Кафедра СИСТЕМ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської кваліфікаційної роботи на тему

РОЗРОБЛЕННЯ СИСТЕМИ УПРАВЛІННЯ РОБОТА-МАНІПУЛЯТОРА З КОМП'ЮТЕРНИМ ЗОРОМ

Студент групи КН-412 **Івасько Олег Ігорович**
(шифр, прізвище та ініціали)

Керівник проекту _____ (**Костянтин Колесник**)

Консультанти _____ (_____)

_____ (_____)

_____ (_____)

_____ (_____)

Завідувач кафедри Михайло ЛОБУР

«___» _____ 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут Інститут комп'ютерних наук та інформаційних технологій

Кафедра Систем автоматизованого проектування

Спеціальність 122 "Комп'ютерні науки"

„ЗАТВЕРДЖУЮ”

Завідувач

кафедри САП

Михайло ЛОБУР

«___» _____ 2025 р.

ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу студента групи КН-412 ОКР Бакалавр .

Івасько Олег Ігорович

(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Розроблення системи управління робота-маніпулятора з комп'ютерним зором

затверджена наказом по університету від 30.04.2025 р. № 1566-4-08

2. Термін подання студентом закінченої роботи (проекту) 06.06.2025

3. Вихідні дані для роботи (проекту) основи з робототехніки, вміння працювати з Arduino Uno R3 та ESP32, знання мов програмування C# та C++ з використанням їх у Unity, основи з обробки зображень та розпізнавання образів, зокрема уміння користуватись платформою EdgeImpulse, практичні знання з проектування та моделювання за допомогою інструментів AutoCAD

4. Зміст розрахунково-пояснювальної записки (перелік питань, які треба розробити) аналіз існуючих роботизованих систем з комп’ютерним зором, обґрунтування вибору апаратного забезпечення, розробка та моделювання конструкції маніпулятора, реалізація комп’ютерного зору та розпізнавання об'єктів

5. Перелік графічного матеріалу збірка моделі робота в AutoCAD, креслення однієї деталі робота, схема архітектури системи та звязків між компонентами, блок-схема алгоритму управління роботом, схема електрична принципова, віртуальна схема підключення сервоприводів до контроллера

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту) Unity, EdgeImpulse, AutoCAD, Fusion 360, CMake, TensorFlow, WebSocket, Node.js

7. Консультування роботи (проекту), із зазначенням розділів роботи

Розділ	Консультант	Завдання видав		Завдання прийняв	

8. Дата, коли видано завдання _____

Керівник **Костянтин Колесник**
(підпис)

Завдання прийняв до виконання **Олег Івасько**
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1	Ознайомлення з темою, постановка мети та завдань, опрацювання літератури	28.04 – 30.04	
2	Аналіз існуючих рішень, вибір апаратних і програмних засобів	01.05 – 03.05	
3	Моделювання та проєктування конструкції маніпулятора	04.05 – 07.05	
4	Розробка програмної логіки керування	08.05 – 10.05	
5	Реалізація комп’ютерного зору, обробка зображень	11.05 – 14.05	
6	Інтеграція компонентів у єдину систему	15.05 – 17.05	
7	Реалізація апаратної частини (збірка, підключення)	18.05 – 21.05	
8	Тестування, коригування, фіксація результатів	22.05 – 24.05	
9	Написання та оформлення звіту	25.05 – 28.05	
10	Остаточне редагування, підготовка до захисту	29.05 – 31.05	

Студент Івасько Олег
(підпис)

Керівник Костянтин Колесник
(підпис)

АНОТАЦІЯ

Івасько О.І., Колесник К.К. Розроблення системи управління робота-маніпулятора з комп'ютерним зором. – Національний університет «Львівська політехніка», Львів, 2025 р.

Розширена анотація

У роботі представлено процес створення керованої системи для настільного робота-маніпулятора, що здатен самостійно виявляти об’єкти та виконувати дії з ними. Основна увага приділяється поєднанню віртуального середовища керування з реальним механічним пристроєм, що забезпечує взаємодію між програмною логікою, системою розпізнавання зображень та фізичним виконанням команд.

Сконструйовано маніпулятор з 4-ма ступенями свободи, розроблено програмну частину для його керування та реалізовано обробку відеозображення з камери. Визначення положення об’єкта в просторі та розрахунок рухів виконуються автоматично, після чого команди передаються на виконавчі механізми. Завдяки модульній структурі система легко розширюється та адаптується до нових завдань.

Проєкт спрямований на практичну демонстрацію можливості створення доступної та функціональної навчальної або експериментальної платформи у сфері робототехніки з використанням комп’ютерного аналізу зображень.

Об’єктом дослідження є процес керування роботизованими маніпуляторами з використанням засобів автоматизації та комп’ютерного зору.

Предметом дослідження є методи побудови апаратно-програмної системи, що забезпечує автоматичне розпізнавання об’єктів у просторі та керування рухами маніпулятора на основі отриманих даних.

Метою даної роботи є створення системи керування настільним роботом-маніпулятором, яка здатна самостійно виявляти та класифікувати об’єкти у полі зору, визначати їх положення та здійснювати точні маніпуляції відповідно до розрахованої траєкторії.

Новизна дослідження полягає у створенні гнучкої, модульної системи, яка об’єднує різнорідні компоненти — механічний маніпулятор, засоби комп’ютерного зору, візуальне середовище моделювання та інтерфейси обміну даними — в єдину функціональну структуру, яку можна модифіковувати, розширювати, та використовувати для виконання різних задач.

Практичне значення одержаних результатів полягає в тому, що розроблена система є доступною для повторення, масштабування та використання в навчальних цілях, лабораторних роботах і дослідницьких проєктах у сфері робототехніки та автоматизованих систем управління.

Ключові слова: робот-маніпулятор, комп’ютерний зір, керування рухом, інверсна кінематика, розпізнавання об’єктів, інтегрована система, апаратно-програмний комплекс, модульна архітектура, автоматизація, Unity.

Перелік використаних джерел:

1. Craig, J. J. Introduction to robotics: Mechanics and control. – 4th ed. – Pearson, 2017. – 448 p.
2. Bishop, C. M. Pattern Recognition and Machine Learning [Електронний ресурс]. – Springer, 2006. – 738 p. – Режим доступу: https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf, вільний. – Загал. з екрана. (04.06.2025).
3. Edge Impulse. Edge Impulse Documentation [Електронний ресурс]. – Режим доступу: https://docs.edgeimpulse.com/, вільний. – Загал. з екрана. (04.06.2025).

ANNOTATION

Ivasko O.I., Kolesnyk K.K. Development of a robot manipulator control system with computer vision and sorting functionalities. – Lviv Polytechnic National University, Lviv, 2025.

Extended annotation

This work presents the development process of a controllable system for a desktop robotic manipulator capable of autonomously detecting and interacting with objects. The focus is placed on combining a virtual control environment with a physical mechanical device, enabling interaction between software logic, image recognition, and the physical execution of commands.

A manipulator with four degrees of freedom was constructed, the software component for its control was developed, and real-time image processing from a camera was implemented. The system automatically detects object positions in space and calculates the required movements, which are then transmitted to the actuators. Due to its modular structure, the system can be easily expanded and adapted to new tasks.

The project demonstrates the practical feasibility of creating an accessible and functional platform for educational or experimental purposes in the field of robotics using computer image analysis.

The object of the study is the process of controlling robotic manipulators using automation tools and computer vision.

The subject of the study is the methods for building a hardware-software system that performs automatic object recognition in space and controls the manipulator's movements based on the obtained data.

The purpose of the work is to develop a control system for a desktop robotic manipulator capable of autonomously detecting and classifying objects within its field of view, determining their position, and performing precise manipulations according to the calculated trajectory.

The novelty of the study lies in the creation of a flexible, modular system that integrates heterogeneous components — a mechanical manipulator, computer vision

tools, a virtual simulation environment, and data exchange interfaces — into a single functional structure, which can be modified, extended, and applied to various tasks.

The practical significance of the obtained results is that the developed system is reproducible, scalable, and suitable for use in educational applications, laboratory work, and research projects in the field of robotics and automated control systems.

Keywords: robotic arm, computer vision, motion control, inverse kinematics, object detection, integrated system, hardware-software complex, modular architecture, automation, Unity.

List of used sources:

1. Craig, J. J. Introduction to robotics: Mechanics and control. – 4th ed. – Pearson, 2017. – 448 p.
2. Bishop, C. M. Pattern Recognition and Machine Learning [Electronic Resource]. – Springer, 2006. – 738 p. – Access Mode: https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf, free. – Viewed on screen. (04.06.2025).
3. Edge Impulse. Edge Impulse Documentation [Electronic Resource]. – Access Mode: https://docs.edgeimpulse.com/, free. – Viewed on screen. (04.06.2025).

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CAD (Computer-Aided Design) –системи автоматизованого проектування

UI (User Interface) – інтерфейс користувача

PWM (Pulse-Width Modulation) – широтно-імпульсна модуляція

HTTP (HyperText Transfer Protocol) – протокол передавання гіпертексту

TCP (Transmission Control Protocol) – протокол керування передаванням

ПК – персональний комп’ютер

HTML (HyperText Markup Language) – мова розмітки гіпертексту

JSON (JavaScript Object Notation) – формат обміну даними

IK (Inverse Kinematics) – інверсна кінематика

IP (Internet Protocol) – інтернет-протокол

DLL (Dynamic Link Library) – бібліотека динамічного компонування

USB (Universal Serial Bus) – універсальна послідовна шина

SCARA (Selective Compliance Assembly Robot Arm) – маніпулятор із вибірковою жорсткістю

CNN (Convolutional Neural Network) – згорткова нейронна мережа

ПКВМ – програмно-керована вбудована машина

ПЗ – програмне забезпечення

ЗМІСТ

ВСТУП 11
1. АНАЛІЗ ПОТОЧНОГУ СТАНУ ІНТЕЛЕКТУАЛЬНОЇ РОБОТОТЕХНІКИ 14
1.1. Огляд сучасних технологій у галузі робототехніки з комп'ютерним зором……..……14
1.1.1. Класифікація роботів-маніпуляторів за призначенням та конструкцією..... 14
1.1.2. Тенденції розвитку систем комп'ютерного зору для промислових роботів……18
1.2. Порівняння існуючих роботів-маніпуляторів 19
1.3. Економічна частина. 21
1.4. Формулювання вимог до системи. 22
1.4.1. Функціональні вимоги до робота-маніпулятора..... 22
1.4.2. Вимоги до системи комп'ютерного зору 24
Висновки до розділу 1 25
2. ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ..... 27
2.1. Вибір платформи для управління та візуалізації. 27
2.2. Інструменти проектування механічної частини. 30
2.3. Вибір архітектури апаратної частини 35
2.3.1. Контролер виконавчих механізмів на базі Arduino UNO R3 36
2.3.2. Модуль комп'ютерного зору на базі ESP32-CAM..... 38
2.4. Вибір протоколів та технологій комунікацій..... 40
2.5. Технології комп'ютерного зору для розпізнавання об'єктів..... 42
3. Апаратно-програмна реалізація системи. 47
3.1. Загальна архітектура системи..... 47
3.2. Проектування робота-маніпулятора..... 48
3.3. Передача відеопотоку та зв’язок між компонентами..... 55
3.3.1. Реалізація WebSocket-сервера 55

3.3.2. Підключення ESP32-CAM до WebSocket-сервера та передача відео……57
3.3.3. Прийом відео в Unity, декодування та відображення 58
3.3.4. Віддалене налаштування параметрів камери..... 59
3.4. Система комп’ютерного зору 61
3.4.1. Розробка моделі в Edge Impulse..... 61
3.4.2. Інтеграція моделі у власну систему 65
3.5. Керування роботом 67
3.5.1. Визначення позиції об’єктів у світових координатах 67
3.5.2. Обрахунок інверсної кінематики. 68
3.6. Взаємодія з Arduino..... 71
3.7. Збірка та тестування системи..... 73
Висновки до розділу 3 75
ВИСНОВКИ..... 77
ДОДАТОК А..... 81
ДОДАТОК Б 89

ВСТУП

Сучасний розвиток інженерії, автоматизації та штучного інтелекту зумовив стрімке зростання інтересу до інтелектуальних технічних систем, зокрема в галузі робототехніки. Роботизовані системи дедалі частіше застосовуються не лише у промисловості, а й в освітніх, дослідницьких та побутових середовищах. Важливим напрямом є створення маніпуляторів, які здатні до автономної взаємодії з навколишнім середовищем за рахунок вбудованих сенсорів, алгоритмів аналізу та планування дій.

Одним із ключових чинників, що забезпечують автономність робототехнічних систем, є використання комп'ютерного зору. Завдяки здатності обробляти візуальну інформацію, робот може самостійно виявляти об'єкти, оцінювати їхнє положення та приймати рішення щодо подальших дій. Поєднання візуального аналізу з системами керування механікою відкриває широкі можливості для створення більш гнучких, інтуїтивно керованих та адаптивних роботів.

У цьому контексті особливий інтерес становить побудова доступної, модульної системи, що поєднує механічного маніпулятора, засоби комп'ютерного зору та віртуальне середовище керування в єдиний узгоджений апаратно-програмний комплекс. Такий підхід дозволяє не лише ефективно реалізувати практичні задачі (наприклад, сортування об'єктів, переміщення тощо), але й забезпечити гнучкість та простоту адаптації до нових функцій, що є надзвичайно важливим у навчальному та дослідницькому середовищі.

Актуальність даної роботи полягає в необхідності розробки технічних рішень, які поєднують доступність, функціональність і гнучкість. Комерційні рішення часто мають високу вартість, складну інтеграцію та обмежені можливості модифікації. У свою чергу, розробка власної системи з відкритою архітектурою дозволяє створити платформу, що може бути використана як база для подальших досліджень, розширень або навчання.

У межах цієї роботи розглядається процес розробки керованої системи для настільного робота-маніпулятора, який здатен автоматично розпізнавати об’єкти, визначати їхнє положення у просторі та виконувати відповідні дії. Система реалізована на основі доступних електронних компонентів, з використанням 3D-друку для виготовлення деталей конструкції, а також віртуального середовища моделювання для інтеграції логіки керування та візуалізації.

Мета роботи – розробити апаратно-програмну систему керування настільним роботом-маніпулятором, яка здатна автономно виявляти об’єкти у полі зору, визначати їх положення у координатному просторі та виконувати захоплення на основі розрахованих траєкторій.

Для досягнення поставленої мети в роботі вирішуються наступні завдання:

- спроєктувати фізичну конструкцію маніпулятора з урахуванням модульності, доступності та простоти виготовлення;
- розробити програмну архітектуру системи з поділом на функціональні компоненти;
- реалізувати передачу відео в реальному часі від камери до візуального середовища;
- створити та навчити модель розпізнавання об’єктів на основі власного датасету;
- реалізувати алгоритм інверсної кінематики для визначення кутів обертання ланок маніпулятора;
- забезпечити стабільну передачу команд на виконавчі елементи та їхню коректну фізичну реалізацію.

У першому розділі буде подано огляд літератури, досліджено типи маніпуляторів, принципи дії систем комп’ютерного зору та сформульовано загальні вимоги до системи.

У другому розділі потрібно буде обґрунтувати вибір архітектури системи, програмної платформи, засобів проектування механіки, способу комунікації між модулями та методів розпізнавання.

У третьому розділі описується процес реалізації системи: створення фізичної конструкції, імпорт моделі в середовище управління, обробку зображень, розрахунок рухів і передавання команд на виконавчі механізми.

1. АНАЛІЗ ПОТОЧНОГУ СТАНУ ІНТЕЛЕКТУАЛЬНОЇ РОБОТОТЕХНІКИ

1.1. Огляд сучасних технологій у галузі робототехніки з комп'ютерним зором

1.1.1. Класифікація роботів-маніпуляторів за призначенням та конструкцією

Роботи-маніпулятори є одним із найпоширеніших класів роботів, що застосовуються в різних галузях – від промисловості до освіти та дослідницьких лабораторій. Вони характеризуються здатністю виконувати маніпуляційні дії з об'єктами фізичного середовища завдяки спеціалізованій механічній конструкції та системі керування. Для систематизації різноманітних конструкцій та принципів функціонування роботів-маніпуляторів доцільно провести їх класифікацію за різними критеріями.

Класифікація за призначенням

Залежно від сфери застосування та функціональних завдань, роботи-маніпулятори можна поділити на такі категорії:

1. Промислові маніпулятори:

- Збиральні – призначені для автоматизації процесів складання виробів з окремих деталей і вузлів.
- Зварювальні – спеціалізовані роботи для виконання різних видів зварювальних операцій (контактне, точкове, дугове зварювання).
- Фарбувальні – забезпечують рівномірне нанесення фарби на поверхні виробів.
- Пакувальні – призначені для автоматизації процесів пакування готової продукції.
- Транспортувальні – здійснюють переміщення об'єктів між різними позиціями.

2. Навчальні маніпулятори:

- Лабораторні – використовуються для проведення експериментів у контрольованих умовах, маніпуляцій з небезпечними речовинами.

• Дослідницькі – застосовуються для вивчення нових методів управління, алгоритмів руху та взаємодії з середовищем.

3. Сервісні маніпулятори:

• Медичні – використовуються для проведення хірургічних операцій, реабілітаційних процедур та допомоги пацієнтам.

• Освітні – призначені для навчання основам робототехніки, програмування та механіки. Такі системи зазвичай мають компактні розміри, доступну вартість та модульну архітектуру, що спрощує їх вивчення і модифікацію. Розроблюваний у рамках даної роботи маніпулятор належить саме до цієї категорії.

• Побутові – виконують рутинні завдання в домашньому господарстві.

4. Спеціалізовані маніпулятори:

• Космічні – розроблені для виконання завдань у космічному просторі.

• Підводні – функціонують у водному середовищі для досліджень, ремонту або будівництва.

• Військові – застосовуються для знешкодження вибухонебезпечних предметів та інших спеціальних операцій.

Класифікація за конструкцією та кінематичною схемою

Конструктивні особливості роботів-маніпуляторів визначають їх робочу зону, маневреність та функціональні можливості:

1. За типом кінематичної схеми:

• Декартові (прямокутні) – мають три лінійні ступені свободи, які відповідають осям прямокутної системи координат (X, Y, Z). Відзначаються високою точністю позиціонування та простотою управління, але мають обмежену робочу зону у формі паралелепіпеда.

• Циліндричні – поєднують один обертальний ступінь свободи навколо вертикальної осі з двома лінійними переміщеннями: вздовж вертикальної осі (Z) та вздовж радіального напрямку. Робоча зона має форму циліндра.

• Сферичні (полярні) – використовують два обертальні та один лінійний суглоби, що дозволяє досягати точок у сферичному просторі.

• Шарнірні (антропоморфні) – складаються з послідовно з'єднаних ланок з обертальними суглобами, що нагадує будову людської руки. Мають високу маневреність та можливість обходити перешкоди.

• SCARA– спеціалізована конструкція з горизонтальними обертальними суглобами та вертикальним лінійним переміщенням, оптимізована для операцій "візьми та розмісти".

• Паралельні – використовують кілька кінематичних ланцюгів для підтримки робочого органа, що забезпечує високу жорсткість конструкції та точність позиціонування.

2. За кількістю ступенів свободи:

• Маніпулятори з трьома ступенями свободи – дозволяють позиціонувати робочий орган у тривимірному просторі без зміни орієнтації.

• Маніпулятори з чотирма ступенями свободи – додають можливість обертання робочого органа навколо однієї осі.

• Маніпулятори з п'ятьма ступенями свободи – забезпечують позиціонування та часткову орієнтацію робочого органа.

• Маніпулятори з шістьма ступенями свободи – дозволяють повністю контролювати положення та орієнтацію робочого органа в просторі.

• Надлишкові маніпулятори – мають більше шести ступенів свободи, що підвищує маневреність та дозволяє обходити перешкоди.

3. За типом приводів:

• Електричні – використовують електродвигуни (сервоприводи, крокові двигуни) для забезпечення точного контролю положення. Відрізняються високою точністю, керованістю та повторюваністю рухів.

• Гідравлічні – застосовують гідроциліндри та гідромотори, що забезпечує високі силові характеристики, але вимагає складної системи гідроприводу.

• Пневматичні – використовують стиснене повітря для приведення в рух виконавчих механізмів. Характеризуються простотою конструкції та високою швидкодією.

• Комбіновані – поєднують різні типи приводів для оптимізації параметрів маніпулятора.

4. За вантажопідйомністю:

• Надлегкі – до 1 кг, використовуються для прецизійних операцій.

• Легкі – від 1 до 10 кг, широко застосовуються в електронній промисловості.

• Середні – від 10 до 100 кг, найпоширеніший клас промислових маніпуляторів.

• Важкі – від 100 до 1000 кг, призначені для переміщення великогабаритних деталей.

• Надважкі – понад 1000 кг, застосовуються в важкому машинобудуванні.

Класифікація за системами управління

1. За типом управління:

• Програмовані – виконують задану послідовність дій, запрограмовану заздалегідь.

• Адаптивні – здатні коригувати програму дій на основі інформації від сенсорів.

• Інтелектуальні – використовують елементи штучного інтелекту для прийняття рішень та планування дій.

2. За рівнем автономності:

• Автоматичні – функціонують без втручання оператора за заданою програмою.

• Напівавтоматичні – частина операцій виконується автоматично, частина – під керуванням оператора.

• Телекеровані – всі дії виконуються під безпосереднім керуванням оператора.

Інтеграція систем комп'ютерного зору з роботами-маніпуляторами значно розширює їх функціональні можливості та сфери застосування. Системи технічного зору дозволяють реалізувати:

- Розпізнавання та локалізацію об'єктів у робочій зоні;
- Контроль якості виконуваних операцій;
- Адаптацію до змін у робочому середовищі;
- Взаємодію з динамічними об'єктами;

У контексті даної роботи, спрямованої на розробку системи управління настільного робота-маніпулятора з комп'ютерним зором, потрібно враховувати взаємозв'язок між конструктивними особливостями маніпулятора та можливостями системи комп'ютерного зору. Проект, що розробляється, призначений для захоплення та переміщення невеликих об'єктів з використанням алгоритмів розпізнавання образів для класифікації об'єктів та визначення їх положення.

Хоча розроблюваний маніпулятор має навчальне призначення, він демонструє принципи, що можуть бути масштабовані для промислових застосувань, і слугує прототипом для більш складних систем автоматизації. При цьому ключовим аспектом розробки є саме система управління, що дозволяє інтегрувати механічні компоненти з алгоритмами комп'ютерного зору для створення цілісної системи, здатної автономно виконувати маніпуляційні задачі.

