МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувачка кафедри,
д-р техн. наук, проф.

_____Ірина ЖУРАВСЬКА

« __ » _____ 202__ р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Система об'єднання відеопотоків у реальному часі на базі Raspberry Pi

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія»

Здобувач _____Данило НЕВІДОМИЙ
підпис

«__» _____ 202__ р.

Керівник канд. фіз.-мат. наук,
доцент кафедри комп'ютерної інженерії _____Сергій ПУЗИРЬОВ
підпис
«__» _____ 202__ р.

Миколаїв – 2026

Факультет	Комп’ютерних наук
Кафедра	Комп’ютерної інженерії
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	123 Комп’ютерна інженерія
Освітня програма	Комп’ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри комп’ютерної інженерії
_____ Ірина ЖУРАВСЬКА
«_____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Невідомого Данила Олександровича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи

Система об’єднання відеопотоків у реальному часі на базі Raspberry Pi

Затверджена наказом по ЧНУ ім. Петра Могили від 25.11.2025 № 294.

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Розробити апаратно-програмний комплекс на базі одноплатного комп’ютера Raspberry Pi для з’єднання відеопотоків з двох або більше камер у реальному часі, що дозволяє отримати єдине панорамне зображення.

4. Перелік питань, що підлягають розробці:
1) проаналізувати існуючі технології панорамного відеоспостереження, апаратні платформи та алгоритми вилучення локальних ознак для обґрунтування вибору математичного апарату й обчислювальної бази в умовах обмежених ресурсів;
2) дослідити математичні моделі просторових перетворень, розрахунку матриці гомографії та методів згладжування артефактів на межі зшивання зображень;

3) спроєктувати апаратну платформу комплексу на базі мікрокомп’ютера Raspberry Pi 5 із підключенням камерних модулів через роз’єми MIPI CSI, а також розробити електронну схему автономного керування та індикації;

4) розробити апаратну частину комплексу на базі мікрокомп'ютера Raspberry Pi 5 із підключенням камерних модулів через роз'єми MIPI CSI та електронною схемою автономного керування й індикації;

5) розробити програмну частину на мові Python з використанням бібліотеки комп’ютерного зору OpenCV для синхронного захоплення кадрів, їх зшивання та виведення результату в реальному часі;

5. Перелік графічних матеріалів

Презентація

6. Консультанти:

Консультант	Кафедра (організація)	Частина роботи

Керівник роботи _____ Сергій ПУЗИРЬОВ

Особистий підпис *Власне ім’я ПРІЗВИЩЕ*

Здобувач _____ Данило НЕВІДОМИЙ

Особистий підпис *Власне ім’я ПРІЗВИЩЕ*

Дата видачі завдання «_____» _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної бакалаврської роботи

Тема: Система об’єднання відеопотоків у реальному часі на базі Raspberry Pi____

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КБР	01.12.2025	10.12.2025	Виконано
2	Огляд літератури за темою роботи	12.12.2025	19.12.2025	Виконано
3	Складання календарного плану КБР	20.12.2025	27.12.2025	Виконано
4	Аналіз предметної області	03.01.2026	08.01.2026	Виконано
5	Розробка проєктних рішень	15.01.2026	02.03.2026	Виконано
6	Моделювання та конструювання АПЗ	03.03.2026	30.03.2026	Виконано
8	Оформлення КБР та презентації	17.04.2026	20.05.2026	Виконано
9	Перший передній захист КБР	22.05.2026	22.05.2026	Виконано
10	Другий передній захист КБР	05.06.2026	05.06.2026	Виконано
11	Завершення оформлення КБР та презентації	07.06.2026	10.06.2026	Виконано
12	Перевірка на академічний плагіат, фальсифікацію та списування	09.06.2026	10.06.2026	Виконано
13	Відгук керівника КБР	10.06.2026	12.06.2026	Виконано
14	Подання КБР рецензенту та рецензування КБР	10.06.2026	13.06.2026	Виконано
15	Подання КБР, її електронної копії та інших документів (відгуку, рецензії)	15.06.2026	17.06.2026	Виконано
16	Захист кваліфікаційної бакалаврської роботи	24.06.2026	24.06.2026	Виконано

Керівник роботи _____ Сергій ПУЗИРЬОВ

Особистий підпис *Власне ім’я ПРІЗВИЩЕ*

Здобувач _____ Данило НЕВІДОМИЙ

Особистий підпис *Власне ім’я ПРІЗВИЩЕ*

«____» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи
«Система об’єднання відеопотоків у реальному часі на базі Raspberry Pi»
Здобувач: Невідомий Данило Олександрович
Керівник: канд. фіз.-мат. наук, доцент Пузирьов Сергій Володимирович

Кваліфікаційна бакалаврська робота присвячена побудові апаратно-програмного комплексу для об'єднання відеопотоків з кількох камер у єдину панораму в режимі реального часу на базі одноплатного мікрокомп'ютера. Актуальність роботи зумовлена наявністю «сліпих зон» у традиційних системах відеоспостереження та необхідністю розробки економічно доступних рішень панорамного зору з обробкою даних безпосередньо на пристрої.

Об'єктом дослідження є процес об'єднання та обробки відеопотоків у реальному часі для створення панорамних зображень. Предметом дослідження є методи та алгоритми комп'ютерного зору, а також апаратно-програмний комплекс на базі мікрокомп'ютера Raspberry Pi 5 для безшовного зшивання відеокадрів. Метою роботи є розробка АПК для з'єднання відеопотоків з двох камер у реальному часі, що дозволяє отримати єдине панорамне зображення. Практична значимість полягає у можливості використання розробленого комплексу як відкритої альтернативи закритим комерційним рішенням у сферах безпеки, робототехніки та мультимедіа.

У першому розділі проведено порівняльний аналіз традиційних підходів (PTZ-камери та Fisheye-об'єктиви), обґрунтовано перевагу багатосенсорного програмного зшивання та сформовано технічні вимоги до комплексу. У другому розділі побудовано математичну модель конвеєра зшивання (ORB, гомографія через DLT, RANSAC, Distance Transform), спроєктовано апаратну частину комплексу з електронною схемою керування й індикації на базі GPIO та розроблено програмну архітектуру багатопотокового зшивання засобами Python та OpenCV. У третьому розділі описано практичну реалізацію апаратно-програмного комплексу: виконано монтаж і пайку плати керування, встановлено програмне середовище на Raspberry Pi OS, проведено калібрування відеосенсорів за шаховою дошкою 9 × 6, налаштовано автономний запуск системи через systemd та реалізовано вебсервер для потокового відтворення панорами у браузері за протоколом MJPEG. Наведено результати тестування: стабільна частота кадрів 24,0 FPS при вихідній роздільності 894 × 461 пікселів.

Робота пройшла апробацію під час XXVIII Всеукраїнської щорічної науково-практичної конференції «Могилянські читання – 2025» (м. Миколаїв, 10–14 листопада 2025 р.).

Пояснювальна записка складається зі вступу, трьох розділів, висновків, переліку джерел посилання та додатків.

Кваліфікаційна бакалаврська робота містить 72 с. (без додатків), 38 рис., 6 табл., 14 формул, 23 джерела посилання та 3 додатка.

***Ключові слова**: Raspberry Pi 5, панорамне відеоспостереження, зшивання відеопотоків, комп'ютерний зір, OpenCV, ORB, гомографія, RANSAC, Edge Computing, MIPI CSI-2, GPIO, реальний час.*

ABSTRACT

of the Bachelor's Thesis

"System to real-time merging of video streams based on Raspberrry PI"

Applicant: Danylo Nevidomyi

Supervisor: Cand. Sc. (Phys.-Math.), Assoc. Prof. Serhii Puzyrov

The bachelor's thesis is devoted to the study of theoretical and practical foundations of building a hardware-software complex for merging video streams from multiple cameras into a single panorama in real time based on a single-board microcomputer. The relevance of the work is driven by growing demands on security and monitoring systems, the presence of blind spots in traditional video surveillance systems, and the need to develop cost-effective and configurable panoramic vision solutions with on-device data processing.

The object of the research is the process of merging and processing video streams in real time to create panoramic images. The subject of the research is the methods and algorithms of computer vision, as well as the hardware-software complex based on the Raspberry Pi 5 microcomputer for seamless video frame stitching. The purpose of the work is to develop a hardware-software complex based on the Raspberry Pi single-board computer for merging video streams from two or more cameras in real time, producing a single panoramic image. The practical significance of the obtained results lies in the possibility of using the developed complex as a flexible and open-source alternative to closed commercial solutions in the fields of security, robotics and multimedia.

The first chapter analyzes the subject area and existing panoramic video surveillance solutions, justifies the advantage of multi-sensor software stitching, compares hardware platforms and formulates a set of technical requirements for the complex. The second chapter presents the mathematical model of the stitching pipeline, including ORB feature detection, homography computation via DLT, outlier filtering using RANSAC, and photometric seam smoothing based on Distance Transform; the hardware circuit based on GPIO ports and the software architecture of multi-threaded stitching using Python and OpenCV are designed. The third chapter describes the practical implementation of the complex: assembly and soldering of the control board, software environment setup on Raspberry Pi OS, stereo camera calibration using a 9 × 6 checkerboard pattern, autonomous system startup via systemd, and a web server for real-time panorama streaming in a browser via MJPEG. Testing results are presented: a stable frame rate of 24.0 FPS at an output resolution of 894 × 461 pixels.

The research findings have been presented and discussed at the XXVIII All-Ukrainian Annual Scientific and Practical Conference "Mohylianski Readings – 2025" (Mykolaiv, November 10–14, 2025).

The thesis consists of an introduction, three chapters, conclusions, a list of references and appendices.

The bachelor's thesis, contains 72 pages (excluding appendices), 38 figures, 6 tables, 14 formulas, 23 references, and 3 appendices.

Keywords: *Raspberry Pi 5, panoramic video surveillance, video stream stitching, computer vision, OpenCV, ORB, homography, RANSAC, Edge Computing, MIPI CSI-2, GPIO, real time.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АПК – апаратно-програмний комплекс

АПЗ – апаратно-програмне забезпечення

КБР – кваліфікаційна бакалаврська робота

ADAS – Advanced Driver Assistance Systems

BRIEF – Binary Robust Independent Elementary Features

CAGR – Compound Annual Growth Rate

CPU – Central Processing Unit

CSI – Camera Serial Interface

DIP – Dual In-line Package

FPC – Flexible Printed Circuit

GPIO – General-Purpose Input/Output

GPU – Graphics Processing Unit

LED – Light-Emitting Diode

MIPI – Mobile Industry Processor Interface

ORB – Oriented FAST and Rotated BRIEF

PTZ – Pan-Tilt-Zoom

RANSAC – Random Sample Consensus

SBC – Single-Board Computer

SSH – Secure Shell

USB – Universal Serial Bus

ВСТУП

У сучасному світі, де вимоги до систем безпеки та моніторингу невпинно зростають, здатність забезпечити безперервний панорамний контроль простору без затримок стає критично важливою. Використання мікрокомп’ютерів у поєднанні з оптимізованими алгоритмами комп’ютерного зору відкриває нові можливості для створення компактних та ефективних систем відеоспостереження на межі мережі (англ. Edge Computing).

Актуальність теми. Стандартні камери відеоспостереження утворюють «сліпі зони» через обмежений кут огляду. Ця проблема вирішується програмним об'єднанням кількох відеопотоків у панораму. Щоб уникнути затримок і перевантаження мережі під час передачі відео на сервери, обчислення необхідно виконувати локально. Тому розробка економічно ефективних систем панорамного зору на базі сучасних мікрокомп’ютерів є актуальною практичною задачею кваліфікаційна бакалаврська робота (КБР).

Об’єкт дослідження – процес об’єднання та обробки відеопотоків у реальному часі для створення панорамних зображень.

Предмет дослідження – методи, алгоритми комп’ютерного зору та апаратно-програмний комплекс (АПК) на базі мікрокомп’ютера Raspberry Pi 5 для безшовного зшивання відеокадрів.

Мета роботи – розробити апаратно-програмний комплекс на базі одноплатного комп’ютера Raspberry Pi для з’єднання відеопотоків з двох камер у реальному часі, що дозволяє отримати єдине панорамне зображення.

Задачі дослідження:

– проаналізувати існуючі технології панорамного відеоспостереження, апаратні платформи та алгоритми вилучення локальних ознак для обґрунтування вибору математичного апарату й обчислювальної бази в умовах обмежених ресурсів;

– дослідити математичні моделі просторових перетворень, розрахунку матриці гомографії та методів згладжування артефактів на межі зшивання зображень;

– спроєктувати апаратну платформу комплексу на базі мікрокомп’ютера Raspberry Pi 5 із підключенням камерних модулів через роз’єми MIPI CSI, а також розробити електронну схему автономного керування та індикації;

– розробити апаратну частину комплексу на базі мікрокомп'ютера Raspberry Pi 5 із підключенням камерних модулів через роз'єми MIPI CSI та електронною схемою автономного керування й індикації;

– розробити програмну частину на мові Python з використанням бібліотеки комп’ютерного зору OpenCV для синхронного захоплення кадрів, їх зшивання та виведення результату в реальному часі;

Розробка АПК практично підтверджує ефективність алгоритмів комп’ютерного зору при генерації панорамного відео. Проєкт розширює потенціал впровадження цієї технології в системи безпеки, робототехніку та мультимедіа, успішно усуваючи проблему «сліпих зон».

Робота пройшла **апробацію** під час XXVIII Всеукраїнської щорічної науково-практичної конференції «Могилянські читання – 2025» (м. Миколаїв, 10–14 листопада 2025 р.) [1].

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ СИСТЕМ ОБ'ЄДНАННЯ ВІДЕОПОТОКІВ У РЕАЛЬНОМУ ЧАСІ. ФОРМУВАННЯ ВИМОГ ДО АПАРАТНО-ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд предметної області та проблематики систем відеоспостереження

Галузь відеоспостереження перебуває на етапі кардинальних структурних змін, зумовлених масовим впровадженням алгоритмів комп'ютерного зору та переходом до парадигми граничних обчислень. Згідно з аналітичними оцінками 2025 року, обсяг глобального ринку відеоспостереження становить 63,1 млрд доларів США, а прогнози на наступне десятиліття свідчать про потенційне зростання до 162,4 млрд доларів із середньорічним темпом приросту близько 10,1 %. Окремої уваги заслуговує сегмент штучного інтелекту в даній галузі: очікується його стрімке зростання від 3,9 млрд доларів у 2024 році до понад 12,4 млрд доларів до кінця 2030 року. Така динаміка пояснюється масовою відмовою від застарілої концепції пасивного відеозапису на користь інтелектуальних систем, здатних до самостійного аналізу обстановки та оперативного реагування на події [2].

Разом з тим, незважаючи на значний прогрес у розвитку програмної аналітики, системи відеоспостереження залишаються вразливими перед фізичним обмеженням – звуженим кутом огляду стандартних об'єктивів. Зазначений фактор є першопричиною утворення так званих «сліпих зон» – ділянок простору, що не охоплюються жодним із встановлених сенсорів. З практичної точки зору, це є найсуттєвішою вразливістю будь-якого охоронного комплексу: зловмисники цілеспрямовано обирають для несанкціонованого проникнення саме ті ділянки об'єкта, які залишаються поза полем зору камер. Аналогічна проблема характерна й для автомобільного сектору: у системах активної допомоги водієві (англ. Advanced Driver Assistance Systems, ADAS) невидимі зони регулярно стають безпосередньою причиною дорожньо-транспортних пригод [3]. Таким чином, стандартна спрямована камера створює лише ілюзію повного контролю, оскільки

навіть найдосконаліший алгоритм аналізу не здатен ідентифікувати загрозу, яка фізично не потрапила в кадр.

З метою нівелювання зазначеного недоліку у галузі традиційно застосовуються два основні апаратні підходи, кожен з яких має суттєві конструктивні обмеження:

– роботизовані PTZ-камери: обладнані моторизованими приводами, що забезпечують фізичне обертання об'єктива по горизонталі та вертикалі, а також можливість оптичного масштабування зображення. Проте механізм послідовного сканування простору неминуче породжує тимчасові сліпі зони в усіх секторах, що перебувають поза активним полем огляду в даний момент;

– камери з надширококутною оптикою типу Fisheye («риб'яче око»): завдяки специфічній випуклій лінзі здатні охоплювати простір у діапазоні від 180 до 360 градусів. Однак такий підхід породжує критичні бочкоподібні геометричні спотворення, усунення яких потребує застосування обчислювально-інтенсивних алгоритмів математичного вирівнювання, що суттєво знижують деталізацію зображення на периферії кадру.

Беручи до уваги зазначені обмеження, галузь дійшла висновку про доцільність застосування багатосенсорних панорамних систем (англ. Multi-sensor cameras) – рішень, у яких декілька незалежних сенсорів із якісними об'єктивами без оптичних викривлень розміщуються в єдиному корпусі або на спільній платформі.

1.2 Детальний огляд та порівняльний аналіз існуючих рішень панорамного відеоспостереження

Вирішення завдання забезпечення максимального візуального охоплення контрольованої території та ліквідації «сліпих зон» передбачає вибір серед трьох фундаментальних технологічних підходів, представлених на сучасному ринку систем відеоспостереження. Перший підхід базується на застосуванні роботизованих PTZ-камер зі змінним напрямком огляду. Другий – передбачає використання надширококутної оптики класу Fisheye. Третій, найбільш інноваційний напрямок, полягає у впровадженні багатосенсорних панорамних

комплексів із програмним зшиванням декількох відеопотоків. Порівняння технологій наведено в табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз технологій панорамного відеоспостереження

Критерій	**PTZ-камера**	**Fisheye-камера**	**Багатосенсорна система**
Кут огляду	Вузький (керований)	180°–360°	180°–360°
Наявність «сліпих зон»	Так (тимчасові)	Ні	Ні
Якість зображення на периферії	Висока (в активній зоні)	Низька (спотворення)	Висока (рівномірна)
Геометричні спотворення	Відсутні	Критичні (бочкоподібні)	Відсутні або мінімальні
Рухомі механічні частини	Так (мотори, редуктори)	Ні	Ні
Надійність / ресурс роботи	Низька (знос механіки)	Висока	Висока
Одночасне охоплення всього периметру	Ні	Так	Так
Обчислювальна складність обробки	Низька	Висока	Висока
Вартість рішення	Середня	Низька–середня	Середня–висока
Гнучкість налаштування	Обмежена	Обмежена	Висока

Наведені у таблиці дані дозволяють виділити ключові закономірності. PTZ-камери забезпечують високу деталізацію лише в активній зоні огляду, проте принципово не здатні контролювати весь периметр одночасно. Fisheye-камери вирішують проблему охоплення, але ціною критичних геометричних спотворень. Багатосенсорні системи поєднують переваги обох підходів – безперервне

охоплення периметру та високу рівномірну якість зображення – за рахунок вищої обчислювальної складності обробки. Для детальнішого обґрунтування цих висновків далі розглянуто кожен із підходів окремо.

1.2.1 Огляд традиційних підходів: PTZ та Fisheye-камери

Серед традиційних апаратних підходів до розширення кута огляду систем відеоспостереження першочергової уваги заслуговують роботизовані PTZ-камери. Конструктивною основою таких пристроїв є система моторизованих приводів, що забезпечують фізичне обертання оптичного блоку у горизонтальній та вертикальній площинах, а також реалізацію потужного оптичного масштабування для детального розгляду віддалених об'єктів. Безперечною експлуатаційною перевагою PTZ-камер є здатність оператора ідентифікувати біометричні характеристики особи або зчитати державний реєстраційний номер транспортного засобу на відстані від десятків до сотень метрів.

Проте саме в основі цього механізму закладено принциповий концептуальний недолік. У момент, коли камера зосереджується на конкретній події у вузькому секторі спостереження, уся решта підконтрольної території залишається абсолютно відкритою та некерованою – система перманентно генерує тимчасові «сліпі зони», що є неприпустимим для об'єктів із підвищеними вимогами до безперервності спостереження. Окрім цього концептуального обмеження, PTZ-камери характеризуються суттєвими інженерними вразливостями: наявність великої кількості складних рухомих механічних елементів – крокових електродвигунів, пасових передач та редукторних вузлів – робить такі пристрої вкрай чутливими до умов експлуатації. Під впливом запиленості, температурних перепадів та механічних вібрацій рухомі вузли поступово зношуються, що призводить до зниження точності позиціонування та скорочення ресурсу роботи пристрою, обумовлюючи необхідність регулярного планового технічного обслуговування. Приклад зображення з PTZ-камери наведено на рис. 1.1.

Рисунок 1.1 – Приклад зображення PTZ-камери [3]

Намагаючись усунути проблему механічного зносу та пов'язану з ним ненадійність, виробники запропонували альтернативне рішення – камери з надширококутною оптикою типу Fisheye. Принцип дії таких пристроїв базується на застосуванні єдиного сенсора у поєднанні зі спеціальною опуклою лінзою, фізичні властивості якої дозволяють захоплювати світлові потоки з усіх напрямків одночасно, забезпечуючи кут огляду у діапазоні від 180 до повних 360 градусів. З конструктивної точки зору це рішення має очевидні переваги: камера є абсолютно нерухомою, що виключає механічний знос і підвищує довговічність пристрою, а весь навколишній простір постійно перебуває у полі зору сенсора.

Однак зазначений підхід породжує нову, не менш критичну проблему оптичного характеру. Зображення, сформоване надширококутною лінзою, характеризується екстремальними бочкоподібними геометричними спотвореннями, що перетворює тривимірний простір на своєрідну сферичну проєкцію. Для приведення такого зображення до придатного для аналізу вигляду програмне забезпечення вимушено застосовує математично складні алгоритми зворотного геометричного перетворення – так званий dewarping. Процес примусового розгортання сферичної проєкції на плоску поверхню є обчислювально інтенсивним та призводить до значної втрати інформативності: якщо в центральній зоні кадру щільність пікселів залишається на прийнятному рівні, то на периферії зображення набуває характерної розмитості та геометричної деформації, що унеможливлює надійну ідентифікацію об'єктів у крайових зонах.

Рисунок 1.2 – Приклад зображення камери «риб’яче око» [3]

Таким чином, обидві розглянуті технології мають принципові функціональні обмеження: PTZ-камери генерують тимчасові «сліпі зони» внаслідок послідовного характеру сканування простору, тоді як Fisheye-камери забезпечують безперервне охоплення периметру ціною критичного погіршення геометричної достовірності та деталізації зображення на периферії кадру.

1.2.2 Огляд комерційних багатосенсорних систем зі зшиванням

Усвідомлення принципових недоліків розглянутих вище технологій спонукало провідних виробників галузі до розробки якісно нового класу обладнання – багатосенсорних панорамних систем. Архітектурна концепція таких комплексів полягає у розміщенні в єдиному корпусі або на спільній монтажній платформі декількох цілком незалежних матриць зображення, оснащених стандартними об'єктивами з мінімальними оптичними спотвореннями. Відеопотоки з усіх сенсорів одночасно надходять на вбудований обчислювальний блок, який за допомогою спеціалізованих алгоритмів здійснює їх просторове суміщення та злиття у режимі реального часу, формуючи єдину безшовну панораму з рівномірно високою роздільною здатністю в усій площині кадру. Принциповою перевагою цього підходу є здатність безперервно контролювати весь

підконтрольний периметр без виникнення «сліпих зон» та без оптичних спотворень, характерних для Fisheye-рішень [4].

На комерційному ринку беззаперечними лідерами у цьому сегменті є глобальні виробники Hikvision та Dahua Technology. Компанія Hikvision просуває панорамні камери серії PanoVu, флагманські двооб'єктивні моделі якої реалізують апаратне зшивання відео безпосередньо у пристрої, забезпечуючи безперервний огляд у секторі 180 градусів. Додатково в ці пристрої інтегровано алгоритми ColorVu для повноколірного знімання за умов недостатнього освітлення та нейромережевий модуль AcuSense для класифікації рухомих об'єктів. Компанія Dahua Technology пропонує двооб'єктивні рішення лінійки TiOC, що поєднують програмне зшивання відеопотоків з алгоритмами Smart Dual Light, які забезпечують мінімізацію сліпих зон навіть за складних умов освітлення [5].

