ДОДАТОК А

Апробація результатів роботи

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2024

(Випуск 2)

[електронне видання]

KITAP

http://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy-avtomatizatsiyi-ta-mehatroniki-kitam

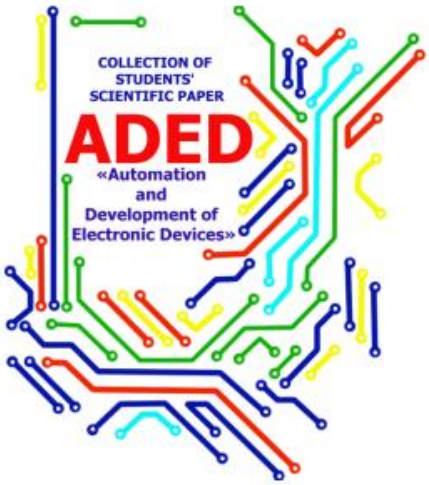
ITED

http://itez.zntu.edu.ua/

http://kafea.kdu.edu.ua

Харків 2024

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(КІТАР)



ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2024

(Випуск 2)

[електронне видання]

Харків 2024

УДК 004.738.5

АВТОМАЗОМАТИЗАЦІЯ ЛОГІСТИЧНИХ СИСТЕМ З ВИКОРИСТАННЯМ КІБЕРФІЗИЧНИХ ПІДХОДІВ

Краснопьоров М.Р., Казановська К.А.
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки, 14
E-mail: mykhailo.krasnoporov@nure.ua, kateryna.kazanovska@nure.ua

Анотація: У статті розглянуто логістичні системи. Доведено, що автоматизація логістичних систем є перспективним напрямком розвитку галузі комп'ютеризованих та робототехнічних системи, що може бути розширений та удосконалений шляхом використання кіберфізичних підходів.

Ключові слова: автоматизація, кіберфізичні системи, складська логістика, автоматизовані складські системи, роботи, RFID.

AUTOMATION OF LOGISTICS SYSTEMS USING CYBER-PHYSICAL APPROACHES

M.Krasnoporov, K. Kazanovska
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, pr. Nauki, 14
E-mail: mykhailo.krasnoporov@nure.ua, kateryna.kazanovska@nure.ua

Abstract: The article examines logistic systems. It has been proven that the automation of logistic systems is a promising direction for the development of computer-based and robotic systems, which can be expanded and improved through the use of cyber-physical approaches.

Keywords: automation, cyber-physical systems, warehouse logistics, automated warehouse systems, robots, RFID.

ВСТУП. В наш час автоматизація, роботизація різних процесів є дуже важливою, і такою, яка направлена на підвищення продуктивності, без втрати якості на будь-якому виробництві.
Також, автоматизовані процеси стрімко розвиваються в усіх сфер людської діяльності. Не є виключенням промислова автоматизація в логістичних процесах: виробництва, складування, відвантаження, транспортування та зберігання. Для реалізації цих напрямків, часто, стали використовувати концепції кіберфізичних систем, так званих cyber-physical system.

Кіберфізична система (КФС) – це механізм, що контролюється або відстежується комп'ютерними алгоритмами і тісно пов'язаний з Інтернетом та його користувачами. В кіберфізичних системах програмне забезпечення тісно пов'язано з фізичними об'єктами. Компоненти КФС взаємодіють на різних часових та просторових рівнях та можуть мати різні, відмінні одна від одної моделі поведінки та взаємодіяти одна з одною різними шляхами, які можуть змінюватися в залежності від контексту.

Прикладами кіберфізичних систем можна вважати «розумні» логістичні, безпілотні автомобільні системи, автоматизовані системи керування,

робототехнічні системи, самокеровані літальні апарати [1]. У КФС використовується міждисциплінарний підхід, який поєднує теорію кібернетики, мехатроніку, промисловий дизайн та науковий метод. Контроль процесів часто пов'язують з вбудованими системами, в яких більше уваги приділяють обчислювальним складовим, і менше — інтенсивному прив'язуванню обчислень до фізичних об'єктів. КФС дещо схожі за архітектурою на Інтернет речей, проте вони мають більш високий рівень взаємозв'язку між фізичними та комп'ютерними компонентами [2-3].

Враховуючи значну потребу нашої держави в постійних, швидких логістичних послугах, можна вважати, обрану тему актуальною та своєчасною.

Автоматизація складу дозволяє оптимізувати робочу силу і здійснити процеси на виробництві без втручання людини, але під повним її контролем. А також вирішить проблеми з витратами на утримання складу. Координують процес комп'ютери або промислові контролери, шляхом керування мобільними платформами, шатловими стелажами, конвеєрними системами і іншими об’єктами для автоматизації роботи складу.

Типи автоматизованих логістичних систем. Автоматизовані стелажі – це універсальне рішення, якщо потрібно оптимізувати простір складських приміщень. Мобільний стелаж – високоефективне рішення для організації для зберігання, що забезпечує найвищу щільність зберігання за рахунок максимального використання вільного простору приміщення. Ще їх називають пересувні стелажі, оскільки вони переміщаються в по напрямних та створюють прохід у необхідному місці.

Шатлові стелажі – відрізняються наявністю рейок на бічній частині. По рейках здійснюється транспортування піддонів за допомогою самохідної платформи, яка і називається шатлом. Завдяки взаємодії платформи зі стелажами площа складу оптимізується і використовується більш опціонально, в порівнянні з іншими способами організації зберігання вантажів. Шатл має автономне джерело живлення. При транспортуванні вантажів він переміщається по рейках уздовж одного каналу.

Переміщення між каналами виконується стандартним навантажувачем. Така конструкція має компактні розміри (часто невелику висоту) і береже робочий простір. Збільшує ефективність роботи в два рази, за рахунок гнучкості в роботі, можливо використовувати різні види палет в одній системі, працюють в режимі нон-стоп за рахунок легкої заміни батарей.

Мобільні системи збільшують ефективність використання площі від 50-100%. Конструкція влаштована таким чином: мобільні бази рухаються по напрямних, які встановлені в бетонній підлозі. У порівнянні зі звичайними фіксованими стійками, для яких потрібно прохід для роботи вантажної техніки через кожні 2 ряди, для мобільних систем необхідний тільки один проїзд і оператор за допомогою пульта відкриває той, який йому необхідний для роботи.

Гравітаційні стелажі є 2-х видів: поличні і палетні, які забезпечують максимальну щільність зберігання. У той же час, гравітаційні поличні системи часто використовують для комплектації замовлень, оскільки вони дозволяють виключити проходи оператора зібравши максимальну кількість товару на мінімальній площі. Конструкція гравітаційних стелажів уможливлює поділ потоків товару: з одного боку - розвантаження, з іншого-підбір.

Конвеєрні системи – це, транспортна система, яка забезпечує переміщення товару в необхідну точку на складі. Застосовуються в процесі завантаження-вивантаження, накопичення, переміщення палет і коробів.

Існує великий асортимент конвеєрних систем, серед яких, найбільшою популярністю в сфері складської логістики користуються стрічковий (прямий і підйомний), роликовий, гнучкий і спіральний конвеєра. Конвеєра сприяють скороченню часу і трудовитрат при переміщенні вантажів.

Особливо корисні на складах з високим товарообігом, адже вони дозволяють більш ефективно використовувати площу складу і збільшити його продуктивність в два рази.

Використовуючи зазначене обладнання та кіберфізичні підходи логічним є створення автоматизованих складських систем.

Автоматизована складська система або автоматизована тра́нспортно-складська система –це така система, що, призначена для автоматизації керування транспортними і складськими пристроями для складання, зберігання, тимчасового нагромадження, розвантаження та доставляння предметів та засобів праці, технологічного оснащення й видалення відходів.

Автоматизовані складські системи не тільки виключають ручну працю, а й дозволяють економити складські площі, прискорювати операції і покращувати контроль за матеріально-технічними запасами, оскільки комп'ютерно-інтегрований комплекс стежить за місцезнаходженням виробу на складі. Ці системи називають також автоматизованими складами.

Робота автоматизованих складів залежить від технологічних інновацій. До таких нововведень відносяться програмне забезпечення для управління складом і складські роботи. Деякі склади використовують один метод автоматизації, тоді як інші інтегрують різні види.

Фізична автоматизація складу означає використання складських роботів для управління запасами на складі. Таке управління передбачає: розвантаження пакетів, запис прийому, систематичне розташування пакетів, отримання конкретних пакетів для виконання замовлень.Усе це трудомісткі завдання, на виконання яких у ручному режимі потрібні години. Саме з цієї дилеми виникає потреба у фізичній автоматизації складів. Існують різні типи складських роботів, рекомендованих для різних складських застосувань.

Шарнірні руки робота – вони з'єднані роботизовані руки які є гнучкими та можуть бути запрограмовані на вивантаження, сортування та впорядкування пакетів. Вони часто сумісні зі складським програмним забезпеченням. Добрий виробник складських роботів також можна налаштувати їх за допомогою датчиків та інших засобів відповідно до потреб вашого складу.

Технології, сприятливі для людини – вони використовуються для збору пакетів і доставки їх персоналу складу, на відміну від працівників, які перетинають склад і перевозять вантажі. Приклади технологій GTP включають автоматизовані керовані транспортні засоби (AGV), Автоматизовані системи зберігання та пошуку (AS/RS)і автономні мобільні роботи (AMR).

Безпілотні керовані апарати – також широко відомі як дрони. Вони використовуються для перевезення невеликих вантажів серед інших завдань з обробки запасів.

Автоматизовані складські роботи мають вбудовані комп'ютеризовані системи, пов'язані з програмним забезпеченням колективного складу. Саме за допомогою цього програмного забезпечення персонал може спілкуватися з роботами, щоб спонукати їх виконувати різні завдання. Їхні рухи контролюються за допомогою датчиків. Це виключає AGV, для яких потрібен маршрут, намічений за допомогою магнітної смуги.

З іншого боку, шарнірні руки робота часто програмуються індивідуально для конкретних завдань. Вони також можуть бути оснащені датчиками, які керуватимуть ними під час сортування та впорядкування пакетів. Однак, на відміну від автоматизованих роботів, вони обробляють запаси зі стаціонарного положення.

Хоча складські роботи в основному розроблені для транспортування пакетів усередині складу, вони також мають інші допоміжні функції. Наприклад, дрони та AMR можуть ідентифікувати пакунки за допомогою датчиків RFID. Таким чином, їх можна використовувати не тільки для перевезення пакунків, але й для виконання інших завдань, як-от інвентаризація та сортування інвентарю.

Автоматизація цифрового складу також називають програмним забезпеченням для автоматизації цифрових процесів або автоматизації складу. Це використання унікального програмного забезпечення для обробки запасів на складі. В ідеалі, з моменту отримання інвентаризації до її відправлення, має бути запис про це в програмному забезпеченні автоматизації складу.

Програми для складської логістики містять, зазвичай, записи про всі предмети, місце їх зберігання та час їх надходження на склад, інші відомості, такі як дати закінчення терміну придатності, які є важливими для керування запасами.

Для збору даних використовують різні методи, такі як сканери штрих кодів, безпілотні літальні апарати, камери, давачі.

Співробітники використовують мобільний сканер штрих-кодів, щоб сканувати деталі штрих-коду інвентарю, а потім вони автоматично завантажуються в програмне забезпечення складу.Автоматизований збір даних проводять з залученням безпілотників або AMP с давачів: RFID використовуються для сканування інвентаризації та оновлення даних у режимі реального часу.

Мітки радіочастотної ідентифікації (RFID) закріплюються або вбудовуються у предмети. Вони передають та отримують інформацію за допомогою читача та антени.

Дані RFID дозволяють відстежувати товари, контролювати обладнання, аналізувати умови транспортування та виявляти неефективність операцій без ручної втручання. В результаті покращується видимість, зменшуються витрати на роботу, збільшується точність та швидкість.

Успіх автоматизації операцій в складському господарстві вимагає постійного зв'язку між кількома елементами в центрах дистрибуції. IoT зв'язує всі різні пристрої та системи в екосистемі складу та відстежує все, щоб збільшити ефективність роботи. Це технологія, яка забезпечує синхронну роботу роботизованих систем, AGV, AMR, ERP та WMS. Реалізовуючи пристрої IoT, менеджери складів можуть контролювати та координувати все з одного пункту взаємодії – від відвантаження, рівнів виробництва, запасів, доставок тощо.

ВИСНОВКИ. Отже, можна вважати, що логістика є перспективним напрямком розвитку для впровадження автоматизованих та автоматичних систем. При правильному та раціональному використанні інструментів автоматизації перевезень, зберігання, складування та зберігання, автоматизовані логістичні системи можуть працювати безперебійно, а їх обслуговування набагато дешевше.

Прийнято рішення продовжувати дослідження у напрямку промислової автоматизації для удосконалення процесів логістики, направляючи вектор на підвищення точності та ступеню автоматизації, шляхом використання та використання кіберфізичних підходів.