1.1.2. Тенденції розвитку систем комп'ютерного зору для промислових роботів

Системи комп'ютерного зору для промислових роботів демонструють стрімкий розвиток, що суттєво розширює їх функціональні можливості та сфери застосування. Основні сучасні тенденції розвитку таких систем включають:

1. **Інтеграція глибокого навчання**. Нейронні мережі, зокрема згорткові (CNN) та трансформери, стали основою більшості систем розпізнавання об'єктів. Це забезпечує точність детекції понад 95% у виробничих умовах при високій варіативності об'єктів.

2. **3D-зір та просторове сприйняття**. Впровадження стереокамер, ToF-сенсорів (Time-of-Flight) та лідарів забезпечує роботам просторове розуміння середовища. Без цього не обійтись при маніпуляціях з об'єктами складної форми та роботи в динамічному оточенні.

3. **Обробка даних на периферійних пристроях**. Периферійне обчислення може суттєво зменшувати затримки обробки даних, що необхідно для швидкодіючих систем промислової автоматизації. Спеціалізовані нейропроцесори та ПКВМ-рішення не лише оптимізують швидкодію, але й енергоспоживання.

4. **Активні системи калібрування та самокорекції**. Сучасні системи здатні автоматично адаптуватися до змін освітлення, вібрацій та інших факторів середовища, що підвищує їхню надійність у промислових умовах.

5. **Семантична сегментація та розуміння сцени**. Перехід від простого розпізнавання до комплексного аналізу виробничої сцени дозволяє роботам приймати більш складні рішення без втручання людини.

6. **Колаборативні системи зору**. Розвиток розподілених систем комп'ютерного зору, де дані від кількох роботів об'єднуються для створення повної моделі робочого простору.

7. **Адаптивне навчання**. Системи, здатні покращувати свою продуктивність під час експлуатації через техніки самонавчання та адаптації до нових типів об'єктів без повного перенавчання моделей.

Технології комп'ютерного зору стали невід’ємним компонентом сучасної промислової робототехніки, забезпечуючи автоматизацію складних процесів, які раніше потребували людського втручання, та підвищуючи гнучкість виробничих ліній при збереженні високих показників точності та надійності.

1.2. Порівняння існуючих роботів-маніпуляторів

У сегменті настільних роботів-маніпуляторів представлено значну кількість комерційних та відкритих рішень із різними характеристиками, функціональними можливостями та ціновими категоріями. Нижче в табл.1.1. наведено порівняльний аналіз найбільш поширених моделей, що представлені на ринку станом на 2025 рік.

Таблиця 1.1.

Порівняльний аналіз існуючих роботів-маніпуляторів

Модель	Вантажо-підйомність, кг	Радіус робочої зони, мм	Точність позиціонування, мм	Ступені свободи	Вартість, грн
ABB YuMi IRB 14000	0,5 (на руку)	559	±0,02	7 (на руку)	8200
Elephant Robotics myCobot 280 Pi	0,25	280	±0,5	6	28 700
UFACTORY xArm 6	5	700	±0,1	6	205 000-287 000
Niryo Ned3Pro	0,5	500	±0,3	6	287 000

Для проектованого настільного маніпулятора найбільш релевантною моделлю для порівняння є myCobot 280 Pi через схожий формфактор, можливості інтеграції з відкритими платформами та доступну вартість при прийнятних технічних характеристиках для переміщення невеликих об'єктів.

1.3. Економічна частина.

В табл.1.2. приведено оцінку витрат на виготовлення маніпулятора.

Таблиця 1.2.

Оцінка витрат на виготовлення маніпулятора

№	Найменування	Кількість	Ціна за од., грн	Сума, грн
1	Сервопривід Tower Pro MG995 180°	2	170	340
2	Сервопривід TowerPro MG90S 180°	2	90	180
3	Сервопривід TowerPro SG90 180°	1	70	70
4	Плата Arduino Uno R3 CH340	1	210	210
5	Модуль Wi-Fi ESP32-CAM з камерою 2MP	1	350	350
6	Гвинт з потайною головкою M3x16 мм Expert Fix DIN 965	50	0.636	31.80
7	Гвинт з потайною головкою, DIN 965, 4.8, цинк білий M2,5x10	25	0.76	19
8	Гайка шестигранна, DIN 934, 6, цинк білий M3	25	0.56	14
9	Гайка шестигранна, DIN 934, 6, цинк білий M2,5	25	0.56	14
10	Підшипник 607-2RS (19*7*6)	5	20	100
11	Підшипник 695 2RS (13*5*4)	2	20	40
12	Блок та кабель живлення	1	250	250

Продовження таблиці 1.2

№	Найменування	Кількість	Ціна за од., грн	Сума, грн
13	Провід	-	-	10
14	PLA пластик	-	-	400
	Загальна вартість матеріалів			2028.8

Економічна ефективність проекту є досягнутою за допомогою:

1. **Зниження витрат на придбання обладнання**: Розроблений маніпулятор коштує в декілька разів менше від вартості комерційних аналогів зі схожою функціональністю.

2. **Гнучкість застосування**: Завдяки модульній архітектурі та відкритому ПЗ, маніпулятор може бути адаптований до різних завдань без додаткових капіталовкладень, що збільшує його економічну цінність.

3. **Освітня цінність**: Використання маніпулятора в освітніх цілях дозволяє знизити витрати на придбання комерційного обладнання для навчальних лабораторій.

Основним економічним обмеженням проекту є більші витрати часу на розробку та інтеграцію порівняно з готовими рішеннями, проте ці витрати компенсуються гнучкістю та зниженням капітальних вкладень.

1.4. Формулювання вимог до системи.

1.4.1. Функціональні вимоги до робота-маніпулятора

Розроблюваний робот-маніпулятор призначений для автоматизованого захоплення, переміщення та позиціонування об'єктів у визначеній робочій зоні. Конструкція маніпулятора повинна забезпечувати необхідну точність і повторюваність операцій, а система керування — гнучкість у взаємодії з користувачем.

Кінематична схема робота має включати не менше 4 ступенів свободи, що забезпечить достатню маніпулятивність та доступ до об'єктів в різних положеннях

у межах робочої зони. Така кількість ступенів свободи дозволить не лише позиціонувати захватний механізм у просторі, але й орієнтувати його відповідно до форми об'єктів.

Функціональні можливості маніпулятора повинні забезпечувати роботу з об'єктами масою до 200 грамів, що є цілком достатнім для навчальних та демонстраційних цілей. При цьому робоча зона повинна охоплювати простір радіусом не менше 250 мм по горизонталі від центральної осі основи та з висотою підйому не менше 150 мм, що дає змогу маніпулювати об'єктами на достатньо великій площі.

Для забезпечення надійного та коректного виконання операцій маніпулювання об'єктами система повинна забезпечувати точність позиціонування робочого органу з похибкою не більше ±5 мм та повторюваність позиціонування з похибкою не більше ±2 мм. Такі параметри точності є прийнятними для розпізнавання та захоплення об'єктів заданого розміру.

Система управління маніпулятором повинна бути реалізована з використанням платформи Unity, що дозволить створити зручний та інтуїтивно зрозумілий інтерфейс користувача. Програмне забезпечення має підтримувати два основні режими роботи:

- Автоматичний режим, при якому робот самостійно виконує запрограмовану послідовність дій;
- Ручний режим, що забезпечує безпосереднє керування маніпулятором оператором через графічний інтерфейс.

Захватний пристрій маніпулятора повинен забезпечувати надійне захоплення та утримання об'єктів з розмірами від 15 до 50 мм, що відповідає типовим розмірам об'єктів, які використовуються в демонстраційних задачах сортування та переміщення.

Важливим аспектом є також енергоефективність системи. Загальне енергоспоживання не повинно перевищувати 25 Вт при номінальному режимі роботи, що дозволить системі функціонувати від стандартного блоку живлення без необхідності використання додаткового охолодження.

Комунікація між компонентами системи має здійснюватися за допомогою стандартних протоколів, що забезпечить модульність та можливість подальшого масштабування системи. Апаратна частина на основі Arduino/ESP32 повинна бути інтегрована з програмним забезпеченням на платформі Unity через WebSocket протокол, що забезпечить швидку та надійну передачу даних.

1.4.2. Вимоги до системи комп'ютерного зору

Система комп'ютерного зору є ключовим компонентом розроблюваного робота-маніпулятора, який забезпечує його здатність до взаємодії з навколишнім середовищем. Основним призначенням цієї системи є розпізнавання та локалізація об'єктів у робочій зоні, визначення їх просторового положення та геометричних характеристик, необхідних для планування операцій захоплення.

Оптична підсистема комп'ютерного зору повинна включати камеру з роздільною здатністю не менше 320×240 пікселів, що забезпечить достатню деталізацію зображення для розпізнавання об'єктів в межах робочої зони. Камера повинна забезпечувати частоту оновлення зображення не менше 15 кадрів на секунду, що є необхідним для відстеження об'єктів та реагування на зміни в робочому просторі в умовах, наближених до реального часу.

Поле зору камери має становити не менше 60° по горизонталі, що дозволить охопити всю робочу зону маніпулятора без необхідності механічного переміщення камери. Фіксоване фокусне налаштування оптичної системи обрано для спрощення конструкції та зменшення вартості при збереженні необхідної функціональності.

Програмно-алгоритмічне забезпечення системи комп'ютерного зору має включати алгоритми розпізнавання об'єктів, здатні класифікувати їх з точністю не нижче 85%.

Система повинна забезпечувати локалізацію об'єктів, тобто визначення їх положення в робочій зоні маніпулятора, з точністю не гірше ±5 мм по координатах X та Y. Така точність є достатньою для коректного планування руху маніпулятора до цільового об'єкта та його захоплення.

Важливою функцією системи комп'ютерного зору є також визначення базових геометричних характеристик об'єктів, таких як приблизні розміри та

форма. Це необхідно для вибору оптимальної стратегії захоплення та орієнтації робочого органу маніпулятора відносно об'єкта.

Система повинна бути достатньо гнучкою для можливості додавання нових класів об'єктів через процедуру повторного навчання, що розширює функціональні можливості робота без необхідності повного перепрограмування алгоритмів комп'ютерного зору. Це особливо важливо для навчальних та дослідницьких цілей, коли виникає потреба в адаптації системи до нових задач.

Висновки до розділу 1

У результаті проведеного аналізу було встановлено, що інтелектуальні робототехнічні системи, зокрема роботи-маніпулятори з інтегрованими системами комп’ютерного зору, активно розвиваються та знаходять широке застосування в різних сферах – від промисловості до освіти. Проведено класифікацію роботів-маніпуляторів за різними ознаками: призначенням, конструктивними особливостями, кількістю ступенів свободи, типом приводів, системами керування, що дозволило визначити найбільш доцільну архітектуру для реалізації навчальної робототехнічної системи.

Особливу увагу приділено аналізу можливостей інтеграції комп’ютерного зору з системами керування роботів. Визначено, що використання таких систем дозволяє істотно підвищити функціональність роботів-маніпуляторів, зокрема шляхом забезпечення адаптації до зміни навколишнього середовища, точного визначення положення об'єктів та їх класифікації в режимі реального часу. Розглянуто сучасні тенденції розвитку в цій галузі, включаючи впровадження глибинного навчання, технологій периферійного обчислення, 3D-зору та семантичної сегментації.

Проведено порівняльний аналіз актуальних моделей настільних маніпуляторів, що дозволило обґрунтувати вибір формфактору та технічних характеристик для проєктованого зразка. Запропоноване рішення базується на низьковартісних компонентах із відкритою архітектурою, що сприяє зниженню

вартості розробки та забезпечує гнучкість у подальшій адаптації системи до нових завдань.

Сформульовано функціональні вимоги до конструкції робота-маніпулятора, його системи керування та комп’ютерного зору. Визначено параметри, що забезпечують належний рівень точності, швидкодії та енергоефективності для навчального застосування. Отримані результати та техніко-економічне обґрунтування підтверджують доцільність розробки власного маніпулятора як альтернативи комерційним рішенням, з можливістю масштабування до більш складних автоматизованих систем.

2. ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1. Вибір платформи для управління та візуалізації.

При проектуванні системи управління для робота-маніпулятора з комп'ютерним зором важливим етапом є вибір відповідної програмної платформи, яка забезпечуватиме реалізацію алгоритмів керування, візуалізацію процесу роботи та зручний інтерфейс для оператора. Для розробки даного проекту було обрано платформу Unity як основний інструмент для створення системи керування та візуалізації.

Unity є потужним кросплатформнним інструментом для розробки інтерактивних додатків, який традиційно використовується в ігровій індустрії, але завдяки своїм функціональним можливостям все частіше застосовується і в технічних рішеннях, зокрема в області робототехніки та систем комп'ютерного зору. Вибір Unity для реалізації даного проекту обґрунтовується наступними перевагами:

1. **Інтегроване середовище для 3D-візуалізації**. Unity надає потужні інструменти для створення тривимірних сцен та моделювання фізичних взаємодій. Це дозволяє не лише візуалізувати модель робота-маніпулятора, але й створювати цифрові двійники для симуляції та тестування алгоритмів керування перед їх застосуванням на реальному обладнанні.

2. **Компонентна архітектура**. Система об'єктів та компонентів Unity (GameObject-Component) дозволяє створювати модульні та розширювані рішення. Це особливо важливо для проекту робота-маніпулятора, де потрібно реалізувати взаємодію різних підсистем: керування двигунами, отримання та обробку даних з сенсорів, взаємодію з користувачем.

3. **Крос-платформність**. Unity підтримує розгортання додатків на різних операційних системах (Windows, macOS, Linux), що забезпечує гнучкість при виборі обладнання для керуючого комп'ютера та спрощує процес впровадження системи.

4. **Підтримка мережевих протоколів**. Unity має вбудовані засоби для роботи з мережевими протоколами, включаючи WebSocket, що є критично важливим для реалізації комунікації між програмним забезпеченням та апаратною частиною (Arduino, ESP32) в режимі реального часу.

5. **Розвинений UI-фреймворк**. Платформа містить готові компоненти для створення інтерфейсу користувача, що значно прискорює розробку зручної системи керування для оператора.

6. **Фізичний рушій**. Вбудований фізичний рушій дозволяє моделювати кінематику робота та взаємодію з об'єктами, що суттєво полегшує розробку алгоритмів захоплення та переміщення.

7. **Активна спільнота та широка документація**. Unity має велику спільноту розробників та детальну документацію, що спрощує процес розв'язання технічних проблем під час реалізації проекту.

Для об'єктивної оцінки вибору було проаналізовано ряд альтернативних платформ, які могли б бути використані для розробки системи керування робота-маніпулятора:

1. ROS (Robot Operating System):
   - Переваги: Спеціалізована система для розробки робототехнічних додатків, великий набір готових бібліотек для керування роботами, підтримка багатьох апаратних платформ.
   - Недоліки: Висока складність освоєння, вимагає хороших знань Linux, обмежені можливості для створення інтерактивних інтерфейсів користувача, складна інтеграція з іншими компонентами системи.
2. MATLAB з Robotics Toolbox:
   - Переваги: Потужні математичні інструменти для моделювання кінематики та динаміки роботів, готові алгоритми для обробки зображень.
   - Недоліки: Висока вартість ліцензії, обмежені можливості для створення інтерактивних інтерфейсів, необхідність додаткової розробки для комунікації з апаратною частиною.
3. Unreal Engine:

• Переваги: Потужні засоби для 3D-візуалізації та реалістичного рендерингу, високопродуктивний фізичний рушій.

• Недоліки: Вища крива навчання порівняно з Unity, надмірна складність для відносно простих завдань, більш складна інтеграція з веб-технологіями.

4. Розробка власного рішення на базі OpenCV та Qt/WPF:

• Переваги: Максимальна гнучкість, можливість оптимізації під конкретну задачу, прямий доступ до низькорівневих API.

• Недоліки: Значно більші витрати часу на розробку, необхідність створення всіх компонентів системи з нуля, складність в підтримці та масштабуванні.

5. Python з бібліотеками (PyQt, OpenCV, NumPy):

• Переваги: Швидка розробка завдяки високорівневій мові, велика кількість бібліотек для обробки зображень та наукових обчислень.

• Недоліки: Обмежені можливості для 3D-візуалізації, нижча продуктивність порівняно з Unity, складність в створенні інтерактивних інтерфейсів.

Таблиця 2.1.

Порівняння платформ для розроблення системи

Критерій	**Unity**	**ROS**	**MATLAB**	**Unreal Engine**	**Python з бібліотеками**
Якість 3D-візуалізації	Висока	Середня	Середня	Дуже висока	Низька
Складність інтеграції з мікроконтроллерами	Низька	Висока	Середня	Середня	Низька
Швидкість розробки	Висока	Низька	Середня	Середня	Висока

Продовження таблиці 2.1.

Критерій	**Unity**	**ROS**	**MATLAB**	**Unreal Engine**	**Python з бібліотеками**
Можливості для створення UI	Високі	Низькі	Середні	Високі	Середні
Підтримка мережевих протоколів	Так	Так	Обмежена	Так	Так
Крос-платформність	Висока	Середня	Висока	Висока	Висока
Вартість	Безкоштовна*	Безкоштовна	Висока	Безкоштовна*	Безкоштовна
Крива навчання	Середня	Висока	Середня	Висока	Низька

*Безкоштовна, проте з певними обмеженнями для великих комерційних проектів.

На основі проведеного порівняльного аналізу можна зробити висновок, що Unity найкраще відповідає вимогам проекту, поєднуючи в собі потужні можливості для 3D-візуалізації з відносно низьким порогом входження, хорошою підтримкою мережевих протоколів та можливістю інтеграції з апаратними компонентами на базі Arduino та ESP32.

2.2. Інструменти проектування механічної частини.

Сучасне проектування роботів-маніпуляторів неможливе без використання спеціалізованого програмного забезпечення для автоматизованого проектування. CAD-системи дозволяють не лише створювати віртуальні моделі механічних

компонентів, але й проводити попередні інженерні розрахунки, аналізувати кінематику механізмів, виявляти можливі колізії та оптимізувати конструкцію перед фізичною реалізацією.

У робототехнічній галузі до систем автоматизованого проектування висуваються особливі вимоги, зумовлені специфікою створюваних механізмів:

1. Точність моделювання — забезпечення високої геометричної точності для запобігання похибок при фізичній реалізації.
2. Можливість кінематичного аналізу — інструменти для моделювання руху ланок маніпулятора та перевірки робочої зони.
3. Інтеграція з системами керування — експорт даних у формати, сумісні з програмами для розробки систем керування.
4. Модульність проектування — можливість створення складних збірок з окремих компонентів.
5. Параметричне моделювання — швидка зміна параметрів конструкції без повного перепроектування.

AutoCAD, розроблений компанією Autodesk, є одним із найпоширеніших інструментів автоматизованого проектування, який в рамках даного проєкту використовувався в навчальній версії для створення механічної частини робота-маніпулятора. Навчальна версія містить всі необхідні інструменти професійного пакету з обмеженням на комерційне використання результатів проектування.

Основні інструменти двовимірного проектування

При розробці робота-маніпулятора активно використовувались наступні можливості 2D-проектування AutoCAD:

- Точність креслення — система координат з точністю до 16 десяткових знаків, що дозволяє створювати креслення з мінімальними геометричними похибками, не критичними для механізмів робота;
- Шари — організація елементів креслення за функціональним призначенням (конструкційні елементи, електроніка, кріплення), що спрощує подальшу роботу та модифікацію проекту;

- Блоки — створення бібліотек стандартних компонентів (сервоприводи, кріпильні елементи, електронні компоненти) з можливістю їх швидкого включення в проект;
- Розмірні лінії та допуски — автоматичне нанесення розмірів з урахуванням допусків на виготовлення деталей згідно з технологічними вимогами;
- Інструменти для створення технічної документації — генерація необхідних креслень та специфікацій для виготовлення деталей відповідно до стандартів;

Основні інструменти тривимірного проектування

Для повноцінного проектування робота-маніпулятора активно використовувалися можливості 3D-моделювання AutoCAD:

- Твердотільне моделювання — створення об'ємних моделей основних конструктивних елементів маніпулятора (основа, ланки, захватний пристрій);
- Поверхневе моделювання — розробка складних геометричних форм корпусних деталей з оптимізацією за масою та міцністю;
- Операції з твердими тілами — об'єднання, віднімання та перетин об'ємів для створення складних геометричних форм;
- Функції скруглення та фаски — забезпечення технологічності виготовлення деталей та зниження концентраторів напружень;
- Дзеркальне відображення та масиви — оптимізація процесу проектування симетричних елементів конструкції;

Інтеграційні можливості AutoCAD

Важливою перевагою AutoCAD, що вплинула на його вибір для проекту, є широкі можливості інтеграції з іншими системами:

- Формати файлів — підтримка широкого спектру форматів для обміну даними з іншими програмами (.dwg, .dxf, .stl, .iges, .sat);

- Експорт в Unity — можливість експорту 3D-моделей в формати, сумісні з Unity, що використовувалася для створення системи управління роботом;
- Імпорт з інших систем — можливість імпортувати моделі та креслення з різних CAD-систем, що забезпечує гнучкість у використанні сторонніх компонентів та бібліотек (підтримка форматів .sat, .3ds, .sldprt, .step);
- Інтеграція з хмарними сервісами — можливість зберігання та спільного використання проектів через Autodesk Cloud, що спрощує колективну роботу та резервне копіювання;
- API для сторонніх розробників — наявність відкритого програмного інтерфейсу для створення спеціалізованих додатків та розширень під конкретні задачі проекту;

Для об'єктивного обґрунтування вибору AutoCAD як основного інструменту проектування механічної частини робота-маніпулятора проведено детальний порівняльний аналіз з найбільш поширеними альтернативними CAD-системами: SolidWorks, Fusion 360.

Таблиця 2.2.

Порівняння інструментів проектування

Критерії оцінки	**AutoCAD**	**SolidWorks**	**Fusion 360**
Функціональні можливості			
Точність 2D-креслення	Висока	Висока	Висока
Твердотільне моделювання	Так	Так	Так
Поверхневе моделювання	Розширене	Розширене	Розширене

Продовження таблиці 2.2.

Критерії оцінки	**AutoCAD**	**SolidWorks**	**Fusion 360**
Інтеграційні можливості			
Різноманітність форматів імпорту/експорту	Висока	Середня	Висока
Експорт в Unity	Так	Обмежений	Так
Хмарне зберігання проекту	Обмежене	Через окремі сервіси	Вбудоване
Системні вимоги та доступність			
Вимоги до комп'ютера	Помірні	Високі	Помірні
Складність освоєння	Середня	Висока	Середня
Додаткові фактори			
Розмір спільноти користувачів	Дуже великий	Великий	Середній
Можливість розширення через плагіни	Висока	Середня	Обмежена

Ключові переваги AutoCAD, що обумовили його вибір для проекту:

1. **Оптимальне співвідношення функціональності та системних вимог** — на відміну від SolidWorks, AutoCAD забезпечує необхідний функціонал без надмірних вимог до апаратного забезпечення, що особливо важливо при обмеженому бюджеті проекту.

