

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему «РОЗРОБКА СИСТЕМИ ДЛЯ РОЗУМНОГО СОРТУВАННЯ ТОВАРІВ»

Виконав: студент 4 курсу, групи КНТ-513сп
спеціальності 123 Комп'ютерна інженерія

Освітня програма (спеціалізація)

Комп'ютерна інженерія

(код і назва спеціальності)

КОНЕВСЬКИЙ Д.М.

(прізвище та ініціали)

Керівник ТІМЕНКО А.В.

(прізвище та ініціали)

Рецензент ХАРИТОНОВ О.Б.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти (освітній ступінь) бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і назва)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

«14» квітня 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

КОНЕВСЬКИЙ Дмитро Михайлович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Розробка системи для розумного сортування товарів»

керівник проекту (роботи) ТИМЕНКО А.В., ст. викл.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «14» квітня 2026 року № 160

2. Строк подання студентом проекту (роботи) 01.06. 2025 року

3. Вихідні дані до проекту (роботи) Raspberry Pi, ZXing, OpenCV, Picamera2, Python

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити)

1) Опис предметної області

2) Обґрунтування вибору технології розробки

3) Реалізація комп'ютерної системи

4) Збірка та монтаж експериментального стенду

5) Тестування системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1–5	ТІМЕНКО А.В., ст. викладач		
Нормоконтроль	ГОЛУБ Т.В., доцент		

7. Дата видачі завдання 15.04. 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Опис предметної області	17.04.2026	
2	Огляд прийнятих рішень	20.04.2026	
3	Вибір програмного забезпечення	23.04.2026	
4	Вибір апаратного забезпечення	27.04.2026	
5	Реалізація комп’ютерної системи	09.05.2026	
6	Збірка і монтаж стенду	15.05.2026	
7	Тестування системи	22.06.2026	
8	Оформлення пояснювальної записки	25.06.2026	

Студент(ка)

(підпис)

Дмитро КОНЕВСЬКИЙ

(Ім’я ПРИЗВИЩЕ)

Керівник проекту (роботи)

(підпис)

Артур ТІМЕНКО

(Ім’я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
91 с., 20 табл., 21 рис., 1 дод., 9 лістингів, 31 джерело.

OPENCV, PICAMERA2, PYTHON, RASPBERRY PI, ZXING, QR-КОД, АПАРАТНО-ПРОГРАМНА СИСТЕМУ СОРТУВАННЯ

Актуальність теми: актуальність теми зумовлена зростанням потреби в автоматизації процесів обробки, розподілу та сортування товарів у логістичних, складських і виробничих системах. У сучасних умовах підприємства прагнуть підвищити швидкість виконання операцій, зменшити кількість помилок під час сортування та мінімізувати вплив людського фактора. Використання інтелектуальних систем, побудованих на основі методів комп'ютерного зору, автоматичної ідентифікації та вбудованих обчислювальних платформ, дає можливість створювати ефективні та доступні рішення для розумного сортування товарів. Розробка такої системи є актуальною як з практичної, так і з науково-технічної точки зору, оскільки поєднує задачі автоматизації, обробки даних, керування виконавчими механізмами та застосування сучасних інформаційних технологій.

Мета роботи: розробити апаратно-програмну систему для розумного сортування товарів, яка забезпечує автоматичне визначення ознак товару, прийняття рішення щодо його маршрутизації та керування виконавчим механізмом сортування.

Об'єкт дослідження: процес автоматизованого сортування товарів у логістичних, складських або виробничих системах.

Предмет дослідження: методи та засоби побудови систем розумного сортування товарів, зокрема алгоритми розпізнавання об'єктів, аналізу їх

характеристик, прийняття рішень щодо маршрутизації та керування виконавчими механізмами.

Практична цінність роботи: практична цінність роботи полягає у створенні працездатного прототипу системи розумного сортування товарів, який може бути використаний для автоматизації процесів на складах, у логістичних центрах, торговельних і виробничих підприємствах. Запропоноване рішення дозволяє підвищити точність сортування, скоротити час обробки товарів, зменшити навантаження на персонал і забезпечити можливість подальшого розширення функціональності системи відповідно до конкретних потреб підприємства.

ЗМІСТ

Скорочення та умовні позначки	7
Вступ.....	8
1 Опис предметної області	10
1.1 Постановка задачі.....	12
1.2 Функціональні та нефункціональні вимоги	15
1.3 Огляд рішень прийнятих рішень	18
2 Обґрунтування вибору технологій розробки.....	37
2.1 Вибір програмного забезпечення	37
3 Реалізація комп'ютерної системи.....	44
3.1 Проектування програмної архітектури та взаємодії модулів	44
3.2 Проектування системи	51
3.3 Об'єктно–орієнтоване проектування системи	57
4 Збірка та монтаж експериментального стенду	60
4.1 Збірка та монтаж стенду	60
4.2 Налаштування зйомки (PiCam) та стабілізація параметрів.....	64
5 Тестування системи	67
5.1 Датасет і навчання моделі.....	67
5.2 Реалізація пайплайна на Raspberry Pi 4.....	74
Висновки.....	81
Перелік джерел посилання	82
Додаток А Слайди презентації	85

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

COCO	– Common Objects in Context, Набір даних «Звичайні об’єкти в контексті»
CVAT	– Computer Vision Annotation Tool, Інструмент розмітки для комп’ютерного зору
OpenCV	– Open Source Computer Vision Library, Бібліотека комп’ютерного зору з відкритим кодом
OS	– Operating System, Операційна система
PWM	– Pulse Width Modulation, Широтно-імпульсна модуляція
QR	– Quick Response Code, Код швидкого реагування
ROI	– Region of Interest, Область інтересу
UML	– Unified Modeling Language, Уніфікована мова моделювання

ВСТУП

Стрімкий розвиток інформаційних технологій та інтелектуальних систем автоматизації відкриває нові можливості для оптимізації виробничих процесів і навчання. Одним із актуальних напрямів є створення навчальних вебплатформ та експериментальних стендів, що поєднують програмні та апаратні рішення для вирішення прикладних завдань [1].

У роботі розглядається розробка навчальної вебплатформи та системи розумного сортування коробок на базі Raspberry Pi 4. Архітектура програмного забезпечення побудована як цілісна інфраструктура, де кожен компонент виконує чітко визначену функцію. Використання стеку libcamera/Picamera2 забезпечує захоплення кадрів, OpenCV реалізує геометричну нормалізацію області QR через перспективне перетворення та гомографію, а pyzbar та ZXing виконують декодування QR-кодів. Інференс нейромережі здійснюється безпосередньо на пристрої за допомогою LiteRT/TFLite, що гарантує продуктивність та оптимізацію під edge-платформи.

Апаратна частина системи включає стенд за схемою «зона зйомки → зона заслінки → розподіл у два лотки», де центральним вузлом є Raspberry Pi 4 Model B. Керування заслінкою реалізовано сервоприводом MG996R, а електричний монтаж виконано з принциповим розділенням живлення. Базова перевірка підтвердила готовність апаратної частини до інтеграції програмних модулів.

Підготовка датасету та навчання моделі організовані як відтворюваний цикл: збір даних зі стенду → розмітка → контроль якості → навчання детектора QR → оцінювання → підготовка edge-моделі. При цьому враховуються реальні умови руху коробок: варіації освітлення, кута нахилу, відстані до камери та розмиття руху. Оцінювання якості моделі здійснюється через стандартні метрики детекції та прикладні метрики інтеграції.

Таким чином, робота поєднує аналіз предметної області, розробку вимог до програмного забезпечення, проєктування архітектури та реалізацію

експериментального стенду. Система повинна бути масштабованою, стандартизованою та придатною для подальшого розвитку у сфері інтелектуальних систем сортування та електронного навчання.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Особливість предметної області полягає у тому, що коробки рухаються, а отже на якість зчитування впливають перспектива (нахил площини QR відносно камери), розмиття руху, відблиски та нерівномірне освітлення, а також часткові перекриття етикетки. Це призводить до того, що пряме декодування QR “з усього кадру” часто дає нестабільний відсоток успішних зчитувань у реальних умовах. З огляду на це обґрунтовується гібридне рішення: нейромережа використовується для локалізації області QR (визначення координат ROI), після чого виконується геометрична нормалізація виділеної області і лише тоді – класичне декодування QR. Геометрична нормалізація для пласкої етикетки коректно формулюється через гомографію та перспективне перетворення, що типово реалізується у OpenCV (зокрема, через операції отримання перспективного перетворення та застосування `warpPerspective`), а сам підхід широко описаний у навчальній документації OpenCV [2].

Функціональна постановка задачі зводиться до формування керуючої дії виконавчого механізму на підставі маршруту з QR у межах часових обмежень рухомої лінії. Вхідними даними системи є відеокадр (або серія кадрів) з PiCam, а вихідними – рішення про лоток (LEFT/RIGHT) та сигнал керування серво заслінки, синхронізований у часі з фізичним положенням коробки. При цьому важливо відокремити “час обчислення рішення” від “часу виконання механічної дії”: рішення формується одразу після детекції та декодування, однак заслінка повинна спрацювати через затримку, яка відповідає проходженню коробки від зони зйомки до точки відхилення. Тому в постановці задачі фіксується необхідність планувальника подій або кінцевого автомата станів, який забезпечує виконання команди у момент $\text{execute_at} = t_0 + \text{delay_gate}$, де t_0 – момент захоплення кадру/ідентифікації коробки, а delay_gate – калібрований параметр траси.

Узагальнена модель предметної області та інформаційно – керуючих потоків наведена на (рис. 1.1), де підкреслено, що система має два контури:

- інформаційний (виявлення та інтерпретація маршруту) ;
- керуючий (планування та актуатор).

Також показано місце застосування нейромережі як модуля локалізації області QR, а не модуля “розпізнавання маршруту”.

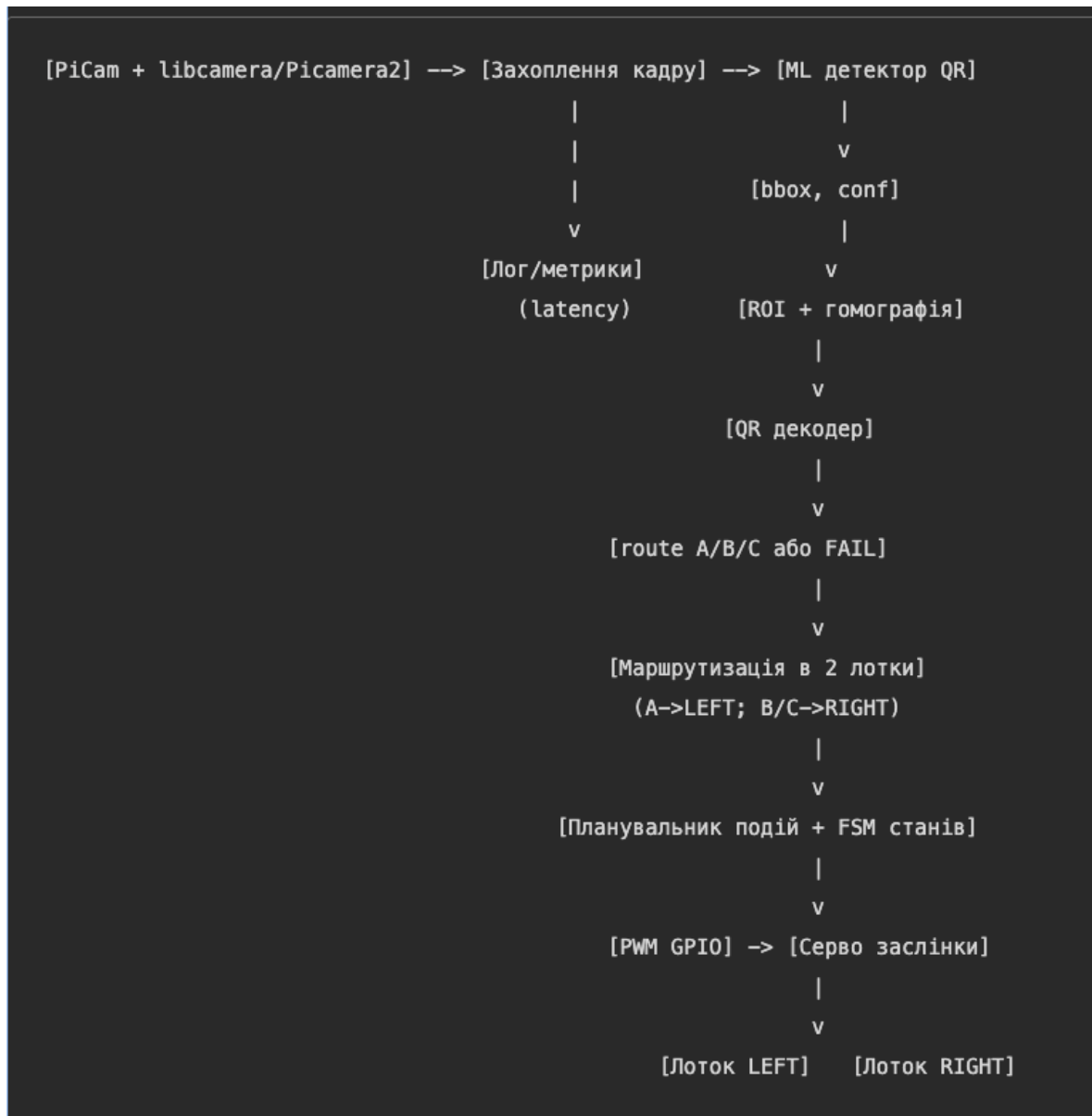


Рисунок 1.1 – Структура предметної області та загальний потік даних у системі сортування

1.1 Постановка задачі

Задача дипломної роботи полягає у створенні апаратно – програмної системи, що виконує автоматизоване сортування коробок у потоці за маршрутом, закодованим у QR, з використанням нейромережі на Raspberry Pi 4 [3]. Предметна область охоплює логістичні процеси внутрішнього переміщення та розподілу вантажних одиниць, у яких кожна коробка повинна бути спрямована у відповідний напрямок згідно з інформацією на етикетці, а точність і швидкодія мають бути достатніми для роботи на рухомій лінії. У цій роботі вважається, що QR-код містить маршрут з множини {A, B, C}, тоді як механічна частина стенду реалізує лише два лотки, тому на рівні керування вводиться функція відображення маршруту у два стани виконавчого механізму (LEFT/RIGHT) із заздалегідь визначеною політикою fail – safe. Використання QR-коду як носія маршруту є доцільним, оскільки QR є стандартизованою символією, для якої визначено формати, правила кодування, вимоги до якості та еталонний підхід декодування, що створює формальну базу для постановки вимог до зчитування та оцінювання якості роботи системи [4].

Оскільки задача включає синхронізацію з рухом, доцільно одразу в постановці зафіксувати типовий сценарій взаємодії компонентів у часі, що дає основу для подальшого проєктування, тестування і вимірювання затримок. Діаграма взаємодії (рис. 1.2) показує, що детектор формує координати ROI, декодер формує маршрут, маршрутизатор визначає цільовий лоток, а планувальник виконує команду на серво в запланований момент, причому в разі помилки декодування застосовується fail-safe (базовий лоток).

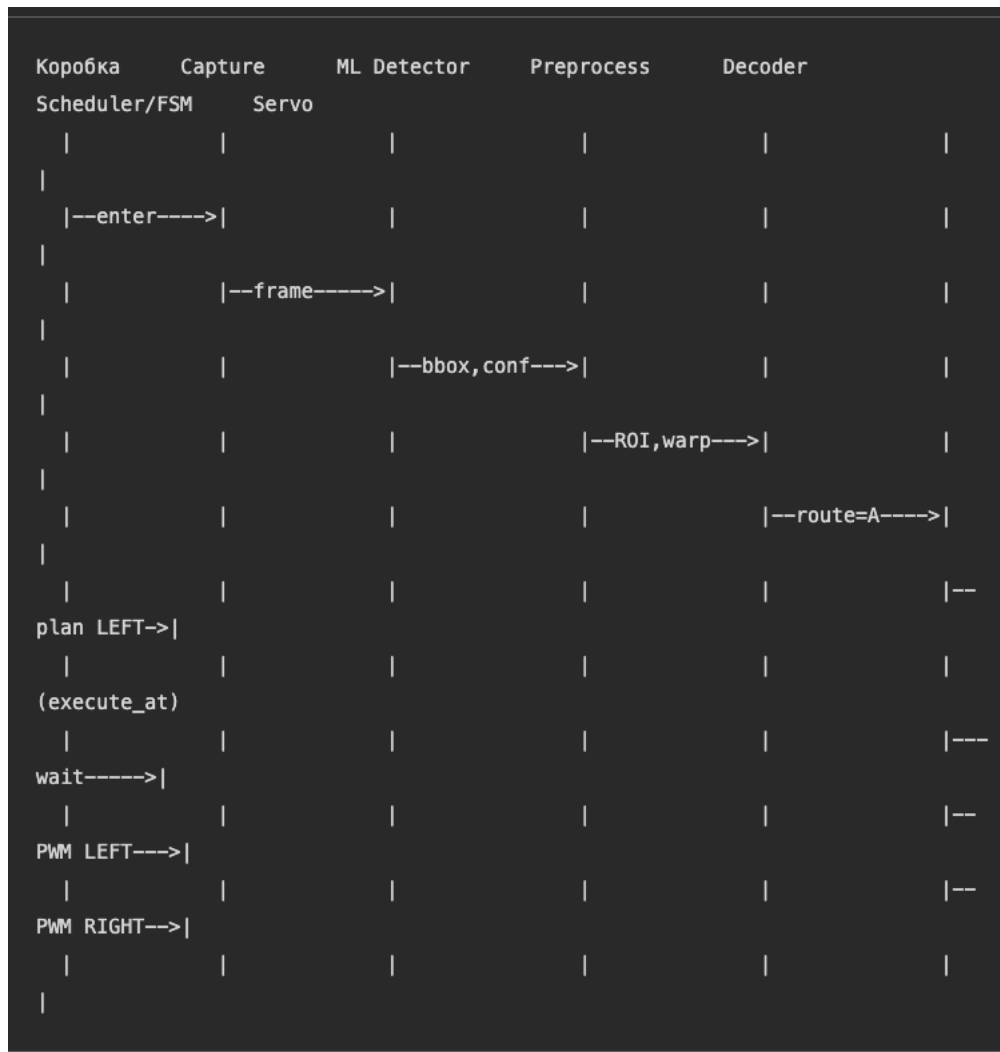


Рисунок 1.2 – Діаграма взаємодії (Sequence) для рухомої коробки та керування заслінкою

Логічна структура системи її основні сутності та їх відповідальності представлена у вигляді діаграми класів (рис. 1.3). Така структура відповідає інженерній практиці розподілу відповідальностей: камера формує кадр, детектор формує локалізацію, препроцесор нормалізує ROI, декодер повертає route, маршрутизатор мапить route у lane, планувальник забезпечує часову синхронізацію, а контролер серво реалізує низькорівневе керування [5].

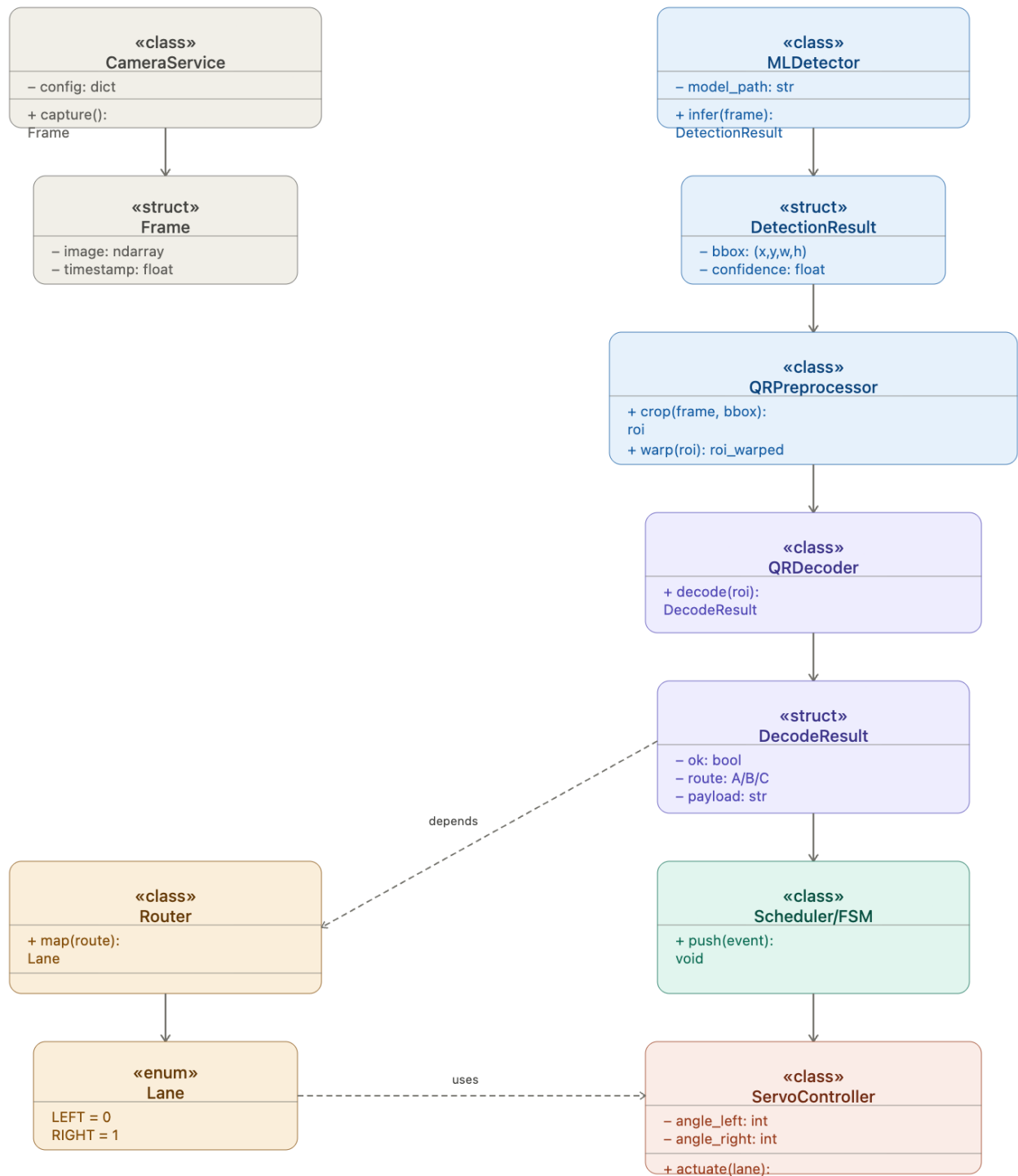


Рисунок 1.3 – Діаграма класів (Class) логічних компонентів системи

1.2 Функціональні та нефункціональні вимоги

Формалізація вимог у постановці задачі повинна містити як функціональні аспекти (що система робить), так і нефункціональні (з якою якістю/швидкістю/стійкістю), причому критерії приймання мають бути вимірюваними на стенді (табл.1.1). У цьому контексті ключовими показниками є частка успішного зчитування QR (read rate), точність сортування у два лотки (sorting accuracy), сумарна end-to-end затримка від моменту захоплення кадру до формування події керування (latency_total) і пропускна здатність стенду у коробках за хвилину (throughput). Ці вимоги узгоджуються з природою QR як стандартизованого коду з визначеними вимогами до якості друку й декодування, з практикою використання перспективних перетворень для нормалізації планарних об'єктів, а також із можливостями стека Picamera2/libcamera для керованого захоплення кадрів і відтворюваних умов експериментів [6].

Таблиця 1.1 – Функціональні та нефункціональні вимоги

ID	Тип	Формулювання
FR-01	FR	система повинна захоплювати кадри з PiCam із фіксованими параметрами зйомки
FR-02	FR	система повинна виконувати ML-локалізацію області QR та повертати bbox і confidence
FR-03	FR	система повинна формувати ROI та виконувати перспективну нормалізацію перед декодуванням
FR-04	FR	система повинна декодувати QR і витягувати маршрут A/B/C з payload
FR-05	FR	система повинна виконувати відображення маршруту у 2 лотки та формувати lane=LEFT/RIGHT
FR-06	FR	система повинна планувати керуючу дію на заслінку за параметром delay_gate (для рухомої лінії)
FR-07	FR	система повинна керувати серво заслінкою через PWM та повертати заслінку у базовий стан
FR-08	FR	система повинна вести лог подій з фіксацією результату, таймінгів і причин помилок
NFR-01	NFR	read rate має бути не нижчим за X% у контрольному прогоні ($N \geq 100$)

Продовження таблиці 1.1

ID	Тип	Формулювання
NFR-02	NFR	sorting accuracy має бути не нижчим за Y% у контрольному прогоні ($N \geq 100$)
NFR-03	NFR	latency_total має бути не більшим за T мс у середньому (або за P95)
NFR-04	NFR	система повинна мати fail-safe поведінку при no_detection/decode_error (базовий лоток)

Оскільки QR містить три значення маршруту, а механіка реалізує два виходи, постановка задачі повинна однозначно визначити правило маршрутизації та поведінку при винятках. У межах стенду доцільно прийняти базовий стан заслінки як RIGHT, а відхилення у LEFT здійснювати лише для маршруту A, тоді як B і C спрямовувати у RIGHT; у разі FAIL (не знайдено QR або не декодується) також застосовується RIGHT з позначенням REJECT у журналі. Відображення маршруту наведено у (табл. 1.2.), яка фактично є частиною специфікації поведінки системи.

