = y1 + h // 2
                UAVs.append((cx, cy, w, h))
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255,
0), 2)
            cv2.imshow("YOLO Detection", frame)
            cv2.waitKey(1)
            yield UAVs
```

turret_control() це потік керування туреллю (сервоприводами X і Y), яка намагається слідкувати за об’єктом на відео. Спочатку задаються параметри початкового положення сервоприводів та розміру зображення. Створюємо об’єкт розпізнавання об’єктів. Якщо розпізнано об’єкт, то виконуємо алгоритм слідування.

Лістинг 13 – Керування камерою коли розпізнано ціль

```
def turret_control(conn):
    positionX = 0
    positionY = 0.75
    step = 0.05
    increasing = True
    frame_width = 1280
    frame_height = 720
    obj_gen = obj_detect()  # Get the generator
    last_search_time = time.time()
    search_interval = 0.5  # seconds
    while True:
        try:
            objs = next(obj_gen)
```

```
except StopIteration:
    print("[INFO] Video stream ended.")
    break
except Exception as e:
    print(f"[ERROR] obj_detect() failed: {e}")
    continue
if obj:  # If objs detected, use tracking mode
    for x_obj, y_obj, w, h in obj:
        error_x, error_y = compute_errors(
            x_obj, y_obj, frame_width, frame_height)
        # Update servos proportionally
        positionX = update_servo_position(
            positionX, error_x, gain=0.05)
        positionY = update_servo_position(
            positionY, error_y, gain=0.05)
        print(f"[SERVO] error X {error_x}")
        print(f"[SERVO] error Y {error_y}")
```

Якщо об’єкт не знайдено, то виконується алгоритм пошуку. Турель обертається до моменту знаходження цілі. Сервопривід рухається не постійно, а з інтервалами 0.5с з метою спрощення розпізнавання об’єкту. Інтервал задається таймером, щоб уникнути блокуючої операції sleep.

Лістинг 14 – Алгоритм пошуку цілі

```
else:
    now = time.time()
    if now - last_search_time >= search_interval:
        last_search_time = now
        if increasing:
            positionX += step
            if positionX >= 1.0:
                positionX = 1.0
                positionY = 0.75
                increasing = False
        else:
```

```
                positionX -= step
                if positionX <= -1.0:
                    positionX = -1.0
                    positionY = 0.75
                    increasing = True
```

Врешті решт, розраховані координати округлюються, переводяться у символьний формат та надсилаються на клієнт.

Лістинг 15 – Відправлення інструкцій повороту турелі

```
            msgX = str(round(positionX, 2))
             msgY = str(round(positionY, 2))
             try:
                 send_message(conn, f"servo_X {msgX}\nservo_Y
{msgY}")
             except OSError as e:
                 print(f"[TURRET ERROR] {e}")
                 break
```

Повна версія коду сервера у ДОДАТКУ Б

3.4.2. Програмування клієнту

Задача клієнту, отримувати інструкції повороту сервоприводів. Імпортуємо бібліотеки

Лістинг 16 – імпорт бібліотек клієнта

```
import socket
import threading
from gpiozero import Servo
from time import sleep
from queue import Queue
```

Лістинг 17 – Додання глобальних змінних клієнта

```
HEADER = 64 # Розмір першого повідомлення у байтах
PORT = 5050 # Порт
SERVER = "192.168.0.105"  # IP сервера. Слухає всі IP-
```

```
інтерфейси
ADDR = (SERVER, PORT)
DISCONNECT_MESSAGE = "-=DISCONNECT=-" # Команда від’єднання
FORMAT = "utf8" # формат кодування повідомлення
```

Керування сервомоторів здійснюється шляхом задання тривалості високого рівня сигналу. Під час ініціалізації об’єктів, ми вказуємо пін, тривалість між сигналами та тривалість максимального та мінімального сигналу. У подальшому, зможемо керувати прогальністю сигналу, вказуючи значення від -1 до 1, які будуть нормалізовані у заданому під час ініціалізації проміжку.

Лістинг 18 – Ініціалізація сервоприводів