ЛІТЕРАТУРА

1. Євсєєв В. В. Методи та моделі кібер-фізичного керування процесами в організаційно-технічних виробничих об'єктах: автореф. дис. ... д-ра техн. наук : 05.13.07 "Автоматизація процесів керування" / В. В. Євсєєв ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2021. – 45 с.
2. Невлюдов І. Ш., і ін. (2019) Трансфер технологій у сучасній науці, освіті та виробництві в умовах четвертої промислової революції «ІНДУСТРІЯ 4.0» / Невлюдов І. Ш., Чала О. О., Олександров Ю. М. // Сучасний рух науки: тези доп. VIII міжнародної науково-практичної інтернет-конференції, 3-4 жовтня 2019 р. – Дніпро, 2, 604-608.
3. Чала, О., Сливка, А. (2023) Рівні засобів IIoT в інформаційних технологіях. Виробництво & Мехатронні Системи: матеріали VII Міжнародної конференції, Харків, С. 51-60.
4. Шостенко С. С. Архітектура програмного забезпечення для супроводження автоматизованих систем оповіщення на виробництві / С. С. Шостенко, О. О. Чала // Виробництво & Мехатронні Системи 2022 : зб. тез. доп. VI-ої Міжнар. конф., 21-22 жовтня 2022 р. – Харків, 2022. – С. 115-117.
5. Vladyslav Basiuk, Svitlana Maksymova, Olena Chala, & Olha Miliutina. (2023). Mobile Robot Position Determining Using Odometry Method. Multidisciplinary Journal of Science and Technology, 3(3), 227–234. Retrieved from https://mjstjournal.com/index.php/mjst/article/view/224
6. Nevliudov Igor, Maksymova Svitlana, Chala Olena, Bronnikov Artem, & Vzhesnievskyi Maksym. (2023). Automated Logistics Processes Improvement in Logistics Facilities. Multidisciplinary Journal of Science and Technology, 3(3), 157–170. Retrieved from https://mjstjournal.com/index.php/mjst/article/view/202
7. Nevliudov, I., Vzhesnievskyi, M., Romashov, Y. і Chala, O. (2023) «Математичне моделювання мехатронних шатлів як об’єктів автоматизації для багаторівневих систем внутрішньоскладської логістики», Сучасний стан наукових досліджень та технологій в промисловості, (4 (26), с. 135–144. doi: 10.30837/ITSSI.2023.26.135.
8. Невлюдов І.Ш., Демська Н.П., Чала О.О., Демська А.І. (2018). Групове управління гнучкими виробничими системами у виготовленні МЕМС виробів. Міжнародна науково-практична конференція «Математичне моделювання процесів в економіці та управлінні проектами і програмами (ММП2018)», Коблево, 10-14 вересня 2018 р. Харків: ХНУРЕ, 101 -103

9. Lighting Control Module Development / Y. Vizir, O. Chala, S. Maksymova, Ahmad Alkhalaileh // Journal of Universal Science Research. – 2023. – № 1(12). – P. 645–657. URI https://openarchive.nure.ua/handle/document/27446

10. Automated System Development for the Printed Circuit Boards Optical Inspection Using Machine Learning Methods / I. Nevliudov, I. Botsman, O. Chala, K. Khrustalev // INFORMATION SYSTEMS AND TECHNOLOGIES (IST'2021) : proceedings of the 10-th International Scientific and Technical Conference, September 13-19, Odessa, 2021. – P. 234-238. URI https://openarchive.nure.ua/handle/document/18350

11. Nevliudov, I., Vzhesnievskyi, M., Romashov, Y., & Chala, O. (2023). Mathematical modeling of mechatronic shuttles as automation objects for multilevel systems of intra-warehouse logistics. INNOVATIVE TECHNOLOGIES AND SCIENTIFIC SOLUTIONS FOR INDUSTRIES, (4(26), 135–144. https://doi.org/10.30837/ITSSI.2023.26.135

12. O. Chala, A. Bronnikov, N. Igor and D. Mospan, "The Use of Neural Networks for the Technological Objects Recognition Tasks in Computer-Integrated Manufacturing," 2022 IEEE 4th International Conference on Modern Electrical and Energy System (MEES), Kremenchuk, Ukraine, 2022, pp. 1-5, doi: 10.1109/MEES58014.2022.10005750.

13. Гіль А. Промислові інтерфейси та протоколи передачі даних інтегрованих систем для автоматизованого управління в умовах Industry 4.0 / А. Гіль, О. Чала, О. Филипенко // Виробництво & Мехатронні Системи 2021 : матеріали V-ої Міжнар. конф., 21-22 жовтня 2021 р. - Харків, 2021. - С. 127-30. https://openarchive.nure.ua/handle/document/18375