2. **Висока стабільність роботи** — багаторічний досвід розробки та оптимізації зробив AutoCAD одним із найбільш стабільних інструментів проектування, що мінімізує ризики втрати даних під час роботи над проектом.

3. **Найбільша спільнота користувачів та найширша база знань** — доступність величезної кількості навчальних матеріалів, форумів та готових рішень суттєво спрощує процес розробки та дозволяє швидко вирішувати технічні проблеми.

4. **Відмінна підтримка форматів для обміну даними** — найширший серед конкурентів набір підтримуваних форматів файлів забезпечує безпроблемну інтеграцію з Unity, системами комп'ютерного зору та іншими компонентами проекту.

5. **Оптимальний баланс між 2D та 3D можливостями** — на відміну від SolidWorks та Fusion 360, які орієнтовані переважно на 3D-моделювання, AutoCAD забезпечує найкращі інструменти для 2D-проектування при збереженні потужних можливостей 3D-моделювання.

6. **Оптимальна крива навчання** — середня складність освоєння у поєднанні з інтуїтивно зрозумілим інтерфейсом дозволяє швидко почати продуктивну роботу без тривалого періоду навчання, необхідного для SolidWorks.

Таким чином, незважаючи на деякі обмеження порівняно зі спеціалізованими рішеннями (SolidWorks та Fusion 360), AutoCAD виявився оптимальним вибором для проекту завдяки унікальному поєднанню доступності, функціональності, стабільності та інтеграційних можливостей, що повністю відповідають вимогам розробки робота-маніпулятора.

2.3. Вибір архітектури апаратної частини

Архітектура апаратної частини робота-маніпулятора з комп'ютерним зором організована за принципом розподілу функціональних завдань між різними модулями з централізованим управлінням:

1. Центральний обчислювальний модуль — персональний комп'ютер з Unity-додатком для обробки даних комп'ютерного зору та розрахунку траєкторій руху.

2. Контролер виконавчих механізмів — Arduino UNO R3 для безпосереднього управління сервоприводами.

3. Модуль комп'ютерного зору — ESP32-CAM для отримання відеопотоку.ф

Такий підхід до організації апаратної архітектури дозволяє оптимально розподілити обчислювальне навантаження, забезпечити швидкодію системи та зберегти гнучкість для подальшого розвитку проекту.

2.3.1. Контролер виконавчих механізмів на базі Arduino UNO R3

Для управління сервоприводами робота-маніпулятора був обраний мікроконтролер Arduino UNO R3 на базі ATmega328P. Даний вибір обумовлений наступними характеристиками платформи:

- **Мікроконтролер**: ATmega328P
- **Робоча напруга**: 5 В
- **Цифрові входи/виходи**: 14 пінів (з них 6 з підтримкою PWM)
- **Аналогові входи**: 6 пінів
- **Флеш-пам'ять**: 32 КБ (з яких 0.5 КБ використовується для завантажувача)
- **SRAM**: 2 КБ
- **EEPROM**: 1 КБ
- **Тактова частота**: 16 МГц

Вибір Arduino UNO R3 як контролера виконавчих механізмів обумовлений наступними факторами:

1. Відповідність функціональним вимогам — технічні характеристики Arduino UNO R3 повністю задовольняють потреби керування сервоприводами маніпулятора, оскільки основні обчислення виконуються на потужнішому центральному комп'ютері з Unity.

2. Висока надійність та стабільність роботи — Arduino UNO R3 має підтверджену часом та численними проектами репутацію надійної платформи, що критично важливо для контролера виконавчих механізмів.

3. Розвинена екосистема та підтримка спільноти — наявність великої кількості бібліотек, прикладів коду та документації значно спрощує інтеграцію контролера в загальну систему управління.

4. Доступність та економічна ефективність — порівняно низька вартість платформи при збереженні необхідної функціональності робить її оптимальним вибором з точки зору бюджету проекту.

5. Простота інтеграції із сервоприводами — наявність спеціалізованих бібліотек для керування сервоприводами та підтримка PWM на декількох виходах дозволяє ефективно керувати множиною виконавчих механізмів.

Таблиця 2.3.

Порівняння характеристик мікроконтроллерів

Критерій	Arduino UNO R3	Raspberry Pi Pico	STM32 Blue Pill	Arduino Mega 2560
Тактова частота	16 МГц	133 МГц	72 МГц	16 МГц
Оперативна пам'ять	2 КБ	264 КБ	20 КБ	8 КБ
Кількість PWM виходів	6	16	12	15
Напруга живлення	5 В	3.3 В	3.3 В	5 В
Складність програмування	Низька	Середня	Середня	Низька
Наявність спеціалізованих бібліотек	Висока	Середня	Середня	Висока
Вартість	Низька	Низька	Низька	Середня

Як видно з табл.2.3., Arduino UNO R3 має оптимальне співвідношення технічних характеристик, простоти використання та доступності, що визначило вибір даної платформи для управління сервоприводами робота-маніпулятора.

2.3.2. Модуль комп'ютерного зору на базі ESP32-CAM

Для реалізації функцій комп'ютерного зору в проекті використовується модуль ESP32-CAM на основі мікроконтролера ESP32 з інтегрованою камерою. Ключові характеристики модуля:

- **Процесор**: Xtensa dual-core 32-bit LX6 з частотою до 240 МГц
- **Пам'ять**: 520 КБ SRAM + 4 МБ PSRAM
- **Флеш-пам'ять**: 4 МБ
- **Wi-Fi**: IEEE 802.11 b/g/n
- **Bluetooth**: BT4.2 та BLE
- **Камера**: OV2640 2 МП (1600×1200 пікселів)
- **Підтримка карт пам'яті**: MicroSD до 4 ГБ
- **Інтерфейси введення-виведення**: UART, SPI, I2C, PWM
- **Напруга живлення**: 5 В

Модуль ESP32-CAM був обраний для системи комп'ютерного зору з наступних причин:

1. **Інтеграція камери та обчислювального модуля** — поєднання мікроконтролера та камери в одному компактному модулі спрощує конструкцію та зменшує кількість компонентів системи.
2. **Вбудована підтримка Wi-Fi** — наявність вбудованого Wi-Fi модуля забезпечує бездротову передачу відеопотоку на центральний обчислювальний модуль без необхідності прокладання додаткових кабелів.
3. **Достатні характеристики камери** — хоча характеристики вбудованої камери OV2640 не є надвисокими (2 МП), вони цілком достатні для виконання базових завдань комп'ютерного зору в контрольованих умовах освітлення. Камера

забезпечує роздільну здатність до 1600×1200 пікселів, що дозволяє чітко розпізнавати об'єкти в робочій зоні маніпулятора.

4. **Висока економічна ефективність** — модуль ESP32-CAM пропонує оптимальне співвідношення функціональності та вартості, що робить його ідеальним вибором для прототипування та академічних проектів з обмеженим бюджетом.

5. **Достатня обчислювальна потужність** — хоча основні алгоритми комп'ютерного зору виконуються на центральному обчислювальному модулі в Unity, ESP32 має достатню потужність для попередньої обробки зображень та стиснення відеопотоку перед передачею.

6. **Гнучкість програмування** — модуль підтримується популярним середовищем Arduino IDE, що спрощує розробку та інтеграцію з іншими компонентами системи.

Таблиця 2.4.

Порівняння характеристик модулів камер

Критерій	ESP32-CAM	Raspberry Pi Camera Module 3	OpenMV Cam H7
Роздільна здатність	2 МП (1600×1200)	12 МП (4056×3040)	0.3 МП (640×480)
Частота кадрів	До 30 fps при 640×480	До 50 fps при 1920×1080	До 75 fps при 320×240
Вбудована обробка зображень	Обмежена	Середня	Розширена
Підтримка Wi-Fi	Вбудована	Через Raspberry Pi	Через додатковий модуль
Вартість	Дуже низька	Середня	Висока

З аналізу табл.2.4. видно, що ESP32-CAM представляє оптимальне рішення з точки зору балансу між функціональністю, простотою інтеграції та вартістю. Хоча інші рішення пропонують вищі технічні характеристики, для реалізації базового функціоналу комп'ютерного зору в контрольованому середовищі можливостей ESP32-CAM цілком достатньо, особливо з огляду на те, що основна обробка зображень відбувається в Unity на потужнішому комп'ютері.

2.4. Вибір протоколів та технологій комунікацій

Для розроблення системи управління роботом-маніпулятором з комп'ютерним зором потрібно обрати протокол комунікації між компонентами системи. Для забезпечення ефективної взаємодії між Unity-платформою та модулем ESP32-CAM необхідно проаналізувати доступні технології з огляду на вимоги проекту.

REST API

REST (Representational State Transfer) — це архітектурний стиль для розподілених систем, який визначає принципи взаємодії з ресурсами. У веб-сервісах REST зазвичай реалізується через HTTP, використовуючи методи GET, POST, PUT, DELETE для роботи з ресурсами.

Особливості REST:

- Широка підтримка у більшості мов програмування;
- Простота реалізації;
- Клієнт-серверна архітектура.

Основним недоліком для даної системи є відсутність повноцінної двосторонньої комунікації у реальному часі, що критично при управлінні роботом-маніпулятором.

UART/Serial

UART (Universal Asynchronous Receiver-Transmitter) - протокол послідовної передачі даних, часто використовуваний у мікроконтролерах:

- Простота реалізації;
- Низький рівень накладних витрат;

- Обмежена дальність передачі;
- Відсутність вбудованих механізмів виявлення та виправлення помилок вищого рівня.

WebSocket

WebSocket — це мережевий протокол, який забезпечує повнодуплексний канал зв'язку поверх одного TCP-з'єднання. На відміну від HTTP, де обмін даними відбувається у форматі "запит–відповідь", WebSocket підтримує постійне з'єднання, що дозволяє клієнту і серверу надсилати дані незалежно один від одного в реальному часі.

Таблиця 2.5.

Порівняння технологій комунікації

Характеристика	WebSocket	HTTP/REST	UART/Serial
Затримка при передачі 100 Б (мс)	5-10	50-100	1-5
Пропускна здатність (кБ/с)	До 10,000	До 5,000	До 115
Тип зв’язку	Повнодуплексний	Напівдуплексний	Повнодуплексний

У контексті розроблюваної системи управління роботом-маніпулятором з комп'ютерним зором, для комунікації в межах системи було обрано WebSocket з таких причин:

1. **Двонаправлена комунікація в реальному часі**: WebSocket забезпечує безперервний канал зв'язку, що дозволяє компонентам системи обмінюватися даними без додаткових накладних витрат на встановлення нових з'єднань. Це критично для передачі даних від модуля комп'ютерного зору (ESP32-CAM) до платформи управління (Unity) з мінімальними затримками.

2. **Гнучка архітектура з централізованим сервером**: Використання окремого WebSocket-сервера, до якого підключаються і Unity, і ESP32-CAM як

клієнти, дозволяє створити гнучку систему з можливістю легкого масштабування та заміни компонентів без перепроектування архітектури.

3. **Інтеграція з Unity**: Unity має вбудовану підтримку WebSocket через бібліотеки WebSocketSharp або NativeWebSocket, що спрощує реалізацію комунікації між інтерфейсом користувача та фізичними компонентами системи.

4. **Підтримка на ESP32**: ESP32 має офіційні бібліотеки для роботи з WebSocket (ArduinoWebsockets або ESPAsyncWebServer), що забезпечує стабільну роботу модуля комп'ютерного зору.

5. **Єдиний протокол для різних типів даних**: WebSocket підтримує як текстові повідомлення (JSON-команди, дані стану), так і бінарні дані (зображення з камери), що дозволяє використовувати єдиний протокол для всіх видів комунікації в системі.

6. **Ефективність використання мережевих ресурсів**: WebSocket значно зменшує накладні витрати на передачу даних порівняно з HTTP, що особливо важливо для передачі зображень з ESP32-CAM.

2.5. Технології комп'ютерного зору для розпізнавання об'єктів

Завдяки системам комп’ютерного зору роботи-маніпулятори можуть розпізнавати об’єкти та точно взаємодіяти з ними, що є ключовим для виконання складних завдань в реальному середовищі. Стрімкий розвиток машинного навчання дозволив реалізувати алгоритми розпізнавання на пристроях з обмеженими ресурсами, що є важливим для вбудованих систем.

Нижче проведено аналіз технологій комп'ютерного зору, оцінено їх придатність для інтеграції з робототехнічними системами та обґрунтовано вибір оптимального рішення.

TensorFlow Lite

TensorFlow Lite представляє собою полегшену версію фреймворку TensorFlow, оптимізовану для мобільних та вбудованих пристроїв. Дана технологія дозволяє розгортати моделі машинного навчання на пристроях з обмеженими обчислювальними ресурсами.

Переваги:

- Висока продуктивність на мобільних пристроях;
- Широкий спектр попередньо навчених моделей;
- Потужна екосистема та підтримка спільноти;

Недоліки:

- Потребує відносно високих обчислювальних ресурсів для ESP32;
- Складність в інтеграції з мікроконтролерними системами;
- Значний обсяг вихідного коду для адаптації до конкретного застосування;

OpenCV

OpenCV (Open Source Computer Vision Library) є відкритою бібліотекою комп'ютерного зору, яка містить понад 2500 оптимізованих алгоритмів.

Переваги:

- Багатий набір функцій для обробки зображень;
- Висока продуктивність;
- Підтримка багатьох платформ;

Недоліки:

- Потребує значних обчислювальних ресурсів;
- Складна у впровадженні на мікроконтролерах;
- Має високий поріг входження для початківців;

Google MobileNet

Google MobileNet — це сімейство ефективних моделей комп'ютерного зору, оптимізованих для мобільних та вбудованих пристроїв.

Переваги:

- Висока точність розпізнавання при невеликому розмірі моделі;
- Ефективне використання обчислювальних ресурсів;
- Хороша швидкість обробки;

Недоліки:

- Потребує додаткових інструментів для розгортання на мікроконтролерах;

- Відсутність інтегрованого середовища розробки;
- Складність адаптації до специфічних задач;

Edge Impulse

Edge Impulse — це платформа для розробки та розгортання моделей машинного навчання безпосередньо на крайових пристроях (edge devices).

Переваги:

- Інтегроване середовище для збору даних, навчання та розгортання;
- Оптимізація моделей під конкретні мікроконтролери;
- Підтримка ESP32 та Arduino як цільових платформ;
- Низький поріг входження завдяки інтуїтивному інтерфейсу;
- Генерація готового коду для цільової платформи;

Недоліки:

- Обмежений набір попередньо навчених моделей;
- Деякі просунуті функції доступні лише в платних тарифах;
- Залежність від онлайн-сервісу для навчання моделей;

Вибір технології Edge Impulse для реалізації системи комп'ютерного зору в роботі-маніпуляторі обґрунтовується наступними факторами:

1. Інтеграція з цільовими платформами — Edge Impulse має нативну підтримку ESP32-CAM та Arduino, що використовуються в даному проєкті, та забезпечує безпроблемну інтеграцію з цими пристроями.
2. Оптимізація для обмежених ресурсів — технологія дозволяє створювати моделі, оптимізовані для роботи на пристроях з обмеженими обчислювальними можливостями, що критично для ESP32-CAM.
3. Комплексне рішення — платформа пропонує повний цикл розробки: від збору та анотації даних до навчання моделі та її розгортання на пристрої.
4. Автоматична генерація коду — Edge Impulse автоматично генерує оптимізований код для цільової платформи, що значно скорочує час розробки та зменшує ймовірність помилок.

5. Висока точність розпізнавання — незважаючи на оптимізацію для вбудованих систем, технологія забезпечує достатню точність для виконання завдань розпізнавання об'єктів у контексті робота-маніпулятора.

Принципи роботи технології розпізнавання образів в Edge Impulse

Технологія розпізнавання образів, яка застосовується в Edge Impulse, базується на глибоких згорткових нейронних мережах (CNN), рис.2.1. Процес розпізнавання об'єктів включає наступні етапи:

1. Збір та підготовка даних

Процес починається зі збору репрезентативного набору зображень об'єктів, які повинен розпізнавати робот-маніпулятор. За допомогою веб-інтерфейсу Edge Impulse ці дані завантажуються, сегментуються та анотуються для подальшого використання при навчанні моделі.

2. Попередня обробка зображень

Зображення проходять етап попередньої обробки, який включає:

- Нормалізацію розміру зображень;
- Корекцію яскравості та контрасту;
- Фільтрацію шумів;
- Сегментацію області інтересу;

3. Вилучення ознак (Feature Extraction)

На цьому етапі з зображень вилучаються характерні ознаки, які дозволяють розрізняти різні об'єкти. Edge Impulse використовує згорткові шари нейронної мережі для виконання цієї задачі.

4. Класифікація

Після вилучення ознак наступний етап — класифікація об'єктів на зображенні. В Edge Impulse використовуються повнозв'язні шари нейронної мережі, які аналізують вилучені ознаки та визначають ймовірність приналежності об'єкта до певного класу.

5. Навчання моделі

Навчання моделі відбувається з використанням алгоритму зворотного поширення помилки (backpropagation), де ваги нейронної мережі коригуються для мінімізації різниці між прогнозованим та фактичним класом об'єкта.

6. Генерація коду та розгортання

Після оптимізації Edge Impulse автоматично генерує C/C++ код, який можна інтегрувати в систему управління, або безпосередньо на платформі ESP32-CAM або Arduino. Цей код включає всі необхідні функції для роботи з моделлю та інтерпретації результатів.

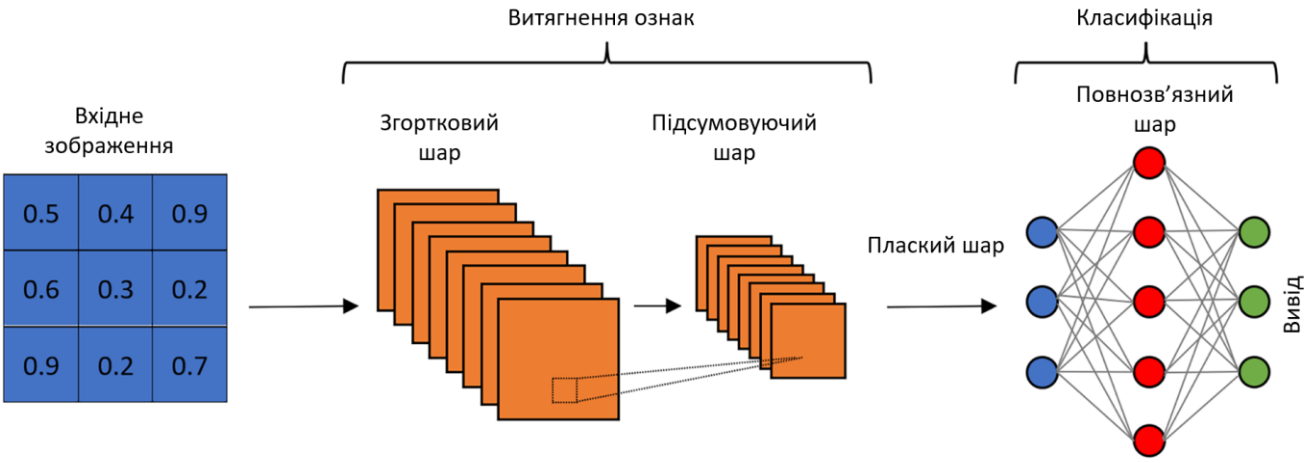


Рис.2.1. Принцип роботи згорткових нейронних мереж

3. Апаратно-програмна реалізація системи.

3.1. Загальна архітектура системи

Розроблена система управління роботом-маніпулятором з комп’ютерним зором реалізує функціональну інтеграцію апаратного та програмного забезпечення на основі сучасних мікроконтролерів, візуального середовища Unity, а також технологій штучного інтелекту для розпізнавання об’єктів. Архітектура системи побудована за модульним принципом, що забезпечує простоту масштабування та адаптації під інші задачі.

Система включає наступні основні компоненти:

- **Unity-середовище**, що виконує роль центрального керуючого модуля, візуалізує модель робота, обробляє відео з камери, виконує розпізнавання об'єктів і розраховує положення маніпулятора;
- **Камера ESP32-CAM**, яка передає відеопотік у вигляді сирих байтів (raw bytes) через WebSocket у Unity;
- **Сервер WebSocket**, реалізований для обміну даними між ESP32-CAM та Unity, а також для передачі конфігураційних параметрів до камери (яскравість, контраст тощо);
- **Модель розпізнавання об'єктів**, створена в середовищі Edge Impulse, експортована до Unity у вигляді бібліотеки;
- **Алгоритм зворотної кінематики**, реалізований у Unity для обчислення необхідних положень сервоприводів на основі координат об'єкта;
- **Мікроконтролер Arduino**, який отримує цільові кути обертання сервоприводів від Unity та безпосередньо керує приводами маніпулятора.

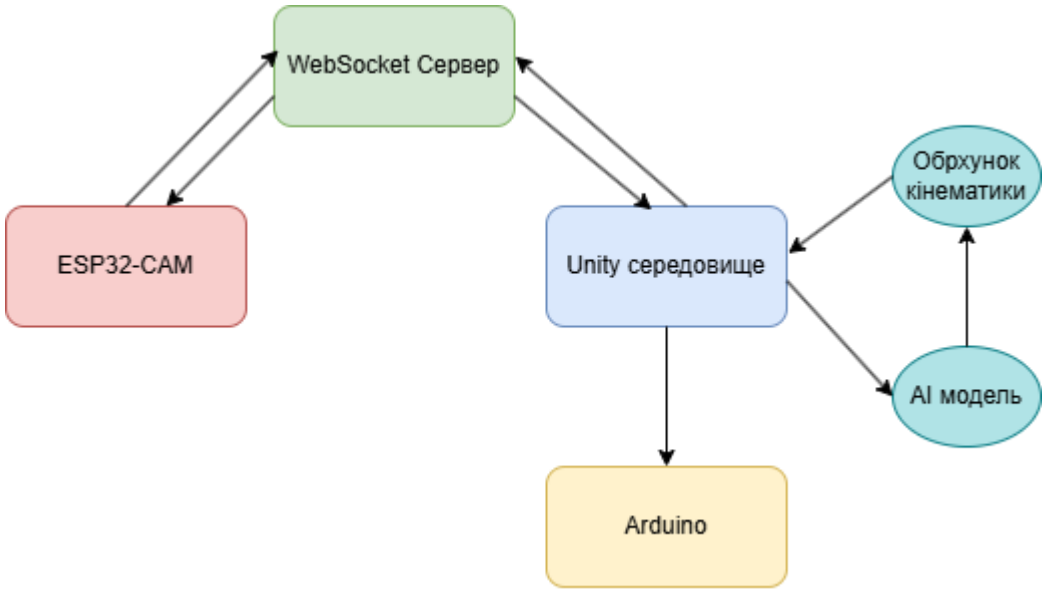


Рисунок 3.1 – Загальна архітектура системи управління роботом-маніпулятором.