```
servo_X = Servo(pin=23,
                min_pulse_width=0.0005,  # 0.5 мс (повністю
вліво)
                max_pulse_width=0.0025, # 2.5 мс (повністю
вправо)
                frame_width=0.02)       # 20 мс між сигналами
(частота 50 Гц)
servo_Y = Servo(pin=24,
                min_pulse_width=0.0005,  # 0.5 мс (повністю
вліво)
                max_pulse_width=0.0025, # 2.5 мс (повністю
вправо)
                frame_width=0.02)       # 20 мс між сигналами
(частота 50 Гц)
```

Цей код використовує ідентичні функції send_message() та recv_all(). Відмінність у мережевому алгоритмі полягає у отриманні повідомлень від сервера. Суть полягає у безперервному отриманні інструкцій та безперервному керуванні сервомоторами. Використання глобальних змінних може призвести до пошкодження даних під час отримання та додавання інформації. Виходом є https://docs.python.org/3.12/library/queue.html
використання черг. У Python модуль queue надає зручні та потокобезпечні

структури даних черг. Він особливо корисний у багатопоточному програмуванні, де потрібно синхронізувати доступ до спільних ресурсів. Черги автоматично блокуються при додаванні або витягуванні елементів, що запобігає конфліктам між потоками. Обидва потоки запускаються чергою у ролі аргументу:

Лістинг 19 – Ініціалізація черги та створення потоків контролю турелі та отримання повідомлень

```
q = Queue()
threading.Thread(target=turret_control, daemon=True,
args=(q,)).start()
threading.Thread(target=receive, daemon=True,
args=(q,)).start()
```

Лістинг 20 – функція отримання повідомлень та їх відправки у чергу

```
def receive(q_msg):
        while True:
            msg_length_raw = recv_all(client, HEADER)
            if not msg_length_raw:
                break  # Server closed connection
            try:
                msg_length =
    int(msg_length_raw.decode(FORMAT).strip())
            except ValueError:
                print(f"Failed to parse message length:
    {msg_length_raw}")
                break
            msg = recv_all(client, msg_length)
            if not msg:
                break
            q_msg.put(msg.decode(FORMAT))
```

Функція контролю рухів турелі дістає повідомлення з черги. У випадку

отримання повідомлення у форматі інструкції, розбиває його на 2 значення, які використовуються для обозначення положеня валів сервоприводів.

Лістинг 21 – Функція контролю руху турелі

```
def turret_control(q_msg):
    positionX = 0
    positionY = 1
    servo_X.value = positionX
    servo_Y.value = 0

    while True:
        msg = q_msg.get()
        print(f"Turret patrol msg: {msg}")
        msg_list = msg.split()
        if msg_list[0] == "servo_X":
            print(msg_list)
            positionX = float(msg_list[1])
            positionY = float(msg_list[3])
            # send(f"servo_X = {servo_X.value}\nservo_Y =
{servo_Y.value}")
        servo_X.value = positionX
        servo_Y.value = positionY
```

Повна версія коду клієнта у ДОДАТКУ В

РОЗДІЛ 4
ТЕСТУВАННЯ

4.1. Демонстрація роботи системи

Для демонстрації роботи системи, необхідно запустити програму обробки запитів та зображення на сервері. Необхідно використовувати те Pyenv середовище, яке використовували під час інсталяції бібліотек.

```
Python Server_processing.py
```

Далі, на клієнті запускаємо програму надсилання відеопотоку.

```
./mjpg_streamer -i "./input_uvc.so -d /dev/video0 -r 1280x720
-f 60" -o "./output_http.so -w ./www"
```

Також, необхідно запустити бібліотеку контролю PWM від імені адміністратора.

```
sudo ./Projects/Sources_to_comp/pigpio-master/pigpiod
```

Нарешті, запускаємо програму обробки мережевих повідомлень та контролю сервомоторів. Перед запуском інтерпретатора, визначаємо змінну середовища, з метою обрання бібліотеки керування PWM:

```
GPIOZERO_PIN_FACTORY=pigpio  python Client_processing.py
```

При виконанні з’єднання, турель починає пошук цілі, а на сервері з’являється вікно, у якому відображається відео з камери. При знаходженні цілі, вона обводиться прямокутником (рис. 4.1), та камера починає слідувати за нею. Якщо камера надто повільно повертається, під час слідування за ціллю, можемо змінити атрибут gain у скрипті сервера. Також, під час роботи системи, турель надсилає діагностичну інформацію на сервер, яка відображається у командному рядку. При виявленні цілі, сервер відображає помилку положення цілі, та наступну обраховану координату. Це дозволяє зберігати log виявлення, (наприклад за допомогою утиліти tee), для подальшого покращення алгоритму наведення. При завершенні від’єднанні клієнта від сервера, клієнт надсилає

команду -=Disconnect=-, що закриває мережеве з’єднання.

Рис. 4.1 – Вікно з відео (placeholder)

4.2. Можливі покращення системи

Ця система має значну кількість складових компонентів, як програмних так і фізичних. У цьому розділі розглянемо проблеми, з якими довелося мати справу під час розробки та тестування макету.

4.2.1. Зручність використання

На цьому етапі розробки системи, вона має лише текстовий інтерфейс ініціалізації та діагностики. Це призводить до значного часу початкової ініціалізації після перезавантаження системи. Зручний інтерфейс роботи з сервером та клієнтом спростить роботу та зекономить час бійців вогневої групи. Складність цієї реалізації полягає у тому, що нема гарантії, що хоч один з елементів системи матиме монітор або екран. Клієнт – орієнтований на автономну роботу, та матиме RPI у ролі міні комп’ютера. Сервер – може бути, будь чим: потужним (або не дуже) ноутбуком, модулем Nvidia Jetson або взагалі платою на основі FPGA, на якій реалізовано енергоефективні нейроакселератори. Тому, єдиний вихід, який збереже основну перевагу

розробленої системи – масштабованість, це використання мобільного інтерфейсу. Він може бути реалізований на смартфоні або планшеті, та мати розширені, у порівнянні з наявним інтерфейсом можливості, такі як:

- початок пошуку цілі;
- ручне керування тureллю;
- виконання калібрації сервомоторів турелі;
- відображення діагностичної інформації
- відображення відео з турелі.

Для реалізації цих можливостей, необхідно розширити серверний та клієнтський код; налаштувати систему таким чином, щоб ініціалізація системи починалася одразу після запуску обладнання. Також, багато роботи буде виконано під час розробки мобільного застосунку. Незважаючи на це, таке рішення є правильним, якщо система матиме перспективу застосування на практиці, особливо зважаючи на те, що підрозділи чи не завжди мають доступ до планшетів з ОС Android.

4.2.2. Проблеми розпізнавання.

Комп’ютерний зір не є безвідмовним. Під час розпізнавання зображення, модель повертає координати об’єкта та впевненість у відсотках у тому, що цей об’єкт дійсно є тим, що ми шукаємо. Часто, модель не розпізнає ціль, особливо, якщо зображення цілі невелике, або якщо вона знаходиться на складному фоні. Питання фону вирішується саме по собі, за рахунок того, що небо не є надто візуально навантаженим простором. Проте, проблема з розпізнаванням малих об’єктів є серйознішою. Мобільні вогневі групи, мають розпізнавати ціль на відстані 1500 метрів. Модель YOLO була обрана через її виняткову швидкість в обробці зображень у реальному часі. Хоч проблема з малими об’єктами доволі розповсюджена з цією моделлю, проте вона однак є оптимальним варіантом для поставленої задачі. Інші моделі не пропонують ті переваги які надає YOLO, а

створення власної моделі для цієї задачі є непосильною задачею без значного досвіду та фінансування.