Таблиця 1.2 – Правило маршрутизації A/B/C у два лотки та fail-safe політика

Вхід (route)	Вихід (lane)	Статус/примітка
A	LEFT	активувати заслінку на t_open
B	RIGHT	базовий режим, без активації заслінки
C	RIGHT	базовий режим, без активації заслінки
FAIL/NULL	RIGHT	fail-safe, логувати як REJECT (причина)

В таблиці 1.3 наведено дані які дозволяють з одного боку, порівнювати оптимізації inference на LiteRT/TFLite [7], а з іншого – робити аналіз проблем декодування з використанням бібліотеки ruzbar або контрольного декодера.

Таблиця 1.3 – Мінімальний формат лог-запису для оцінювання якості

Поле	Опис
timestamp	час події (t0) або час формування рішення
route	A/B/C або NULL при FAIL
lane	LEFT/RIGHT

Продовження таблиці 1.3

Поле	Опис
fail_reason	no_detection / decode_error / invalid_route / other
status	OK/FAIL
det_confidence	confidence детектора QR
t_infer_ms	час інференсу ML
t_decode_ms	час ROI+warp+decode
latency_total_ms	сумарний час обробки до постановки події
execute_at	запланований час актуатора ($t_0 + \text{delay_gate}$)

Вимога “працювати у потоці” зумовлює необхідність мінімальної формалізації алгоритму в постановці задачі, щоб надалі можна було однозначно реалізувати і перевірити поведінку. Такий алгоритм, у стислому вигляді, полягає у захопленні кадру, локалізації QR нейромережею, формуванні нормалізованого ROI через перспективне перетворення, декодуванні, мапінгу route у lane, плануванні події на момент $t_0 + \text{delay_gate}$ і керуванні серво заслінки; при будь-якій відмові зчитування застосовується fail-safe маршрут у RIGHT. Формальну форму цього процесу наведено як псевдокод у лістинг 1.1, який використовується як “контракт поведінки” для наступного розділу реалізації.

Лістинг 1.1 – Узагальнений псевдокод роботи системи у потоці

```

t0 = now(); frame = capture();
bbox, conf = ml_detect_qr(frame);
if bbox absent: route = FAIL;
else roi = warp_perspective(crop(frame, bbox)); route =
decode_qr(roi);
lane = (route == A) ? LEFT : RIGHT; // B/C/FAIL ->
RIGHT
schedule(execute_at = t0 + delay_gate, lane);
if now >= execute_at and lane == LEFT: servo_left(t_open);
servo_right();
log(t0, route, lane, status, fail_reason, conf, timings);

```

Таким чином, постановка задачі в межах предметної області формує чіткий, вимірюваний і відтворюваний контур “ідентифікація–маршрутизація–актуація” [8], де QR як стандартизований носій маршруту задає формальну основу вимог, OpenCV забезпечує коректні геометричні перетворення для нормалізації ROI,

Picamera2/libcamera забезпечують контрольоване отримання вхідних даних, LiteRT/TFLite забезпечує виконання нейромережі на edge-пристрої, а pyzbar/ZXing забезпечують декодування QR на етапі інтерпретації.

1.3 Огляд рішень прийнятих рішень

Сортування коробок у логістичних та складських процесах належить до задач автоматичної ідентифікації та маршрутизації одиниць вантажу, де кожна коробка має бути віднесена до певного напрямку або зони обробки на основі закодованих даних (ідентифікатор, напрямок, пріоритет, тип обробки). У межах цієї роботи розглядається сценарій, коли маршрут коробки міститься у QR-коді у вигляді символу з множини {A, B, C}, а фізично система має лише два лотки, тому на рівні керування вводиться функція відображення маршруту у два стани заслінки (наприклад, $A \rightarrow \text{LEFT}$, $B/C \rightarrow \text{RIGHT}$), із застосуванням безпечної політики деградації, коли за будь-яких помилок зчитування коробка спрямовується у базовий лоток, а подія фіксується як виняток у журналі. В основі використання QR лежить міжнародна специфікація QR-коду, що визначає формати символу, методи кодування, вимоги до якості друку та правила корекції помилок, тобто QR є стандартизованим носієм даних, придатним до промислового застосування.

З практичної точки зору, на відміну від стаціонарного сканування “впритул” ручним сканером, зчитування QR камерою в русі має типові ускладнення: перспектива, коли площа коду нахилена до оптичної осі; розмиття руху через відносне переміщення коробки; відблиски та нерівномірне освітлення на пакувальній плівці; часткові перекриття етикетки; наявність кількох контрастних текстур на коробці, які заважають пошуку коду. Саме ці фактори знижують відсоток успішного декодування при прямому застосуванні декодера “на весь кадр”, тому в сучасних інженерних рішеннях часто застосовують або попередню

обробку зображення (підвищення контрасту, бінаризація, геометричне вирівнювання), або локалізацію області інтересу (ROI), щоб декодувати лише ту ділянку, де реально міститься код. Геометричне вирівнювання для коду, надрукованого на майже пласкій етикетці, природно формулюється як задача планарної гомографії та перспективного перетворення, що підтримується інструментами OpenCV, зокрема через концепцію гомографії й операцію перспективного “warp”.

Огляд можливих підходів до побудови системи сортування доцільно почати з класифікації “як отримується маршрут” та “як здійснюється відхилення потоку”. За першим критерієм існує підхід з окремим лазерним/CCD–сканером штрихкодів або QR, який дає високу стабільність на стандартизованих етикетках, проте є дорожчим і менш гнучким у лабораторному стенді; альтернативою є камера (PiCam) із програмним зчитуванням, що дешевше, гнучкіше та дозволяє розширювати функціональність (логування кадрів, діагностика відмов, облік якості друку). За другим критерієм, для двох лотків найпростішим є заслінка–відхиляч (diverter gate), що у базовому положенні пропускає коробку в основний канал, а в активному – переводить у відвідний канал; такий принцип застосовується в сортувальних і divert–системах конвеєрів як один із типових механізмів перенаправлення потоку на іншу лінію, що підкреслює релевантність обраної механіки для задачі маршрутизації.

У межах цієї дипломної роботи обрана архітектура “камера + неймережа + класичний декодер + заслінка”, оскільки вона дозволяє поєднати надійність декодування стандартизованого QR з підвищеною стійкістю локалізації коду в складних умовах [9]. Неймережа застосовується не для “розпізнавання маршруту” (бо маршрут уже закодований у QR), а для локалізації області коду на кадрі; після цього ROI нормалізується геометрично й подається на декодер. Такий гібридний підхід є інженерно виправданим для Raspberry Pi 4, де ресурси обмежені, і важливо мінімізувати обсяг пікселів, що передаються на декодування, а також стабілізувати умови для декодера. Для inference на edge–пристрої доцільно використовувати легкий on–device рантайм, орієнтований на низьку

затримку і оптимізації, зокрема LiteRT (наступник/еволюція TensorFlow Lite) як інструмент для виконання .tflite-моделей на різних платформах та з підтримкою оптимізацій на кшталт квантизації.

Платформою виконання обрано Raspberry Pi 4, а джерелом зображення – PiCam [10]. Для такої конфігурації принципово важливими є стабільне захоплення кадрів і відтворюваність параметрів зйомки (експозиція, баланс білого, роздільна здатність), тому на рівні програмного забезпечення доцільно спиратися на сучасний стек libcamera і Python-інтерфейс Picamera2, що позиціонується як libcamera-орієнтована заміна попереднього “legacy” стека і забезпечує стандартний шлях роботи з камерами Raspberry Pi [3]. Обчислювальна частина, пов’язана з нейромережевою локалізацією, повинна виконуватися з прийнятною затримкою на edge-пристрої, тому як рантайм inference логічно розглядати on-device середовище, орієнтоване на низьку латентність і оптимізації, зокрема LiteRT (еволюція TensorFlow Lite), що підтримує типовий pipeline “конвертація → оптимізація/квантизація → виконання на пристрої”. Декодування QR у Python-частині може виконуватися бібліотекою pyzbar як обгорткою над zbar, яка вміє працювати з ndarray OpenCV та повертати не лише payload, а й геометрію знайденого коду, що зручно для аналізу відмов; альтернативою для контрольних перевірок може слугувати ZXing як відома бібліотека для 1D/2D-кодів із підтримкою QR [11].

Оскільки джерелом зображення обрано PiCam, критичною є стабільна робота стека камери та можливість налаштування режимів зйомки. У сучасних збірках Raspberry Pi OS рекомендованим підходом є використання libcamera й Python-бібліотеки Picamera2, яка позиціонується як заміна “legacy” стеку та надає контроль над конфігурацією потоків, роздільною здатністю, а також над параметрами експозиції й балансу білого, що важливо для повторюваності експериментів.

Для декодування QR у програмному стеку розглядаються типові бібліотеки. Практично зручним варіантом у Python є pyzbar, що є обгорткою над zbar і вміє декодувати QR з масивів OpenCV або зображень Pillow, повертаючи також

геометрію знайденого символу (прямокутник/полігон), що корисно для діагностики; як альтернативний або контрольний інструмент може застосовуватись ZXing (Zebra Crossing) – багатоформатна бібліотека для 1D/2D–кодів, яка широко використовується в Java–екосистемі і підтримує QR як окремий формат .

Порівняння підходів зчитування маршруту з коробки та їх придатність до сортування наведено у таблиці 1.4. Тут критерії зведені до трьох блоків: стійкість до умов зйомки, обчислювальна складність на Raspberry Pi 4 та потенціал масштабування. Ключовими критеріями є read rate (частка успішно зчитаних кодів), sorting accuracy (частка правильно відсортованих коробок) та latency (час від появи коробки в зоні зйомки до формування керуючої дії), оскільки саме вони визначають практичну цінність системи у реальному потоці.

Таблиця 1.4 – Порівняння підходів зчитування маршруту з коробки та їх придатність до сортування

Підхід	Принцип	Переваги	Недоліки	Придатність
P1	пряме декодування QR з повного кадру	проста реалізація, мінімум модулів	низька стійкість до перспективи/відблисків/blur, падіння read rate	середня (швидко, але нестабільно)
P2	класична CV–підготовка + декодування (пороги, контури, warp)	краще за P1 при контрольованих умовах, без навчання	складна підбірка параметрів, погано масштабується на різні умови	середня (потребує тонкого тюнінгу)
P3	ML–локалізація QR + ROI + warp + декодер	висока стійкість, стабільніший	потребує датасету та навчання, inference додає latency	висока (за умови легкої моделі і оптимізації)
P4	окремий промисловий сканер	найвища стабільність менше залежить від освітлення	вартість, менша гнучкість для прототипу, інтеграція	висока, але недоцільно для бюджетного стенду

Узагальнена структурна модель предметної області, яка відображає основні сутності “коробка–маршрут–рішення–виконавчий механізм”, наведена на (рис. 1.4). Ця схема потрібна для того, щоб показати, що в системі є два контури: інформаційний (зчитування/інтерпретація) і керуючий (планування/актуація). Важливо, що при русі коробок ключовим стає часовий зв’язок між моментом спостереження і моментом фізичної дії, тому в архітектурі потрібно явно виділити планувальник подій або кінцевий автомат, який забезпечує виконання команди у правильний момент часу.



Рисунок 1.4 – Структура предметної області “ідентифікація–маршрутизація–сортування”

З позиції архітектури рішення, система розкладається на послідовний конвеєр обробки: захоплення кадру, локалізація QR нейромережею, формування ROI, перспективна нормалізація, декодування, мапінг у два стани, планування актуатора, керування серво та журналювання. Такий конвеєр зручно представити як функціональну архітектуру (рис. 1.5), де кожен блок має чіткі входи/виходи, а

також контрольні точки вимірювання часу, потрібні для експериментів. Підкреслюється, що перспективна нормалізація реалізується на основі гомографії та операцій перспективного перетворення OpenCV, а виконання моделі на Raspberry Pi забезпечується edge-рантаймом на кшталт LiteRT/TFLite.

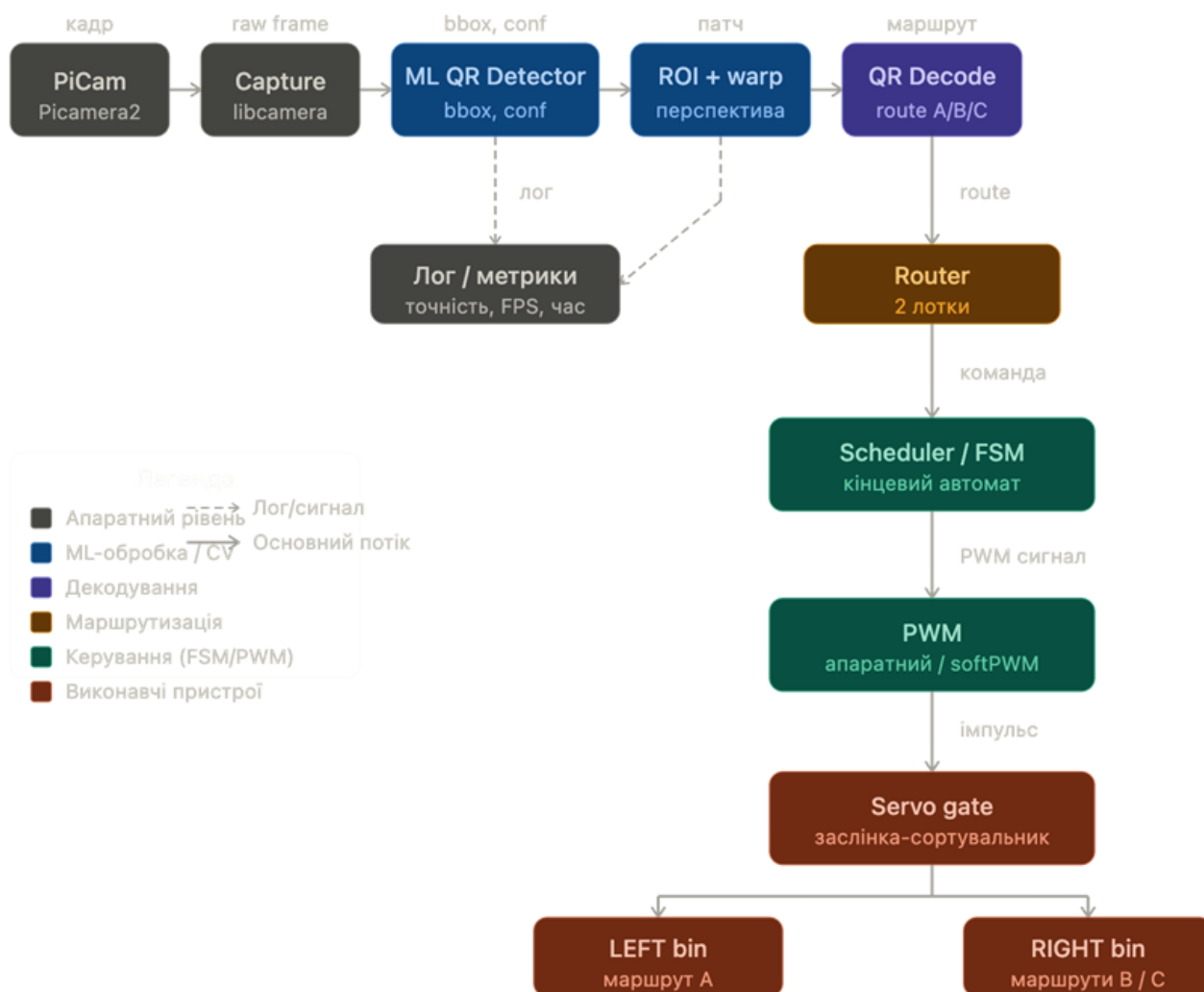


Рисунок 1.5 – Функціональна архітектура рішення “PiCam → ML → ROI → декодер → заслінка”

Взаємодія модулів у часі для типового сценарію, коли коробка має маршрут А і повинна бути відхилена в LEFT, наведена на (рис. 1.6). Діаграма підкреслює, що команда серво не подається “миттєво” після декодування, а планується із затримкою `delay_gate`, яка залежить від відстані між зоною зйомки та заслінкою і швидкості конвеєра; при цьому end-to-end latency впливає на те, яку саме

затримку потрібно планувати, що зумовлює необхідність калібрування параметрів у експериментальній частині.

Рисунок 1.3 — Деталізований конвеєр обробки з таймінгами та подіями

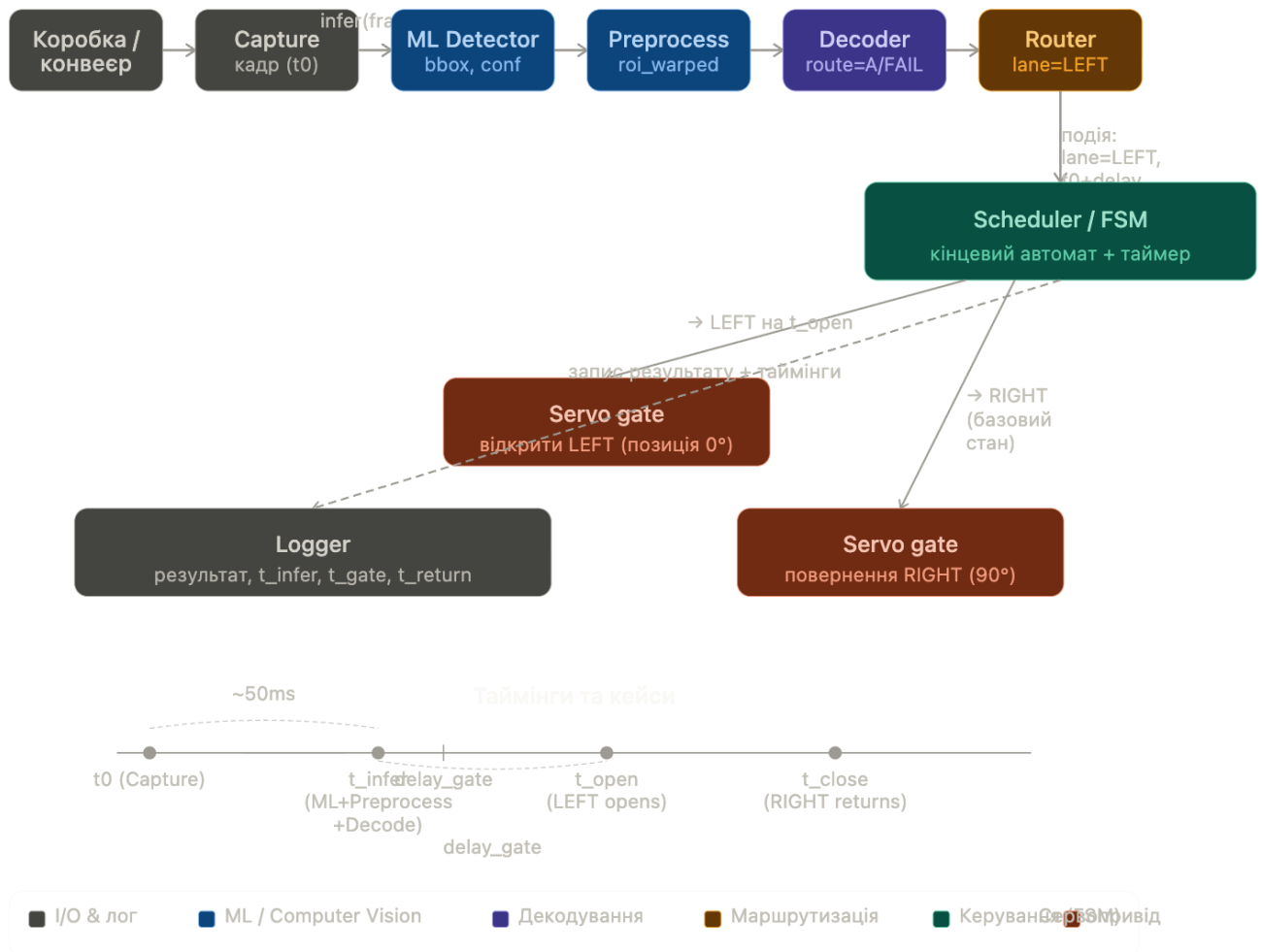


Рисунок 1.6 – Діаграма взаємодії (Sequence) для випадку route=A

Для формалізації програмної структури, що відповідає за відтворюваність і тестованість, на етапі аналізу основні сутності й їх відповідальності у вигляді UML-подібної діаграми класів відображені на рисунку. 1.7, яка показує правильний розподіл ролей: камера лише постачає кадр, детектор повертає координати, препроцесор нормалізує ROI, декодер повертає route, маршрутизатор перетворює його у керуючий стан, а планувальник відповідає за часову синхронізацію з механікою. Наявність Logger/Storage підкреслює вимогу до вимірюваності, тобто можливість будувати графіки й таблиці результатів без ручного підрахунку.

Рисунок 1.4 — Архітектура системи: компоненти, сервіси та типи даних

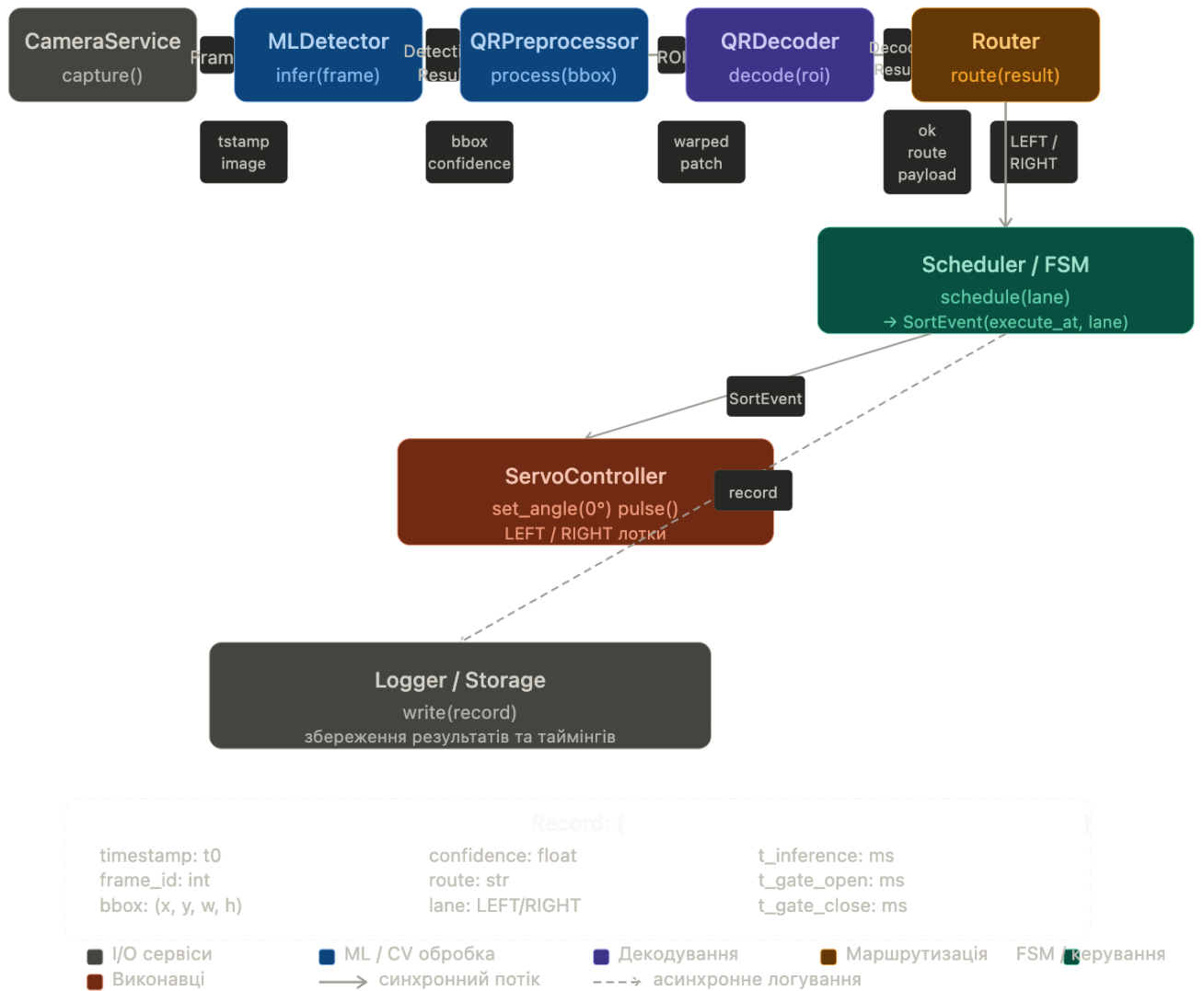


Рисунок 1.7 – Діаграма класів (Class) логічних компонентів системи

Для камери обрано Picamera2 [12] як сучасний інтерфейс над libcamera, оскільки він підтримується Raspberry Pi екосистемою та відповідає вимогам керованого захоплення кадрів і налаштувань.

Для inference придатний LiteRT/TFLite, оскільки він позиціонується як високопродуктивний on-device рантайм і пропонує типову схему “отримати .tflite → оптимізувати (квантизація) → виконати на пристрої”, що прямо відповідає задачі мінімізації затримки на edge.