ДОДАТОК Б

Код програми

Класи модуля комп’ютерного зору:

```
namespace ComputerVision;
/// <summary>
/// Клас для реалізації комп'ютерного зору
/// Забезпечує підключення до камери, захоплення відеопотоку,
/// отримання окремих кадрів та управління процесом відеострімінгу
/// </summary>
public class ComputerVisionProcessor : IDisposable
{
    // HttpClient для завантаження зображень з IP-камери
    private readonly HttpClient _httpClient = new HttpClient();
    // Подія, що повідомляє про зміну статусу
    public event Action<string>? StatusChanged;
    // Подія для зовнішньої обробки помилок відеопотоку
    public event Action<Exception>? VideoStreamError;
    private string _cameraStatusMessage = "Не встановлено";
    public string CameraStatusMessage
    {
        get => _cameraStatusMessage;
        private set
        {
            _cameraStatusMessage = value;
            // Виклик події оновлення статусу
            StatusChanged?.Invoke(_cameraStatusMessage);
        }
    }

    // URL потоку камери
    private readonly string _ipCameraUrl;

    // Подія, що повідомляє про зміну статусу відеопотоку
    public event Action<bool>? VideoRunningChanged;
    private bool _isRunning = false;
    public bool VideoRunningStatus
    {
        get => _isRunning;
        private set
        {
            _isRunning = value;
            // Виклик події оновлення статусу
            VideoRunningChanged?.Invoke(_isRunning);
        }
    }

    // Токен для скасування асинхронної задачі
```

```
    private CancellationTokenSource? _cts;

    /// <summary>
    /// Конструктор класу. Приймає URL камери
    /// </summary>
    public ComputerVisionProcessor(string ipCameraUrl)
    {
        _ipCameraUrl = ipCameraUrl;
        // таймаут для HTTP запитів (3 секунд)
        _httpClient.Timeout = TimeSpan.FromSeconds(3);
    }

    /// <summary>
    /// Асинхронний метод для захоплення кадру з камери
    /// </summary>
    /// <returns>Масив байт, що містить JPEG-зображення з камери</returns>
    public async Task<byte[]> CaptureFrameBytesAsync()
    {
        byte[] imageBytes = await _httpClient.GetByteArrayAsync(_ipCameraUrl)
            .ConfigureAwait(false);
        if (imageBytes == null || imageBytes.Length == 0)
            throw new Exception("Не вдалося отримати кадр. Отримано порожній масив байтів.");
        return imageBytes;
    }

    /// <summary>
    /// Метод для ініціалізації відеозахоплення
    /// </summary>
    public void InitializeVideo()
    {
        VideoRunningStatus = false;
    }

    /// <summary>
    /// Асинхронний метод для запуску відеопотоку
    /// Завантажує JPEG-зображення з камери та передає їх через callback-метод
    /// </summary>
    public async Task StartVideoStreamAsync(Action<Bitmap> onFrameReceived, int frameDelay = 20)
    {
        if (VideoRunningStatus)
            return;

        VideoRunningStatus = true;
        _cts = new CancellationTokenSource();

        while (VideoRunningStatus && !_cts.Token.IsCancellationRequested)
        {
            try
            {
                byte[] imageBytes = await _httpClient.GetByteArrayAsync(_ipCameraUrl)
```

```
                .ConfigureAwait(false);
            using Mat mat = Cv2.ImDecode(imageBytes, ImreadModes.Color);
            if (mat.Empty())
                throw new Exception("Отримано пустий кадр з камери.");

            Bitmap bitmap = MatToBitmap(mat);
            onFrameReceived?.Invoke(bitmap);

            await Task.Delay(frameDelay, _cts.Token)
                .ConfigureAwait(false);
        }
        catch (Exception ex)
        {
            // Виклик події для зовнішньої обробки
            VideoStreamError?.Invoke(ex);
            // зупинка циклу, щоб статус оновився
            VideoRunningStatus = false;
            break;
        }
    }
}

/// <summary>
/// Метод для перевірки з'єднання з камерою
/// Якщо підключення вдалося, оновлюється властивість CameraStatusMessage і повертається true
/// Інакше, при невдалої спробі, властивість містить повідомлення про помилку і викидається виняток
/// </summary>
/// <returns>true, якщо з'єднання встановлено, інакше false.</returns>
public async Task<bool> CheckCameraConnectionAsync()
{
    try
    {
        byte[] imageBytes = await _httpClient.GetByteArrayAsync(_ipCameraUrl)
            .ConfigureAwait(false);
        if (imageBytes == null || imageBytes.Length == 0)
        {
            CameraStatusMessage = "Камера не відповідає або повернула порожній кадр.";
            Console.WriteLine(DateTime.Now + "  " + CameraStatusMessage);
            throw new Exception(CameraStatusMessage);
        }
        CameraStatusMessage = "Підключено";
        return true;
    }
    catch (Exception ex)
    {
        CameraStatusMessage = $"Помилка підключення: {ex.Message}";
        Console.WriteLine(DateTime.Now + "  " + CameraStatusMessage);
        throw new Exception(CameraStatusMessage);
    }
```

```
        }

        /// <summary>
        /// Перетворює об'єкт Mat в об'єкт Bitmap
        /// </summary>
        /// <param name="mat">Вхідне зображення у форматі Mat</param>
        /// <returns>Зображення у форматі Bitmap</returns>
        private static Bitmap MatToBitmap(Mat mat)
        {
            using MemoryStream stream = new MemoryStream();
            // Запис Mat у потік у форматі BMP.
            mat.WriteToStream(stream, ".bmp");
            stream.Position = 0;
            return new Bitmap(stream);
        }

        /// <summary>
        /// Метод для зупинки відеопотоку
        /// </summary>
        public void StopVideoStream()
        {
            VideoRunningStatus = false;
            _cts?.Cancel();
        }

        /// <summary>
        /// Звільнення використовуваних ресурсів
        /// </summary>
        public void Dispose()
        {
            _httpClient.Dispose();
            _cts?.Dispose();
        }
    }


namespace ArUcoObjectMeasurement
{
    /// <summary>
    /// Результат вимірювання одного об’єкта
    /// </summary>
    public class ObjectMeasurement : INotifyPropertyChanged
    {
        /// <summary>
        /// Ідентифікатор об’єкта
        /// </summary>
        public int ObjectId { get; set; }

        /// <summary>
        /// Центр об’єкта по осі X (у фізичних одиницях, см)
        /// </summary>
        private float _centerX;
```

```
public float CenterX
{
   get => _centerX;
   set
   {
      if (_centerX != value)
      {
         _centerX = value;
         OnPropertyChanged(nameof(CenterX));
      }
   }
}

/// <summary>
/// Центр об’єкта по осі Y (у фізичних одиницях, см)
/// </summary>
private float _centerY;
public float CenterY
{
   get => _centerY;
   set
   {
      if (_centerY != value)
      {
         _centerY = value;
         OnPropertyChanged(nameof(CenterY));
      }
   }
}

/// <summary>
/// Площа контуру (у пікселях)
/// </summary>
public double ContourArea { get; set; }

/// <summary>
/// класифікація об’єкту (від ШІ)
/// </summary>
private string _classification = "";
public string Classification
{
   get => _classification;
   set
   {
      if (_classification != value)
      {
         _classification = value;
         OnPropertyChanged(nameof(Classification));
      }
   }
}
```

```
        /// <summary>
        /// впевненість класифікації об’єкту (від ШІ)
        /// </summary>
        private double _confidence;
        public double Confidence
        {
            get => _confidence;
            set
            {
                if (_confidence != value)
                {
                    _confidence = value;
                    OnPropertyChanged(nameof(Confidence));
                }
            }
        }

        /// <summary>
        /// сортувальний бокс (користувач призначає)
        /// </summary>
        private SortBoxModel _sortBox;
        public SortBoxModel SortBox
        {
            get => _sortBox;
            set
            {
                if (_sortBox != value)
                {
                    _sortBox = value;
                    OnPropertyChanged(nameof(SortBox));
                }
            }
        }

        // Реалізація INotifyPropertyChanged:
        public event PropertyChangedEventHandler? PropertyChanged;

        protected void OnPropertyChanged(string propertyName)
            => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    /// <summary>
    /// Клас для обробки зображень з ArUco-маркерами, визначення контурів об’єктів та перетворення
    /// у фізичну систему координат. Результати обчислень зберігаються у вигляді вирізаних зображень
    /// та списку вимірювань
    /// </summary>
    public class ArUcoObjectMeasurementProcessor
    {
        /// <summary>
        /// Шлях до папки для збереження вирізаних зображень
```

```
        /// </summary>
        public string OutputFolder { get; set; }

        /// <summary>
        /// Ширина столу (фізична координата правого краю, см)
        /// </summary>
        public float TableWidth { get; set; }

        /// <summary>
        /// Висота столу (фізична координата верхнього краю, см)
        /// </summary>
        public float TableHeight { get; set; }

        /// <summary>
        /// Мінімальна площа контуру, нижче якої контур ігнорується
        /// </summary>
        public double MinAreaThreshold { get; set; }

        /// <summary>
        /// Поріг відстані для фільтрації контурів, близьких до маркерів
        /// </summary>
        public double MarkerDistanceThreshold { get; set; }

        /// <summary>
        /// Використовуваний словник ArUco-маркерів
        /// </summary>
        public PredefinedDictionaryName MarkerDictionaryName { get; set; }


        private Dictionary _dictionary;
        private DetectorParameters _detectorParams = new DetectorParameters();

        /// <summary>
        /// Конструктор за замовчуванням встановлює дефолтні значення для параметрів роботи
        /// </summary>
        public ArUcoObjectMeasurementProcessor()
        {
            // вирізані об’єкти зберігаються у папці "CroppedObjects"
            OutputFolder = "CroppedObjects";
            // Фізичні розміри столу
            TableWidth = App.ConfigService.Config.Table.Width;
            TableHeight = App.ConfigService.Config.Table.Height;
            // Параметри для фільтрації контурів
            MinAreaThreshold = 2000.0;
            MarkerDistanceThreshold = 40.0;
            // Використання стандартного словника з маркерами 4x4 (50 маркерів)
            MarkerDictionaryName = PredefinedDictionaryName.Dict4X4_50;

            _dictionary = CvAruco.GetPredefinedDictionary(MarkerDictionaryName);
        }
```

```
    /// <summary>
    /// Основний метод, що обробляє зображення за заданим шляхом. Він виконує:
    /// 1. Детекцію ArUco-маркерів
    /// 2. Побудову перспективної трансформації на основі центрів маркерів
    /// 3. Пошук та фільтрацію контурів
    /// 4. Вимірювання параметрів об’єктів
    /// 5. Візуалізацію результатів та збереження вирізаних об’єктів
    /// </summary>
    public (List<ObjectMeasurement> Measurements, Mat Image, int FoundedMarkersCount) Image(byte[] image)
    {
        //декодуємо отримані байти у зображення типу mat
        Mat frame = Cv2.ImDecode(image, ImreadModes.Color);
        if (frame.Empty())
        {
            Logger.WriteToLogs("Не вдалося декодувати отримане зображення.");
            return (null, null, 0);
        }

        // детекція ArUco маркерів
        Point2f[][] markerCorners;
        int[] markerIds;
        Point2f[][] rejectedCandidates;
        CvAruco.DetectMarkers(frame, _dictionary, out markerCorners, out markerIds, _detectorParams, out rejectedCandidates);

        if (markerIds == null || markerIds.Length == 0)
        {
            string message = "Markers not found";
            int baseline = 0;
            Size textSize = Cv2.GetTextSize(message, HersheyFonts.HersheySimplex, 1.0, 2, out baseline);
            Point textOrigin = new Point((frame.Width - textSize.Width) / 2, (frame.Height + textSize.Height) / 2);
            Cv2.PutText(frame, message, textOrigin, HersheyFonts.HersheySimplex, 1.0, Scalar.Red, 2);
            return (new List<ObjectMeasurement>(), frame, 0);
        }

        // малювання знайдених маркерів на основному зображенні
        CvAruco.DrawDetectedMarkers(frame, markerCorners, markerIds);

        // обчислення центрів маркерів та формування словника: id -> центр
        Dictionary<int, Point2f> markerCenters = new Dictionary<int, Point2f>();
        for (int i = 0; i < markerIds.Length; i++)
        {
            int id = markerIds[i];
            Point2f center = new Point2f(markerCorners[i].Average(pt => pt.X),
                                         markerCorners[i].Average(pt => pt.Y));
            markerCenters[id] = center;
        }
```

```
        // Перевірка наявності необхідних маркерів (0, 1, 2, 3)
        int[] requiredIds = new int[] { 0, 1, 2, 3 };
        foreach (int reqId in requiredIds)
        {
            if (!markerCenters.ContainsKey(reqId))
            {
                throw new InvalidOperationException($"Маркер з id {reqId} не знайдено.");
            }
        }

        // формування точок для перспективної трансформації
        // Порядок: 0 – нижній лівий, 1 – верхній лівий, 2 – верхній правий, 3 – нижній правий
        Point2f[] srcPoints = new Point2f[4]
        {
            markerCenters[0],
            markerCenters[1],
            markerCenters[2],
            markerCenters[3]
        };

        // Побудова фізичної системи координат, за розмірами столу
        Point2f[] dstPoints = new Point2f[4]
        {
            new Point2f(2, 2),
            new Point2f(2, TableHeight - 2),
            new Point2f(TableWidth - 2, TableHeight - 2),
            new Point2f(TableWidth - 2, 2)
        };

        Mat perspectiveMatrix = Cv2.GetPerspectiveTransform(srcPoints, dstPoints);

        // Попередня обробка зображення для пошуку контурів (відтінки сірого, поріг, морфологія)
        Mat gray = new Mat();
        Cv2.CvtColor(frame, gray, ColorConversionCodes.BGR2GRAY);
        Mat thresh = new Mat();
        Cv2.Threshold(gray, thresh, 40, 255, ThresholdTypes.BinaryInv | ThresholdTypes.Otsu);
        Mat kernel = Cv2.GetStructuringElement(MorphShapes.Rect, new Size(3, 3));
        Cv2.MorphologyEx(thresh, thresh, MorphTypes.Close, kernel, iterations: 2);

        // Пошук контурів
        Point[][] contours;
        HierarchyIndex[] hierarchy;
        Cv2.FindContours(thresh, out contours, out hierarchy, RetrievalModes.External, ContourApproximationModes.ApproxSimple);

        if (contours == null || contours.Length == 0)
        {
            throw new InvalidOperationException("Об'єкти не знайдено.");
        }
```

```
        // підготовка анотованого зображення (черзе клонування основного)
        Mat annotatedImage = frame.Clone();
        // повторне малювання маркерів для видимості на анотованому зображенні
        CvAruco.DrawDetectedMarkers(annotatedImage, markerCorners, markerIds);

        List<ObjectMeasurement> measurements = new List<ObjectMeasurement>();
        int objectCount = 0;

        // створення папки для збереження вирізаних об’єктів
        Directory.CreateDirectory(OutputFolder);

        // обробка кожного контуру: фільтрація, обчислення параметрів, анотація та збереження вирізаних фрагментів
        foreach (var contour in contours)
        {
            double area = Cv2.ContourArea(contour);
            if (area < MinAreaThreshold)
                continue;

            Rect originalRect = Cv2.BoundingRect(contour);
            // Відкидання контурів, що розташовані близько до країв зображення
            if (originalRect.X <= 2 || originalRect.Y <= 2 ||
                originalRect.X + originalRect.Width >= frame.Width - 2 ||
                originalRect.Y + originalRect.Height >= frame.Height - 2)
                continue;

            Point contourCenter = new Point(
                originalRect.X + originalRect.Width / 2,
                originalRect.Y + originalRect.Height / 2
            );

            // фільтрація контурів, що знаходяться близько до маркерів
            bool isMarkerContour = false;
            foreach (var markerCenter in markerCenters.Values)
            {
                double distance = Math.Sqrt(Math.Pow(contourCenter.X - markerCenter.X, 2) +
                                            Math.Pow(contourCenter.Y - markerCenter.Y, 2));
                if (distance < MarkerDistanceThreshold)
                {
                    isMarkerContour = true;
                    break;
                }
            }
            if (isMarkerContour)
                continue;

            // перетворення координат контуру у фізичну систему за допомогою перспективної трансформації
            Point2f[] contourF = Array.ConvertAll(contour, pt => new Point2f(pt.X, pt.Y));
            Point2f[] transformedContour = Cv2.PerspectiveTransform(contourF, perspectiveMatrix);
```

```
            Rect rectTransformed = Cv2.BoundingRect(transformedContour);
            Rect2f boundingRect = new Rect2f(rectTransformed.X, rectTransformed.Y, rectTransformed.Width, rectTransformed.Height);
            float objectCenterX = boundingRect.X + boundingRect.Width / 2;
            float objectCenterY = boundingRect.Y + boundingRect.Height / 2;

            //формування об'єктних вимірювань та збереження результату
            objectCount++;
            ObjectMeasurement measurement = new ObjectMeasurement
            {
               ObjectId = objectCount,
               CenterX = objectCenterX,
               CenterY = objectCenterY,
               ContourArea = area
            };
            measurements.Add(measurement);

            // анотація зображення малювання контуру, прямокутника та тексту
            Cv2.DrawContours(annotatedImage, new Point[][] { contour }, -1, Scalar.Red, 2);
            Cv2.Rectangle(annotatedImage, originalRect, Scalar.Green, 2);
            string text = $"X: {objectCenterX:F2}, Y: {objectCenterY:F2}";
            Point textOrg = new Point(originalRect.X, originalRect.Y - 5);
            Cv2.PutText(annotatedImage, text, textOrg, HersheyFonts.HersheySimplex, 0.7, Scalar.Blue, 2);

            // Збереження вирізаного об’єкту
            SaveCroppedObject(frame, originalRect, objectCount);
         }

         // повернення списку вимірювань, анотованого зображення та кількість знайдених маркерів
         return (measurements, annotatedImage, markerIds.Length);
      }

      /// <summary>
      /// Обробляє зображення, представлене у вигляді масиву байтів
      /// Декодує зображення, виконує детекцію ArUco-маркерів,
      /// відображає їх на зображенні, додає текстове повідомлення з кількістю знайдених маркерів,
      /// і повертає оброблене зображення та кількість маркерів
      /// </summary>
      public (Mat ProcessedImage, int MarkerCount) ProcessImage(byte[] imageBytes)
      {
         // Декодування отриманого масиву байт у зображення типу Mat
         Mat frame = Cv2.ImDecode(imageBytes, ImreadModes.Color);
         if (frame.Empty())
         {
            Console.WriteLine("Не вдалося декодувати отримане зображення.");
            return (null, 0);
         }

         // дтекція ArUco маркерів на зображенні
```

```
        Point2f[][] markerCorners;
        int[] markerIds;
        Point2f[][] rejectedCandidates;
        CvAruco.DetectMarkers(frame, _dictionary, out markerCorners, out markerIds, _detectorParams, out rejectedCandidates);

        // Якщо маркери знайдено то малюємо їх на зображенні
        if (markerIds != null && markerIds.Length > 0)
        {
            CvAruco.DrawDetectedMarkers(frame, markerCorners, markerIds);
            string message = $"Found {markerIds.Length} markers";
            int baseline = 0;
            Size textSize = Cv2.GetTextSize(message, HersheyFonts.HersheySimplex, 1.0, 2, out baseline);
            // Розташування тексту посередині зображення
            Point textOrigin = new Point((frame.Width - textSize.Width) / 2, (frame.Height + textSize.Height) / 2);
            Cv2.PutText(frame, message, textOrigin, HersheyFonts.HersheySimplex, 1.0, Scalar.Red, 2);
            return (frame, markerIds.Length);
        }
        else
        {
            // Якщо маркерів не знайдено то виводимо відповідне повідомлення
            string message = "Markers not found";
            int baseline = 0;
            Size textSize = Cv2.GetTextSize(message, HersheyFonts.HersheySimplex, 1.0, 2, out baseline);
            Point textOrigin = new Point((frame.Width - textSize.Width) / 2, (frame.Height + textSize.Height) / 2);
            Cv2.PutText(frame, message, textOrigin, HersheyFonts.HersheySimplex, 1.0, Scalar.Red, 2);
            return (frame, 0);
        }
    }


    /// <summary>
    /// Приватний метод для вирізання заданої області з зображення та збереження у вказану папку
    /// </summary>
    private void SaveCroppedObject(Mat image, Rect rect, int index)
    {
        //Встановлення величини додаткового простору навколо об’єкта (в пікселях)
        int margin = 50;

        // обчислення нових координат і розмірів з урахуванням margin, не виходячи за межі зображення
        int newX = Math.Max(rect.X - margin, 0);
        int newY = Math.Max(rect.Y - margin, 0);
        int newWidth = Math.Min(rect.Width + 2 * margin, image.Width - newX);
        int newHeight = Math.Min(rect.Height + 2 * margin, image.Height - newY);
```

```
            // створення нового rect з додатковим простором
            Rect extendedRect = new Rect(newX, newY, newWidth, newHeight);

            // вирізання області з зображення за допомогою нового прямокутника
            Mat cropped = new Mat(image, extendedRect);
            string filename = Path.Combine(OutputFolder, $"object_{index}.png");
            Cv2.ImWrite(filename, cropped);
            Logger.WriteToLogs($"Об'єкт #{index} збережено у файл: {filename}");
        }

    }
}
```