Перший спосіб вирішення цієї проблеми, це створення кращого датасету, який буде на розпізнавання малих об’єктів. Це найпростіше рішення, хоч воно і може мати небажані наслідки. Модель мусить працювати як з цілями, які знаходяться близькій відстані, так і з тими, що знаходяться на більш віддаленій. Ознаки, які будуть отримані з цих цілей, можуть надто відрізнятися, погіршуючи якість моделі, зменшуючи точність, збільшуючи кількість хибних спрацьовувань.

Другий спосіб, це модифікація існуючої моделі. Під час тренування моделі для цієї роботи, використовувалася стандартна модель YOLOv8. Спробою оптимізації моделі під поставлені задачі, може бути зміна Neck моделі. Як зазначалося раніше, Neck об’єднує ознаки з різних шарів мережі. Змінивши пріоритет ознак з ранніх шарів, (менш багатих на семантику, але краще вказуючих положення об’єкта) можемо отримати краще розпізнавання. Цей спосіб набагато складніший у реалізації, але, у перспективі, здатний значною мірою покращити працездатність моделі під час роботи з малою кількістю класів та дрібними об’єктами на простому фоні.

4.2.3. Апаратна складова

Цей дипломний проект націлений на розробку прототипу, який буде використовуватися у роді перевірки концепту та як стенд для подальшої експериментації та розробки. Проміжною метою є перевірка доцільності та працездатності технічного рішення. Кінцевою, якщо і доволі амбітною метою, є упровадження повноцінної системи у бойову роботу підрозділів. Якщо подальша розробка та доопрацювання системи матиме позитивні результати, можна буде перейти до повноцінного прототипу, придатного до польових випробувань. У

своєму кінцевому вигляді, система може мати повноцінний повнорозмірний корпус, здатний утримувати високоякісну камеру з цифровим нічним баченням та дуже значною роздільною здатністю. Також, кінцевий прототип матиме високоякісні потужні сервоприводи, що забезпечуватимуть стабільний та надійний рух турелі.

Чи не найбільш важливим питанням, є обрання оптимального сервера для обробки зображення за допомогою нейромережі. Зі збільшенням роздільної здатності камери, необхідно збільшувати роздільну здатність мережі. Зі збільшенням роздільної здатності мережі, збільшуватиметься ширина шарів, та, очевидно, кількість обрахунків. Щоб підтримувати час обробки одного зображення у межах, які можна буде назвати “обробкою у реальному часі”, потрібно використовувати більші обчислювальні потужності. Модель, яка могла працювати на ноутбуці, зі збільшенням ширини мережі, буде оброблювати відеопотік, який більше нагадуватиме слайд шоу, а не відео. Зрозуміло, що такий формат не дозволить коректно знаходити помилку положення камери. Альтернативою може бути Nvidia Jetson Nano, хоч і це рішення може мати короткотерміновий ефект. Важливим моментом є також живлення сервера, оскільки сучасні GPU можуть бути дуже вимогливими до енергозабезпечення. Це потребуватиме або великого переносного акумулятора (по типу EcoFlow), або генератора. Зрозуміло, що чим меншим буде енергоспоживання, тим менше доведеться перейматися додатковими елементами живлення.

Найдешевшим та найзручнішим рішенням, може бути використання віддаленого сервера. Переміщення сервера у віддалену локацію, можливо, у одне з міст, у якому нема проблем з електропостачанням. Це дозволить зменшити логістичні витрати та усуне потребу обслуговувати сервер. На перший погляд, це оптимальне рішення, та у цивільному житті воно було б найбільш доцільним. Проте, специфіка використання та умови ведення бойових дій вноситимуть свої корективи. Для комунікації з віддаленим сервером доведеться використовувати інтернет з’єднання. В залежності від місця дислокації МВГ можна

використовувати різні способи доступу до інтернет мережі.