Відеосигнал з камери надходить у вигляді байтового потоку через WebSocket-з’єднання до Unity. Отримане зображення передається до моделі розпізнавання, яка визначає наявність цільового об’єкта. Після цього координати об’єкта перетворюються у світові координати сцени Unity, а алгоритм зворотної кінематики розраховує відповідні кути для кожного суглоба маніпулятора. Розраховані параметри передаються на Arduino, який генерує сигнали керування сервоприводами.

Таким чином, система забезпечує повністю автономне визначення положення об’єкта, розрахунок руху маніпулятора та фізичне керування ним у реальному часі.

3.2. Проектування робота-маніпулятора

Визначення вимог до конструкції

Робот має настільну модифікацію та призначений для роботи в умовах обмеженого простору. Основними вимогами до конструкції були:

- Модульність — усі елементи виготовляються окремо, що дозволяє легко замінювати пошкоджені чи невдалі компоненти.
- Простота монтажу та демонтажу — конструкція має зручні з'єднання гвинтами та болтами.

- Підтримка технології 3D-друку — усі деталі проектувались з урахуванням можливості виготовлення на побутовому 3D-принтері.
- Обмеження по складності — було обрано 4 ступені свободи, щоб зберегти баланс між функціональністю та простотою реалізації.

Завдяки зазначеним критеріям, робот є доступним як для самостійного складання, так і для використання у навчальних лабораторіях або хобі-проєктах.

Кінематична схема

Кінематична структура маніпулятора включає такі основні вузли:

1. Основа з поворотним механізмом, яка дозволяє обертати всю конструкцію навколо вертикальної осі. Для реалізації цього обертання використовується кругла напрямна рейка з підшипниками.
2. Плече, яке складається з двох симетричних паралельних деталей, що забезпечують підвищену жорсткість конструкції. Воно під’єднане до основи за допомогою підшипника, що дозволяє вільне обертання.
3. Передпліччя — також виконане у вигляді подвійної симетричної конструкції та з’єднане з плечем на підшипнику.
4. Механізм захвату — встановлений на кінці передпліччя, жорстко закріплений до сервопривода, яким безпосередньо приводиться у дію.

Рух у кожному з трьох основних шарнірів забезпечується окремими сервоприводами. Конструкція забезпечує базову маніпуляційну здатність у вертикальній площині, а також дозволяє роботу обертатись навколо своєї осі.

Моделювання в AutoCAD

Першим ділом було змодельовано моделі-заглушки для сервоприводів. Хоч ці деталі не будуть надруковані на 3D-принтері, вони сильно допомагають в процесі проектування, визначення потрібних розмірів та перевірки на коректність під час моделювання. Оскільки деталізація для таких моделей не потрібна, вони були зроблені досить примітивними, проте з чітким дотриманням розмірів. В проекті використовується 3 різних сервоприводи, але в двох з них співпадають розміри та форма, тому було створено дві модель, які зображені на рис. 3.1.

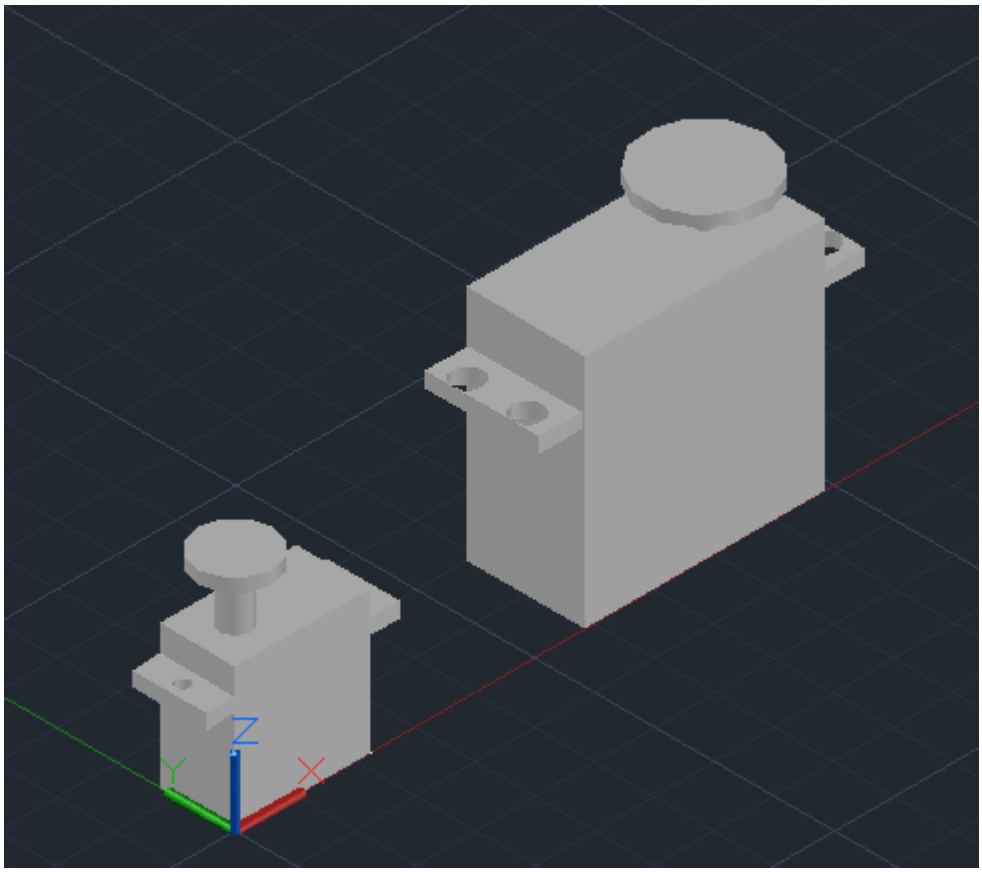


Рис. 3.1. Примітивні моделі сервоприводів MG90s(SG90) та MG995.

Оскільки використані в проекті сервоприводи йдуть з вже готовими наконечниками, було вирішено цим скористатись та зробити отвори в місцях кріплення деталей до сервоприводів, як на рис. 3.2, рис. 3.3. та рис. 3.4.

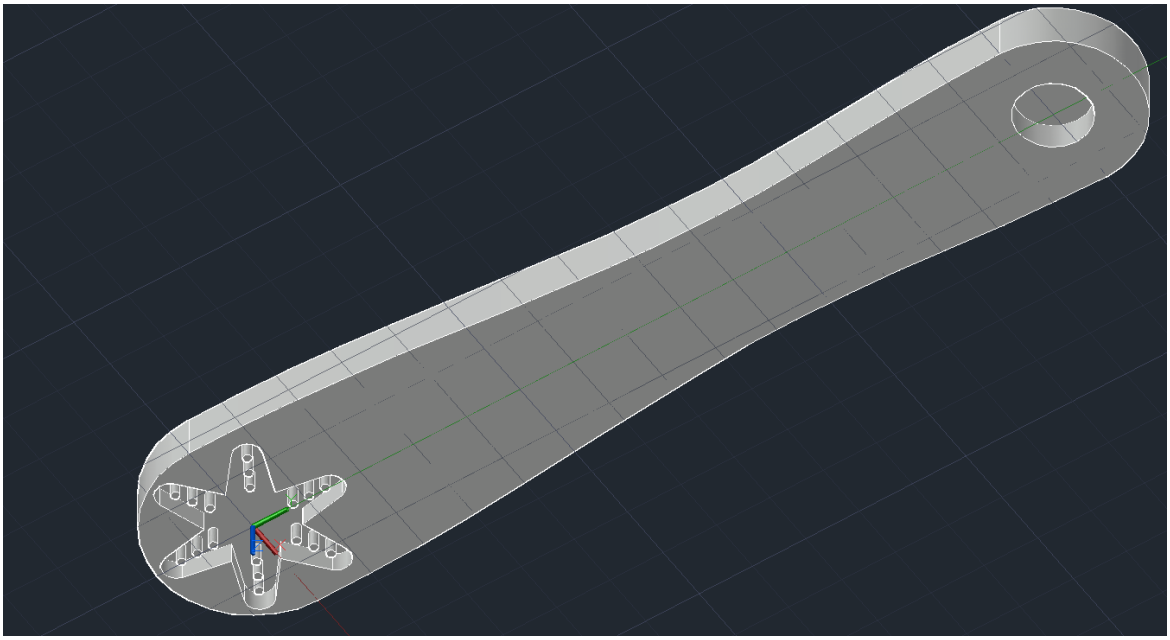


Рис. 3.2. Праве плече робота, яке кріпиться до сервопривода MG995.

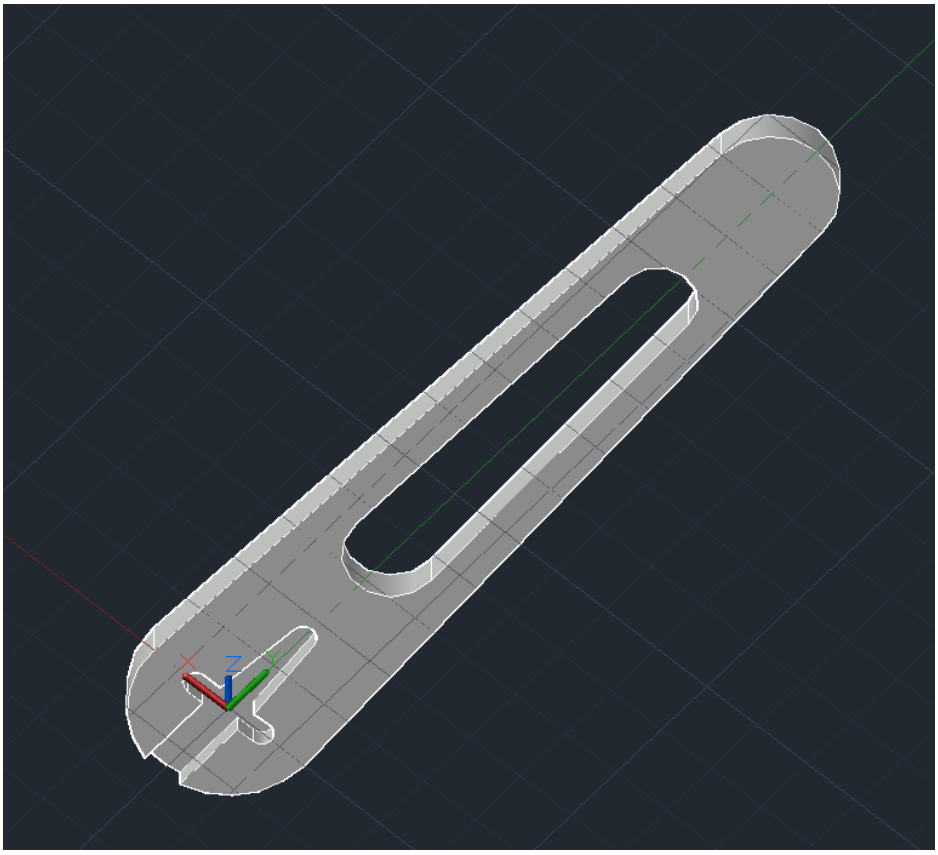


Рис. 3.3. Ліве передпліччя робота, яке кріпиться до сервопривода MG90s.

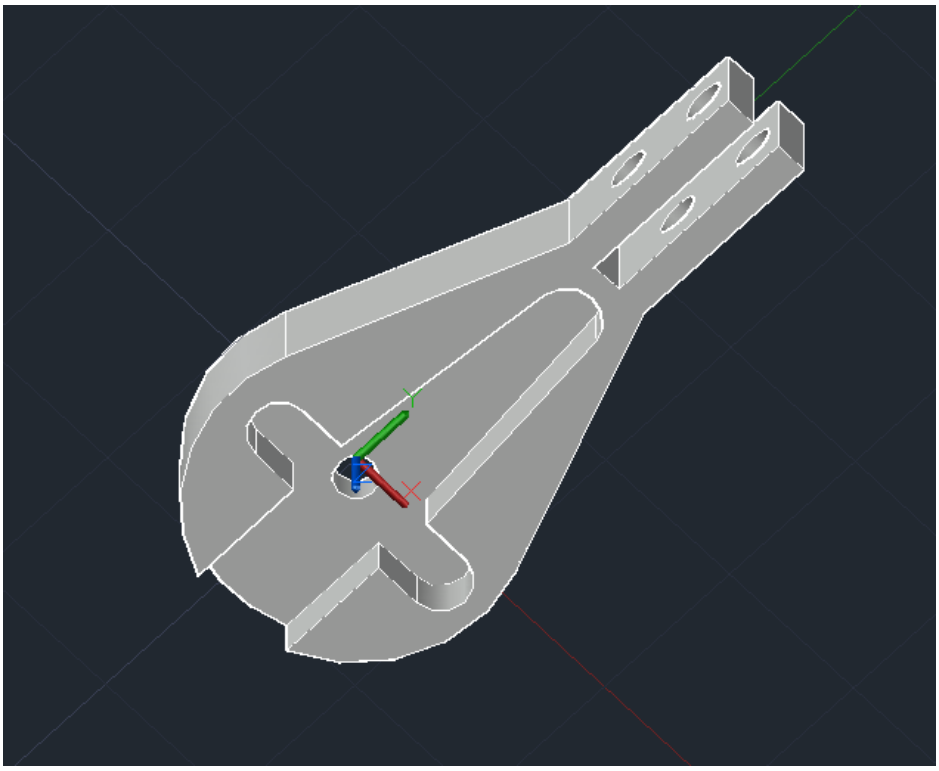


Рис. 3.4. Кріплення механізму захвату до сервопривода MG90s.

Як видно на рис. 3.2., в плечах робота передбачено отвори під підшипники. Вони запресовуються в середину та слугують механізмом крутіння в місцях суглобів.

Для реалізації механізму захвату було обрано конструкцію, що базується на зубчастій передачі (шестернях), як зображено на рис. 3.5. По-перше, шестерні дозволяють синхронізувати рух обох "пальців" захвату, забезпечуючи рівномірне відкривання та закривання, що важливо для надійного утримання об’єктів та уникнення перекосу. По-друге, така передача забезпечує механічне посилення зусилля, що дозволяє використовувати малопотужний сервопривід (наприклад, SG90). Крім того, шестерні легко виготовляються за допомогою 3D-друку і не потребують складного механічного обслуговування, що відповідає загальній концепції доступності, простоти й модульності конструкції.

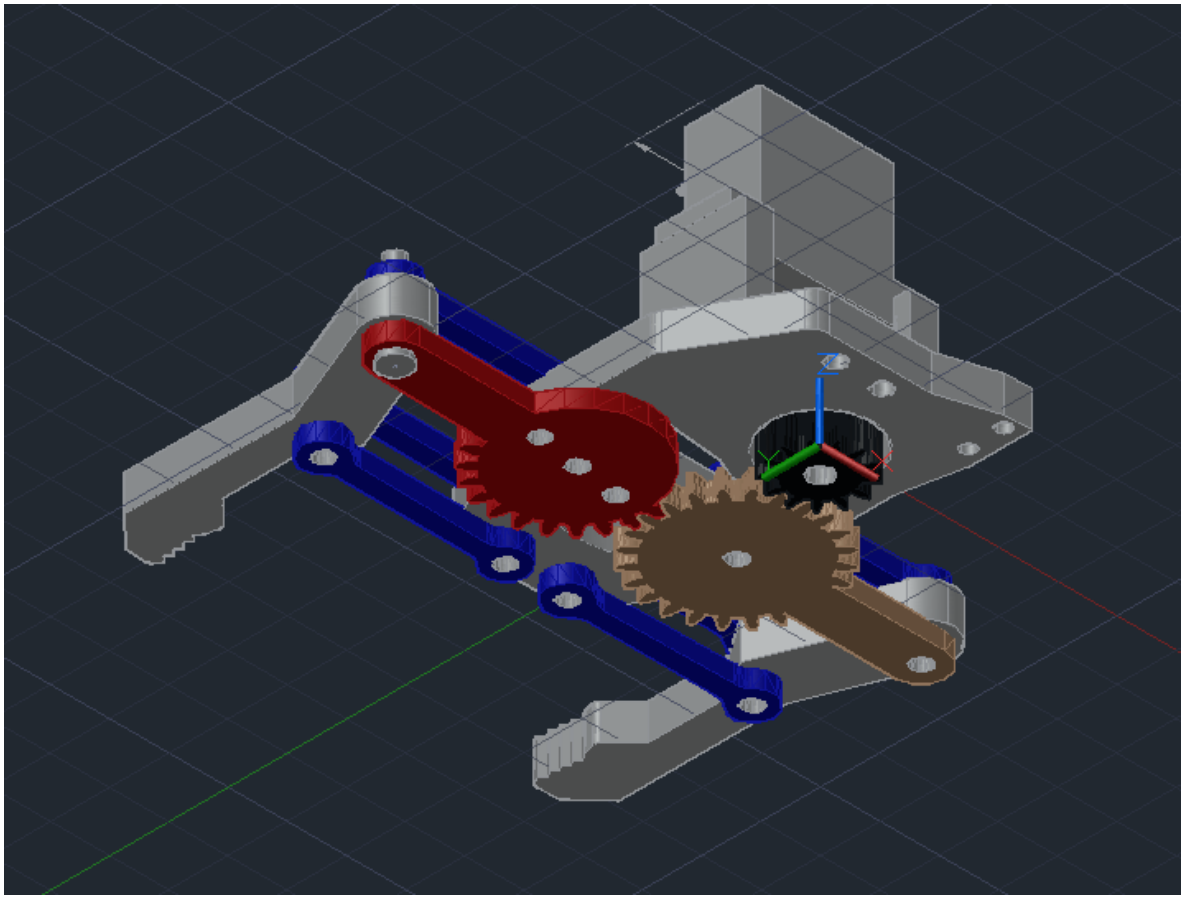


Рис. 3.5. Механізм захвату робота.

Після того, як усі деталі були змодельовані, було виконано повну збірку робота в системі AutoCAD (рис. 3.6) для перевірки правильності обрахунків,

виявлення місць пересікань, якщо такі існують, і для загальної оцінки вигляду макета.

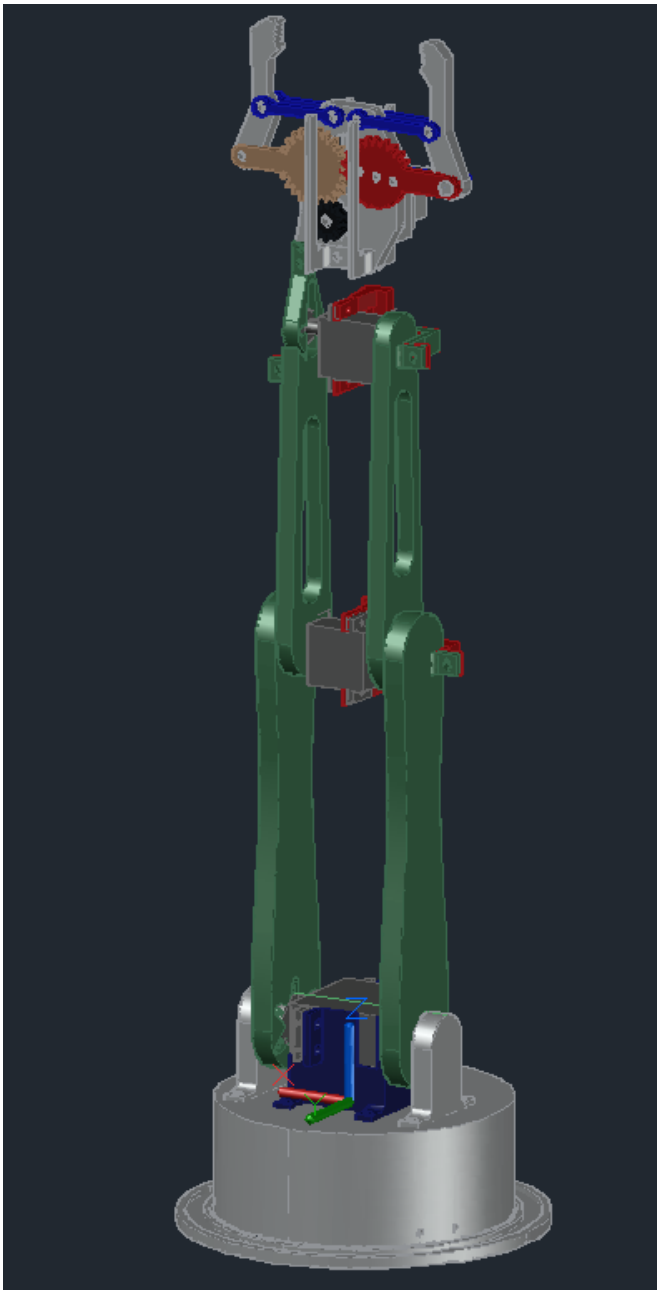


Рис. 3.6. Збірка робота в системі AutoCAD.

Експорт моделі

Наступним етапом потрібно було експортувати модель робота в Unity. Найкращим форматом для цього є .fbx, але AutoCAD не підтримує цей формат експорту напряму. Було вирішено застосувати додатковий інструмент моделювання як посередник між основною системою проектування та платформою керування. Найкращим варіантом для цього виявився AutoDesk Fusion 360.

Модель робота було експортовано з AutoCAD в Fusion 360 у форматі .sat, а після цього експортована з Fusion 360 як .fbx. Перевагами цього формату над іншими, наприклад, .obj, полягають в тому, що модель зберігається не однією сіткою скінчених елементів, а поділена на підоб’єкти, з яких легко скласти ієрархію. Це важливо в тому випадку, коли модель повинна бути не статична, а мати рухомі елементи, як в нашому випадку, де потрібно візуалізувати повороти кісток робота в місцях суглобів, або показати принцип роботи механізму захвату, який можна реалізувати за допомогою фізичного рушія в Unity.

Нижче зображено імпортовану в платформу керування модель робота, рис. 3.7.