Рисунок 1.3 – Hikvision PanoVu Dual-lens [5]

Рисунок 1.4 – Dahua Technology TiOC [5]

Незважаючи на очевидні технічні переваги, практичне застосування комерційних рішень пов'язане з двома системними обмеженнями. Першим і найбільш очевидним є висока вартість обладнання професійного класу. У наведеній нижче табл. 1.2 порівняння вартості рішень панорманого зображення.

Таблиця 1.2 – Порівняльний аналіз вартості рішень панорамного відеоспостереження

Рішення	**Компоненти / Модель**	**Орієнтовна вартість, USD**
Hikvision PanoVu DS-2CD6D24G1-IZS	Двооб'єктивна камера з апаратним зшиванням	~800–1200
Dahua Technology TiOC IPC-HDW3849H-AS-PV	Двооб'єктивна камера з алгоритмами Smart Dual Light	~600–950
Власне АПЗ на базі Raspberry Pi 5	Raspberry Pi 5 (4 Гбайт) + 2× камерний модуль + Active Cooler + компоненти схеми керування	~120–160

Як випливає з наведених даних, вартість власної розробки є у 5–8 разів нижчою порівняно з найдоступнішими комерційними аналогами, що робить її використання економічно доцільним для наукових досліджень, освітніх установ та малого і середнього бізнесу.

Другим, і з інженерної точки зору більш принциповим обмеженням, є закрита пропрієтарна архітектура комерційних рішень. Для розробника це означає повну відсутність доступу до внутрішнього конвеєра обробки відеоданих: неможливо модифікувати базовий алгоритм зшивання відповідно до специфічних вимог конкретного об'єкта, інтегрувати власні нейромережеві моделі або розширити функціональність системи нестандартними аналітичними модулями. Система повністю залежить від програмного забезпечення виробника, яке не підлягає зміні стороннім розробником.

Саме сукупність зазначених обмежень – висока вартість та закрита архітектура – обумовлює наукову обґрунтованість та практичну доцільність

розробки власного АПЗ. Застосування відкритої платформи на базі мікрокомп'ютера у поєднанні з доступними модулями камер, що підключаються через швидкісні інтерфейси MIPI CSI, кардинально змінює парадигму розробки подібних систем. Такий підхід забезпечує розробнику повний і безпосередній контроль над алгоритмами комп'ютерного зору на всіх етапах обробки відеоданих, дозволяючи створити гнучкий, масштабований та легко адаптований інструмент для вирішення широкого спектру прикладних задач без технологічної залежності від закритого програмного забезпечення стороннього виробника.

1.3 Аналіз існуючих апаратних платформ для обробки відео в реальному часі

Реалізація концепції панорамного спостереження безпосередньо на місці розгортання системи висуває жорсткі вимоги до центрального обчислювального вузла комплексу. Мікрокомп'ютер має бути здатним безперервно приймати, буферизувати та математично обробляти два відеопотоки високої роздільної здатності у режимі реального часу без пропуску кадрів та програмних зависань. Аналіз актуальної пропозиції ринку одноплатних комп'ютерів (англ. Single-Board Computer, SBC) для подібних задач виявляє дві принципово відмінні архітектурні філософії: вузькоспеціалізовані платформи з акцентом на апаратне прискорення нейромережевих обчислень (зокрема, лінійка NVIDIA Jetson) та універсальні високопродуктивні мікрокомп'ютери нового покоління, абсолютним лідером серед яких наразі є Raspberry Pi 5 [6].

1.3.1 Порівняльний аналіз архітектур: Raspberry Pi 5 проти NVIDIA Jetson Nano

Платформи серії NVIDIA Jetson позиціонуються виробником як спеціалізований інструмент для розгортання важких нейромережевих моделей на периферійних пристроях. Їхньою беззаперечною перевагою є потужний графічний прискорювач з підтримкою технології CUDA, що забезпечує виняткову продуктивність при виконанні матричних обчислень, характерних для інференсу

глибоких нейронних мереж. Проте в контексті задачі відеозшивання ця перевага є менш релевантною, ніж може здаватися. Переважна частина класичних алгоритмів комп'ютерного зору, що складають конвеєр обробки у розроблюваній системі – вилучення ключових точок методом ORB, робастна фільтрація відповідностей алгоритмом RANSAC та безпосереднє злиття пікселів засобами бібліотеки OpenCV – є CPU-bound задачами, тобто виконуються виключно на ядрах центрального процесора. Базова версія Jetson Nano оснащена процесором на архітектурі ARM Cortex-A57, яка є морально застарілою та суттєво поступається сучасним аналогам за одноядерною продуктивністю. Спроба перекласти ці обчислення на GPU пов'язана з необхідністю постійного копіювання великих масивів відеоданих між оперативною пам'яттю системи та відеопам'яттю GPU, що породжує значні накладні витрати на передачу даних через шину і нівелює потенційний виграш у швидкості.

Рисунок 1.5 – Jetson Xavier NX Base Board [6]

Розробники Raspberry Pi 5 обрали принципово інший підхід, зосередившись на максимізації саме процесорної продуктивності. Плата оснащена системою на кристалі Broadcom BCM2712 з чотириядерним процесором ARM Cortex-A76, що функціонує на тактовій частоті до 2,4 ГГц. Мікроархітектура Cortex-A76 належить до сучасного покоління високопродуктивних ядер та демонструє значно вищу

інструкційну ефективність порівняно з Cortex-A57. За результатами незалежних тестів та вимірювань на платформі cpubenchmark.net, Raspberry Pi 5 перевершує Jetson Nano у суто процесорних завданнях приблизно у 2,5–3,0 рази. Версія плати з 4 Гбайт оперативної пам'яті стандарту LPDDR4X додатково забезпечує необхідну пропускну здатність шини пам'яті для паралельної буферизації двох відеопотоків без звернення до повільного файлу підкачки. Зображення одноплатного комп’ютера Raspberry Pi 5 наведено на рис. 1.6.

Рисунок 1.6 – Raspberry Pi 5 [7]

Для об'єктивного порівняння обчислювальної потужності обох платформ у задачах, характерних для конвеєра відеозшивання, доцільно звернутися до результатів незалежного бенчмаркінгу. На рис. 1.7 наведено дані платформи cpubenchmark.net, що відображають інтегральний рейтинг процесорних ядер ARM Cortex-A57 (застосовується у Jetson Nano) та ARM Cortex-A76 (застосовується у Raspberry Pi 5) у багатопотокових обчисленнях.

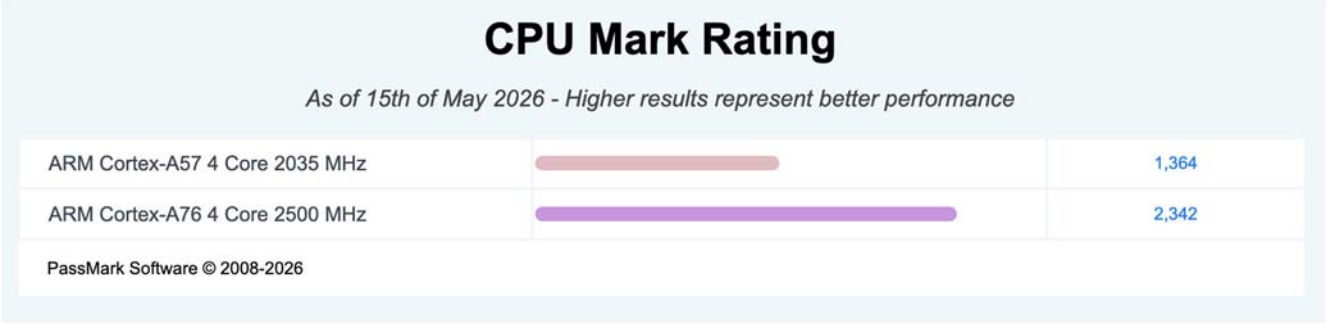


Рисунок 1.7 – Порівняння бенчмарків процесорів на сайті cpubenchmark.net

Наведені дані однозначно свідчать про те, що для задачі CPU-орієнтованого відеозшивання у реальному часі Raspberry Pi 5 є архітектурно більш обґрунтованим вибором порівняно з платформами серії Jetson Nano.

1.3.2 Інтерфейси підключення відеосенсорів та пропускна здатність каналу передачі даних (MIPI CSI-2)

Вибір інтерфейсу підключення відеосенсорів є критично важливим параметром для систем, що вимагають передачі нестиснених відеопотоків із мінімальною затримкою. Протокол USB, незважаючи на широке розповсюдження, є концептуально непридатним для цієї задачі: його пакетна структура передачі даних породжує значні системні накладні витрати та вносить помітну та нестабільну затримку, несумісну з вимогами синхронного захоплення кадрів.

Raspberry Pi 5 є першою моделлю у своїй лінійці, яка отримала два незалежні 4-смугові трансивери стандарту MIPI CSI-2 (англ. Mobile Industry Processor Interface Camera Serial Interface 2), підключені безпосередньо до контролера вводу/виводу RP1. Це дозволяє підключити два незалежні камерні модулі напряму до системної шини без додаткових мультиплексорів чи плат-розширювачів. Фізично ці інтерфейси реалізовані у вигляді компактних 22-контактних роз'ємів із кроком між контактами 0,5 мм. Для підключення стандартних камерних модулів із 15-контактними роз'ємами застосовуються гнучкі перехідні FPC-шлейфи форм-фактору 22-pin до 15-pin. Використання подовжених шлейфів завдовжки 300 мм забезпечує конструктивну свободу для рознесення об'єктивів на необхідну відстань, їх жорсткої фіксації на стенді та точного виставлення кутів огляду, необхідних для формування коректної панорами [7]. Приклад двох портів MIPI CSI-2 наведено нижче на рис. 1.8.

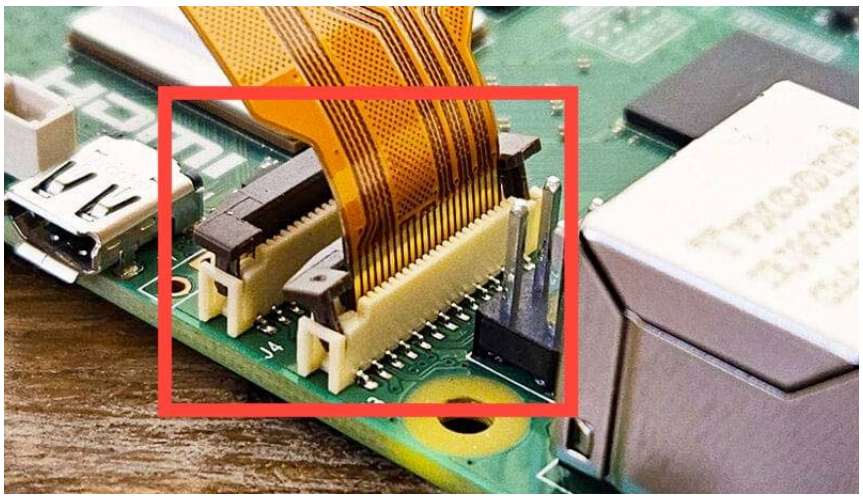


Рисунок 1.8 – Два порти MIPI CSI-2 на Raspberry Pi 5 [7]

Наявність двох незалежних портів MIPI CSI-2 є одним із ключових чинників вибору саме Raspberry Pi 5 як обчислювальної платформи для розроблюваного АПК, оскільки забезпечує прямий низьколатентний канал передачі відеоданих від сенсорів до процесора.

1.3.3 Енергоспоживання, тепловиділення та забезпечення стабільності тривалої роботи

Безперервне виконання математично інтенсивних операцій зшивання двох відеопотоків зі швидкістю 30 кадрів на секунду створює екстремальне та стале теплове навантаження на процесор BCM2712. В умовах пікового завантаження загальне енергоспоживання плати може сягати 8–12 Вт, переважна частина якого розсіюється у вигляді тепла безпосередньо на кристалі. За відсутності ефективної системи тепловідведення температура процесора за короткий час досягає граничної робочої позначки 80–85 °C, після чого автоматично спрацьовує вбудований механізм теплового захисту – термальний троттлінг. Цей захисний алгоритм примусово знижує тактову частоту процесора для запобігання термічному пошкодженню кристала. У контексті системи відеозшивання наслідком троттлінгу є катастрофічне падіння частоти кадрів та фактичний вихід системи з режиму реального часу.

Для гарантування стабільної довготривалої роботи комплексу є обов'язковим встановлення фірмового модуля активного охолодження Raspberry Pi Active Cooler.

Ця система складається з масивного анодованого алюмінієвого радіатора та відцентрового вентилятора, швидкість якого динамічно регулюється операційною системою за допомогою широтно-імпульсної модуляції залежно від поточних показань температурних датчиків ядра. Така конфігурація забезпечує надійне утримання температури кристала в безпечному діапазоні та дозволяє процесору стабільно функціонувати на максимальній тактовій частоті 2,4 ГГц протягом багатогодинних сесій безперервного відеозшивання. Зображення фірмового Active Cooler наведено на рис. 1.9 нижче.

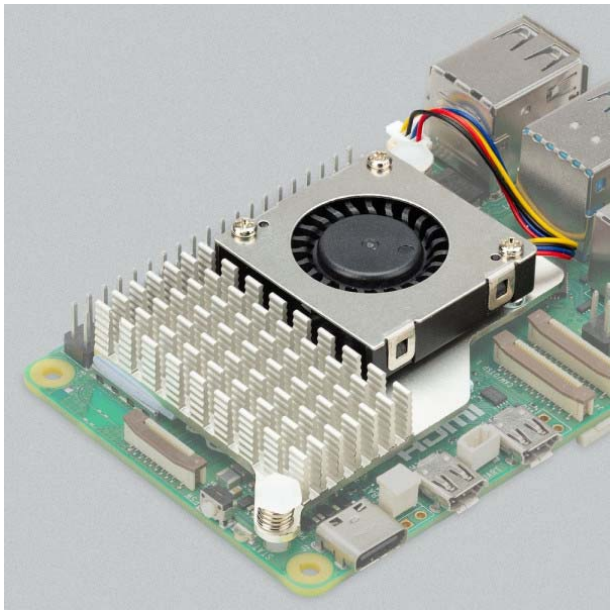


Рисунок 1.9 – Raspberry Pi 5 Active Cooler [7]

За результатами проведеного апаратного аналізу можна впевнено стверджувати, що поєднання мікрокомп'ютера Raspberry Pi 5 з модулем активного охолодження та подовженими FPC-шлейфами для прямого підключення відеосенсорів є найбільш обґрунтованою та продуктивно виправданою апаратною платформою для реалізації розроблюваного АПК. Практична доцільність використання Raspberry Pi як обчислювальної основи систем відеоспостереження підтверджується і в наукових дослідженнях [8].

1.4 Формування вимог до апаратно-програмного забезпечення комплексу

За результатами детального аналізу предметної області та обґрунтування оптимальної апаратної архітектури постає завдання чіткого та структурованого формулювання технічних вимог до всіх складових майбутнього комплексу: програмного конвеєра обробки відео, центрального обчислювального вузла, відеосенсорів, а також фізичного інтерфейсу взаємодії оператора з пристроєм. Сформований перелік вимог слугуватиме фундаментом для подальшого математичного моделювання алгоритмів та безпосередньої практичної реалізації системи.

1.4.1 Функціональні вимоги до програмного забезпечення

Для реалізації математично складного конвеєра обробки зображень обрано бібліотеку комп'ютерного зору OpenCV у поєднанні з мовою програмування Python [9], а для низькорівневої взаємодії з камерними модулями – відкритий фреймворк libcamera. Програмний конвеєр має відповідати таким критичним вимогам.

Першою та фундаментальною вимогою є забезпечення точної синхронізації відеопотоків. Захоплення кадрів з обох незалежних камерних модулів має здійснюватися синхронно з мінімальним часовим зсувом між моментами експозиції. Недотримання цієї вимоги неминуче призводить до появи характерних візуальних артефактів на зшитому зображенні – просторових розривів у відображенні рухомих об'єктів, що перетинають шов панорами. Для розв'язання цієї задачі застосовується вбудований механізм програмної синхронізації на рівні драйверів libcamera, який динамічно підлаштовує момент експозиції підпорядкованої (клієнтської) камери під тактовий ритм основної (серверної).

Другою критичною вимогою є глибока алгоритмічна оптимізація. Аналіз існуючих досліджень у галузі панорамного відеоспостереження підтверджує, що класичні конвеєри на основі вилучення локальних ознак здатні забезпечити зшивання у реальному часі навіть в умовах ресурсно обмежених платформ [10]. З

огляду на мету досягнення стабільної та високої частоти кадрів виключно за рахунок ресурсів центрального процесора, застосування обчислювально важких методів вилучення ознак є неприйнятним. Підходи на основі наскрізного глибокого навчання, попри вищу точність зшивання, потребують нейронного прискорювача та суттєво більшого обсягу обчислень [11], що є несумісним із характеристиками цільової платформи. Відтак, базовим алгоритмом детектування та опису ключових точок визначено ORB, а для робастної фільтрації хибних відповідностей та обчислення матриці гомографії застосовується алгоритм RANSAC.

Третьою обов'язковою вимогою є повна автономність функціонування системи. АПК не має вимагати від оператора ручного запуску програмних компонентів після подачі живлення. Програмне забезпечення повинно функціонувати як системна служба операційної системи: після увімкнення пристрій самостійно завантажується, ініціалізує відеосенсори та переходить у стан готовності, не вимагаючи підключення периферійних пристроїв вводу/виводу чи встановлення віддаленого мережевого з'єднання.

1.4.2 Вимоги до апаратної платформи та відеосенсорів

З огляду на надвисоку обчислювальну інтенсивність задачі відеозшивання у реальному часі, апаратна база комплексу має відповідати суворим вимогам щодо як продуктивності, так і термічної стабільності під тривалим навантаженням.

Центральним обчислювальним ядром визначено мікрокомп'ютер Raspberry Pi 5 у конфігурації щонайменше з 4 Гбайт оперативної пам'яті стандарту LPDDR4X. Зазначений обсяг є мінімально необхідним для забезпечення достатньої пропускної здатності шини пам'яті при одночасній буферизації двох відеопотоків без деградації продуктивності внаслідок звернення до файлу підкачки.

Система охолодження є безкомпромісною конструктивною вимогою. Процес відеозшивання утримує завантаження ядер Cortex-A76 на рівні, близькому до 100 %, що без ефективного тепловідведення призводить до термального троттлінгу та відповідного падіння частоти кадрів. Обов'язковим є встановлення фірмового модуля активного охолодження Raspberry Pi Active Cooler.

Відеосистема комплексу складається з двох окремих камерних модулів, підключених безпосередньо до портів MIPI CSI-2 мікрокомп'ютера через подовжені FPC-шлейфи завдовжки 300 мм. Механічне кріплення сенсорів на стенді має бути абсолютно жорстким: будь-які мікроскопічні зміщення або вібрації камер одна відносно одної призведуть до інвалідації розрахованої матриці гомографії та появи геометричних артефактів на шві панорами.

1.4.3 Вимоги до апаратного інтерфейсу керування та індикації

Для забезпечення повністю автономного режиму роботи комплексу розроблено спеціалізовану електронну панель керування, що взаємодіє з портами GPIO мікрокомп'ютера. Призначення пінів наведено на табл. 1.3 нижче.

Таблиця 1.3 – Призначення пінів GPIO (нумерація BCM)

Компонент	GPIO (BCM)	Призначення
Кнопка START	17	Запуск процесу відеозшивання
Кнопка STOP	27	Зупинка процесу відеозшивання
LED жовтий	22	Індикація режиму готовності
LED зелений	23	Індикація активної роботи
LED червоний	24	Індикація помилки

Керування системою здійснюється через фізичні тактові кнопки типу DIP 4-pin. Програмна обробка натискань реалізована не через примітивне циклічне опитування стану порту, а через механізм апаратних переривань, що виключає нераціональне використання ресурсів процесора. Для фіксації стабільного логічного рівня та захисту від паразитних електромагнітних наведень на вхідний пін кожна кнопка підключається з використанням підтягувального резистора номіналом 10 кОм.

Візуальна індикація стану комплексу реалізована на базі триколірної системи світлодіодів: жовтий сигналізує про режим очікування, зелений – про активне зшивання відеопотоків, червоний – про виникнення програмної помилки. Оскільки

порти GPIO Raspberry Pi 5 мають жорстке обмеження за максимальним вихідним струмом (не більше 16 мА на контакт), підключення світлодіодів безпосередньо до пінів є неприпустимим. Кожен LED-елемент підключається послідовно через захисний струмообмежувальний резистор номіналом 220 Ом, що знижує робочий струм кола до безпечного рівня близько 5,9 мА.

Для забезпечення механічної надійності та довговічності в умовах реальної експлуатації всі електронні компоненти панелі керування – резистори, тактові кнопки та світлодіоди – розпаяні на двосторонній макетній платі зі склотекстоліту розмірами 60×80 мм. Підключення панелі до пінів мікрокомп'ютера здійснюється за допомогою набору перемичок типу «мама-тато» завдовжки 200 мм. План стенду наведено на рис. 1.10.

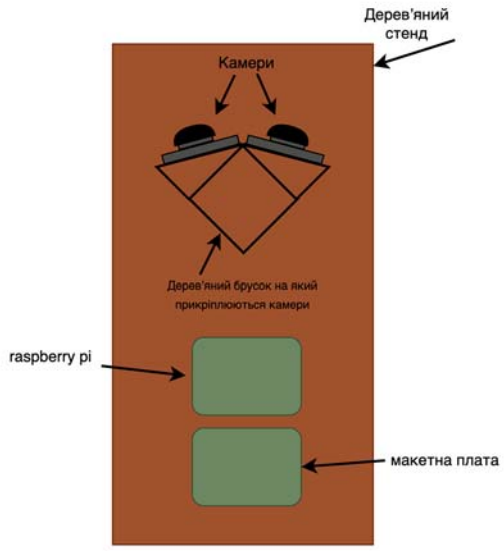


Рисунок 1.10 – Схема розміщення компонентів АПК: вид зверху

Наведена схема ілюструє компонування елементів АПК на дерев'яному стенді. Камерні модулі закріплюються на верхній частині бруска під фіксованим кутом одна відносно одної для забезпечення необхідної зони перекриття полів зору. Мікрокомп'ютер Raspberry Pi 5 та макетна плата з електронною панеллю керування розміщуються на основній площині стенда. Така конструкція забезпечує жорстку фіксацію взаємного положення камер, що є критичною умовою стабільної роботи алгоритму зшивання.

Висновки до розділу 1

У першому розділі здійснено комплексний аналітичний огляд предметної області сучасних систем відеоспостереження та обґрунтовано критичну актуальність проблеми усунення «сліпих зон», які є фундаментальною вразливістю як охоронних комплексів, так і систем автономного керування транспортними засобами.

Проведено порівняльний аналіз існуючих апаратних підходів до розширення кута огляду. Встановлено, що роботизовані PTZ-камери генерують тимчасові зони відсутності контролю внаслідок послідовного характеру сканування простору, тоді як Fisheye-об'єктиви, забезпечуючи безперервне охоплення периметру, спричиняють критичні бочкоподібні геометричні спотворення зображення, що унеможливлюють якісну ідентифікацію об'єктів на периферії кадру. Обґрунтовано, що багатосенсорне програмне зшивання відеопотоків є архітектурно найбільш досконалим підходом, що усуває недоліки обох традиційних технологій. Визначено економічну та інженерну недоцільність застосування готових комерційних рішень для академічних і дослідницьких цілей, що підтверджено порівняльним аналізом вартості. З метою уникнення мережевих затримок та забезпечення незалежності від зовнішньої інфраструктури обґрунтовано доцільність реалізації обчислювального конвеєра за концепцією граничних обчислень.