У містах, далеких від лінії фронту, можна використовувати мобільний інтернет. В залежності від міста, можна отримати швидкості, придатні навіть для відсилання відеопотоку високої роздільної здатності (особливо з використанням кодеків стискання відео). Проблеми постають у той час, коли починається глушіння мобільного зв'язку. Я не можу розкрити специфіку, або причину, з якої виконуються такі дії, проте це траплялося неодноразово. У цьому випадку, система виявлення не функціонуватиме будь яким чином.

Другий спосіб, придатний як для міських, так і для польових умов, це використання Starlink. Незважаючи на той факт, що не у всі підрозділи мають цю систему супутникового зв'язку, вона, на сьогодні є основним способом зв'язку та комунікації у війську. Хоч і доволі стійка до загального глушіння за допомогою засобів РЕБ, Starlink вразливий до спуфінгу: явища, коли ворог підроблює метадані та надсилає інформацію на антену, видаючи себе за супутник. Це призведе до втрати з'єднання, та знову ж таки, втрати дієздатності системи виявлення.

Ще один недолік віддаленого сервера, це затримки (ping). Надсилання даних через інтернет, на сервер, який може знаходитися за сотні кілометрів від турелі, неминуче призведе до декількох додаткових мілісекунд затримки, яких би не було, якби сервер знаходився у локальній мережі. Затримки можуть бути збільшеними, у випадку, якщо Starling знаходиться у зоні дії заглушаючого РЕБ.

Одним із найцікавіших технічних рішень для прискорення нейронних мереж може бути реалізація на FPGA. Перенесення обчислень нейромережі на програмовану логіку дозволяє гнучко адаптувати архітектуру до конкретної задачі. Це забезпечує як пришвидшення обробки, так і суттєве зменшення енергоспоживання. На перший погляд, це виглядає як ідеальне рішення — особливо для задач з обмеженими ресурсами або жорсткими вимогами до затримки.

Втім, у цивільному житті така гнучкість може виявитись надмірною. FPGA

складні у розробці — навіть за наявності готових моделей, їх оптимізація під специфіку конкретної мікроархітектури вимагає високого рівня кваліфікації. Крім того, час розробки значно довший у порівнянні з використанням CPU або GPU, де готові фреймворки беруть на себе більшу частину складності.

У процесі подальшого доопрацювання системи, може виникнути ситуація, у наслідку якої реалізація за допомогою FPGA буде єдиним доцільним варіантом, з причин детально розглянутих раніше. У будь якому випадку, ця технологія на сьогодні є не надто популярною, але тою, що має великий потенціал. Реалізація системи виявлення БПЛА на програмованій логіці, потребує детальнішого дослідження, та може бути саме тим, що дозволить їй досягти того рівня ефективності та продуктивності, що по справжньому дозволить їй бути придатною для бойового застосування.

ВИСНОВКИ

Результатом виконання роботи, стала розробка та реалізація системи виявлення БПЛА. Було отримано інформацію про створення цієї системи оцінено перспективи подальшої реалізації

У першому розділі, розглянуто різні алгоритми машинного навчання. Серед традиційних алгоритмів, було досліджено HOG виокремлення ознак. Також, було розглянуто згорткові нейромережі, виконано порівняння їх з традиційними методами машинного навчання. Було розглянуто основні принципи роботи найпопулярніших моделей виявлення зображень та обрано ту, що найкраще підходить для роботи у реальному часі.

У другому розділі, розглянуто архітектуру системи. Описано основні структурні елементи та способи їх взаємодії, зазначено основні етапи розробки та алгоритм поводження турелі. Було описано обладнання, яке використовувалося під час побудови макету.

У третьому розділі, було описано кроки, необхідні для підготовки сервера та клієнта до початку розробки та виконання програмного забезпечення. Вказано інструкції, щодо інсталяції бібліотек та утиліт. Розглянуто основні моменти, на які треба звернути увагу під час під’єднання електроніки турелі. Розглянуто CAD редактор FreeCAD, його переваги та можливості. Надано спосіб його використання та відображено опорно рухові елементи, розроблені з його допомогою. Висвітлено готову фізичну модель. Після того, описано алгоритм, який виконується на сервері та клієнті. Надано пояснення використаних програмних рішень.