Клас керування роботом:

```
namespace RobotController;
    /// <summary>
    /// Клас RobotController забезпечує управління роботизованим маніпулятором
    /// через G-коди, що передаються по послідовному порт
    /// </summary>
    public class RobotController : IDisposable
    {
        private SerialPort _serialPort;
        private bool _disposed = false;

        /// <summary>
        /// Подія, що повідомляє про зміну статусу підключення
        /// </summary>
        public event Action<string>? StatusChanged;

        /// <summary>
        /// Властивість, що повертає стан підключення до порту
        /// </summary>
        public bool IsConnected => _serialPort != null && _serialPort.IsOpen;

        // Зберігаємо назва порту
        private readonly string _portName;
        /// <summary>
        /// Назва порту, в який підключено робот
        /// </summary>
        public string PortName => _portName;

        /// <summary>
        /// Конструктор класу. Ініціалізує з'єднання із заданим послідовним портом
        /// </summary>
        /// <param name="portName">Назва COM-порту</param>
        /// <param name="baudRate">Швидкість з'єднання (бод). За замовчуванням –
115200</param>
        public RobotController(string portName, int baudRate = 115200)
        {
            _portName = portName;
```

```
    _serialPort = new SerialPort(portName, baudRate)
    {
        DataBits = 8,
        Parity = Parity.None,
        StopBits = StopBits.One,
        Handshake = Handshake.None,
        DtrEnable = true,
        RtsEnable = true,
        ReadTimeout = 2000, // мс
        WriteTimeout = 2000 // мс
    };

    try
    {
        // відкрити порт
        _serialPort.Open();

        // Якщо порт відкрився успішно, повідомити вищому рівню
        StatusChanged?.Invoke("Підключено");

        // затримка для ініціалізації контролера
        Thread.Sleep(2000);
    }
    catch (Exception e)
    {
        Logger.WriteToLogs($"{DateTime.Now} e.Message");
        throw;
    }

    // калібрування робота
    HomeAll();
    // скидаємо всі рядки
    _serialPort.DiscardInBuffer();
}

/// <summary>
/// Статичний метод для отримання списку доступних послідовних портів
/// Це дозволяє користувачу вибрати потрібний порт для підключення до робота
/// </summary>
/// <returns>Масив рядків з назвами портів</returns>
public static string[] GetAvailablePorts()
{
    return SerialPort.GetPortNames();
}

/// <summary>
/// Деструктор класу
/// </summary>
~RobotController()
{
    Dispose(false);
}
```

```
        /// <summary>
        /// Метод Dispose для звільнення ресурсів та закриття послідовного порту
        /// </summary>
        /// <summary>
        /// Закриває порт і звільняє ресурси
        /// </summary>
        public void Dispose()
        {
            if (_disposed)
                return;

            _disposed = true;

            try
            {
                if (_serialPort != null && _serialPort.IsOpen)
                {
                    _serialPort.Close();
                    StatusChanged?.Invoke("Відключено");
                }
            }
            catch (Exception e)
            {
                Logger.WriteToLogs($"{DateTime.Now} Помилка при закритті порту: {e.Message}");
            }
            finally
            {
                _serialPort?.Dispose();
            }
        }

        /// <summary>
        /// Захищена імплементація Dispose
        /// </summary>
        /// <param name="disposing">Якщо true, звільняються як керовані, так і некеровані ресурси</param>
        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    if (_serialPort != null)
                    {
                        if (_serialPort.IsOpen)
                        {
                            _serialPort.Close();
                        }
                        _serialPort.Dispose();
                    }
```

```
            }
            _disposed = true;
        }
    }

    /// <summary>
    /// Універсальний метод відправлення команд у вигляді рядка до робота
    /// </summary>
    /// <param name="command">Команда для відправлення (наприклад, "G90" або "G0 X10 Y20 Z30 F1000")</param>
    public void SendCommand(string command)
    {
        if (_serialPort == null || !_serialPort.IsOpen)
        {
            throw new InvalidOperationException("Serial port is not open.");
        }

        // Очищення вхідного буферу перед тим, як писати нову команду
        _serialPort.DiscardInBuffer();

        if (!command.EndsWith("\r\n"))
            command += "\r\n";

        _serialPort.Write(command);
    }

    /// <summary>
    /// Лінійне переміщення (швидкий/робочий рух) – G0/G1 з вказаними координатами та швидкістю
    /// </summary>
    /// <param name="x">Координата X</param>
    /// <param name="y">Координата Y</param>
    /// <param name="z">Координата Z</param>
    /// <param name="feedRate">Швидкість переміщення.</param>
    public void MoveLinear(double x, double y, double z, double feedRate = 0)
    {
        string command = $"G0 X{x:0.###}Y{y:0.###}Z{z:0.###}";
        Console.WriteLine(command);
        Logger.WriteToLogs(command);
        if (feedRate > 0)
        {
            command += $" F{feedRate:0.###}";
        }

        SendCommand(command);
    }

    /// <summary>
    /// Команда повернення в "home"-позицію (G28)
    /// </summary>
    public void HomeAll()
    {
```

```
        SendCommand("G28");
    }

    /// <summary>
    /// виконує захоплення об’єкту
    /// </summary>
    public void GrabObject()
    {
        SendCommand("M3");
    }
    /// <summary>
    /// відпускає об’єкт
    /// </summary>
    public void ReleaseObject()
    {
        SendCommand("M5");
    }

    /// <summary>
    /// Запит поточних координат (M114). Після надсилання читаємо всі рядки відповіді
    /// до таймауту і повертаємо лише останній
    /// </summary>
    /// <returns>Об’єкт RobotResponse, зібраний з останнього рядка відповіді</returns>
    public RobotResponse ReportCoordinates()
    {
        if (_serialPort == null || !_serialPort.IsOpen)
            throw new InvalidOperationException("Serial port is not open.");

        // Перед надсиланням очистити буфери
        _serialPort.DiscardInBuffer();
        _serialPort.DiscardOutBuffer();

        // надсилання запиту
        SendCommand("M114");

        var lines = new List<string>();
        // Короткий таймаут, щоб швидко зупинитися, коли рядків більше нема
        int originalTimeout = _serialPort.ReadTimeout;
        _serialPort.ReadTimeout = 200;

        try
        {
            while (true)
            {
                // читання рядоку до CRLF
                string response = _serialPort.ReadLine().Trim();
                if (!string.IsNullOrEmpty(response))
                    lines.Add(response);
            }
        }
        catch (TimeoutException)
```

```
            {
            }
            finally
            {
                // повернення початкового таймауту
                _serialPort.ReadTimeout = originalTimeout;
            }

            if (lines.Count == 0)
                throw new InvalidOperationException("No response received for M114.");

            // розпарсити останній рядок відповіді
            string lastLine = lines[lines.Count - 1];
            return RobotResponse.Parse(lastLine);
        }

        /// <summary>
        /// Метод для зчитування відповіді з порту з заданим тайм-аутом
        /// </summary>
        public string ReadResponse(int timeoutMilliseconds = 2000)
        {
            if (_serialPort == null || !_serialPort.IsOpen)
            {
                throw new InvalidOperationException("Serial port is not open.");
            }

            _serialPort.ReadTimeout = timeoutMilliseconds;
            try
            {
                string response = _serialPort.ReadLine();
                return response.Trim();
            }
            catch (TimeoutException)
            {
                return "No response (timeout).";
            }
        }
    }

    public class RobotResponse
    {
        /// <summary>Час у форматі hh:mm:ss.fff</summary>
        public DateTime Timestamp { get; set; }

        /// <summary>Рівень повідомлення, наприклад INFO або ERROR</summary>
        public string Level { get; set; } = string.Empty;

        /// <summary>Текст повідомлення без координат</summary>
        public string Message { get; set; } = string.Empty;

        /// <summary>G-код або інша команда, якщо виявлено (наприклад
"G28")</summary>
```

```
    public string? Command { get; set; }

    public double? X { get; set; }
    public double? Y { get; set; }
    public double? Z { get; set; }
    public double? E { get; set; }

    private static readonly Regex LineRegex = new Regex(
        // необов'язковий префікс часу та стрілки
        @"^(?:(?<time>\d{2}:\d{2}:\d{2}\.\d{3})\s*->\s*)?"
        + @"(?<level>\w+):\s*"                    // рівень
        + @"(?<msg>.*?)"                          // основне повідомлення
        +          @"(?:\s*\[X:(?<x>[-\d\.]+)\s+Y:(?<y>[-\d\.]+)\s+Z:(?<z>[-\d\.]+)\s+E:(?<e>[-
\d\.]+)\])?"  // координати
        + @"\s*$",
        RegexOptions.Compiled
    );

    /// <summary>
    /// Розпарсити одну строку логів у об’єкт RobotResponse
    /// Поля, що не знайдені, залишаються null або порожніми
    /// Якщо вхідний рядок без таймштампу, Timestamp = DateTime.Now
    /// </summary>
    public static RobotResponse Parse(string line)
    {
        var m = LineRegex.Match(line);
        if (!m.Success)
            throw new FormatException($"Неможливо розпарсити рядок: {line}");

        // Парсимо час, якщо він є; інакше ставимо поточний
        DateTime timestamp;
        var timeStr = m.Groups["time"].Value;
        if (!string.IsNullOrEmpty(timeStr))
        {
            timestamp          =          DateTime.ParseExact(timeStr,          "HH:mm:ss.fff",
CultureInfo.InvariantCulture);
        }
        else
        {
            timestamp = DateTime.Now;
        }

        // Рівень та повідомлення
        var level = m.Groups["level"].Value;
        var msg   = m.Groups["msg"].Value.Trim();

        // Витягуємо G-код із msg, якщо є
        var cmdMatch = Regex.Match(msg, @"\bG\d+\b");
        string? command = cmdMatch.Success ? cmdMatch.Value : null;

        // Допоміжний метод для парсингу числа з групи
        static double? ParseGroup(Group g) =>
```

```
            g.Success ? double.Parse(g.Value, CultureInfo.InvariantCulture) : (double?)null;

        return new RobotResponse
        {
            Timestamp = timestamp,
            Level     = level,
            Message   = msg,
            Command   = command,
            X         = ParseGroup(m.Groups["x"]),
            Y         = ParseGroup(m.Groups["y"]),
            Z         = ParseGroup(m.Groups["z"]),
            E         = ParseGroup(m.Groups["e"])
        };
    }

    /// <summary>
    /// Форматує об’єкт RobotResponse назад у рядок
    /// Включає лише ті частини, які не null
    /// </summary>
    public string ToLogLine()
    {
        var sb = new StringBuilder();
        sb.Append(Timestamp.ToString("HH:mm:ss.fff", CultureInfo.InvariantCulture))
            .Append(" -> ")
            .Append(Level)
            .Append(": ")
            .Append(Message);

        if (!string.IsNullOrEmpty(Command))
            sb.Append(" Command:").Append(Command);

        var parts = new List<string>();
        if (X.HasValue) parts.Add($"X:{X.Value:0.##}");
        if (Y.HasValue) parts.Add($"Y:{Y.Value:0.##}");
        if (Z.HasValue) parts.Add($"Z:{Z.Value:0.##}");
        if (E.HasValue) parts.Add($"E:{E.Value:0.##}");

        if (parts.Count > 0)
        {
            sb.Append(" [")
                .Append(string.Join(" ", parts))
                .Append("]");
        }

        return sb.ToString();
    }

    /// <summary>
    /// Спробувати розпарсити рядок. Якщо не відповідає шаблону — повернути false, не кидати виключення
    /// </summary>
    public static bool TryParse(string line, out RobotResponse? result)
```

```
        {
            var m = LineRegex.Match(line);
            if (!m.Success)
            {
                result = null;
                return false;
            }
            // Якщо успішно — скористатися Parse
            try
            {
                result = Parse(line);
                return true;
            }
            catch
            {
                result = null;
                return false;
            }
        }

    }
```

Класи модуля ШІ:

```
namespace AIManager;

public class AIManager
{
    private readonly MLContext _mlContext;
    struct InceptionSettings
    {
        public const int ImageHeight = 224;
        public const int ImageWidth = 224;
        public const float Mean = 117;
        public const float Scale = 1;
        public const bool ChannelsLast = true;
    }

    static string _assetsPath = Path.Combine(Environment.CurrentDirectory, "assets");
    static string _inceptionTensorFlowModel =
        Path.Combine(_assetsPath, "inception", "tensorflow_inception_graph.pb");

    public AIManager()
    {
        _mlContext = new MLContext();
    }
    public ModelOutput ClassifySingleImage(ITransformer model, string predictSingleImage)
    {
        var imageData = new ModelInput()
        {
            ImagePath = predictSingleImage
        };
```

```
        // Make prediction function (input = ImageData, output = ImagePrediction)
        PredictionEngine<ModelInput, ModelOutput> predictor = _mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(model);
        ModelOutput? prediction = predictor.Predict(imageData);

        return prediction;
    }

    public ITransformer LoadModel(string modelPathAndName, out DataViewSchema modelSchema)
    {
        // Load trained model
        return _mlContext.Model.Load(modelPathAndName, out modelSchema);
    }


    public ITransformer TrainModel(
        string imagesFolder,
        string trainTagsTsv,
        string testTagsTsv,
        string saveModelPath,
        out IEnumerable<ModelOutput> imagePredictionData,
        IProgress<int> progressReporter,
        IProgress<string> logReporter)
    {
        //побудова пайплайну
        logReporter.Report("Побудова пайплайну...");
        IEstimator<ITransformer> pipeline = BuildPipeline(imagesFolder);
        progressReporter.Report(5);

        //завантаження тренувальних даних
        logReporter.Report("Завантаження тренувальних даних...");
        IDataView trainingData = _mlContext.Data.LoadFromTextFile<ModelInput>(
            path: trainTagsTsv,
            hasHeader: false,
            separatorChar: ' ');
        progressReporter.Report(9);

        // тенування моделі
        logReporter.Report("Тренування моделі...");
        ITransformer newModel = pipeline.Fit(trainingData);
        progressReporter.Report(20);

        //Завантаження тестових даних
        logReporter.Report("Завантаження тестових даних...");
        IDataView testData = _mlContext.Data.LoadFromTextFile<ModelInput>(
            path: testTagsTsv,
            hasHeader: false,
            separatorChar: ' ');
        progressReporter.Report(5);
```

```
        // прогнозування для тестових даних
        logReporter.Report("Прогнозування тестових даних...");
        IDataView predictions = newModel.Transform(testData);
        imagePredictionData = _mlContext.Data.CreateEnumerable<ModelOutput>(predictions,
reuseRowObject: true);
        progressReporter.Report(10);

        //Обчислення метрик якості моделі
        logReporter.Report("Обчислення метрик якості моделі...");
        var metrics = _mlContext.MulticlassClassification.Evaluate(
            predictions,
            labelColumnName: "LabelKey",
            predictedLabelColumnName: "PredictedLabel");

        logReporter.Report($"LogLoss: {metrics.LogLoss:F4}");
        logReporter.Report("PerClassLogLoss: " + string.Join(" , ",
metrics.PerClassLogLoss.Select(x => x.ToString("F4"))));
        progressReporter.Report(3);

        // збереження моделі
        logReporter.Report("Збереження моделі...");
        string finalModelPath = Path.Combine(saveModelPath, "baseModel.zip");
        _mlContext.Model.Save(newModel, trainingData.Schema, finalModelPath);
        progressReporter.Report(5);

        return newModel;
    }

    private IEstimator<ITransformer> BuildPipeline(string imagesFolder)
    {
        return _mlContext.Transforms.LoadImages(
                outputColumnName: "input",
                imageFolder: imagesFolder,
                inputColumnName: nameof(ModelInput.ImagePath))
            .Append(_mlContext.Transforms.ResizeImages(
                outputColumnName: "input",
                imageWidth: InceptionSettings.ImageWidth,
                imageHeight: InceptionSettings.ImageHeight,
                inputColumnName: "input"))
            .Append(_mlContext.Transforms.ExtractPixels(
                outputColumnName: "input",
                interleavePixelColors: InceptionSettings.ChannelsLast,
                offsetImage: InceptionSettings.Mean))
            .Append(_mlContext.Model.LoadTensorFlowModel(_inceptionTensorFlowModel)
                .ScoreTensorFlowModel(
                    outputColumnNames: new[] { "softmax2_pre_activation" },
                    inputColumnNames: new[] { "input" },
                    addBatchDimensionInput: true))
            .Append(_mlContext.Transforms.Conversion.MapValueToKey(
                outputColumnName: "LabelKey",
                inputColumnName: "Label"))
            .Append(_mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy(
```

```
            labelColumnName: "LabelKey",
            featureColumnName: "softmax2_pre_activation"))
        .Append(_mlContext.Transforms.Conversion.MapKeyToValue(
            "PredictedLabelValue",
            "PredictedLabel"))
        .AppendCacheCheckpoint(_mlContext);
  }
}


namespace AIManager;

public class ModelInput
{
  [LoadColumn(0)]
  public string? ImagePath;

  [LoadColumn(1)]
  public string? Label;
}


namespace AIManager;

public class ModelOutput : ModelInput
{
      public float[]? Score;
      public string? PredictedLabelValue;
}
```