На основі порівняльного аналізу апаратних платформ доведено, що мікрокомп'ютер Raspberry Pi 5 є оптимальним вибором для задачі відеозшивання в реальному часі. Ключовими чинниками цього вибору є висока продуктивність чотириядерного процесора ARM Cortex-A76, що значно перевищує показники альтернативних платформ у CPU-залежних задачах комп'ютерного зору, а також наявність двох незалежних інтерфейсів MIPI CSI-2, що забезпечують прямий низьколатентний канал передачі відеоданих від сенсорів до процесора без додаткових апаратних розширювачів. Окремо підкреслено необхідність застосування фірмового модуля активного охолодження Raspberry Pi Active Cooler, що гарантує стабільну роботу процесора на максимальній тактовій частоті 2,4 ГГц

у режимі тривалого безперервного відеозшивання та унеможливлює деградацію продуктивності внаслідок теплового троттлінгу.

Сформовано комплекс технічних вимог до всіх складових АПК: визначено програмний стек на базі фреймворку libcamera та бібліотеки OpenCV, обґрунтовано вибір алгоритмів ORB та RANSAC як основи конвеєра обробки, а також встановлено вимоги до схемотехнічного проєктування панелі керування з фізичними кнопками, триколірною світлодіодною індикацією та розпаюванням компонентів на макетній платі. Визначено вимогу до синхронного захоплення кадрів з обох камерних модулів для унеможливлення появи просторових розривів у зображенні рухомих об'єктів на шві панорами.

Сукупність проведеного огляду, порівняльних аналізів та сформованих вимог утворює науково-технічний фундамент для подальшого дослідження. Отримані результати забезпечують перехід до другого розділу роботи, в якому здійснюється детальне математичне моделювання алгоритмів комп'ютерного зору – екстракції локальних ознак, обчислення матриці гомографії та згладжування артефактів зшивання – а також безпосереднє проєктування та реалізація АПК.

2 МАТЕМАТИЧНІ МЕТОДИ ТА ПРОЄКТУВАННЯ СИСТЕМИ ОБ'ЄДНАННЯ ВІДЕОПОТОКІВ

2.1 Математичне моделювання алгоритмів комп'ютерного зору для об'єднання відеопотоків

Розробка інтелектуальної системи панорамного відеоспостереження вимагає створення надійної математичної моделі, здатної описувати процеси захоплення, трансформації та злиття візуальної інформації. Об'єднання двох розрізнених відеопотоків у єдину безшовну панораму в режимі реального часу є складною нелінійною алгоритмічною задачею. Її розв'язання здійснюється через послідовний конвеєр перетворень: виявлення локальних ознак, знаходження геометричних відповідностей, обчислення матриці проєктивного перетворення (гомографії), відсіювання статистичних аномалій (помилкових точок) та безпосереднє згладжування масивів пікселів [12].

2.1.1 Математична модель виявлення та опису локальних ознак (ORB)

У класичному комп'ютерному зорі для пошуку відповідностей між двома зображеннями традиційно використовувалися алгоритми SIFT або SURF. Незважаючи на високу точність, ці методи вимагають значних витрат процесорного часу. З огляду на архітектурні обмеження цільової платформи, фундаментальною математичною основою для виявлення ключових точок у розроблюваній системі обрано алгоритм ORB [13].

Алгоритм ORB виявляє ознаки (кути) за допомогою методу FAST [14]. Для забезпечення інваріантності до обертання ORB використовує метод центроїда інтенсивності. Для локального регіону *S(p)* навколо ключової точки *p* обчислюються просторові моменти інтенсивності:

$$m_{pq} = \sum_{x,y \in S(p)} x^p y^q I(x,y) \tag{2.1}$$

де *I(x,y)* – інтенсивність пікселя з координатами *(x,y)*;

p та *q* належать множині {0,1}.

Положення центроїда інтенсивності *C* визначається як відношення моментів першого та нульового порядків:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{2.2}$$

Вектор, що з'єднує геометричний центр локального вікна з обчисленим центроїдом інтенсивності *C*, формує напрямну, яка слугує домінуючою орієнтацією ключової точки. Кут нахилу цього вектора θ обчислюється за допомогою функції арктангенса:

$$\theta = \arctan2\,(m_{01}, m_{10}) = \arctan2\left(\sum_{(x,y)\in S(p)} yI(x,y)\,,\ \sum_{(x,y)\in S(p)} xI(x,y)\right) \tag{2.3}$$

Визначивши орієнтацію кожної ключової точки, алгоритм формує дескриптор. ORB використовує модифікований дескриптор rBRIEF [15]. Патерн вибірки повертається на знайдений кут θ за допомогою матриці двовимірного обертання:

$$\begin{bmatrix} u_i' \\ v_i' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{2.4}$$

Міра подібності (відстань) між двома дескрипторами P_t та P_r обчислюється за допомогою відстані Геммінга, що виконується шляхом побітового виключного «АБО» (XOR, ⊕) з подальшим підрахунком встановлених бітів:

$$d\,(P_t, P_r) = \sum_{m=1}^{256} P_t^m \ \oplus\ P_r^m \tag{2.5}$$

Менші значення відстані Геммінга відповідають більш подібним дескрипторам. Завдяки бінарній природі rBRIEF та апаратній підтримці інструкцій підрахунку встановлених бітів, обчислення цієї метрики виконується вкрай ефективно навіть на платформах з обмеженими обчислювальними ресурсами, що робить ORB оптимальним вибором для задач реального часу на Raspberry Pi 5.

2.1.2 Алгебраїчна модель геометричного вирівнювання: Матриця Гомографії та DLT-алгоритм

Знайдені пари відповідних точок формують масив даних для обчислення математичної трансформації. Геометричний зв'язок між двома площинами зображень описується 2D-проєктивним перетворенням – матрицею гомографії H. У рамках однорідних координат будь-яка точка двовимірного зображення подається як вектор $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$. Проєктивне відображення точки з першого кадру в точку на другому кадрі описується матричним множенням:

$$w \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.6}$$

Тут w виступає масштабним коефіцієнтом. Для знаходження параметрів матриці застосовується алгоритм Direct Linear Transform [16]. Математична суть алгоритму полягає у перетворенні нелінійної задачі на систему лінійних однорідних рівнянь. Розкривши матричний запис та поділивши перші два рядки на третій, отримуємо:

$$x' = \frac{h_1 x + h_2 y + h_3}{h_7 x + h_8 y + h_9} \tag{2.7}$$

$$y' = \frac{h_4 x + h_5 y + h_6}{h_7 x + h_8 y + h_9} \tag{2.8}$$

Перенесення знаменників та групування членів формує лінійні рівняння:

$$-x h_1 - y h_2 - h_3 + x' x h_7 + x' y h_8 + x' h_9 = 0 \tag{2.9}$$

$$-x h_4 - y h_5 - h_6 + y' x h_7 + y' y h_8 + y' h_9 = 0 \tag{2.10}$$

Цю систему можна подати у вигляді $A_i h = 0$, де h – вектор-стовпець шуканих параметрів гомографії розмірністю 9×1. Для знаходження параметрів потрібно мінімум 4 пари точок. У реальних умовах система рівнянь $Ah = 0$ стає

перевизначеною і розв'язується методом сингулярного розкладу матриці (SVD). Розв'язком є власний вектор матриці $A^T A$ з найменшим власним значенням.

2.1.3 Стохастична оптимізація та фільтрація аномалій: Алгоритм RANSAC

Метод найменших квадратів є фундаментально вразливим до грубих статистичних викидів. Для подолання цієї проблеми застосовується ітераційний стохастичний алгоритм RANSAC [17].

Алгоритм працює за принципом парадигми гіпотез і перевірок: він випадковим чином обирає мінімально необхідну підмножину точок (для 2D- гомографії $s = 4$ пари точок), обчислює кандидатну матрицю гомографії та застосовує її до всіх інших точок датасету. Точки, похибка яких є меншою за певний поріг, класифікуються як корисні (inliers).

Необхідна кількість ітерацій T визначається за ймовірнісною формулою:

$$T = \frac{\log(1-p)}{\log(1-(1-e)^s)} \tag{2.11}$$

де p – цільова ймовірність успіху (наприклад, 0.99);

e – частка хибних точок (outliers) у масиві даних;

s – кількість точок у вибірці ($s = 4$).

Адаптивне визначення кількості ітерацій за наведеною формулою забезпечує гнучкість роботи алгоритму в умовах різного співвідношення корисних та хибних точок: при низькій частці викидів кількість ітерацій залишається невеликою, що мінімізує обчислювальні витрати, тоді як при значній зашумленості даних кількість ітерацій автоматично зростає для гарантування статистичної надійності знайденої моделі гомографії.

2.1.4 Алгоритми математичного згладжування артефактів на швах (Alpha Blending та Distance Transform)

Після геометричного трансформування за допомогою матриці гомографії просте накладання одного зображення на інше утворює видимі шви. Для

фотометричного вирівнювання використовується метод зваженого згладжування на основі карти відстаней (Feathering / Distance Transform).

Для кожного з вихідних зображень $I_k(x, y)$ обчислюється карта відстаней $w_k(x, y)$. Значення кожного пікселя в цій матриці пропорційне евклідовій відстані від цієї точки до найближчого недійсного пікселя (межі кадру).

$$w_k(x, y) = \min_{(x_0, y_0)} \left\{ \sqrt{(x - x_0)^2 + (y - y_0)^2} \mid I_k(x_0, y_0) \text{ is invalid} \right\} \quad (2.12)$$

Інтенсивність кожного фінального пікселя результуючої панорами $I_{blend}(x, y)$ у зоні перекриття обчислюється як зважене середнє відповідних пікселів оригінальних зображень:

$$I_{blend}(x, y) = \frac{w_1(x, y) I_1(x, y) + w_2(x, y) I_2(x, y)}{w_1(x, y) + w_2(x, y)} \quad (2.13)$$

Цей математичний підхід гарантує плавний градієнтний перехід від зображення однієї камери до іншої, приховуючи мікроскопічні відмінності в експозиції та оптичне віньєтування.

2.2 Апаратне проєктування обчислювального блоку, інтерфейсів та кріплень

Для реалізації системи панорамного зору в режимі реального часу обрано мікрокомп'ютер Raspberry Pi 5 із 4 Гбайт оперативної пам'яті. Вибір цієї апаратної платформи зумовлений її високою обчислювальною потужністю та наявністю необхідних спеціалізованих інтерфейсів для паралельного підключення кількох відеосенсорів безпосередньо до системної шини [18].

2.2.1 Архітектурні особливості Raspberry Pi 5: Процесор BCM2712 та контролер I/O RP1

Основним обчислювальним ядром мікрокомп'ютера є система на кристалі Broadcom BCM2712, що містить 64-бітний чотириядерний процесор ARM Cortex-A76 з максимальною тактовою частотою 2,4 ГГц. Принциповою архітектурною відмінністю Raspberry Pi 5 від попередніх поколінь є впровадження двочипової схеми побудови: управління всіма периферійними підсистемами – камерами,

портами USB та пінами GPIO – покладено на окремий виділений контролер вводу/виводу RP1, з'єднаний з основним процесором через шину PCI Express 2.0.

Зазначене архітектурне рішення є критично важливим для розроблюваного АПК. Делегування всіх операцій введення/виведення на окремий чип повністю звільняє ядра Cortex-A76 від рутинної обробки апаратних переривань, зосереджуючи їхні обчислювальні ресурси виключно на виконанні математично інтенсивних алгоритмів комп'ютерного зору – пошуку та зіставлення дескрипторів, обчислення матриці гомографії та злиття пікселів.

2.2.2 Інтерфейс MIPI CSI-2 та схема підключення камер

Для передачі нестиснених відеопотоків з високою роздільною здатністю та мінімальною затримкою підключення камер через інтерфейс USB є неприйнятним: пакетна архітектура цього протоколу породжує значні системні накладні витрати та нестабільний джиттер, несумісний із вимогами синхронного захоплення кадрів. Контролер RP1 забезпечує підтримку двох незалежних 4-смугових трансиверів стандарту MIPI CSI-2, фізично реалізованих на платі у вигляді компактних 22-контактних роз'ємів із кроком між контактами 0,5 мм.

Оскільки більшість серійних камерних модулів оснащені 15-контактними роз'ємами, для їх підключення використовуються гнучкі перехідні FPC-шлейфи форм-фактору 22-pin до 15-pin. В АПК застосовуються два таких шлейфи завдовжки 300 мм. Схема апаратного підключення відеосенсорів є такою:

– камера 1 (лівий ракурс): підключається через перший шлейф FPC 300 мм до порту CAM/DISP 0 на Raspberry Pi 5;

– камера 2 (правий ракурс): підключається через другий шлейф FPC 300 мм до порту CAM/DISP 1 на Raspberry Pi 5.

Застосування подовжених шлейфів замість стандартних коротких є конструктивною необхідністю: вони забезпечують просторову свободу для формування коректної стереобази та рознесення об'єктивів на необхідну відстань уздовж стенда. При монтажі слід уникати перегинів кабелів під гострим кутом, щоб не пошкодити провідники високочастотних ліній передачі даних.

2.2.3 Термальний менеджмент: Система активного охолодження Active Cooler

Одночасна обробка двох відеопотоків у режимі реального часу зі швидкістю 30 кадрів на секунду створює екстремальне та стале теплове навантаження на процесор BCM2712. За відсутності ефективного тепловідведення температура кристала швидко досягає граничної робочої позначки, що спровокує спрацювання захисного алгоритму – термального троттлінгу, тобто примусового зниження тактової частоти. У контексті відеозшивання наслідком цього є миттєве падіння частоти кадрів та фактичний вихід системи з режиму реального часу.

Для гарантування стабільної роботи комплексу застосовується фірмовий модуль Raspberry Pi Active Cooler. Він складається з масивного анодованого алюмінієвого радіатора, що охоплює процесор, оперативну пам'ять та чип RP1, і вбудованого відцентрового вентилятора, підключеного до виділеного 4-контактного роз'єму на платі. Швидкість вентилятора, що може досягати 8000 об/хв, динамічно регулюється операційною системою засобами широтно-імпульсної модуляції залежно від поточних показань вбудованих температурних датчиків ядра. Це забезпечує утримання процесора на максимальній тактовій частоті 2,4 ГГц протягом багатогодинних сесій безперервного відеозшивання.

2.3 Схемотехнічне проєктування керування, компонентна база та пайка

Головною експлуатаційною вимогою до АПК є його повна автономність: оператор не повинен підключати монітор, клавіатуру чи встановлювати мережеве з'єднання для запуску процесу зшивання. Керування комплексом здійснюється за допомогою фізичних кнопок, а поточний стан системи відображається світлодіодною індикацією. Для реалізації цього задуму спроєктовано спеціалізовану електронну панель керування, що взаємодіє з портами GPIO мікрокомп'ютера.

2.3.1 Характеристики компонентної бази та електричні обмеження мікрокомп'ютера

Оскільки портами GPIO на Raspberry Pi 5 керує контролер RP1, архітектура їхнього електричного захисту є суворою. Порти функціонують на логічному рівні 3,3 В, а абсолютна максимальна напруга на контактах не повинна перевищувати 3,63 В – перевищення цього порогу призводить до незворотного термічного пробою чипа RP1. Кожен вивід має жорстке обмеження за вихідним струмом – не більше 16 мА, що виключає можливість безпосереднього підключення навантажень.

Для розробки безпечної та надійної схеми використовується наступний набір електронних компонентів:

- макетна плата двостороння (склотекстоліт);
- тактові кнопки DIP 4-pin;
- резистори 10 кОм;
- резистори 220 Ом;
- світлодіоди 5 мм (жовтий, зелений, червоний);
- набір перемичок 200 мм.

Наведений перелік компонентів повністю відповідає вимогам, сформованим у підрозділі 1.4. Обрана елементна база є широкодоступною, має низьку вартість та забезпечує надійне функціонування схеми у поєднанні з контролером RP1 мікрокомп'ютера Raspberry Pi 5. Подальший розрахунок електричних параметрів кіл індикації та керування виконано у наступному підпункті на основі закону Ома.

2.3.2 Математичний розрахунок кіл індикації та керування

Для безпечної реалізації LED-індикації необхідно математично розрахувати номінал захисного струмообмежувального резистора на основі закону Ома. Вихідна напруга порту GPIO у стані логічної одиниці (HIGH) становить $U_{src} = 3{,}3$ В. Типове пряме падіння напруги на напівпровідниковому p-n переході світлодіода становить приблизно $U_{led} \approx 2{,}0$ В. Щоб забезпечити яскраве світіння

та гарантувати безпеку порту (з двократним запасом міцності від ліміту 16 мА), задамо цільовий робочий струм кола на рівні $I = 8$ мА.

Необхідний опір R_{led} розраховується як:

$$R_{led} = \frac{U_{src} - U_{led}}{I} = \frac{3.3\ B - 2.0\ B}{0.008\ A} = \frac{1.3\ B}{0.008\ A} = 162.5\ \Omega \qquad (2.14)$$

З наявної компонентної бази обрано найближче більше стандартне значення – резистор номіналом 220 Ом. Застосування цього резистора знизить фактичний струм кола до абсолютно безпечних ≈ 5,9 мА.

Підключення тактової кнопки вимагає вирішення проблеми "висячого потенціалу". Коли кнопка розімкнена, вхідний пін діє як антена, вловлюючи електромагнітні перешкоди, що призводить до хаотичної генерації хибних станів. Для фіксації логічного рівня застосовується підтягувальний резистор номіналом 10 кОм.

Логіка роботи кола керування:

– кнопка підключається між інформаційним портом GPIO та шиною GND;

– між інформаційним портом та лінією живлення 3,3 В підключається резистор 10 кОм;

– у стані спокою струм через резистор утримує на порту стабільний потенціал 3,3 В (логічна «1»);

– при натисканні кнопки контакти замикаються на GND, потенціал порту миттєво падає до 0 В (логічний «0»), а струм безпечно стікає через резистор 10 кОм, що виключає коротке замикання.

Для забезпечення механічної надійності всі компоненти панелі керування розпаяні на двосторонній макетній платі зі склотекстоліту. Підключення панелі до пінів мікрокомп'ютера здійснюється за допомогою перемичок «мама-тато».

2.4 Інтеграція керування на рівні ОС: Автономний запуск та Systemd

Вимога повної автономності АПК передбачає, що програмне забезпечення системи має автоматично ініціалізуватися одразу після подачі живлення, без будь-якого втручання оператора. Реалізація цього механізму здійснюється засобами

системного менеджера systemd – стандартної підсистеми ініціалізації та управління службами в операційній системі Raspberry Pi OS [19].

Архітектурно програмний стек АПК поділяється на два компоненти. Перший – launcher.py (див. *Додаток Б*) – є постійно активною фоновою службою, що здійснює моніторинг стану GPIO: відстежує натискання фізичних кнопок START та STOP, управляє триколірною світлодіодною індикацією та запускає або зупиняє основний процес зшивання. Другий компонент – stitch.py (див. *Додаток Б*) – є безпосередньо процесом відеозшивання, що породжується launcher як дочірній підпроцес і завершується за командою STOP або у разі виникнення критичної помилки. Така дворівнева архітектура забезпечує стабільність системи: аварійне завершення процесу зшивання не призводить до втрати контролю над GPIO та індикацією.

Послідовність автономного запуску комплексу після подачі живлення є такою:

1) завантаження Raspberry Pi OS;
2) ініціалізація контролера RP1 та периферії;
3) запуск systemd-служби;
4) ініціалізація GPIO-пінів;
5) вмикання жовтого світлодіода (режим готовності);
6) очікування натискання кнопки START оператором.

Наведена послідовність етапів автономного запуску повністю реалізована засобами стандартного менеджера служб systemd, що гарантує детермінований порядок ініціалізації компонентів та відмовостійкість роботи комплексу. У разі аварійного завершення процесу зшивання служба автоматично перезапускається без втручання оператора, що забезпечує безперервність роботи АПК в реальних умовах експлуатації.

2.5 Програмна архітектура та логіка багатопотокового зшивання (Python та OpenCV)

Програмна частина АПК реалізована мовою Python із використанням бібліотеки комп'ютерного зору OpenCV [20] та фреймворку Picamera2 для низькорівневого доступу до камерних модулів через інтерфейс libcamera [21]. Для досягнення максимальної продуктивності застосовується багатопотокова архітектура, що дозволяє виконувати захоплення кадрів та їх математичну обробку паралельно, не блокуючи один одного.

2.5.1 Багатопотокова архітектура захоплення кадрів

Захоплення кадрів з обох камер здійснюється у двох незалежних потоках виконання. Кожен потік обслуговує одну камеру: безперервно отримує чергове зображення через Picamera2 та поміщає його у власну чергу фіксованої місткості. Використання потокобезпечних черг як буферів між потоком захоплення та потоком обробки є ключовим механізмом синхронізації: головний потік вибирає по одному кадру з кожної черги лише тоді, коли обидва доступні, що природним чином вирівнює можливі мікроскопічні розбіжності у часових мітках між камерами [22].

2.5.2 Конвеєр обробки та зшивання кадрів

Після отримання синхронізованої пари кадрів головний потік виконує послідовний конвеєр обробки, побудований на алгоритмах, обґрунтованих у розділі 2.1.

На першому етапі обидва кадри конвертуються у відтінки сірого для підвищення швидкості роботи детектора ключових точок. Алгоритм ORB виявляє характерні точки та обчислює для кожної бінарний дескриптор rBRIEF. Попарне зіставлення дескрипторів здійснюється методом грубої сили з відстанню Геммінга. Отримані пари відповідностей фільтруються тестом Лоу (відношення відстаней менше 0,75) для відкидання неоднозначних збігів.

На другому етапі відфільтровані відповідності передаються алгоритму RANSAC, який обчислює матрицю гомографії H розміром 3×3. Після першого успішного обчислення матриця H кешується у пам'яті та використовується для всіх наступних кадрів без повторного пошуку ключових точок. Перерахунок гомографії відбувається лише у разі різкого падіння кількості інлаєрів нижче встановленого порогу, що свідчить про зміну геометрії сцени.

На третьому етапі кадр другої камери трансформується функцією cv2.warpPerspective відповідно до матриці H та проєктується на площину першого кадру. У зоні перекриття обох зображень застосовується алгоритм зваженого згладжування на основі карти відстаней (feathering / alpha blending на базі Distance Transform), математична модель якого детально описана у підрозділі 2.1. Результуючий зшитий кадр формує безшовну панораму.

2.5.3 Вебсервер для перегляду відеопотоку

Для забезпечення можливості дистанційного перегляду панорами з будь-якого пристрою в локальній мережі без встановлення додаткового програмного забезпечення реалізовано вбудований HTTP-сервер на базі фреймворку Flask [23]. Готовий зшитий кадр кодується у формат JPEG та передається клієнтам за протоколом MJPEG – стандартним потоковим форматом, що підтримується безпосередньо у браузерах без будь-яких плагінів.

HTTP-сервер функціонує у окремому потоці виконання та не впливає на продуктивність основного конвеєра зшивання. Доступ до відеопотоку здійснюється за адресою pano-cam:8000 з будь-якого комп'ютера, планшета або смартфона, підключеного до тієї самої локальної мережі.

Висновки до розділу 2

У другому розділі здійснено повний цикл проєктування та реалізації апаратно-програмного комплексу системи об'єднання відеопотоків у реальному часі.