У четвертому розділі, було описано алгоритм переведення сервера та турелі у робочий стан; розглянуто можливі способи покращення. Описано, яким чином можна покращити зручність використання за допомогою смартфону або планшету, розглянуто способи покращення роботи макету за допомогою оптимізації моделі нейромережі виявлення образів та використання інших

апаратних рішень.

Виконання поставленого завдання показало, що створення системи виявлення об’єктів у реальному часі, потребує значних матеріально технічних витрат та глибокої експертизи у обраній сфері діяльності. Данна робота – це перша сходинка, яка дозволила оцінити перспективи, побачити недоліки реалізації, та націлитися на способи їх вирішення.

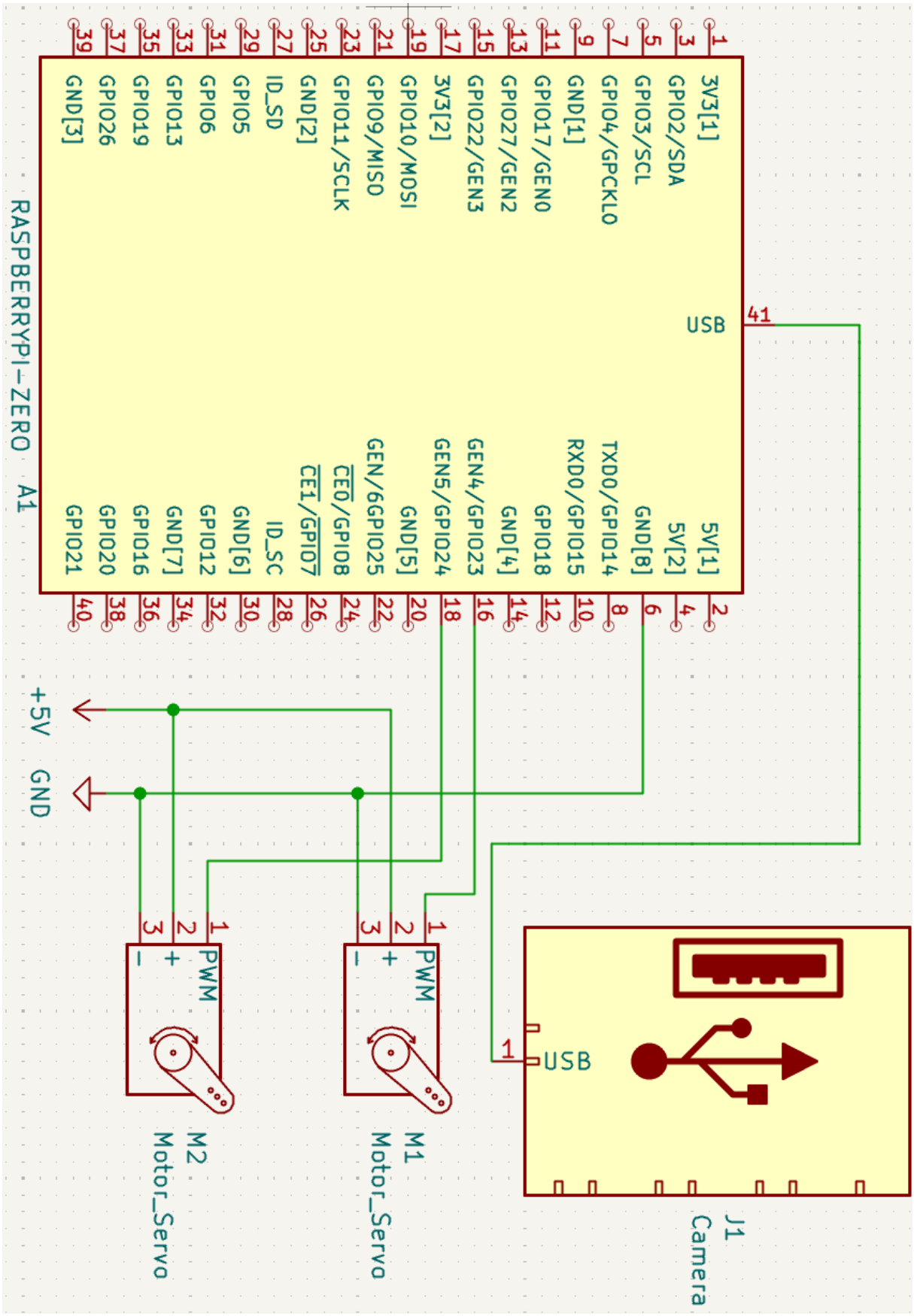
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sindhu V., Nivedha S., Prakash M. An Empirical Science Research on Bioinformatics in Machine Learning. Journal of Mechanics of Continua and Mathematical Sciences. JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES. 2020. P. 88
2. Vehicle Detection | HOG-SVM | python
   YouTube : веб-сайт.
   URL : https://www.youtube.com/watch?v=8_P257eFEqA (Дата звернення: 5.05.2025)
3. Kinsley H., Kukieła D. Neural networks from scratch: 2020. P. 13
4. Convolutional Neural Network | Deep Learning
   Developers Breach : веб-сайт.
   URL : https://developersbreach.com/convolution-neural-network-deep-learning/ (Дата звернення: 7.05.2025)
5. Understanding and Implementing Faster R-CNN.
   Medium : веб-сайт.
   URL : https://medium.com/@RobuRishabh/understanding-and-implementing-faster-r-cnn-248f7b25ff96 (Дата звернення: 30.04.2025)
6. Understanding SSD MultiBox – Real-Time Object Detection In Deep Learning
   Towards data science: веб-сайт.
   URL : https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab/ (Дата звернення: 25.4.2025)
7. What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object
   arxiv: веб-сайт.
   URL : https://arxiv.org/html/2408.15857v1 (Дата звернення: 8.05.2025)
8. Haq, H.B., Suwansantisuk, W. and Chamnongthai, K. An Empirical Science An optimized deep learning method for video summarization based on the user object

of interest. *International Journal of Advanced Computer Science and Applications*, 2023. P. 88

9. Simple Python Version Management: pyenv
   GitHub: веб-сайт.
   URL : https://github.com/pyenv/pyenv/blob/master/README.md (Дата звернення: 04.17.2025)
10. CUDA Toolkit 12.9 Downloads
    Nvidia: веб-сайт.
    URL : https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Debian&target_version=12&target_type=deb_local (Дата звернення: 11.05.2025)
11. The pigpio library
    pigpio: веб-сайт.
    URL : https://abyz.me.uk/rpi/pigpio/ (Дата звернення: 5.04.2025)
12. mjpg-streamer
    GitHub: веб-сайт.
    URL : https://github.com/jacksonliam/mjpg-streamer/blob/master/README.md (Дата звернення: 5.04.2025)
13. Model Training with Ultralytics YOLO
    Ultralytics: веб-сайт.
    URL : https://docs.ultralytics.com/modes/train/ (Дата звернення: 15.03.2025)
14. Socket Objects
    python: веб-сайт.
    URL : https://docs.python.org/3/library/socket.html#socket-objects (Дата звернення: 12.05.2025)

ДОДАТОК А


RASPBERRYPI-ZERO A1
3V3[1]
GPIO2/SDA
GPIO3/SCL
GPIO4/GPCKLO
GND[1]
GPIO17/GEN0
GPIO27/GEN2
GPIO22/GEN3
3V3[2]
GPIO10/MOSI
GPIO9/MISO
GPIO11/SCLK
GND[2]
ID_SD
GPIO5
GND[6]
GPIO12
GND[7]
GPIO16
GPIO20
GPIO21
USB
5V[1]
5V[2]
GND[8]
TXD0/GPIO14
RXD0/GPIO15
GPIO18
GND[4]
GEN4/GPIO23
GEN5/GPIO24
GND[5]
GEN/6GPIO25
CE0/GPIO8
CE1/GPIO7
ID_SC
GND[6]
GPIO12
GND[7]
GPIO16
GPIO20
GPIO21
+5V
GND
PWM
M2
Motor_Servo
PWM
M1
Motor_Servo
USB
J1
Camera

ДОДАТОК Б