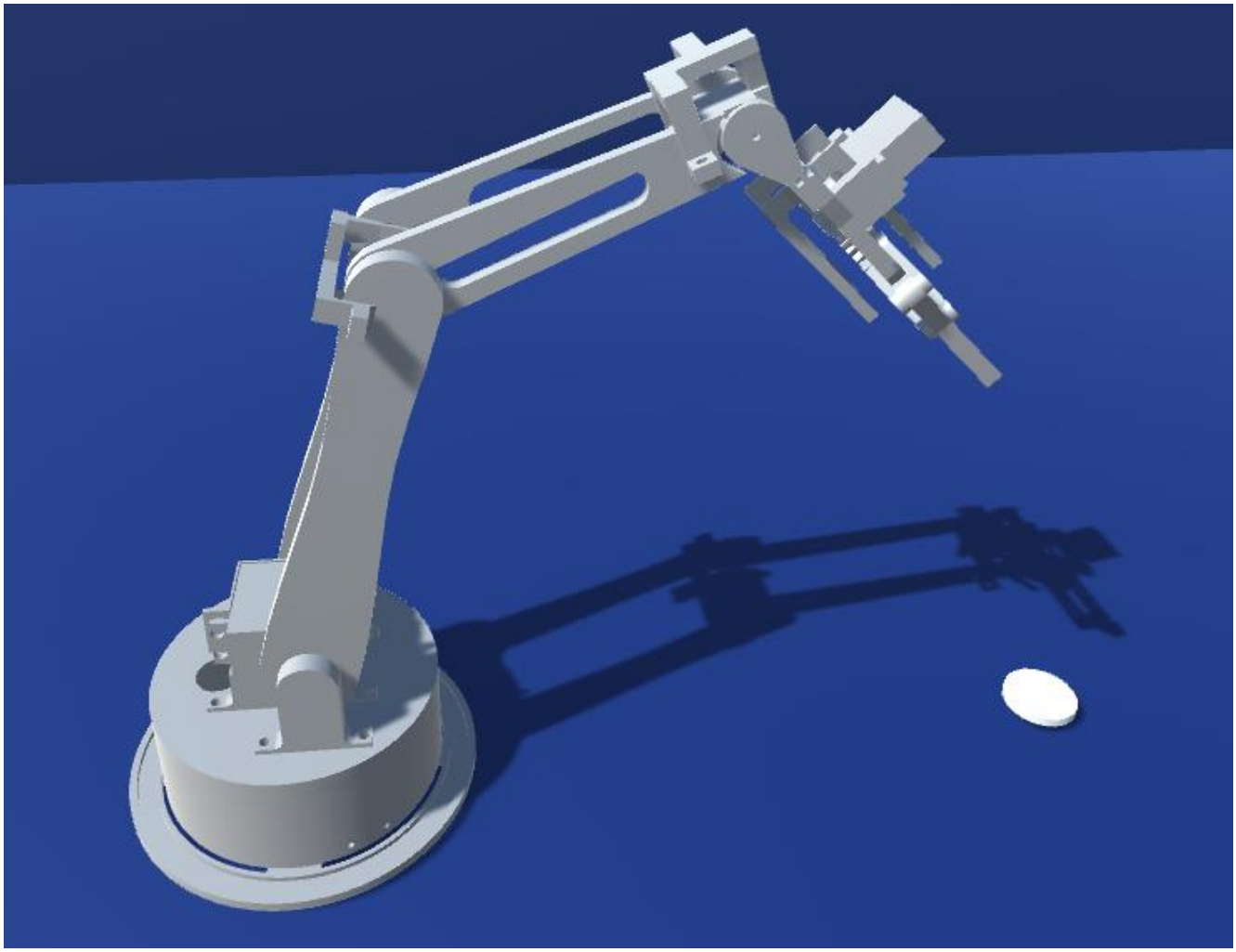


Рис. 3.7. Модель робота в середовищі Unity.

3.3. Передача відеопотоку та зв’язок між компонентами

Система обміну даними між ESP32-CAM та програмною частиною в Unity побудована на базі WebSocket-з’єднання. Такий підхід забезпечує двосторонній асинхронний обмін повідомленнями з мінімальною затримкою, що є критично важливим для інтерактивних систем керування та комп’ютерного зору.

3.3.1. Реалізація WebSocket-сервера

WebSocket-сервер було реалізовано для забезпечення централізованої точки комунікації між ESP32-CAM і Unity. Сервер запускається на хості, в даному випадку на ПК, який виступає у ролі посередника між камерою та середовищем візуалізації/керування.

WebSocket-сервер написно на мові програмування JavaScript з використанням платформи Node.js та бібліотеки ws. Сервер також містить HTTP-обробник, який у разі потреби може надавати веб-клієнту HTML-сторінку.

Основні характеристики сервера:

- Порт WebSocket: 8888
- Порт HTTP: 8000
- Бібліотеки: express, ws, path
- Принцип роботи: Сервер приймає WebSocket-з’єднання від клієнтів (ESP32 та Unity), приймає повідомлення (кадри зображень, параметри), і розсилає їх до всіх інших клієнтів.

Програмна реалізація сервера складається з двох скриптів(рис. 3.8., рис. 3.9):

- server.js, в якому міститься логіка сервера
- client.html – скрипт базової сторінки

server.js

C: > UnityProjects > RoboticArm > CameraWebSocket > server.js > ...

```
const path = require('path');
const express = require('express');
const WebSocket = require('ws');
const app = express();

const WS_PORT  = 8888;
const HTTP_PORT = 8000;

const wsServer = new WebSocket.Server({port: WS_PORT}, ()=> console.log(`WS Server is listening at ${WS_PORT}`));

let connectedClients = [];
wsServer.on('connection', (ws, req)=>{
    console.log('Connected');
    connectedClients.push(ws);

    ws.on('message', data => {
        connectedClients.forEach((ws,i)=>{
            if(ws.readyState === ws.OPEN){
                ws.send(data);
            }else{
                connectedClients.splice(i ,1);
            }
        })
    });
});

app.get('/client',(req,res)=>res.sendFile(path.resolve(__dirname, './client.html')));
app.listen(HTTP_PORT, ()=> console.log(`HTTP server listening at ${HTTP_PORT}`));
```

Рис.3.8. Скрипт логіки сервера

client.html

C: > UnityProjects > RoboticArm > CameraWebSocket > client.html > html

```
<html>

<head>
    <title>Client</title>
</head>

<body>
    <img src="">
    <script>
        const img = document.querySelector('img');
        const WS_URL = 'ws:///192.168.0.113:8888';
        const ws = new WebSocket(WS_URL);
        let urlObject;
        ws.onopen = () => console.log(`Connected to ${WS_URL}`);
        ws.onmessage = message => {
            const arrayBuffer = message.data;
            if (urlObject) {
                URL.revokeObjectURL(urlObject);
            }
            urlObject = URL.createObjectURL(new Blob([arrayBuffer]));
            img.src = urlObject;
        }
    </script>
</body>

</html>
```

Рис.3.9. Скрипт базової HTML-сторінки

Для того, щоб запустити WebSocket сервер, у терміналі потрібно виконати команду *node server.js*, перебуваючи в папці з цим файлом.

Після цього для перевірки правильності виконання, можна зайти в браузері за посиланням http://localhost:8000/client, як зображено на рис. 3.10.

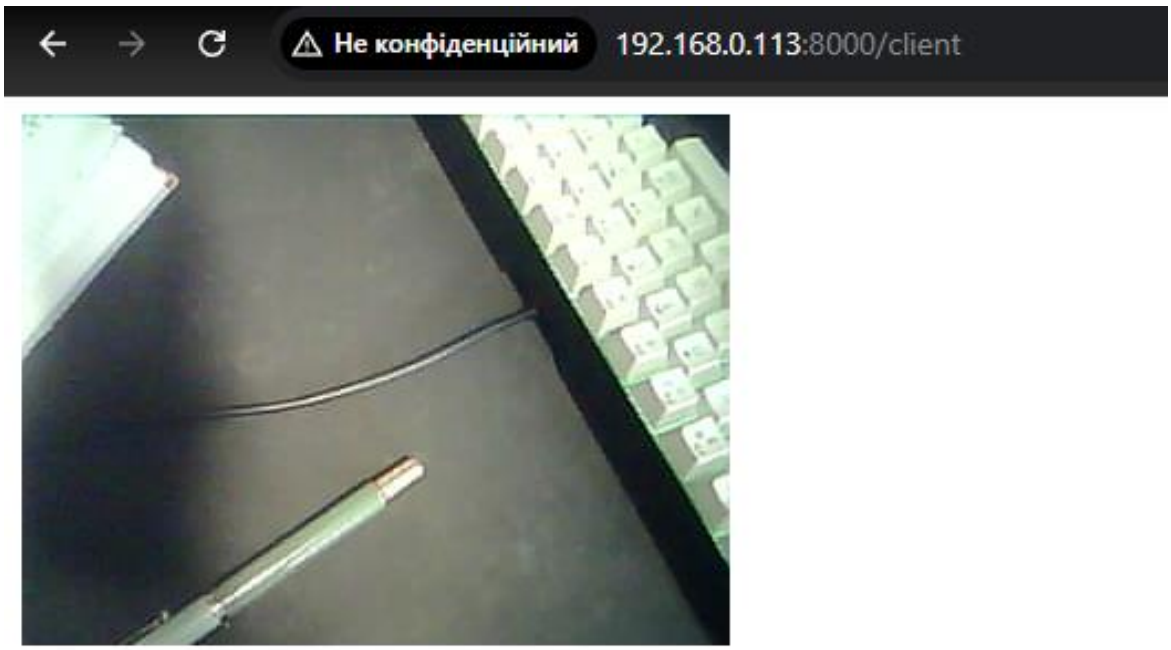


Рис. 3.10. Базова сторінка сервера, з вже підключеною камерою

3.3.2. Підключення ESP32-CAM до WebSocket-сервера та передача відео

Для передачі зображення з камери ESP32-CAM до системи візуалізації в Unity, реалізовано підключення ESP32 до WebSocket-сервера. Це забезпечує стабільний двосторонній зв’язок, що дозволяє не лише передавати зображення, а й отримувати команди для зміни параметрів камери.

Для цього використовуються такі бібліотеки:

- esp_camera.h
- WiFi.h
- ArduinoWebsockets.h
- ArduinoJson.h

Підключення до Wi-Fi та ініціалізація камери

Для того, щоб передавати дані, спочатку потрібно підключитись до мережі. Оскільки наша мережа локальна, потрібно, щоб усі компоненти системи були підключені до спільної мережі. Для цього на камері ESP32-CAM виконується команда:

WiFi.begin(ssid, password),

де *ssid* – назва мережі Wi-Fi, а *password* – пароль до неї.

Підключення до WebSocket-сервера

Після підключення до мережі, ESP32 виконує підключення до WebSocket-сервера. Для цього виконується команда:

client.connect(websocket_server_host, websocket_server_port, "/"),

де *websocket_server_host* та *websocket_server_port* – IP-адреса пристрою, на якому запущений сервер, та порт, на якому працює сервер, відповідно.

Передача відеопотоку

У циклі loop() відбувається постійна зйомка кадру та передача його у форматі JPEG через WebSocket командою:

client.sendBinary((const char) fb->buf, fb->len)*,

де *fb->buf* та *fb->len* – буффер та довжина буфферу відповідно.

3.3.3. Прийом відео в Unity, декодування та відображення

Для прийому відео з камери ESP32-CAM в Unity використовується бібліотека NativeWebSocket, яка забезпечує підтримку WebSocket-з'єднань у кросплатформенних проєктах на Unity.

Основні етапи реалізації:

1. **Ініціалізація WebSocket-з’єднання:** На початку виконання скрипта створюється об’єкт WebSocket та встановлюється підключення до сервера за локальною IP-адресою, де працює WebSocket-сервер.
2. **Обробка вхідного відеопотоку:** Після отримання байтів з WebSocket-повідомлення, відбувається декодування зображення у формат JPEG або

Uncompressed (в залежності від налаштувань). Якщо задано формат Uncompressed, зображення без жодних змін завантажеться в текстуру. Якщо заданий формат - JPEG, то виконується обробка зображення, тобто форматування «сирих» даних в JPEG стандарт; обрізається та масштабується таким чином, щоб зображення було підтримуваним алгоритмами комп’ютерного зору. Формат декодування та розміри зображення задаються в файлі конфігурації Config.

3. Вивід в інтерфейс користувача

У скрипті CamToTexture (див. Додаток А) кадри автоматично оновлюються кожного разу при отриманні нового зображення, забезпечуючи режим реального часу. Об’єкт RawImage, що прив’язаний до UI, оновлюється кожним кадром, створюючи ефект відеопотоку. На рис. 3.11. зображено вивід відеопотоку в середовищі Unity.

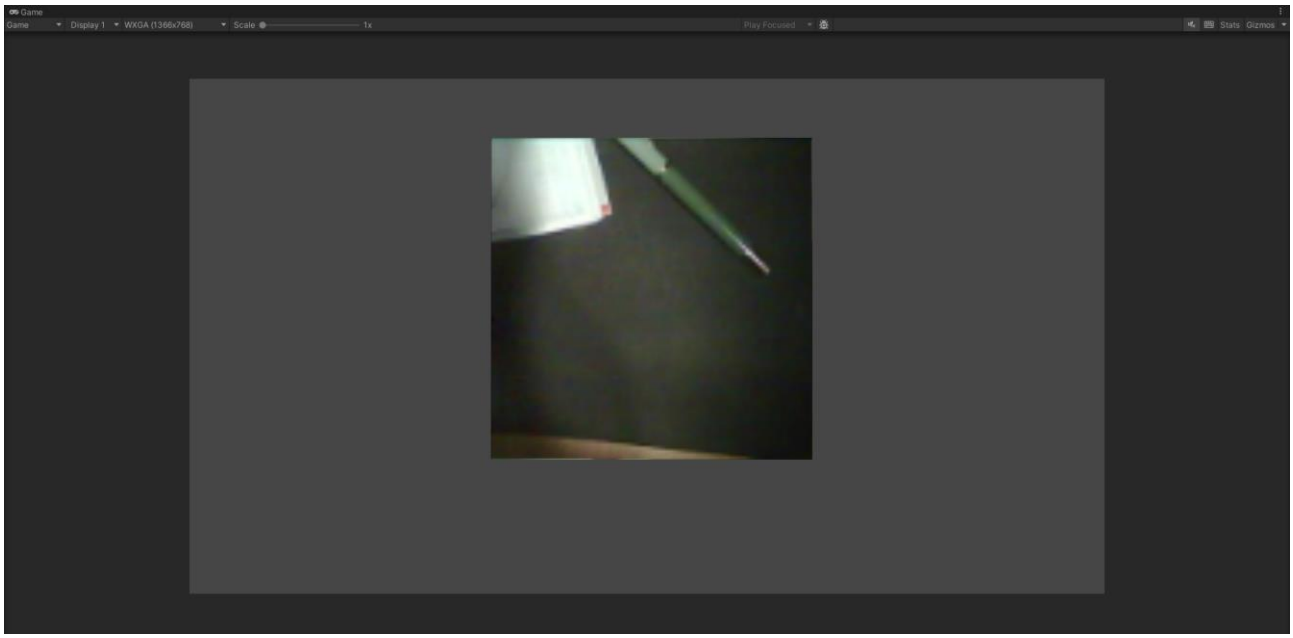


Рис. 3.11. Вивід відеопотоку в середовищі Unity.

3.3.4. Віддалене налаштування параметрів камери

Потрібним функціоналом є можливість змінювати параметри камери для досягнення потрібного результату якості зображення та продуктивності. Оскільки WebSocket дозволяє одночасно відправляти та отримувати дані, було реалізовано передачу даних в JSON форматі на камеру ESP-32 CAM.

Відправлення даних на Unity

WebSocket дозволяє надсилати дані в форматі JSON, тому це було використано в цілях надсилання команд. Приклад структури JSON-повідомлення:

```
{
"command": "set_parameter",
"parameter": "brightness",
"value": 1
}
```

За допомогою цієї структури можна надсилати різні команди, в тому числі команди для зміни параметрів.

Отримання даних на ESP32-CAM

Коли на WebSocket надходить нове повідомлення, камера намагається розпарсити повідомлення в текст. Якщо це вдається, то повідомлення розбивається на частини та визначається, яку команду треба виконати та які параметри потрібно змінити. Для цього використовується перебір наперед заготовлених команд, серед них:

- brightness – яскравість, діапазон від -2 до 2;
- contrast – контрасність, діапазон від -2 до 2;
- saturation – насиченість, діапазон від -2 до 2;
- framesize – роздільна здатність кадру, наприклад, FRAMESIZE_QVGA;

Це дозволяє динамічно адаптувати параметри зображення під поточні умови зйомки напряму з Unity, як показано на рис. 3.12.

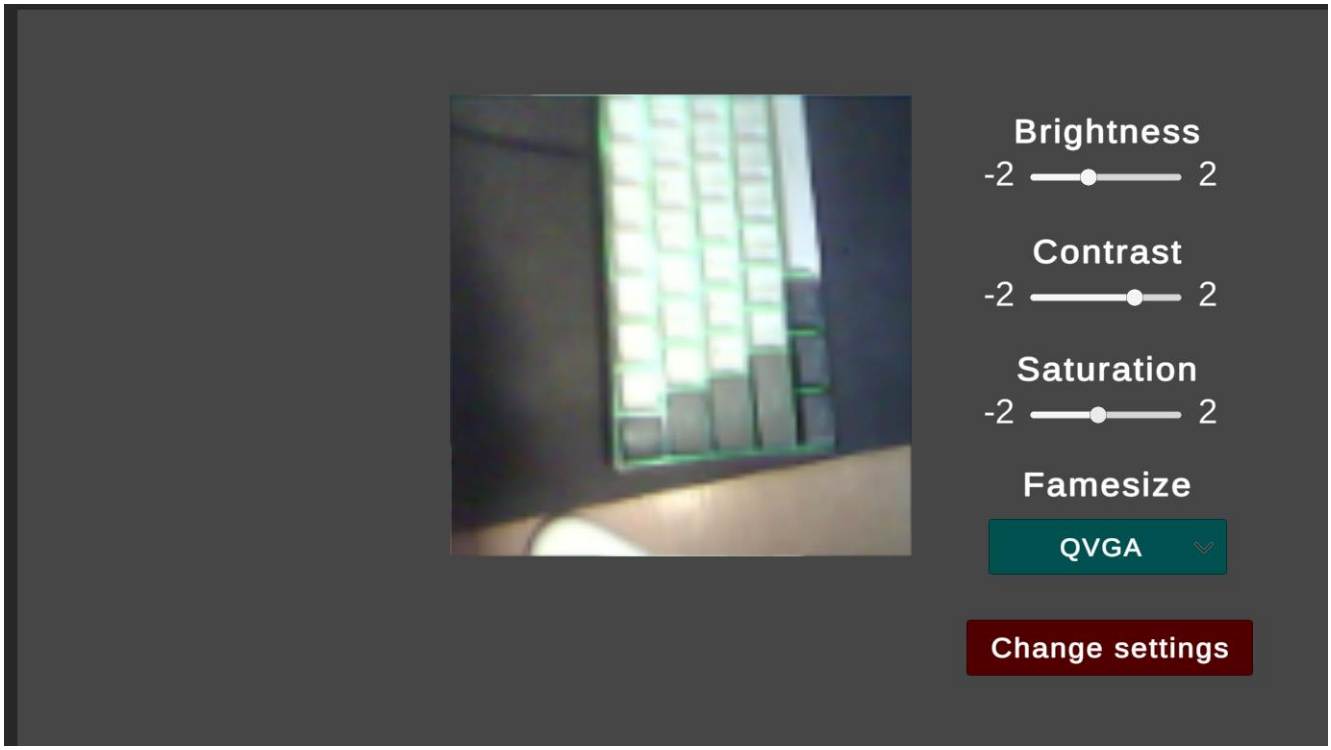


Рис. 3.12. Користувацький інтерфейс задання параметрів камери.

3.4. Система комп’ютерного зору

3.4.1. Розробка моделі в Edge Impulse

Розробка моделі системи комп’ютерного зору починається на платформі Edge Impulse, яка надає зручний веб-інтерфейс для створення, навчання та тестування моделей машинного навчання.

Створення проєкту та налаштування параметрів

На початковому етапі створюється новий проєкт для класифікації зображень. Визначаються основні параметри: розмір вхідного зображення, кількість класів, архітектура нейромережі, а також обирається структура обробки вхідних даних.

Збір і підготовка даних

Для навчання системи комп’ютерного зору, що виконує розпізнавання об’єктів, необхідною умовою є якісна та репрезентативна навчальна вибірка. У рамках розробки була реалізована власна система збору зображень безпосередньо з камери **ESP32-CAM**, що використовується у фінальній конфігурації робота. Це дозволяє отримати максимально релевантні дані без необхідності в додатковій камері або лабораторному стенді.

У середовищі **Unity** розроблено сцену, в якій приймається відеопотік у форматі raw-байтів обробляється і візуалізується в реальному часі у вигляді текстури (рис. 3.13.). За ініціативою користувача поточне зображення з камери може бути збережене у форматі **JPG** безпосередньо з Unity. Таким чином, реалізовано інтерактивний процес збирання знімків, що дозволяє оператору швидко змінювати положення об’єктів у полі зору камери і фіксувати нові варіанти без затримок.

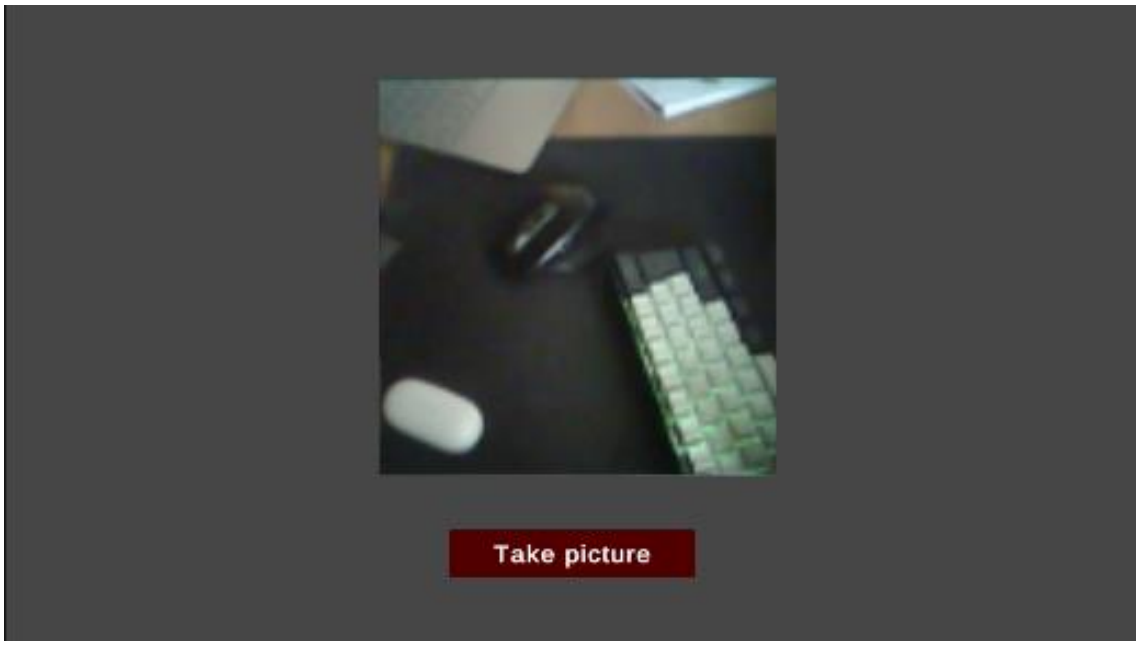


Рис. 3.13. Приклад користувацького інтерфейсу збору даних для моделі.