Розроблено математичну модель конвеєра комп'ютерного зору, що охоплює: алгоритм виявлення та опису локальних ознак ORB на основі методу центроїда інтенсивності та дескриптора rBRIEF; алгебраїчну модель проєктивного перетворення та алгоритм DLT для обчислення матриці гомографії H; стохастичний алгоритм RANSAC для робастної фільтрації хибних відповідностей; а також метод зваженого згладжування на основі Distance Transform для усунення артефактів на шві панорами.

Виконано апаратне проєктування комплексу: обґрунтовано вибір двочипової архітектури Raspberry Pi 5 з контролером RP1, розроблено схему підключення двох камерних модулів через інтерфейси MIPI CSI-2 з використанням подовжених FPC-шлейфів, визначено конструктивне рішення жорсткої фіксації оптичної системи на стенді та обґрунтовано необхідність системи активного охолодження для забезпечення стабільної тривалої роботи.

Спроєктовано електронну панель керування: проведено розрахунок кіл індикації та керування за законом Ома, визначено номінали захисних резисторів (220 Ом для LED-кіл та 10 кОм для pull-up кіл кнопок), описано компонентну базу та технологію монтажу на макетній платі зі склотекстоліту.

Реалізовано механізм автономного запуску системи засобами менеджера служб systemd, що забезпечує автоматичну ініціалізацію комплексу при подачі живлення без участі оператора.

Розроблено програмну архітектуру системи зшивання на мові Python з використанням OpenCV та Picamera2, також вбудований MJPEG вебсервер на базі Flask для дистанційного перегляду панорами з довільного пристрою в локальній мережі.

3 ОБ’ЄДНАННЯ ВІДЕОПОТОКІВ У РЕАЛЬНОМУ ЧАСІ НА БАЗІ RASPBERRY PI

3.1 Підготовка апаратного середовища та встановлення залежностей

Третій розділ КБР присвячено безпосередній практичній реалізації апаратно-програмного комплексу, спроєктованого у другому розділі. Підготовка апаратного середовища є першим та фундаментальним етапом розгортання системи, від коректності виконання якого залежить працездатність усіх наступних компонентів. У межах цього підрозділу здійснюється встановлення та конфігурація операційної системи з організацією віддаленого доступу, фізичне підключення камерних модулів через інтерфейси MIPI CSI-2 з верифікацією їх розпізнавання на рівні драйверів, встановлення комплексу програмних залежностей та формування файлової структури проєкту у системному каталозі.

3.1.1 Встановлення операційної системи та базова конфігурація Raspberry Pi 5

Основою програмного середовища розроблюваного АПК є операційна система Raspberry Pi OS (64-bit) – офіційний дистрибутив на базі Debian, оптимізований виробником безпосередньо під апаратну архітектуру BCM2712. Для проєкту обрано версію з довгостроковою підтримкою на базі Debian 12 «Bookworm», оскільки саме вона є першою, що містить вбудовану підтримку нового стеку libcamera, необхідного для роботи з двома камерами одночасно через інтерфейси MIPI CSI-2.

Для запису образу операційної системи на microSD-карту використовується офіційна утиліта Raspberry Pi Imager. Початковий екран встановлення операційної системи наведено на рис. 3.1.

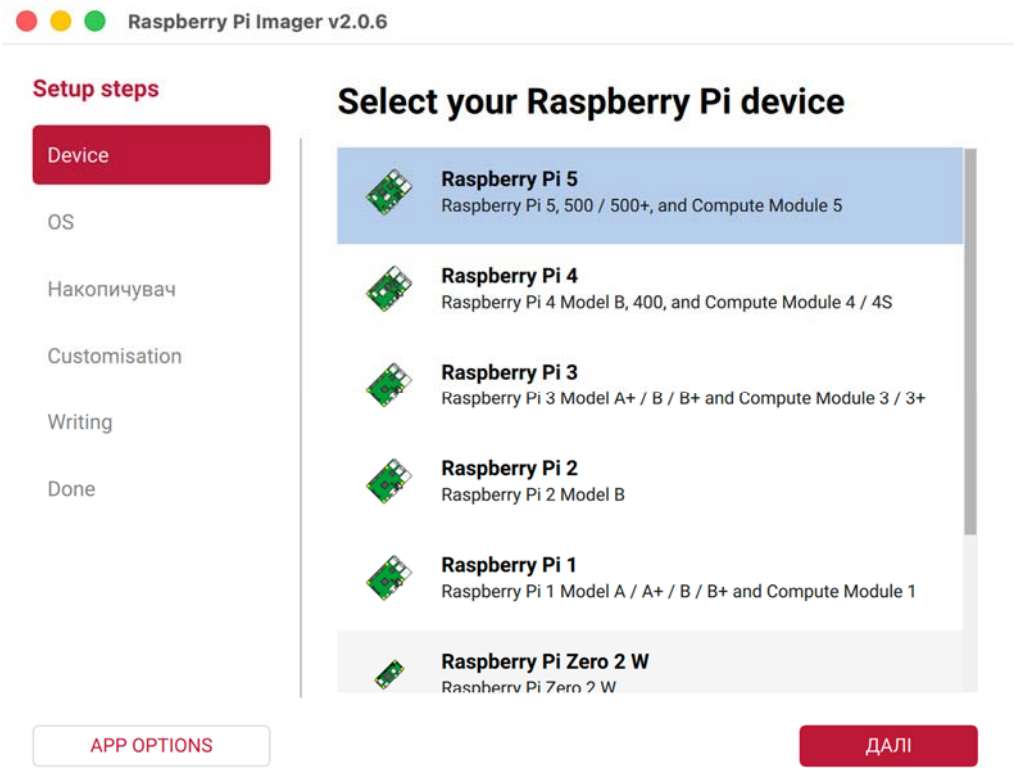


Рисунок 3.1 – Інтерфейс Raspberry Pi Imager із вибраною моделлю пристрою

У налаштуваннях Imager перед записом необхідно задати параметри: ім'я хоста, ім'я користувача та пароль, дані WiFi-мережі, а також активувати SSH. Це дозволяє одразу після першого завантаження підключитись до плати дистанційно без необхідності підключення монітора чи клавіатури.

3.1.2 Підключення та перевірка камерних модулів через MIPI CSI-2

Перед встановленням програмних залежностей необхідно фізично підключити обидва камерних модулі та перевірити їх коректне розпізнавання системою.

Підключення виконується через два порти CAM/DISP на Raspberry Pi 5 за допомогою гнучких FPC-шлейфів (22pin to 15pin, 300 мм). Порт CAM/DISP 0 є ближчим до роз'єму USB-C живлення, CAM/DISP 1 – ближчим до портів USB. При підключенні шлейфа необхідно підняти засувку роз'єму, вставити шлейф контактами донизу та опустити засувку до клацання. Під’єднані шлейфи наведено на рис. 3.2.

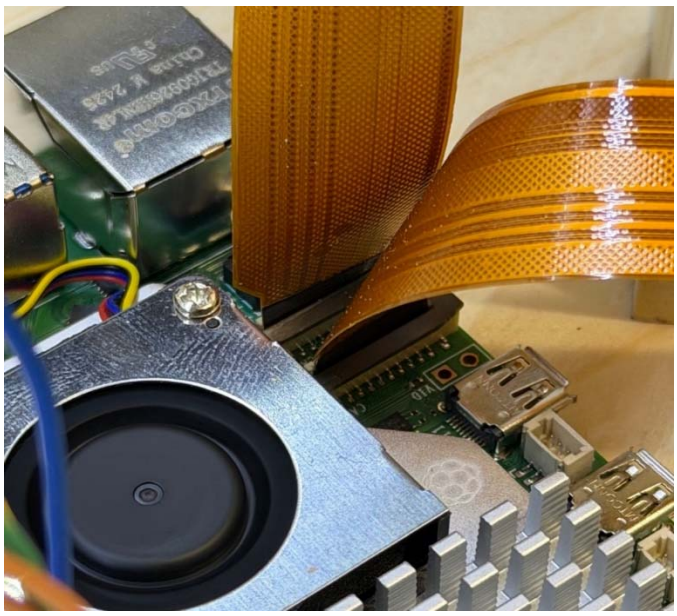


Рисунок 3.2 – Підключення FPC-шлейфів до портів CAM/DISP 0 та CAM/DISP 1 на Raspberry Pi 5

Після підключення камер та завантаження системи виконується перевірка їх розпізнавання. Вивід про працездатність камер наведено на рис. 3.3.

```
dan@pano-cam:~ $ rpicam-hello --list-cameras
Available cameras
-----------------
0 : ov5647 [2592x1944 10-bit GBRG] (/base/axi/pcie@1000120000/rp1/i2c@88000/ov5647@36)
    Modes: 'SGBRG10_CSI2P' : 640x480 [62.50 fps - (16, 0)/2560x1920 crop]
                             1296x972 [46.34 fps - (0, 0)/2592x1944 crop]
                             1920x1080 [32.81 fps - (348, 434)/1928x1080 crop]
                             2592x1944 [15.63 fps - (0, 0)/2592x1944 crop]

1 : ov5647 [2592x1944 10-bit GBRG] (/base/axi/pcie@1000120000/rp1/i2c@80000/ov5647@36)
    Modes: 'SGBRG10_CSI2P' : 640x480 [62.50 fps - (16, 0)/2560x1920 crop]
                             1296x972 [46.34 fps - (0, 0)/2592x1944 crop]
                             1920x1080 [32.81 fps - (348, 434)/1928x1080 crop]
                             2592x1944 [15.63 fps - (0, 0)/2592x1944 crop]
```

Рисунок 3.3 – Виведення двох розпізнаних камер

Виведення команди підтверджує коректне розпізнавання обох камерних модулів на шинах контролера RP1: камера 0 зареєстрована за адресою i2c@88000, камера 1 – за адресою i2c@80000. Обидва сенсори ідентифіковані як ov5647 з максимальною роздільною здатністю 2592×1944 пікселів у 10-бітному форматі GBRG. Для задач зшивання у реальному часі використовуватиметься режим 640 × 480 при 62,50 кадрів на секунду, що забезпечує оптимальний баланс між якістю зображення та обчислювальним навантаженням на процесор.

3.1.3 Встановлення програмних залежностей

Усі програмні компоненти АПК реалізовані на мові Python 3 з використанням бібліотек, що встановлюються через стандартні менеджери пакетів apt та pip. Перелік встановлених бібліотек наведено у табл. 3.1.

Таблиця 3.1 – Перелік залежностей із їхніми призначеннями

Пакет	Версія	Спосіб встановлення	Призначення
python3	3.11+	системна	Мова програмування
python3-opencv	4.8+	apt	Алгоритми комп'ютерного зору (ORB, RANSAC, warpAffine)
python3-numpy	1.24+	apt	Матричні операції, робота з масивами пікселів
python3-gpiozero	2.0+	apt	Абстракція GPIO для кнопок та світлодіодів
python3-lgpio	–	apt	Бекенд gpiozero для Raspberry Pi 5
libcamera-apps	–	apt	Утиліти rpicam-vid, rpicam-hello для роботи з камерами
python3-picamera2	–	apt	Python-обгортка над libcamera (для калібрування)

Після встановлення усіх пакетів виконується верифікація коректності інсталяції шляхом перевірки версій ключових бібліотек безпосередньо з інтерпретатора Python. Результат перевірки наведено на рис. 3.4.

```
dan@pano-cam:~ $ python3 -c "import cv2, numpy, gpiozero; print(cv2.__version__, numpy.__version__)"
4.10.0 2.2.4
dan@pano-cam:~ $
```

Рисунок 3.4 – Виведення команди перевірки версій бібліотек

Виведені версії бібліотек підтверджують успішне встановлення всіх програмних залежностей у системному середовищі Raspberry Pi OS. Версія OpenCV 4.10.0 містить актуальні реалізації алгоритмів ORB, RANSAC та функцій геометричних перетворень, а версія NumPy 2.2.4 забезпечує ефективну роботу з матричними структурами даних. Перелічений програмний стек повністю готовий до подальшого розгортання файлової структури проєкту.

3.1.4 Розгортання файлів проєкту

Усі файли АПК розміщуються у системному каталозі /opt/panorama/, що є стандартним місцем для локально встановленого програмного забезпечення. Розміщення у /opt (а не у домашньому каталозі користувача) є принциповим: systemd-сервіс, що запускає launcher.py при старті системи, працює в окремому контексті та потребує абсолютних шляхів до файлів, незалежних від конкретного користувача. Структуру проєкту наведено на рис. 3.5.

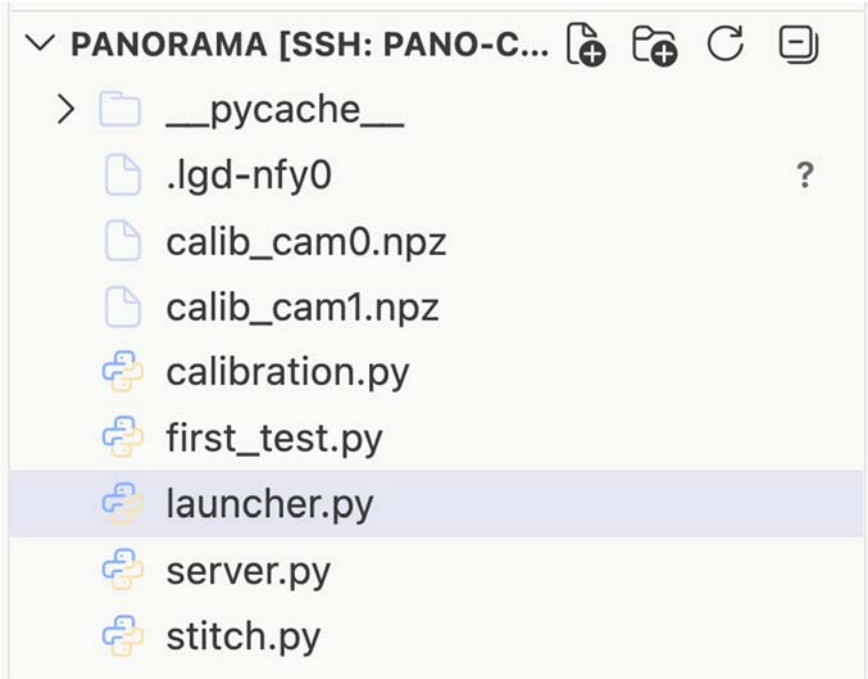


Рисунок 3.5 – Структура проєкту

Файли calib_cam0.npz та calib_cam1.npz генеруються на етапі калібрування камер і на цьому етапі ще відсутні.

3.2 Монтаж та пайка електронної панелі керування

Відповідно до схемотехнічних рішень, обґрунтованих у підрозділі 2.3, на даному етапі здійснюється практична реалізація електронної панелі керування. Процес охоплює розподіл GPIO-пінів між компонентами, розрахунок електричних параметрів кіл індикації та керування, пайку всіх елементів на макетній платі та програмну верифікацію працездатності зібраної схеми.

3.2.1 Схема підключення компонентів до GPIO

Електронна панель керування реалізує фізичний інтерфейс взаємодії оператора з АПК: дві тактові кнопки для запуску та зупинки процесу зшивання і три світлодіоди для індикації поточного стану системи. Усі компоненти підключаються до контролера вводу/виводу RP1 мікрокомп'ютера Raspberry Pi 5 через інтерфейс GPIO. Розподіл пінів наведено в табл. 3.2.

Таблиця 3.2 – Розподіл GPIO-пінів панелі керування

Компонент	BCM-номер	Фізичний пін	Призначення
Кнопка START	GPIO17	11	Запуск процесу зшивання
Кнопка STOP	GPIO27	13	Зупинка процесу зшивання
LED жовтий	GPIO22	15	Режим очікування (Ready)
LED зелений	GPIO23	16	Активна робота (Running)
LED червоний	GPIO24	18	Помилка (Error)
Живлення 3.3 В	–	17	Шина живлення кіл кнопок
Земля GND	–	14	Спільна земля схеми

Усі задіяні піни сконцентровані у компактній групі фізичних пінів 11–18, що спрощує розведення з'єднань на макетній платі та мінімізує довжину провідників до роз'єму GPIO. Візуалізацію розташування задіяних пінів на роз'ємі Raspberry Pi 5 наведено на рис. 3.6.

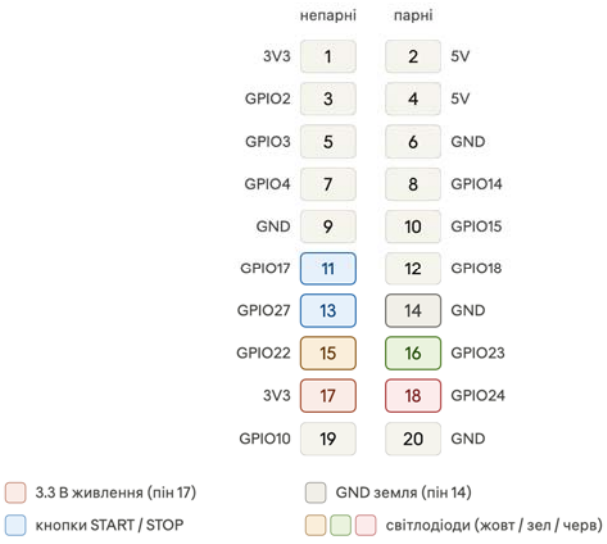


Рисунок 3.6 – Розташування задіяних пінів на роз'ємі GPIO Raspberry Pi 5

Обрана конфігурація залишає вільними решту пінів GPIO, що забезпечує можливість подальшого розширення функціональності панелі керування без конфлікту з існуючою схемою.

Принципову схему панелі керування наведено на рис. 3.7.

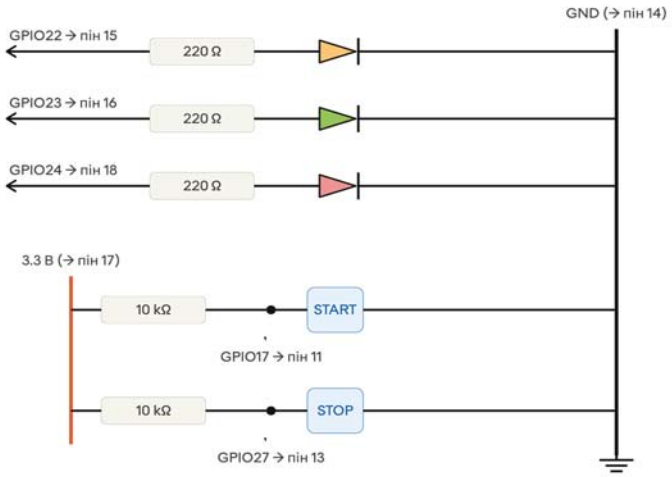


Рисунок 3.7 – Схема панелі керування

Схема містить два незалежних кола: коло індикації та коло керування.

Коло індикації побудоване за однотипною схемою для кожного зі світлодіодів. GPIO-пін виступає джерелом струму. Послідовно з кожним світлодіодом увімкнено захисний резистор номіналом 220 Ом, катод світлодіода підключено до спільної шини GND (фізичний пін 14). Коли контролер RP1 встановлює на піні логічну одиницю (3.3 В), через коло протікає струм, що розраховується за законом Ома:

$$I = \frac{(Usrc - Uled)}{R} = \frac{(3{,}3 - 2{,}0)}{220} \approx 5{,}9\ \text{мА} \qquad (3.1)$$

Отримане значення є безпечним з дворазовим запасом відносно максимально допустимого струму піна (16 мА).

Коло керування реалізовано за схемою з підтягувальним резистором. Один вивід кнопки підключено до інформаційного GPIO-піна, інший – до GND. Між лінією 3,3 В (фізичний пін 17) та інформаційним піном увімкнено підтягувальний резистор 10 кОм. У стані спокою (кнопка не натиснута) резистор утримує на вході

стабільний рівень логічної одиниці (3,3 В), виключаючи появу «висячого потенціалу». При натисканні кнопка замикає пін на землю, рівень падає до 0 В – контролер фіксує логічний нуль. Струм, що стікає через резистор, при цьому становить лише 3,3 / 10000 = 0,33 мА, що повністю виключає перевантаження лінії живлення.

3.2.2 Пайка компонентів на макетній платі

Усі електронні компоненти розпаяні на двосторонній макетній платі зі склотекстоліту розміром 60×80 мм. Використання пайки замість тимчасових з'єднань на безпаяльній макетній платі є принциповою конструктивною вимогою: вібрації та механічні навантаження, неминучі під час експлуатації, призводять до ненадійного контакту в тимчасових з'єднаннях, що може спровокувати хибні спрацювання кнопок або переривання кіл індикації.

Порядок монтажу компонентів на платі:

1) першими впаюються резистори – найнижчі компоненти. Резистори 220 Ом розміщуються у ряд вздовж ліній індикації, резистори 10 кОм – у ряд вздовж ліній керування;

2) далі встановлюються тактові кнопки DIP 4-pin. Корпус кнопки фіксується у чотирьох отворах, що забезпечує механічну стійкість ще до пайки;

3) останніми впаюються світлодіоди. Перед встановленням обов'язково перевіряється полярність: довга нога – анод (+), підключається до резистора; коротка нога – катод (−), підключається до шини GND;

4) з'єднання між компонентами на зворотному боці плати виконуються залудженими перемичками з обрізків ніжок резисторів або окремим монтажним проводом;

5) підключення до GPIO Raspberry Pi 5 здійснюється через набір перемичок типу «мама-тато» довжиною 200 мм, що з'єднують контактні майданчики плати з відповідними пінами роз'єму мікрокомп'ютера.

Особливу увагу під час пайки приділено температурному режиму: для запобігання тепловому пошкодженню напівпровідникових p-n переходів

світлодіодів використовувався паяльник з регульованою температурою (320 °C) та припій з низькотемпературним флюсом. Тривалість контакту жала паяльника з виводом компонента не перевищувала 3 секунд, що відповідає рекомендаціям виробників щодо безпечної пайки оптоелектронних елементів. Після завершення монтажу всі контактні з'єднання візуально перевірені на наявність характерних дефектів – холодної пайки, мостиків між сусідніми майданчиками, недостатнього змочування виводів припоєм. Зворотний бік плати додатково оглянуто з використанням збільшувального скла для виявлення мікротріщин у місцях механічного навантаження. Розпаяну електронну панель наведено нижче на рис. 3.8.

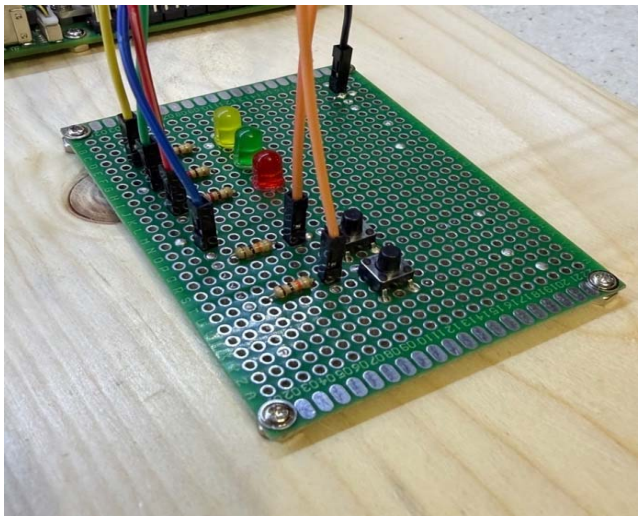


Рисунок 3.8 – Готова електронна панель керування, розпаяна на макетній платі

На фото видно розміщення всіх компонентів: три світлодіоди (жовтий, зелений, червоний) у верхній частині плати з послідовно увімкненими резисторами 220 Ом, дві тактові кнопки DIP 4-pin у центральній частині з підтягувальними резисторами 10 кОм, та набір перемичок «мама-тато», що з'єднують плату з роз'ємом GPIO Raspberry Pi 5.

3.2.3 Перевірка електричних кіл

Після завершення пайки виконується програмна перевірка коректності підключення кожного компонента безпосередньо з терміналу Raspberry Pi 5

засобами бібліотеки gpiozero, без запуску основної програми. Справну роботу світлодіодів наведено на рис. 3.9.