Для перспективної корекції використовується OpenCV як де-факто стандартний набір інструментів комп’ютерного зору; його документація прямо

описує поняття гомографії та демонструє застосування перспективного перетворення для планарних об'єктів, що узгоджується з фізичною природою QR-етикетки на коробці.

Для декодування QR у Python практичним є `pyzbar` [13], який працює з `ndarray` OpenCV і повертає декодовані дані разом із геометрією детекції, що спрощує налагодження; це підтверджується описом бібліотеки та її підтримуваними типами входу. Альтернативно для валідації або порівняння може бути використаний ZXing як багатоформатна бібліотека для штрихкодів, включно з QR, що широко відома у практиці розробки.

Підсумовуючи огляд, обране рішення “ML-локалізація + перспективна нормалізація + класичне декодування + серво-заслінка” забезпечує баланс між надійністю та реалізованістю на Raspberry Pi 4, а також створює достатню науково-технічну новизну для дипломної роботи, оскільки включає обробку відеоданих на edge, інтеграцію ML-модуля, роботу у часових обмеженнях та апаратне керування виконавчим механізмом. Для формального переходу від аналізу до проєктування у роботі доцільно зафіксувати перелік критеріїв оцінювання (`read rate`, `sorting accuracy`, `latency`, `throughput`) та набір сценаріїв відмов (`no_detection`, `decode_error`, `invalid_route`, `timing_error`), які надалі стануть основою програми випробувань і експериментальної частини, а також будуть відображені в структурі журналу подій і графіках результатів (лістинг 1.2).

Лістинг 1.2 – Псевдокод гібридного зчитування та планування спрацювання заслінки

```
t0 = now(); frame = capture();
bbox, conf = detect_qr(frame);
if bbox absent -> route = FAIL;
else roi = warp(crop(frame, bbox)); route = decode_qr(roi);
lane = (route == A) ? LEFT : RIGHT;          // B/C/FAIL -> RIGHT
(fail-safe)
event.execute_at = t0 + delay_gate; push(event);
when now >= execute_at: if lane==LEFT -> servo(LEFT) for t_open then
servo(RIGHT);
log(timestamp=t0, route, lane, conf, latency=now()-t0);
```

У цьому розділі обґрунтовується проєктне рішення та формуються вимоги до апаратно–програмної реалізації системи розумного сортування коробок за маршрутом, закодованим у QR, із використанням неймережі на Raspberry Pi 4. Вихідною передумовою є те, що QR–код як символіка має формалізовані вимоги до структури, зон функціональних шаблонів, кодування та корекції помилок, які визначені стандартом ISO/IEC 18004:2024, і саме ці вимоги задають базову рамку до якості зображення, наявності “quiet zone”, допусків друку та сценаріїв часткового пошкодження коду.

Разом з тим, у практичних умовах руху коробок відносно камери ключовими факторами деградації зчитування є перспектива, розмиття, відблиски та часткові перекриття, тому архітектура системи обирається гібридною: неймережевий модуль застосовується для надійної локалізації області QR на кадрі, після чого виконується геометричне нормування (корекція перспективи) і лише тоді – класичне декодування QR; такий поділ мінімізує обчислювальні витрати на Raspberry Pi 4 і підвищує відсоток успішного читання за рахунок точного виділення ROI.

Геометричне нормування [14] спирається на поняття планарної гомографії та перспективного перетворення, що реалізуються у OpenCV через обчислення матриці перетворення і функцію `warping`, а теоретичні передумови й практичні приклади такого підходу описані в офіційній документації OpenCV.

Для підсистеми декодування QR у складі програмного забезпечення розглядаються бібліотеки, що надають функції розпізнавання 2D–кодів із зображень, зокрема:

- `pyzbar` як Python–обгортка над `ZBar` підтримує зчитування QR з масивів OpenCV і повертає не лише дані, а й локалізацію знайденого коду, що корисно для діагностики та логування;
- `ZXing` є відомою кросплатформенною бібліотекою для обробки 1D/2D штрихкодів, включно з QR, і може використовуватися як еталонний декодер у тестах порівняння.

Вхідні дані системи формуються камерою Raspberry Pi, а для PiCam доцільно використовувати сучасний стек libcamera і Python–інтерфейс Picamera2, який є рекомендованою заміною застарілого Picamera та надає керований доступ до конфігурацій потоків і параметрів зйомки. При цьому фіксація експозиції, балансу білого та роздільної здатності є критичною для зменшення міжкадрової варіативності й стабілізації вхідного розподілу для нейромережі. Враховуючи обмеження обчислювальних ресурсів Raspberry Pi 4, inference–частину доцільно реалізувати через легкий рантайм on–device машинного навчання на кшталт TensorFlow Lite / LiteRT, який орієнтований на низькі затримки та оптимізацію під edge–пристрої, включно з квантизацією та оптимізованими ядрами виконання. Це дозволяє сформулювати вимірювану мету оптимізації: мінімізувати latency і зберегти прийнятний рівень точності локалізації QR.

Розроблювана система виконує сортування в два лотки, тоді як QR містить маршрут A/B/C; отже, на рівні бізнес–логіки вводиться функція маршрутизації, яка відображає значення маршруту у два керуючі стани заслінки, наприклад правило $A \rightarrow \text{LEFT}$, $B/C \rightarrow \text{RIGHT}$, а у випадку невдалого зчитування застосовується fail–safe політика за замовчуванням у RIGHT з маркуванням події як REJECT у журналі. Така політика відповідає вимогам безпечної деградації: система не повинна створювати небезпечні механічні колізії та має зберігати контрольовану поведінку при відмовах камери, нейромережі або декодера.

Для реалізації fail–safe на рівні виконавчого механізму задається “базове” положення заслінки (RIGHT), яке не потребує активного утримання і дозволяє пропускати коробки без дії; активне положення (LEFT) вмикається лише для випадків маршруту A, після чого повертається до базового стану. Узагальнена структурна схема апаратно–програмного комплексу наведена на (рис. 1.8), де показано головні потоки даних і керування, а також точки вимірювання часових параметрів (затримка обробки, затримка до заслінки, тривалість відкриття).

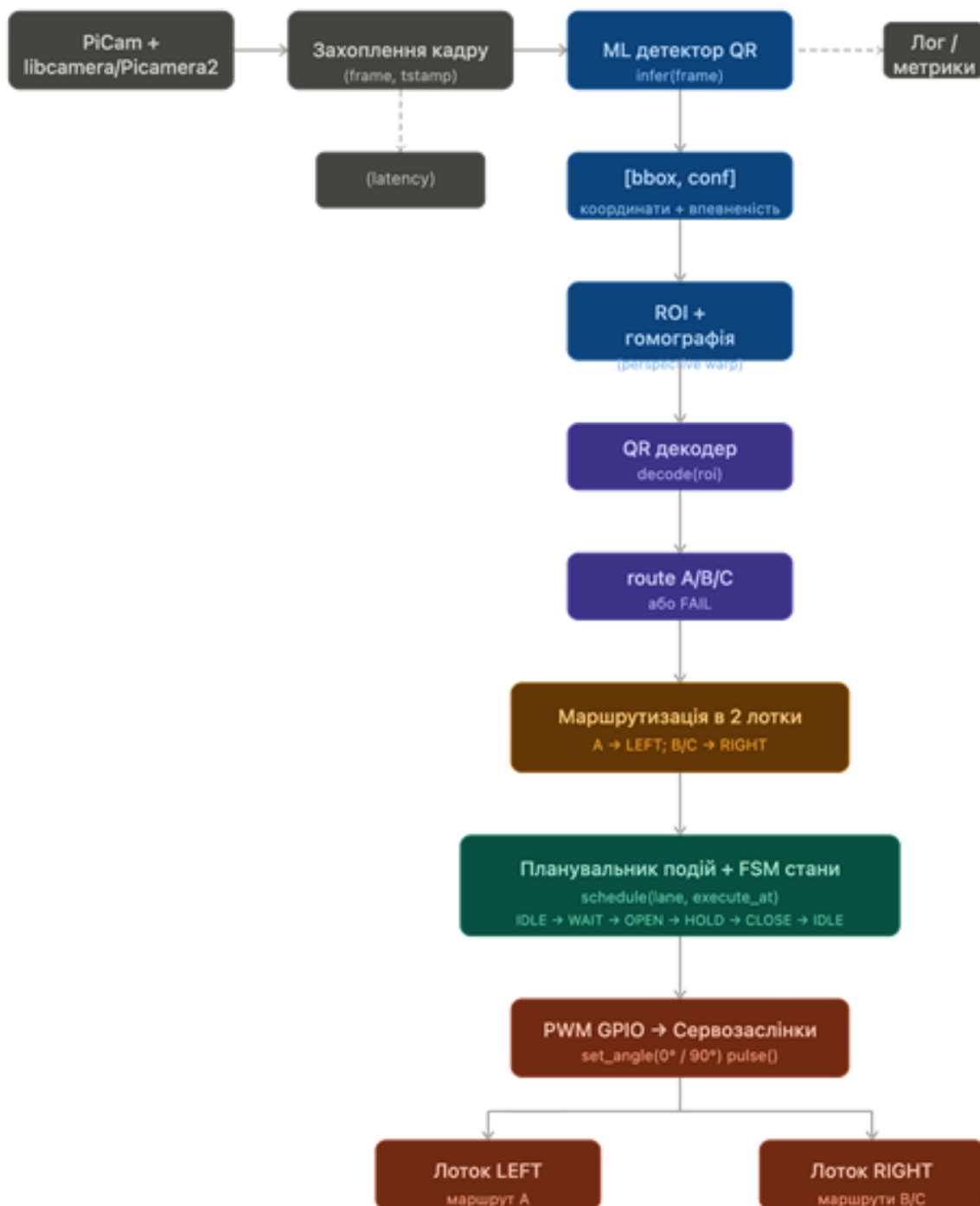


Рисунок 1.8 – Загальна архітектура системи сортування (дані та керування)

Компонентна взаємодія програмних модулів подається як логічна структура, де кожен модуль має чітко визначений контракт входів і виходів, що спрощує тестування та заміну реалізацій (наприклад, зміна декодера або формату моделі). Важливо, що для рухомих коробок рішення про сортування має бути не лише правильним, а й синхронізованим у часі з фізичним положенням коробки

відносно заслінки. Тому в архітектурі вводиться планувальник подій, який створює “заплановану дію” з моментом виконання $execute_at = t_now + delay_gate$. Затримка $delay_gate$ визначається як час проходження від зони зйомки до заслінки з урахуванням затримки обробки кадру і запасу на варіації швидкості та механічні люфти.

В роботі це обґрунтовується як калібрувальний параметр, що визначається експериментально і фіксується в конфігурації.

Розгортання на апаратній платформі описується діаграмою розгортання (рис. 1.9), де Raspberry Pi 4 є edge-вузлом, який виконує захоплення, inference – декодування та керування серво; така централізація мінімізує затримки на мережеву взаємодію і відповідає сценаріям локальної автоматизації [15].



Рисунок 1.9 – Діаграма розгортання (Deployment) для Raspberry Pi 4

Діаграма взаємодії у часі відображає ключовий сценарій “коробка з маршрутом А”, де заслінка має відкритися в правильний момент; саме ця діаграма дозволяє формально описати, що *latency* обробки не дорівнює *delay_gate* і що керування приводом виконується подією, а не “відразу після розпізнавання”.

Взаємодія подана на (рис. 1.10) як спрощена UML–послідовність, де вимірюються два часові інтервали: T_{proc} – обробка кадру та *delay_gate* – очікування до моменту дії.

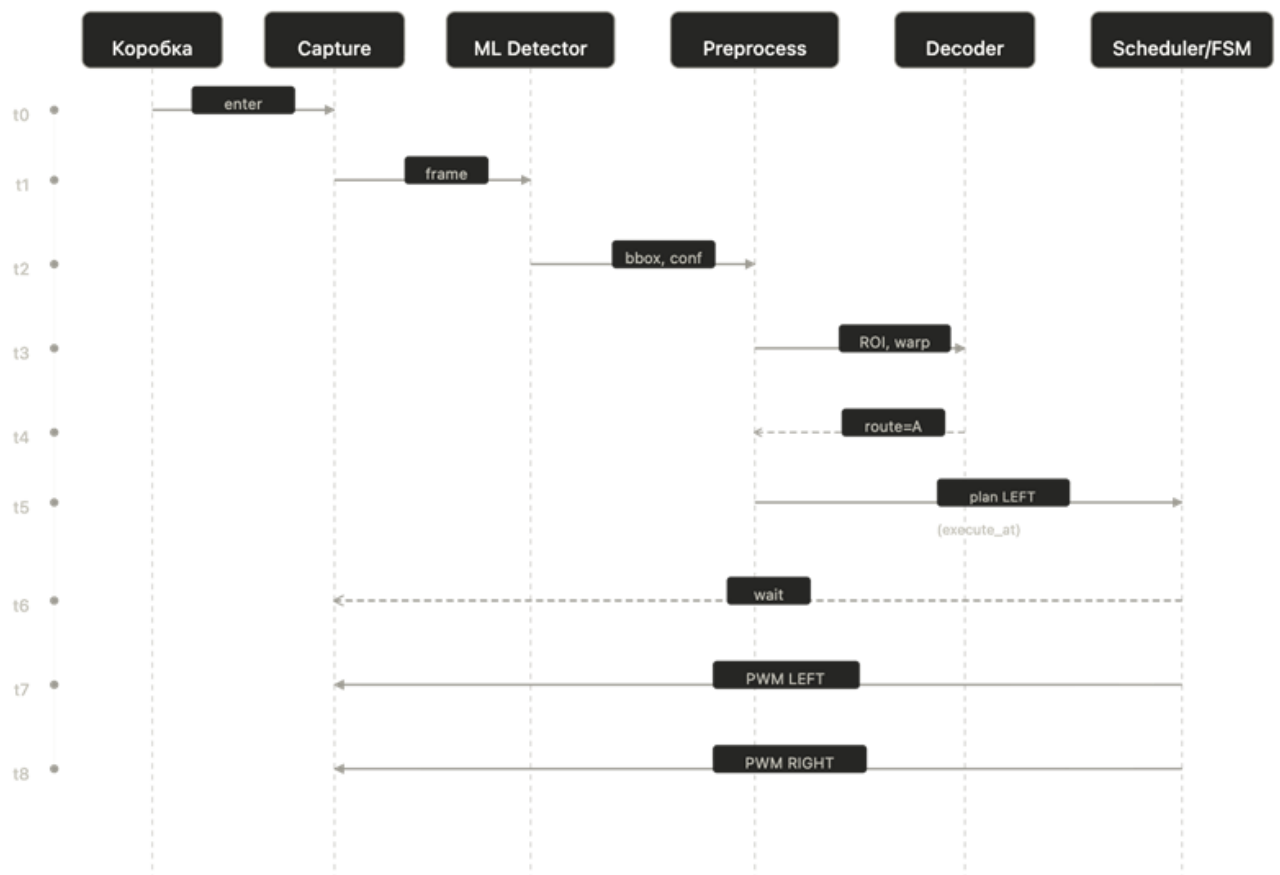


Рисунок 1.10 – Діаграма послідовностей (Sequence) для сортування маршруту А

Структура класів і даних у програмній реалізації пропонується такою, щоб відокремити “дані про подію” від “виконання”, а також забезпечити відтворюваність експериментів за рахунок стандартизованого журналу.

На рисунку 1.11 наведено UML–діаграму класів де показано, що модуль детекції повертає структуру *DetectionResult*, модуль декодування повертає *DecodeResult*, а планувальник працює з об’єктами *SortEvent*, що спрощує

тестування `DetectionResult` і `DecodeResult` можна підміняти мок-даними без камери.

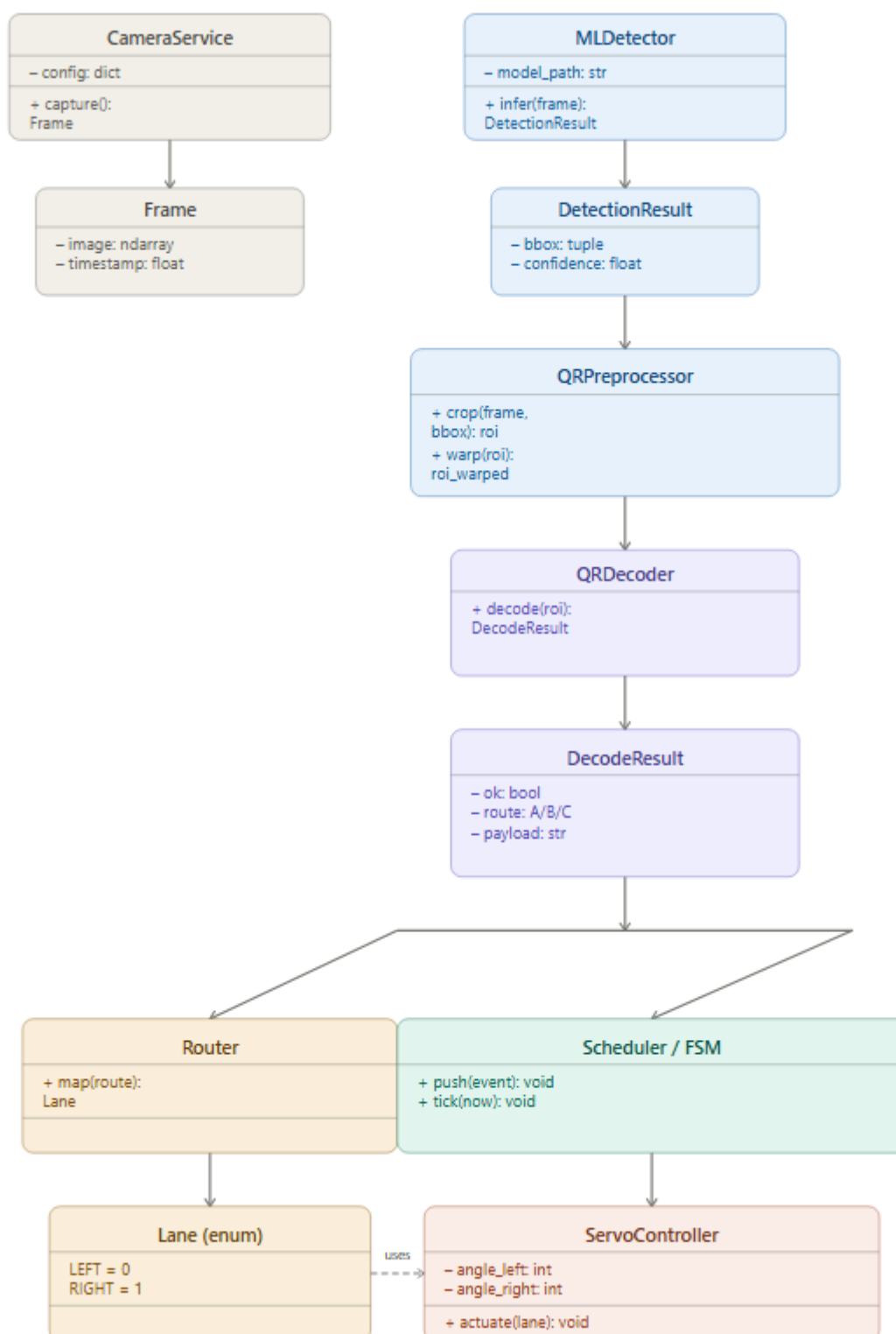


Рисунок 1.11 – Діаграма класів (Class) основних сутностей системи

Вимоги до системи, сформульовані на підставі функціональної моделі представлені у вигляді таблиці 1.6. Вони охоплюють функціональні вимоги до обробки відеопотоку, детекції та зчитування QR, керування заслінкою, ведення журналу, а також нефункціональні вимоги до затримок, продуктивності й безпечної деградації. З огляду на те, що PiCamera2 є рекомендованим інтерфейсом роботи з PiCam у стеку libcamera, у вимогах фіксується застосування цього інтерфейсу та контроль параметрів зйомки. Також фіксується застосування перспективної корекції OpenCV як етапу підвищення якості декодування, а для декодера – використання бібліотеки, що працює з зображеннями OpenCV і підтримує QR (наприклад pyzbar), або застосування ZXing як контрольного інструменту у тестах.

Таблиця 1.6 – Вимоги до системи (FR/NFR) та критерії приймання

Позначення	Тип	Формулювання вимоги / критерію
FR-01	FR	система повинна захоплювати кадри з PiCam через PiCamera2/libcamera
FR-02	FR	система повинна виконувати ML-детекцію області QR на кадрі та повертати bbox і confidence
FR-03	FR	система повинна виконувати виділення ROI та корекцію перспективи (гомографія) перед декодуванням
FR-04	FR	система повинна декодувати QR та витягувати маршрут A/B/C, визначений у payload
FR-05	FR	система повинна відображати маршрут A/B/C у два лотки за фіксованим правилом
FR-06	FR	система повинна керувати заслінкою на серво через PWM GPIO у подієвому режимі із затримкою delay_gate
FR-07	FR	система повинна вести журнал: час, route, lane, статус (OK/FAIL), confidence, latency_total, причина FAIL
FR-08	FR	система повинна мати fail-safe: при FAIL декодування або відсутності QR спрямовувати коробку в базовий лоток (RIGHT) і логувати REJECT
NFR-01	NFR	середня end-to-end затримка (latency_total) не повинна перевищувати заданого порогу T (встановлюється експериментально)
NFR-02	NFR	продуктивність повинна забезпечувати не менше N коробок/хв на стенді при заданій швидкості руху

Продовження таблиці 1.6

Позначення	Тип	Формулювання вимоги / критерію
NFR-03	NFR	точність сортування (sorting accuracy) повинна бути не нижче Y% у контрольному прогоні ($N \geq 100$)
NFR-04	NFR	частка успішного читання QR (read rate) повинна бути не нижче X% у контрольному прогоні
NFR-05	NFR	система повинна переходити у безпечний режим при відмовах камери/серво (припинення керування або базовий стан RIGHT)

Для забезпечення відтворюваності експериментів і зручності аналізу відмов, формат журналу подій визначається окремо як інтерфейс між “онлайн роботою” та “офлайн аналізом”. У журналі доцільно зберігати як технологічні показники (затримки, confidence), так і ідентифікаційні (route, lane) та діагностичні (код помилки). Формат журналу подано у (табл. 1.7), який може реалізовуватися як CSV або SQLite; з точки зору диплома, важливим є сам контракт даних, а не конкретна СУБД.

Таблиця 1.7 – Формат запису журналу сортування (лог події)

Поле	Тип	Опис
timestamp	datetime/ epoch	час події (початок обробки або момент рішення)
frame_id	int	порядковий номер кадру/події
route	char	значення A/B/C або NULL при FAIL
lane	enum	LEFT або RIGHT
status	enum	OK або FAIL
fail_reason	string	no_detection / decode_error / invalid_route / timeout
det_confidence	float	confidence детектора QR
t_capture_ms	float	час захоплення кадру
t_infer_ms	float	час інференсу ML
t_decode_ms	float	час корекції/декодування
latency_total_ms	float	сумарна end-to-end затримка
execute_at_ms	float	запланований момент актуатора (відносно timestamp)

Реалізація планувальника подій і керування заслінкою описується як програмний фрагмент, оскільки саме він забезпечує узгодження між обчислювальною частиною (ML+decode) і фізикою руху (відстань/швидкість).

Концептуально алгоритм передбачає, що після отримання route та визначення lane формується SortEvent із міткою часу виконання, а далі окремий цикл або таймер здійснює tick() і, коли поточний час перевищує execute_at, подає команду на серво: перевести заслінку в положення LEFT на час t_open та повернути в RIGHT. Оскільки OpenCV-етап перспективної корекції базується на гомографії, а робота з камерою – на PiCamera2, у лістингу 1.3 відображено лише логіку планування та актуатора, що є незалежною від конкретних бібліотек захоплення та декодування.

Лістинг 1.3 – Псевдокод планувальника подій та керування заслінкою

```
on_new_box(frame):
    t0 = now()
    det = ml_detect(frame) -> (found, bbox, conf)
    if not found:
        log(status=FAIL, reason=no_detection, lane=RIGHT, conf=conf,
latency=now()-t0)
        return

    roi = crop(frame, bbox)
    roi2 = perspective_warp(roi) //
гомографія/warpPerspective
    dec = qr_decode(roi2) -> (ok, route)

    if not ok or route not in {A,B,C}:
        lane = RIGHT // fail-safe
        log(status=FAIL, reason=decode_error, lane=lane, conf=conf,
latency=now()-t0)
    else:
        lane = (route == A) ? LEFT : RIGHT
        log(status=OK, route=route, lane=lane, conf=conf,
latency=now()-t0)

    event.execute_at = t0 + delay_gate
    event.lane = lane
    scheduler.push(event)

scheduler_tick():
    while scheduler.peek().execute_at <= now():
        ev = scheduler.pop()
        if ev.lane == LEFT:
```

```

servo_set(angle_left)
sleep(t_open)
servo_set(angle_right)          // базовий стан
else:
    // нічого не робити: базовий стан RIGHT
    continue

```

З огляду на два можливих типи сервоприводу [16], що використовуються у стендах (SG90 або MG996R), вибір серво визначається моментом навантаження заслінки та вимогами до надійності, а також окремим живленням серво із спільною “землею” з Raspberry Pi. Це пояснюється як інженерний захід стабілізації живлення та зменшення ризику перезавантажень контролера під час пікових струмів. Важливо, що вимоги до нейромережевого модуля формулюються не тільки через метрики детекції (precision/recall, mAP), але й через системні метрики read rate та sorting accuracy, які є кінцевими показниками якості роботи комплексу. Навіть при високій точності локалізації можливі помилки сортування через некоректний delay_gate або недостатній t_open, тому в експериментальній частині робота планується як серія калібрувань, де спочатку вимірюється швидкість руху коробок і відстань від зони зйомки до заслінки. Далі оцінюється середня затримка обробки кадру, і на основі цього підбираються delay_gate та t_open, після чого перевіряється стабільність у контрольному прогоні на кількості коробок не менше 100. У контексті оптимізації inference на edge-пристрої вказується, що використання on-device рантайму, орієнтованого на низькі затримки, дозволяє розглядати квантизацію моделі як практичний інструмент зменшення latency при прийнятному падінні точності, що прямо відповідає ідеології LiteRT/TFLite [17].