PictureTaker.cs

C: > UnityProjects > RoboticArm > ArduinoProject > Assets > PictureTaker.cs > ...

```
using System;
using System.IO;
using UnityEngine;

0 references
public class PictureTaker : MonoBehaviour
{
    1 reference
    [SerializeField] private CamToTexture camToTexture;
    1 reference
    [SerializeField] private string folderPath;


    1 reference
    public void SaveJPG()
    {
        string name = $"Image_{DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss")}.jpg";
        SaveJPG(camToTexture.tex, name);
    }

    1 reference
    void SaveJPG(Texture2D tex, string fileName)
    {
        if (tex == null)
        {
            Debug.LogError("Texture is null!");
            return;
        }

        byte[] bytes = tex.EncodeToJPG(90); // 90 is the quality (0-100)
        string path = Path.Combine(folderPath, fileName);

        File.WriteAllBytes(path, bytes);

        Debug.Log("Saved image to: " + path);
    }
}

```

Рис. 3.14. Приклад коду для збереження зображень.

Після збору зображень, JPG-файли групуються за класами та анотуються відповідно до типу моделі (наприклад, обмежувальними рамками для детекції, рис. 3.15.) напряму у середовищі **Edge Impulse** для подальшого використання при навчанні моделі штучного інтелекту.

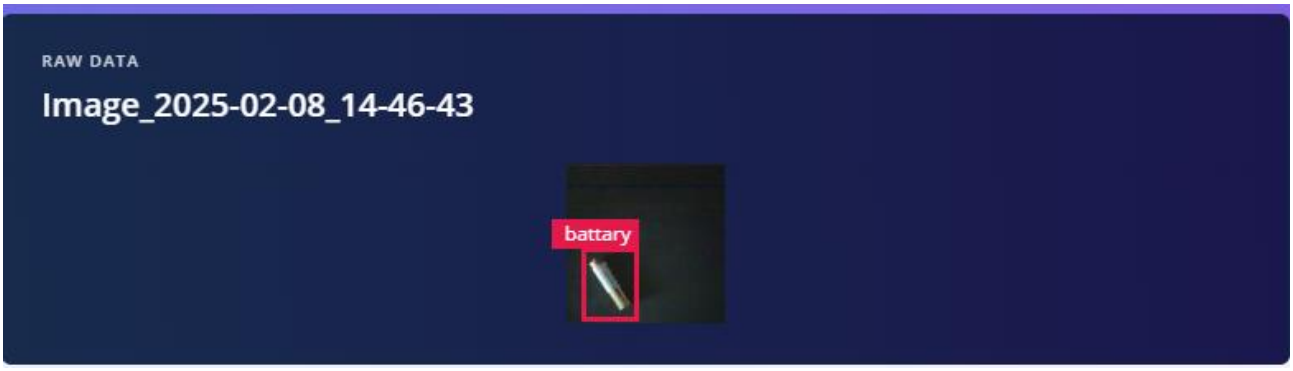


Рис. 3.15. Анотація зображення з встановленням обмежувальних рамок об’єкта.

Тренування моделі

На основі підготовлених даних модель проходить етап тренування. Edge Impulse автоматично виконує обробку зображень, витягує ознаки, тренує нейромережу та оптимізує її параметри. Після завершення користувач отримує базові метрики точності та ефективності, як зображено на рис. 3.16.

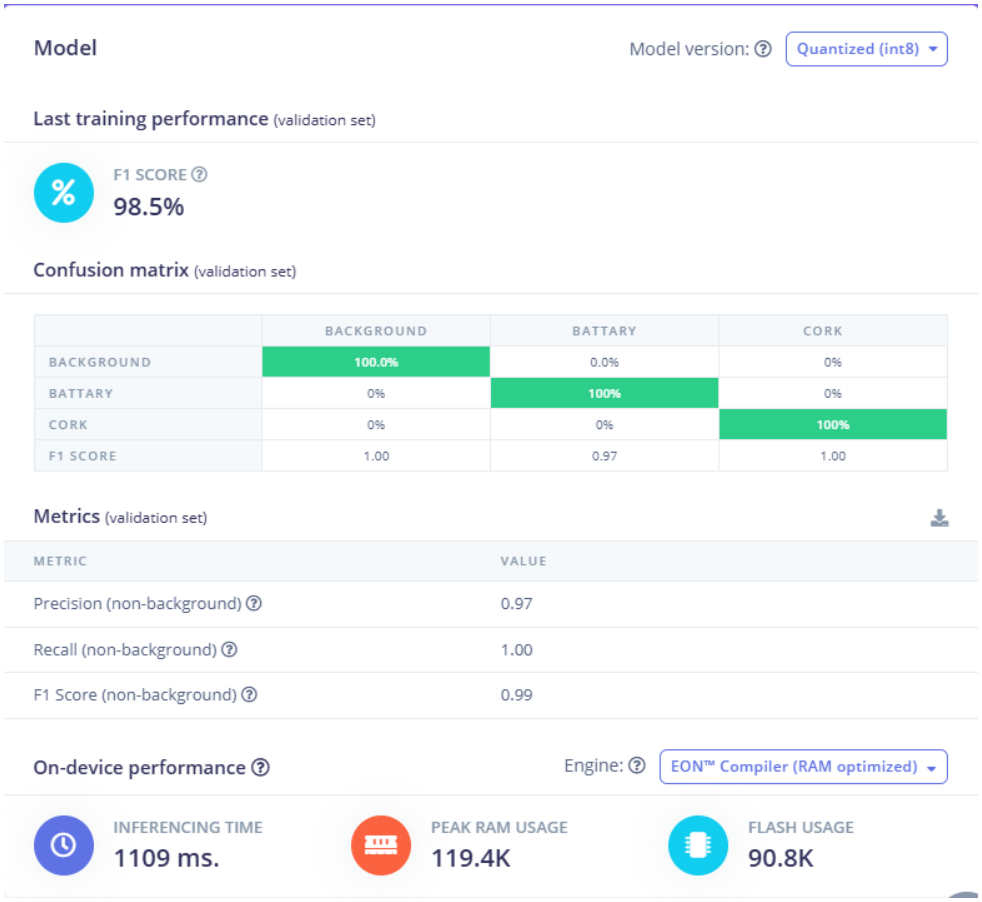


Рис. 3.16. Продуктивність згенерованої моделі.

Тестування

Тестування виконується як на окремих зображеннях, так і на валідаційній вибірці. Це дозволяє оцінити реальну якість моделі, виявити слабкі місця та, за потреби, повторити тренування з кращими параметрами або більшим обсягом даних. Після тестування вибірки можна отримати графік класифікації (рис. 3.17.), по якому видно успішність роботи моделі.

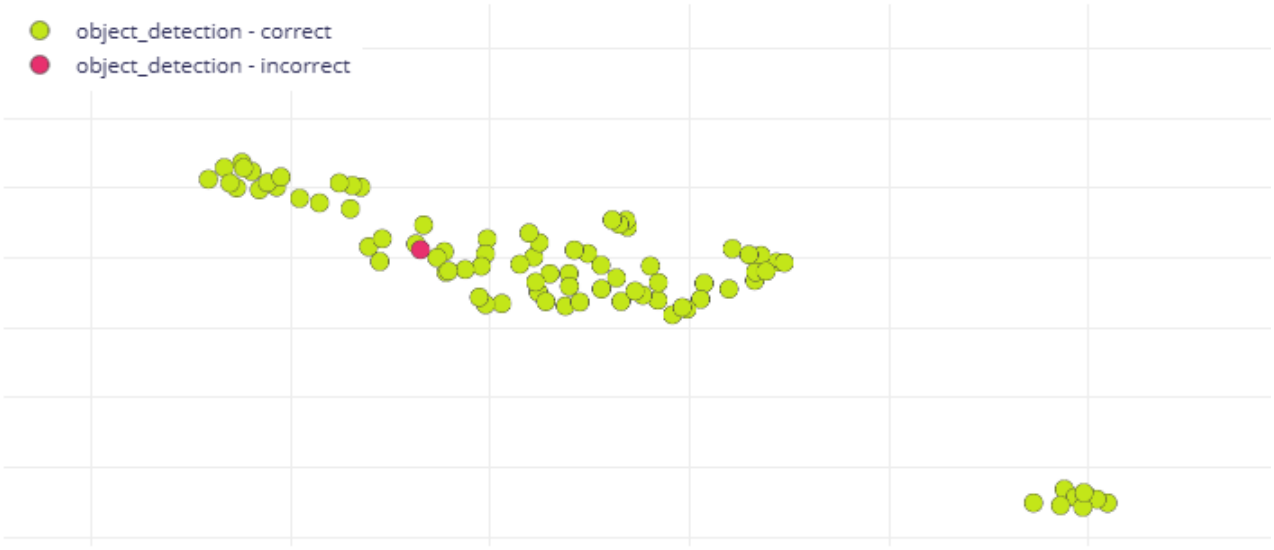


Рис. 3.17. Графік класифікації тестової вибірки.

3.4.2. Інтеграція моделі у власну систему

Експорт моделі у вигляді бібліотеки C++

Edge Impulse дозволяє експортувати модель як C++ бібліотеку, яка не потребує зовнішніх залежностей. Проте для того, щоб використовувати цю бібліотеку в Unity-проєкті, потрібно зробити інтерфейсний клас, який буде викликати функції розпізнавання з створеної моделі, а Unity-скрипти будуть звертатись саме до інтерфейсу.

Для цього було створено власну структуру даних, яка містить інформацію про обмежувальні прямокутники – результат виконання класифікації, які повертаються під час виклику функції розпізнавання. Усі функції, з якими повинне взаємодіяти середовище Unity, повинні бути позначені як такі, що здатні до експорту (див. Додаток А, файл EILibraryLoader.cs).

Після того, як був створений новий клас, потрібно зібрати актуальну бібліотеку, яка буде включати новий файл. Для цієї задачі був використаний CMake.

CMake — це кросплатформений інструмент для автоматизації процесу складання (build) програмного забезпечення. Його основна мета — спростити створення файлів складання для різних операційних систем і середовищ розробки.

Взаємодія Unity з бібліотекою

Для того, щоб можна було викликати функції з динамічної бібліотеки, в Unity потрібно створити скрипт, який їх буде імпортувати. Принцип імпорту функцій з C++ бібліотеки в C# базується на механізмі P/Invoke (Platform Invocation Services). Він дозволяє C# викликати функції, які реалізовані в нативній DLL-бібліотеці, написаній на C або C++. Для цього у C# використовують атрибут DllImport, що вказує ім’я DLL та підпис функції. Наприклад:

[DllImport("liblib")]

private static extern CustomResult RunClassifierResult(byte[] buffer);

Важливо дотримуватись правильної відповідності типів, в даному випадку потрібно було створити ідентичну структуру даних в C# тій, яка була створена в C++. На рис. 3.18. проілюстровано створену структуру.

```
// Define the C# equivalent of the C++ struct
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
4 references
public struct EiImpulseResultBoundingBox
{
    [MarshalAs(UnmanagedType.LPStr)]
    1 reference
    public string label;
    0 references
    public uint x;
    0 references
    public uint y;
    0 references
    public uint width;
    0 references
    public uint height;
    1 reference
    public float value;
}
```

Рис. 3.18. Структура даних, яка містить необхідні дані для опису обмежувальної рамки.

Тепер, коли інтегровано бібліотеку з моделлю розпізнавання образів, можна передавати дані з камери в функції розпізнавання та отримувати результати, на основі яких обраховувати положення робота.

3.5. Керування роботом

3.5.1. Визначення позиції об’єктів у світових координатах

Після обробки зображення моделлю розпізнавання образів, яка розгорнута за допомогою Edge Impulse, виводиться список виявлених об'єктів, кожен із яких описується обмежувальною рамкою (bounding box). Ця рамка задається координатами нижнього лівого кута (у пікселях) та розмірами по ширині й висоті.

Для визначення фактичного положення об'єкта у просторі виконується декілька послідовних етапів:

1. **Обчислення центру об'єкта.**

Центр рамки визначається як середина за шириною та висотою рамки, відносно її нижнього лівого кута:

$$x_{center} = x + \frac{width}{2}, \tag{3.1}$$

$$y_{center} = y + \frac{height}{2}, \tag{3.2}$$

де *x, y* – координати лівого нижнього кута, а *width* та *height* – ширина та висота зображення в пікселях відповідно.

2. **Перетворення піксельних координат у локальні метричні координати**.

Щоб визначити положення об'єкта у фізичному просторі (наприклад, у сантиметрах), застосовується масштабування на основі попередньо виміряного реального поля огляду камери. Наприклад, якщо ширина зображення становить 96 пікселів, а реальна ширина поля зору камери — 20 см, то кожен піксель відповідає ≈0.208 см:

$$x_{local} = \frac{x_{center}}{image\_width} \times real\_width, \tag{3.3}$$

де *image_width* – ширина зображення в пікселях, *real_width* – ширина поля зору камери в метричній системі.

Координати визначаються в локальній системі відліку, де початок (0, 0) розміщується у нижньому лівому куті зображення.

3. **Перетворення в світову систему координат.**

Щоб узгодити координати з реальною сценою, локальна система координат зміщується та обертається відповідно до положення камери на роботі. Оскільки положення камери при розпізнаванні об’єктів фіксоване, це дає змогу легко застосувати афінне перетворення (наприклад, зміщення по осі X і Z, якщо використовується горизонтальна площина):

$$x_{world} = x_{local} + offset_x, \quad (3.4)$$

$$z_{world} = z_{local} + offset_z, \quad (3.5)$$

де *offset* – зміщення від початку світових координат до початку координат зображення.

Таким чином, центр розпізнаного об’єкта у зображенні можна однозначно перевести у світові координати сцени, якими користується робот для планування руху та взаємодії з навколишнім середовищем.

3.5.2. Обрахунок інверсної кінематики.

Інверсна кінематика (Inverse Kinematics, IK) — це метод у робототехніці та комп’ютерній графіці, який дозволяє розрахувати необхідні кути обертання суглобів багатоланкової системи (наприклад, роботизованого маніпулятора), щоб кінцевий ефектор (рука, інструмент, захоплювач) досяг певного положення в просторі. Це обернена задача до прямої кінематики, де відомі кути суглобів, а необхідно знайти положення кінцевого ефектора.

У класичній інверсній кінематиці задача формулюється так:

Дано положення та орієнтацію кінцевої точки. Знайти такі значення кутів суглобів, які дозволяють маніпулятору досягнути цієї точки.

Ця задача може мати:

- одне рішення;
- багато рішень;
- або взагалі не мати розв’язку (якщо ціль знаходиться поза досяжною зоною).

Методи розв’язання бувають:

- аналітичні — точні формули для простих конфігурацій;

- числові — наближені методи, які застосовуються до складних моделей (наприклад, метод Ньютона-Рафсона або градієнтний спуск).

Реалізація інверсної кінематики у проєкті

В рамках даної роботи використовується аналітичне рішення для двохвимірного випадку. Робот має три основні відрізки (кінцівки). Беручи до уваги конструкцію механізму захвату, найкращим варіантом орієнтації кінцевої кісти є саме перпендикулярно до поверхні, на якій розташовані об’єкти. Ця деталь вносить зміни в постановку задачі, тому що тепер цільовою точкою є не позиція об’єкта, до якого потрібно добратись, а суглоб між другою та третьою кістками.

Остання кінцівка умовно розділена на дві частини. Все тому, що основа робота та об’єкт для захвату лежать на одній площині, проте початок першої кістки зміщений по осі Y(по висоті) догори. Для зручності зміщуємо початок координат в початок першої кістки. Після цього визначається, яка частина третьої кісти знаходиться під початком координат, а яка над. Оскільки зміщення по висоті відоме, і воно завжди однакове, то знайти частину, яка лежить над початком координатами не складно, потрібно відняти зміщення від довжини третьої кістки.

$$h1 = c - h2, \tag{3.6}$$

де *h1* – висота над початком координат, *c* – довжина третьої кістки, *h2* – відоме зміщення.

Наступним етапом потрібно знайти гіпотенузу g, трикутника, який утворюється основною віссю та відрізком *h1*. Вона знадобиться в подальшому для обчислення кутів між усіма кістками.

$$g^2 = d^2 + h_1^2, \tag{3.7}$$

$$g = \sqrt{d^2 + h_1^2}, \tag{3.8}$$

де d – відрізок між початком координат та точкою поділу останньої кістки.

Для того, щоб правильно позиціонувати робота, потрібно знайти кути α, β та γ. Можна використати теорему косинусів:

$$\alpha_0 = \arccos(\frac{a^2+g^2-b^2}{2ag}), \quad (3.9)$$

$$\beta = \arccos(\frac{a^2+b^2-g^2}{2ab}), \quad (3.10)$$

Оскільки сума кутів трикутника дорівнює 180°, то

$$\gamma_0 = 180 - (\alpha_0 - \beta) \quad (3.11)$$

Синус кута в прямокутному трикутнику визначається як протилежний катет поділений на гіпотенузу, звідки знайдемо кути α_1 та γ_1:

$$\alpha_1 = \arcsin(\frac{h1}{g}), \quad (3.12)$$

$$\gamma_1 = \arcsin(\frac{d}{g}) \quad (3.13)$$

Тепер можемо знайти повні кути α та γ:

$$\alpha = \alpha_0 + \alpha_1, \quad (3.14)$$

$$\gamma = \gamma_0 + \gamma_1 \quad (3.15)$$

На рис. 3.19. графічно зображено задачу обрахунку положення кісток робота.

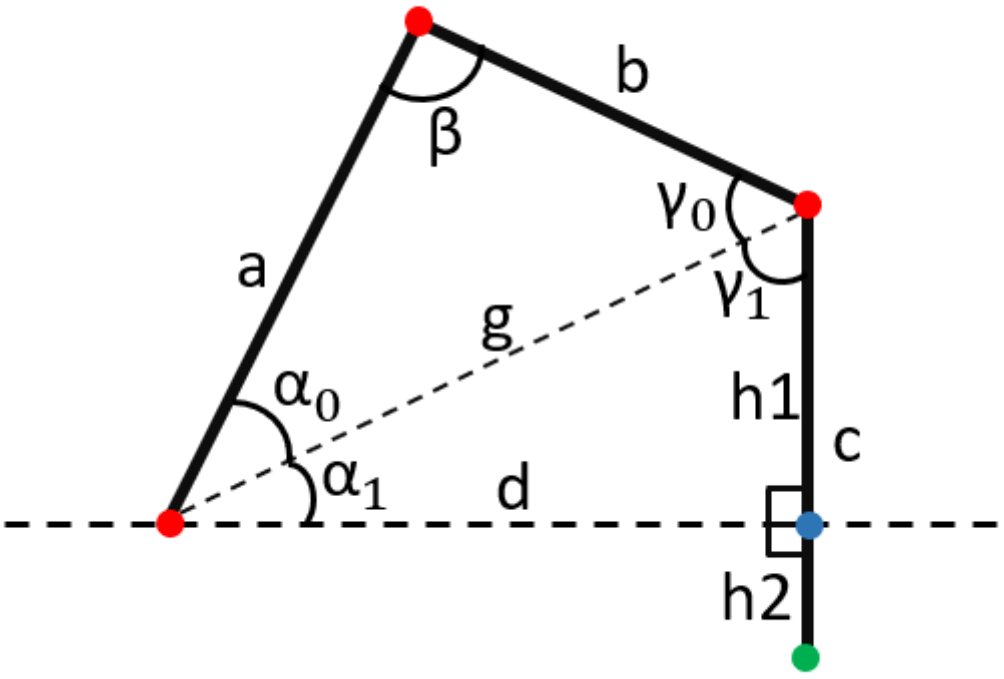


Рис. 3.19. Графічне зображення задачі інверсної кінематики.

Можливі альтернативи

Для складніших систем із більшою кількістю ступенів свободи, або для тривимірних випадків, де аналітичне розв’язання ускладнене або неможливе, застосовуються інші методи:

- Метод Джейкобіана— обчислення похідних функції положення кінцевого ефектора відносно кутів суглобів.
- FABRIK (Forward And Backward Reaching Inverse Kinematics) — ітераційний геометричний підхід для досягнення цілі.
- IK Solvers у Unity — Unity містить готові рішення для задачі інверсної кінематики.

3.6. Взаємодія з Arduino.

Для керування рухами маніпулятора на апаратному рівні використовується плата Arduino UNO, до якої під’єднано п’ять сервоприводів. Передача команд на Arduino здійснюється з середовища Unity через послідовний порт (Serial Port), що дозволяє передавати дані з комп’ютера на мікроконтролер у вигляді текстових команд.

Система побудована за принципом «Unity → Arduino → Сервоприводи». У Unity обчислюються кути повороту всіх сегментів маніпулятора за допомогою інверсної кінематики, після чого ці кути передаються Arduino, яка вже безпосередньо керує сервоприводами за допомогою PWM-сигналів.

Передача даних з Unity

У Unity реалізовано клас, який відкриває вказаний COM-порт, обчислює кути повороту та надсилає їх на Arduino. Відправка здійснюється в текстовому форматі команд типу:

ANGLES,45.20,123.55,76.32,160.00,150.00

Така команда складається з ключового слова (ANGLES) і п’яти десяткових значень, розділених комами. Ці значення відповідають кутам повороту усіх сервоприводів. На рис. 3.20. зображено функцію відправки даних з Unity на Arduino через послідовний порт.

```
3 references
private void SendData(string command, params float[] values)
{
    if(!port.IsOpen)
        return;

    Debug.Log("Sending data to Arduino");
    string message = command;
    foreach (float val in values)
    {
        message += "," + val.ToString("F2", System.Globalization.CultureInfo.InvariantCulture); // Формуємо рядок
    }

    Debug.Log(message);

    port.Write(message);
}
```

Рис. 3.20. Функція відправки даних про кути через послідовний порт.

Обробка даних на Arduino

На Arduino приймається текстовий рядок, який розділяється на компоненти за допомогою символів коми. Витягнуті значення перетворюються у float, обмежуються в межах 0–180° (допустимий діапазон для більшості сервоприводів) і зберігаються як цільові позиції.

Далі кожен сервопривід повільно обертається до нової цілі, інкрементуючи або декрементуючи свій поточний кут, щоб забезпечити плавний рух.