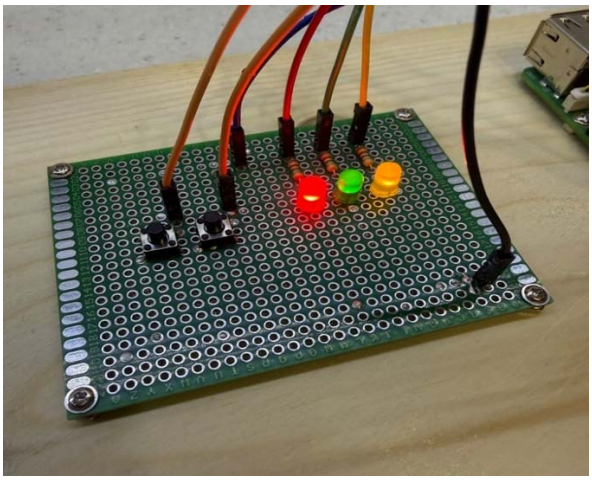


Рисунок 3.9 – Перевірка роботи світлодіодів

Результат тестування підтверджує коректне функціонування кіл індикації. Наступним кроком є верифікація роботи тактових кнопок за допомогою тестового скрипту, що реєструє натискання кнопок START та STOP і виводить відповідні повідомлення у термінал. Результат виконання скрипту наведено на рис. 3.10.

```
dan@pano-cam:~/diploma $ python button_test.py
Press buttons (Ctrl+C to exit)...
START pressed
START pressed
START pressed
STOP pressed
START pressed
STOP pressed
START pressed
STOP pressed
START pressed
```

Рисунок 3.10 – Виведення терміналу під час перевірки кнопок

Підсумовуючи, розраховано номінали захисних резисторів для кіл індикації (220 Ом, робочий струм ~5.9 мА) та підтягувальних резисторів для кіл керування (10 кОм). Усі компоненти – два тактові DIP-перемикачі та три світлодіоди з відповідними резисторами – розпаяні на макетній платі зі склотекстоліту 60 × 80 мм і підключені до контролера RP1 мікрокомп'ютера через інтерфейс GPIO.

Виконана програмна перевірка підтвердила коректне функціонування всіх елементів схеми. Панель керування повністю готова до інтеграції з програмним модулем launcher.py.

3.3 Калібрування відеосенсорів

Перед реалізацією алгоритмічного конвеєра зшивання необхідно усунути оптичні спотворення, притаманні камерним модулям. У цьому підрозділі обґрунтовано необхідність калібрування, описано процедуру збору калібрувальних знімків за допомогою шахової мішені, наведено отримані параметри внутрішньої матриці та коефіцієнти дисторсії для кожного з двох сенсорів, а також продемонстровано результат корекції спотворень.

3.3.1 Теоретична основа та необхідність калібрування

Модулі камер IMX219, що використовуються в АПК, як і будь-які реальні оптичні системи, мають геометричні спотворення зображення – дисторсію. Навіть незначна бочкоподібна або подушкоподібна дисторсія об'єктива призводить до того, що прямі лінії реального простору відображаються на матриці як криві. Якщо подавати такі спотворені кадри безпосередньо на алгоритм зшивання, ORB виявлятиме ключові точки у геометрично неправильних позиціях, матриця гомографії розраховуватиметься з похибкою, а на межі зшивання з'являтимуться помітні геометричні розриви.

Калібрування дозволяє математично описати оптичну систему кожної камери двома наборами параметрів:

1) матриця камери – внутрішні параметри: фокусна відстань у пікселях по осях X та Y (fx, fy) та координати головної точки (cx, cy);

2) коефіцієнти дисторсії – п'ять параметрів (k1, k2, p1, p2, k3), що описують радіальні та тангенціальні спотворення об'єктива.

Маючи ці параметри, функція *cv2.remap()* будує карти переадресації пікселів і усуває дисторсію з кожного кадру ще до передачі його на конвеєр зшивання.

3.3.2 Калібрувальний шаблон

Для калібрування використовується стандартний метод Чжана, реалізований у бібліотеці OpenCV (див. *Додаток Б*). Метод базується на захопленні серії зображень плоского шаблону з відомою геометрією з різних ракурсів. Як калібрувальний шаблон використовується роздрукована на аркуші А4 шахова дошка з параметрами: 9 × 6 внутрішніх кутів, розмір клітинки 30 мм. Нижче, на рис. 3.11 наведено приклад роздрукованої шахівниці для калібрування камер.

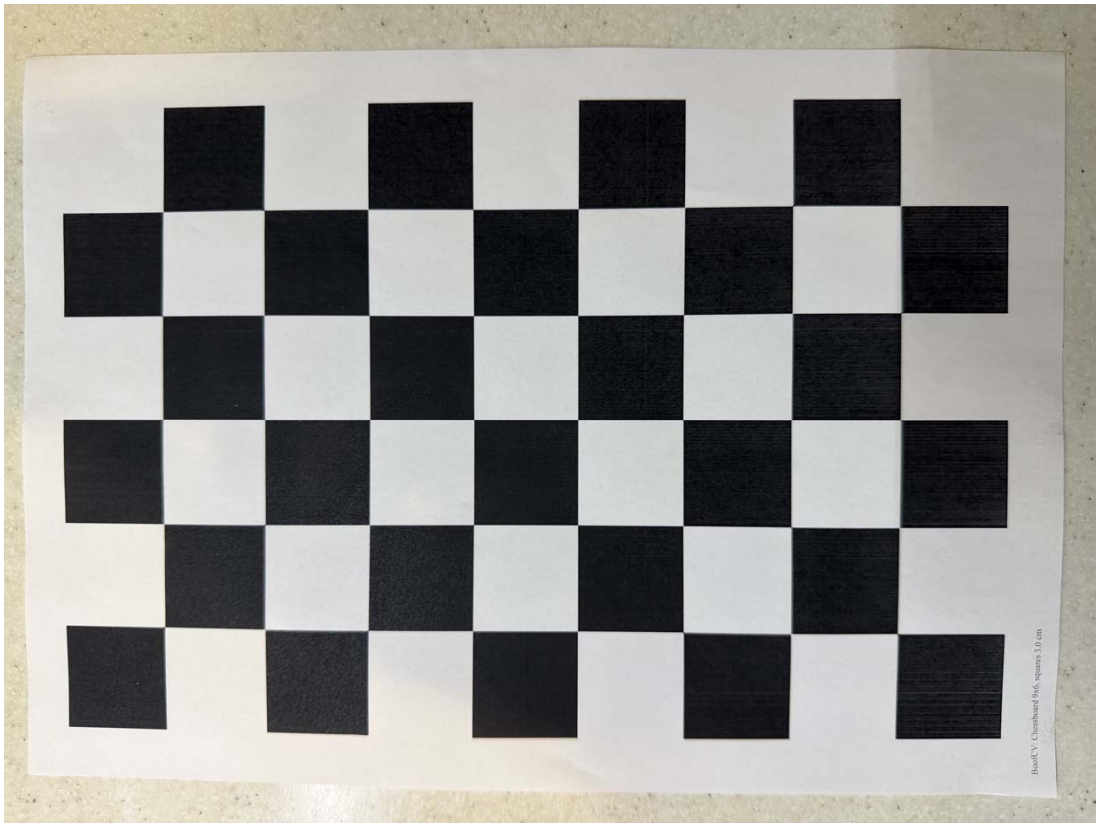


Рисунок 3.11 – Калібрувальна шахова дошка 9 × 6, клітинка 30 мм

Внутрішні кути шахової дошки є точками, де сходяться чотири клітинки. Саме їхні координати OpenCV автоматично знаходить функцією *cv2.findChessboardCorners()* і уточнює до субпіксельної точності функцією *cv2.cornerSubPix()*.

Загальну логіку роботи скрипту калібрування подано у вигляді блок-схеми (див. *Додаток А*).

3.3.3 Методика проведення калібрування

Для отримання якісних результатів калібрування необхідно дотримуватись певної методики при захопленні знімків. Скрипт вимагає 15 успішних кадрів із виявленою шаховою дошкою. Щоб ці кадри рівномірно покривали поле зору камери, шаблон під час зйомки необхідно:

1) переміщувати по всіх зонах кадру – лівий край, правий край, верх, низ, центр;

2) нахиляти під різними кутами відносно камери – по горизонталі та вертикалі;

3) змінювати відстань до камери – ближче та далі.

Коли скрипт виявляє шахову дошку у кадрі, він автоматично захоплює знімок та відображає детектовані кути кольоровими лініями. Статусний рядок у верхній частині вікна показує кількість захоплених кадрів. Приклад процесу калібрування камер наведено на рис. 3.12.

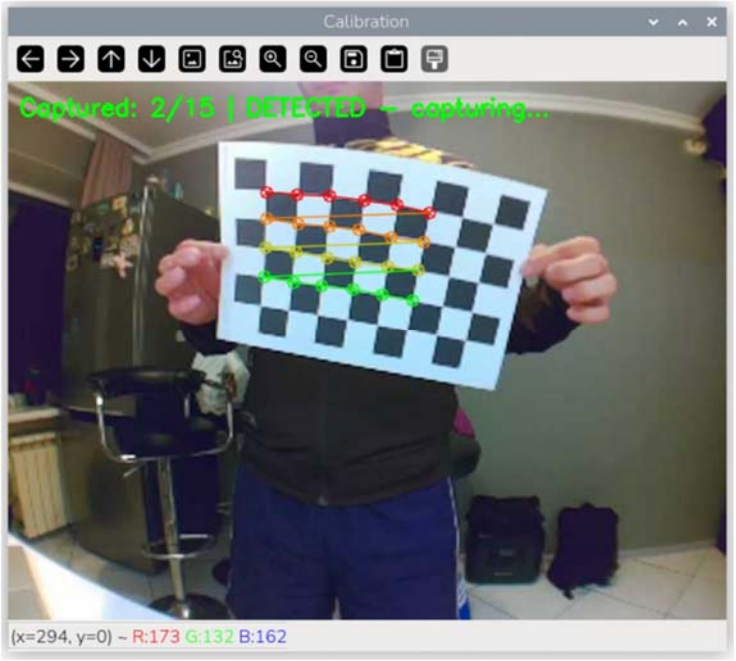


Рисунок 3.12 – Вікно скрипту калібрування в момент автоматичного захоплення кадру з виявленими кутами шахової дошки

На рисунку зафіксовано момент автоматичного захоплення другого з п'ятнадцяти кадрів. Алгоритм успішно розпізнав внутрішні кути шахової

мішені 9 × 6, що підтверджується статусним повідомленням «DETECTED – capturing...» та накладеними маркерами у точках перетину. Процедура повторюється для кожної камери окремо при різних положеннях та кутах нахилу мішені для забезпечення рівномірного покриття поля зору.

3.3.4 Результати калібрування

Після накопичення 15 кадрів скрипт обчислює параметри моделі камери методом найменших квадратів та виводить результати у термінал. Ключовим показником якості калібрування є RMS reprojection error – середньоквадратична похибка перепроєкції, що вимірює середнє відхилення між реальними координатами кутів шахової дошки та координатами, обчисленими через знайдену модель камери. Значення менше 0,5 пікселя вважається якісним результатом. Виведення результатів калібрування для камери 0 наведено на рис. 2.13

```
RMS reprojection error: 0.3467  (good if < 0.5)

CAM_MATRIX = [[382.66661981,   0.        , 317.65987592],
 [  0.        , 383.38128581, 242.79040112],
 [  0.        ,   0.        ,   1.        ]]

DIST_COEFFS = [-0.3493519 ,  0.15212309,  0.00101287, -0.00286866, -0.0358745

Saved to calib_cam0.npz
dan@pi5:~/diploma $
```

Рисунок 3.13 – Результат виконання calibrate.py для камери 0

На рис. 3.14 наведено результат калібрування для камери 1.

```
RMS reprojection error: 0.3968  (good if < 0.5)

CAM_MATRIX = [[432.07603772,   0.        , 314.64159604],
 [  0.        , 434.41183908, 225.96721937],
 [  0.        ,   0.        ,   1.        ]]

DIST_COEFFS = [-0.36522326,  0.11854811,  0.00723288, -0.00040088,  0.04143939]

Saved to calib_cam1.npz
dan@pi5:~/diploma $
```

Рисунок 3.14 – Результат виконання calibrate.py для камери 1

Обидва значення RMS reprojection error – 0,3467 та 0,3968 пікселя – знаходяться в межах норми (< 0,5 пкс), що підтверджує високу якість калібрування. Від'ємне значення коефіцієнта k1 у обох камерах вказує на наявність бочкоподібної дисторсії – характерної для компактних об'єктивів модулів IMX219. Параметри збережено у файли calib_cam0.npz та calib_cam1.npz у каталозі /opt/panorama/.

3.4 Програмна реалізація системи зшивання відеопотоків

Після завершення підготовки апаратного середовища, складання панелі керування та калібрування сенсорів наступним етапом є програмна реалізація системи зшивання. У цьому підрозділі розглянуто загальну архітектуру програмного забезпечення, детально описано алгоритмічний конвеєр обробки кадрів у модулі зшивання, механізм трансляції панорами через HTTP-сервер та логіку модуля автономного керування з GPIO-інтерфейсом.

3.4.1 Загальна архітектура програмного забезпечення

Програмна частина АПК складається з трьох модулів, кожен з яких відповідає за окрему функціональну зону системи:

1) stitch.py – головний модуль: захоплення кадрів, увесь конвеєр обробки зображень та публікація результату;
2) server.py – HTTP-сервер трансляції панорами у браузер;
3) launcher.py – модуль автономного керування: обробка GPIO-кнопок, управління процесом stitch.py та світлодіодна індикація.

Узагальнені блок-схеми роботи основних програмних модулів наведено в *Додатку А*.

Ключовим архітектурним рішенням є те, що launcher.py запускає stitch.py не як імпортований модуль, а як окремий підпроцес через *subprocess.Popen()*. Це дає змогу повністю зупиняти та перезапускати увесь конвеєр обробки відео натисканням фізичної кнопки, не перериваючи роботу самого launcher.py і не втрачаючи контроль над GPIO та світлодіодами.

3.4.2 Багатопотокове захоплення кадрів

Захоплення відеопотоків з двох камер реалізовано у двох паралельних потоках виконання. Кожен потік запускає окремий процес rpicam-vid через системний виклик та зчитує MJPEG-потік з його стандартного виведення побайтово (див. *Додаток Б*). На рис. 3.15 наведено приклад коду зі *stitch.py*.

```
cmd = [
    "rpicam-vid", "--camera", str(cam_id),
    "--width", str(W), "--height", str(FRAME_H),
    "--framerate", "30", "--codec", "mjpeg",
    "--output", "-", "--nopreview", "--timeout", "0",
]
proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL, bufsize=0)
```

Рисунок 3.15 – Приклад коду зі stitch.py

Використання MJPEG як транспортного кодеку замість сирого YUV є свідомим компромісом: апаратний MJPEG-кодек IMX219 майже не навантажує процесор, тоді як передача нестисненого YUV потребувала б удвічі більшої пропускної здатності шини даних. Декодування MJPEG назад у BGR виконується функцією *cv2.imdecode()* вже у потоці захоплення.

Синхронізація між потоками захоплення та головним потоком обробки реалізована через спільні змінні f0[0] та f1[0], захищені окремими об'єктами threading.Lock. Головний цикл зшивання щоразу читає останній доступний кадр з кожного буфера. На рис. 3.16 наведено приклад коду з файлу *stitch.py*.

```
with lock0:
    fl = f0[0].copy() if f0[0] is not None else None
with lock1:
    fr = f1[0].copy() if f1[0] is not None else None
```

Рисунок 3.16 – Приклад коду зі stitch.py

Логіку роботи основного модуля зшивання *stitch.py* подано у вигляді блок-схеми (див. *Додаток А*).

Таким чином, потоки захоплення безперервно оновлюють спільні буфери, а головний цикл зшивання отримує з них найактуальніші кадри без блокувань та затримок.

3.4.3 Конвеєр обробки зображень

Після захоплення кожна пара кадрів проходить через послідовний конвеєр перетворень.

Ректифікація усуває геометричні спотворення об'єктива на основі параметрів калібрування. Карти переадресації пікселів обчислюються одноразово при старті програми. Приклад коду наведено на рис. 3.17.

```
ud_l1, ud_l2 = cv2.initUndistortRectifyMap(
    CAM_MTX_L, DIST_L, None, new_mtx_l, (W, FRAME_H), cv2.CV_32FC1)
```

Рисунок 3.17 – Приклад коду зі stitch.py

Безпосереднє усунення дисторсії з кожного кадру виконується функцією *cv2.remap()* за попередньо обчисленими картами, що є значно ефективнішим порівняно з викликом *cv2.undistort()* для кожного кадру окремо.

Циліндрична проекція застосовується до ректифікованих кадрів для їх відображення на циліндричну поверхню. Це перетворення вирівнює горизонтальні лінії у зоні перекриття двох камер та усуває перспективні спотворення на краях кадрів, що суттєво покращує якість зшивання при великих кутах між камерами.

Детектор ORB знаходить до 3000 ключових точок у кожному кадрі. Для їх зіставлення використовується метод найближчого сусіда з перевіркою відношення відстаней (коефіцієнт 0,75) – так звана перевірка Лоу, яка наведена на рис. 3.18.

```
raw = bf.knnMatch(des_r, des_l, k=2)
good = [p[0] for p in raw if len(p)==2 and p[0].distance < 0.75*p[1].distance]
```

Рисунок 3.18 – Приклад коду зі stitch.py

На основі відібраних пар точок алгоритм RANSAC обчислює матрицю часткового афінного перетворення estimateAffinePartial2D, що моделює поворот, масштабування та зсув між двома кадрами. При запуску системи було знайдено

125 надійних відповідностей зі 154 кандидатів, залишковий кут повороту між камерами склав −4,08 градуса.

Зшивання виконується функцією *stitch()* при кожному кадрі. Права камера трансформується на полотно функцією *cv2.warpAffine()*, ліва розміщується на тому ж полотні зі зміщенням. У зоні перекриття застосовується зважене змішування на основі мінімально-енергетичного шву та карт відстаней. Приклад коду наведено на рис. 3.19.

```
pano = (canvas_left.astype(np.float32) * wL +
        warped_right.astype(np.float32) * wR).astype(np.uint8)
```

Рисунок 3.19 – Приклад коду зі stitch.py

Ваги wL та wR обчислені одноразово при ініціалізації конвеєру і зберігаються як постійні масиви, що виключає їх перерахунок у кожному кадрі.

3.4.4 Ініціалізація конвеєру та рекалібрування

При старті stitch.py виконує фазу ініціалізації: чекає 2 секунди на прогрів камер, після чого робить до 30 спроб знайти геометричну трансформацію між кадрами. Як тільки трансформацію знайдено, функція build_pipeline() одноразово обчислює розмір полотна, зміщення, шов та ваги змішування – усі дані, що залишаються незмінними протягом усієї сесії роботи.

Якщо у процесі роботи якість зшивання погіршилась (наприклад, через зміну освітлення), користувач може ініціювати рекалібрування натисканням кнопки у вебінтерфейсі без перезапуску системи. Подія передається через server.recalib_request – об'єкт threading.Event, який головний цикл перевіряє після кожного кадру.

3.4.5 Результати роботи конвеєру

Під час тестування система стабільно працює зі швидкістю 24,0 FPS при роздільній здатності вхідних кадрів 640 × 480 пікселів. Мінімальне значення,

зафіксоване під час тестування, склало 23,6 FPS. Розмір результуючого полотна панорами до кадрування становить 927 × 569 пікселів, після застосування функції обрізання до найбільшого внутрішнього прямокутника – 894 × 461 піксель. Успішна робота програми наведена на рис. 3.20.

```
dan@pano-cam:/opt/panorama $ python stitch.py
Warming up cameras...
Computing alignment...
[M] inliers 125/154 | residual rotation -4.08deg
[canvas] (927, 569) offset (0, 61) crop (0, 61, 894, 461)
Alignment found on attempt 1
Streaming at http://192.168.31.250:8000/
24.0 FPS
```

Рисунок 3.20 – Виведення терміналу при запуску stitch.py: фаза ініціалізації, параметри полотна та стабільний FPS

Архітектура з трьох модулів (stitch.py, server.py, launcher.py) забезпечує чіткий поділ відповідальності між обробкою відео, трансляцією та керуванням. Багатопотокове захоплення через rpicam-vid з MJPEG-транспортом мінімізує навантаження на процесор при захопленні. Конвеєр обробки – ректифікація, циліндрична проєкція, ORB + RANSAC, зшивання з мінімально-енергетичним швом – виконується на ресурсах центрального процесора ARM Cortex-A76 і забезпечує стабільну швидкість обробки 24,0 FPS, що відповідає вимогам реального часу.

3.5 Реалізація автономного керування та GPIO-інтерфейсу

Після реалізації та верифікації конвеєра зшивання необхідно забезпечити можливість керування системою без підключення до мережевого терміналу чи графічного середовища. У цьому підрозділі описано концепцію автономної роботи комплексу, програмну логіку модуля керування та реєстрацію системного сервісу для автоматичного запуску при подачі живлення.

3.5.1 Концепція автономної роботи

Ключовою експлуатаційною вимогою до АПК є повна автономність – система має функціонувати без підключення монітора, клавіатури чи мережевого терміналу. Оператор взаємодіє з комплексом виключно через фізичні кнопки на панелі керування, а поточний стан системи відображається світлодіодною індикацією.

Для реалізації цієї концепції розроблено модуль launcher.py, який виконує три функції:

– керує запуском та зупинкою процесу зшивання відео у відповідь на натискання кнопок;

– відображає поточний стан системи через три світлодіоди;

– автоматично відновлює роботу після збоїв завдяки інтеграції зі systemd.

Алгоритм роботи модуля автономного керування launcher.py наведено у блок-схемі (див. *Додаток А*).

3.5.2 Реалізація модуля launcher.py

Центральним елементом модуля є функція *start_stitcher(),* яка запускає stitch.py як окремий підпроцес та передає йому власну групу процесів (див. *Додаток Б*). Приклад коду з *launcher.py* наведено на рис. 3.21.

```
proc = subprocess.Popen(
    STITCH_CMD, cwd=str(BASE_DIR),
    stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL,
    preexec_fn=os.setsid,
)
```

Рисунок 3.21 – Приклад коду зі launcher.py

Використання *os.setsid()* є принциповим: воно створює для stitch.py окрему групу процесів. Завдяки цьому при зупинці сигнал SIGINT надсилається не лише безпосередньо до stitch.py, а до всієї групи, включно з дочірніми процесами

rpicam-vid, які він породжує. Це гарантує чисте звільнення камерних ресурсів. Приклад коду наведено на рис. 3.22.

```
os.killpg(os.getpgid(p.pid), signal.SIGINT)
p.wait(timeout=5)
```

Рисунок 3.22 – Приклад коду зі launcher.py

Відстеження стану запущеного підпроцесу виконується в окремому потоці *_supervise()*. Він чекає завершення фази прогріву, встановлює зелений LED і блокується на *p.wait()* – системному виклику, що повертає керування лише після завершення підпроцесу. Це дозволяє миттєво реагувати на аварійне завершення stitch.py без циклічного опитування. Приклад коду наведено на рис. 3.23.

```
def _supervise():
    time.sleep(WARMUP_S)
    with lock:
        p = proc
    if p is None:
        return
    if p.poll() is not None:
        set_state_error()
        return
    set_state_running()
    rc = p.wait()
    with lock:
        if proc is not p:
            return
        proc = None
    if rc in (0, -signal.SIGINT, -signal.SIGTERM):
        set_state_ready()
    else:
        set_state_error()
```

Рисунок 3.23 – Приклад коду зі launcher.py

Логіка функції *_supervise()* забезпечує коректну обробку всіх можливих сценаріїв завершення підпроцесу: штатне завершення після натискання кнопки STOP повертає систему у стан готовності, тоді як аварійне завершення з ненульовим кодом повернення переводить її у стан помилки з відповідною світлодіодною індикацією.