2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Вибір програмного забезпечення

Вибір програмного забезпечення для системи розумного сортування коробок на Raspberry Pi 4 виконується з урахуванням трьох визначальних чинників:

- необхідності керованого та відтворюваного захоплення кадрів з PiCam;
- потреби у швидкому виконанні нейромережі на edge-платформі з обмеженими ресурсами;
- вимоги забезпечити стабільне декодування QR після геометричної нормалізації області інтересу.

Оскільки QR-код є стандартизованою символією, що має визначені правила формування та декодування, програмна частина має гарантувати коректне витягування маршруту з payload у межах формальних вимог і якості символу.

Для роботи з камерою Raspberry Pi у сучасних версіях Raspberry Pi OS доцільно використовувати стек libcamera та бібліотеку Picamera2 як libcamera-орієнтований Python-інтерфейс, що позиціонується як заміна попереднього “legacy” стека, надає контроль над конфігурацією потоків і є типовим шляхом інтеграції PiCam у прикладні системи комп’ютерного зору [18].

Це безпосередньо підтримує вимогу повторюваності експериментів, оскільки дозволяє стабілізувати параметри зйомки (роздільну здатність, частоту, режими автоекспозиції/балансу білого) на рівні конфігурації захоплення.

Для попередньої обробки та геометричної нормалізації області QR обирається OpenCV, оскільки він містить усталені механізми перспективного перетворення (`warpPerspective`) та базові концепції гомографії для планарних об’єктів, що відповідає фізичній природі QR-етикетки на коробці та зменшує вплив перспективи на декодування.

Для виконання нейромережевого детектора на Raspberry Pi 4 у програмному стеку закладається on-device inference рантайм LiteRT (еволюція TensorFlow Lite), який орієнтований на низьку латентність, пропонує типовий шлях розгортання моделей у форматі .tflite та підтримує оптимізаційні механізми (зокрема квантизацію), що є важливим для досягнення необхідної швидкодії на edge-пристроях.

Декодування QR у Python-частині доцільно реалізувати через pyzbar як обгортку над zbar, оскільки вона підтримує вхідні дані у форматі numpy ndarray (типово отримані через OpenCV), забезпечує декодування QR та повертає також геометричні атрибути символу (обмежувальний прямокутник/полігон), що корисно для діагностики та журналювання відмов [19].

Для контрольних перевірок або порівняння якості декодування в складних випадках може використовуватися ZXing як відома бібліотека обробки 1D/2D-кодів із підтримкою QR-коду, що дозволяє валідувати результати основного декодера або застосувати альтернативний шлях розпізнавання у тестовому середовищі.

На рисунку 2.1 наведено узагальнену схему програмного стека і точок взаємодії, де показано, що PiCamera2 формує кадри та часові мітки, LiteRT/TFLite виконує inference детектора, OpenCV виконує crop/warp ROI, pyzbar (або контрольний ZXing) виконує декодування, а модулі маршрутизації й планування команд перетворюють результат у PWM-керування сервоприводом.

Таким чином, стек програмного забезпечення проєктується як модульний конвеєр, де камера, нейромережа, OpenCV-нормалізація та декодер є незалежними компонентами, а верхній рівень керування (Router + Scheduler/FSM) забезпечує бізнес-логіку маршрутизації A/B/C у два лотки та часову синхронізацію із серво-заслінкою.

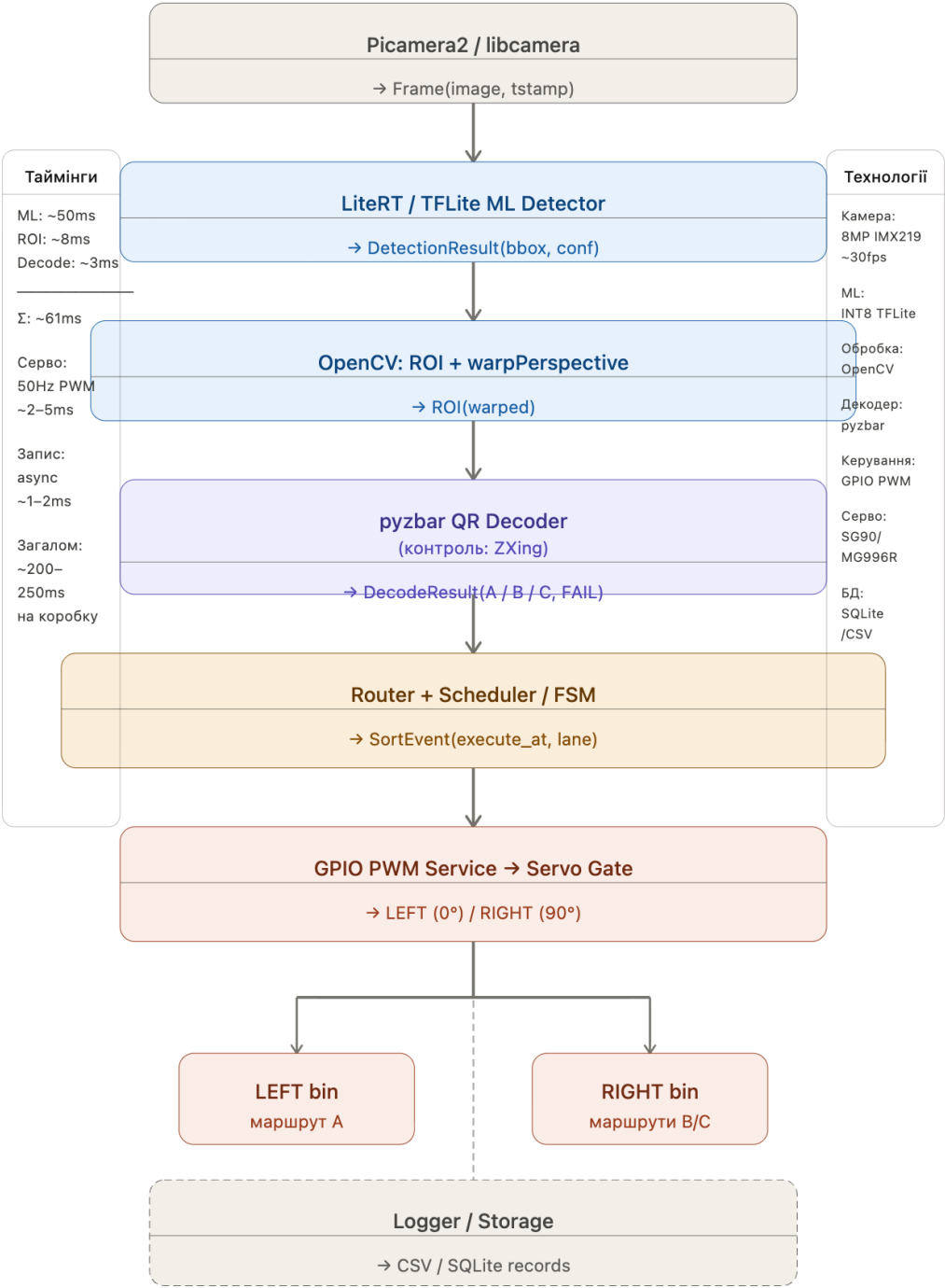


Рисунок 2.1 – Програмний стек системи та взаємодія модулів

В таблиці 2.1 наведено вибір програмних компонентів за їх критеріями сумісності з Raspberry Pi 4, можливості роботи з PiCam, продуктивності на edge та придатності до діагностики.

Таблиця 2.1 – Вибір програмного забезпечення: кандидати та обґрунтування

Компонент	Варіант 1 (обрано)	Варіант 2 (альтернатива)	Обґрунтування вибору
Захоплення кадру	Picamera2/ libcamera	інші/legacy підходи	сучасний шлях для PiCam, кероване захоплення
CV-обробка	OpenCV (warp/ homography)	самописні перетворення	стандартні та документовані перспективні операції
ML рантайм	LiteRT/TFLite (.tflite)	інші runtime	орієнтація на on-device і оптимізації для latency
QR декодування	pyzbar (zbar)	ZXing як контроль	простий Python- інтерфейс + геометрія; контроль альтернативою
Стандартизація QR	інтерпретація згідно ISO/IEC 18004	—	QR як формалізований носії маршруту
Журналювання	CSV/SQLite (контракт даних)	—	відтворюваність експериментів і аналіз відмов

Щоб узгодити вибрані програмні компоненти з модульною структурою системи, у проєкті фіксуються інтерфейси (контракти) між підсистемами у вигляді сутностей даних, що дозволяє замінювати реалізації без зміни бізнес-логіки, наприклад міняти декодер або формат моделі.

Для практичної реалізації необхідно також формалізувати мінімальний набір конфігураційних параметрів, які напряду пов'язані з вибраним ПЗ: параметри камери (роздільна здатність, fps), параметри inference (шлях до моделі, поріг confidence), параметри перспективної нормалізації (розмір ROI/цільова сітка) і параметри керування заслінкою (кути та таймінги).

Шаблон конфігурації який показує, що вибране ПЗ підтримує параметризацію а отже забезпечує відтворюваність експериментів та зручність налагодження наведено у лістингу 2.1.

Лістинг 2.1 – Узагальнений шаблон конфігурації програмних модулів

```
camera: picamera2/libcamera; resolution; fps; exposure/awb режим;
ml: litert/tflite; model_path; conf_threshold;
cv: opencv; roi_size; warp_target_size;
decoder: pyzbar; fallback_decoder: zxing (optional);
routing: A->LEFT; B->RIGHT; C->RIGHT; FAIL->RIGHT;
timing: delay_gate_ms; t_open_ms;
servo: angle_left; angle_right; pwm_frequency;
logging: csv/sqlite; fields; retention;
```

Обране програмного забезпечення забезпечує відповідність постановці задачі та обмеженням Raspberry Pi 4, а саме:

- icamera2/libcamera дає контрольоване отримання кадру з PiCam;
- OpenCV реалізує перспективну нормалізацію ROI через документовані апарат гомографії;
- LiteRT/TFLite дає виконання неймережі на edge з орієнтацією на продуктивність;
- pyzbar забезпечує практичне декодування QR у Python-потіці з підтримкою OpenCV-масивів;
- ZXing може використовуватись як контрольний інструмент або альтернативний шлях декодування;
- стандарт QR закріплює коректність інтерпретації даних маршруту.

2.2 Вибір апаратного забезпечення

Вибір апаратного забезпечення для системи розумного сортування коробок за маршрутом, закодованим у QR, визначається потребою виконувати неймережеву локалізацію QR та подальше декодування безпосередньо на edge-платформі, забезпечити стабільне та відтворюване захоплення кадрів камерою, а також реалізувати надійне керування серво-заслінкою у потоці рухомих коробок [20].

Як обчислювальну платформу обрано Raspberry Pi 4 Model B, оскільки він має MIPI CSI-порт для підключення камери, стандартний 40-контактний GPIO для керування виконавчими пристроями, а також офіційно рекомендоване живлення 5V через USB-C з мінімальним струмом 3A, що безпосередньо впливає на стабільність роботи при підключенні периферії та виконанні обчислень. Камерний модуль у цьому проєкті фіксується як Raspberry Pi Camera Module 2 із сенсором Sony IMX219 (8 Мп), який підключається шлейфом до CSI та підтримує відеорежими, достатні для задач комп'ютерного зору в лабораторному стенді, включно з 1080p та підвищеними частотами кадрів на нижчих роздільних здатностях; це важливо, оскільки підвищення частоти зйомки зменшує ймовірність втрати різкості на рухомих об'єктах і дозволяє точніше “зловити” QR у кадрі.

Виконавча частина реалізується механізмом заслінки, який змінює траєкторію коробки у два лотки, тому ключовим елементом є сервопривід. У межах проєкту розглядаються два типові серво:

- SG90 як мікросерво з відносно малим крутним моментом (порядку 1.8 кг·см при 4.8V), що підходить лише для легких заслінок і низьких механічних навантажень;
- MG996R як сервопривід підвищеного моменту з металевим редуктором (порядку 9.4 кг·см при 4.8V та до ≈ 11 кг·см при 6V), який є більш придатним для заслінки, що контактує з рухомою коробкою.

При цьому для MG996R наводяться значення струму зупинки (stall current) близько 2.5A при 6V, що прямо вказує на необхідність окремого живлення серво та запасу по струму. Керування стандартними RC-сервоприводами здійснюється PWM-сигналом із типовою частотою близько 50 Гц, де кут визначається шириною імпульсу; отже, апаратно достатньо одного GPIO-контакту для сигнальної лінії, але обов'язковою є спільна “земля” між Raspberry Pi та джерелом живлення серво, щоб рівні сигналу PWM були коректно визначені відносно одного потенціалу.

Апаратна архітектура стенду, у якій камера Module 2 підключається до CSI-порту Raspberry Pi 4, а серво підключається до одного з GPIO (сигнал PWM) та до окремого джерела 5–6V. Таке розділення живлення є принциповим, оскільки Raspberry Pi 4 має власні вимоги до вхідного живлення 5V/3A, а пікові навантаження серво (особливо MG996R) можуть спричиняти просідання напруги і нестабільність роботи при живленні з одного джерела без достатнього запасу

Враховуючи, що коробки рухаються, до апаратної частини доцільно додати контрольоване освітлення, а саме LED-підсвітку з розсіювачем, як засіб підвищення стабільності кадру і зменшення впливу відблисків та тіней на QR, а також (за можливості) датчик присутності як тригер моменту t_0 для підвищення точності синхронізації керування заслінкою. Однак навіть без датчика система залишається реалізовною, хоча вимоги до калібрування таймінгів стають жорсткішими через більшу невизначеність моменту “спостереження” коробки в кадрі. Відповідність апаратних компонентів задачі та їх роль у системі узагальнено в таблиці 2.2, де Raspberry Pi 4 визначає обчислювальні та інтерфейсні можливості, Pi Camera Module 2 формує первинні дані для ML і декодування, а серво визначає надійність актуатора; при цьому обґрунтування окремого живлення серво прямо впливає з наведених електричних характеристик MG996R і рекомендацій щодо живлення Raspberry Pi 4.

Щоб формально узгодити апаратну конфігурацію з часовими вимогами рухомої лінії, апаратно-програмний контур керування подається як структура “сенсор → обчислення → актуатор”, де камера Module 2 є сенсором, Raspberry Pi4 – обчислювальним ядром, а серво-заслінка – актуатором; у такій структурі апаратна частина повинна забезпечити стабільність живлення і повторювану геометрію (фіксовану відстань між зоною зйомки та заслінкою), і саме ці умови визначають відтворюваність подальших експериментів.

Таблиця 2.2 – Склад апаратної частини (обрані компоненти) та обґрунтування вибору

Компонент	Обраний варіант	Роль у системі	Обґрунтування
Обчислювальна платформа	Raspberry Pi 4 Model B	edge-обробка та GPIO	CSI для камери, GPIO для PWM, живлення 5V/3A
Камера	Raspberry Pi Camera Module 2 (IMX219)	кадри для CV/ML	8 Мп, відеорежими, CSI підключення
Сервопривід (рекоменд.)	MG996R	привід заслінки	високий момент, метал редуктор, stall ~2.5A @6V
Сервопривід (альт.)	SG90	легка заслінка	малий момент ~1.8 кг·см @4.8V, тільки для малих навантажень
Живлення Raspberry Pi	5.1V/≥3A (USB-C)	стабільність системи	рекомендований мінімум струму для RPi 4
Живлення серво	5–6V, окреме, із запасом по струму	пікові навантаження	MG996R має високий піковий/зупинковий струм
Керування серво	GPIO PWM, частота ~50 Hz	формування імпульсів	типова схема керування RC-серво PWM 50Hz

3 РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

3.1 Проєктування програмної архітектури та взаємодії модулів

Проєктування програмної архітектури (рис.3.1) виконується як побудова модульного конвеєра обробки, у якому кожен компонент має чітко визначені входи й виходи, а взаємодія між компонентами фіксується контрактами даних, що забезпечує тестованість, замінність реалізацій і відтворюваність експериментів. Архітектура орієнтована на Raspberry Pi 4 як edge-платформу, де захоплення кадрів здійснюється через Picamera2 як libcamera-based Python інтерфейс для камер Raspberry Pi.

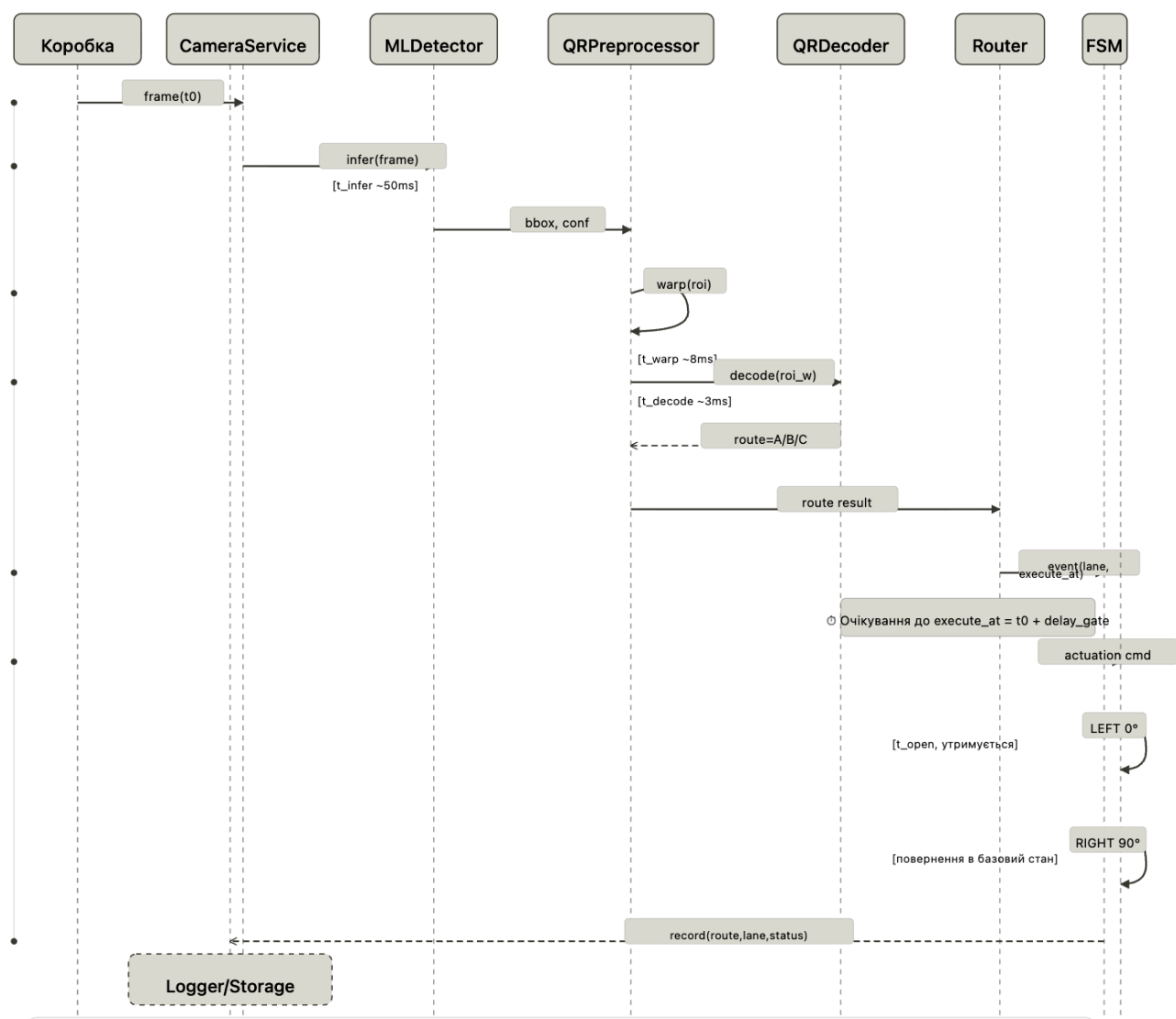


Рисунок 3.1 – Компонентна архітектура програмного забезпечення та потоки даних

Інференс нейромережі виконується on-device рантаймом LiteRT/TFLite, який позиціонується як високопродуктивний фреймворк для виконання моделей на пристроях з оптимізаціями та рекомендованими сучасними API, геометрична нормалізація області QR реалізується стандартними механізмами гомографії та перспективного перетворення OpenCV, а декодування QR виконується окремим декодером на кшталт рузбар, який прямо підтримує вхідні дані у вигляді OpenCV/numpy ndarray. Додатково у тестовому режимі допускається використання ZXing як еталонного/контрольного декодера, оскільки він є мультиматною бібліотекою обробки 1D/2D кодів, включно з QR [22].

На рівні структурного поділу програмна система проєктується у вигляді сервісів (або модулів) із розмежуванням відповідальностей:

- камера постачає кадри з часовими мітками;
- ML–детектор повертає координати QR у кадрі;
- препроцесор формує ROI та виконує warp для зменшення перспективних спотворень;
- декодер повертає route або помилку;
- маршрутизатор відображає A/B/C у два лотки;
- планувальник подій синхронізує керуючу дію із рухом коробки;
- контролер серво виконує PWM–керування;
- логер фіксує результати та таймінги.

Такий розподіл відповідає принципу “один модуль – одна відповідальність”, знижує зв’язність та дозволяє окремо вимірювати латентність кожної стадії, що важливо для оцінювання продуктивності на edge–пристрої при виконанні inference LiteRT/TFLite і при роботі з камерним стеком Picamera2/libcamera.

Взаємодія модулів часі проєктується як подієвий сценарій, де рішення про лоток формується після декодування, але фізична дія відбувається у запланований момент, що відокремлює “час обчислення” від “часу актуатора” і дозволяє стабільно працювати у потоці. Таке розділення особливо важливе на Raspberry Pi 4, де варіативність часу inference і декодування може змінюватися залежно від навантаження, а планувальник подій компенсує ці флуктуації, використовуючи калібрований параметр `delay_gate`. У цьому контексті камера керується як джерело кадру, де Picamera2 формує послідовність кадрів та підтримує базові патерни захоплення у Python, після чого ML–рантайм LiteRT/TFLite виконує inference детектора, а OpenCV виконує перспективну нормалізацію (гомографію) перед декодуванням.

Інтерфейси між модулями проєктуються як прості структури даних, що передаються без “прихованих” залежностей (табл.3.1):

- Frame містить зображення та часову мітку;
- DetectionResult містить bbox і confidence;

- ROI містить нормалізоване зображення QR та, за потреби, контрольні точки;
- DecodeResult містить ok/route/payload;
- SortEvent містить lane та execute_at;
- LogRecord містить значення для оцінювання якості.

Вибір таких контрактів також прямо узгоджується з можливостями ruzbar, який повертає не лише дані, а й геометричні атрибути декодування та приймає на вході numpy/OpenCV масиви, і з тим, що OpenCV стандартно оперує матрицями зображень для трансформацій

Таблиця 3.1 – Контракти даних між модулями (інтерфейси обміну)

Сутність	Поля (мінімум)	Джерело → Споживач
Frame	image, timestamp, frame_id	CameraService → MLDetector
DetectionResult	bbox(x,y,w,h), confidence, det_time_ms	MLDetector → QRPreprocessor
ROI	image_roi_warped, roi_bbox, warp_info(optional)	QRPreprocessor → QRDecoder
DecodeResult	ok, route(A/B/C), payload, decode_time_ms	QRDecoder → Router
Lane	LEFT/RIGHT	Router → Scheduler/FSM
SortEvent	lane, execute_at, t_open, source_frame_id	Scheduler/FSM → ServoController
LogRecord	timestamp, route, lane, status, fail_reason, timings, conf	Any → Logger/Storage

З огляду на необхідність експериментальної перевірки метрик (read rate, sorting accuracy, latency, throughput), проектування передбачає стандартний формат журналу, який дозволяє автоматизовано формувати таблиці й графіки та знаходити причини помилок. Оскільки ruzbar декодує та повертає структури з типом коду і координатами, у лог доцільно включати confidence детектора та причину fail (no_detection, decode_error тощо) для аналізу, а також розкладку таймінгів, щоб окремо оцінити внесок inference LiteRT/TFLite і геометричних перетворень OpenCV (табл.3.2).