```
import time
import socket
import threading
from time import sleep
import cv2

HEADER = 64
PORT = 5050
# SERVER = socket.gethostbyname(socket.gethostname())
SERVER = "0.0.0.0"
ADDR = (SERVER, PORT)
DISCONNECT_MESSAGE = "-=DISCONNECT=-"
FORMAT = "utf8"

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(ADDR)


def recv_all(conn, length):
    data = b''
    while len(data) < length:
        packet = conn.recv(length - len(data))
        if not packet:
            return None  # connection closed
        data += packet
    return data


def send_message(conn, msg):
    message = msg.encode(FORMAT)
    msg_length = len(message)
    send_length = str(msg_length).encode(FORMAT)
    send_length += b' ' * (HEADER - len(send_length))
    conn.send(send_length)
    conn.send(message)


def handle_client(conn, addr):
    print(f"[NEW CONNECTION] {addr} connected.")

    welcome = "Hello from server"
    message = welcome.encode(FORMAT)
    msg_length = len(message)
    send_length = str(msg_length).encode(FORMAT)
    send_length += b' ' * (HEADER - len(send_length))
    conn.send(send_length)
    conn.send(message)
    # i = 0
    threading.Thread(target=turret_control,          daemon=True,
args=(conn, )).start()
```

```
    connected = True

    while connected:
        # move_servo()
        # send_message(conn, f"Hello from server function! {i}")
        # i += 1
        msg_length_raw = recv_all(conn, HEADER)
        if not msg_length_raw:
            break  # Connection closed

        try:
            msg_length                                          =
int(msg_length_raw.decode(FORMAT).strip())
        except ValueError:
            print(f"Failed    to    parse    message    length:
{msg_length_raw}")
            break

        msg = recv_all(conn, msg_length)
        if not msg:
            break  # Connection closed

        msg = msg.decode(FORMAT)
        if msg == DISCONNECT_MESSAGE:
            connected = False
        print(f"[{addr}] {msg}")
    conn.close()


def start_server():
    server.listen()
    print(f"[LISTENING] Sever is listening on {SERVER}")
    while True:
        conn, addr = server.accept()
        threading.Thread(target=handle_client,       args=(conn,
addr)).start()
        print(f"[ACTIVE CONNECTIONS] {threading.active_count() -
1}")


def turret_control(conn):
    positionX = 0
    positionY = 0.75
    step = 0.05
    increasing = True

    frame_width = 1280
    frame_height = 720

    obj_gen = obj_detect()  # Get the generator
```

```
    last_search_time = time.time()
    search_interval = 0.5  # seconds

    while True:
        try:
            objs = next(obj_gen)
        except StopIteration:
            print("[INFO] Video stream ended.")
            break
        except Exception as e:
            print(f"[ERROR] obj_detect() failed: {e}")
            continue

        if obj:  # If objs detected, use tracking mode
            for x_obj, y_obj, w, h in objs:
                error_x, error_y = compute_errors(
                    x_obj, y_obj, frame_width, frame_height)

                # Update servos proportionally
                positionX = update_servo_position(
                    positionX, error_x, gain=0.05)
                positionY = update_servo_position(
                    positionY, error_y, gain=0.05)
                print(f"[SERVO] error X {error_x}")
                print(f"[SERVO] error Y {error_y}")

        else:
            #    Perform    search    movement    only    every
`search_interval` seconds
            now = time.time()
            if now - last_search_time >= search_interval:
                last_search_time = now
                if increasing:
                    positionX += step
                    if positionX >= 1.0:
                        positionX = 1.0
                        positionY = 0.75
                        increasing = False
                else:
                    positionX -= step
                    if positionX <= -1.0:
                        positionX = -1.0
                        positionY = 0.75
                        increasing = True

        msgX = str(round(positionX, 2))
        msgY = str(round(positionY, 2))
        try:
            send_message(conn, f"servo_X {msgX}\nservo_Y {msgY}")
        except OSError as e:
            print(f"[TURRET ERROR] {e}")
```