3.5.3 Захист від хибних спрацювань кнопок

При тестуванні панелі керування було виявлено характерну проблему: електромагнітний шум у момент замикання контактів кнопки START наводився на лінію GPIO27 та викликав серію хибних спрацювань кнопки STOP одразу після натискання START.

Апаратне усунення цієї проблеми потребувало б додаткових конденсаторів на платі. Натомість реалізовано програмне вирішення: протягом 5 секунд після фіксації натискання START будь-які події від кнопки STOP ігноруються. Приклад коду наведено на рис. 3.24.

```
IGNORE_STOP_S = 5

def stop_stitcher():
    if time.time() - start_ts < IGNORE_STOP_S:
        print("[launcher] STOP ignored (too soon after START)")
        return
    ...
```

Рисунок 3.24 – Приклад коду зі launcher.py

Значення 5 секунд обрано з запасом відносно реального часу ініціалізації ститчера (близько 2–3 секунд), що одночасно вирішує дві задачі: блокує хибні спрацювання та унеможливлює зупинку системи до завершення фази прогріву камер.

3.5.4 Інтеграція зі systemd

Щоб launcher.py запускався автоматично при кожному вмиканні Raspberry Pi 5 без будь-якого втручання користувача, його реєструють як системну службу systemd. Файл конфігурації служби розміщується за шляхом /etc/systemd/system/panorama.service. Конфігурація файлу наведена на рис. 3.25.

```
[Unit]
Description=Panorama Launcher (GPIO + stitcher)
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=pi
Group=pi
WorkingDirectory=/opt/panorama
ExecStart=/usr/bin/python3 -u /opt/panorama/launcher.py
Restart=on-failure
RestartSec=3
KillMode=mixed
TimeoutStopSec=10

[Install]
WantedBy=multi-user.target
```

Рисунок 3.25 – код із panorama.service

Директива After=network-online.target гарантує, що служба запускається лише після того, як система отримала IP-адресу – це необхідно для коректного визначення локальної IP-адреси у server.py при старті вебсервера. Параметр Restart=on-failure забезпечує автоматичне відновлення служби у разі її аварійного завершення з паузою 3 секунди між спробами. KillMode=mixed при зупинці служби надсилає SIGTERM безпосередньо до launcher.py, а потім SIGKILL до всієї групи процесів – це гарантує завершення і дочірнього stitch.py з його rpicam-vid. Успішна робота служби наведена на рис. 3.26.

```
dan@pano-cam:~ $ sudo systemctl status panorama.service
● panorama.service - Panorama Launcher (GPIO + stitcher)
     Loaded: loaded (/etc/systemd/system/panorama.service; enabled; preset: enabled)
     Active: active (running) since Fri 2026-06-05 01:55:08 EEST; 42min ago
 Invocation: c6c6ae52b0544423981fc1ad512d1615
   Main PID: 1879 (python3)
      Tasks: 46 (limit: 4265)
        CPU: 43.286s
     CGroup: /system.slice/panorama.service
             ├─1879 /usr/bin/python3 -u /opt/panorama/launcher.py
             ├─2538 /usr/bin/python3 -u /opt/panorama/stitch.py
             ├─2545 rpicam-vid --camera 1 --width 640 --height 480 --framerate 30 --codec mjpeg --output - --nopreview --timeout 0
             └─2546 rpicam-vid --camera 0 --width 640 --height 480 --framerate 30 --codec mjpeg --output - --nopreview --timeout 0

Jun 05 01:55:08 pano-cam systemd[1]: Started panorama.service - Panorama Launcher (GPIO + stitcher).
Jun 05 01:55:09 pano-cam python3[1879]: [launcher] ready. Press START.
Jun 05 02:37:32 pano-cam python3[1879]: [launcher] START
Jun 05 02:37:32 pano-cam python3[1879]: [launcher] STOP ignored (too soon after START)
dan@pano-cam:~ $
```

Рисунок 3.26 – Виведення systemctl status panorama.service
з активним станом служби

Після успішної активації служби жовтий світлодіод на панелі керування засвічується автоматично при кожному вмиканні живлення Raspberry Pi 5,

підтверджуючи готовність системи до роботи без будь-яких додаткових дій з боку оператора.

3.6 Вебсервер трансляції панорами в реальному часі

Завершальним компонентом АПК є вебсервер, що забезпечує доступ до панорамного зображення з будь-якого пристрою у локальній мережі. У цьому підрозділі обґрунтовано вибір протоколу трансляції, описано архітектуру серверного модуля та механізм мережевого виявлення пристрою за допомогою mDNS.

3.6.1 Вибір протоколу трансляції

Для передачі відеопотоку панорами на клієнтські пристрої обрано протокол MJPEG через звичайний HTTP. Суть протоколу полягає у безперервній передачі послідовності JPEG-кадрів у межах одного HTTP-з'єднання з використанням типу вмісту multipart/x-mixed-replace. Браузер, отримавши перший кадр, замінює зображення наступним щойно воно надходить – користувач бачить плавне відео без будь-яких плагінів чи додаткового програмного забезпечення.

Перевагами такого підходу є підтримка у будь-якому сучасному браузері, сумісність з усіма пристроями у локальній мережі – ПК, ноутбук, смартфон – та відсутність необхідності у зовнішніх залежностях на кшталт FFmpeg чи GStreamer.

3.6.2 Архітектура модуля server.py

Модуль реалізовано на базі стандартного класу ThreadingHTTPServer з бібліотеки http.server, що входить до складу Python без додаткового встановлення. Маршрути вебсервера наведені в табл. 3.5.

Таблиця 3.5 – HTTP-маршрути вебсервера

Маршрут	Відповідь	Призначення
/	HTML-сторінка	Головна сторінка з відеопотоком
/stream	multipart/x-mixed-replace	MJPEG-потік панорами
/recalibrate	text/plain «ok»	Запит рекалібрування

Обмін даними між stitch.py та сервером організовано через спільний буфер останнього JPEG-кадру, захищений м'ютексом. Ститчер публікує кожен готовий кадр викликом server.update_frame(), не очікуючи поки клієнт його отримає. Завдяки цьому швидкість обробки відео повністю незалежна від кількості підключених клієнтів та швидкості їхнього мережевого з'єднання.

Логіку обробки HTTP-запитів у модулі server.py подано у блок-схемі (див. *Додаток А*).

3.6.3 MJPEG-потік та рекалібрування

Обробник маршруту /stream після надсилання відповідного заголовка входить у нескінченний цикл і безперервно відправляє клієнту останній доступний кадр з буфера. Затримка між кадрами обмежує частоту відправки до 30 кадрів на секунду незалежно від швидкості з'єднання. Відключення клієнта обробляється перехопленням мережевих винятків – з'єднання закривається без жодного впливу на роботу ститчера та інших підключених клієнтів.

Вебінтерфейс містить кнопку «Recalibrate». При її натисканні браузер надсилає запит на маршрут /recalibrate, сервер встановлює внутрішній прапор, який головний цикл stitch.py перевіряє після кожного кадру. При його встановленні виконується повторний пошук ключових точок та перебудова конвеєру зшивання без перезапуску всього процесу. Це дозволяє оператору відновити якість зшивання дистанційно, не маючи фізичного доступу до пристрою. Успішна робота вебсерверу та камер наведено на рис. 3.27.

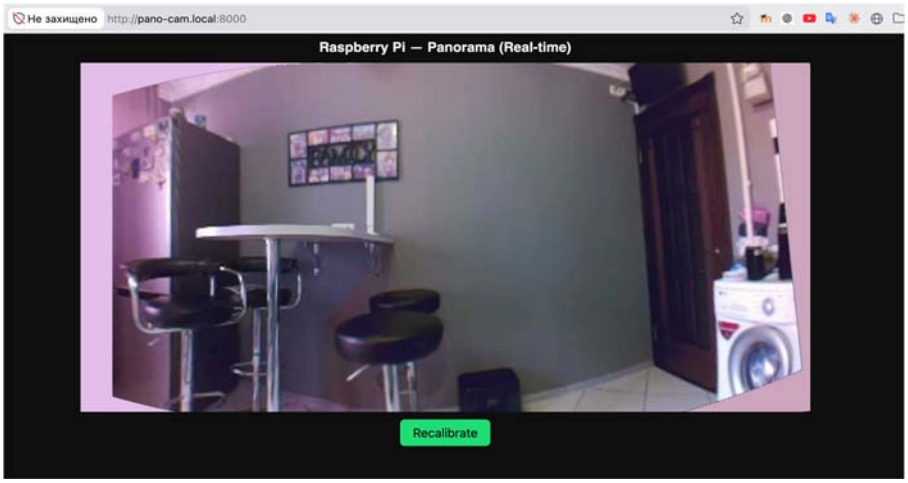


Рисунок 3.27 – Вебінтерфейс перегляду панорами у браузері з кнопкою рекалібрування

На рисунку продемонстровано вебінтерфейс, доступний за адресою pano-cam.local:8000. Панорамне зображення транслюється у режимі реального часу, а кнопка «Recalibrate» дозволяє оператору ініціювати повторне калібрування гомографії безпосередньо з браузера без необхідності доступу до терміналу Raspberry Pi.

3.6.4 Доступ до потоку через mDNS

Raspberry Pi OS за замовчуванням містить встановлений та активований демон Avahi – реалізацію протоколу мережевого виявлення mDNS/DNS-SD. Принцип роботи Avahi полягає в автоматичному оголошенні пристрою у локальній мережі під іменем формату <hostname>.local без необхідності налаштування централізованого DNS-сервера. Оскільки ім'я хоста Raspberry Pi 5 виставлено як pano-cam, Avahi реєструє пристрій у мережі під адресою pano-cam.local і автоматично прив'язує її до поточної IP-адреси пристрою (див. *Додаток Б*).

Це означає, що для перегляду панорами оператору достатньо відкрити у браузері адресу pano-cam.local:8000 – вводити IP-адресу не потрібно. Ім'я pano-cam.local працює стабільно навіть якщо IP-адреса Raspberry Pi 5 змінилась після перезавантаження маршрутизатора, оскільки Avahi автоматично оголошує нову адресу у мережі. Це суттєво спрощує експлуатацію системи, особливо при підключенні з мобільних пристроїв. Статус служби наведено на рис. 3.28 нижче.

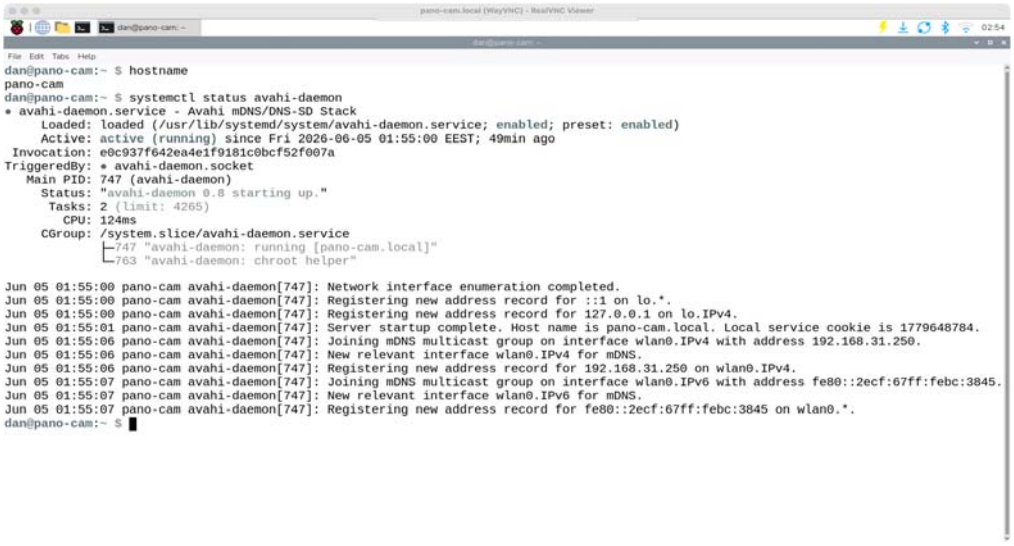


Рисунок 3.28 – Статус служби avahi-daemon: hostname pano-cam.local зареєстровано у локальній мережі

Сумісність охоплює всі основні платформи: Windows 10/11, macOS, iOS, Android та Linux. Єдиним винятком є старіші версії Windows – у такому випадку як резервний варіант залишається пряме введення IP-адреси, яку stitch.py виводить у термінал при старті.

Висновки до розділу 3

У третьому розділі здійснено повну практичну реалізацію апаратно-програмного комплексу панорамного відеоспостереження на базі мікрокомп'ютера Raspberry Pi 5.

Підготовка апаратного середовища включала встановлення операційної системи Raspberry Pi OS (64-bit) на базі Debian 12 «Bookworm», конфігурацію мережевого доступу та підключення двох камерних модулів IMX219 через інтерфейси MIPI CSI-2. Коректність підключення підтверджена розпізнаванням обох сенсорів на рівні драйверів стеку libcamera.

Виконано монтаж та пайку електронної панелі керування на макетній платі зі склотекстоліту 60×80 мм. Схема включає два тактові DIP-перемикачі (GPIO17, GPIO27) та три світлодіоди (GPIO22, GPIO23, GPIO24) з відповідними захисними та підтягувальними резисторами. Програмна верифікація підтвердила коректне функціонування всіх елементів схеми.

Проведено калібрування обох відеосенсорів за методом шахової мішені 9 × 6 на основі 15 захоплених кадрів. Отримані значення середньоквадратичної похибки перепроєкції склали 0,3467 та 0,3968 пікселів для камер 0 та 1 відповідно, що підтверджує високу якість калібрування.

Програмна частина АПК реалізована у вигляді трьох взаємодіючих модулів. Модуль stitch.py реалізує повний конвеєр обробки зображень: ректифікацію, циліндричну проекцію, виявлення ключових точок алгоритмом ORB, обчислення афінного перетворення та фотометричне згладжування шву на основі Distance Transform. Досягнута стабільна швидкість обробки 24,0 FPS при вихідному розмірі полотна 894 × 461 пікселів. Модуль server.py забезпечує трансляцію панорами у браузер за протоколом MJPEG через HTTP з підтримкою mDNS-адресації pano-cam.local:8000. Модуль launcher.py реалізує автономне керування системою через GPIO-інтерфейс із захистом від хибних спрацювань кнопок.

Реєстрація системного сервісу panorama.service у менеджері systemd забезпечує автоматичний запуск комплексу при подачі живлення без участі оператора.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі розроблено апаратно-програмний комплекс на базі мікрокомп'ютера Raspberry Pi 5 для об'єднання відеопотоків з двох камер у єдину панораму в режимі реального часу. Результати виконаної роботи дозволяють зробити наступні висновки.

У першому розділі проведено аналіз предметної області та існуючих рішень систем панорамного відеоспостереження. Встановлено, що традиційні підходи – PTZ-камери та камери типу «риб'яче око» – мають суттєві концептуальні обмеження: перші генерують тимчасові «сліпі зони» внаслідок механічного повороту об'єктива, другі спричиняють критичні геометричні спотворення зображення. На основі порівняльного аналізу обґрунтовано перевагу багатосенсорного підходу з програмним зшиванням відеопотоків. Проведено порівняння апаратних платформ, за результатами якого доведено, що мікрокомп'ютер Raspberry Pi 5 з процесором ARM Cortex-A76 у CPU-залежних задачах комп'ютерного зору перевершує платформу NVIDIA Jetson Nano у 2,5–3,0 рази та є оптимальним вибором завдяки наявності двох незалежних інтерфейсів MIPI CSI-2. Сформовано комплекс технічних вимог до програмного та апаратного забезпечення комплексу.

У другому розділі побудовано математичну модель алгоритмічного конвеєра зшивання відеопотоків. Обґрунтовано вибір алгоритму ORB для виявлення та опису локальних ознак на основі методу центроїда інтенсивності та бінарного дескриптора rBRIEF з метрикою Геммінга. Сформовано алгебраїчну модель 2D-проєктивного перетворення на основі матриці гомографії та алгоритму DLT. Для підвищення робастності обчислень щодо хибних відповідностей інтегровано стохастичний алгоритм RANSAC. Розроблено метод фотометричного згладжування артефактів на швах зшивання на основі зваженого блендингу з картою відстаней. Спроєктовано апаратну частину комплексу: схему підключення двох камерних модулів через FPC-шлейфи до портів CAM/DISP 0 та CAM/DISP 1, систему активного охолодження процесора BCM2712 та електронну панель керування з фізичними кнопками та RGB-індикацією на базі портів GPIO з

розрахованими захисними резисторами (220 Ом для LED-кіл та 10 кОм для кіл підтягування кнопок).

У третьому розділі здійснено повну практичну реалізацію АПК. Виконано підготовку апаратного середовища, монтаж та пайку електронної панелі керування на макетній платі зі склотекстоліту 60×80 мм, калібрування відеосенсорів за методом шахової мішені 9×6, за результатами якого отримано значення середньоквадратичної похибки перепроєкції 0,3467 та 0,3968 пікселів для камер 0 та 1 відповідно. Програмна реалізація забезпечує стабільну швидкість обробки 24,0 FPS при вихідному розмірі полотна 894×461 пікселів. Реєстрація сервісу panorama.service у менеджері systemd забезпечує повністю автономний запуск комплексу без участі оператора. Трансляція панорами у браузер здійснюється за протоколом MJPEG через HTTP з підтримкою mDNS-адресації pano-cam.local:8000, що забезпечує доступ з будь-якого пристрою локальної мережі без введення IP-адреси.

Практична значимість роботи полягає у тому, що розроблений комплекс є економічно доступною та конфігурованою альтернативою закритим комерційним рішенням таких виробників, як Hikvision та Dahua. Відкрита архітектура системи дозволяє вільно модифікувати алгоритми комп'ютерного зору та інтегрувати власні аналітичні модулі. Пристрій функціонує у повністю автономному режимі без необхідності підключення монітора чи клавіатури, забезпечує керування через фізичну панель та трансляцію результуючого відеопотоку через вебінтерфейс у локальній мережі.

Отримані результати підтверджують доцільність використання сучасних одноплатних мікрокомп'ютерів у поєднанні з алгоритмами бібліотеки OpenCV для побудови ефективних систем Edge Computing у сфері відеоспостереження. Перспективним напрямом подальшого розвитку комплексу є розширення кількості підключених камерних модулів, застосування апаратного прискорення засобами VideoCore VII, а також інтеграція модулів аналітики на основі нейронних мереж для детектування та класифікації об'єктів у панорамному відеопотоці в реальному часі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Невідомий Д. О., Пузирьов С. В. Система об’єднання відеопотоків у реальному часі на базі Raspberry Pi. *Могилянські читання – 2025* : тези доп. XXVIII Всеукр. наук.-практ. конф., Миколаїв, 10–14 листоп. 2025 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2025. С. 77–80.

2. AI in video surveillance market report 2024-2030 [285 pages & 259 tables]. MarketsandMarkets. URL: https://www.marketsandmarkets.com/Market-Reports/ai-in-video-surveillance-market-84216922.html (Last accessed: 15.05.2026).

3. The rising threat of security camera blind spots in 2025. Johns Brothers Security. URL: https://johnsbrotherssecurity.com/security-camera-blind-spots/ (Last accessed: 15.05.2026).

4. Fisheye cameras vs multi-sensor cameras: what is best for your needs?. Rigility. URL: https://www.rigility.com/fisheye-cameras-vs-multi-sensor-cameras-what-is-best-for-your-needs/ (Last accessed: 15.05.2026).

5. Hikvision vs dahua: key differences for home and business security in. Smarket. URL: https://www.smarket.com.au/blogs/diy-for-beginners/hikvision-vs-dahua-key-differences-for-home-and-business-security-in-australia (Last accessed: 15.05.2026).

6. Jetson nano vs raspberry pi 5 for AI: the ultimate performance and val. ThinkRobotics.com. URL: https://thinkrobotics.com/blogs/learn/jetson-nano-vs-raspberry-pi-5-for-ai-the-ultimate-performance-and-value-comparison (Last accessed: 15.05.2026).

7. Introducing: raspberry pi 5! - raspberry pi. Raspberry Pi. URL: https://www.raspberrypi.com/news/introducing-raspberry-pi-5/ (Last accessed: 15.05.2026).

8. Chaichana K., Netinant P., Pukdesree S. Design and Implementation of an IoT-Based Surveillance System using Raspberry Pi, Camera, and Motion Sensor // ICEEG 2023. Plymouth, UK, 2023. P. 200–204. DOI: 10.1145/3599609.3599638.

9. Suchý I., Turčaník M. Implementation and Evaluation of Video Codecs in OpenCV for Real-Time Object Detection. *2023 Communication and Information Technologies (KIT)*. 2023. DOI: 10.1109/KIT59097.2023.10297064.

10. Zhu J., Guo J., Ding K., Wang G., Zhou Y., Li W. Real-Time Panoramic Surveillance Video Stitching Method for Complex Industrial Environments. *Sensors (Basel, Switzerland).* 2025. Vol. 26. № 1:186. DOI: 10.3390/s26010186.

11. Wang P., Qin P., Chai R., Zeng J., Zhao P., Chen Z., Han B. End-to-End Online Video Stitching and Stabilization Method Based on Unsupervised Deep Learning. *Applied Sciences*. 2025. Vol. 15. № 11:5987. DOI: 10.3390/app15115987.

12. Soltanpour S., Cheng I., Basu A. A Survey on Feature-Based and Deep Image Stitching. *VISIGRAPP 2025: Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications.* 2025. Vol. 3. P. 425–432. DOI: 10.5220/0013368500003912.

13. Xu S., Chen S., Xu R., Wang C., Lu P., Guo L. Local feature matching using deep learning: A survey. Information Fusion. 2024. Vol. 107. Article 102344. DOI: 10.1016/j.inffus.2024.102344.

14. Sharma S. K., Jain K., Shukla A. K. A Comparative Analysis of Feature Detectors and Descriptors for Image Stitching. Applied Sciences. 2023. Vol. 13. № 10. Article 6015. DOI: 10.3390/app13106015.

15. Liao Y., Di Y., Zhu K., Zhou H., Lu M., Zhang Y., Duan Q., Liu J. Local feature matching from detector-based to detector-free: a survey. Applied Intelligence. 2024. Vol. 54. № 5. P. 3954–3989. DOI: 10.1007/s10489-024-05330-3.

16. Barath D., Mishkin D., Polic M., Förstner W., Matas J. A Large Scale Homography Benchmark // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Vancouver, Canada, 2023. P. 21360–21370. DOI: 10.1109/CVPR52729.2023.02046.

17. Ivashechkin M., Barath D., Matas J. VSAC: Efficient and Accurate Estimator for H and F // ICCV 2021. Montreal, Canada, 2021. P. 15243–15252. DOI: 10.1109/ICCV48922.2021.01499.

18. Raspberry Pi hardware documentation. Raspberry Pi Ltd. URL: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (Last accessed: 15.05.2026).

19. systemd System and Service Manager: official documentation. The systemd Authors. URL: https://systemd.io (Last accessed: 15.05.2026).

20. OpenCV: Open Source Computer Vision Library: official documentation. OpenCV Team. URL: https://docs.opencv.org (Last accessed: 15.05.2026).

21. libcamera: Open Source Camera Stack for Linux: official documentation. The libcamera Project. URL: https://libcamera.org (Last accessed: 15.05.2026).

22. The Picamera2 Library: official documentationi. Raspberry Pi Ltd. URL: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf (Last accessed: 15.05.2026).

23. Flask Web Framework: official documentation. The Pallets Projects. URL: https://flask.palletsprojects.com (Last accessed: 15.05.2026).