Класи модуля конфігурації:

```
namespace Infrastructure;

public class ConfigService
{
  private readonly string _filePath;
  public RoboOptixConfig? Config { get; private set; }
  public string? ErrorMessage { get; private set; }

  public ConfigService(string filePath)
  {
    _filePath = filePath;

    try
    {
      Config = XmlConfigReader.LoadConfig(_filePath);
    }
    catch (Exception ex)
    {
      Config = null;
      ErrorMessage = ex.Message;
```

```
        }
    }

    public bool IsValid => Config != null; // Чи успішно завантажено конфіг

    /// <summary>
    /// Додає новий об’єкт до конфігурації
    /// Перед додаванням перевіряє, що об’єкт з таким Label ще не існує
    /// </summary>
    /// <param name="newObject">Об’єкт для додавання</param>
    /// <returns>true, якщо додавання успішне; false в іншому випадку</returns>
    public bool AddObject(ObjectConfig newObject)
    {
        if (Config == null)
        {
            ErrorMessage = "Конфігурація не завантажена.";
            return false;
        }

        //перевірка що Label ще не використовується (порівняння без урахування регістру)
        if (Config.Objects.Any(o => o.Label.Equals(newObject.Label,
StringComparison.OrdinalIgnoreCase)))
        {
            ErrorMessage = "Об’єкт з таким Label вже існує.";
            return false;
        }

        Config.Objects.Add(newObject);
        SaveConfig();
        return true;
    }

    /// <summary>
    /// Редагування існуючого об’єкту конфігурації, використовуючи оновлені дані
    /// Перевірка, що новий Label не дублюється серед інших об’єктів
    /// </summary>
    /// <param name="updatedObject">Об’єкт з оновленими даними (Id має збігатися з
існуючим)</param>
    /// <returns>true, якщо редагування успішне; false в іншому випадку</returns>
    public bool UpdateObject(ObjectConfig updatedObject)
    {
        if (Config == null)
        {
            ErrorMessage = "Конфігурація не завантажена.";
            return false;
        }

        //пошук існуючого об’єкту за Id
        var existingObj = Config.Objects.FirstOrDefault(o => o.Id == updatedObject.Id);
        if (existingObj == null)
        {
            ErrorMessage = $"Об’єкт з Id '{updatedObject.Id}' не знайдено.";
```

```
            return false;
        }

        // чи новий Label не дублюється серед інших, крім самого об’єкта
        if (Config.Objects.Any(o => o.Id != updatedObject.Id &&
                               o.Label.Equals(updatedObject.Label,
StringComparison.OrdinalIgnoreCase)))
        {
            ErrorMessage = "Інший об’єкт з таким Label вже існує.";
            return false;
        }

        // Оновлення властивості
        existingObj.Label = updatedObject.Label;
        existingObj.Path = updatedObject.Path;
        existingObj.Status = updatedObject.Status;

        SaveConfig();
        return true;
    }

    /// <summary>
    /// Видаляє об’єкт з конфігурації за його Id
    /// </summary>
    /// <param name="objectId">Ідентифікатор об’єкта для видалення</param>
    /// <returns>true, якщо видалення успішне; false, якщо об’єкт не знайдено або виникла
помилка</returns>
    public bool DeleteObject(Guid objectId)
    {
        if (Config == null)
        {
            ErrorMessage = "Конфігурація не завантажена.";
            return false;
        }

        var obj = Config.Objects.FirstOrDefault(o => o.Id == objectId);
        if (obj == null)
        {
            ErrorMessage = $"Об’єкт з Id '{objectId}' не знайдено.";
            return false;
        }

        Config.Objects.Remove(obj);
        SaveConfig();
        return true;
    }

    /// <summary>
    /// Змінює статус об'єктів у конфігу
    /// </summary>
    public void UpdateObjectStatus(Guid objectId, string newStatus)
    {
```

```
        if (Config?.Objects == null || Config.Objects.Count == 0)
        {
            Logger.WriteToLogs("Config Service Помилка: Об'єкти відсутні у конфігурації.");
            return;
        }

        // пошук об'єкту із заданим Id
        var obj = Config.Objects.FirstOrDefault(o => o.Id == objectId);

        if (obj == null)
        {
            Logger.WriteToLogs($"Config Service Об'єкт із ID '{objectId}' не знайдено.");
            return;
        }

        Logger.WriteToLogs($"Config Service Оновлення статусу: {obj.Label} (ID: {obj.Id})
з {obj.Status} на {newStatus}");
        obj.Status = newStatus;

        // збереження оновленої конфігурації
        SaveConfig();
    }

    /// <summary>
    /// Зберігає поточну конфігурацію у XML-файл
    /// </summary>
    public void SaveConfig()
    {
        if (Config == null)
        {
            Logger.WriteToLogs("Config Service Помилка: Немає даних для збереження.");
            return;
        }

        try
        {
            XmlSerializer serializer = new(typeof(RoboOptixConfig));
            using FileStream fileStream = new(_filePath, FileMode.Create);
            serializer.Serialize(fileStream, Config);
            Logger.WriteToLogs("Конфігурація успішно збережена!");
        }
        catch (Exception ex)
        {
            Logger.WriteToLogs($"Config Service Помилка збереження конфігурації:
{ex.Message}");
        }
    }

}


namespace Infrastructure;
```

```
public class XmlConfigReader
{
   public static RoboOptixConfig LoadConfig(string filePath)
   {
      if (!File.Exists(filePath))
         throw new FileNotFoundException($"Файл не знайдено: {filePath}");

      try
      {
         XmlSerializer serializer = new(typeof(RoboOptixConfig));
         using FileStream fileStream = new(filePath, FileMode.Open);
         return (RoboOptixConfig)serializer.Deserialize(fileStream) ?? new
RoboOptixConfig();
      }
      catch (Exception ex)
      {
         throw new Exception($"Помилка читання XML: {ex.Message}", ex);
      }
   }
}
```