```
                break


    def update_servo_position(current_value, error, gain=0.0001):
        # Proportional control
        delta = error * gain
        new_value = current_value + delta
        # Clamp to [-1.0, 1.0]
        new_value = max(-1.0, min(1.0, new_value))
        return new_value


    def compute_errors(x_obj, y_obj, frame_width, frame_height):
        error_x = (frame_width / 2 - x_obj) / (frame_width / 2)  # -
    1 to 1
        error_y = - (frame_height / 2 - y_obj) / (frame_height / 2)
    # -1 to 1
        return error_x, error_y


      def yolo_detect():
          model = YOLO("best.pt")
          capture =
cv2.VideoCapture("http://192.168.0.104:8080/?action=stream")

          if not capture.isOpened():
              print("[ERROR] Cannot open video stream")
              return

          while True:
              ret, frame = capture.read()
              if not ret:
                  print("[WARNING] Failed to read frame")
                  yield []
                  continue

              results = model(frame)[0]
              UAVs = []

              for box in results.boxes:
                  cls = int(box.cls[0])
```

```
                    x1, y1, x2, y2 = map(int, box.xyxy[0])
                    w = x2 - x1
                    h = y2 - y1
                    cx = x1 + w // 2
                    cy = y1 + h // 2
                    UAVs.append((cx, cy, w, h))
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255,
0), 2)
                cv2.imshow("YOLO Detection", frame)
                cv2.waitKey(1)
            yield UAVs


    # threading.Thread(target=yolo_detect, args=(q_detect))

    print("[STARTRING] Server is starting")
    threading.Thread(target=start_server, daemon=True).start()
    # start_server()
    try:
        while True:
            sleep(1)
    except KeyboardInterrupt:
        print("Exiting gracefully...")
        server.close()
```

ДОДАТОК В

```
import socket
import threading
from gpiozero import Servo
from time import sleep
from queue import Queue
servo_X = Servo(pin=23,
                min_pulse_width=0.0005,
                max_pulse_width=0.0025,
                frame_width=0.02)
servo_Y = Servo(pin=24,
                min_pulse_width=0.0005,
                max_pulse_width=0.0025,
                frame_width=0.02)
servo_X.value = 0
servo_Y.value = 0.75
HEADER = 64
PORT = 5050
SERVER = "192.168.0.105"
ADDR = (SERVER, PORT)
DISCONNECT_MESSAGE = "-=DISCONNECT=-"
FORMAT = "utf8"
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)
def recv_all(conn, length):
    data = b''
    while len(data) < length:
        packet = conn.recv(length - len(data))
        if not packet:
            return None  # connection closed
        data += packet
    return data
```

```
    def send(msg):
        message = msg.encode(FORMAT)
        msg_length = len(message)
        send_length = str(msg_length).encode(FORMAT)
        send_length += b' ' * (HEADER - len(send_length))
        client.send(send_length)
        client.send(message)
    def turret_control(q_msg):
        positionX = 0
        positionY = 1
        servo_X.value = positionX
        servo_Y.value = 0
        while True:
            msg = q_msg.get()
            print(f"Turret patrol msg: {msg}")
            msg_list = msg.split()
            if msg_list[0] == "servo_X":
                print(msg_list)
                positionX = float(msg_list[1])
                positionY = float(msg_list[3])
                #  send(f"servo_X  =  {servo_X.value}\nservo_Y  =
{servo_Y.value}")
            servo_X.value = positionX
            servo_Y.value = positionY
    def receive(q_msg):
        while True:
            msg_length_raw = recv_all(client, HEADER)
            if not msg_length_raw:
                break  # Server closed connection
            try:
                msg_length                                          =
int(msg_length_raw.decode(FORMAT).strip())
            except ValueError:
                print(f"Failed   to   parse   message   length:
```

```
{msg_length_raw}")
                    break
                msg = recv_all(client, msg_length)
                if not msg:
                    break
                # print("[SERVER]", msg.decode(FORMAT))
                q_msg.put(msg.decode(FORMAT))
        q = Queue()
        threading.Thread(target=turret_control,           daemon=True,
args=(q,)).start()
        threading.Thread(target=receive,                  daemon=True,
args=(q,)).start()
        send("Hello!")
        try:
            while True:
                sleep(1)
        except KeyboardInterrupt:
            print("Exiting gracefully...")
            send(DISCONNECT_MESSAGE)
            client.close()
```