ДОДАТОК А

Блок-схеми роботи АПЗ

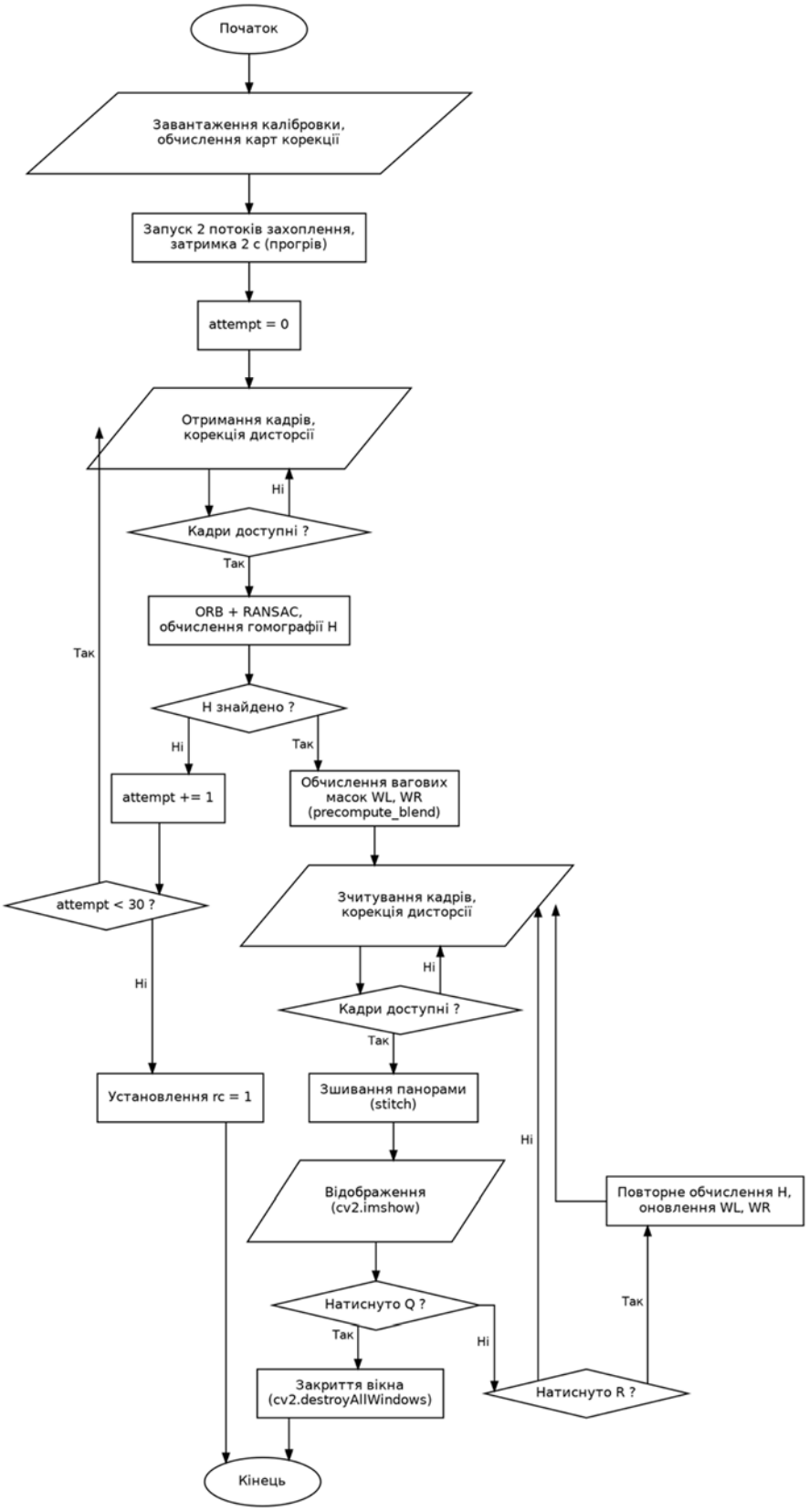


Рисунок А.1 – Блок-схема calibration.py

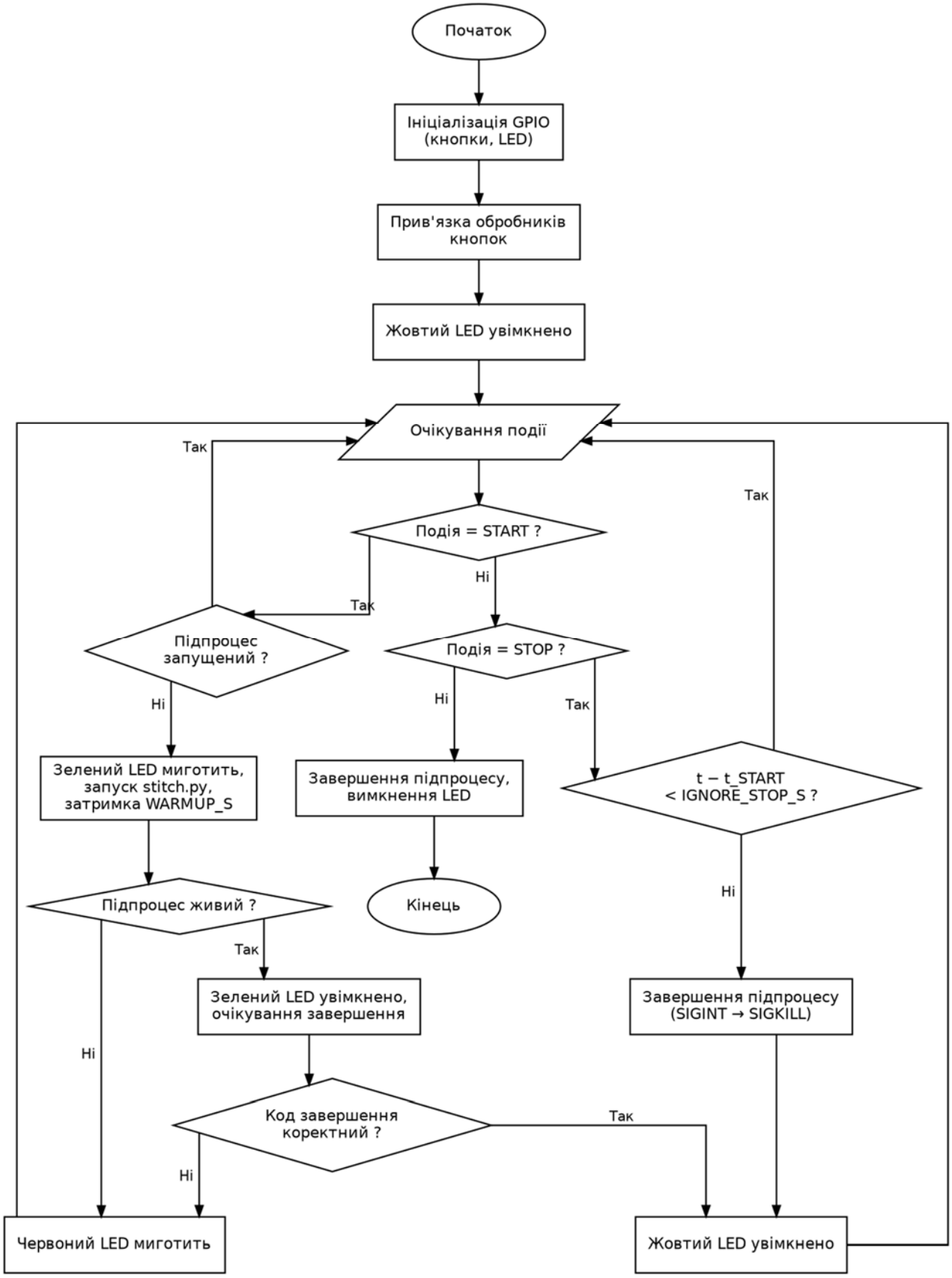


Рисунок А.2 – Блок-схема launcher.py

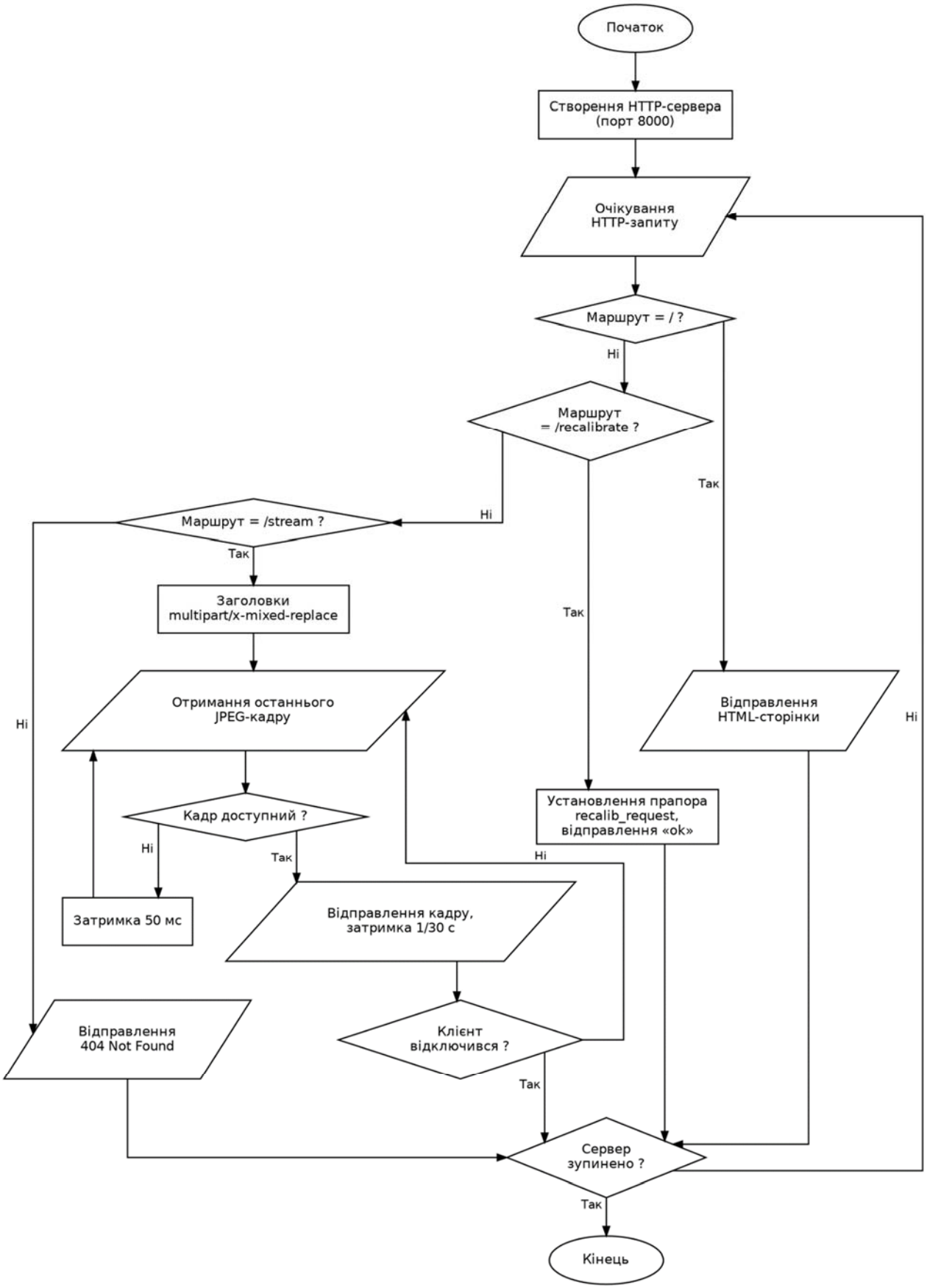


Рисунок А.3 – Блок-схема server.py

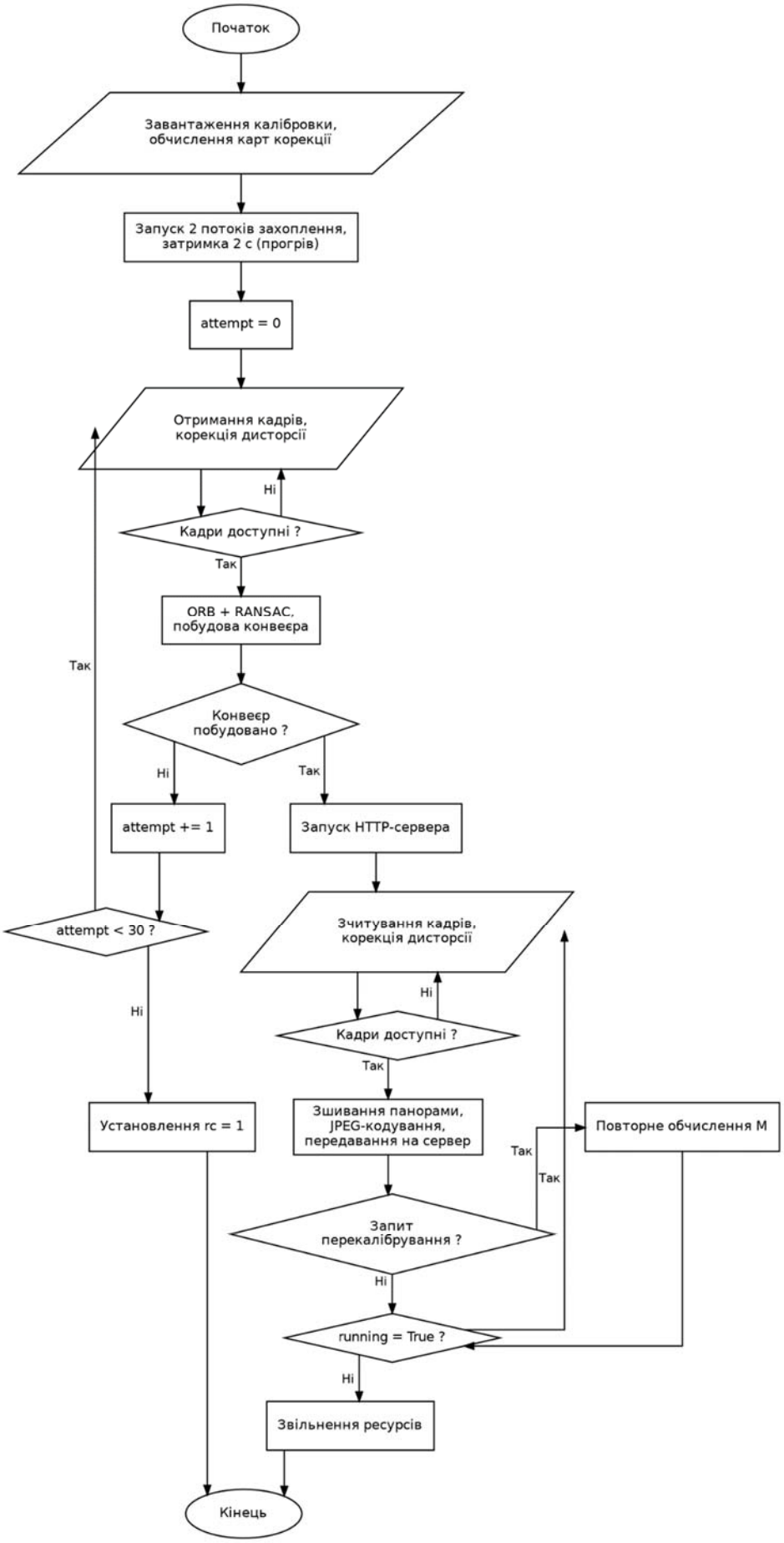


Рисунок А.4 – Блок-схема stitch.py

ДОДАТОК Б

Програмні коди для АПК

Б.1 Код програми calibration.py

```
import cv2
import threading
import subprocess
import numpy as np
import time

f0 = [None]
f1 = [None]
lock0 = threading.Lock()
lock1 = threading.Lock()
running = True

W, FRAME_H = 640, 480

calib0 = np.load("calib_cam0.npz")
calib1 = np.load("calib_cam1.npz")
CAM_MTX_L, DIST_L = calib0["camera_matrix"], calib0["dist_coeffs"]
CAM_MTX_R, DIST_R = calib1["camera_matrix"], calib1["dist_coeffs"]

new_mtx_l, _ = cv2.getOptimalNewCameraMatrix(CAM_MTX_L, DIST_L, (W, FRAME_H), 0, (W, FRAME_H))
new_mtx_r, _ = cv2.getOptimalNewCameraMatrix(CAM_MTX_R, DIST_R, (W, FRAME_H), 0, (W, FRAME_H))

# Undistort maps as float so they can be fused with the cylindrical warp
ud_l1, ud_l2 = cv2.initUndistortRectifyMap(CAM_MTX_L, DIST_L, None, new_mtx_l, (W, FRAME_H),
cv2.CV_32FC1)
ud_r1, ud_r2 = cv2.initUndistortRectifyMap(CAM_MTX_R, DIST_R, None, new_mtx_r, (W, FRAME_H),
cv2.CV_32FC1)


def cylindrical_maps(K, w, h):
    # Inverse map: cylindrical pixel -> source planar pixel
    f = (K[0, 0] + K[1, 1]) / 2.0
    cx, cy = K[0, 2], K[1, 2]
    xc, yc = np.meshgrid(np.arange(w, dtype=np.float32), np.arange(h, dtype=np.float32))
    theta = (xc - cx) / f
    src_x = f * np.tan(theta) + cx
    src_y = (yc - cy) / np.cos(theta) + cy
    return src_x.astype(np.float32), src_y.astype(np.float32)


# Fuse undistort + cylindrical into a single remap per camera (zero per-frame overhead)
cyl_lx, cyl_ly = cylindrical_maps(new_mtx_l, W, FRAME_H)
cyl_rx, cyl_ry = cylindrical_maps(new_mtx_r, W, FRAME_H)
MAP_L1 = cv2.remap(ud_l1, cyl_lx, cyl_ly, cv2.INTER_LINEAR)
MAP_L2 = cv2.remap(ud_l2, cyl_lx, cyl_ly, cv2.INTER_LINEAR)
MAP_R1 = cv2.remap(ud_r1, cyl_rx, cyl_ry, cv2.INTER_LINEAR)
MAP_R2 = cv2.remap(ud_r2, cyl_rx, cyl_ry, cv2.INTER_LINEAR)


def rectify_left(frame):
    return cv2.remap(frame, MAP_L1, MAP_L2, cv2.INTER_LINEAR)

def rectify_right(frame):
    return cv2.remap(frame, MAP_R1, MAP_R2, cv2.INTER_LINEAR)


def capture_camera(cam_id, frame_ref, lock):
    cmd = [
        "rpicam-vid", "--camera", str(cam_id),
        "--width", str(W), "--height", str(FRAME_H),
```

```
        "--framerate", "30", "--codec", "mjpeg",
        "--output", "-", "--nopreview", "--timeout", "0",
    ]
    proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL, bufsize=0)
    buf = b""
    while running:
        chunk = proc.stdout.read(4096)
        if not chunk:
            break
        buf += chunk
        s = buf.find(b'\xff\xd8')
        e = buf.find(b'\xff\xd9')
        if s != -1 and e != -1 and e > s:
            jpg = buf[s:e+2]
            buf = buf[e+2:]
            frame = cv2.imdecode(np.frombuffer(jpg, np.uint8), cv2.IMREAD_COLOR)
            if frame is not None:
                with lock:
                    frame_ref[0] = frame
    proc.terminate()


def compute_homography(img_left, img_right):
    """Calibration step on cylindrical frames: find H once at startup."""
    gray_l = cv2.cvtColor(img_left, cv2.COLOR_BGR2GRAY)
    gray_r = cv2.cvtColor(img_right, cv2.COLOR_BGR2GRAY)

    orb = cv2.ORB_create(nfeatures=2000)
    kp_l, des_l = orb.detectAndCompute(gray_l, None)
    kp_r, des_r = orb.detectAndCompute(gray_r, None)
    if des_l is None or des_r is None or len(kp_l) < 10 or len(kp_r) < 10:
        return None

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
    raw_matches = bf.knnMatch(des_r, des_l, k=2)

    good = []
    for m_pair in raw_matches:
        if len(m_pair) == 2 and m_pair[0].distance < 0.75 * m_pair[1].distance:
            good.append(m_pair[0])
    if len(good) < 10:
        return None

    src_pts = np.float32([kp_r[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp_l[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

    H, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
    return H


def precompute_blend(H, src_shape, canvas_size):
    """Distance-transform feather weights, computed once (H is fixed)."""
    cw, ch = canvas_size
    h_src, w_src = src_shape[:2]

    left_mask = np.zeros((ch, cw), np.uint8)
    left_mask[:h_src, :w_src] = 255
    ones = np.full((h_src, w_src), 255, np.uint8)
    right_mask = cv2.warpPerspective(ones, H, (cw, ch))

    dt_l = cv2.distanceTransform(left_mask, cv2.DIST_L2, 3)
    dt_r = cv2.distanceTransform(right_mask, cv2.DIST_L2, 3)
    total = dt_l + dt_r
    total[total == 0] = 1.0
    wL = (dt_l / total).astype(np.float32)
    wR = (dt_r / total).astype(np.float32)
    return cv2.merge([wL, wL, wL]), cv2.merge([wR, wR, wR])
```

```
def stitch(left, right, H, canvas_size, wL, wR):
    cw, ch = canvas_size
    warped_right = cv2.warpPerspective(right, H, (cw, ch))
    canvas_left = np.zeros((ch, cw, 3), np.uint8)
    canvas_left[:left.shape[0], :left.shape[1]] = left
    pano = canvas_left.astype(np.float32) * wL + warped_right.astype(np.float32) * wR
    return pano.astype(np.uint8)


t0 = threading.Thread(target=capture_camera, args=(0, f0, lock0), daemon=True)
t1 = threading.Thread(target=capture_camera, args=(1, f1, lock1), daemon=True)
t0.start()
t1.start()

print("Warming up cameras...")
time.sleep(2)

print("Computing homography...")
H = None
for attempt in range(30):
    with lock0:
        fl = f0[0].copy() if f0[0] is not None else None
    with lock1:
        fr = f1[0].copy() if f1[0] is not None else None
    if fl is None or fr is None:
        time.sleep(0.1)
        continue
    H = compute_homography(rectify_left(fl), rectify_right(fr))
    if H is not None:
        print(f"Homography found on attempt {attempt+1}")
        break
    time.sleep(0.2)

if H is None:
    print("ERROR: could not compute homography. Check camera overlap.")
    running = False
    exit(1)

CANVAS_W = W * 2
canvas_size = (CANVAS_W, FRAME_H)
WL, WR = precompute_blend(H, (FRAME_H, W), canvas_size)

print("Streaming. Press Q to quit, R to recalibrate.")
prev_t = time.time()
fps_count = 0

while True:
    with lock0:
        fl = f0[0].copy() if f0[0] is not None else None
    with lock1:
        fr = f1[0].copy() if f1[0] is not None else None
    if fl is None or fr is None:
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        continue

    fl_u = rectify_left(fl)
    fr_u = rectify_right(fr)
    pano = stitch(fl_u, fr_u, H, canvas_size, WL, WR)

    fps_count += 1
    if fps_count >= 30:
        now = time.time()
        fps = fps_count / (now - prev_t)
        prev_t = now
        fps_count = 0
        cv2.setWindowTitle("Panorama", f"Panorama - {fps:.1f} FPS")

    cv2.imshow("Panorama", pano)
```

```
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        running = False
        break
    elif key == ord('r'):
        print("Recalibrating...")
        new_H = compute_homography(fl_u, fr_u)
        if new_H is not None:
            H = new_H
            WL, WR = precompute_blend(H, (FRAME_H, W), canvas_size)
            print("Done.")

cv2.destroyAllWindows()
```