Таблиця 3.2 Формат лог-запису для оцінювання якості роботи системи

Поле	Опис
timestamp	час події t0 (момент кадру/детекції)
frame_id	ідентифікатор кадру/події
route	A/B/C або NULL
lane	LEFT/RIGHT
status	OK/FAIL
fail_reason	no_detection / decode_error / invalid_route / timing_error
det_confidence	confidence від MLDetector
t_capture_ms	час захоплення кадру (за потреби)
t_infer_ms	час інференсу моделі
t_warp_ms	час ROI+warpPerspective
t_decode_ms	час декодування QR
latency_total_ms	t_infer_ms + t_warp_ms + t_decode_ms (+ накладні витрати)
execute_at_ms	запланований момент актуатора (t0 + delay_gate)

Для опису внутрішньої структури ПЗ та взаємозв'язків між класами було створено UML-діаграму класів, де видно, що модулі залежать не один від одного напряму, а від контрактів даних, що зменшує зв'язність і підвищує замінність реалізацій, наприклад, заміна декодера ruzbar на ZXing у тестовому режимі не змінює Router і Scheduler. Така заміна є виправданою, бо ruzbar прямо підтримує роботу з OpenCV/numpy масивами, а ZXing є окремою незалежною бібліотекою для QR та інших форматів.

Поведінка системи у потоці коробок проєктується як кінцевий автомат станів (FSM), що уніфікує обробку штатних і аварійних ситуацій та мінімізує “розсіпані” перевірки по коду; ключовим у цьому автоматі є те, що після прийняття рішення система переходить у стан очікування планової дії execute_at, а при відмовах переводить систему у fail-safe сценарій (базовий лоток) і фіксує помилку у журналі.

Наведено спрощену FSM-модель, яка одночасно відображає і логіку, і точки, де зручно вимірювати час виконання стадій [23].

Окремим аспектом проєктування є організація виконання у часі та конкурентність, оскільки робота з камерою та inference можуть виконуватися у різних потоках або через черги, щоб уникнути блокувань і втрат кадрів. Pícamera2

підтримує основні операції захоплення та внутрішні механізми доставки кадрів у Python–додаток, а LiteRT/TFLite орієнтований на оптимізоване виконання моделей на пристрої з можливістю використовувати сучасні runtime–інтерфейси, тому практично доцільно проєктувати “подію сортування” як атомарний об’єкт, який проходить конвеєром, а планувальник подій обробляє чергу SortEvent незалежно від того, як саме отримано кадр (лістинг 3.1).

Лістинг 3.1 – Узагальнений псевдокод обробки подій та взаємодії модулів

```
while system_running:
    frame = camera.capture()                # Frame(image, t0,
frame_id) [Picamera2]
    det = detector.infer(frame)              # DetectionResult(bbox,
conf, t_infer_ms) [LiteRT/TFLite]
    if not det.bbox:
        router.fail_safe(frame, reason="no_detection") # lane=RIGHT
        continue
    roi = preprocessor.warp(frame, det)       # ROI(warped_image,
t_warp_ms) [OpenCV]
    dec = decoder.decode(roi)                # DecodeResult(ok,
route, t_decode_ms) [pyzbar/ZXing]
    lane = router.map(dec)                   # LEFT if route==A else
RIGHT
    scheduler.push(SortEvent(lane, execute_at=t0+delay_gate,
t_open=t_open))
    scheduler.tick(now())                   # якщо настав час
execute_at -> ServoController.actuate()
    logger.write(LogRecord(...timings..., ...status...))
..
```

Щоб завершити проєктування і забезпечити практичну реалізацію, структура репозиторію та межі модулів фіксуються як частина архітектури, оскільки вона впливає на супровідність коду і якість документування. У таблиці 3.3 наведено рекомендовану структуру директорій, де розділено код захоплення, ML, CV–препроцес, декодер, керування, конфігурацію та тести, а також виділено місце для моделей і логів.

Таблиця 3.3 – Рекомендована структура проекту (репозиторій)

/ src	
/ camera	CameraService, конфігурації Picamera2
/ ml	MLDetector, завантаження .tflite/.litert, пороги
/ cv	ROI, warpPerspective, фільтри
/ decode	pyzbar декодер, опційно ZXing wrapper
/ control	Scheduler/FSM, ServoController, PWM
/ routing	Router, правила A/B/C → LEFT/RIGHT, fail-safe)
/ logging	Logger, формати CSV/SQLite, метрики
/ utils	таймери, типи, спільні структури даних
models	qr_detector.tflite
/ config	yaml/json конфігурації camera/ml/timing/servo
tests	модульні тести контрактів і “сухі” прогони
docs	схеми, діаграми, опис параметрів
logs	результати експериментів

Розроблена архітектура програмного забезпечення для системи розумного сортування коробок на базі Raspberry Pi 4 формує цілісну інфраструктуру (рис.3.2), у якій кожен компонент виконує чітко визначену функцію. Використання стеку libcamera/Picamera2 забезпечує кероване захоплення кадрів із камери Raspberry Pi, що є базовим етапом обробки даних.

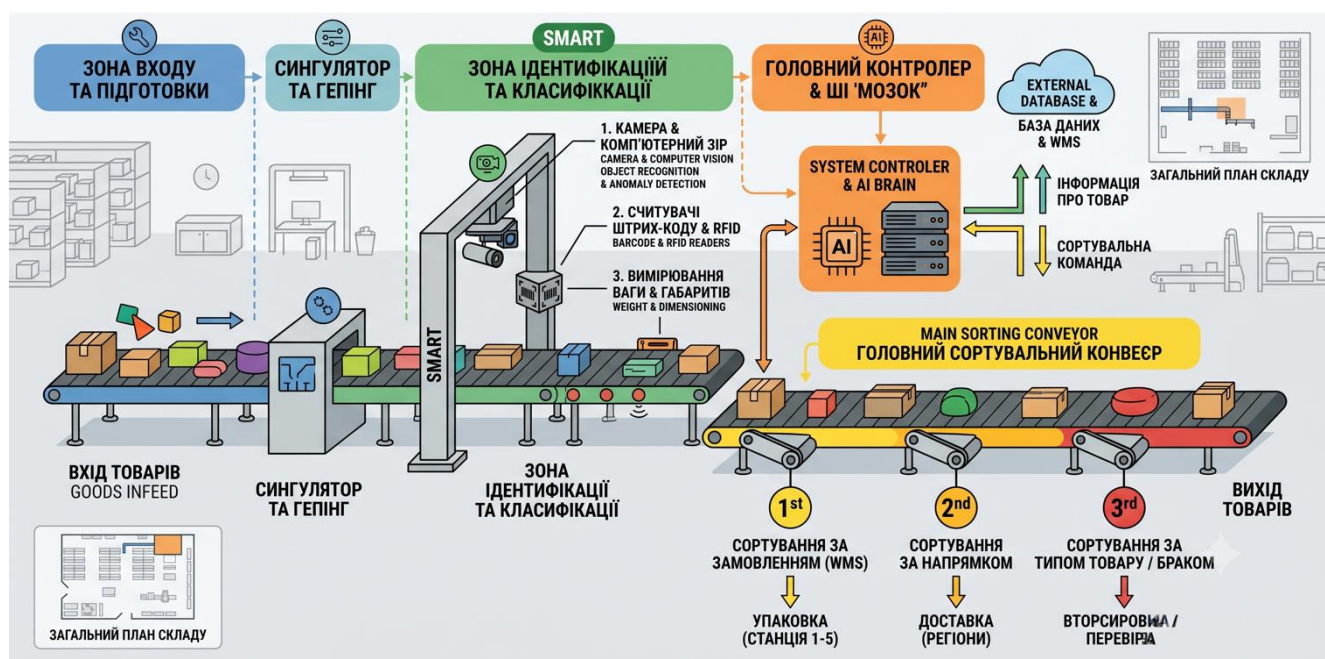


Рисунок 3.2 – Схематична модель розумної системи сортування товарів

Для геометричної нормалізації області QR застосовується OpenCV з механізмами перспективного перетворення та гомографії, що дозволяє мінімізувати вплив нахилу та деформацій на точність декодування.

Декодування QR-кодів у Python-частині реалізовано через pyzbar як обгортку над zbar, яка безпосередньо підтримує формат numpy ndarray, що забезпечує інтеграцію з OpenCV. Для контрольних перевірок у складних випадках використовується ZXing як еталонний мультиформатний декодер. Інференс нейромережі виконується безпосередньо на пристрої за допомогою LiteRT/TFLite, що гарантує високу продуктивність та оптимізацію під edge-платформи.

Таким чином, система орієнтована на роботу в реальному часі, поєднує сучасні бібліотеки та фреймворки, а її архітектура є масштабованою, стандартизованою та придатною для подальшого розвитку. Це забезпечує ефективне та надійне сортування коробок із використанням QR-ідентифікації, що відповідає сучасним вимогам до інтелектуальних систем автоматизації.

3.2 Проєктування системи

Проєктування системи розумного сортування коробок на Raspberry Pi 4 виконується як побудова узгодженого апаратно-програмного комплексу, у якому інформаційний контур зчитування маршруту з QR пов'язаний із керуючим контуром керування серво-заслінкою з урахуванням руху коробки [24]. Базовою передумовою є використання QR як стандартизованого носія даних, для якого визначено правила кодування, вимоги до якості та еталонні принципи декодування, а тому проєктне рішення спрямоване не на “розпізнавання маршруту” нейромережею, а на підвищення надійності локалізації QR у кадрі та стабілізацію умов декодування через ROI і перспективну нормалізацію.

Для PiCam доцільно застосувати стек libcamera з бібліотекою Picamera2 як сучасним інтерфейсом захоплення кадрів і керування параметрами зйомки, що забезпечує відтворюваність експериментів і стабільність вхідного потоку даних.

Геометрична нормалізація області QR проєктується через планарну гомографію та перспективне перетворення, оскільки QR-етикетка є майже пласкою, а OpenCV надає стандартні підходи і функції для отримання перспективного перетворення і його застосування до ROI, що зменшує вплив нахилу та спотворень на декодування.

Для виконання неймережі на edge-пристрої обирається on-device рантайм на кшталт LiteRT (еволюція TensorFlow Lite), що орієнтований на низьку латентність і підтримує оптимізації, включно з квантизацією, а отже відповідає вимозі запуску ML на Raspberry Pi 4 без надлишкового навантаження.

Декодування QR у Python-частині проєктується як окремий модуль на базі бібліотеки ruozbar, яка працює з ndarray OpenCV і повертає декодовані дані разом з геометрією символу, що корисно для діагностики та легування. Як контрольний варіант для валідації результатів може бути використаний ZXing як відома бібліотека для QR та інших штрихкодів.

Механічна частина проєктується як найпростіший divert-вузол для двох напрямків, де заслінка в базовому положенні пропускає коробку в основний канал, а в активному – відхиляє у відповідний канал; такий принцип є типовим для divert/sortation рішень у конвеєрних системах і є релевантним до задачі маршрутизації

Загальна архітектура системи фіксується як набір модулів із чіткими входами/виходами та контрольними точками вимірювання затримок, що потрібно як для реалізації, так і для експериментального підтвердження критеріїв якості.

На рисунку 3.3 наведено функціональну архітектуру, де видно, що неймережа повертає лише координати області QR, далі відбувається нормалізація ROI та декодування, після чого застосовується правило відображення маршруту A/B/C у два лотки і планується спрацювання серво-заслінки на момент, синхронізований з фізичним переміщенням коробки.

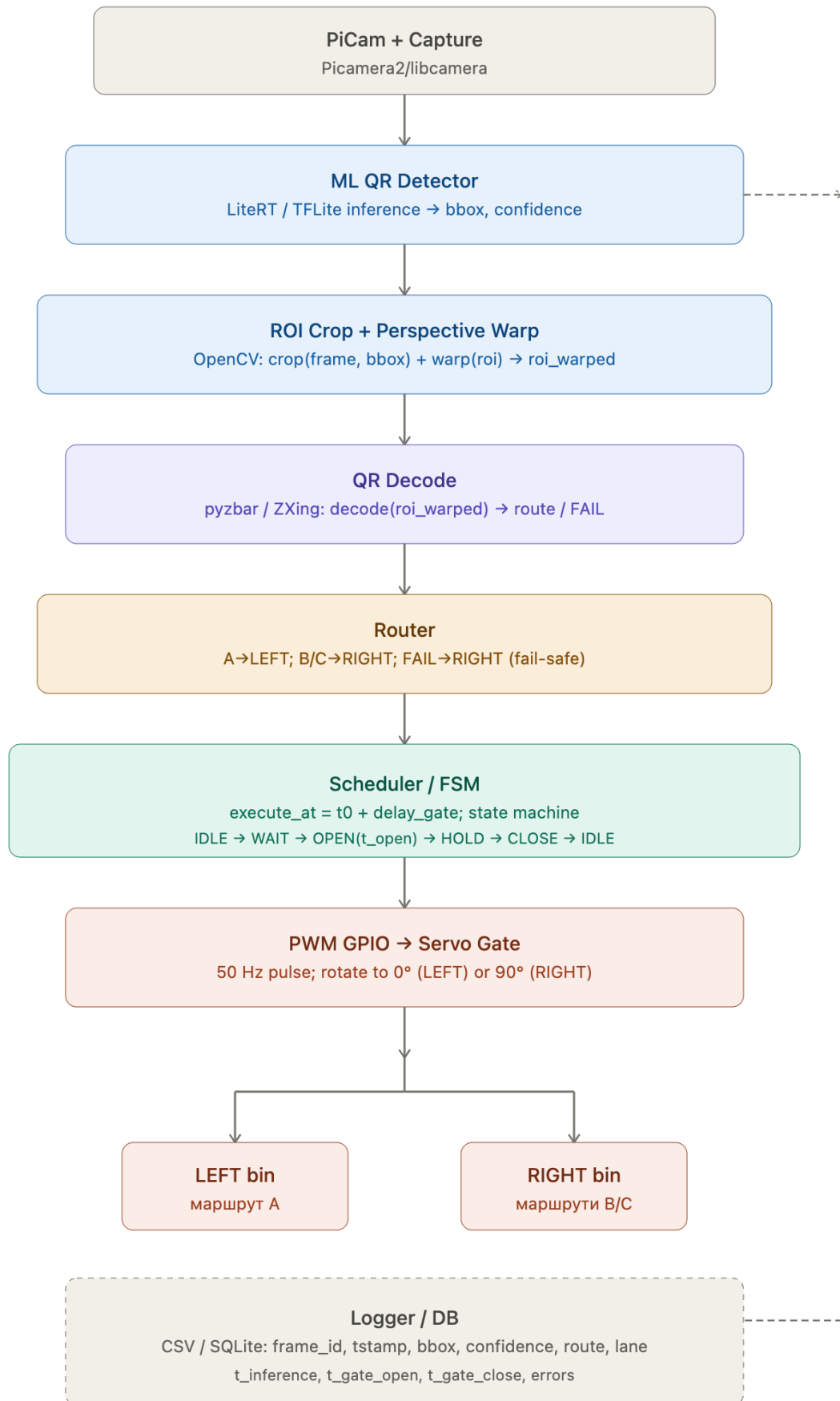


Рисунок 3.3 – Функціональна архітектура системи (дані, рішення, керування)

Розгортання компонентів на апаратній платформі визначає межі відповідальності Raspberry Pi 4 як edge-вузла, на якому одночасно виконуються захоплення кадру, ML-інференс, геометрична обробка, декодування та керування PWM. Така централізація мінімізує додаткові затримки і робить систему автономною, а вимоги до камери й зйомки реалізуються через стек Picamera2/libcamera. В діаграмі розгортання, окремо підкреслено необхідність незалежного живлення серво з обов'язковою спільною “землею” з Raspberry Pi як інженерний аспект забезпечення стабільності роботи.

Ключовою проектною особливістю для рухомих коробок є часовий зв'язок між моментом отримання інформації (t_0) і моментом фізичного відхилення коробки заслінкою, тому модуль планування подій проектується як окрема підсистема, що приймає рішення (`lane`) і обчислює час виконання (`execute_at`) за каліброваним параметром `delay_gate`, а також визначає тривалість утримання заслінки в активному стані `t_open`. Такий підхід дозволяє уникнути “миттєвого” керування одразу після декодування, яке було б некоректним у потоці, і формально описати керування через подієву модель. Послідовність взаємодії між компонентами містить як етапи обробки (інференс, `warp`, декод), так і етап відкладеного актуатора.

Проектування структури програмних модулів виконується з акцентом на тестованість і замінність компонентів, тому їх інтерфейси фіксуються як контракти даних [25]. Камера повертає кадр із часовою міткою, детектор повертає прямокутник `bbox` і `confidence`, препроцесор повертає нормалізоване ROI, декодер повертає `route` або помилку, маршрутизатор перетворює `route` у `lane`, а планувальник повертає команди серво з урахуванням часу. У вигляді діаграми класів це описується як набір сутностей, де центральною сутністю є `SortEvent`, що переносить рішення у часі.

Щоб специфікувати модулі без надлишкової деталізації коду, доцільно подати таблицю відповідальностей (табл. 3.4), у якій кожному модулю призначено роль, вхід і вихід, а також ключові показники, які логуються для подальшої оцінки. Перспективна нормалізація ґрунтується на інструментах

OpenCV, інференс моделі виконується on-device рантаймом LiteRT/TFLite, захоплення кадру реалізується через Picamera2/libcamera, а декодування виконується pyzbar/ZXing.

Таблиця 3.4 – Модулі системи та їх відповідальності (вхід/вихід)

Модуль	Вхід	Вихід	Призначення
CameraService	налаштування зйомки	Frame(image,tstamp)	захоплення кадрів з PiCam
MLDetector	Frame	DetectionResult(bbox,conf)	локалізація QR на кадрі
QRPreprocesso	Frame + DetectionResult	ROI(warped_image)	crop + perspective warp
QRDecoder	ROI	DecodeResult(ok,route,payload)	декодування QR
Router	DecodeResult	Lane(LEFT/RIGHT)	мапінг A/B/C у 2 лотки
Scheduler/FSM	Lane + t0 + params	команди актуатора (події)	синхронізація з рухом коробки
ServoController	команди з Scheduler/FSM	PWM сигнал	керування серво заслінкою
Logger/Storage	події та таймінги	записи CSV/SQLite	збір метрик та причин відмов

Важливою частиною проєктування є фіксація параметрів калібрування, оскільки саме вони пов'язують програмне рішення з фізикою руху коробок. У цьому проєкті параметри `delay_gate` та `t_open` визначаються експериментально, але вже на етапі проєктування вони вводяться як конфігураційні змінні, що дозволяє змінювати поведінку системи без модифікації коду. Структура конфігурації подається у лістингу 3.2 цей шаблон параметрів відповідає принципу відтворюваності: параметри, з якими отримано експериментальні результати, можна зафіксувати і повторити.

Лістинг 3.2 – Шаблон конфігурації параметрів калібрування і керування camera

```
resolution: 1280x720
fps: 30
```

```

exposure_mode: fixed_or_locked
awb_mode: fixed_or_locked

ml:
  model_path: /models/qr_detector.tflite
  conf_threshold: 0.5

routing:
  rule: "A->LEFT; B->RIGHT; C->RIGHT; FAIL->RIGHT"

timing:
  delay_gate_ms: 450
  t_open_ms: 300

servo:
  angle_right: 90
  angle_left: 30
  pwm_frequency_hz: 50

```

Проектування таблиці параметрів (табл. 3.5) дозволяє формально визначити, які саме величини мають бути виміряні або підібрані під час калібрування, і як вони впливають на якість сортування. Окремо підкреслюється, що *latency* обробки кадру і *delay_gate* є різними величинами: *latency* впливає на те, коли стає відомим *lane*, але не визначає момент фізичного відхилення, який прив'язаний до кінематики лінії; цей зв'язок і обґрунтовує наявність планувальника подій.

Таблиця 3.5 – Параметри часової синхронізації та їх інженерний зміст

Параметр	Одиниці	Зміст
<i>t0</i>	ms	момент захоплення кадру або фіксації коробки в зоні зйомки
<i>latency_total</i>	ms	час від <i>t0</i> до отримання рішення <i>lane</i> (обробка кадру)
<i>delay_gate</i>	ms	час проходження коробки від зони зйомки до заслінки
<i>execute_at</i>	ms	момент спрацювання актуатора $execute_at = t0 + delay_gate$
<i>t_open</i>	ms	тривалість утримання заслінки у положенні LEFT

Розглянута архітектура системи формує цілісну та взаємопов'язану інфраструктуру, у якій кожен компонент виконує чітко визначену функцію. Стандартизованість QR задає єдині правила інтерпретації даних, модулі

PiCamera2/libcamera забезпечують кероване захоплення кадрів, а OpenCV реалізує перспективну нормалізацію області інтересу (ROI). Виконання моделей на edge-пристроях здійснюється за допомогою LiteRT/TFLite, тоді як pyzbar/ZXing відповідають за декодування QR-кодів. Механізм заслінки як типовий divert-елемент забезпечує перенаправлення потоку у два лотки.

Всі взаємозв'язки між модулями формалізовані через структурні схеми та таблиці контрактів, що створює прозору, стандартизовану та керовану архітектуру. Такий підхід гарантує узгодженість роботи системи, її масштабованість та можливість подальшого розвитку без порушення цілісності.

3.3 Об'єктно-орієнтоване проєктування системи

У системі розумного сортування коробок застосовується об'єктно-орієнтований підхід, що дозволяє чітко структурувати функціональні модулі у вигляді класів та їхніх взаємозв'язків, що спрощує розробку, тестування та масштабування [26]:

- CameraController – відповідає за роботу з камерою Raspberry Pi через PiCamera2/libcamera, має методи для ініціалізації, захоплення кадрів та налаштування параметрів;
- ImageProcessor – виконує попередню обробку зображень, нормалізацію області QR, перспективні перетворення та гомографію;
- QRDetector – реалізує локалізацію QR-кодів у кадрі за допомогою нейромережі, оцінює якість детекції та експортує edge-модель;
- QRDecoder – відповідає за декодування QR-кодів із використанням pyzbar та додатково ZXing для контрольних перевірок;
- GateController – керує заслінкою через сервопривід MG996R, реалізує відкриття, закриття та перенаправлення потоку;

– SortingSystem – інтегрує всі попередні модулі, забезпечує обробку коробки, визначення маршруту та логування результатів.

Структуру та зв'язки між об'єктами наведено на рисунку 3.4 у UML–діаграмі класів.

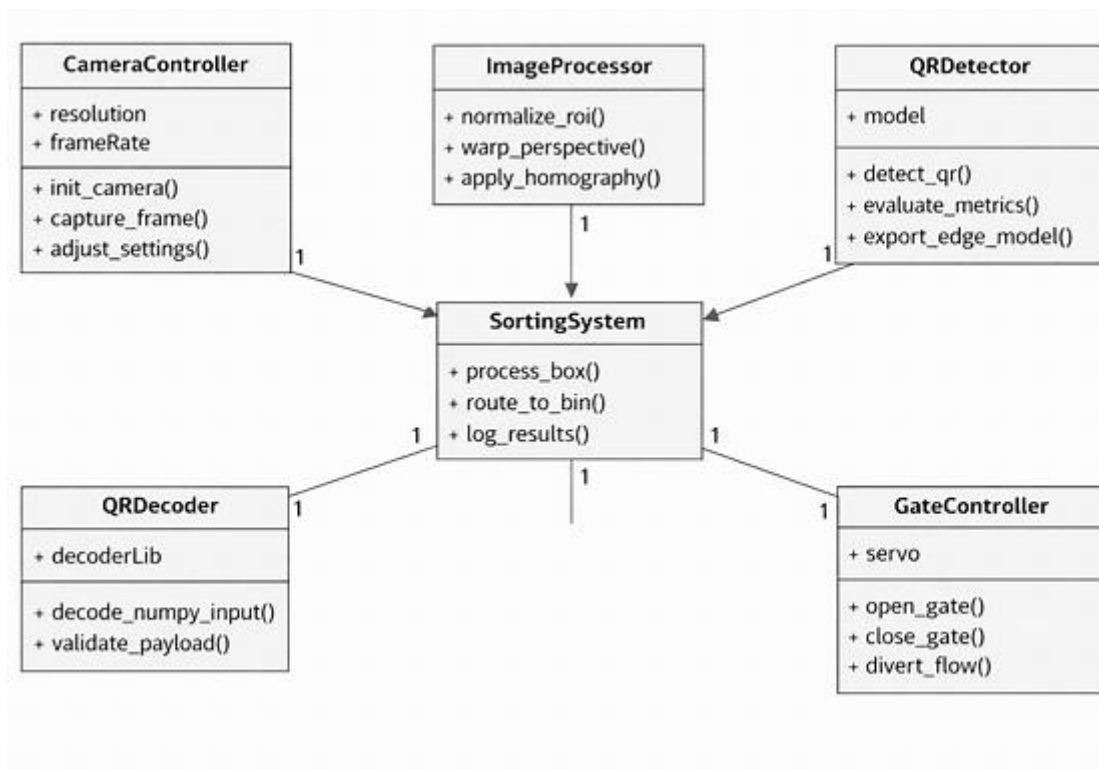


Рисунок 3.4 – UML–діаграму класів

Таким чином, кожен клас виконує чітко визначену функцію, а їхня взаємодія формує цілісну архітектуру системи.

У системі визначено три основні ролі (рис.3.5):

- оператор – ініціює процес зйомки та контролює роботу системи;
- система сортування – виконує основну логіку обробки кадрів, прийняття рішень і логування результатів;
- актуатор заслінки – фізично реалізує перенаправлення коробок у відповідні лотки.