Сервоприводи не обертаються довільно — вони керуються PWM-сигналами (широтно-імпульсна модуляція). Arduino генерує PWM-сигнал, який задає позицію серво:

- PWM-сигнал з частотою 50 Гц (період 20 мс),
- Ширина імпульсу визначає кут:
  - 1 мс ≈ 0°
  - 1.5 мс ≈ 90°
  - 2 мс ≈ 180°

Щоб уникнути різких рухів, що можуть пошкодити маніпулятор або об’єкти, введено затримку між кроками. Затримка та крок обертання (±1 градус за цикл) створюють ілюзію плавного руху.

Також передбачена можливість калібрування кожного серво за допомогою змінних *firstAngleOffset*, *secondAngleOffset* тощо — для врахування механічного зсуву або неправильного монтажу.

3.7. Збірка та тестування системи

Оскільки однією з вимог конструкції була модульність, останнім етапом роботи є саме збірка макету.

Перш за все, потрібно перевірити положення сервоприводів та впевнетись, що вони знаходяться в своїх початкових позиціях, і лише потім кріпити до них інші деталі. Якщо сервоприводи будуть не в правильному положенні, або не буде враховано їхню орієнтацію, це може призвести до неправильної роботи робота, виходу обертань сервоприводів за межі визначених значень, і в наслідок цього до механічних пошкоджень. В деяких випадках, початкове положення сервопривода повинно бути не 0°, а 180°, тобто він має повертатись в протилежному напрямку.

Для того, щоб зібрати макет, потрібно з’єднювати деталі в певній послідовності. Основна частина робота має симетричну конструкцію, як видно на рис. 3.21, яку потрібно починати скріплювати з нижньої лівої сторони, йдучи до гори, після чого перейти до правої частини, починаючи згори та йдучи до низу.

Оскільки для робота не було створено власної плати, для підключення сервоприводів до Arduino використовувалась макетна плата(рис. 3.22), на яку подається живлення через USB-кабель. Через те, що плата Arduino та решта системи мають різне джерело живлення, їх потрібно під’єднати до спільної землі для коректної роботи. Як видно на рис. 3.22, земля мікроконтроллера під’єднана до землі макетної плати зеленим кабелем. В Додатку Б, Аркуш 3, зображено електричну принципову схему.

Рис. 3.21. Зібраний макет робота.

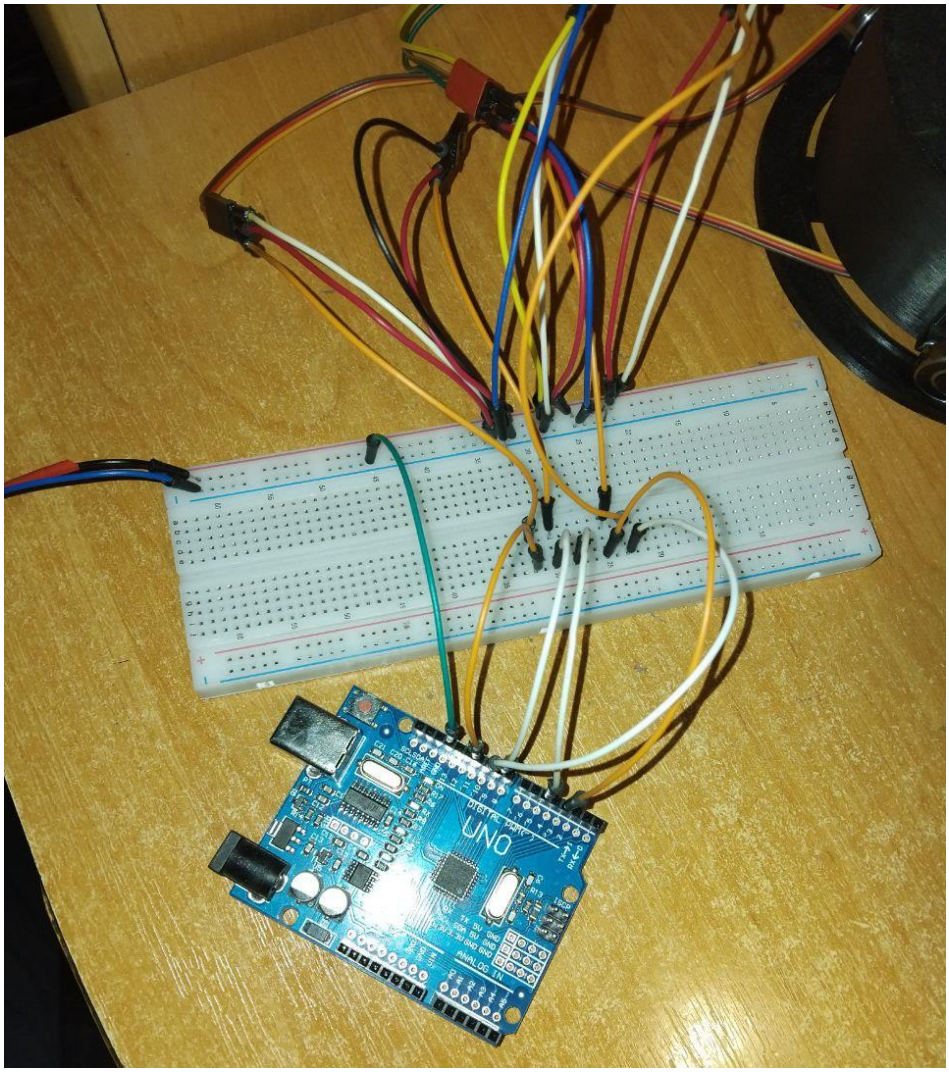


Рис. 3.22. Підключення сервоприводів до мікроконтроллера.

У результаті було протестовано зібрану систему на правильність виконання. Усі сервоприводи працюють коректно та виконують команди, передані через Unity. Механізм захвату виконує свою функцію та працює без дефектів, виконуючи поставлені задачі.

Висновки до розділу 3

У цьому розділі було повністю реалізовано апаратно-програмну систему керування настільним роботом-маніпулятором з комп’ютерним зором. Розроблено модульну архітектуру, яка об’єднує робота, камеру ESP32-CAM, WebSocket-сервер, Unity-середовище для візуалізації й керування, а також мікроконтролер Arduino для безпосереднього управління сервоприводами. Сконструйовано фізичний макет маніпулятора з чотирма ступенями свободи, з урахуванням вимог

до компактності, модульності та сумісності з 3D-друком. Усі деталі змодельовано в AutoCAD, після чого виконано цифрову збірку для перевірки точності перед експортом у середовище Unity.

Було реалізовано систему передачі відеопотоку з ESP32-CAM до Unity через WebSocket-з'єднання, що забезпечило стабільну трансляцію зображення в реальному часі. На базі отриманих кадрів у середовищі Unity реалізовано інструмент для збирання датасету, який був використаний для навчання моделі розпізнавання об'єктів у сервісі Edge Impulse. Отриману модель експортовано у форматі C++ бібліотеки, інтегровано в Unity через інтерфейс P/Invoke та адаптовано для використання у власному скриптовому середовищі.

Виконано повний цикл обробки даних: розпізнавання об'єктів, обрахунок їх координат у світовій системі, розв'язання задачі інверсної кінематики з використанням аналітичного методу, та генерація цільових кутів для кожного суглоба маніпулятора. Кути передаються з Unity на Arduino через послідовний порт, де вони обробляються і перетворюються на PWM-сигнали для керування сервоприводами. Забезпечено плавне переміщення механізмів без ривків і перевантаження.

На завершення система була зібрана, протестована та успішно виконала поставлені завдання. Усі її компоненти працюють узгоджено: відеопотік передається і обробляється, об'єкти розпізнаються, рухи маніпулятора відповідають розрахованим траєкторіям, а механізм захвату виконує функції утримання. Результати свідчать про повну працездатність системи та її готовність до подальшого розширення або адаптації під нові задачі.

ВИСНОВКИ

У межах виконаної роботи було реалізовано повноцінну апаратно-програмну систему керування настільним роботом-маніпулятором з використанням комп’ютерного зору. Система розроблена як гнучкий модульний комплекс, що забезпечує повний цикл функціонування: від візуального сприйняття середовища до фізичного виконання дій. Особливістю реалізації є інтеграція різнорідних компонентів — вбудованої камери, програмного середовища візуалізації, модуля розпізнавання об'єктів, алгоритмів кінематичного розрахунку та фізичного виконавчого пристрою — в єдину взаємодіючу структуру.

Під час виконання проєкту було не лише технічно з'єднано окремі елементи, але й реалізовано функціональний алгоритм, що дозволяє системі діяти автономно: отримувати відеопотік, виявляти цільові об'єкти, визначати їх положення у просторі, обчислювати необхідні параметри руху та передавати керуючі сигнали на виконавчі механізми. Всі етапи обробки виконуються локально, без залучення зовнішніх хмарних сервісів, що дозволяє використовувати систему незалежно від інтернет-з’єднання, у реальному часі та з низькою затримкою.

Ключовим досягненням є розробка узгодженої архітектури, яка дозволяє ефективно інтегрувати засоби комп’ютерного зору в систему керування маніпулятором. Такий підхід демонструє приклад створення доступної навчальної або дослідницької платформи, яка легко відтворюється, модифікується та масштабовується. Завдяки застосуванню відкритих апаратних і програмних рішень, розробка не залежить від ліцензійних обмежень і може бути адаптована до різних рівнів складності та різних навчальних програм.

Розроблена система має широкі перспективи для подальшого використання й розвитку. У навчальному контексті вона може слугувати основою для лабораторних робіт з курсів робототехніки, мехатроніки, систем керування або комп’ютерного зору. У дослідницькій сфері її можна використовувати як базу для вивчення алгоритмів обробки зображень, розпізнавання об’єктів, адаптивного керування або тестування нових підходів до інверсної кінематики. Крім того,

система придатна для використання як прототип у хобі-проєктах або під час створення макетів майбутніх повнофункціональних промислових рішень.

Завдяки модульності, систему можна розширити новими функціональними блоками: наприклад, додати захват з датчиком сили, змінити модель розпізнавання, перейти до 3D-кінематики, реалізувати контроль зворотного зв’язку або навчання за допомогою підкріплення. Це відкриває можливості для глибшого дослідження взаємодії між штучним інтелектом і фізичним світом.

Таким чином, у ході виконання роботи було виконано не лише технічне завдання створення працездатної системи, але й закладено основу для її подальшого розвитку як універсального дослідницько-навчального інструмента. Отримані результати свідчать про доцільність поєднання сучасних підходів до візуального сприйняття, алгоритмічного керування й апаратної реалізації в єдиній інтегрованій системі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Arduino. (n.d.). *Arduino UNO Rev3*. Retrieved June 4, 2025, from https://www.arduino.cc/en/Main/ArduinoBoardUno
2. Craig, J. J. (2017). *Introduction to robotics: Mechanics and control* (4th ed.). Pearson.
3. Bishop, C. M. (2006). *Pattern recognition and machine learning* [E-book]. Springer. Retrieved June 4, 2025, from https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf
4. Edge Impulse. (n.d.). *Edge Impulse Documentation*. Retrieved June 4, 2025, from https://docs.edgeimpulse.com/
5. Unity Technologies. (n.d.). *Unity Documentation*. Retrieved June 4, 2025, from https://docs.unity3d.com/
6. Santos, R. (n.d.). *ESP32-CAM Video Streaming Web Server (works with Home Assistant)*. Random Nerd Tutorials. Retrieved June 4, 2025, from https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-assistant/
7. Mozilla Developer Network. (n.d.). *WebSockets API*. Retrieved June 4, 2025, from https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
8. Dreyer, E. (n.d.). *NativeWebSocket: WebSocket client implementation for Unity* [GitHub repository]. Retrieved June 4, 2025, from https://github.com/endel/NativeWebSocket
9. MathWorks. (n.d.). *Inverse kinematics – Robotics System Toolbox*. Retrieved June 4, 2025, from https://www.mathworks.com/help/robotics/ref/inversekinematics-system-object.html
10. Tinkercad. (n.d.). *Tinkercad Circuits*. Retrieved June 4, 2025, from https://www.tinkercad.com/learn/circuits

11. Autodesk. (n.d.). *Fusion Quick Start Guide*. Retrieved June 4, 2025, from https://www.autodesk.com/learn/ondemand/curated/fusion-quick-start-guide
12. Autodesk. (n.d.). *AutoCAD Quick Start Guide*. Retrieved June 4, 2025, from https://www.autodesk.com/learn/ondemand/curated/autocad-quick-start-guide
13. TowerPro. (n.d.). *TowerPro MG995 Servo – Datasheet*. Retrieved June 4, 2025, from https://www.servodatabase.com/servo/towerpro/mg995
14. TowerPro. (n.d.). *TowerPro SG90 Micro Servo – Datasheet*. Retrieved June 4, 2025, from https://servodatabase.com/servo/towerpro/sg90
15. Hansson, D., & Torstensson, W. (2024). *A low-cost robotic arm using machine learning and image classification* [Bachelor's thesis, Malmö University]. Retrieved June 4, 2025, from https://mau.diva-portal.org/smash/get/diva2:1915913/FULLTEXT02.pdf

ДОДАТОК А

Фрагмент коду системи

Файл ArmControll.ino

```
#include <Servo.h>
Servo servos[4];
int currentPos[4] = {90, 160, 0, 180};     // Поточна позиція кожного серво
int targetPos[4]  = {90, 160-6, 0+10, 180-20}; // Цільова позиція для кожного серво
int delayTime = 20;                 // Час затримки між кроками (чим більший — тим повільніше)

float firstAngleOffset = 0;
float secondAngleOffset = 0;
float thirdAngleOffset = 0;
float fourthAngleOffset = 0;

void setup() {
  Serial.begin(9600);

  servos[0].attach(9);
  servos[1].attach(10);
  servos[2].attach(5);
  servos[3].attach(3);
}

void loop() {
  if (Serial.available()) {
    String data = Serial.readStringUntil('\n'); // Читаємо рядок
    int firstComma = data.indexOf(',');
    int secondComma = data.indexOf(',', firstComma + 1);
    int thirdComma = data.indexOf(',', secondComma + 1);
    int fourthComma = data.indexOf(',', thirdComma + 1);
    String command = data.substring(0, firstComma); // Виділяємо команду

    if (command == "ANGLES") {
      float angle1 = data.substring(firstComma + 1, secondComma).toFloat();
      float angle2 = data.substring(secondComma + 1, thirdComma).toFloat();
      float angle3 = data.substring(thirdComma + 1, fourthComma).toFloat();
```

```
        float angle4 = data.substring(fourthComma + 1).toFloat();

        angle1+= firstAngleOffset;
        angle2+=secondAngleOffset;
        angle3 += thirdAngleOffset;
        angle4 += fourthAngleOffset;

        angle1 = constrain(angle1, 0, 180);
        angle2 = constrain(angle2, 0, 180);

        targetPos[0] = angle1;
        targetPos[1] = angle2;
        targetPos[2] = angle3;
        targetPos[3] = angle4;
    }
  }

  bool allReached = false;

  // Поки не всі серво досягли своїх цільових позицій
  while (!allReached) {
   allReached = true;

   for (int i = 0; i < 4; i++) {
    if (currentPos[i] < targetPos[i]) {
     currentPos[i]++;
     servos[i].write(currentPos[i]);
     allReached = false;
    } else if (currentPos[i] > targetPos[i]) {
     currentPos[i]--;
     servos[i].write(currentPos[i]);
     allReached = false;
    }
   }

   delay(delayTime);
  }

  // Пауза перед новим циклом (можеш тут змінювати targetPos)
  delay(200);
}
```

Файл EILibraryLoader.cs

```
public class TestLibraryLoader : MonoBehaviour
{
   [SerializeField] private ArmController armController;

   [DllImport("liblib")]
   private static extern CustomResult RunClassifierResult(byte[] buffer);

   [DllImport("liblib")]
   private static extern CustomResult RunClassifierWithStaticData();

    // Define the C# equivalent of the C++ struct
   [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi)]
   public struct EiImpulseResultBoundingBox
   {
      [MarshalAs(UnmanagedType.LPStr)]
      public string label;
      public uint x;
      public uint y;
      public uint width;
      public uint height;
      public float value;
   }

   [StructLayout(LayoutKind.Sequential)]
   public struct CustomResult
   {
      public IntPtr bounding_boxes; // Pointer to array of bounding boxes
      public uint bounding_boxes_count;

   }

   [SerializeField ] Texture2D testImg;



   private byte[] buffer;

   [SerializeField] private CamToTexture camToTexture;
   [SerializeField] private BoundingBoxes boundingBoxes;


   private void Update()
   {
```

```
            if(Input.GetKeyDown(KeyCode.F))
            {
                buffer = camToTexture.buffer;

                if(buffer!=null)
                {
                    CustomResult result =  RunClassifierResult(buffer);
                   Debug.Log($"Bounding boxes count: {result.bounding_boxes_count}");

                    if (result.bounding_boxes != IntPtr.Zero)
                    {
                                EiImpulseResultBoundingBox[] eiBoundingBoxes = new
EiImpulseResultBoundingBox[result.bounding_boxes_count];

                        for (int i = 0; i < result.bounding_boxes_count; i++)
                        {
                            IntPtr bboxPtr = (IntPtr)((long)result.bounding_boxes + i *
Marshal.SizeOf(typeof(EiImpulseResultBoundingBox)));

                                                                eiBoundingBoxes[i] =
Marshal.PtrToStructure<EiImpulseResultBoundingBox>(bboxPtr);
                        }

                        foreach (var bbox in eiBoundingBoxes)
                        {
                            Debug.Log($"Label: {bbox.label}, Value: {bbox.value}");
                        }

                        boundingBoxes.SetBoundingBoxes(eiBoundingBoxes);
                        armController.ProcessEIResult(eiBoundingBoxes);
                    }
                }
            }
        }

        public float[] DoubleArrayToFloat(double[] array)
        {
            float[] newArr = new float[array.Length];
            for(int i = 0; i<array.Length; i++)
            {
                newArr[i] = (float)array[i];
            }
            return newArr;
        }
    }
```

Файл CamToTexture.cs

```
public class CamToTexture : MonoBehaviour
{
   [SerializeField] private RawImage rawImage;
   [SerializeField] private Config config;

   private NativeWebSocket.WebSocket _webSocket;

   public Texture2D tex;

   [HideInInspector] public byte[] buffer;

   async void Start()
   {
                                                _webSocket = new
NativeWebSocket.WebSocket("ws://192.168.0.113:8888");
      _webSocket.OnOpen += () => { print("Connection Open!"); };
      _webSocket.OnError += (e) => { print("Error :" + e); };
      _webSocket.OnClose += (e) => { print("Connection Close!"); };
      _webSocket.OnMessage += (bytes) =>
      {
         if (bytes.Length > 0)
         {
            if(config.ImgFormat == Config.ImageFormat.JPEG)
            {
               tex = LoadJPEGTexture(bytes);
            }
            else
            {
               tex = LoadUncompressedTexture(bytes);
            }

            rawImage.texture = tex;
         }
      };
      await _webSocket.Connect();
   }

   void Update()
   {
      _webSocket.DispatchMessageQueue();

      if(Input.GetKeyDown(KeyCode.F))
      {
```

```
            SetCameraParameter("framesize", framesizeValue);
        }
    }

    public async void SetCameraParameter(string parameter, int value)
    {
        if (_webSocket.State == WebSocketState.Open)
        {
                                                    string message =
$"{{\"command\":\"set_parameter\",\"parameter\":\"{parameter}\",\"value\":{
value}}}";
            await _webSocket.SendText(message);
        }
    }

    private Texture2D LoadJPEGTexture(byte[] bytes)
    {
        Texture2D texture = new Texture2D(2, 2, TextureFormat.RGB24, false);
        texture.LoadImage(bytes);

        texture = CropAndResize(texture, config.ImageSize);
        buffer = ConvertToRGB888(texture);
        return texture;
    }

    private Texture2D LoadUncompressedTexture(byte[] bytes)
    {
       Texture2D texture = new Texture2D(config.ImageSize, config.ImageSize,
TextureFormat.RGB24, false);
        texture.LoadRawTextureData(bytes);
        texture.Apply();

        buffer = bytes;

        return texture;
    }

    public Texture2D CropAndResize(Texture2D sourceTexture, int newSize)
    {
        int size = Mathf.Min(sourceTexture.width, sourceTexture.height);

        int startX = (sourceTexture.width - size) / 2;
        int startY = (sourceTexture.height - size) / 2;
```

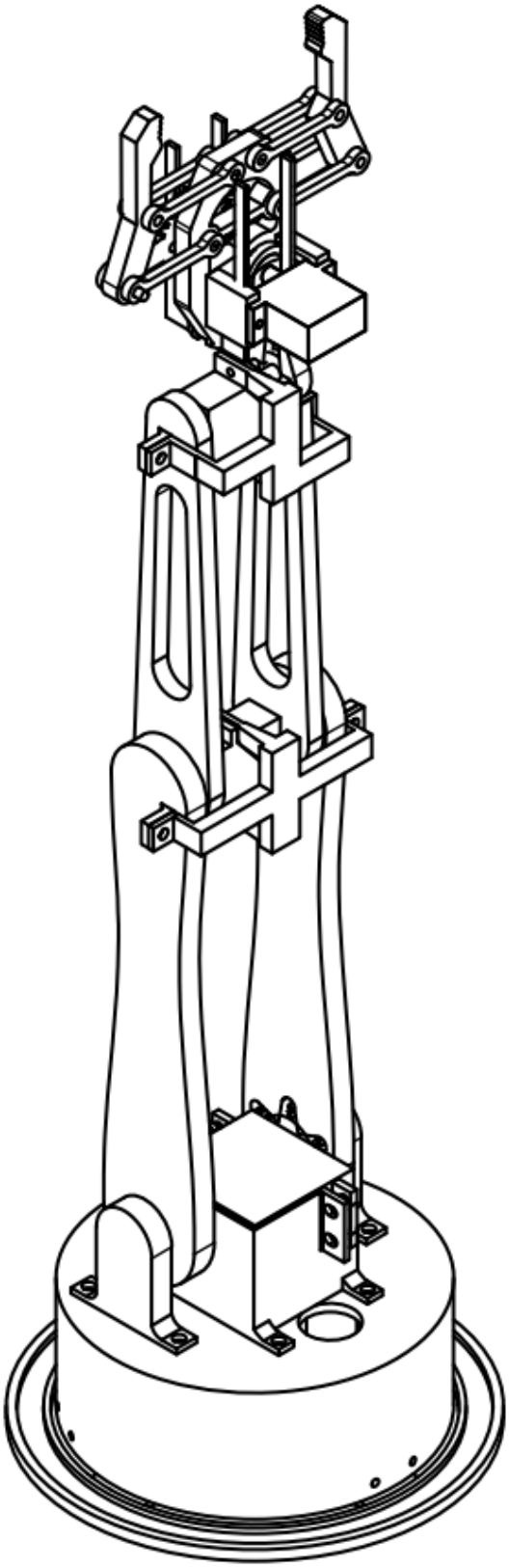
```
        Texture2D croppedTexture = CropTexture(sourceTexture, startX, startY, size, size);

        return ResizeTexture(croppedTexture, newSize, newSize);
    }

    private Texture2D CropTexture(Texture2D source, int startX, int startY, int width, int height)
    {
        Texture2D cropped = new Texture2D(width, height);
        Color[] pixels = source.GetPixels(startX, startY, width, height);
        cropped.SetPixels(pixels);
        cropped.Apply();
        return cropped;
    }

     private Texture2D ResizeTexture(Texture2D source, int newWidth, int newHeight)
    {
                RenderTexture rt = RenderTexture.GetTemporary(newWidth, newHeight);
        rt.filterMode = FilterMode.Bilinear;

        Graphics.Blit(source, rt);

          Texture2D resizedTexture = new Texture2D(newWidth, newHeight, source.format, false);
        RenderTexture.active = rt;
            resizedTexture.ReadPixels(new UnityEngine.Rect(0, 0, newWidth, newHeight), 0, 0);
        resizedTexture.Apply();

        RenderTexture.active = null;
        RenderTexture.ReleaseTemporary(rt);

        return resizedTexture;
    }

    public byte[] ConvertToRGB888(Texture2D texture)
    {
        Color32[] pixels = texture.GetPixels32();
        byte[] rgbData = new byte[pixels.Length * 3];

        for (int i = 0, j = 0; i < pixels.Length; i++)
        {
```

```
            rgbData[j++] = pixels[i].r; // Red
            rgbData[j++] = pixels[i].g; // Green
            rgbData[j++] = pixels[i].b; // Blue
        }

        return rgbData;
    }

    private async void OnApplicationQuit()
    {
        await _webSocket.Close();
    }
}
```