Клас головного вікна програми:

```
namespace RoboOptix;

public partial class MainWindow : Window, INotifyPropertyChanged
{
   // Поля для зберігання станів
   private bool _isCameraConnected = false;
   private bool _isRobotConnected = false;

   private bool _robotSubscribed = false;
   // Флаг, що вказує, чи потрібно зупинити сортування
   private bool _cancelSorting = false;
   // поле для властивості CanStart
   private bool _canStart = false;
   // поле для властивості CanStop
   private bool _canStop = false;
   // Допоміжне поле для запобігання одночасному виклику сортування
   private bool _isSortingInProgress = false;
   // Приватне поле для роботи з процесором комп'ютерного зору
   private readonly ComputerVisionProcessor _visionProcessor;
   // Приватне поле для роботи з класом для визначення маркерів
   private readonly ArUcoObjectMeasurementProcessor _arucoProcessor;
   //Приватне поле для роботи з класом ШІ
   private AIManager.AIManager _aiManager;
   //Приватне поле для роботи з класом конфігурації
   private RoboOptixConfig? _config = App.ConfigService.Config;
   // URL відеопотоку
   private string _ipCameraUrl { get; set; }
```

```
    // ініціалізація списку знайдених на фото об’єктів
    public ObservableCollection<ObjectMeasurement> FoundObjects { get; set; } = new ObservableCollection<ObjectMeasurement>();
    // ця колекція міститиме по одному об’єкту з кожною класифікацією (щоб користувач міг призначати класифікаціям сортувальні бокси)
    public ObservableCollection<ObjectMeasurement> UniqueClassifications { get; set; }
        = new ObservableCollection<ObjectMeasurement>();
    // список СБ
    public ObservableCollection<SortBoxModel> SortBoxes { get; set; } = new ObservableCollection<SortBoxModel>();

    // Властивість для кнопки Пачати
    public bool CanStart
    {
        get => _canStart;
        private set
        {
            if (_canStart != value)
            {
                _canStart = value;
                OnPropertyChanged(nameof(CanStart));
            }
        }
    }

    // Властивість для кнопки Зупинити
    public bool CanStop
    {
        get => _canStop;
        private set
        {
            if (_canStop != value)
            {
                _canStop = value;
                OnPropertyChanged(nameof(CanStop));
            }
        }
    }

    private bool _isVideoStreamOn;
    public bool IsVideoStreamOn
    {
        get => _isVideoStreamOn;
        set
        {
            if (_isVideoStreamOn != value)
            {
                _isVideoStreamOn = value;
                OnPropertyChanged(nameof(IsVideoStreamOn));
                OnPropertyChanged(nameof(IsVideoStreamOff));
            }
        }
```

```
}

private bool _isClacifyImageOff = true;
public bool IsClacifyImageOff
{
    get => _isClacifyImageOff;
    set
    {
        if (_isClacifyImageOff != value)
        {
            _isClacifyImageOff = value;
            OnPropertyChanged(nameof(IsClacifyImageOff));
        }
    }
}
public bool IsVideoStreamOff
    => !IsVideoStreamOn;
public event PropertyChangedEventHandler? PropertyChanged;
protected void OnPropertyChanged(string propertyName)
    => PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

private ITransformer _model { get; set; }

// ширина столу (з конфігурації)
private float _tableWidth { get; set; }
// висота столу з конфігурації
private float _tableHieght { get; set; }
public string RobotPortDisplay
{
    get
    {
        if (App.Robot != null && App.Robot.IsConnected)
            return App.Robot.PortName;
        return "Не підключено";
    }
}

public string CameraUrlDisplay
{
    get
    {
        if (_isCameraConnected)
            return _config.CameraIP;
        return "Не підключено";
    }
}

public MainWindow()
{
    InitializeComponent();
    _ipCameraUrl = $"{_config.CameraIP}/jpg";
```

```
        _visionProcessor = new ComputerVisionProcessor(_ipCameraUrl);
        _arucoProcessor = new ArUcoObjectMeasurementProcessor();
        _aiManager = new AIManager.AIManager();

        _tableWidth = _config.Table.Width;
        _tableHieght = _config.Table.Height;

        // Підписка на подію оновлення статусу івдеопотоку
        // Підписка на зміну статусу камери
        _visionProcessor.StatusChanged += status =>
        {
            // Оскільки цей код може виконуватись не в UI-потоці,
            // використовується диспетчер для оновлення UI
            Dispatcher.UIThread.InvokeAsync(() =>
            {
                Logger.WriteToLogs($"{DateTime.Now} Статус камери: {status}");
                if (status == "Підключено")
                {
                    StartVideoStream();
                    CameraStatusEllipse.Fill = Brushes.ForestGreen;
                    _isCameraConnected = true;
                }
                else
                {
                    CameraStatusEllipse.Fill = Brushes.Crimson;
                    _isCameraConnected = false;
                }
                OnPropertyChanged(nameof(CameraUrlDisplay));
                UpdateCanStartStop();
            });
        };
        // Підписка на зміну статусу робота
        if (App.Robot != null)
        {
            _isRobotConnected = App.Robot.IsConnected;
            RobotStatusEllipse.Fill = _isRobotConnected ? Brushes.ForestGreen :
Brushes.Crimson;

            App.Robot.StatusChanged += robotStatus =>
            {
                Dispatcher.UIThread.InvokeAsync(() =>
                {
                    Logger.WriteToLogs($"{DateTime.Now} Статус робота: {robotStatus}");
                    if (robotStatus == "Підключено")
                    {
                        _isRobotConnected = true;
                        RobotStatusEllipse.Fill = Brushes.ForestGreen;
                    }
                    else
                    {
                        _isRobotConnected = false;
                        RobotStatusEllipse.Fill = Brushes.Crimson;
```

```
            }
            OnPropertyChanged(nameof(RobotPortDisplay));
            UpdateCanStartStop();
        });
    };
}
// Підписка на зміну колекції FoundObjects
FoundObjects.CollectionChanged += FoundObjectsCollectionChanged;
foreach (var item in FoundObjects)
{
    item.PropertyChanged += ObjectMeasurementPropertyChanged;
}
_visionProcessor.VideoRunningChanged += status =>
{
    Dispatcher.UIThread.InvokeAsync(() =>
    {
        IsVideoStreamOn = status;
        if(IsVideoStreamOn)
            CameraStatusEllipse.Fill = Brushes.ForestGreen;
        else
            CameraStatusEllipse.Fill = Brushes.Crimson;
    });
};
// Підписка на подію помилки відеопотоку
_visionProcessor.VideoStreamError += ex =>
{
    Dispatcher.UIThread.InvokeAsync(() =>
    {
        // виведення помилки користувачеві
        Logger.WriteToLogs($"{DateTime.Now:HH:mm:ss} Video stream error:
{ex.Message}");
    });
};
_visionProcessor.InitializeVideo();

// Спроба підписатися на App.Robot, якщо він уже є
SubscribeToRobotStatus();

// встановлення CanStart/CanStop відповідно до початкових флагів
UpdateCanStartStop();

Logger.WriteToLogs($"{DateTime.Now} Перевірка з’єднання з камерою...
{_ipCameraUrl}");
// Підписка на подію відкриття вікна Avalonia
this.Opened += async (_, __) => await CheckCameraConnectionAsync();

// Завантаження моделі ІШІ
DataViewSchema dataViewSchema;
_model = _aiManager.LoadModel(
    Path.Combine(_config.AI.ModelsPath, $"{_config.AI.Model.Name}.zip"),
    out dataViewSchema);
```

```
        // заповнення SortBoxes
        LoadSortBoxesFromConfig();
        foreach (var box in SortBoxes)
        {
            Logger.WriteToLogs($"{DateTime.Now}: Boxes from XML – {box.Name} ({box.X}, {box.Y}, {box.Z})");
        }

        DataContext = this;
    }
    /// <summary>
    /// Якщо App.Robot не null і ще немає підписки, то підписуємося на StatusChanged
    /// </summary>
    public void SubscribeToRobotStatus()
    {
        if (_robotSubscribed)
            return;

        if (App.Robot != null)
        {
            _robotSubscribed = true;

            // Встановлюємо початковий стан еліпса
            _isRobotConnected = App.Robot.IsConnected;
            RobotStatusEllipse.Fill = _isRobotConnected ? Brushes.ForestGreen : Brushes.Crimson;
            OnPropertyChanged(nameof(RobotPortDisplay));

            App.Robot.StatusChanged += robotStatus =>
            {
                Dispatcher.UIThread.InvokeAsync(() =>
                {
                    Logger.WriteToLogs($"{DateTime.Now} Статус робота: {robotStatus}");
                    if (robotStatus == "Підключено")
                    {
                        _isRobotConnected = true;
                        RobotStatusEllipse.Fill = Brushes.ForestGreen;
                    }
                    else
                    {
                        _isRobotConnected = false;
                        RobotStatusEllipse.Fill = Brushes.Crimson;
                    }
                    OnPropertyChanged(nameof(RobotPortDisplay));
                    UpdateCanStartStop();
                });
            };
        }
    }
    // Якщо до FoundObjects додаються/видаляються елементи
    private void FoundObjectsCollectionChanged(object? sender, NotifyCollectionChangedEventArgs e)
```

```
    {
        if (e.NewItems != null)
        {
            foreach (ObjectMeasurement newObj in e.NewItems!)
                newObj.PropertyChanged += ObjectMeasurementPropertyChanged;
        }
        if (e.OldItems != null)
        {
            foreach (ObjectMeasurement oldObj in e.OldItems!)
                oldObj.PropertyChanged -= ObjectMeasurementPropertyChanged;
        }

        UpdateCanStartStop();
    }
    // Якщо у будь-якого ObjectMeasurement змінилася якась властивість
    private void ObjectMeasurementPropertyChanged(object? sender,
PropertyChangedEventArgs e)
    {
        if (e.PropertyName == nameof(ObjectMeasurement.SortBox))
        {
            UpdateCanStartStop();
        }
    }
    /// <summary>
    /// Перевіряє три умови:
    /// 1) робот підключений (_isRobotConnected)
    /// 2) камера підключена (_isCameraConnected)
    /// 3) у FoundObjects є хоча б один елемент і в кожного призначений SortBox != null
    /// Якщо всі виконуються, CanStart = true, інакше false
    /// Крім того, CanStop = _isRobotConnected
    /// </summary>
    private void UpdateCanStartStop()
    {
        bool haveObjectsWithBoxes = FoundObjects.Count > 0
                                    && FoundObjects.All(o => o.SortBox != null);

        CanStart = _isCameraConnected
                   && _isRobotConnected
                   && haveObjectsWithBoxes;

        CanStop = _isRobotConnected;
    }
    private void LoadSortBoxesFromConfig()
    {
        // Для кожного BoxConfig з XML у App.ConfigService.Config.Boxes
        // створюємо SortBox і додаємо у колекцію
        foreach (var boxCfg in _config.Boxes)
        {
            SortBoxes.Add(new SortBoxModel(boxCfg));
        }
    }
    private void ToogleMenuBtnClick(object? sender, RoutedEventArgs e)
```

```
{
    // Змінюємо IsPaneOpen у MySplitView
    MenuSplitView.IsPaneOpen = !MenuSplitView.IsPaneOpen;

    ToggleMenuBtn.Content = MenuSplitView.IsPaneOpen ? ">" : "<";
}

/// <summary>
/// Обробник для запуску відеопотоку
/// Запуск відеопотоку здійснюється "fire-and-forget"
/// </summary>
private void StartVideoStream()
{
    try
    {
        // Запуск відеопотоку
        _ = _visionProcessor.StartVideoStreamAsync(OnFrameReceived);
        VideoFrame.IsVisible = _visionProcessor.VideoRunningStatus;
    }
    catch (Exception ex)
    {
        Logger.WriteToLogs($"{DateTime.Now} {ex.Message}");
    }
}

/// <summary>
/// Делегат, який викликається при отриманні нового кадру
/// Цей метод перетворює отриманий кадр у Mat,
/// виконує асинхронну детекцію та аннотацію маркерів
/// і потім оновлює Image VideoFrame з аннотованим зображенням
/// </summary>
/// <param name="frame">Отриманий кадр у форматі Bitmap</param>
private void OnFrameReceived(Bitmap frame)
{
    // Оновлення UI на головному потоці
    Dispatcher.UIThread.Post(() =>
    {
        byte[] frameBytes = ConvertBitmapToByteArray(frame);
        // Обробка зображення для визначення маркерів
        (Mat annotatedMat, _) = _arucoProcessor.ProcessImage(frameBytes);

        // конвертація аннотованого Mat у Avalonia Bitmap
        Bitmap annotatedBitmap = ConvertMatToAvaloniaBitmap(annotatedMat);

        VideoFrame.Source = annotatedBitmap;
    });
}

public static byte[] ConvertBitmapToByteArray(Bitmap bitmap)
{
    if (bitmap == null)
        throw new ArgumentNullException(nameof(bitmap));
```

```
        using (var ms = new MemoryStream())
        {
            // збереження зображення у MemoryStream
            bitmap.Save(ms);
            return ms.ToArray();
        }
    }

    /// <summary>
    /// Конвертує OpenCvSharp.Mat у Avalonia.Media.Imaging.Bitmap
    /// Використовується MemoryStream і запис формату BMP
    /// </summary>
    private Bitmap ConvertMatToAvaloniaBitmap(Mat mat)
    {
        using MemoryStream ms = new MemoryStream();
        // Запис Mat у потік у форматі BMP
        mat.WriteToStream(ms, ".bmp");
        ms.Position = 0;
        return new Bitmap(ms);
    }

    // Обробник для перевірки з'єднання з камерою
    private async void CheckCameraConnection_Click(object? sender, RoutedEventArgs e)
        => await CheckCameraConnectionAsync();

    // приватний метод перевірки з’єднання з камерою
    private async Task CheckCameraConnectionAsync()
    {
        try
        {
            _isCameraConnected = await _visionProcessor.CheckCameraConnectionAsync();
        }
        catch (Exception ex)
        {
            Logger.WriteToLogs($"{DateTime.Now} {ex.Message}");
        }
    }

    /// <summary>
    /// Обробник натискання кнопки для класифікації зображення
    /// Захоплюється кадр без малювання маркерів, зберігається у тимчасовий файл,
    /// після чого запускається метод ProcessImage для визначення об’єктів
    /// Результат – аннотоване зображення та інформація про об'єкти, відображаються в
інтерфейсі
    /// </summary>
    private async void ClassifySingleImageOnClick(object? sender, RoutedEventArgs e)
    {
        FoundObjects.Clear();
        ObjectsTable.ItemsSource = FoundObjects;
        IsClacifyImageOff = false;
```

```
        try
        {
            // захоплення кадру без додаткової обробки
            byte[] imageBytes = await _visionProcessor.CaptureFrameBytesAsync();

            // створення Avalonia Bitmap із отриманих байтів
            Bitmap capturedImage;
            using (var ms = new MemoryStream(imageBytes))
            {
                capturedImage = new Bitmap(ms);
            }

            SelectedImage.Source = capturedImage;

            // Обробка зображення для визначення маркерів
            (List<ObjectMeasurement> measurements, Mat annotatedMat, int markersCount) =
_arucoProcessor.Image(imageBytes);

            // сортування за ContourArea від найменшого до найбільшого
            measurements = measurements
                .OrderBy(m => m.ContourArea)
                .ToList();

            // Відображення інформації про об’єкти
            string objectsInfo = "Знайдені об'єкти:\n";
            foreach (var m in measurements)
            {
                objectsInfo +=
                    $"Об'єкт #{m.ObjectId}: Центр=({m.CenterX:F2}; {m.CenterY:F2}),
Площа={m.ContourArea:F0}\n";
                // Розміри={m.Width:F2}x{m.Height:F2}
                Logger.WriteToLogs(objectsInfo);
                FoundObjects.Add(m);
            }

            //// обробка вирізаних об’єктів ШІ
            foreach (var obj in measurements)
            {
                string objFile = Path.Combine(Environment.CurrentDirectory, "CroppedObjects",
$"object_{obj.ObjectId}.png");
                ModelOutput modelOutput;
                if (File.Exists(objFile))
                {
                    // класифікація асинхронно, щоб не блокувати UI
                    modelOutput = await Task.Run(() => _aiManager.ClassifySingleImage(_model,
objFile));

                    // Створення Bitmap з файлу та установка джерела зображення
                    var bitmap = new Bitmap(objFile);
                    SelectedImage.Source = bitmap;

                    // Запис результатів
```

```
                obj.Classification = modelOutput.PredictedLabelValue;
                // Округлення максимального значення до двох знаків після коми
                double rawConfidence = modelOutput.Score?.Max() ?? 0.0;
                obj.Confidence = Math.Round(rawConfidence * 100, 2);

                // перевірка впевненості з мінімально прохідною
                if (modelOutput.Score?.Max() * 100 <=
_config.AI.Model.ConfidenceThreshold)
                {
                  //кящо недостатня впевненість, то одразу призначажмо сортувальний
бокс по дефолту
                  obj.SortBox = SortBoxes.Where(b => b.Name == "Default").FirstOrDefault();
                }

                // затримка, щоб користувач встиг подивитись фото
                await Task.Delay(2000);
              }
            }

            ObjectsTable.ItemsSource = measurements;
            UpdateUniqueClassifications();

            // Конвертуємо аннотований Mat у Avalonia Bitmap
            Bitmap annotatedBitmap = ConvertMatToAvaloniaBitmap(annotatedMat);
            SelectedImage.Source = annotatedBitmap;
        }
        catch (InvalidOperationException ex)
        {
            Logger.WriteToLogs($"{DateTime.Now} {ex.Message}");
            IsClacifyImageOff = true;
        }
        catch (Exception ex)
        {
            Logger.WriteToLogs($"{DateTime.Now} {ex.Message}");
            IsClacifyImageOff = false;
        }
    }
    private void UpdateUniqueClassifications()
    {
        UniqueClassifications.Clear();

        // отримання усіх класифікацій, що зустрілися в FoundObjects
        var distinctNames = FoundObjects
            .Select(o => o.Classification)
            .Distinct(StringComparer.OrdinalIgnoreCase);

        // для кожного унікального імені створення “шаблонного” ObjectMeasurement
        // із заповненим тільки Classification, решта залишиться дефолтною
        foreach (var name in distinctNames)
        {
            var om = new ObjectMeasurement
            {
```

```
            Classification = name,
            SortBox = null
        };
        UniqueClassifications.Add(om);
    }
}
protected override void OnClosed(EventArgs e)
{
    base.OnClosed(e);
    _visionProcessor.StopVideoStream();
    _visionProcessor.Dispose();
}

private async void StartSortingProcess(object? sender, RoutedEventArgs e)
{
    // блокування кнопки Start
    CanStart = false;
    // Скидання флагу скасування (на випадок, якщо раніше натискали Stop)
    _cancelSorting = false;

    try
    {
        foreach (var obj in FoundObjects)
        {
            // Якщо Stop натиснуто — вийти з циклу
            if (_cancelSorting)
                break;

            Logger.WriteToLogs($"{obj.ObjectId}  {obj.Classification}  {obj.SortBox}
{obj.CenterX} {obj.CenterY}");
            App.Robot?.MoveLinear(
                (obj.CenterX * 10 - (_tableWidth * 10) / 2),
                obj.CenterY * 10 - (_tableHieght * 10) / 2 + 174,
                -10
            );
            // wait 2 second
            await Task.Delay(TimeSpan.FromSeconds(2));

            App.Robot?.MoveLinear(
                (obj.CenterX * 10 - (_tableWidth * 10) / 2),
                obj.CenterY * 10 - (_tableHieght * 10) / 2 + 174,
                -60
            );
            // wait 2 second
            await Task.Delay(TimeSpan.FromSeconds(2));

            App.Robot?.GrabObject();
            // wait 1 sec
            await Task.Delay(TimeSpan.FromSeconds(1));

            // перенесення об’єкту
             App.Robot?.MoveLinear(obj.SortBox.X, obj.SortBox.Y, 100);
```

```
            // wait 3 second
            await Task.Delay(TimeSpan.FromSeconds(2));

            // опускання об’єкту
             App.Robot?.MoveLinear(obj.SortBox.X, obj.SortBox.Y, obj.SortBox.Z);
            // wait 3 second
            await Task.Delay(TimeSpan.FromSeconds(2));

            App.Robot?.ReleaseObject();
            // wait 2 sec
            await Task.Delay(TimeSpan.FromSeconds(2));

            App.Robot?.MoveLinear(0, 174, 120);
            App.Robot?.GrabObject();
            App.Robot?.GrabObject();
            App.Robot?.ReleaseObject();
        }
    }
    catch (FormatException exception)
    {
        Console.WriteLine(exception);
    }
    catch(Exception exception)
    {
        Logger.WriteToLogs($"{DateTime.Now} {exception.Message}");
    }
    finally
    {
        // Після завершення або переривання – знову дозволити Start/заблокувати Stop
        _cancelSorting = false;
        UpdateCanStartStop();
    }
}

private void DoneBtnClick(object? sender, RoutedEventArgs e)
{
    foreach (var unique in UniqueClassifications)
    {
        // ім’я класифікації та обраний бокс
        var clsName = unique.Classification;
        var chosenBox = unique.SortBox;

        // Перебір усіх реальних об’єктів, у яких Classification == clsName
        var matchingObjects = FoundObjects
            .Where(o => string.Equals(o.Classification, clsName,
StringComparison.OrdinalIgnoreCase));

        foreach (var obj in matchingObjects)
        {
            obj.SortBox = chosenBox;
        }
    }
```

```
}

private void ObjectManageMenuBtnClick(object? sender, RoutedEventArgs e)
{
   ObjectManagerView objectManageWindow = new ObjectManagerView();
   objectManageWindow.ShowDialog(this);
}

private void ArmManageMenuBtnClick(object? sender, RoutedEventArgs e)
{
   ArmManage armManageWindow = new ArmManage();
   armManageWindow.ShowDialog(this);
}

private void AIManageMenuBtnClick(object? sender, RoutedEventArgs e)
{
   AIManage aiManageWindow = new AIManage();
   aiManageWindow.ShowDialog(this);
}

private async void ObjectsTableDoubleTapped(object? sender, TappedEventArgs e)
{
   if (ObjectsTable.SelectedItem is not ObjectMeasurement selectedObj)
      return;

   // зберегти старі значення до того, як відкриватимемо вікно
   var oldCenterX       = selectedObj.CenterX;
   var oldCenterY       = selectedObj.CenterY;
   var oldClassification = selectedObj.Classification;
   var oldSortBox       = selectedObj.SortBox;
   var oldConfidence    = selectedObj.Confidence;

   // список усіх можливих класифікацій
   var config = App.ConfigService?.Config;
   var classificationOptions = config == null
      ? new List<string>()
      : config.Objects.Select(o => o.Label).Distinct().ToList();

   // модальне вікно редагування
   var detailsWindow = new ObjectDetailsWindow(
      selectedObj,
      new ObservableCollection<string>(classificationOptions),
      SortBoxes);

   var result = await detailsWindow.ShowDialog<bool>(this);
   // якщо користувач натиснув “Зберегти”
   if (result)
   {
      // порівняння нових значеннь з тими, що були
      var sb = new System.Text.StringBuilder();
      bool anyChange = false;
```

```
            if (!oldCenterX.Equals(selectedObj.CenterX))
            {
                sb.AppendLine($"CenterX: {oldCenterX} → {selectedObj.CenterX}");
                anyChange = true;
            }

            if (!oldCenterY.Equals(selectedObj.CenterY))
            {
                sb.AppendLine($"CenterY: {oldCenterY} → {selectedObj.CenterY}");
                anyChange = true;
            }

            if (!string.Equals(oldClassification, selectedObj.Classification,
StringComparison.Ordinal))
            {
                sb.AppendLine($"Classification: \"{oldClassification}\" →
\"{selectedObj.Classification}\"");
                anyChange = true;
            }

            if (oldSortBox?.Name != selectedObj.SortBox?.Name)
            {
                var oldBoxName = oldSortBox?.Name ?? "(null)";
                var newBoxName = selectedObj.SortBox?.Name ?? "(null)";
                sb.AppendLine($"SortBox: \"{oldBoxName}\" → \"{newBoxName}\"");
                anyChange = true;
            }

            if (anyChange)
            {
                // виведення всього одним логом або декількома рядками
                Logger.WriteToLogs($"Змінено поля для
ObjectId={selectedObj.ObjectId}:\n{sb}");
                // оновлення списку класифікацій
                UpdateUniqueClassifications();
            }
            else
            {
                Logger.WriteToLogs($"ObjectId={selectedObj.ObjectId} переглянуто детально,
але змін не внесено");
            }
        }
    }

    private void StopSortingProcess(object? sender, RoutedEventArgs e)
    {
        // зупинення сортування
        _cancelSorting = true;
        // поновлення CanStart через UpdateCanStartStop()
        UpdateCanStartStop();
        // зупинка відеопотоку
```

```
        _visionProcessor.StopVideoStream();
    }
}
```