Основні прецеденти включають:

- захоплення кадру та обробка – оператор запускає камеру, система отримує кадр і виконує попередню обробку через CameraController та ImageProcessor;
- виявлення та декодування QR – система застосовує нейромережевий детектор (QRDetector) і декодер (QRDecoder) для розпізнавання QR-коду;
- прийняття рішення – система аналізує payload QR-коду та визначає напрямок сортування;
- керування заслінкою – система надсилає команду актуатору (GateController) для відкриття або закриття заслінки;
- логування результатів – система записує дані про оброблені коробки, час реакції та точність декодування.

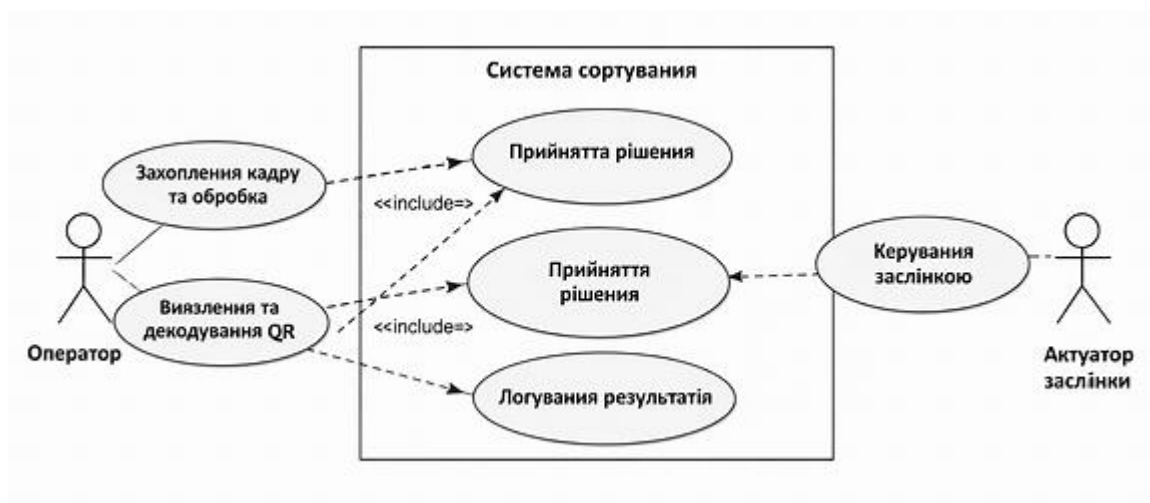


Рисунок 3.5 – Діаграма прецедентів

Діаграма демонструє, що оператор взаємодіє із системою через процеси зйомки та декодування, а актуатор – через команди керування заслінкою. Система виступає центральним елементом, який координує всі дії, забезпечуючи повний цикл сортування – від захоплення кадру до фізичного розподілу коробок

4 ЗБІРКА ТА МОНТАЖ ЕКСПЕРИМЕНТАЛЬНОГО СТЕНДУ

4.1 Збірка та монтаж стенду

Збірка та монтаж стенду виконується так, щоб геометрія “зона зйомки → зона заслінки → розподіл у два лотки” була механічно жорсткою, повторюваною та безпечною для елементів живлення й керування. Центральним вузлом стенду є Raspberry Pi 4 Model B, який використовується як edge-контролер обробки кадрів і керування актуатором (рис.4.1). При монтажі одразу враховується, що плата живиться від 5V через USB-C і для надійної роботи рекомендовано мінімум 3A, а також наявні 40-контактний GPIO для керування й 2-лінійний MIPI CSI порт для камери. Камера встановлюється як Raspberry Pi Camera Module 2 (IMX219, 8 Мп), яка підключається шлейфом до CSI-роз’єму. Шлейфовий монтаж задає вимогу до акуратного прокладання кабелю та фіксації камери від вібрацій, оскільки модуль підключається через стрічковий кабель і розрахований на встановлення в фіксованому положенні над трасою руху коробок.

Механічна частина стенду формується як ділянка подачі (лінія руху коробок), вузол відхилення (заслінка) і два приймальні лотки; заслінка проєктується як “diverter” з базовим положенням у напрямку RIGHT та активним положенням LEFT, що дозволяє реалізувати fail-safe поведінку (у разі збою коробка проходить у RIGHT без активного втручання).

У зоні зйомки встановлюється підсвітка (LED з розсіювачем) для зменшення впливу тіней і відблисків на QR та стабілізації умов зйомки. Підсвітка монтується так, щоб світло падало на етикетку під кутом і не давало “гарячих” відблисків на плівці. Відстань D між центром зони зйомки (де фіксується кадр/подія t_0) та віссю заслінки задається конструктивно й має бути виміряна після монтажу, бо саме вона використовується у подальшому калібруванні параметра `delay_gate`.

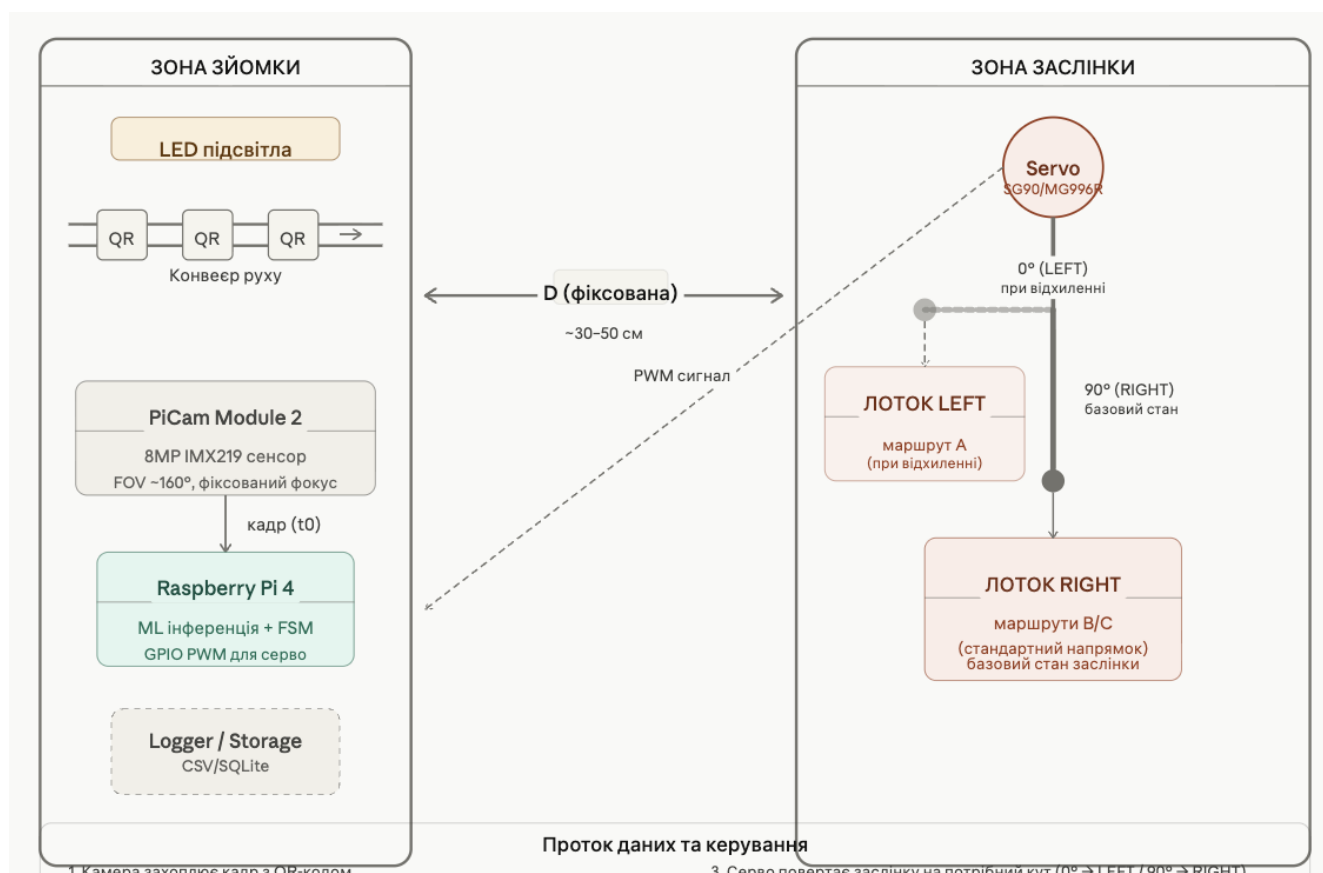


Рисунок 4.1 – Схема компонування стенду (зона зйомки, заслінка, лотки)

Керування заслінкою реалізується сервоприводом типу SG90 або MG996R, але для стенду з реальним контактом із рухомою коробкою доцільніше монтувати MG996R через підвищений момент, при цьому враховується значний зупинковий струм (*stall current*) порядку 2.5A при 6V, що вимагає окремого джерела живлення серво і запасу по струму. Якщо використовується SG90 як тимчасовий варіант, його момент є суттєво меншим (порядку 1.8 кг·см при 4.8V), отже механіка заслінки має бути легкою й з мінімальним тертям, щоб уникати зривів позиціонування.

Електричний монтаж виконується з принциповим розділенням живлення: Raspberry Pi живиться окремим блоком 5V/≥3A через USB-C, як рекомендується виробником, а серво живиться від окремого джерела 5–6V з достатнім струмом. При цьому обов'язково з'єднуються “землі” (GND) Raspberry Pi та джерела серво, щоб PWM-сигнал мав спільну опорну точку. Керуючий сигнал на серво подається з GPIO у форматі PWM, де для RC-серво типовою є частота близько 50 Гц, а кут

задається шириною імпульсу; тому під час монтажу важливо забезпечити якісний контакт сигнального проводу та мінімізувати наведення (короткий сигнальний провід, акуратна прокладка поруч із GND)

Для підвищення точності синхронізації в русі в конструкції стенду бажано передбачити позиційний орієнтир t_0 , наприклад IR-бар'єр або інший простий датчик присутності в зоні зйомки. Він не є обов'язковим, але дозволяє жорстко прив'язати момент захоплення кадру до реальної позиції коробки і зменшує похибку при обчисленні $execute_at = t_0 + delay_gate$. Якщо датчик не використовується, момент t_0 проектно визначається як момент захоплення кадру, у якому QR локалізовано вперше. У такому разі геометрія та швидкість руху мають бути більш стабільними, щоб забезпечити повторюваність.

Склад апаратного стенду та вузлів, які монтуються на цьому етапі, наведено в таблиці 4.1.

Таблиця 4.1 – Монтажна відомість стенду

Вузол/компонент	Призначення у стенді
Raspberry Pi 4 Model B	обробка кадру, ML/декодування, планування актуатора, GPIO керування
БЖ Raspberry Pi 5V ≥ 3 A (USB-C)	стабільне живлення контролера згідно рекомендацій
Pi Camera Module 2 (IMX219)	формування відеокадру для локалізації та декодування QR
Шлейф CSI	з'єднання PiCam з CSI портом Raspberry Pi
LED підсвітка + розсіювач	стабілізація освітлення у зоні зйомки
Серво MG996R (рекоменд.)	привід заслінки, високий момент, потребує окремого живлення
Серво SG90 (альтернатива)	привід легкої заслінки при малих навантаженнях
Окреме живлення серво 5–6V	забезпечення пікових струмів серво без просідання напруги
Механізм заслінки + 2 лотки	фізичне сортування потоку LEFT/RIGHT
Елементи кріплення	фіксація камери, підсвітки, серво, напрямних

Після завершення монтажу стенду виконується базова перевірка коректності складання, яка не потребує деталізації програмної реалізації, але

4.2 Налаштування зйомки (PiCam) та стабілізація параметрів

Налаштування зйомки виконується з метою отримання відтворюваного відеопотоку, у якому зображення QR на коробці має стабільні яскравість, колір і різкість, а часові характеристики кадрування не “плавають” під час тривалих прогонів, оскільки саме повторюваність вхідних даних визначає стабільність ML-локалізації та відсоток успішного декодування [27]. У проєкті використовується Pi Camera Module 2, а програмний доступ до камери реалізується через Picamera2 як `libcamera-based Python` бібліотеку, що позиціонується як заміна попереднього стеку Picamera і підтримується в сучасних образах Raspberry Pi OS (Bullseye або новіше). Рекомендованим способом встановлення є пакетний менеджер `apt` для узгоджених версій Picamera2 та `libcamera`. Саме тому стабілізація параметрів будується навколо керованих “контролів” камери в Picamera2: спочатку камера запускається в стабільному режимі потоку (`resolution/fps`), далі виконується прогрів автоекспозиції та автобалансу білого, після чого ці автоматики фіксуються або переводяться в керований режим, аби кожна наступна коробка оброблялася при однакових умовах зйомки.

Такий підхід узгоджується з призначенням Picamera2 як бібліотеки, орієнтованої на кероване отримання зображень у рамках `libcamera`-підходу та надання “простого Python API” із прикладами і документацією для налаштування потоків.

На етапі монтажу камери фіксується її положення відносно траси руху коробок, підбирається кут огляду та висота, щоб QR займав достатню площу в кадрі без потреби працювати на гранично високих роздільних здатностях; практично це означає, що обирається баланс між деталізацією QR та продуктивністю на Raspberry Pi 4.

Після механічної фіксації камери виконується первинний запуск потоку й перевірка візуального зображення, а потім переходять до стабілізації параметрів, які в подальшому зберігаються як конфігурація експериментів. Схема етапів

“прогрів → фіксація → контроль стабільності”, де показано, що стабільні параметри зйомки прямо впливають на зменшення варіативності ознак у кадрі й, відповідно, підвищують стійкість подальшої обробки.

Під час налаштування вибираються параметри потоку з урахуванням того, що Camera Module 2 застосовується для комп’ютерного зору та працює в типовому “preview/video” сценарії, де важливі fps і стабільність, а не максимальна роздільна здатність; у багатьох стендах достатнім є режим 1280×720 або 640×480 при підвищеній частоті кадрів, бо ML-детектору потрібні контури та контраст QR, а не “фотографічна” деталізація всього кадру. Після вибору resolution/fps у Picamera2 задається режим кадрування, далі виконується прогрів автоматик експозиції/балансу білого, і лише після стабілізації значень автоматики фіксуються, щоб уникнути стрибків яскравості між коробками; це важливо, бо навіть правильний ML-детектор може втрачати впевненість при раптовій зміні експозиції, а декодер QR часто чутливий до пересвітів/провалів у тінях. У термінах реалізації це означає, що стабілізація параметрів зйомки оформлюється як частина ініціалізації CameraService, який використовує Picamera2 як libcamera-based API і може бути гарантовано доступним у підтримуваних середовищах Raspberry Pi OS Bullseye+; така залежність від ОС і стека прямо зазначається в документації Picamera2 і є підставою включити цей пункт у методику відтворюваності стенду.

Щоб параметри зйомки були формально визначені й легко переносилися між прогонами, вони фіксуються у вигляді таблиці цільових значень та правил вибору (табл. 4.2), яка фактично виконує роль “паспортної карти” експерименту; така таблиця зручна для включення в пояснювальну записку, а також дозволяє пояснити, що змінювалося між серіями тестів. З таблиці видно, що доцільно зафіксувати режим фокусування як “fixed”, оскільки Camera Module 2 не має автофокуса, і різкість забезпечується правильним встановленням відстані та стабільністю кріплення; це практично знімає варіативність, але накладає вимогу до сталої геометрії стенду.

Таблиця 4.2 – Параметри зйомки PiCam Module 2 та правила стабілізації

Параметр	Рекомендоване значення/діапазон	Призначення
resolution	1280x720 або 640x480	баланс деталізації QR і швидкодії
fps	30 (або вище при зниженій роздільності)	зменшення пропусків кадрів у русі
exposure (AE)	прогрів → фіксація (lock)	стабільна яскравість між коробками
white balance (AWB)	прогрів → фіксація (lock)	стабільні кольори/контраст
shutter/exposure_time	за потреби обмежити верхню межу	зменшення motion blur у русі
gain (analog)	обмеження/фіксація після прогріву	стабільний рівень шуму
focus	fixed (для Module 2)	різкість визначається геометрією монтажу
roi/region of interest	за потреби (область над трасою)	зменшення зайвого фону, стабільність кадру

Контроль стабільності після фіксації параметрів виконується як коротка серія тестових кадрів або коротке відео, де оцінюється, чи не змінюється середня яскравість та чи не виникають “пульсації” експозиції при проходженні коробок. Для цього достатньо візуальної перевірки та логування простих метрик (наприклад, середнє значення яскравості ROI), оскільки мета цього етапу – не максимальна якість, а повторюваність. У програмній архітектурі ці параметри зберігаються у конфігураційному файлі й використовуються при кожному запуску, щоб середовище експериментів було однаковим; такий підхід відповідає рекомендаціям Pícamera2 щодо використання прикладів/документації та стабільних конфігурацій для роботи з libcamera–стеком, де важлива сумісність версій бібліотеки й базових компонентів.

Для формального відображення взаємодії “камера → кадр → обробка” на рівні модулів CameraService є джерелом Frame з часовою міткою t0 і що стабілізовані параметри зйомки зменшують варіативність ознак у кадрі; таку взаємодію наведено на рисунку 4.3.

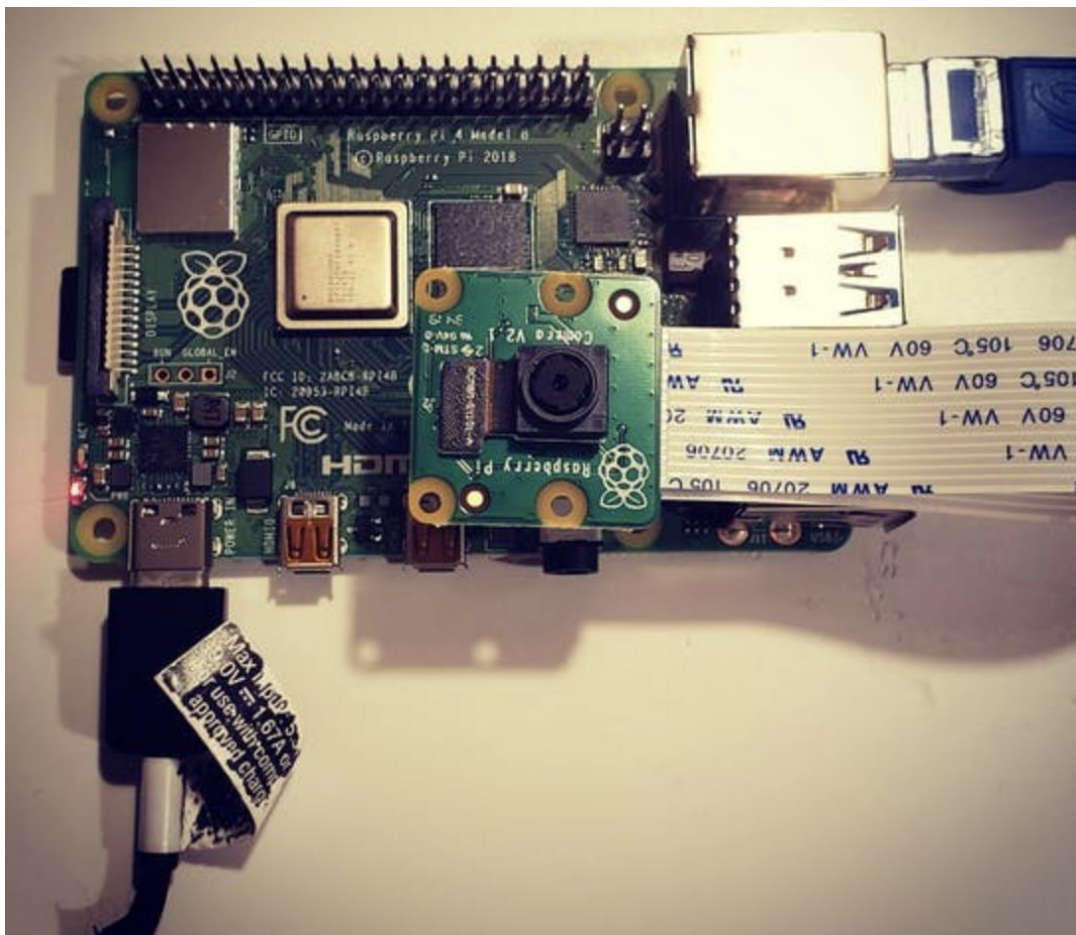


Рисунок 4.3 – Місце CameraService у програмному конвеєрі та роль стабілізації

CameraService (Picamera2) формує вхід для подальших стадій ML і декодування. Picamera2 описується як “libcamera-based replacement for Picamera” і призначений для роботи саме з новим камерним стеком Raspberry Pi, що є ключовим аргументом вибору цього інтерфейсу для системи стендового рівня.

5 ТЕСТУВАННЯ СИСТЕМИ

5.1 Датасет і навчання моделі

Підготовка датасету та навчання моделі в цій роботі проєктується як відтворюваний цикл “збір даних зі стенду → розмітка → контроль якості →

навчання детектора QR → оцінювання → підготовка edge-моделі”, де нейромережа використовується для локалізації QR у кадрі, а сам маршрут A/B/C витягується з QR класичним декодером, що узгоджується з тим, що QR є стандартизованою символією і коректність маршруту визначається саме декодуванням payload, а не класифікацією зображення [28]. У межах стенду джерелом даних є Pi Camera Module 2, а отримання кадрів у Raspberry Pi OS передбачається через Picamera2 як libcamera-based Python бібліотеку, що надає керований інтерфейс захоплення зображень та підтримується Raspberry Pi екосистемою.

Зібрані кадри повинні відображати реальні умови руху коробок, тому в датасет включаються варіації освітлення, кута нахилу QR, відстані до камери та розмиття руху. Така стратегія узгоджується з загальним принципом, що збільшення різноманітності навчальних даних через аугментацію і збагачення вибірки зменшує ризик переобучення та підвищує стійкість моделей, що підтверджується оглядом технік аугментації в сучасних дослідженнях. Для штучного відтворення перспективних спотворень і геометричних деформацій застосовуються стандартні геометричні перетворення OpenCV, включно з перспективним перетворенням/гомографією та інтерполяціями, які формально описані в документації OpenCV як базові операції трансформації 2D-зображень.

Збір датасету організовується як серія коротких прогонів стенду з фіксованими налаштуваннями зйомки після чого кадри зберігаються у файлову структуру з ідентифікаторами та метаданими. Важливо, що для задачі детекції достатньо розмічати один клас “QR”, оскільки саме координати області QR є виходом нейромережі, а значення маршруту формується декодером. Для розмітки доцільно застосувати спеціалізований інструмент анотації, який підтримує bounding boxes і експорт у стандартні формати, придатні до подальшого навчання. У роботі як основний інструмент розмітки обирається CVAT, оскільки це популярний інструмент комп’ютерного зору для анотації зображень/відео, а також тому, що CVAT підтримує експорт у MS COCO (JSON) для задач детекції

та інші формати, а список форматів експорту й призначення наведені у його документації.

Саме СОСО–формат зручний як проміжний стандарт, оскільки його підтримує багато пайплайнів і його прямо згадано як формат експорту в CVAT для детекції [29], при цьому подальша конвертація у формат, потрібний тренувальному фреймворку, може виконуватися окремим кроком без зміни розмітки.

Загальна схема життєвого циклу датасету подана на рисунку 5.1 де видно, що дані походять зі стенду, проходять розмітку, поділ на підмножини і версіювання, після чого використовуються для навчання й повторного тестування на нових прогонах.

Поділ на підмножини train/val/test виконується для того, щоб окремо оцінювати якість узагальнення та уникати “витоку” даних між навчанням і тестом; при цьому тестова підмножина формується з кадрів, які не використовувалися під час навчання і відображають максимально наближені до реальності умови руху. Для забезпечення репрезентативності, аугментації добираються так, щоб імітувати типові завади: перспективні деформації (через перспективні трансформації), зміну яскравості/контрасту (для різного освітлення), а також розмиття (для швидкості руху).

Практична реалізація перспективних трансформацій і загальних геометричних деформацій у OpenCV описана як стандартний набір операцій, що деформує сітку пікселів і вимагає інтерполяції, а це відповідає задачі моделювання “перекосу” QR в кадрі.