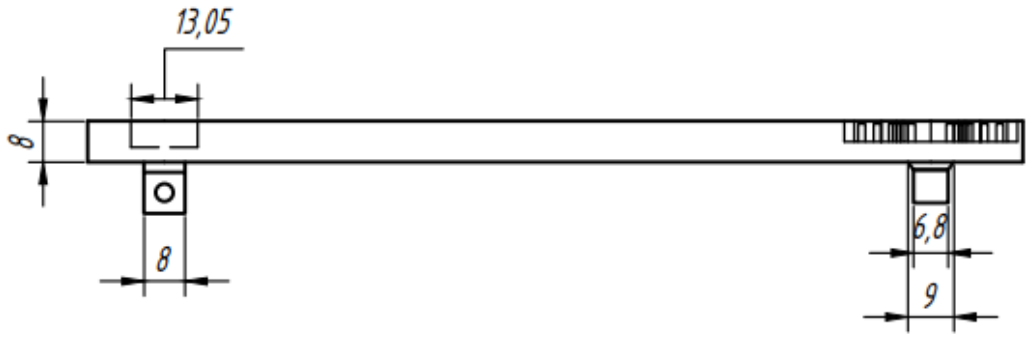
Файл Config.cs

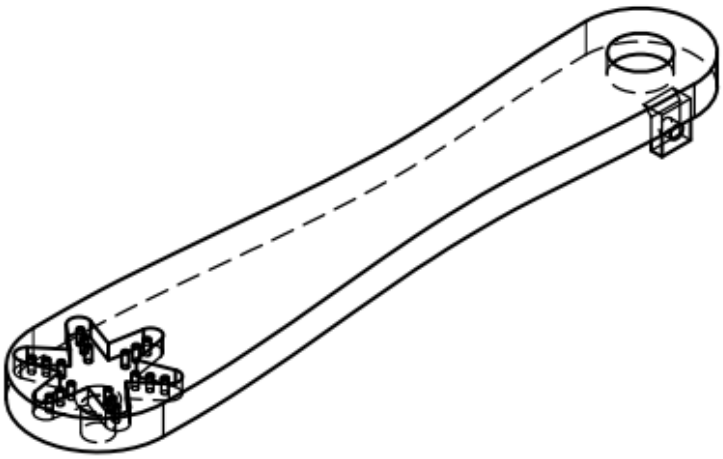
```
public class Config : ScriptableObject
{
    public enum ImageFormat
    {
        JPEG,
        Uncompressed
    }
    [Header("Camera settings")]
    public int ImageSize;
    public ImageFormat ImgFormat;
    [Tooltip("In millimeters")]
    public Vector2 CameraCaptureArea;
    [Range(0,1)]

    [Header("Classification settings")]
    public float ClassificationThreshold;

    [Header("Robot settings")]
    public float Bone1Length = 15.0f;
    public float Bone2Length = 13.0f;
    public float Bone3Length = 9.5f;
    public float HeightOffset = 8f;
    public Vector2 RobotPosition;
    public float BaseAngleOffset;
    public float Bone1AngleOffset;
    public float Bone2AngleOffset;
    public float Bone3AngleOffset;
}
```

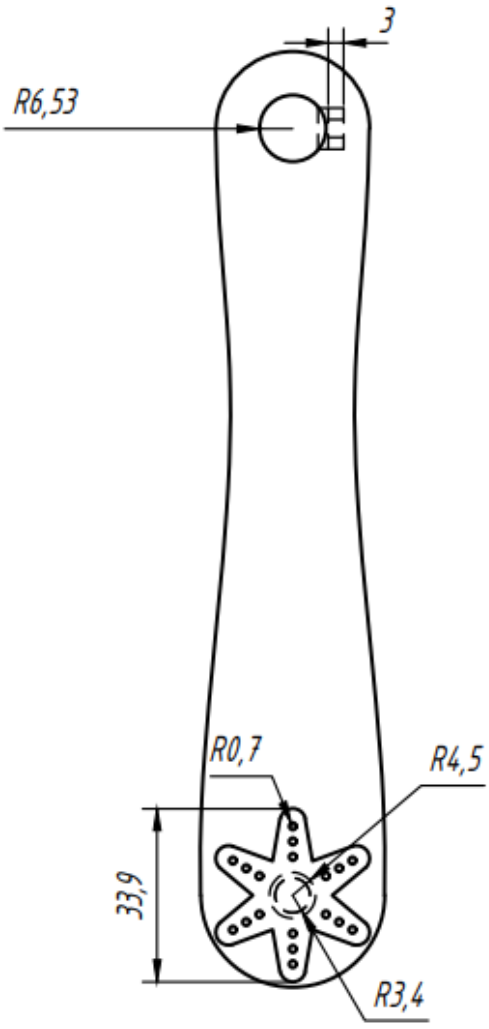
ДОДАТОК Б
Графічні матеріали



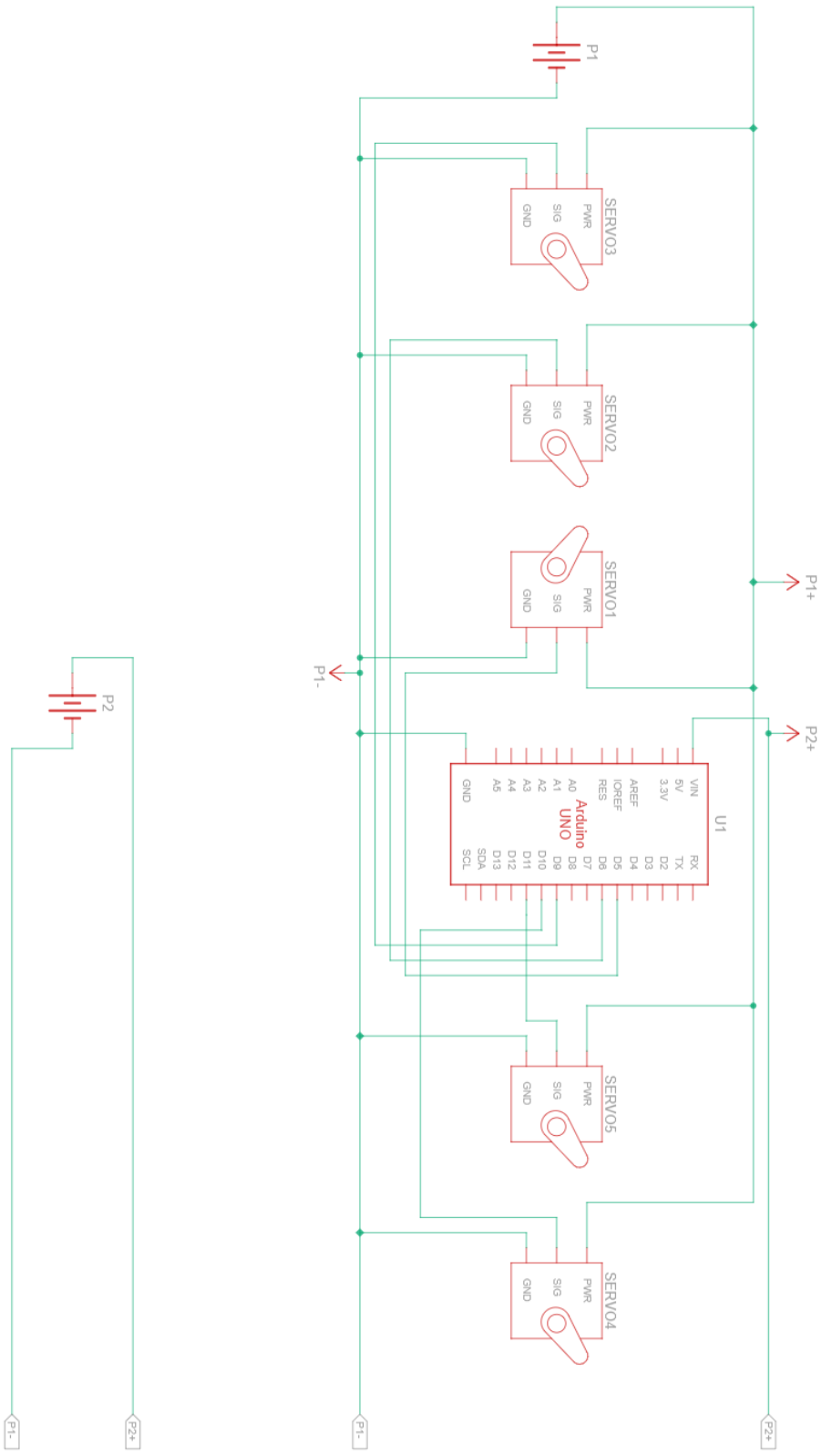
					Бакалаврська кваліфікаційна робота		
Змн.	Арк.	Прізвище	Підпис	Дата	Віртуальна збірка робота	Літера	Маса / Масштаб
Розроб.		Івасько О.І.				Н	1:2
Керівник		Колесник К.К.				Аркуш 1	Аркушів 6
Н. Контр.						НУ «ЛП», ІКНІ, каф. САП, гр. КН-412	
Зав. каф.		Лобур М.В.					







					Бакалаврська кваліфікаційна робота	
Змн.	Арк.	Прізвище	Підпис	Дата	Креслення деталі	Літера: Н; Маса: ; Масштаб: 1:2
Розроб.		Івасько О.І.				Аркуш 2; Аркушів 6
Керівник		Колесник К.К.				НУ «ЛП», ІКНІ, каф. САП, гр. КН-412
Н. Контр.						
Зав. каф.		Лобур М.В.				



Бакалаврська кваліфікаційна робота

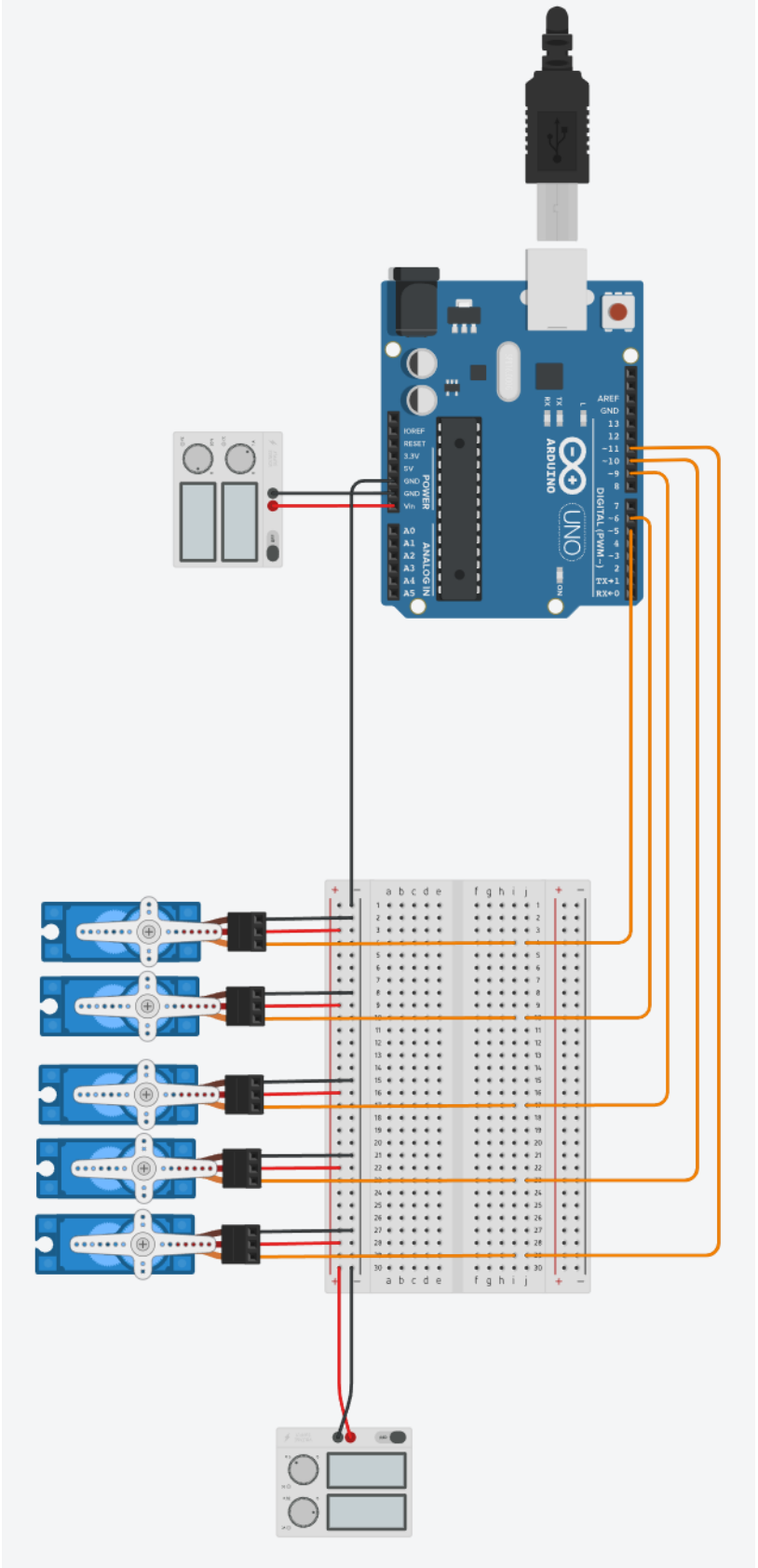
Змн.	Арк.	Прізвище	Підпис	Дата
Розроб.		Івасько О.І.		
Керівник		Колесник К.К.		
Н. Контр.				
Зав. каф.		Лобур М.В.		

Схема електрична принципова

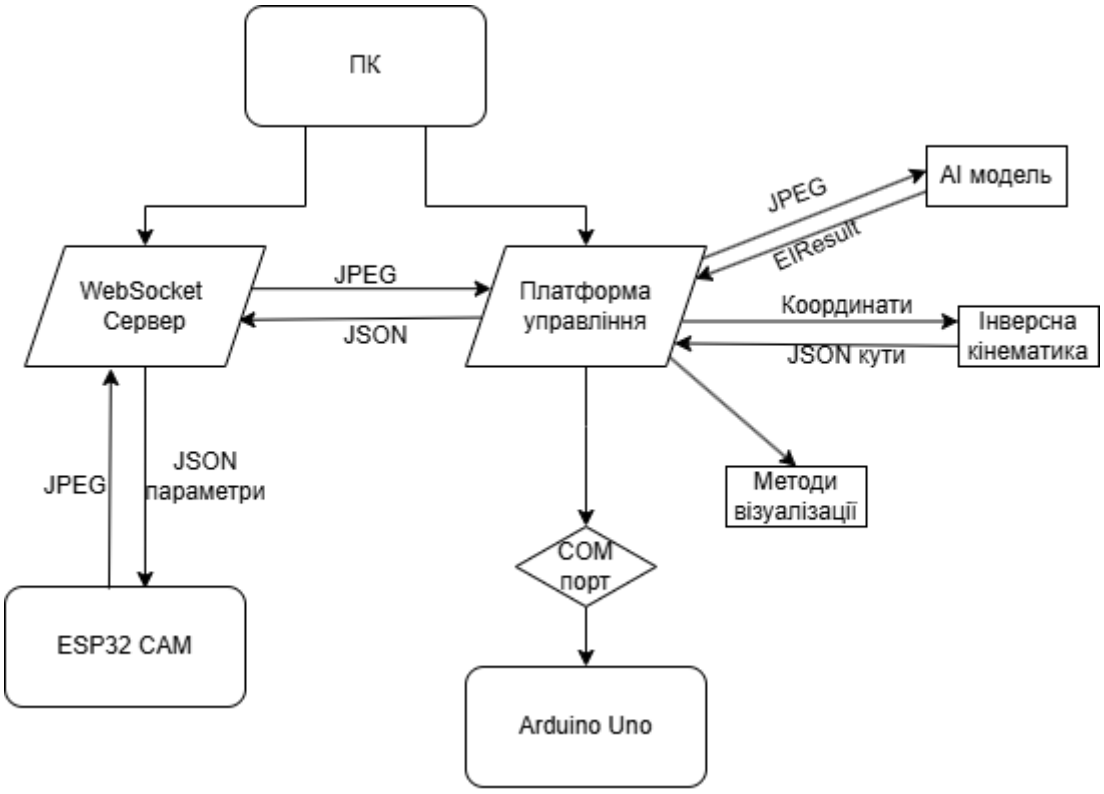
Літера	Маса	Масштаб
Н		

Аркуш 3	Аркушів 6

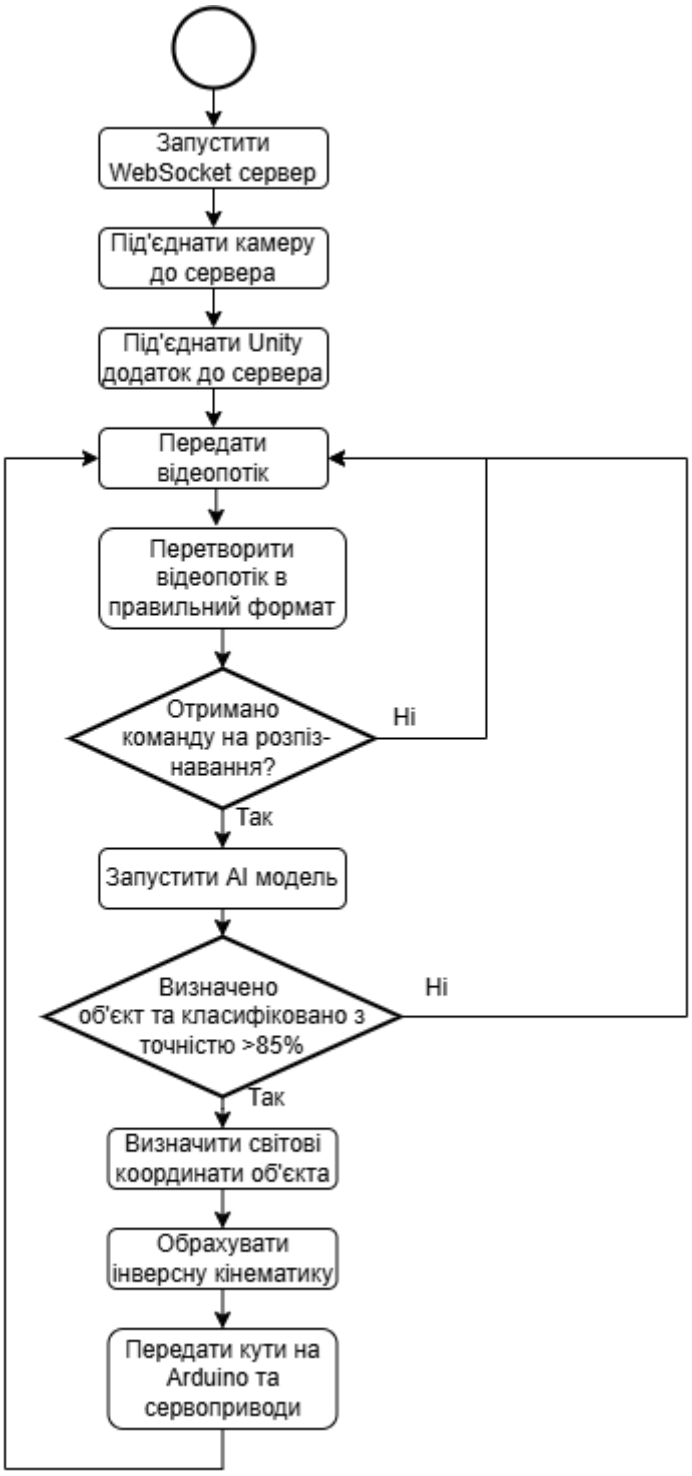
НУ «ЛП», ІКНІ,
каф. САП, гр. КН-412



					Бакалаврська кваліфікаційна робота	
Змн.	Арк.	Прізвище	Підпис	Дата	Віртуальна апаратна збірка	Літера: Н; Маса; Масштаб
Розроб.		Івасько О.І.				
Керівник		Колесник К.К.				Аркуш 4; Аркушів 6
Н. Контр.						НУ «ЛП», ІКНІ, каф. САП, гр. КН-412
Зав. каф.		Лобур М.В.				



					Бакалаврська кваліфікаційна робота	
Змн.	Арк.	Прізвище	Підпис	Дата	Структурна схема	Літера: Н; Маса; Масштаб
Розроб.		Івасько О.І.				Аркуш 5; Аркушів 6
Керівник		Колесник К.К.				НУ «ЛП», ІКНІ, каф. САП, гр. КН-412
Н. Контр.						
Зав. каф.		Лобур М.В.				



					Бакалаврська кваліфікаційна робота	
Змн.	Арк.	Прізвище	Підпис	Дата	Блок-схема роботи системи	Літера: Н; Маса; Масштаб
Розроб.		Івасько О.І.				
Керівник		Колесник К.К.				Аркуш 6 / Аркушів 6
Н. Контр.						НУ «ЛП», ІКНІ, каф. САП, гр. КН-412
Зав. каф.		Лобур М.В.				