Код прошивки камери:

```
#include "OV2640.h"
#include <WiFi.h>
#include <WebServer.h>
#include <WiFiClient.h>

#include "SimStreamer.h"/Users/rina/Documents/diplom/Micro-RTSP-master/examples/ESP32-devcam/wifikeys_template.h
#include "OV2640Streamer.h"/Users/rina/Documents/diplom/Micro-RTSP-master/examples/wifikeys_template.h
#include "CRtspSession.h"/Users/rina/Documents/diplom/Micro-RTSP-master/src/OV2640.cpp

// #define ENABLE_OLED //if want use oled ,turn on thi macro
// #define SOFTAP_MODE // If you want to run our own softap turn this on
#define ENABLE_WEBSERVER
// #define ENABLE_RTSPSERVER

#ifdef ENABLE_OLED
#include "SSD1306.h"
#define OLED_ADDRESS 0x3c
#define I2C_SDA 14
#define I2C_SCL 13
SSD1306Wire display(OLED_ADDRESS, I2C_SDA, I2C_SCL, GEOMETRY_128_32);
bool hasDisplay; // we probe for the device at runtime
#endif

OV2640 cam;

#ifdef ENABLE_WEBSERVER
WebServer server(80);
#endif

#ifdef ENABLE_RTSPSERVER
WiFiServer rtspServer(8554);
#endif


#ifdef SOFTAP_MODE
IPAddress apIP = IPAddress(192, 168, 1, 1);
#else
#include "wifikeys.h"
#endif

#ifdef ENABLE_WEBSERVER
void handle_jpg_stream(void)
```

```
{
   WiFiClient client = server.client();
   String response = "HTTP/1.1 200 OK\r\n";
   response += "Content-Type: multipart/x-mixed-replace; boundary=frame\r\n\r\n";
   server.sendContent(response);

   while (1)
   {
      cam.run();
      if (!client.connected())
         break;
      response = "--frame\r\n";
      response += "Content-Type: image/jpeg\r\n\r\n";
      server.sendContent(response);

      client.write((char *)cam.getfb(), cam.getSize());
      server.sendContent("\r\n");
      if (!client.connected())
         break;
   }
}

void handle_jpg(void)
{
   WiFiClient client = server.client();

   cam.run();
   if (!client.connected())
   {
      return;
   }
   String response = "HTTP/1.1 200 OK\r\n";
   response += "Content-disposition: inline; filename=capture.jpg\r\n";
   response += "Content-type: image/jpeg\r\n\r\n";
   server.sendContent(response);
   client.write((char *)cam.getfb(), cam.getSize());
}

void handleNotFound()
{
   String message = "Server is running!\n\n";
   message += "URI: ";
   message += server.uri();
   message += "\nMethod: ";
   message += (server.method() == HTTP_GET) ? "GET" : "POST";
   message += "\nArguments: ";
   message += server.args();
   message += "\n";
   server.send(200, "text/plain", message);
}
#endif
```

```
#ifdef ENABLE_OLED
#define LCD_MESSAGE(msg) lcdMessage(msg)
#else
#define LCD_MESSAGE(msg)
#endif

#ifdef ENABLE_OLED
void lcdMessage(String msg)
{
   if(hasDisplay) {
      display.clear();
      display.drawString(128 / 2, 32 / 2, msg);
      display.display();
   }
}
#endif

CStreamer *streamer;

void handle_jpg_stream_mjpeg() {
  WiFiClient client = server.client();
  String response = "HTTP/1.1 200 OK\r\n";
  response += "Content-Type: multipart/x-mixed-replace; boundary=frame\r\n\r\n";
  server.sendContent(response);

  while (client.connected()) {
    camera_fb_t *fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Каптур кадру не вдалося");
      break;
    }

    String partHeader = "--frame\r\n";
    partHeader += "Content-Type: image/jpeg\r\n";
    partHeader += "Content-Length: " + String(fb->len) + "\r\n\r\n";
    server.sendContent(partHeader);
    client.write(fb->buf, fb->len);
    server.sendContent("\r\n");

    esp_camera_fb_return(fb);

    // Невелика затримка між кадрами
    delay(30);
  }
}


void setup()
{
  #ifdef ENABLE_OLED
    hasDisplay = display.init();
    if(hasDisplay) {
```

```
      display.flipScreenVertically();
      display.setFont(ArialMT_Plain_16);
      display.setTextAlignment(TEXT_ALIGN_CENTER);
    }
  #endif
    LCD_MESSAGE("booting");

    Serial.begin(115200);
    while (!Serial)
    {
      ;
    }
    cam.init(esp32cam_aithinker_config);

    IPAddress ip;


#ifdef SOFTAP_MODE
    const char *hostname = "devcam";
    // WiFi.hostname(hostname); // FIXME - find out why undefined
    LCD_MESSAGE("starting softAP");
    WiFi.mode(WIFI_AP);
    WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
    bool result = WiFi.softAP(hostname, "12345678", 1, 0);
    if (!result)
    {
      Serial.println("AP Config failed.");
      return;
    }
    else
    {
      Serial.println("AP Config Success.");
      Serial.print("AP MAC: ");
      Serial.println(WiFi.softAPmacAddress());

      ip = WiFi.softAPIP();
    }
#else
    LCD_MESSAGE(String("join ") + ssid);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED)
    {
      delay(500);
      Serial.print(F("."));
    }
    ip = WiFi.localIP();
    Serial.println(F("WiFi connected"));
    Serial.println("");
    Serial.println(ip);
#endif
```

```
  LCD_MESSAGE(ip.toString());

#ifdef ENABLE_WEBSERVER
  // Додаємо новий endpoint для MJPEG-стріму
  server.on("/mjpeg", HTTP_GET, handle_jpg_stream_mjpeg);
  server.on("/", HTTP_GET, handle_jpg_stream);
  server.on("/jpg", HTTP_GET, handle_jpg);
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.print("Station MAC: ");
  Serial.println(WiFi.macAddress());
#endif

#ifdef ENABLE_RTSPSERVER
  rtspServer.begin();

  //streamer = new SimStreamer(true);             // our streamer for UDP/TCP based RTP
transport
  streamer = new OV2640Streamer(cam);           // our streamer for UDP/TCP based RTP
transport
#endif
}

void loop()
{
#ifdef ENABLE_WEBSERVER
  server.handleClient();
#endif

#ifdef ENABLE_RTSPSERVER
  uint32_t msecPerFrame = 100;
  static uint32_t lastimage = millis();

  // If we have an active client connection, just service that until gone
  streamer->handleRequests(0); // we don't use a timeout here,
  // instead we send only if we have new enough frames
  uint32_t now = millis();
  if(streamer->anySessions()) {
    if(now > lastimage + msecPerFrame || now < lastimage) { // handle clock rollover
      streamer->streamImage(now);
      lastimage = now;

      // check if we are overrunning our max frame rate
      now = millis();
      if(now > lastimage + msecPerFrame) {
        printf("warning exceeding max frame rate of %d ms\n", now - lastimage);
      }
    }
  }

  WiFiClient rtspClient = rtspServer.accept();
  if(rtspClient) {
```

```
        Serial.print("client: ");
        Serial.print(rtspClient.remoteIP());
        Serial.println();
        streamer->addSession(rtspClient);
    }
#endif
}

// copy this file to wifikeys.h and edit
const char *ssid =     "";         // Put your SSID here
const char *password = "";     // Put your PASSWORD here
```

ДОДАТОК В

Демонстраційний матеріал

АКТУАЛЬНІСТЬ РОЗРОБКИ

Сучасне виробництво стикається з проблемою зростання витрат на ручне сортування й необхідність підтримувати стабільну якість при мінливих умовах освітлення, фону та форми продукції.

Запропонований комплекс поєднує маркерно-орієнтовану калібровку ArUco з адаптивними алгоритмами попередньої обробки зображень, що гарантує надійну детекцію контурів і маркерів навіть за нерівномірного освітлення.

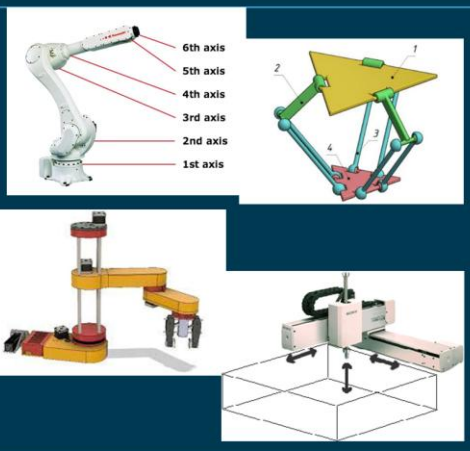
Натренована на TensorFlow модель, інтегрована в програмне забезпечення, дозволяє класифікувати різноманітні типи виробів без необхідності перепрограмування.

3

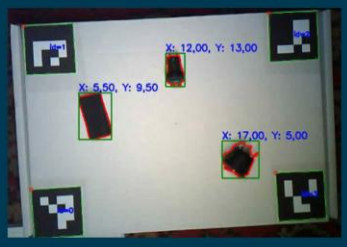
ЗАСТОСУВАННЯ РОБОТІВ-МАНІПУЛЯТОРІВ У ПРОМИСЛОВОСТІ

Роботи-маніпулятори, що наслідують рухи людської руки, виконують перенесення вантажів, фарбування, зварювання та інші операції з високою точністю завдяки вбудованим датчикам, камерам і системам комп'ютерного зору. Існують чотири основні конструктивні типи: шестикоординатні роботи, дельта-роботи, SCARA і декартові маніпулятори, кожен із яких оптимізований для певного класу завдань.