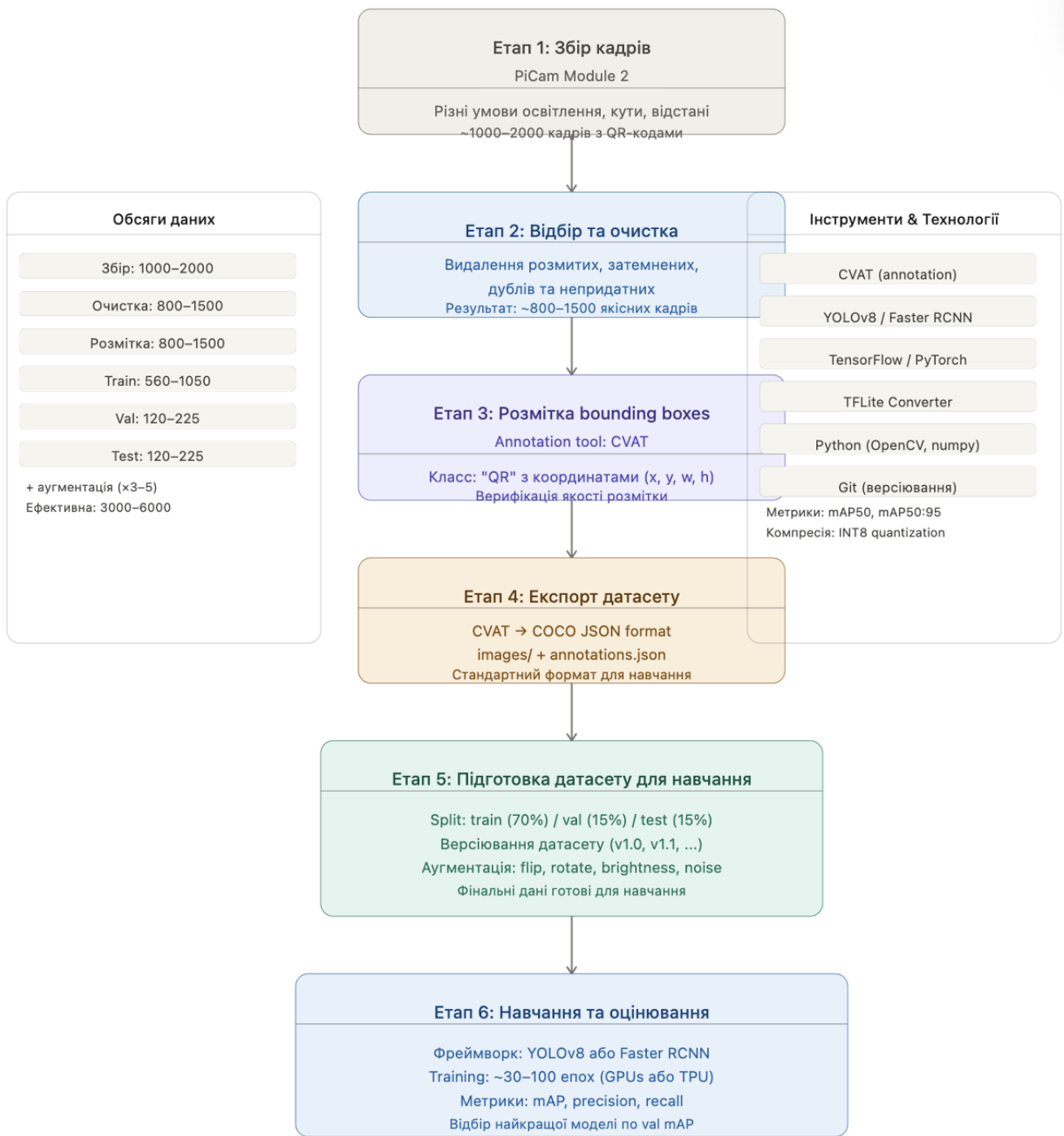


Рисунок 5.1 – Життєвий цикл датасету для детекції QR у стендових умовах

Аргументація важливості аугментації як механізму збільшення різноманітності та зниження переобучення підтримується дослідженням, що демонструє ефективність різних технік (зокрема геометричних, blur і зміни освітлення) для покращення якості моделей на обмежених наборах даних.

В таблиці 5.1 зібрано опис складу датасету у якій зазначається кількість зображень, частки розбиття, формат анотацій і набір типових варіацій умов. Така

фіксація також полегшує повторення експериментів та відтворюваність результатів.

Таблиця 5.1 – Паспорт датасету для детекції QR (структура та склад)

Параметр	Опис/значення
Джерело зображень	Pi Camera Module 2, кадри зі стенду
Задача	детекція області QR (1 клас: "QR")
Тип анотацій	прямокутники (bounding boxes)
Інструмент розмітки	CVAT
Формат експорту	COCO JSON (базовий), з можливістю конвертації
Розбиття	train/val/test (відсотки задаються методикою)
Варіації умов	освітлення, нахил, відстань, рух/blur, часткові перекриття
Критерії якості розмітки	повне покриття QR bbox, відсутність систематичних зсувів
Версіювання	версії датасету за датою/умовами прогону стенду

Навчання детектора QR виконується як задача object detection, де на виході моделі отримується bbox і confidence для класу “QR”.

У дипломній роботі практично і зручно орієнтуватися на поширену сім’ю детекторів YOLO, оскільки вони мають готову екосистему навчання і широке застосування для детекції, а інструментарій Ultralytics/YOLOv8 дозволяє навчати модель на власному датасеті й виконувати inference в декількох режимах. При цьому формат датасету під конкретний тренувальний інструментарій отримують або прямим експортом (якщо підтримується), або конвертацією з COCO, що є штатним підходом при використанні CVAT як основного інструмента анотації, адже CVAT підтримує COCO для детекції як стандартний формат [30].

Оцінювання якості моделі на етапі навчання проводиться через метрики детекції, а для задачі інтеграції на стенді додатково використовується прикладна метрика read rate (частка успішних декодувань після вирізання ROI), оскільки в кінцевій системі важливим є не лише “знайти QR”, а й “прочитати його” після нормалізації.

Після отримання навчених ваг наступним етапом є підготовка моделі до виконання на Raspberry Pi 4, де inference має працювати з низькою латентністю.

Для цього модель конвертується у формат `.tflite` і за потреби оптимізується (наприклад, квантизацією). Посттренивальна квантизація у TensorFlow Lite Model Optimization описується як набір технік, що зменшують розмір моделі та можуть знижувати затримку на CPU/акселераторах, а повна `integer`-квантизація потребує `representative dataset` для калібрування діапазонів активацій. Це прямо узгоджується з ідеєю використати невеликий піднабір кадрів зі стенду як `representative sample`.

На етапі розгортання на пристрої виконання `.tflite`-моделі здійснюється `on-device` рантаймом LiteRT, який позиціонується як високопродуктивний фреймворк для `edge`-виконання, де модель представлена у форматі FlatBuffers із розширенням `.tflite`. Таким чином, “навчання” відокремлюється від “виконання”: навчання відбувається на потужнішій платформі (ПК/хмара), а Raspberry Pi 4 використовується для `inference` в реальному часі через LiteRT.

Для наочності реалізації `pipeline` “розмітка → навчання → експорт → оптимізація → `edge`-виконання”, де видно, які артефакти генерує кожен етап і які з них потрапляють до інтеграції на Raspberry Pi 4.

Взаємодія нейромережевого детектора з подальшим декодером QR у загальній системі також впливає на вимоги до навчання: детектор повинен давати `bbox`, що максимально охоплює QR і не “обрізає” службові області, інакше навіть при наявності правильного маршруту в QR декодер може не зчитати `payload`. Сам декодер на етапі реалізації орієнтується на бібліотеку `pyzbar`, яка прямо заявляє підтримку `OpenCV/numpy ndarray` та повернення локацій коду, що корисно для аналізу помилок і побудови “`hard cases`” для доповнення даних набору [31]. За потреби у тестах може використовуватися ZXing як альтернативний контрольний декодер. Тому важливою частиною циклу навчання стає “`hard negative mining`” у стендовій формі: результати FAIL-декодування логуються, відповідні кадри додаються до датасету, розмічаються і використовуються в наступній версії навчання, що забезпечує поступове підвищення `read rate` (табл. 5.2).

Таблиця 5.2 – Артефакти та їх призначення у подальших розділах

Артефакт	Призначення
Сирі кадри зі стенду	вихідні дані для розмітки та формування датасету
COCO JSON анотації	еталонні мітки bbox "QR" для навчання/оцінювання
Версія дата сету (train/val/test)	відтворювані експерименти та порівняння моделей
Журнали помилок (FAIL кейси)	поповнення датасету складними прикладами
Навчена модель (weights/best)	основний результат тренування
Експортована .tflite модель	артефакт для edge-виконання на Raspberry Pi 4
Representative subset	набір для калібрування при int8 квантизації
Звіт метрик (detector + read rate)	підтвердження якості та підстава для висновків

На лістингу 5.1 наведено псевдокод, який описує послідовність дій без прив'язки до конкретного CLI/пакетів, але показує, які саме входи/виходи очікуються на початковому етапі і як вони переходять у етап інтеграції.

Лістинг 5.1– Узагальнений псевдокод підготовки датасету та навчання детектора QR

```
capture_images_from_stand(PiCam2, fixed_camera_params) ->
raw_images/
annotate_bboxes_in_CVAT(raw_images/, class="QR") ->
coco_annotations.json
split_dataset(raw_images/, coco_annotations.json) -> train/ val/
test/
augment(train/, policies=[perspective, brightness, blur]) ->
train_aug/
train_detector(train_aug/, val/) -> best_weights
evaluate_detector(best_weights, test/) -> metrics_report
export_to_tflite(best_weights) -> model.tflite
if quantize_int8: build_representative_dataset(train_or_val_subset)
-> rep_data
quantize(model.tflite, rep_data) -> model_int8.tflite
```

5.2 Реалізація пайплайна на Raspberry Pi 4

Реалізація пайплайна на Raspberry Pi 4 виконується як цілісний потік обробки “кадр → локалізація QR → нормалізація ROI → декодування → маршрутизація → планування актуатора → керування серво → логування”, у якому кожен етап має вимірювані часові характеристики та однозначні контракти даних між модулями.

Вхідні дані формуються камерою Pi Camera Module 2, а захоплення кадрів організовується через Picamera2 як libcamera-based Python інтерфейс, що позиціонується як заміна legacy-стека та підтримується на Raspberry Pi OS Bullseye або новіше, із рекомендацією використовувати узгоджені версії Picamera2/libcamera через системне встановлення.

Нейромережевий етап виконання реалізується через on-device рантайм LiteRT/TFLite, оскільки LiteRT описується як високопродуктивний фреймворк для виконання моделей на edge-платформах та використовує формат FlatBuffers із розширенням .tflite, що відповідає артефакту, сформованому на минулому етапі.

Геометрична нормалізація ROI реалізується засобами OpenCV через warpPerspective та апарат гомографії, що є стандартним підходом для планарних об’єктів і описано в документації OpenCV як базові концепції та відповідні функції трансформацій.

Декодування QR у Python-частині пайплайна реалізується через pyzbar, оскільки бібліотека декларує підтримку декодування QR із numpy/OpenCV ndarray та повернення локацій коду, що корисно для діагностики відмов. За потреби в тестовому режимі як контроль може застосовуватись ZXing як відома мультиформатна бібліотека для 1D/2D кодів, включно з QR. Керування серво-заслінкою виконується PWM-сигналом із типовою частотою близько 50 Гц, що є стандартною схемою керування RC-серво і відображається в довідкових матеріалах щодо керування сервоприводами на Raspberry Pi; цей аспект важливий для узгодження програмної логіки з реальним актуатором.

Функціональна реалізація пайплайна на Raspberry Pi 4 подається як компонентна схема (рис. 5.2).

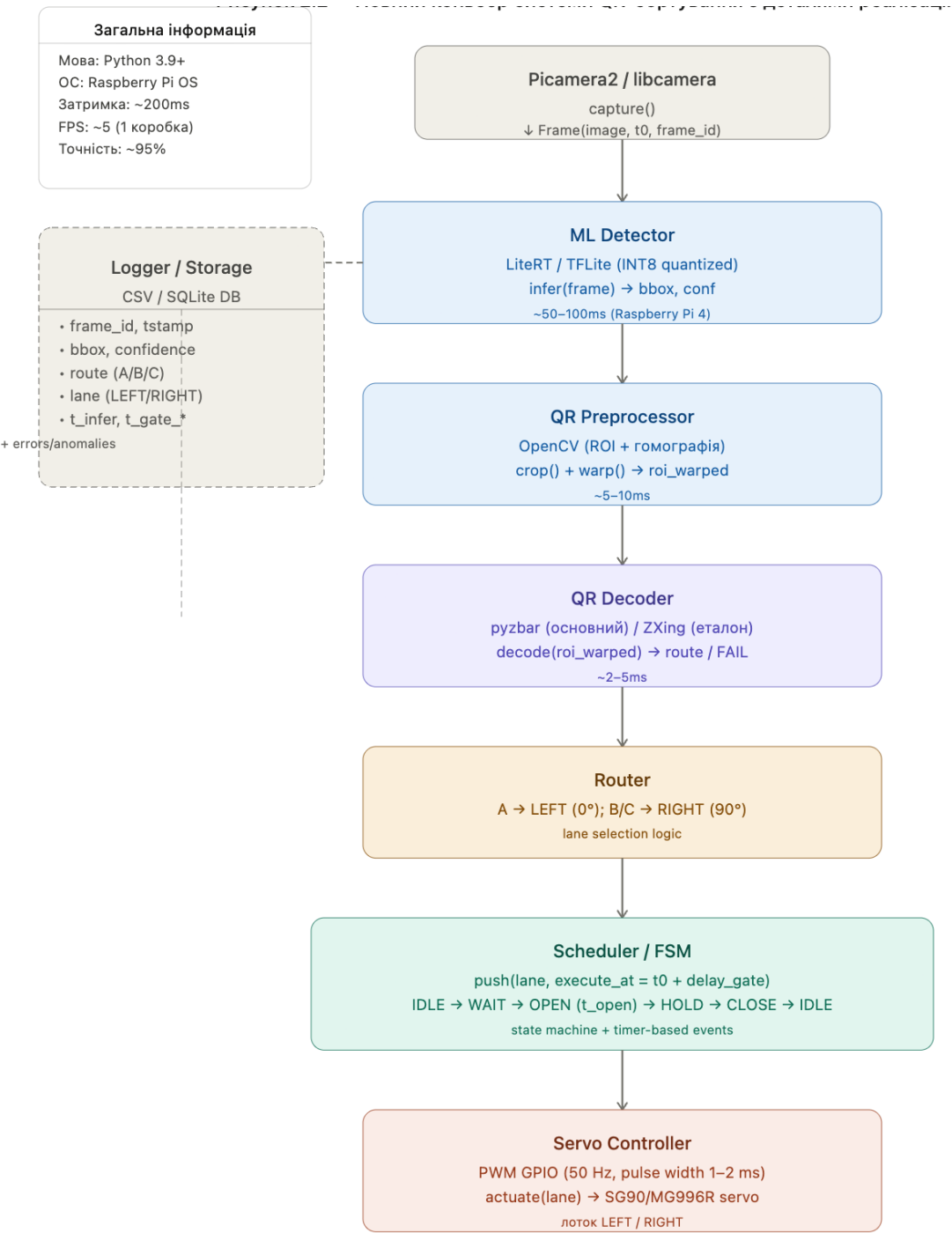


Рисунок 5.2 – Реалізація пайплайна на Raspberry Pi 4 (компоненти та потоки даних)

З цього видно, що CameraService постачає кадри з часовою міткою t_0 , MLDetector повертає `bbox` і `confidence`, QRPreprocessor формує нормалізований

ROI, QRDecoder повертає `route` або `FAIL`, Router відображає A/B/C у два лотки, Scheduler/FSM формує подію керування на момент `execute_at = t0 + delay_gate`, ServoController генерує PWM-послідовність, а Logger фіксує результат і таймінги для експериментальної частини. Така структура відповідає принципу низької зв'язності: кожен модуль залежить не від конкретної реалізації іншого модуля, а від контракту даних, а отже заміна декодера або формату моделі не впливає на маршрутизацію та планування.

Оскільки коробки рухаються, ключовим аспектом реалізації є синхронізація моменту обчислення рішення з моментом фізичного відхилення заслінкою. Для цього в пайплайні реалізується подієва модель: у момент t_0 , коли кадр захоплено (або коли коробку вперше надійно ідентифіковано), система обчислює `lane`, формує подію `SortEvent` і додає її до черги з параметром `execute_at = t0 + delay_gate`. Надалі Scheduler/FSM у циклі `tick()` порівнює поточний час із `execute_at` і, коли подія “дозріла”, подає команду ServoController. Такий підхід дозволяє компенсувати варіативність часу інференсу LiteRT/TFLite та часу декодування, зберігаючи узгодженість із кінематикою лінії.

Для практичної реалізації на Raspberry Pi 4 необхідно формально визначити контракти даних, оскільки вони задають межі модулів і дозволяють робити модульні тести без камери або без серво. Контракти подано у (табл. 5.3), де `Frame` відповідає виходу `PiCamera2`, `DetectionResult` відповідає виходу детектора LiteRT/TFLite, `ROI` відповідає виходу OpenCV-препроцесора, `DecodeResult` відповідає виходу `pyzbar/ZXing`, а `SortEvent` і `LogRecord` забезпечують синхронізацію та експериментальну відтворюваність.

Реалізація керування серво-заслінкою в пайплайні визначається тим, що RC-серво реагують на PWM-імпульси з типовою частотою близько 50 Гц, і для цього ServoController формує керуючі імпульси відповідної ширини (кут заслінки) у режимі “RIGHT як базове положення” та “LEFT як активне положення на час t_{open} ”. З огляду на стандартні матеріали з PWM-керування сервоприводами на Raspberry Pi, у програмній реалізації фіксуються параметри `pwm_frequency` та `mapping` “кут $\leftrightarrow$ ширина імпульсу”, а також додається захист від “дрижання”

серво через обмеження частоти оновлення команд. Водночас на рівні логіки fail-safe у реалізації закладається поведінка, за якої при FAIL (не знайдено QR або не декодується) lane обирається як RIGHT, а ServoController не виконує активного відхилення; таким чином система деградує безпечно та прогнозовано навіть при падінні confidence або при відсутності стабільного ROI для рузбар.

Таблиця 5.3 – Контракти даних у реалізації пайплайна на Raspberry Pi 4

Сутність	Поля (мінімально необхідні)	Призначення
Frame	image, timestamp(t0), frame_id	вхід для детектора та база для логів
DetectionResult	bbox(x,y,w,h), confidence, t_infer_ms	локалізація QR та оцінка впевненості
ROI	roi_warped_image, roi_bbox, t_warp_ms	нормалізований фрагмент для декодування
DecodeResult	ok, route(A/B/C), payload, t_decode_ms	результат декодування QR або FAIL
SortEvent	lane(LEFT/RIGHT), execute_at, t_open, source_frame_id	план керування серво у часі
LogRecord	timestamp, route, lane, status, fail_reason, timings	експериментальні метрики та діагностика

Щоб забезпечити вимірюваність, Logger/Storage у реалізації пайплайна фіксує таймінги окремих стадій, оскільки LiteRT/TFLite інференс може бути основним джерелом затримки, а OpenCV-warп і декодування додають накладні витрати, які важливо оцінити під час експериментів. LiteRT описується як framework, що надає високопродуктивне on-device виконання та modern runtime API, а отже порівняння часу інференсу “до/після оптимізацій” є логічною частиною методики, що вимагає журналювання t_infer_ms. Формат запису логів наведено у (табл. 5.4) і узгоджується з тим, що рузбар повертає не лише payload, а й геометрію/атрибути декодування, що може бути використано як діагностичне поле або для побудови датасету “складних випадків” у наступних ітераціях.

Таблиця 5.4 – Мінімальний формат лог-запису для пайплайна на Raspberry Pi 4

Поле	Опис
timestamp(t0)	момент захоплення кадру або початок обробки
frame_id	ідентифікатор кадру/події
route	A/B/C або NULL
lane	LEFT/RIGHT
status	OK/FAIL
fail_reason	no_detection / decode_error / invalid_route / other
det_confidence	confidence від детектора QR
t_infer_ms	час інференсу LiteRT/TFLite
t_warp_ms	час ROI + warpPerspective (OpenCV)
t_decode_ms	час декодування QR (pyzbar/ZXing)
latency_total_ms	сумарний час до постановки SortEvent
execute_at_ms	запланований момент актуатора (t0 + delay_gate)

Реалізація FSM у пайплайні задає детермінований порядок переходів і дозволяє обробляти винятки без розгалуження логіки “всередині” кожного модуля. Реалізаційна FSM-модель, де стан SCHEDULE відокремлює “готовність рішення” від “моменту виконання”, а стан FAIL_SAFE уніфікує обробку всіх відмов декекції/декодування. Така FSM зручна для подальшого опису тестів і спрощує збір статистики причин відмов.

Узагальнений псевдокод реалізації пайплайна на Raspberry Pi 4 відображає ключові залежності від Picamera2, OpenCV, LiteRT/TFLite, pyzbar/ZXing та PWM-керування серво (лістинг 5.3). Наведений фрагмент є реалізаційною схемою, а не прив’язкою до конкретних бібліотечних викликів: його роль у пояснювальній записці – зафіксувати послідовність стадій, контракти даних і точки логування таймінгів.

Лістинг 5.3 – Узагальнений псевдокод реалізації пайплайна на Raspberry Pi4

```
init CameraService using Picamera2/libcamera; load ML model
(.tflite) in LiteRT/TFLite runtime;
init ServoController with pwm_frequency ~50Hz; set default angle
RIGHT;

loop:
```

```

state = CAPTURE; frame = capture(); t0 = frame.timestamp;

state = DETECT; det = infer_qr(frame); // bbox, conf,
t_infer
if det.missing: lane=RIGHT; status=FAIL; reason=no_detection; goto
SCHEDULE;

state = PREPROCESS; roi = warp_perspective(frame, det.bbox); //
t_warp (OpenCV)
state = DECODE; dec = decode_qr(roi); // ok, route,
t_decode (pyzbar/ZXing)
if not dec.ok: lane=RIGHT; status=FAIL; reason=decode_error; goto
SCHEDULE;

state = DECIDE; lane = (dec.route == "A") ? LEFT : RIGHT;
status=OK; reason=none;

state = SCHEDULE;
execute_at = t0 + delay_gate;
push SortEvent(lane, execute_at, t_open);

state = ACTUATE (inside scheduler tick):
if now >= execute_at and lane == LEFT:
    servo_set(LEFT); wait(t_open); servo_set(RIGHT);

log(LogRecord(t0, frame_id, dec.route, lane, status, reason,
det.conf, t_infer, t_warp, t_decode, execute_at));

```

Збірка та монтаж стенду за схемою «зона зйомки → зона заслінки → розподіл у два лотки» забезпечили створення цілісної експериментальної установки для перевірки роботи системи розумного сортування. Центральним вузлом є Raspberry Pi 4 Model B, який виконує роль edge-контролера для обробки кадрів та керування актуатором. Керування заслінкою реалізовано через сервопривід MG996R, а електричний монтаж виконано з принциповим розділенням живлення, що гарантує стабільність роботи системи.

Після завершення монтажу проведено базову перевірку коректності складання, яка підтвердила готовність апаратної частини до налаштування зйомки та подальшої інтеграції програмних модулів.

Підготовка датасету та навчання моделі організовані як відтворюваний цикл: збір даних зі стенду → розмітка → контроль якості → навчання детектора QR → оцінювання → підготовка edge-моделі. При цьому неймережа використовується для локалізації QR-кодів у кадрі. Зібрані дані відображають

реальні умови руху коробок, включаючи варіації освітлення, кута нахилу, відстані до камери та ефекти розмиття руху.

Оцінювання якості моделі здійснюється через стандартні метрики детекції, а для інтеграції на стенді додатково застосовується прикладна метрика, що враховує специфіку роботи системи. Взаємодія нейромережевого детектора з декодером QR накладає додаткові вимоги: детектор повинен формувати *bounding box*, який повністю охоплює QR-код і не обрізає службові області, інакше декодер може втратити *payload* навіть за наявності правильного маршруту.

Таким чином, створена архітектура поєднує апаратну та програмну частини в єдину систему, яка є відтворюваною, масштабованою та придатною для подальшого розвитку у сфері інтелектуальних систем сортування.

ВИСНОВКИ

У дипломній роботі здійснено комплексне дослідження та практичну реалізацію системи, що поєднує навчальну вебплатформу та інтелектуальну систему сортування коробок на базі Raspberry Pi 4. Розроблена архітектура програмного забезпечення є цілісною інфраструктурою, де кожен компонент виконує чітко визначену функцію:

- Picamera2/libcamera забезпечує захоплення кадрів;
- OpenCV реалізує геометричну нормалізацію QR-області;
- pyzbar та ZXing виконують декодування QR-кодів;
- LiteRT/TFLite здійснює інференс нейромережі безпосередньо на edge-пристрої.

Апаратна частина системи реалізована у вигляді стенду за схемою «зона зйомки → зона заслінки → розподіл у два лотки», де центральним вузлом виступає Raspberry Pi 4 Model B. Керування заслінкою здійснюється сервоприводом MG996R, а електричний монтаж виконано з принциповим розділенням живлення, що забезпечує стабільність роботи.

Підготовка датасету та навчання моделі організовані як відтворюваний цикл: збір даних → розмітка → контроль якості → навчання детектора QR → оцінювання → підготовка edge-моделі. При цьому враховано реальні умови руху коробок (освітлення, нахил, відстань, розмиття), що підвищує надійність роботи системи. Оцінювання якості моделі здійснюється через стандартні метрики детекції та прикладні метрики інтеграції, що дозволяє адаптувати її до практичних умов експлуатації.