Б.2 Код програми stitch.py

```
import cv2
import threading
import subprocess
import numpy as np
import time

import server  # web server module

f0 = [None]
f1 = [None]
lock0 = threading.Lock()
lock1 = threading.Lock()
running = True

W, FRAME_H = 640, 480
CYL_F_SCALE = 1.2  # 1.0-1.5 keeps alignment sharp; higher = flatter but center starts to drift
HTTP_PORT = 8000
JPEG_QUALITY = 80

# Load per-camera calibration
calib0 = np.load("calib_cam0.npz")
calib1 = np.load("calib_cam1.npz")
CAM_MTX_L, DIST_L = calib0["camera_matrix"], calib0["dist_coeffs"]
CAM_MTX_R, DIST_R = calib1["camera_matrix"], calib1["dist_coeffs"]

new_mtx_l, _ = cv2.getOptimalNewCameraMatrix(CAM_MTX_L, DIST_L, (W, FRAME_H), 0, (W, FRAME_H))
new_mtx_r, _ = cv2.getOptimalNewCameraMatrix(CAM_MTX_R, DIST_R, (W, FRAME_H), 0, (W, FRAME_H))

ud_l1, ud_l2 = cv2.initUndistortRectifyMap(CAM_MTX_L, DIST_L, None, new_mtx_l, (W, FRAME_H),
cv2.CV_32FC1)
ud_r1, ud_r2 = cv2.initUndistortRectifyMap(CAM_MTX_R, DIST_R, None, new_mtx_r, (W, FRAME_H),
cv2.CV_32FC1)


def cylindrical_maps(K, w, h, f_scale=1.0):
    f = (K[0, 0] + K[1, 1]) / 2.0 * f_scale
    cx, cy = K[0, 2], K[1, 2]
    xc, yc = np.meshgrid(np.arange(w, dtype=np.float32), np.arange(h, dtype=np.float32))
    theta = (xc - cx) / f
    src_x = f * np.tan(theta) + cx
    src_y = (yc - cy) / np.cos(theta) + cy
    return src_x.astype(np.float32), src_y.astype(np.float32)


cyl_lx, cyl_ly = cylindrical_maps(new_mtx_l, W, FRAME_H, CYL_F_SCALE)
cyl_rx, cyl_ry = cylindrical_maps(new_mtx_r, W, FRAME_H, CYL_F_SCALE)
MAP_L1 = cv2.remap(ud_l1, cyl_lx, cyl_ly, cv2.INTER_LINEAR)
MAP_L2 = cv2.remap(ud_l2, cyl_lx, cyl_ly, cv2.INTER_LINEAR)
MAP_R1 = cv2.remap(ud_r1, cyl_rx, cyl_ry, cv2.INTER_LINEAR)
MAP_R2 = cv2.remap(ud_r2, cyl_rx, cyl_ry, cv2.INTER_LINEAR)
```

```
def rectify_left(frame):
    return cv2.remap(frame, MAP_L1, MAP_L2, cv2.INTER_LINEAR)


def rectify_right(frame):
    return cv2.remap(frame, MAP_R1, MAP_R2, cv2.INTER_LINEAR)


def capture_camera(cam_id, frame_ref, lock):
    cmd = [
        "rpicam-vid", "--camera", str(cam_id),
        "--width", str(W), "--height", str(FRAME_H),
        "--framerate", "30", "--codec", "mjpeg",
        "--output", "-", "--nopreview", "--timeout", "0",
    ]
    proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL, bufsize=0)
    buf = b""
    while running:
        chunk = proc.stdout.read(4096)
        if not chunk:
            break
        buf += chunk
        s = buf.find(b'\xff\xd8')
        e = buf.find(b'\xff\xd9')
        if s != -1 and e != -1 and e > s:
            jpg = buf[s:e+2]
            buf = buf[e+2:]
            frame = cv2.imdecode(np.frombuffer(jpg, np.uint8), cv2.IMREAD_COLOR)
            if frame is not None:
                with lock:
                    frame_ref[0] = frame
    proc.terminate()


def compute_transform(img_left, img_right):
    """Similarity on cylindrical frames -> stable for wide divergence."""
    gray_l = cv2.cvtColor(img_left, cv2.COLOR_BGR2GRAY)
    gray_r = cv2.cvtColor(img_right, cv2.COLOR_BGR2GRAY)

    orb = cv2.ORB_create(nfeatures=3000)
    kp_l, des_l = orb.detectAndCompute(gray_l, None)
    kp_r, des_r = orb.detectAndCompute(gray_r, None)
    if des_l is None or des_r is None or len(kp_l) < 10 or len(kp_r) < 10:
        return None

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
    raw = bf.knnMatch(des_r, des_l, k=2)
    good = []
    for p in raw:
        if len(p) == 2 and p[0].distance < 0.75 * p[1].distance:
            good.append(p[0])
    if len(good) < 15:
        print(f"[M] too few matches: {len(good)}")
        return None

    src = np.float32([kp_r[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
    dst = np.float32([kp_l[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

    M, mask = cv2.estimateAffinePartial2D(src, dst, method=cv2.RANSAC, ransacReprojThreshold=3.0)
    if M is None:
        return None

    n_in = int(mask.ravel().sum())
    angle = np.degrees(np.arctan2(M[1, 0], M[0, 0]))
    print(f"[M] inliers {n_in}/{len(good)} | residual rotation {angle:+.2f}deg")
    return M


def shift_affine(M, ox, oy):
```

```
    M3 = np.vstack([M, [0, 0, 1]]).astype(np.float64)
    T3 = np.array([[1, 0, ox], [0, 1, oy], [0, 0, 1]], dtype=np.float64)
    return (T3 @ M3)[:2, :].astype(np.float32)


def compute_canvas(M, w, h):
    corners = np.float32([[0, 0], [w, 0], [w, h], [0, h]]).reshape(-1, 1, 2)
    warped = cv2.transform(corners, M).reshape(-1, 2)
    allpts = np.vstack([corners.reshape(-1, 2), warped])
    xmin, ymin = np.floor(allpts.min(axis=0)).astype(int)
    xmax, ymax = np.ceil(allpts.max(axis=0)).astype(int)
    return (int(xmax - xmin), int(ymax - ymin)), (int(-xmin), int(-ymin))


def largest_inner_rect(mask):
    """Largest all-True axis-aligned rectangle -> (x, y, w, h)."""
    h, w = mask.shape
    heights = np.zeros(w, dtype=np.int32)
    best = (0, 0, 0, 0)
    best_area = 0
    for y in range(h):
        heights = np.where(mask[y], heights + 1, 0)
        stack = []
        x = 0
        while x <= w:
            cur = int(heights[x]) if x < w else 0
            start = x
            while stack and stack[-1][1] > cur:
                sx, sh = stack.pop()
                area = sh * (x - sx)
                if area > best_area:
                    best_area = area
                    best = (sx, y - sh + 1, x - sx, sh)
                start = sx
            stack.append((start, cur))
            x += 1
    return best


def find_seam(left_canvas, right_canvas, overlap):
    """Vertical min-energy seam through the overlap (DP). Returns seam x per row."""
    gl = cv2.cvtColor(left_canvas, cv2.COLOR_BGR2GRAY).astype(np.float64)
    gr = cv2.cvtColor(right_canvas, cv2.COLOR_BGR2GRAY).astype(np.float64)
    energy = np.where(overlap, np.abs(gl - gr), 1e9)  # high cost outside overlap
    h, w = energy.shape
    cost = energy.copy()
    back = np.zeros((h, w), np.int32)
    cols = np.arange(w)
    for y in range(1, h):
        prev = cost[y - 1]
        left = np.full(w, np.inf); left[1:] = prev[:-1]    # inf at borders so edges never chosen
        right = np.full(w, np.inf); right[:-1] = prev[1:]
        stacked = np.vstack([left, prev, right])
        idx = np.argmin(stacked, axis=0)                   # 0=left, 1=mid, 2=right
        cost[y] += np.min(stacked, axis=0)
        back[y] = cols + (idx - 1)
    seam = np.zeros(h, np.int32)
    seam[-1] = int(np.argmin(cost[-1]))
    for y in range(h - 2, -1, -1):
        seam[y] = back[y + 1, seam[y + 1]]
    np.clip(seam, 0, w - 1, out=seam)
    return seam


def seam_blend_mask(seam, left_mask, right_mask, canvas_size, feather=20):
    cw, ch = canvas_size
    xs = np.arange(cw, dtype=np.float32)[None, :]
    s = seam[:, None].astype(np.float32)
    wL = np.clip((s - xs) / feather + 0.5, 0.0, 1.0)  # 1 left of seam, 0 right
```

```
    wR = 1.0 - wL
    wL *= (left_mask > 0)
    wR *= (right_mask > 0)
    total = wL + wR
    total[total == 0] = 1.0
    wL /= total
    wR /= total
    return cv2.merge([wL, wL, wL]).astype(np.float32), cv2.merge([wR, wR, wR]).astype(np.float32)


def stitch(left, right, M_shift, canvas_size, offset, wL, wR, crop):
    cw, ch = canvas_size
    ox, oy = offset
    warped_right = cv2.warpAffine(right, M_shift, (cw, ch))
    canvas_left = np.zeros((ch, cw, 3), np.uint8)
    canvas_left[oy:oy + left.shape[0], ox:ox + left.shape[1]] = left
    pano = (canvas_left.astype(np.float32) * wL + warped_right.astype(np.float32) *
wR).astype(np.uint8)
    x, y, cw_, ch_ = crop
    if cw_ > 0 and ch_ > 0:
        pano = pano[y:y + ch_, x:x + cw_]
    return pano

def build_pipeline(M, fl_u, fr_u):
    """Canvas + seam mask + crop, computed once from calibration frames."""
    canvas_size, offset = compute_canvas(M, W, FRAME_H)
    if canvas_size[0] > W * 4 or canvas_size[1] > FRAME_H * 4:
        print(f"[!] canvas too large {canvas_size} -> bad alignment, ignoring")
        return None
    cw, ch = canvas_size
    ox, oy = offset
    M_shift = shift_affine(M, offset[0], offset[1])

    canvas_left = np.zeros((ch, cw, 3), np.uint8)
    canvas_left[oy:oy + FRAME_H, ox:ox + W] = fl_u
    warped_right = cv2.warpAffine(fr_u, M_shift, (cw, ch))
    left_mask = np.zeros((ch, cw), np.uint8)
    left_mask[oy:oy + FRAME_H, ox:ox + W] = 255
    right_mask = cv2.warpAffine(np.full((FRAME_H, W), 255, np.uint8), M_shift, (cw, ch))

    overlap = (left_mask > 0) & (right_mask > 0)
    if overlap.sum() < 100:
        print("[!] no overlap for seam")
        return None

    seam = find_seam(canvas_left, warped_right, overlap)
    WL, WR = seam_blend_mask(seam, left_mask, right_mask, canvas_size)
    crop = largest_inner_rect((left_mask > 0) | (right_mask > 0))
    print(f"[canvas] {canvas_size} offset {offset} crop {crop}")
    return M_shift, canvas_size, offset, WL, WR, crop

t0 = threading.Thread(target=capture_camera, args=(0, f0, lock0), daemon=True)
t1 = threading.Thread(target=capture_camera, args=(1, f1, lock1), daemon=True)
t0.start()
t1.start()

print("Warming up cameras...")
time.sleep(2)

print("Computing alignment...")
pipeline = None
for attempt in range(30):
    with lock0:
        fl = f0[0].copy() if f0[0] is not None else None
    with lock1:
        fr = f1[0].copy() if f1[0] is not None else None
    if fl is None or fr is None:
        time.sleep(0.1)
        continue
```

```
    fl_u = rectify_left(fl)
    fr_u = rectify_right(fr)
    M = compute_transform(fl_u, fr_u)
    if M is not None:
        pipeline = build_pipeline(M, fl_u, fr_u)
        if pipeline is not None:
            print(f"Alignment found on attempt {attempt + 1}")
            break
    time.sleep(0.2)

if pipeline is None:
    print("ERROR: alignment failed. Check camera overlap.")
    running = False
    exit(1)

M_shift, canvas_size, offset, WL, WR, CROP = pipeline

# Start web server (background thread)
server.start(HTTP_PORT)

prev_t = time.time()
fps_count = 0

try:
    while running:
        with lock0:
            fl = f0[0].copy() if f0[0] is not None else None
        with lock1:
            fr = f1[0].copy() if f1[0] is not None else None
        if fl is None or fr is None:
            time.sleep(0.01)
            continue

        fl_u = rectify_left(fl)
        fr_u = rectify_right(fr)
        pano = stitch(fl_u, fr_u, M_shift, canvas_size, offset, WL, WR, CROP)

        # Encode to JPEG and publish to the web server
        ok, enc = cv2.imencode(".jpg", pano, [cv2.IMWRITE_JPEG_QUALITY, JPEG_QUALITY])
        if ok:
            server.update_frame(enc.tobytes())

        fps_count += 1
        if fps_count >= 30:
            now = time.time()
            fps = fps_count / (now - prev_t)
            prev_t = now
            fps_count = 0
            print(f"\r{fps:.1f} FPS", end="", flush=True)

        # Recalibration triggered from the browser button
        if server.recalib_request.is_set():
            server.recalib_request.clear()
            print("\nRecalibrating...")
            new_M = compute_transform(fl_u, fr_u)
            if new_M is not None:
                new_pipe = build_pipeline(new_M, fl_u, fr_u)
                if new_pipe is not None:
                    M_shift, canvas_size, offset, WL, WR, CROP = new_pipe
                    print("Done.")

except KeyboardInterrupt:
    print("\nStopping...")
finally:
    running = False
    time.sleep(0.3)
```

Б.3 Код програми launcher.py

```
#!/usr/bin/env python3
"""Headless launcher: GPIO buttons -> stitch.py subprocess. Status via LEDs."""

import os
import signal
import subprocess
import sys
import time
from pathlib import Path
from threading import Thread, Lock

from gpiozero import Button, LED

# --- Pin map (BCM) ---
PIN_BTN_START = 17
PIN_BTN_STOP = 27
PIN_LED_READY = 22   # yellow
PIN_LED_RUN = 23     # green
PIN_LED_ERROR = 24   # red

BASE_DIR = Path(__file__).resolve().parent
STITCH_CMD = [sys.executable, "-u", str(BASE_DIR / "stitch.py")]

WARMUP_S = 4          # let cameras come up before declaring "running"
IGNORE_STOP_S = 5     # ignore STOP within this window after START (anti-noise)

led_ready = LED(PIN_LED_READY)
led_run = LED(PIN_LED_RUN)
led_error = LED(PIN_LED_ERROR)
btn_start = Button(PIN_BTN_START, pull_up=True, bounce_time=0.15)
btn_stop = Button(PIN_BTN_STOP, pull_up=True, bounce_time=0.15)

proc = None
start_ts = 0.0
lock = Lock()


def all_leds_off():
    led_ready.off(); led_run.off(); led_error.off()


def set_state_ready():
    led_run.off(); led_error.off(); led_ready.on()


def set_state_running():
    led_ready.off(); led_error.off(); led_run.on()


def set_state_error():
    led_ready.off(); led_run.off(); led_error.blink(on_time=0.3, off_time=0.3)


def _supervise():
    """Own thread: wait out warmup, set LED, then block until the process exits."""
    global proc
    time.sleep(WARMUP_S)
    with lock:
        p = proc
    if p is None:
        return
    if p.poll() is not None:        # died during warmup
        set_state_error()
        return
    set_state_running()
    rc = p.wait()                   # block until stitch.py exits
```

```
    with lock:
        if proc is not p:           # already replaced/stopped elsewhere
            return
        proc = None
    if rc in (0, -signal.SIGINT, -signal.SIGTERM):
        set_state_ready()
    else:
        print(f"[launcher] stitch.py exited rc={rc}")
        set_state_error()


def start_stitcher():
    global proc, start_ts
    with lock:
        if proc and proc.poll() is None:
            return
        print("[launcher] START")
        start_ts = time.time()
        led_ready.off(); led_error.off()
        led_run.blink(on_time=0.15, off_time=0.15)  # warming up
        proc = subprocess.Popen(
            STITCH_CMD, cwd=str(BASE_DIR),
            stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL,
            preexec_fn=os.setsid,  # own process group for clean kill
        )
    Thread(target=_supervise, daemon=True).start()


def stop_stitcher():
    global proc
    # Anti-noise: a STOP right after START is almost certainly electrical, not human
    if time.time() - start_ts < IGNORE_STOP_S:
        print("[launcher] STOP ignored (too soon after START)")
        return
    with lock:
        p = proc
        if p is None or p.poll() is not None:
            proc = None
            set_state_ready()
            return
        print("[launcher] STOP")
        proc = None
    try:
        os.killpg(os.getpgid(p.pid), signal.SIGINT)
        p.wait(timeout=5)
    except subprocess.TimeoutExpired:
        os.killpg(os.getpgid(p.pid), signal.SIGKILL)
    except ProcessLookupError:
        pass
    set_state_ready()


def handle_exit(*_):
    global proc
    with lock:
        p = proc
        proc = None
    if p and p.poll() is None:
        try:
            os.killpg(os.getpgid(p.pid), signal.SIGINT)
            p.wait(timeout=5)
        except subprocess.TimeoutExpired:
            os.killpg(os.getpgid(p.pid), signal.SIGKILL)
        except ProcessLookupError:
            pass
    all_leds_off()
    sys.exit(0)
```

```
signal.signal(signal.SIGTERM, handle_exit)
signal.signal(signal.SIGINT, handle_exit)

btn_start.when_pressed = start_stitcher
btn_stop.when_pressed = stop_stitcher

set_state_ready()
print("[launcher] ready. Press START.")
signal.pause()
```

Б.4 Код програми server.py

```
import time
import socket
import threading
from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer

# Shared state between stitcher and web server
_latest_jpeg = [None]
_jpeg_lock = threading.Lock()
recalib_request = threading.Event()

PAGE = """<!DOCTYPE html>
<html><head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<title>Panorama Stream</title>
<style>
  body{margin:0;background:#111;color:#eee;font-family:sans-serif;text-align:center}
  h1{font-size:1.1rem;padding:10px;margin:0}
  img{max-width:100%;height:auto;display:block;margin:0 auto}
  button{margin:10px;padding:8px 16px;font-size:1rem;border:0;border-
radius:6px;background:#2d7;color:#000;cursor:pointer}
</style>
</head>
<body>
<h1>Raspberry Pi – Panorama (Real-time)</h1>
<img src="/stream">
<div><button onclick="fetch('/recalibrate')">Recalibrate</button></div>
</body></html>"""


def update_frame(jpeg_bytes):
    """Called by the stitcher to publish the newest panorama."""
    with _jpeg_lock:
        _latest_jpeg[0] = jpeg_bytes


def get_local_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(("8.8.8.8", 80))
        return s.getsockname()[0]
    except Exception:
        return "127.0.0.1"
    finally:
        s.close()


class StreamHandler(BaseHTTPRequestHandler):
```

```
    def log_message(self, *args):
        pass  # silence per-request logging

    def do_GET(self):
        if self.path in ("/", "/index.html"):
            body = PAGE.encode("utf-8")
            self.send_response(200)
            self.send_header("Content-Type", "text/html; charset=utf-8")
            self.send_header("Content-Length", str(len(body)))
            self.end_headers()
            self.wfile.write(body)

        elif self.path == "/recalibrate":
            recalib_request.set()
            self.send_response(200)
            self.send_header("Content-Type", "text/plain")
            self.end_headers()
            self.wfile.write(b"ok")

        elif self.path == "/stream":
            self.send_response(200)
            self.send_header("Content-Type", "multipart/x-mixed-replace; boundary=frame")
            self.send_header("Cache-Control", "no-cache, private")
            self.end_headers()
            try:
                while True:
                    with _jpeg_lock:
                        buf = _latest_jpeg[0]
                    if buf is None:
                        time.sleep(0.05)
                        continue
                    self.wfile.write(b"--frame\r\n")
                    self.wfile.write(b"Content-Type: image/jpeg\r\n")
                    self.wfile.write(f"Content-Length: {len(buf)}\r\n\r\n".encode())
                    self.wfile.write(buf)
                    self.wfile.write(b"\r\n")
                    time.sleep(1 / 30)
            except (BrokenPipeError, ConnectionResetError):
                pass  # client disconnected
        else:
            self.send_response(404)
            self.end_headers()


def start(port=8000):
    """Launch the web server in a background daemon thread."""
    server = ThreadingHTTPServer(("0.0.0.0", port), StreamHandler)
    threading.Thread(target=server.serve_forever, daemon=True).start()
    print(f"Streaming at http://{get_local_ip()}:{port}/")
```

ДОДАТОК В

Матеріали апробації роботи

В.1 XXVIII Всеукраїнська науково-практична конференція «Могилянські читання – 2025»

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ ім. Петра Могили

**«МОГИЛЯНСЬКІ ЧИТАННЯ – 2025:
досвід та тенденції розвитку суспільства в Україні: глобальний,
національний та регіональний аспекти»**

XXVIII Всеукраїнська науково-практична конференція

ТЕЗИ ДОПОВІДЕЙ

ТЕХНІЧНІ НАУКИ

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

Підсекції:

- **КОМП’ЮТЕРНА ІНЖЕНЕРІЯ**

УДК 004.353.4

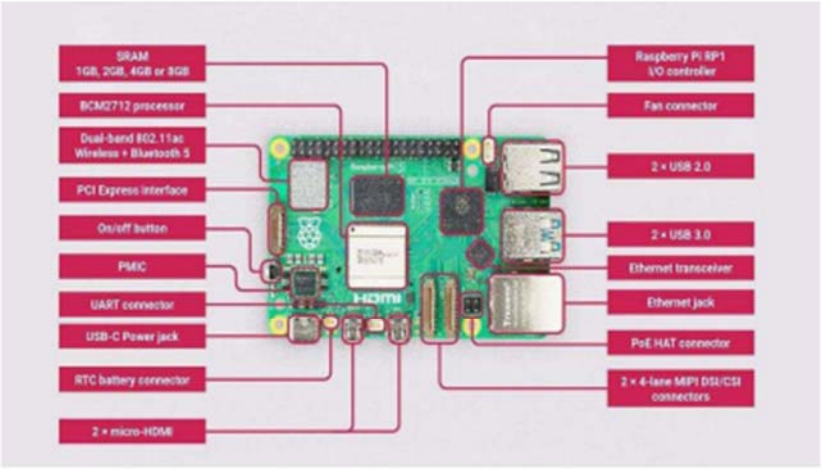
Невідомий Д. О.,
бакалаврант кафедри комп’ютерної інженерії,
Пузирьов С. В.,
канд. фіз.-мат. наук, доцент кафедри комп’ютерної інженерії,
Чорноморський нац. ун-т ім. Петра Могили, м. Миколаїв, Україна

СИСТЕМА ОБ’ЄДНАННЯ ВІДЕОПОТОКІВ У РЕАЛЬНОМУ ЧАСІ НА БАЗІ RASPBERRY PI

Актуальність теми. Сучасні системи відеоспостереження зіштовхуються з проблемою обмеженого кута огляду стандартних камер. Це створює "сліпі зони" в системах безпеки та ускладнює створення повного панорамного зображення. Вирішенням цієї проблеми є програмне об'єднання відеопотоків з кількох камер в єдину панораму.

Мета роботи. Розробити програмно-апаратний комплекс на базі одноплатного комп'ютера Raspberry Pi для з’єднання відеопотоків з двох або більше камер у реальному часі, що дозволяє отримати єдине панорамне зображення.

Апаратна та програмна платформа. В якості центрального обчислювального пристрою використовується одноплатний комп'ютер Raspberry Pi 5, на ньому присутні два універсальні роз’єми MIPI CSI/DSI, які використовуються для підключення двох Raspberry Pi Camera Module. Програмна частина реалізована на мові Python з використанням бібліотеки комп'ютерного зору OpenCV.



77