На ринку провідні виробники – FANUC, KUKA та Universal Robots – пропонують рішення різного рівня автоматизації та гнучкості, що дають змогу інтегрувати робототехніку у різні виробничі процеси.



4



Відкрита кросплатформена бібліотека з реалізованими алгоритмами та підтримкою роботи з маркерами ArUco є OpenCV (Open Source Computer Vision Library)

ЗАСТОСУВАННЯ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ

Системи комп'ютерного зору для сортування об'єктів на виробництві дозволяють визначати положення, форму та тип виробів із високою швидкістю й точністю, замінюючи рутинну ручну працю. Зображення з камери обробляється в реальному часі: спочатку виконується попередня фільтрація (згладжування, вирівнювання освітленості), потім виділяються об'єкти.

Поширені алгоритми обробки зображень для виявлення об'єктів:

- Порогова бінаризація
- Морфологічні операції (відкриття/закриття)
- Пошук контурів із фільтрацією за площею
- Сегментація за кольором
- Шаблонне співставлення
- Маркерні методи (ArUco, AprilTag) для точної побудови координатної сітки

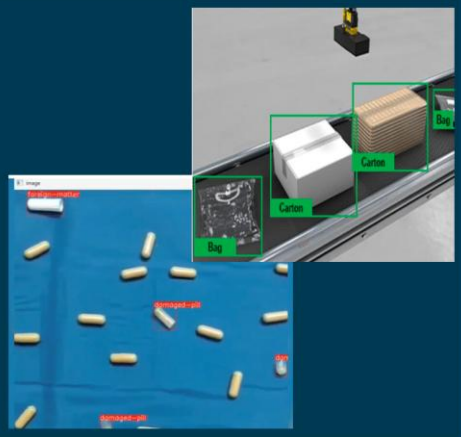
5

ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ

ШІ може бути використаний для автоматизації обробки заявок клієнтів, класифікації відгуків клієнтів, автоматизації інвентаризації, визначення та класифікації дефектів тощо.

Для автоматизації обробки заявок клієнтів типом ШІ є обробка документів, для класифікації відгуків клієнтів – класифікація категорій, для автоматизації інвентаризації – виявлення об'єктів.

Для швидкої побудови власної моделі класифікації на базі глибокого навчання зазвичай використовують переднавчені мережі, такі як YOLO, TensorFlow або EfficientNetB0, які вже навчені на масштабному наборі та здатні розпізнавати загальні ознаки об'єктів.



6

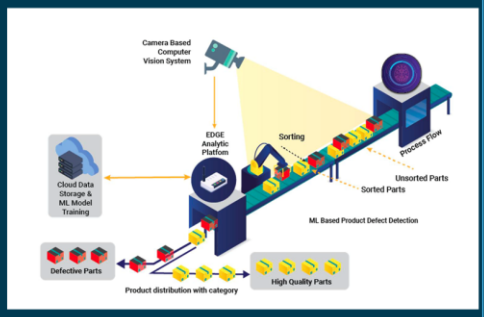
АНАЛІЗ КОМПОНЕНТІВ РОЗРОБКИ

Класичні системи комп'ютерного зору швидкі й дешеві, але крихкі до змін освітлення і положення камери

Нейромережеві рішення гнучкі в розпізнаванні нових форм, потребують потужного GPU і мають більшу затримку

Маркерні системи дозволяють виконувати стійку й точну побудову координат, вимагають нанесення маркерів на поверхню

Запропонований підхід поєднує ArUco-калібровку для точності ≤ 3 мм з класифікацією моделлю ШІ для гнучкості без додаткового апаратного навантаження



7

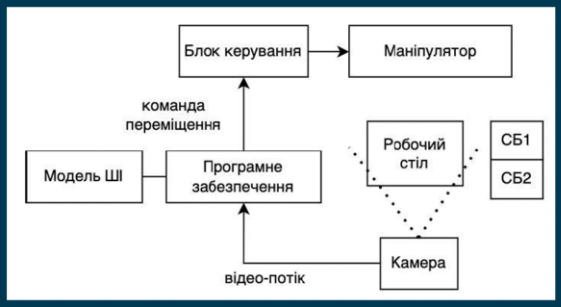
АРХІТЕКТУРА ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

Камера знімає відеопотік робочої поверхні, на яку нанесені ArUco-маркери

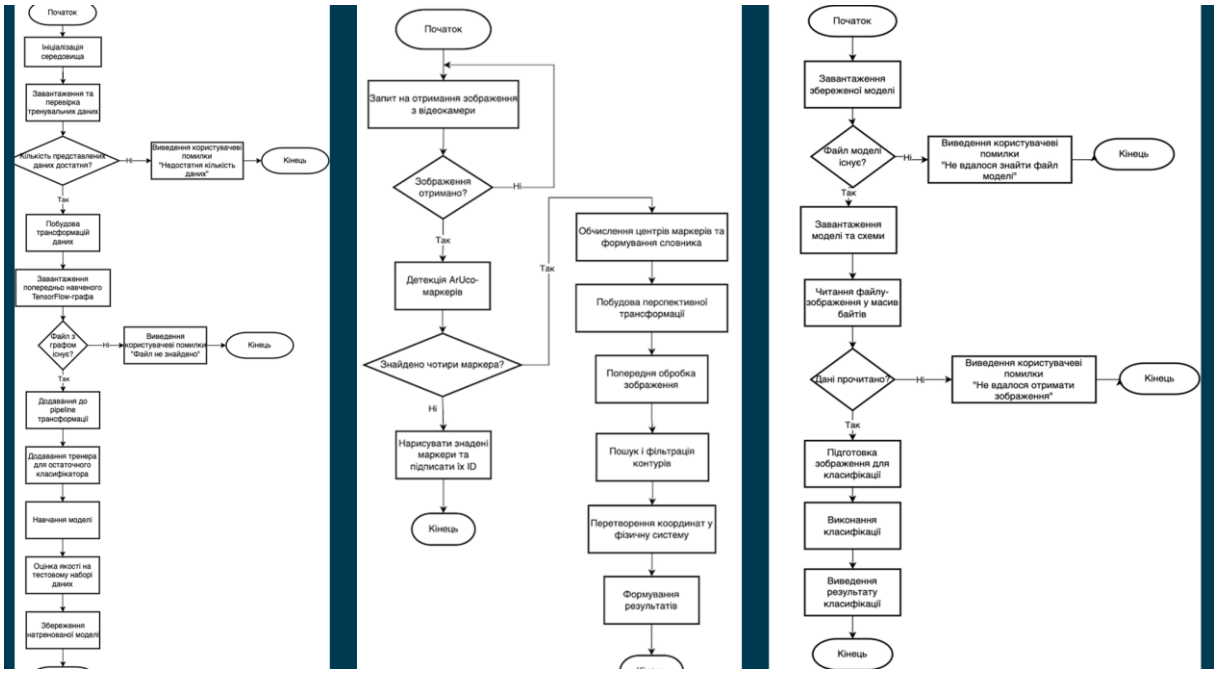
Програмне забезпечення обробляє кадри, формує команду переміщення й надсилає її в блок керування

Модель ШІ повертає мітку класу для кожного об'єкта

Маніпулятор переміщує кожен об'єкт у відповідний сортувальний бокс (СБ1, СБ2 …).



8



РЕАЛІЗОВАНА ПРОГРАМА

- Підключення та налаштування пристроїв
- Відображення відеопотоку
- Обробка зображення в реальному часі
- Класифікація об'єктів
- Формування команд керування
- Управління циклом сортування
- Конфігураційні налаштування
- Інтерфейс оператора
- Тестові функції

- Мова програмування C#
- Архітектурний патерн MVC
- Графічний фреймворк AvaloniaUI
- Бібліотека комп'ютерного зору OpenCvSharp4
- Бібліотека машинного навчання ML.NET
- 3D друкований робот-маніпулятор за відкритим проектом від 20sffactory
- Камера ESP32-cam

10

50 об'єктів відсортовано за 9 хв 5 с;

втрачено відеопотік 2 рази;

початкова похибка класифікації – 24%;

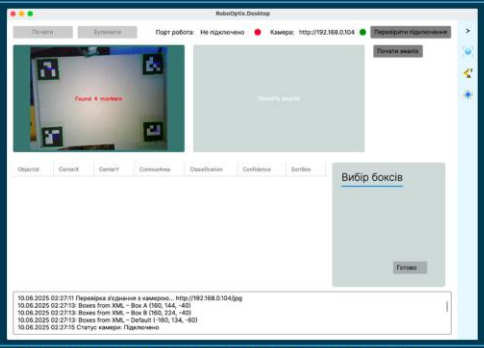
донавчено модель на невірно класифікованих зразках – похибка знизилася до 4%.

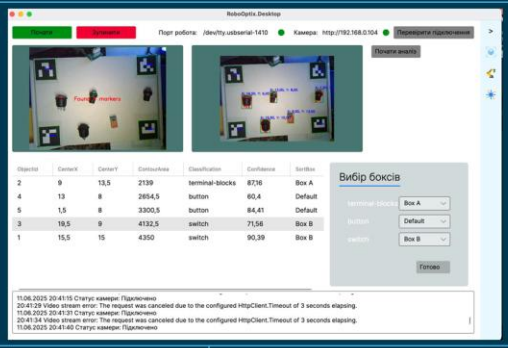
Координати визначаються точно (максимально можлива похибка ≤3 мм).

Сортування відбувається коректно, кожен об'єкт перенесено до відповідного боксу.

Відеопотік має незначні переривання.

Точність класифікації об'єктів підвищується після проведення додаткового навчання.





РЕЗУЛЬТАТИ ВИПРОБУВАНЬ

11

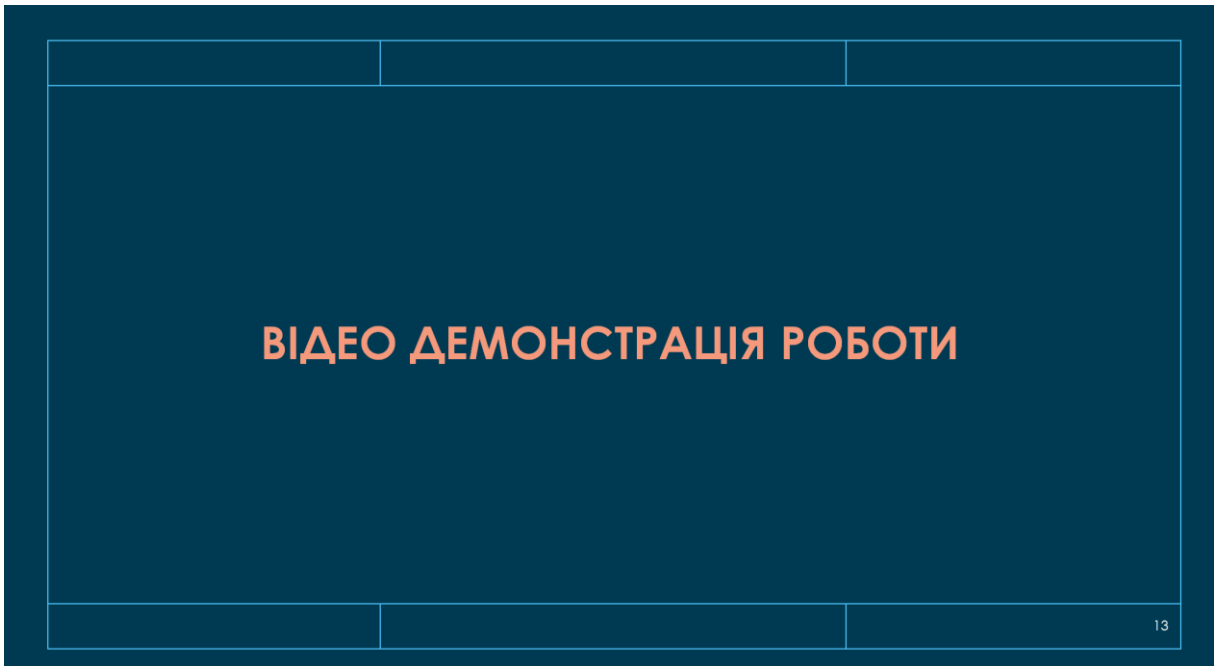
ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмне забезпечення для налаштування та керування процесом сортування роботом-маніпулятором з модулем комп'ютерного зору та елементами штучного інтелекту, за рахунок чого покращено процес сортування. Для демонстрації роботи розробленого програмного забезпечення було зібрано макет, який включає в себе робот маніпулятор та камеру. Проведені випробування показали, що координати об'єктів визначаються стабільно та похибка визначення не перевищує 3 мм, шляхом донавчання моделі ШІ можна досягнути високої точності класифікації об'єктів.

Матеріали за тематикою роботи було опубліковано у статті.

Проведені дослідження відповідають цілям сталого розвитку, зокрема Цілі сталого розвитку 9 – «Створення стійкої інфраструктури, сприяння всеохоплюючій і сталій індустріалізації та інноваціям».

12


ВІДЕО ДЕМОНСТРАЦІЯ РОБОТИ
13

№ докумен-та	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК. 505400.044 ПЗ	Пояснювальна записка	А4,106 с.
		Додаткові матеріали	
2		Демонстраційний матеріал у	А4, 13 с.
		вигляді презентації	

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК. 505400.044 ВД			
Розроб.		Казановська К.А.			Розроблення автоматизованого модуля керування роботом маніпулятором з використанням системи комп'ютерного зору з елементами штучного інтелекту Відомість кваліфікаційної роботи	Літера	Аркуш	Аркушів
Перевір.		Новоселов С.П.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н.П.						
Затв.		Невлюдов І.Ш.						