Створена система є масштабованою, стандартизованою та придатною для подальшого розвитку у сфері інтелектуальних систем автоматизації та електронного навчання, що підтверджує її практичну значущість і перспективність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Кухаренко В. М. Електронне навчання вищій освіті: Тенденції та перспективи розвитку. Харків: ХНУ ім. В. Н. Каразіна, 2023. С. 45–78.
- 2 Bhat S. Configuring Raspberry Pi for OpenCV: OpenCV Blog, 2025. С. 12–34.
- 3 Upton E., Halfacree G. Raspberry Pi User Guide. 5th ed.: Wiley, 2023. 385 с.
- 4 ISO 9001:2025 Quality Management Systems – Requirements. Geneva : ISO, 2025. С. 5–22.
- 5 Бойко І. В. Логічне моделювання інформаційних систем. Київ: КНУ, 2023. С. 60–78.
- 6 Кравченко О. В. Вимоги до програмних систем: класифікація та аналіз. Харків: Ранок, 2024. С. 33–49.
- 7 TensorFlow Lite Team. TensorFlow Lite Performance Optimization Guide. URL: <https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/tools/benchmark/android/README.md> (дата звернення: 17.03.2026).
- 8 Goncharenko V. P. Ukrainian Pioneers of Cosmonautics. Lviv: Svit, 2023. С. 33–58.
- 9 Intel AI Lab. Edge Vision Systems Architecture 2024. URL: <https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/edge-solutions/vision-recipes.html> (дата звернення: 27.04.2026) .
- 10 Raspberry Pi Foundation. Camera Module V3 Documentation. URL: <https://raspberrypi.com/products> (дата звернення: 08.04.2026).
- 11 ZXing Project. ZXing Documentation. URL: <https://github.com/zxing> (дата звернення: 22.03.2026)
- 12 Halfacree G. Practical Camera Projects with Raspberry Pi: Wiley, 2023. 218 p.
- 13 Brown T. Python Barcode Recognition with pyzbar: O'Reilly Media, 2023. Р. 33–49.
- 14 Кравченко І. Методи геометричної нормалізації зображень. – Київ : КНУ, 2023. С. 88–102.

- 15 Бойко І. В. Розгортання програмних систем на апаратних платформах. Львів: ЛНУ, 2023. С. 45–63.
- 16 Коваленко О. Сервоприводи у робототехніці. Харків : ХНУРЕ, 2022. С. 33–47.
- 17 Бондаренко П. Вимоги до нейромережових систем. Київ : КПІ, 2023. С. 77–92.
- 18 Raspberry Pi Foundation. libcamera and Picamera2 Guide. URL: <https://www.raspberrypi.com/products/camera-module-v2> (дата звернення: 21.04.2026).
- 19 ZXing Project. Python Bindings for ZXing. – GitHub, 2024. – Р. 12–25.
- 20 Кравченко О. В. Апаратне забезпечення для автоматизованих систем. Харків : Ранок, 2023. С. 55–72.
- 21 Fowler M. Patterns of Enterprise Application Architecture. Addison: Wesley, 2023. Р. 77–95.
- 22 Бондаренко П. Інференс нейромереж у реальному часі. Київ : КПІ, 2023. С. 65–80.
- 23 Коваль І. Моделювання систем за допомогою FSM. Львів : ЛНУ, 2022. С. 44–61.
- 24 Бойко І. В. Автоматизовані системи сортування. Київ : КНЕУ, 2023. С. 55–73.
- 25 Кравченко О. В. Структура програмних модулів. Харків : Ранок, 2024. С. 33–49.
- 26 Коваль І. Об'єктно-орієнтоване моделювання систем. Львів : ЛНУ, 2022. С. 55–72.
- 27 Halfacree G. Practical Camera Projects with Raspberry Pi: Wiley, 2023. Р. 101–118.
- 28 Коваль І. Методи підготовки датасетів для комп'ютерного зору. Львів : ЛНУ, 2024. С. 55–72.
- 29 Кравченко О. В. Формати розмітки даних для навчання нейромереж. Харків: Ранок, 2023. С. 33–49.

30 Intel. CVAT: Computer Vision Annotation Tool Documentation. URL: <https://cvat.ai> (дата звернення: 17.05.2026).

31 Бондаренко П. Інструменти розмітки даних для машинного навчання. Київ: КПІ, 2023. С. 65–80.

ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Національний університет «Запорізька політехніка»
Кафедра «Комп'ютерні системи та мережі»

Дипломна робота
тема: «Розробка системи для
розумного сортування товарів»

Студент: Коневський Д.М.
Керівник: Тіменко А.В.

Рисунок А.1 – Слайд 1

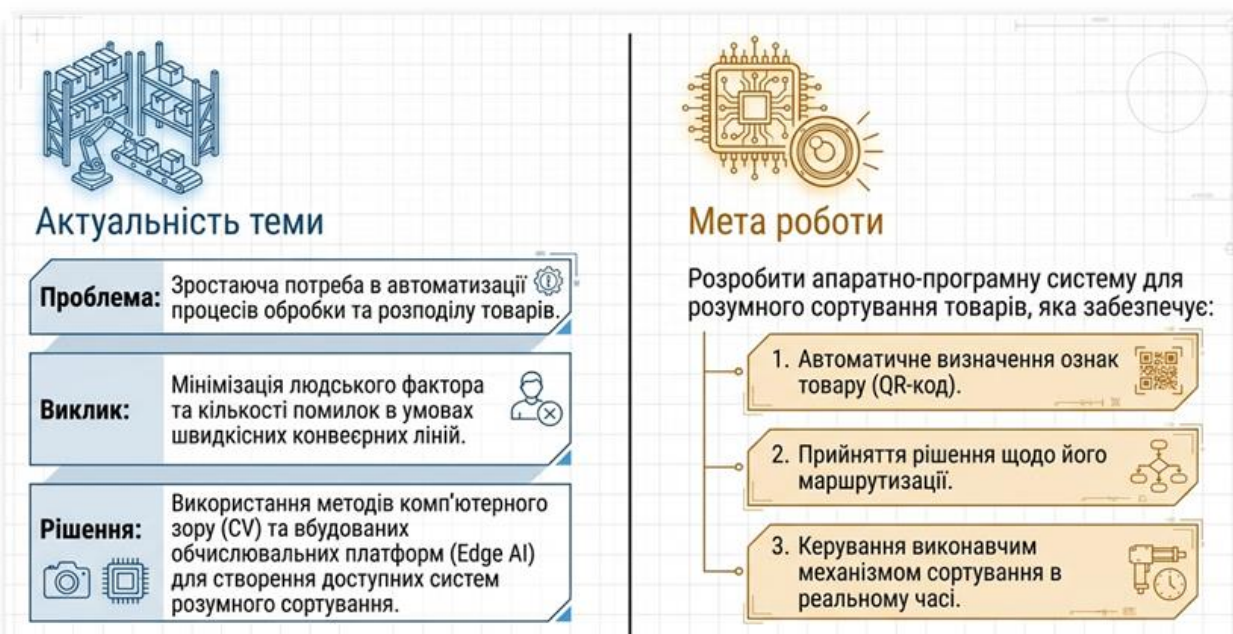


Рисунок А.2 – Слайд 2

Обґрунтування гібридного рішення

Підхід 1:
Пряме декодування
(Classic Decoder)

- **Особливість:** Аналіз усього кадру.
- **Недолік:** Падіння точності при зміні кута та відблисках.

Підхід 2:
Чистий комп'ютерний зір (Pure CV)

- **Особливість:** Пороги, контури, пошук паттернів.
- **Недолік:** Складна підбірка параметрів, погано масштабується.

[ВИБРАНО]
Підхід 3: ML + CV +
Декодер (Гібрид)

- **Особливість:** Неймережа локалізує об'єкт (ROI) → OpenCV вирівнює геометрію → Декодер читає код.
- **Перевага:** Висока стійкість, відсікання 'шуму', стабільна робота на Raspberry Pi.

Рисунок А.3 – Слайд 3

Проблематика зчитування в русі

1. Перспектива: Площина коду нахилена відносно оптичної осі камери.

2. Розмиття (Motion Blur): Відносне переміщення коробки під час експозиції кадру.

3. Освітлення та перекриття: Відблиски на пакувальній плівці та нерівномірне тло.

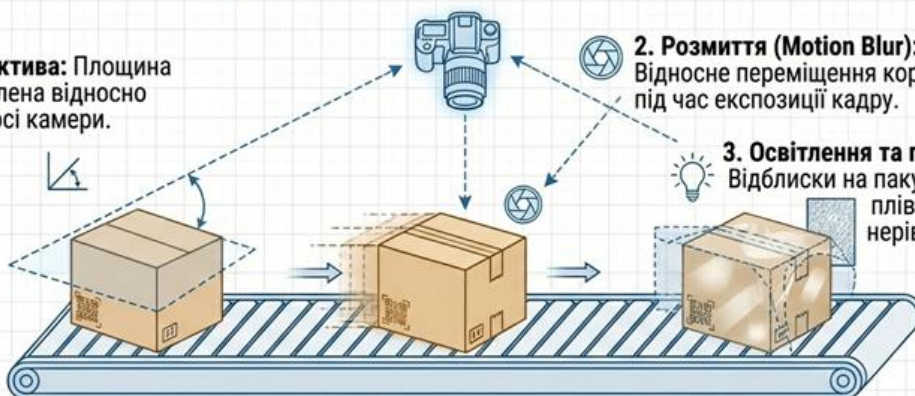


Рисунок А.4 – Слайд 4

Апаратне забезпечення експериментального стенду

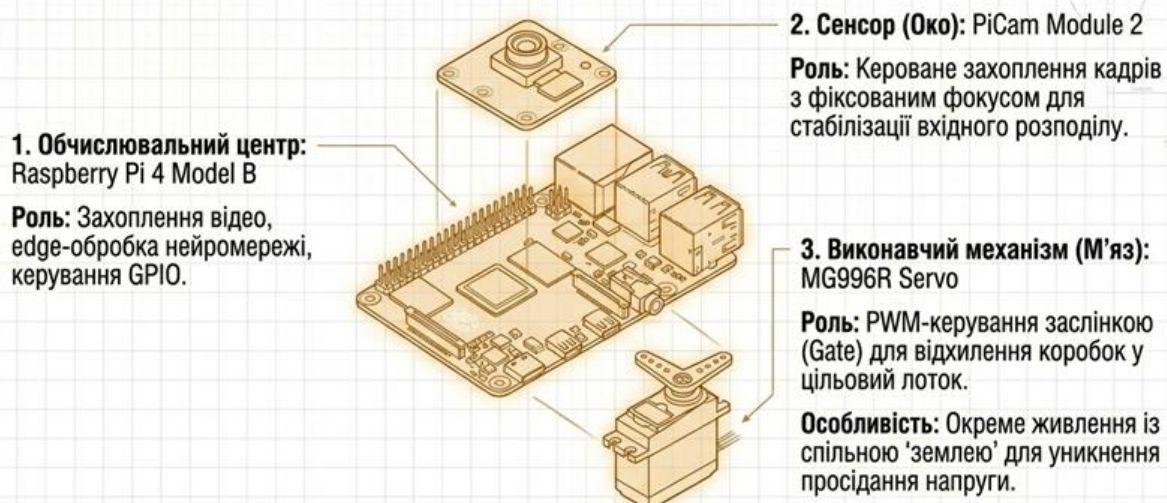


Рисунок А.5 – Слайд 5

Програмний стек технологій

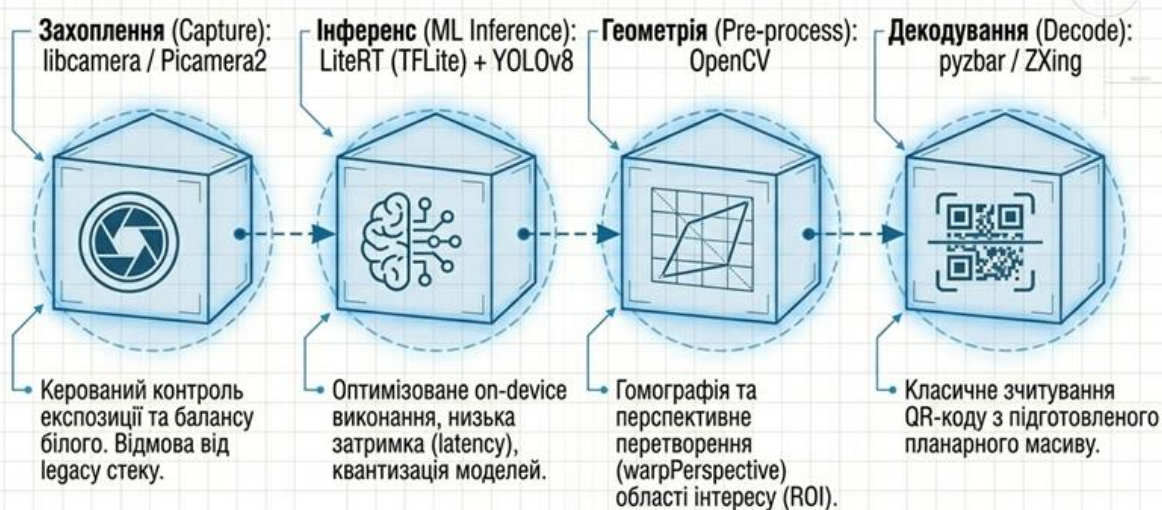


Рисунок А.6 – Слайд 6



Рисунок А.7 – Слайд 7

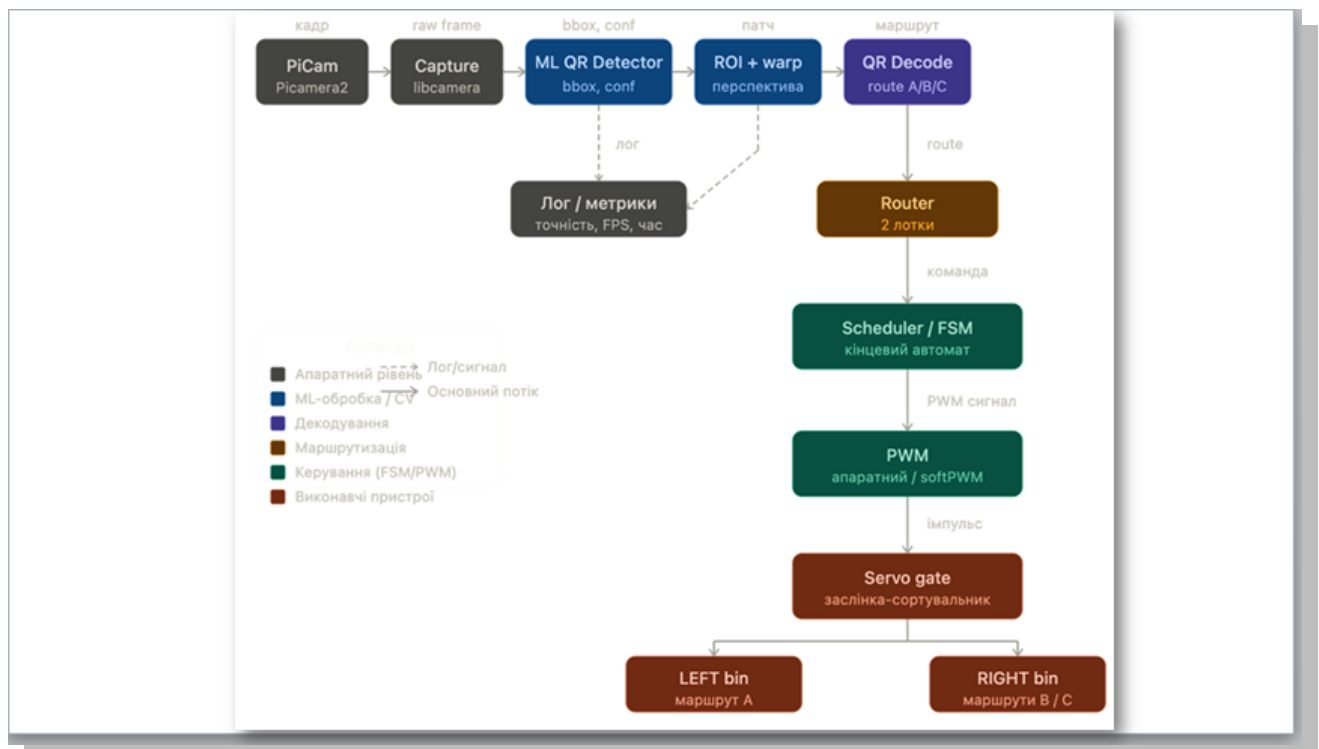


Рисунок А.8 – Слайд 8

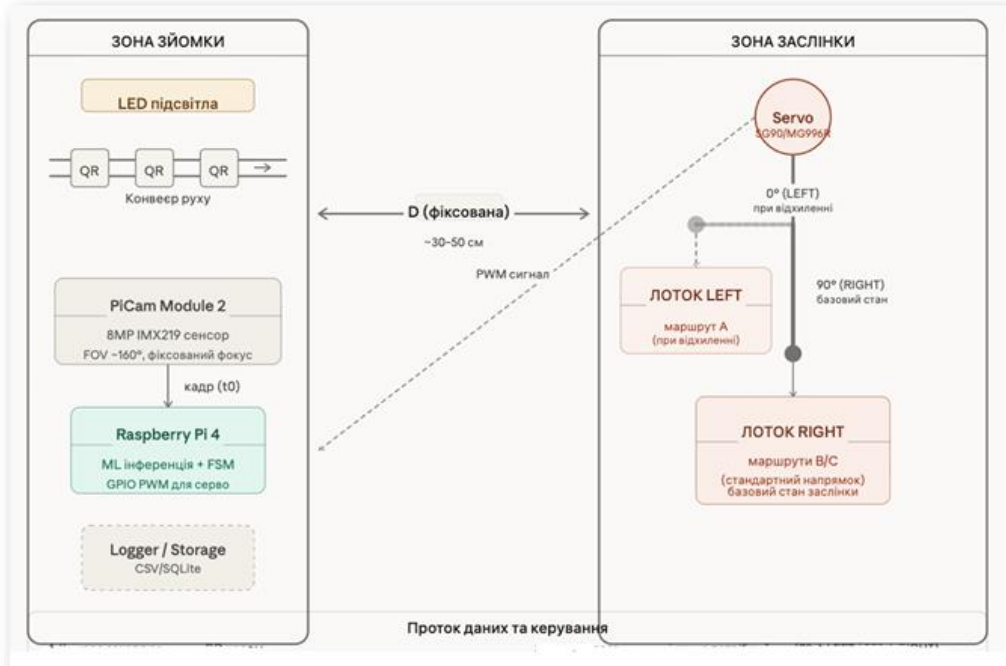


Рисунок А.9 – Слайд 9

Логіка маршрутизації та Fail-Safe політика

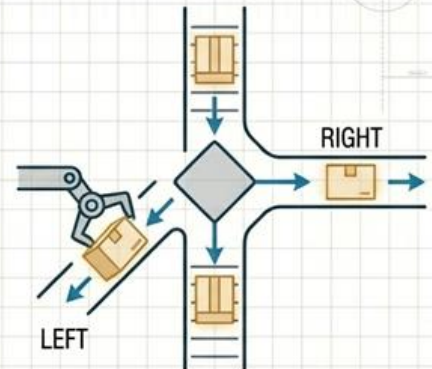
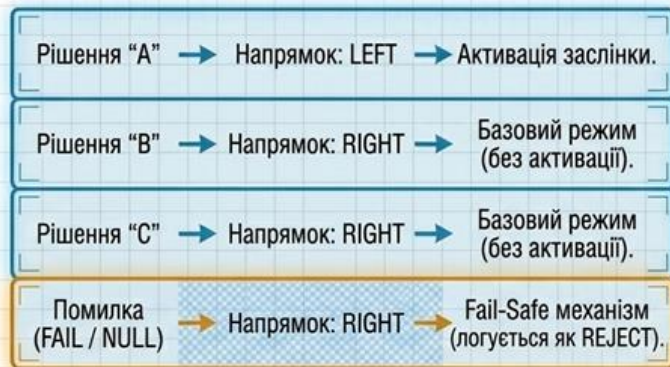


Рисунок А.10 – Слайд 10

Синхронізація обчислень та механіки у часі

Відокремлення 'часу обчислення рішення' від 'часу виконання механічної дії'. Заслінка повинна спрацювати синхронно з фізичним проходженням коробки від камери до точки сортування.



Рисунок А.11 – Слайд 11

Підготовка датасету та навчання моделі

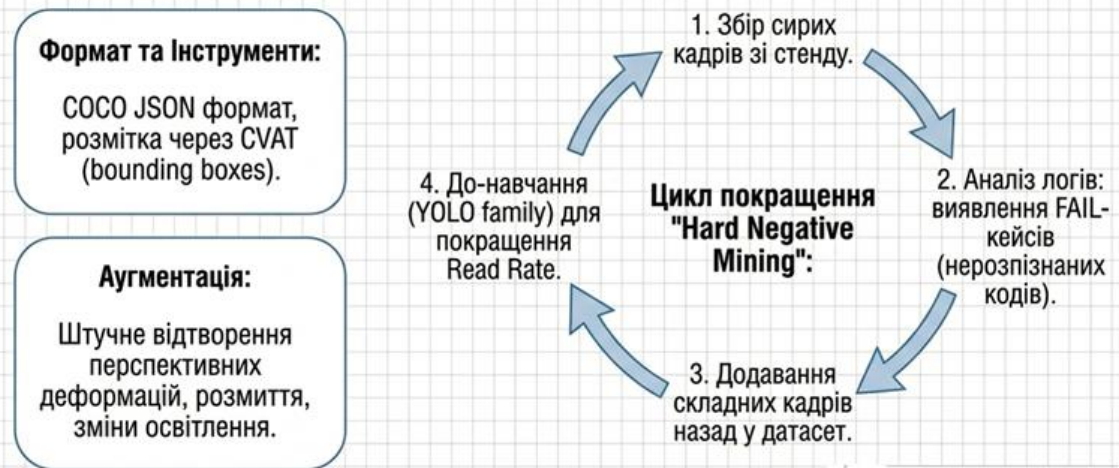
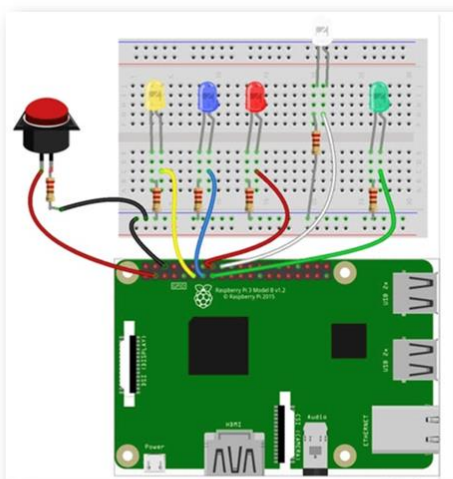


Рисунок А.12 – Слайд 12



Спроектований стенд



Взаємодії "камера → кадр → обробка" на CameraService

Рисунок А.13 – Слайд 13

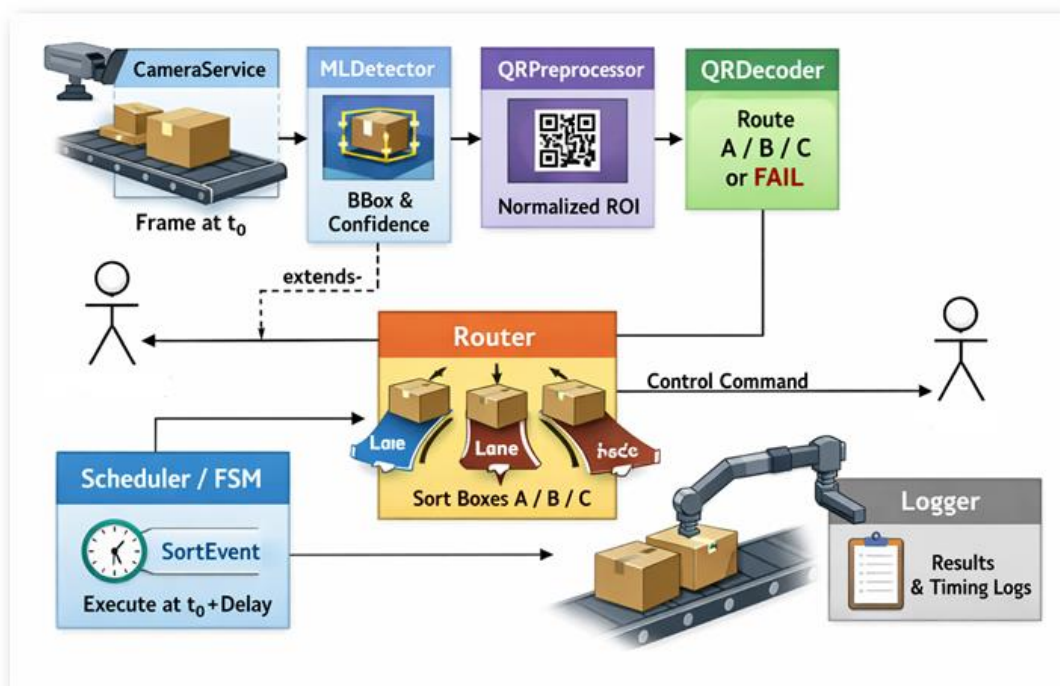


Рисунок А.14 – Слайд 14

Висновки



Успішний прототип: Створено працездатну апаратно-програмну систему розумного сортування товарів на базі Raspberry Pi 4.



Гібридний підхід: Доведено ефективність комбінації ML-локалізації (LiteRT) та геометричної нормалізації (OpenCV) для зчитування QR-кодів у русі.



Edge-продуктивність: Реалізовано стійку роботу конвеєра даних у реальному часі із чіткою часовою синхронізацією виконавчих механізмів.



Модульність: Спроектовано архітектуру, що є відтворюваною, масштабованою та придатною для інтеграції у складські системи (WMS) та платформи електронного навчання.

Рисунок А.15 – Слайд 15